Modular Tracking Framework: A Unified Approach to Registration based Tracking

by

Abhineet Singh

A thesis submitted in partial fulfilment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

This thesis presents a new way to conceptualize and study image registration based visual trackers by decomposing them into three constituent sub modules - search method, appearance model and state space model. It shows how this approach can be used to break down existing trackers and thus unify the myriad of contributions that have been made in this domain in over three decades of its existence.

Further, a modular framework for registration based tracking is introduced to provide practical validity to this formulation. It provides highly efficient C++ implementations for a large subset of trackers introduced in literature to date and is designed to be easily extensible with additional methods. It follows this decomposition closely through extensive use of generic programming to provide a convenient interface to plug in a new method for any sub module and test it against all possible combinations of methods for the others. This can not only help to evaluate the new method in a more comprehensive manner but also make it immediately available for deployment in any project that uses the framework.

Finally, three existing image similarity measures are adapted for high precision tracking and a new one is introduced to serve as case studies for the proposed approach of analyzing registration based tracking. Experiments are conducted using synthetic data as well as four large publicly available datasets with over 100000 frames in all to ensure the statistical validity of the results.

Acknowledgements

I would like to extend my most sincere gratitude to my supervisor Professor Martin Jägersand. I greatly appreciate his kindness in allowing me to be a part of his research group as well as the invaluable guidance and encouragement he has provided over the years.

Furthermore, I would like to thank all my colleagues in the Computer Vision and Robotics lab in the Department of Computing Science. I have only been able to complete my work because of their help, contributions and excellent collaboration. In particular, I would like to thank Ankush Roy, Xi Zhang, Oscar Ramirez, Mennatullah Siam, Vincent Zhang and Xuebin Qin for their help in everything from practical matters to discussions of ideas. Many parts of my work are owing to their suggestions and contributions.

I would also like to thank the University of Alberta Department of Computing Science and my supervisor for support through research and teaching assistantships.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

RBT	*Registration based tracking/tracker*
OLT	*Online learning and detection based tracking/tracker*
DOF	*Degrees of Freedom*
LK	*Lucas Kanade*
SM	*Search Method*
AM	*Appearance Model*
SSM	*State Space Model*
SSIM	*Structural Similarity*
RIU	*Ratio Image Uniformity*
NGF	*Normalized Gradient Fields*
MTF	*Modular Tracking Framework*
ViSP	*Visual Servoing Platform*
DSST	*Discriminative Scale Space Tracking*
KCF	*Kernelized Correlation Filter*
CMT	*Clustering of Static-Adaptive Correspondences for Deformable Object Tracking*
Struck	*Structured Output Tracking with Kernels*
MIL	*Multiple Instance Learning*
TLD	*Tracking-Learning-Detection*
FragTrack	*Fragments based Tracking*

GOTURN	*Generic Object Tracking Using Regression Networks*
SSD	*Sum of Squared Differences*
NCC	*Normalized Cross Correlation*
MI	*Mutual Information*
ILM	*Illumination Model*
GD	*Gradient Descent*
GN	*Gauss Newton*
LM	*Levenberg Marquardt*
FCLK	*Forward Compositional Lucas Kanade*
ICLK	*Inverse Compositional Lucas Kanade*
FALK	*Forward Additive Lucas Kanade*
IALK	*Inverse Additive Lucas Kanade*
FCSD	*Forward Compositional Steepest Descent*
ESM	*Efficient Second order Minimization*
AESM	*Additive Efficient Second order Minimization*
NN	*Nearest neighbor*
FLANN	*Fast Library for Approximate Nearest Neighbors*
KDT	*KD Tree search*
HKMT	*Hierarchical K-Means Tree search*
GNN	*Graph based Nearest Neighbor*
FGNN	*FLANN based GNN*
PF	*Particle Filter*
SMC	*Sequential Monte Carlo*
RANSAC	*Random Sample Consensus*
LMS	*Least Median of Squares*

SPI	*Selective Pixel Integration*
PCA	*Principal Components Analysis*
DFM	*Deep Feature Map*
KLD	*Kullback-Leibler Divergence*
SCV	*Sum of Conditional Variance*
RSCV	*Reversed Sum of Conditional Variance*
LSCV	*Localized Sum of Conditional Variance*
ZNCC	*Zero mean Normalized Cross Correlation*
CCRE	*Cross Cumulative Residual Entropy*
SPSS	*Sum of Pixelwise Structural Similarity*
GB	*Gain & Bias*
PGB	*Piecewise Gain & Bias*
RBF	*Radial Basis Function*
IST	*Isotropic Scaling and Translation*
AST	*Anisotropic Scaling and Translation*
TPS	*Thin Plate Splines*
DLT	*Direct Linear Transform*
CBH	*Corner Based Homography*
SR	*Success Rate*
FR	*Failure Rate*
FPS	*Frames Per Second*

Chapter 1

Introduction

1.1 Motivation

Visual tracking is an important field in computer vision with diverse application domains including robotics, surveillance, targeting systems, medical imaging and augmented reality. Registration based tracking (**RBT**), also known as direct visual tracking in literature [2, 3, 4, 5], is an important sub field thereof that considers the tracking problem as one of image registration where the object is tracked by warping each image in a sequence to align the object patch with the template. Trackers of this type are especially suited to robotics and augmented reality (AR) applications like visual servoing, autonomous navigation and pose estimation where fast and high precision tracking is required.

In recent years, online learning and detection based trackers (**OLT**s) have become more popular [6, 7, 8, 9] due to their robustness to changes in the object's appearance which makes them better suited to long term tracking. Even so, these are often unsuitable for robotics applications for the following two reasons:

- They are too slow [10] to allow real time execution of tasks where multiple trackers have to be run simultaneously or tracking is only a small part of a larger system with more computationally intensive modules that use its result to make higher level deductions about the environment.

- They are not precise enough [10] to provide the exact object pose with sub pixel alignment required for the success of these tasks, being usually limited to the estimation of simple transformations of the target patch like translation and scaling.

RBTs are thus a better match for these applications as being several times faster and also capable of estimating higher degrees-of-freedom (DOF) transformations like affine and homography. The advantages of greater speed and precision are not without downsides, however, as RBTs cannot employ computationally expensive algorithms to update their model of the object's appearance while also having to deal with higher dimensional search space that makes it easier to get stuck in local optima. As a result, they are prone to failure when the object undergoes significant appearance changes due to factors like occlusions and lighting variations or when completely novel views of the object are presented by deformations or large pose changes.

These limitations have contributed to the greater popularity of OLTs which in turn has made them the subject of several recent studies [7, 11, 12, 8, 13]. Though useful for obtaining a general idea of the state of progress in this field, such studies rarely provide any feedback that may help to improve these trackers. This is because their scope is restricted to finding the trackers that work best under a variety of challenging conditions by testing them on representative sequences and little to no analysis is conducted regarding *why* specific trackers work better than others for given challenges. This is understandable since such trackers differ widely in design and have little in common that may be used to relate them to each other and perform comparative analysis from a design perspective. A relatively recent work [14] has indeed attempted a systematic breakdown of OLTs into five modules, much like the current work. As expected, however, only a relatively small subset of trackers could be fitted into the framework proposed there. Even so, the module level analysis performed on those few trackers did present some unexpected findings which indicates the general usefulness of this strategy.

This thesis will show that RBTs are fortunately not subject to this drawback and can be decomposed into three sub modules - search method (**SM**) (chapter 4), appearance model (**AM**) (chapter 5) and state space model (**SSM**) (chapter 6) - that makes their systematic analysis feasible. Though this decomposition is somewhat obvious and indeed has been noted before [15, 4], it has never been explored systematically or used to improve the study of this paradigm of tracking. It is the intent of this work to fill this gap by using such a decomposition to unify the multitude of contributions that have been made in this field since the original Lucas Kanade (**LK**) tracker was introduced thirty five years ago [16].

When a new tracker is introduced in this field, it usually contributes to only one or two of these sub modules while using existing methods for the rest (Sec. 2.2.1).

Since these are often selected arbitrarily by the authors, they may not be optimal for the new method. The potential downsides of such limited testing are twofold. Firstly, it may give false indications about a method's capability and prevent it from achieving its full potential. For instance, an AM that outperforms others with a given SM might not do so with other SMs and an SM may perform better with an AM other than the one it is tested with. Secondly, with different contributions not being well related to each other, the overall progress in this field has become somewhat fragmented, making it difficult to gain a high level idea of where it stands. The proposed breakdown can not only reduce this fragmentation but also help to experimentally find the best combination of methods for these sub modules while providing a model within which the contributions of any new tracker can be clearly demarcated and thus studied better [10, 17]. The practical applicability of this approach is demonstrated by comparing several existing methods for these sub modules not only with each other but also with three new AMs - structural similarity (**SSIM**) (Sec. 5.2.4), ratio image uniformity (**RIU**) (Sec. 5.2.6) and normalized gradient fields (**NGF**) (Sec. 5.2.7) - that are introduced here and fit within this framework. Several new and interesting insights about the behaviors and properties of these methods are obtained by such comprehensive testing (Chapter 9).

Besides these theoretical concerns, this thesis also intends to address a more practical need that has emerged in recent years. Though several major advances have been made in this domain since the introduction of the LK tracker [16], yet efficient open source implementations of recent trackers are surprisingly difficult to find. In fact, the only such tracker offered by the popular OpenCV library [18], uses a pyramidal implementation of the original algorithm [19]. Similarly, the ROS library [20], widely used by the robotics community, currently does not have any package that implements a modern RBT. The XVision system [21] did introduce a full tracking framework including a video pipeline. However, it implements several variants of the same algorithm [22] that only gives reasonable tracking performance with low DOF motion. In addition, it is not well documented and has not been updated for a long time which makes it quite difficult to install on modern systems due to many obsolete dependencies. Even the fairly recent MRPT library [23] includes only a version of the original LK tracker, much like OpenCV, apart from a low DOF particle filter based tracker which is too imprecise and slow to be considered relevant for our target applications. An existing system that makes an attempt to follow the proposed framework is the template tracker module of the Visual Servoing Platform (ViSP) library [24]

but it has several shortcomings that are enumerated later (Sec. 7.6).

In the absence of good open source implementations of modern trackers, most robotics and AR research groups either use these out dated methods or implement their own custom trackers. These, in turn, are often not made publicly available or are tailored to suit very specific needs and so require significant reprogramming to be useful for an unrelated project. Further, many new methods are implemented by the authors in scripting languages like MATLAB and Python due to the ease of prototyping these offer compared to compiled languages like C and C++. However, unless fast implementations of these methods exist, they are unlikely to find wider applicability in actual projects where speed is crucial. The result is again that older methods are employed even though better alternatives exist simply because a good implementation is readily available for the former.

To address this need for a fast tracking library targeted specifically at robotics and AR applications, this thesis introduces Modular Tracking Framework (**MTF**)[1] - a generic system for RBT that provides highly efficient implementations for a large subset of trackers introduced in literature to date and is designed to be easily extensible with additional methods. MTF conceptualizes an RBT as being composed of the three aforementioned sub modules. These are designed to be semi independent with SM being treated as a way to use the functionality in AM and SSM - through a well defined interface - to solve the tracking problem. By following the decomposition closely through extensive use of generic programming, MTF provides a convenient interface to plug in a new method for any sub module and test it against existing methods for the other two. This will not only help to compare the new method against existing ones in a more comprehensive way but also make it immediately available to any project that uses MTF. This latter is particularly significant since, once the contribution of a new tracker has been identified to be in a specific sub module, it is much easier to add a new method for that sub module than to implement the complete tracker from scratch.

1.2 Contributions

To summarize, following are the main contributions of this work:

- Present a unifying formulation for RBT that decomposes it into three sub mod-

[1] available at http://webdocs.cs.ualberta.ca/~vis/mtf/

ules - SM, AM and SSM. This can be seen as an extension of the seminal work of Baker & Matthews [25] to account for several advancements since its publication.

- Show how existing trackers can be broken down using this approach and related to each other in a better way to counter the fragmentation that ails this field.
- Combine several combinations of existing methods for these sub modules with each other and perform comparative analysis to show what new insights that were missing from their original papers can be thus obtained and also achieve a new state of the art in high precision tracking.
- Adapt three image similarity measures - SSIM, RIU and NGF - for high precision RBT and introduce a simpler but faster version of SSIM called SPSS.
- Evaluate these models comprehensively using the proposed approach to demonstrate its merits for testing new methods in this domain. Experiments are performed using 4 large and challenging datasets with over 100,000 frames in all to ensure statistical significance of the results.
- Compare RBTs against state of the art OLTs to validate the suitability of the former for precise tracking applications.
- Introduce a highly efficient and extensible open source tracking framework called MTF that follows this decomposition closely and is targeted at robotics and AR applications. All results are generated using it and so are easily reproducible.

1.3 Thesis Outline

A brief review of existing literature relevant to this work is presented in chapter 2 followed by details of the proposed formulation in chapters 3 through 6. More specifically, notations and a broad overview of the decomposition are provided in chapter 3 with detailed descriptions of SM, AM and SSM following in chapters 4, 5 and 6 respectively. Chanter 7 presents details of MTF system design including several use cases and comparisons with existing systems. This is followed by a description of the datasets and evaluation metrics used and the actual results and analysis in chapters 8 and 9 respectively. Finally, chapter 10 concludes with a summary of this work along with suggestions regarding directions for its future extensions.

Chapter 2

Background

Visual tracking has been a well researched field in computer vision due to its manifold applications. There have been many surveys and reviews of visual tracking in recent times [6, 7, 11, 12, 9, 13] though, as mentioned before, they have largely focused on OLTs which indeed make up a vast majority of trackers in the literature. Several different ways have been proposed to classify these trackers, with the most high level one being the twofold division into generative and discriminative trackers [6, 12, 11]. The former are those that generate a set of candidate bounding boxes and find the one that best matches their model of the object's appearance while the latter use machine learning to train a binary classifier to distinguish between the object and the background. Liu et. al [12] further subdivided both classes into three categories corresponding to the use of templates, subspace analysis and spare representations in the former and to feature selection, classifier update and metric learning in the latter. As another example of such classification, Wu wt. al [7] categorized trackers on the basis of the target representation method, search mechanism, model update strategy and utilization of contextual information. Similarly, Yang et. al [6] proposed a classification based on the type of features used by the tracker - gradient, color, texture and spatio-temporal features as well as those that utilize a fusion of multiple features. Yilmaz et. al [26] presented a complete hierarchical taxonomy of tracking methods though it is rather outdated now. Three categories were proposed at the top level based on the object representation type - point, kernel and silhouette. Each of these were divided into two subcategories - point tracking into deterministic and probabilistic methods, kernel tracking into multi view and template based trackers and silhouette tracking into contour evolution and shape matching based approaches. Several of these were further divided into a third and even fourth level.

None of these attempts to classify OLTs have, however, been widely adopted so far nor do they provide any significant insights into the workings of OLTs due to the large variations that are inherent in the techniques adopted by such trackers. As a result, this chapter will only consider two categories of trackers that are of relevance to this study - OLTs and RBTs.

2.1 Online Learning & Detection based Tracking

Since a detailed review of OLTs is not relevant to this work, the coverage of this domain will be limited to brief descriptions of the state of the art OLTs that have been compared with low DOF RBTs (Sec. 9.3) to demonstrate their unsuitability for fast and high precision tracking applications. These trackers can be regarded as a fairly good representative sample of the techniques typically adopted in this field.

2.1.1 Discriminative Scale Space Tracking (DSST)

DSST was introduced by Danelljan et. al [27] as an extension of the minimum output sum of squared error or MOSSE correlation filter based tracker [28] for estimating scale in addition to translation. They first considered learning a single 3D correlation filter for joint translation and scale estimation which, however, proved to be too computationally expensive for real time tracking. Therefore, they proposed a faster version where two different correlation filters are learned - a 2D filter for translation and a separate 1D filter for scaling. The former is formulated exactly as in [28]. The latter is trained by extracting features from a set of patches of different sizes centered at the target location and performing the same non linear least squares optimization in Fourier domain as the translational filter. During tracking, the translation filter is first used to estimate the location of the object center, then the scaling filter is applied at this location to estimate the scale factor. The two filter approach was found to be both more accurate and faster than the single 3D filter method. The PCA-HOG features [29] used there for image representation were also shown to perform better than raw intensity values as used, for instance, in the original MOSSE tracker [28].

2.1.2 Kernelized Correlation Filter (KCF) Tracking

KCF tracker was introduced by Henriques et. al [30, 31]. Similar to DSST, KCF too uses discriminative correlation filters though, unlike DSST, it does not perform scale

estimation. KCF can be seen as extending the MOSSE tracker [28] with kernels, though the underlying idea is more general and can be applied to any regularized least squares classifier. This idea is based on the observation that a great deal of redundancy exists in the sampled patches extracted from around the target patch that serve as negative samples for training the classifier in discriminative tracking. It was further noted that, if dense sampling is used, i.e. all possible translated patches are extracted by performing cyclic shifts of the base sample, the training matrix generated by stacking the samples in rows has a circulant structure which greatly speeds up many operations involving it. Finally, kernel functions that satisfy certain conditions were proven to preserve the circulant structure of matrices when used to transform these to carry out non linear regression through the kernel trick. As a result, kernelized non linear regression was shown to be achievable at the same computational cost as linear regression. The speed up results from the fact that all circulant matrices, irrespective of the base sample, are diagonalized by the discrete Fourier transform which allows otherwise costly operations like matrix inversion to be performed on the diagonal elements. In its initial version [30], KCF was formulated for single channel features and only raw pixel intensities were used. The later version [31], however, extended it for multi channel features and was shown to offer significant performance improvement with HOG features.

2.1.3 Clustering of Static-Adaptive Correspondences for Deformable Object Tracking (CMT)

CMT was introduced by Nebehay & Pugfelder [32] and uses a part based approach that somewhat resembles that of RKLT [33]. The overall tracker is made up of a static and an adaptive component. The former establishes correspondence between the initial set of key points detected in the first frame and those in the current frame using a global search process [34]. The latter only estimates the current positions of the key points from the previous frame by computing their sparse optical flow [16] while using the forward-backward error criterion [35] to exclude potentially incorrect correspondences. The combined correspondences from both components are then clustered using a pairwise geometric dissimilarity measure to perform agglomerative clustering. This measure is based on the "geometric compatibility" between the two correspondences that is defined as the difference in the Euclidean distances between the initial and transformed positions of the key points generated by applying

a consistent similitude transformation estimated from the combined correspondences. The largest cluster thus created is taken to contain key points on the object while all remaining ones are relegated to the background. Finally, this set is augmented by candidate correspondences whose geometric dissimilarity from it is less than a threshold. The augmented set is used for computing a similitude transform that, when applied to the initial bounding box, gives the current tracker location.

2.1.4 Structured Output Tracking with Kernels (Struck)

Struck was introduced by Hare et. al [36, 37]. It uses a modified tracking-by-detection framework where a multi class structured output SVM classifier [38] is trained to directly provide the geometric transformation between consecutive frames so that it acts like a prediction function instead. This is opposed to the usual approach of training a binary classifier with positive and negative samples, evaluating it on a set of candidate patches and choosing the one with the maximum response. The tracker can thus dispense with the generation of labels for training samples which is often responsible for poorly trained classifiers since no principled way exists to perform the labeling and heuristics have to be used instead. The classifier is also provided with direct access to the transformations so these can be incorporated into the learning algorithm to improve it further.

A disadvantage of the change in search space is that the process of training the SVM now involves solving a large quadratic program with many constraints. Struck handles this optimization in real time by using the LaRank solver of Bordes et al [39], based on sequential minimal optimization, which enables the large quadratic program to be broken down into a set of smaller programs that can each be solved analytically. Struck also adopts a budgeting mechanism to further facilitate the real time evaluation of the SVM by compensating for the curse of kernelisation that causes the number of support vectors and thus the evaluation time to grow in an unbounded manner as more training samples are added. After a certain limit is reached, the addition of a new support vector causes an existing one, selected heuristically, to be removed and the coefficients of remaining support vectors to be adjusted accordingly. The original version of Struck [36] could only handle translation but was extended in [37] to estimate scale too.

2.1.5 Multiple Instance Learning (MIL) Tracking

MIL tracker was introduced by Babenko et. al [40, 41] and is also based on the tracking-by-detection framework like Struck but uses an MIL classifier [42] instead of SVM. MIL too tries to solve the problem of the inherent ambiguity in the labeling of samples but, unlike Struck that did away with labeling altogether, it uses a weak form of labeling where the classifier is presented with a set of patches and a label is provided for the entire set (known as a "bag") rather than each individual sample. A bag is labeled as positive if it contains at least one positive sample and negative otherwise. This classifier was adapted for online learning using a boosting based approach similar to the MILBoost [43] and online AdaBoost [43, 44] algorithms.

Boosting works by combining the outputs from a set of weak classifiers to produce an additive strong classifier. During tracking, first the strong classifier trained in the last frame is used to update the tracker location by evaluating its response within a search radius s and choosing the location with the maximum response. Next, positive samples are extracted from a radius $r < s$ around this location and negative samples from the annular region with radius $> r$ and $< \beta$ where $\beta > r$. All positive samples are placed in one bag while each negative sample goes into a separate bag. These are used for training the boosting classifier by maximizing the log likelihood of bags where the probability of a bag being positive is estimated in terms of that of its instances using the Noisy-OR model. Finally, the response of each weak classifier is taken to be the log of odds ratio produced using a Haar like feature and a couple of Gaussian distributions that are learned online.

2.1.6 Tracking-Learning-Detection (TLD)

TLD was introduced by Kalal et. al [45] as a long term tracker capable of detecting when the object is no longer present in the scene as well as of reinitializing itself when needed. As its name suggests, TLD involves running three components simultaneously - a tracking component that uses the median flow tracker [35] to track the object between consecutive frames, a detection component that searches the entire image for possible matches of the object appearance learned so far and a learning component that consolidates the outputs of the other two components to estimate their failures and improve future detections. The latter uses P-N learning to generate positive and negative training samples for the classifier used by the detector module.

P-N learning uses a labeled and an unlabeled set of samples as inputs to train

the classifier through a form of iterative bootstrapping. The trained classifier from the previous iteration is first used to classify the unlabeled samples. These are then analyzed by two kinds of "experts" - a P-expert and an N-expert that respectively estimate the false negatives and false positives and add them to the labeled training set relabeled as positive and negative samples respectively. The P-expert exploits temporal structure in the video by relabeling negative detections corresponding to the tracker location as positive. This improves the classifier's generalization ability by providing positive training examples containing new object appearances. The N-expert exploits spatial structure by relabeling positive detections far away from the tracker location as negative which helps to improve the classifier's discriminative ability by adding novel background patches to the negative training set. Both experts also make mistakes like the detector but their independence allows these to compensate each other.

2.1.7 Realtime Compressive Tracking (RCT)

RCT was introduced by Zhang et. al [46] and uses principles from compressive sensing theory [47] to represent the object patch with very sparse features in the compressed domain. A set of rectangle filters are first used to extract Haar like features [48] at multiple scales which are concatenated into a very high dimensional feature vector. The dimensionality of this vector is then reduced by applying random projection through a very sparse Gaussian matrix containing only 4 non zero entries in each row. This matrix is computed off line and remains fixed during tracking so the computational load of the dimensionality reduction step is very small. The tracker itself works within the standard tracking-by-detection framework where nearby and far away samples from the target location are extracted in each frame and used respectively as positive and negative training examples for a naive Bayes classifier. This classifier is then evaluated in a small search window to find the patch with the maximum response as the target location in the next frame. The main advantage of this tracker lies in its ability to utilize almost all the information from the very high dimensional multi scale features without suffering from the curse of dimensionality.

2.1.8 Fragments based Tracking (FragTrack)

FragTrack was introduced by Adam et. al [49] and is the oldest tracker considered in this section. This is also the only one that does not have an online learning

component. FragTrack uses a histogram based similarity measure to detect the object in each frame. In order to overcome the problem of spatial information loss that such measures typically present, the template is divided into a number of fragments or sub patches and a histogram is computed for each sub patch. Unlike conventional parts based methods, these fragments are selected arbitrarily and not based on a model of the object. To locate the object in the current frame, a number of candidate patches in a search radius around the last known location are considered. Each of these is also divided into sub patches similar to the template and histograms are computed for all the sub patches. The overall similarity score for a candidate patch is then obtained by computing the dissimilarity between the histograms of corresponding sub patches in the template and in that patch, arranging the dissimilarity scores in increasing order and choosing the score corresponding to the 25^{th} percentile. This method of combining the dissimilarity scores was inspired by the least median of squares robust estimation method [50] and was chosen over simpler approaches like the sum of scores in order to minimize the impact of outliers. The Earth Mover's Distance [51] is used as the histogram similarity measure. Since this tracker involves computing a very large number of histograms, its real time operation is only possible due to the integral histogram approach [52] that converts the computation of the histogram of any rectangular subregion in an image into a constant time operation once the integral image has been computed for each bin.

2.1.9 Generic Object Tracking Using Regression Networks (GOTURN)

GOTURN was introduced by Held et. al [53] and is the only OLT here that uses deep learning. This trains a deep convolutional neural network (CNN) to learn generic relationships between the change in object appearance and the motion it undergoes between consecutive frames so that it can directly regress on the object location by comparing patches extracted from the previous and current frames in the tracking sequence. The network architecture consists of two branches with each having 5 pre-trained convolutional layers from CaffeNet [54] that were trained on the ImageNet dataset [55]. The outputs of both branches are feature vectors corresponding to their input patches. These are concatenated into a single vector used as input for three fully connected layers that are trained entirely off line using both labeled videos and still images. The output layer has 4 nodes that produce the coordinates of the top left

and bottom right corners of the bounding box corresponding to the tracker location.

Each training instance consists of a pair of image patches - one centered on the object and one where the object has been translated and scaled. For videos, these patches are generated from consecutive frames by taking the object centered patch from the first frame and several random crops from the second. For still images, both patches are generated from the same image but using the same process as for videos. Training on videos helps to make the network invariant to challenges like background clutter, illumination variations, pose changes and occlusions while training on still images of a large variety of objects helps to prevent over fitting on the relatively limited selection of objects in the videos. This enables the network to generalize well to completely novel objects not seen during training. The network is trained to favor smoother motions by generating the random crops for training using Laplace distributions that were found to represent the changes in object location and size between consecutive frames in real videos better than Gaussian distributions. At runtime, the patch corresponding to the tracker location in the previous frame along with a crop of the search region in the current image are used as inputs to the two convolutional branches of the network and its output is the predicted location of the object relative to this crop.

2.2 Registration based Tracking

This sub field of tracking is widely regarded as having been pioneered by Lucas & Kanade in 1981 [16] after whom the LK tracker is named. The formulation introduced there was the so called forward additive variant (FALK) of this tracker where the search for the warp is carried out in the current image and incremental updates to the warping parameters are added to their old values to obtain the new ones. This remained the only formulation for over 15 years during which tracking of point features, also known as optical flow estimation, seemed to have been popular and many methods were suggested for selecting the most suitable points for tracking [56, 57, 58, 59] along with the use of coarse to fine tracking approach [60, 61, 19, 62] to handle large motions better. A related SM called difference decomposition [63] was also introduced as a generalization of FALK.

The next major advance in this field was in the work of Hager & Belheumer [64, 22] who proposed a way to do away with having to compute the image gradient in each frame by approximating it with the gradient of the first image. This approach has

since been dubbed the inverse additive formulation of LK (IALK). This was also one of the first applications of photometric parameter estimation to make the tracking more illumination invariant. This latter aspect was further extended in [65]. A generalization of IALK, somewhat analogous to the one for FALK [63], was also proposed in the form of the hyperplane approximation technique [66].

IALK was followed by the compositional variant of the forward algorithm (FCLK) applied to image mosaicing [61]. FCLK was shown to offer the advantage of requiring the gradients of the warping function to be computed at the identity element which significantly simplified the corresponding expressions. The inverse variant of this algorithm (ICLK) was introduced soon thereafter by Baker & Matthews [67, 25] who showed that even further computations savings can be obtained by searching the initial image for the warp rather than the current one as this meant that the image gradient and the Hessian of the objective function needed to be computed only once. They also unified all other variants of LK within a common framework and showed these to be equivalent both experimentally and theoretically to first order approximations. This work was followed by a series of technical reports [68, 69, 70, 71, 72] where these algorithms were examined in detail along with several variations on the search methods [68], error metric [69], illumination models [70], use of priors [71] and extension to 3D tracking [72]. The contents of these reports were somewhat similar to the current work though significantly more limited in scope as they considered only different types of gradient descent SMs and many variants and extensions of the same underlying sum of squared difference (SSD) AM.

The only newer LK variant introduced since has been the efficient second order minimization (ESM) approach of Benhimane & Malis [73]. This combined the forward and inverse compositional methods by using the gradient information from both the current and initial images. This combination was shown to provide a second order convergence as opposed to the first order convergence of all previous variants. There have since been many different applications and extensions of this method without any change to the core algorithm. These include applications in visual servoing [74, 75, 76, 77, 78], SLAM [79, 80], augmented reality [81], 3D reconstruction [82] and camera pose estimation [83] along with extensions for coarse to fine estimation [75], illumination models [84, 85] and use with multi channel [77, 85], catadioptric [86] and omni directional [78] imagery. There have also been several relatively minor improvements to handle specific challenges like motion blur [87, 81] and effective resolution degradation [88]. ESM has further been shown to be the middle point of

a continuum of methods that estimate the warp simultaneously from both the initial and current images [89], of which ICLK and FCLK are the two extremes.

Apart from these, the main developments in this field in the last two decades have been the adaptation of several new AMs and stochastic SMs for RBT. The AMs include mutual information (MI) [90, 91, 92, 93, 94], normalized cross correlation (NCC) [3], cross cumulative residual entropy (CCRE) [4] and sum of conditional variance (SCV) [95, 96, 97] along with its reversed (RSCV) [98] and localized (LSCV) [5] variants. The stochastic SMs include nearest neighbor search (NN) [98], particle filters (PF) [99, 100, 101, 102, 103, 104, 105, 106, 1] and RANSAC [33]. Many of the PF based trackers also incorporated online learning in the AM, for instance, by applying PCA [99] on a dynamically adapted set of basis images. There have also been several AMs introduced in medical image registration that have not yet been applied for high DOF RBT. These include ratio image uniformity (RIU) [107, 108, 109, 110], partitioned intensity uniformity (PIU) [110, 108] and normalized gradient fields (NGF) [111, 112, 113, 114]. Another measure called structural similarity (SSIM) [115], that has been popular for evaluating the quality of image compression algorithms [116], has been applied for low DOF tracking [117, 118, 119, 120, 121] but not for RBT.

As the so called robust models [4] - those that cannot be expressed as some variation of SSD - have become more widely used, generalizations have been proposed for ICLK [122] and ESM [123] to handle these. Other advances in RBT in recent years include extensions like the ability to use feature histograms instead of pixels [124], estimate 3D object pose by enforcing planar constraints [125], handle images in the Fourier domain [126] or those containing radial distortion from wide field of view cameras [127], detect and remove false features at low computational cost [128], perform higher order approximation of the Taylor expansion [129], deal with deformable objects through active appearance models [130, 131] and incorporate maximum likelihood classifiers [132], known camera parameters [133], sparse representation [134] or subspace constraints [135] to improve tracking performance. There has also been some progress in obtaining a better theoretical understanding of the different variants of LK [136].

2.2.1 Fragmentation

Though a lot of contributions have been made in this domain, the ideas in different works have not been well connected or related to each other so that the overall

progress in the field has become somewhat fragmented. The proposed decomposition of RBTs into three sub modules can help to solve this problem. Many of the above mentioned works have presented novel ideas for only one of these submodules while using existing methods, often selected arbitrarily, for the rest. For instance, Hager & Belheumer [22], Shum & Szeliski [137], Baker & Matthews [67] and Benhimane & Malis [73] all introduced new variants of FALK [16] but only tested these with SSD AM. Similarly, Dick et. al [98], Zhang et. al [33] and Kwon et. al [1] combined their respective stochastic SMs with only a single AM - RSCV, SSD and NCC respectively - though the latter also performed online learning through PCA.

Conversely, Richa et. al [95], Scandaroli et. al [3] and Dame et. al [92] introduced SCV, MI and NCC as AMs but combined these only with a single SM - ESM in the first one and ICLK in the other two. Even more recent works that use illumination models (ILMs) (Sec. 5.3), including Bartoli [138, 139], Silvera & Malis [84, 77, 85] and Silvera [140], have combined their respective ILMs with only a single SM in each case - ICLK in the first one and ESM in all others. Finally, most combinations of SMs and AMs have been tested with only one SSM - either homography [73, 76, 92] or affine [67, 99]. In fact, Benhimane & Malis [73] mentioned that their SM only works with SL3 SSM (Sec. 6.6) though experiments conducted in this work have shown (Sec. 9.3) that it performs equally well with others.

2.3 Summary

This chapter presented brief descriptions of the OLTs tested in Sec. 9.3 along with a summary of the diverse contributions made to RBT in the last three and a half decades. It also exemplified the resultant fragmentation in this field that the proposed formulation aims to resolve.

Chapter 3

Registration based Tracking

RBT is a type of visual tracking where the object is located in each image in a sequence by finding the warp or geometrical transformation that, when applied to the object patch within the image, will align or register it with the template or the target patch (Fig. 3.1). The latter is typically extracted from the first image when the tracker is initialized though it can also be updated as tracking progresses to account for changes in the object's appearance. The task of image alignment or registration is equivalent to getting the warped patch to appear as similar as possible to the target patch which in turn is accomplished by maximizing some measure of similarity between the patches.

Before casting this as an optimization problem where the proposed decomposition can be applied, relevant notation is introduced that will be used in the rest of this work.

3.1 Notations

Let $I_t : \mathbb{R}^2 \mapsto \mathbb{R}$ refer to an image captured at time t so that a video stream is modeled as a sequence of images $\{I_t | t \geq 0\}$ and the tracker is initialized by selecting the target object's location in I_0. I_t is treated as a smooth function of real values using sub pixel interpolation [93] for non integral locations. The image patch containing N pixels corresponding to the tracked object's location in I_t is denoted by $\mathbf{I_t}(\mathbf{x_t}) = [I_{1t}, I_{2t}, ..., I_{Nt}]^T \in \mathbb{R}^N$ where $\mathbf{x_t} = [\mathbf{x}_{1t}, ..., \mathbf{x}_{Nt}] \in \mathbb{R}^{2\times N}$ with $\mathbf{x}_{kt} = [x_{kt}, y_{kt}]^T \in \mathbb{R}^2$ being the Cartesian coordinates of the k^{th} point in image space and $I_{kt} = I_t(x_{kt}, y_{kt})$ the corresponding pixel value.

Note that $N = N_x \times N_y$ where N_x and N_y are the sampling resolutions in the

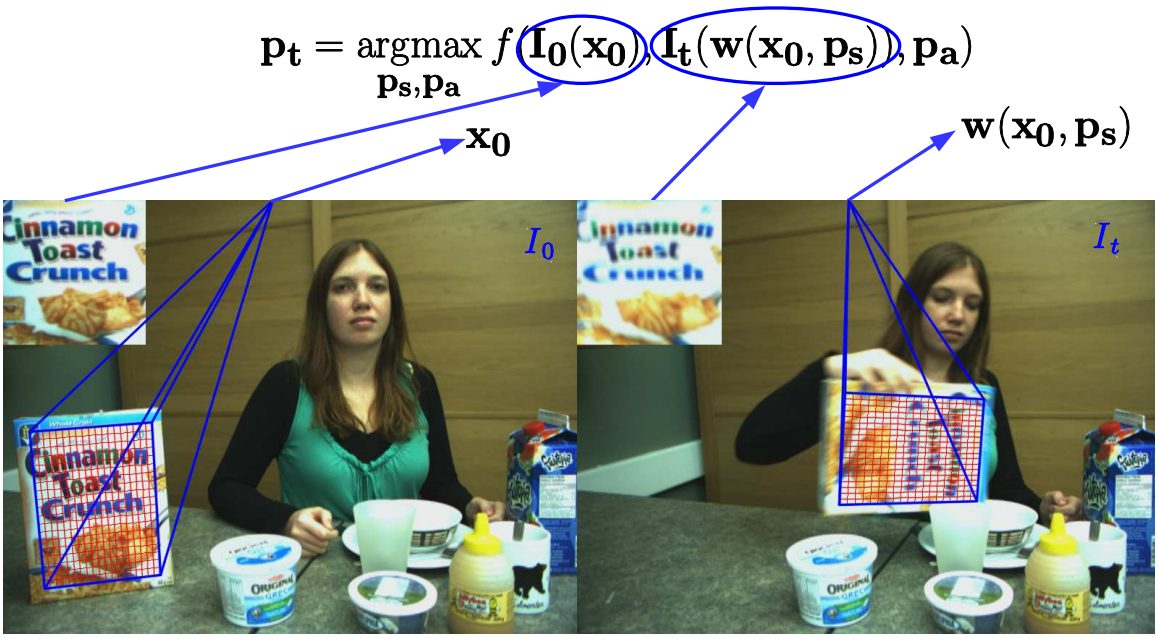


Figure 3.1: Two frames from a sequence showing the different components of registration based tracking. The grid of points $\mathbf{x}$ where pixel values are extracted are shown in red and the corresponding patches are in the top left corners. For better visibility, the grid is sampled at 25×25 though higher resolutions may be used in practice.

horizontal and vertical directions respectively and can be chosen independently of the actual size of the image patch in pixels. Alternatively, $\mathbf{x}_{kt}$ can be seen as the intersection locations of a regularly spaced grid of points that is superimposed over the actual object patch (Fig. 3.1). It is assumed that $\mathbf{x_t}$ contains points from this 2D grid in *row major* order so that $k = k_x + (k_y - 1)N_x$ for $1 \leq k_x \leq N_x$ and $1 \leq k_y \leq N_y$. Using higher values of N_x, N_y typically leads to a smoother and easier to optimize objective function at the cost of higher computational complexity. In practice, a resolution of 50×50 provides a good compromise between accuracy and speed (Sec. A.1.3).

Further, $\mathbf{w}(\mathbf{x}, \mathbf{p_s}) : \mathbb{R}^{2\times N} \times \mathbb{R}^S \mapsto \mathbb{R}^{2\times N}$ denotes a warping function of S parameters $\mathbf{p_s} = [p_{1s}, ..., p_{Ss}]$ that represents the set of allowable image motions of the tracked object by specifying the deformations that can be applied to $\mathbf{x_0}$ to align $\mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_s}))$ with $\mathbf{I_0}(\mathbf{x_0})$. It can be noted that S is equivalent to the DOF of image motion. The $\mathbf{p_s}$ that corresponds to the optimal warp found by the tracker for I_t will be referred to as $\mathbf{p_{st}}$ with $\mathbf{x_t} = \mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}})$. An estimate for $\mathbf{p_{st}}$ for which an incremental update is sought by the tracker, typically $\mathbf{p_{s(t-1)}}$, will be referred to as $\hat{\mathbf{p}}_{\mathbf{st}}$ and the corresponding

$\mathbf{x}$ as $\hat{\mathbf{x}}_\mathbf{t}$. Where clear from context, $\mathbf{I_0}(\mathbf{x_0})$ and $\mathbf{I_t}(\hat{\mathbf{x}}_\mathbf{t})$ might also be referred to simply as $\mathbf{I_0}$ and $\mathbf{I_t}$ respectively.

It is theoretically possible to estimate the x, y translations for each element $\mathbf{x_{kt}}$ of $\mathbf{x_t}$ so that $S = 2 \times N$ and the DOF are thus limited only by the sampling resolution. The resultant process is known as optical flow estimation [141, 142, 143] when performed independently for each $\mathbf{x_{kt}}$ using pixels from a small patch around it and non parametric warping [144] when done simultaneously for all components of $\mathbf{x_t}$ using $\mathbf{I_t}(\mathbf{x_t})$. The former is not the subject of this work while the latter is usually not tenable in practice due to the number of unknowns ($2 \times N$) exceeding the available equations (N). As a result, this work focuses only on parametric warping functions, common examples of which include homography ($S = 8$), affine ($S = 6$), similitude ($S = 4$), isometry ($S = 3$) and translation ($S = 2$) [15].

Finally $f(\mathbf{I}^*, \mathbf{I^c}, \mathbf{p_a}) : \mathbb{R}^N \times \mathbb{R}^N \times \mathbb{R}^A \mapsto \mathbb{R}$ refers to a function of A parameters that measures the similarity between two patches - the reference or template patch $\mathbf{I}^*$, typically extracted from I_0, and a candidate patch $\mathbf{I^c}$ extracted from I_t. RBT works on the assumption that the similarity between two patches bears a positive correlation with the likelihood of them corresponding to different poses of the same object so that maximizing f is synonymous to finding the object patch. Popular examples of f with $A = 0$ include sum of squared differences (SSD) [22], normalized cross correlation (NCC) [3] and mutual information (MI) [93]. So far, the only examples with $A \neq 0$, to the best of the author's knowledge, are those with an illumination model (**ILM**) [65, 70, 84, 139] where f is expressed as $f(\mathbf{I}^*, \mathbf{g}(\mathbf{I^c}, \mathbf{p_a}))$ with $\mathbf{g} : \mathbb{R}^N \times \mathbb{R}^A \mapsto \mathbb{R}^N$ accounting for differences in lighting conditions under which I_0 and I_t were captured. As for $\mathbf{p_s}$, $\mathbf{p_{at}}$ and $\hat{\mathbf{p}}_\mathbf{at}$ respectively refer to the optimal $\mathbf{p_a}$ for I_t and its estimate. It should be noted that vectors and matrices, including functions that output these, are denoted with bold fonts and scalars with standard ones.

Before concluding this section, some notational clarity is provided regarding mixed expressions containing linear combinations of matrices, vectors and scalars, many of which will be needed to succinctly express the derivatives of f and $\mathbf{w}$ in chapters 5 and 6. Any binary operation - addition, subtraction, multiplication or division - involving a scalar as one operand and a vector or matrix as the other indicates that the operation is performed between the scalar and *each element* of that vector/matrix. Similar rule applies with assignments - a scalar being assigned to a vector/matrix will result in each element of that vector/matrix being set equal to that scalar. The convention of denoting the said scalar with bold font will also be omitted to main-

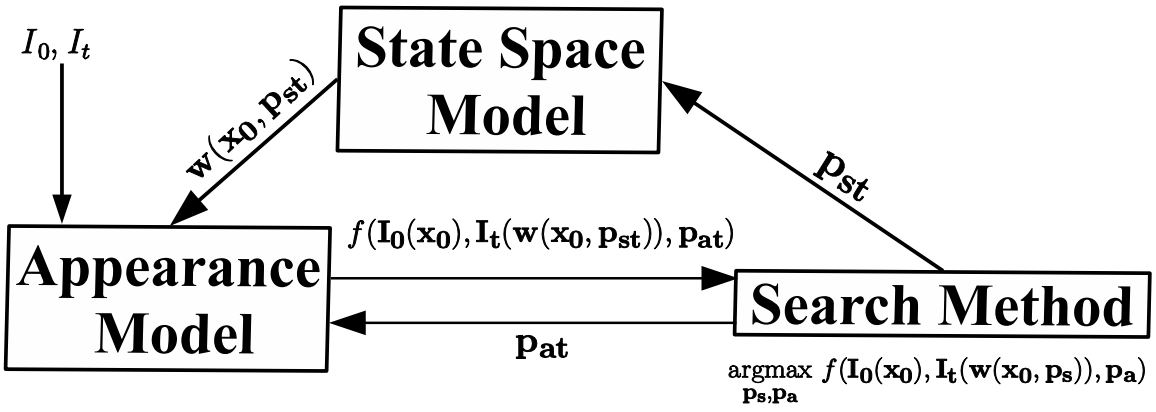


Figure 3.2: Decomposition of registration based tracking showing the interaction between the resultant sub modules

tain consistency with other binary operations. Further, an expression involving the division of a vector/matrix by another indicates element-wise division and the two operands will therefore always have the same dimensions. Finally, the addition or subtraction between a vector and a matrix, unless otherwise indicated, denotes **row-wise** operation on the matrix, i.e. the operation is performed element-wise between the vector and each row of the matrix.

3.2 Decomposition

Using the above notation, RBT can be formulated (Eq. 3.1) as a search problem where the goal is to find the optimal parameters $\mathbf{p_t} = [\mathbf{p_{st}}, \mathbf{p_{at}}] \in \mathbb{R}^{S+A}$ that maximize the similarity, measured by f, between the target patch $\mathbf{I}^* = \mathbf{I_0}(\mathbf{x_0})$ and the warped image patch $\mathbf{I^c} = \mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}}))$, that is,

$$\mathbf{p_t} = \underset{\mathbf{p_s},\mathbf{p_a}}{\text{argmax}}\ f(\mathbf{I_0}(\mathbf{x_0}), \mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_s})), \mathbf{p_a}) \tag{3.1}$$

As has been observed before [15, 4], this formulation gives rise to an intuitive way to decompose the tracking task into three modules - the similarity metric f, the warping function $\mathbf{w}$ and the optimization approach. These can be designed to be semi independent in the sense that any given optimizer can be applied unchanged to several combinations of methods for the other two sub modules which in turn interact only through a well defined and consistent interface. In this work, these sub modules are respectively referred to as appearance model (AM), state space model (SSM) and

search method (SM).

Fig. 3.2 shows the effective flow of information between the three sub modules though in practice SM serves as the interface between AM and SSM which do not interact directly. For each new frame I_t:

1. SM computes the optimum parameters for $\mathbf{w}$ and f and passes these respectively to the SSM and AM.

2. SSM then warps the initial grid points $\mathbf{x_0}$ using these parameters and passes the resultant points $\mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}})$ to the AM

3. AM extracts the pixel values at these points and computes the similarity of the resultant patch with the template using $\mathbf{p_{at}}$ which it then passes back to the SM.

4. SM uses this similarity to find the parameters $\mathbf{p_{t+1}}$ that maximize it for the next frame I_{t+1}.

It should be noted that the proposed decomposition does not imply that *all* methods for any two of these sub modules can be combined with each other in a meaningful way. For instance, there are SMs like difference decomposition [63] and hyperplane approximation [66] that only make sense with a specific AM - SSD (Sec. 5.1.1) in both these cases. Similarly, there are many AMs, especially those that employ online learning [145, 99, 146, 147, 101, 100, 148], that are not differentiable and so cannot be used with the gradient based SMs (Sec. 4.1) that are the most popular ones in the literature. The scope of this current work is limited to only those SMs and AMs that make sense when combined with each other and more general formulations are deferred to future extensions.

3.3 Summary

This chapter provided details of the algebraic notation used in the remainder of this thesis along with an overview of the proposed decomposition of RBT and the interaction between the resultant sub modules. More detailed descriptions of the three sub modules along with examples follow in the next three chapters.

Chapter 4

Search Methods

SM is the optimization procedure that searches for the warped patch in I_t that best matches the template $\mathbf{I}^*$. Many different types of SMs have been used in engineering and statistical applications [149, 150] including methods for both local and global optimization. Local optimization is often carried out using gradient based methods like steepest descent, conjugate gradient, Newton's method and quasi Newton methods. Global optimization, on the other hand, fares better with stochastic methods that include simulated annealing, particle swarm optimization and evolutionary techniques like genetic algorithms. Due to the real time nature of visual tracking, however, most of these latter are too slow to be applied here though they have been used successfully for offline image registration [151]. Even amongst the former, only the faster methods can be used. This also rules out constrained optimization techniques like regularization though these too have been shown to make image registration more stable and robust [144].

The SMs considered here can be divided into three main categories including the two mentioned above - gradient descent and stochastic search - as well as composite SMs that include elements from both.

4.1 Gradient Descent (GD) Search

This category comprises SMs where the first and second order derivatives of f with respect to $\mathbf{p}$ are used to determine a search direction in which iterative steps are taken till convergence occurs. This approach has been the most popular one for RBT in literature due to its speed and simplicity and forms the basis of the classic LK tracker [16]. As shown by Baker & Matthews [25], this algorithm can be formulated

in four different ways depending on which image is searched for the warped patch - I_t or I_0 - and how the parameters of the warping function are updated in each iteration - additive or compositional. The four resulting variants are called forward additive (**FALK**) [16], inverse additive (**IALK**) [22], forward compositional (**FCLK**) [15] and inverse compositional (**ICLK**) [67]. These were analyzed both theoretically and experimentally and shown to be equivalent to first order terms while also exhibiting similar convergence behavior in static tests. Baker & Matthews [25] also tested several approximations to the Newton's method including Gauss Newton (**GN**), Levenberg Marquardt (**LM**), steepest descent and diagonal Hessian. Of these, GN and LM were found to perform similarly and also better than the rest including even the true Newton's method. Hence, these are the only formulations tested here. A forward compositional variant of the steepest descent algorithm (FCSD) described in [25, Section 4.3] was also implemented in MTF and tested but performed very poorly with all but 2 DOF SSM and so has not been considered here.

A more recent update to this approach was in the form of the efficient second order minimization (**ESM**) [73] technique that attempted to make the best of both inverse and forward formulations by using the mean of the Jacobians computed from I_0 and I_t. In addition to RBT, ESM has also found successful application in visual servoing [76, 77, 78] and SLAM [79, 80]. It has since been shown [89, 152] that ESM is in fact the middle point of a continuum of methods - of which ICLK and FCLK are the two ends - where I_0 and I_t are warped simultaneously to find the optimal $\mathbf{p}$. This latter generalization has given rise to additional variants of LK like symmetric gradient and bidirectional gradient methods. Though these can be regarded as distinct SMs within the proposed framework, they are not considered here to keep the scope of this study manageable.

In addition to these relatively significant strides in the field of GD based SMs, several extensions have also been proposed to these SMs to handle specific challenges like motion blur [87], effective resolution degradation [88], better Taylor series approximation [129] and selection of optimal features [57, 58, 59] or patch subsets [153, 154]. These too can be fit within the framework, either as distinct SMs or as variants thereof, but are not considered here for the same reason.

Finally, it may be noted that, in analogy with the forward and inverse formulations of LK, ESM too can have an additive variant along with the compositional one that has been used in literature. Though this has been implemented in MTF as Additive ESM or AESM, it was not found to perform as well as ESM and, not having furnished

any useful insights about this SM, has been excluded from this study.

The varied approaches adopted by the five SMs in this category have one thing in common - they solve Eq. 3.1 iteratively by estimating an incremental update $\Delta\mathbf{p_t}$ to the optimal parameters $\mathbf{p_{t-1}}$ at time $t-1$ using some variant of the Newton's method as:

$$\Delta\mathbf{p_t} = -\hat{\mathbf{H}}^{-1}\hat{\mathbf{J}}^T \tag{4.1}$$

where $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ respectively are estimates for the Jacobian $\mathbf{J} = \partial f/\partial \mathbf{p}$ and the Hessian $\mathbf{H} = \partial^2 f/\partial \mathbf{p}^2$ of f w.r.t. $\mathbf{p}$. For any formulation that seeks to decompose this class of trackers (among others) in the manner described in the previous chapter, the chain rule for first and second order derivatives is indispensable and relevant expressions will be introduced next to make the subsequent descriptions easier. For the sake of simplicity, it has been assumed that $A = 0$ (or $\mathbf{p} = \mathbf{p_s}$) in expressions that follow in the remainder of this section though extensions for $A \neq 0$ are straightforward. The Jacobian can be decomposed as:

$$\mathbf{J} = \frac{\partial f(\mathbf{I}(\mathbf{w}(\mathbf{p})))}{\partial \mathbf{p}} = \frac{\partial f}{\partial \mathbf{I}} \nabla\mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} \tag{4.2}$$

where $\dfrac{\partial f}{\partial \mathbf{I}}$ is the $1\times N$ Jacobian of f w.r.t. $\mathbf{I}$, $\nabla\mathbf{I}$ is the $N\times 2$ spatial gradient of $\mathbf{I}$ and $\dfrac{\partial \mathbf{w}}{\partial \mathbf{p}}$ is the $2\times S\times N$ Jacobian of the $\mathbf{x}$ w.r.t. $\mathbf{p}$. Note that the multiplication between the last two terms in Eq. 4.2 is carried out on a per pixel basis by multiplying each row of $\nabla\mathbf{I}$ with the corresponding $2\times S$ block of the tensor $\dfrac{\partial \mathbf{w}}{\partial \mathbf{p}}$ and stacking the results into the rows of an $N\times S$ matrix $\dfrac{\partial \mathbf{I}}{\partial \mathbf{p}}$. This detail is assumed implicit in the notation for brevity.

The generic decomposition for the Hessian is given as:

$$\mathbf{H} = \frac{\partial \mathbf{I}}{\partial \mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I}}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I}}{\partial \mathbf{p}^2} = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla\mathbf{I}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \nabla\mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \frac{\partial f}{\partial \mathbf{I}} \left(\nabla^2\mathbf{I} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla\mathbf{I} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \right) \tag{4.3}$$

It follows from these expressions that a reasonable way to divide the computation of $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ between AM and SSM is to have the former compute terms involving $\mathbf{I}$ and f ($\nabla\mathbf{I}$, $\nabla^2\mathbf{I}$, $\partial f/\partial \mathbf{I}$ and $\partial^2 f/\partial \mathbf{I}^2$) while the latter computes those with $\mathbf{w}$ ($\partial \mathbf{w}/\partial \mathbf{p}$, $\partial^2 \mathbf{w}/\partial \mathbf{p}^2$). Further, these generic expressions do not give the whole scope of the decompositions since the exact forms of $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$ as well as the way these are split vary for different variants of LK. Since f is a function of *two* patches, $\mathbf{J}$ and $\mathbf{H}$

depend on which of the two corresponding images is being searched for $\mathbf{p_t}$ as only the patch corresponding to this image can be regarded as a function of $\mathbf{p}$ and this in turn is the one to be regarded as the variable while differentiating f, i.e. $\mathbf{I} = \mathbf{I}^*$ if I_0 is being searched and $\mathbf{I^c}$ otherwise. Also, due to the way the Taylor series expansion is formulated, the exact forms of the image derivatives $\frac{\partial \mathbf{I}}{\partial \mathbf{p}}$ and $\frac{\partial^2 \mathbf{I}}{\partial \mathbf{p}^2}$ too depend on whether additive or compositional updates are used. The reader is referred to Baker & Matthews [25] for more details though formulations relevant to the functions in MTF (Tables 7.1 and 7.3), including several extensions to [25], are also presented next.

4.1.1 Jacobian

Denoting $\mathbf{w}(\mathbf{x}, \mathbf{p_s})$ with $\mathbf{w}(\mathbf{p})$ for conciseness ($A = 0$ and $\mathbf{x}$ is constant in this context), the formulations for $\hat{\mathbf{J}}$ used by FALK and FCLK are given as:

$$\hat{\mathbf{J}}_{fa} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t}))} \nabla \mathbf{I_t}|_{\mathbf{x}=\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t})} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\hat{\mathbf{p}}_\mathbf{t}} \tag{4.4}$$

$$\hat{\mathbf{J}}_{fc} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t}))} \nabla \mathbf{I_t}(\mathbf{w})|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p_0}} \tag{4.5}$$

where $\nabla \mathbf{I_t}(\mathbf{w})$ in Eq. 4.5 refers to the gradient of I_t warped using $\hat{\mathbf{p}}_\mathbf{t}$, i.e. I_t is first warped back to the coordinate frame of I_0 using $\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t})$ to obtain $I_t(\mathbf{w})$ whose gradient is then evaluated at $\mathbf{x} = \mathbf{x_0}$. It can be further expanded [25] as:

$$\nabla \mathbf{I_t}(\mathbf{w})|_{\mathbf{x}=\mathbf{x_0}} = \nabla \mathbf{I_t}|_{\mathbf{x}=\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t})} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{x}}\right|_{\mathbf{p}=\hat{\mathbf{p}}_\mathbf{t}} \tag{4.6}$$

where $\frac{\partial \mathbf{w}}{\partial \mathbf{x}}$ is the $2 \times N \times 2 \times N$ derivative of $\mathbf{w}$ w.r.t. $\mathbf{x}$. The multiplication in Eq. 4.6 can be quite complex and computationally expensive to perform in the general case where the warped location of each point in $\mathbf{x}$ depends on one or more of the remaining points. However, for SSMs that perform rigid transformations, including all SSMs considered in this work, this can be computed point wise by multiplying each row of $\nabla \mathbf{I_t}$ with the corresponding 2×2 derivative $\frac{\partial \mathbf{w}(\mathbf{x}_k, \mathbf{p})}{\partial \mathbf{x}_k}$ and stacking the resultant 1×2 vectors in rows.

Since $\nabla \mathbf{I_t}$ is usually the most computationally expensive part of $\mathbf{J}_{fc}$ and $\mathbf{J}_{fa}$, the so called inverse methods - ICLK and IALK - replace this with the gradient of $\nabla \mathbf{I_0}$

for efficiency as the latter only needs to be computed once. The specific expressions for these methods are:

$$\hat{\mathbf{J}}_{ic} = \left.\frac{\partial f}{\partial \mathbf{I}^*}\right|_{\mathbf{I}^*=\mathbf{I_0}(\mathbf{x_0})} \nabla \mathbf{I_0}|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p_0}} \tag{4.7}$$

$$\hat{\mathbf{J}}_{ia} = \left.\frac{\partial f}{\partial \mathbf{I^c}}\right|_{\mathbf{I^c}=\mathbf{I_t}(\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t}))} \nabla \mathbf{I_0}|_{\mathbf{x}=\mathbf{x_0}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1}\right|_{\mathbf{p}=\hat{\mathbf{p}}_\mathbf{t}} \left.\frac{\partial \mathbf{w}}{\partial \mathbf{p}}\right|_{\mathbf{p}=\hat{\mathbf{p}}_\mathbf{t}} \tag{4.8}$$

where the middle two terms in Eq. 4.8 are derived from Eqs. 4.4 and 4.6 by assuming [22] that $\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t})$ perfectly aligns I_t with I_0, i.e. $I_t(\mathbf{w}) = I_0$ so that

$$\nabla \mathbf{I_t}(\mathbf{w}) = \nabla \mathbf{I_0} \tag{4.9}$$

In its original paper [76], ESM was formulated as using a modified form of Eq. 4.5 to compute $\hat{\mathbf{J}}$ where $\nabla \mathbf{I_t}(\mathbf{w})$ was replaced by the mean of $\nabla \mathbf{I_0}$ and $\nabla \mathbf{I_t}(\mathbf{w})$. Since this formulation is only applicable to SSD, however, a generalized version [123, 3] is considered here that uses the *difference* between FCLK and ICLK Jacobians:

$$\hat{\mathbf{J}}_{esm} = \hat{\mathbf{J}}_{fc} - \hat{\mathbf{J}}_{ic} \tag{4.10}$$

4.1.2 Hessian

For clarity and brevity, evaluation points for the various terms have been omitted in the equations that follow as being obvious from analogy with the previous section.

It is generally assumed [25, 76] that the second term of Eq. 4.3 is too costly to compute and too small near convergence to matter and so is omitted to give the so called Gauss Newton (GN) Hessian:

$$\mathbf{H}_{gn} = \frac{\partial \mathbf{I}}{\partial \mathbf{p}}^T \frac{\partial^2 f}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I}}{\partial \mathbf{p}} \tag{4.11}$$

Though $\mathbf{H}_{gn}$ works very well for SSD (and in fact even better than $\mathbf{H}$ [25, 93]), it is well known [93, 3] to *not* work well with other AMs like MI and NCC for which an approximation to the Hessian *after convergence* has to be used by assuming perfect alignment or $\mathbf{I_t}(\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t})) = \mathbf{I_0}(\mathbf{x_0})$. The resultant approximation is here referred to as the **Self Hessian** or $\mathbf{H}_{self}$ and, as this substitution can be made by setting either $\mathbf{I^c} = \mathbf{I_0}(\mathbf{x_0})$ or $\mathbf{I}^* = \mathbf{I_t}(\mathbf{w}(\hat{\mathbf{p}}_\mathbf{t}))$, two forms for $\mathbf{H}_{self}$ are obtained which are respectively deemed to be the Hessians for ICLK and FCLK:

$$\hat{\mathbf{H}}_{ic} = \mathbf{H}^*_{self} = \frac{\partial \mathbf{I_0}}{\partial \mathbf{p}}^T \frac{\partial^2 f(\mathbf{I_0}, \mathbf{I_0})}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I_0}}{\partial \mathbf{p}} + \frac{\partial f(\mathbf{I_0}, \mathbf{I_0})}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p}^2} \tag{4.12}$$

$$\hat{\mathbf{H}}_{fc} = \mathbf{H}^{\mathbf{c}}_{self} = \frac{\partial \mathbf{I_t}}{\partial \mathbf{p}}^T \frac{\partial^2 f(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I}^2} \frac{\partial \mathbf{I_t}}{\partial \mathbf{p}} + \frac{\partial f(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I}} \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2} \tag{4.13}$$

It is interesting to note that $\mathbf{H}_{gn}$ has the exact same form as $\mathbf{H}_{self}$ for SSD (since $\frac{\partial f_{ssd}(\mathbf{I_0}, \mathbf{I_0})}{\partial \mathbf{I}} = \frac{\partial f_{ssd}(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I}} = \mathbf{0}$ (Sec. 5.1.1)) so it seems that interpreting Eq. 4.11 as the first order approximation of Eq. 4.3, as in [25, 93], is incorrect and it should instead be seen as a special case of $\mathbf{H}_{self}$.

$\hat{\mathbf{H}}_{fa}$ differs from $\hat{\mathbf{H}}_{fc}$ only in the way $\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}$ and $\frac{\partial \mathbf{I_t}}{\partial \mathbf{p}}$ are computed for the two, as given in Eqs. 4.14 and 4.15 respectively:

$$\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}(fa) = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \tag{4.14}$$

$$\frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p}^2}(fc) = \frac{\partial \mathbf{w}}{\partial \mathbf{p}}^T \nabla^2 \mathbf{I_t}(\mathbf{w}) \frac{\partial \mathbf{w}}{\partial \mathbf{p}} + \nabla \mathbf{I_t}(\mathbf{w}) \frac{\partial^2 \mathbf{w}}{\partial \mathbf{p}^2} \tag{4.15}$$

where $\nabla^2 \mathbf{I_t}(\mathbf{w})$ can be expanded by differentiating Eq. 4.6 as:

$$\nabla^2 \mathbf{I_t}(\mathbf{w}) = \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{x}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2} \tag{4.16}$$

$\hat{\mathbf{H}}_{ia}$ is identical to $\hat{\mathbf{H}}_{fa}$ except that $\nabla \mathbf{I_0}$ and $\nabla^2 \mathbf{I_0}$ are used to approximate $\nabla \mathbf{I_t}$ and $\nabla^2 \mathbf{I_t}$. The expression for the former is in Eq. 4.8 while that for the latter can be derived by differentiating both sides of Eq. 4.6 after substituting Eq. 4.9:

$$\nabla^2 \mathbf{I_0} = \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^T \nabla^2 \mathbf{I_t} \frac{\partial \mathbf{w}}{\partial \mathbf{x}} + \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2}$$

which gives:

$$\begin{aligned} \nabla^2 \mathbf{I_t}(ia) &= \left(\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \right)^T \left[\nabla^2 \mathbf{I_0} - \nabla \mathbf{I_t} \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2} \right] \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \\ &= \left(\frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \right)^T \left[\nabla^2 \mathbf{I_0} - \left(\nabla \mathbf{I_0} \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \right) \frac{\partial^2 \mathbf{w}}{\partial \mathbf{x}^2} \right] \frac{\partial \mathbf{w}}{\partial \mathbf{x}}^{-1} \end{aligned} \tag{4.17}$$

where the second equality again follows from Eqs. 4.6 and 4.9. Finally, the ESM Hessian corresponding to the Jacobian in Eq. 4.10 is the *sum* of FCLK and ICLK Hessians:

$$\hat{\mathbf{H}}_{esm} = \hat{\mathbf{H}}_{fc} + \hat{\mathbf{H}}_{ic} \tag{4.18}$$

4.1.2.1 Levenberg Marquardt (LM)

All of the aforementioned forms of $\hat{\mathbf{H}}$ can be modified for using the LM method instead of the generalized GN formulation described above by replacing $\hat{\mathbf{H}}$ with $\hat{\mathbf{H}}+\delta\text{diag}(\hat{\mathbf{H}})$ where $\text{diag}(\hat{\mathbf{H}})$ refers to a diagonal matrix populated with the entries in the principal diagonal of $\hat{\mathbf{H}}$ and δ is a non-negative adaptive weight factor. δ is updated at each iteration depending on the performance of its existing value. Let δ_k refer to the value of δ in the k^{th} iteration and $\tilde{\delta} > 1$ to an update factor. If the $\Delta\mathbf{p}$ computed using δ_k causes f to increase, then $\delta_{k+1} = \delta_k\tilde{\delta}$. Conversely, if f decreases, then $\delta_{k+1} = \delta_k/\tilde{\delta}$ and the application of $\Delta\mathbf{p}$ to update $\mathbf{p}$ is also undone. This algorithm is taken unchanged from Baker & Matthews [25] and the reader is referred thence for more details.

4.1.3 Parameter Update

In addition to the forms of $\hat{\mathbf{J}}$ and $\hat{\mathbf{H}}$, these SMs also differ in the way they apply the estimated incremental update $\Delta\mathbf{p_t}$ to $\mathbf{p_{t-1}}$ to obtain $\mathbf{p_t}$. The two additive formulations - FALK and IALK - apply it by direct addition, i.e. $\mathbf{p_t} = \mathbf{p_{t-1}} + \Delta\mathbf{p_t}$. FCLK and ESM, on the other hand, do so by composition:

$$\mathbf{p_t} = \mathbf{p_{t-1}} \circ \Delta\mathbf{p_t} \tag{4.19}$$

where $\circ$ denotes the composition operator which is defined as $\mathbf{w}(\mathbf{x}, \mathbf{p_{t-1}} \circ \mathbf{\Delta p_t}) = \mathbf{w}(\mathbf{w}(\mathbf{x}, \Delta\mathbf{p_t}), \mathbf{p_{t-1}})$. ICLK uses composition too but it first inverts $\Delta\mathbf{p_t}$:

$$\mathbf{p_t} = \mathbf{p_{t-1}} \circ \Delta\mathbf{p_t^{-1}} \tag{4.20}$$

where the inversion operator $\mathbf{p_{t-1}^{-1}}$ is defined by the equality $\mathbf{w}(\mathbf{w}(\mathbf{x}, \mathbf{p^{-1}}), \mathbf{p}) = \mathbf{x}$.

4.2 Stochastic Search

A significant limitation of GD SMs, in common with most iterative non linear optimization methods [155], is that they are prone to getting stuck in local maxima of f, which, as mentioned before, makes such SMs more suited to local search. This leads to tracking failures when changes in the object's appearance due to factors like occlusions, motion blur and illumination variations introduce false maxima in the functional surface of f in addition to the desired global maximum. An alternative approach that avoids this problem is stochastic search where random samples for $\Delta\mathbf{p}$ are generated according to some distribution, usually Gaussian, and the optimal warp

is deduced from the fitness of these samples. Random sampling can help to cover a larger portion of the search space of $\mathbf{p}$ rather than being constrained to follow the local gradient direction. This work considers four SMs in this category which can be further divided into two subcategories on the basis of how the sample generation and fitness evaluation are accomplished.

4.2.1 Direct Sampling

SMs in this subcategory generate samples by directly drawing them from a suitable $S + A$ dimensional probability distribution and consider the fitness of each to be proportional to the similarity f of the corresponding warped patch with the template. Their performance thus depends heavily on the number and quality of stochastic samples used. While the former is limited only by the available computational resources, the latter is a bit harder to guarantee for a general SSM/AM. For methods that draw samples from a Gaussian distribution, the quality thereof is determined by the covariance matrix used and, to the best of the author's knowledge, no widely accepted method exists to estimate this in the general case. Most works in literature have either used heuristics or performed extensive hand tuning to get acceptable results, sometimes even using different values for each tested sequence [1].

Given this, a reasonable way to decompose these SMs to fit the proposed framework is to delegate the responsibility of generating the set of samples and estimating its mean entirely to the SSM (for $\mathbf{p_s}$) and AM (for $\mathbf{p_a}$) while letting the latter evaluate the suitability of each sample by providing the likelihood of the corresponding patch. Since the definition of what constitutes a good sample and how the mean of a sample set is to be evaluated depends on the SSM/AM, as do any heuristics for generating these samples (like the variance for each component of $\mathbf{p}$), such a decomposition ensures both theoretical validity and good performance in practice.

There are two SMs in this category that differ in the image - I_0 or I_t - from where the patches corresponding to their samples are extracted. These are respectively nearest neighbor search and particle filter and are described next.

4.2.1.1 Nearest Neighbor Search (NN)

NN has recently been used for RBT [98] thanks to the Fast Library for Approximate Nearest Neighbors (FLANN) [156] that makes real time search feasible; its previous applications had been limited to OLT [157]. This SM generates samples from I_0 during

tracker initialization by applying a set of random perturbations $\{\Delta\mathbf{p_i}|1 \leq i \leq K\}$ to the initial grid points $\mathbf{x_0}$, where K is the number of samples. The corresponding warped patches $\mathbf{I_0}(\mathbf{w}(\mathbf{x_0}, \Delta\mathbf{p_i}))$ are used to build an index which is then searched to find the nearest neighbor to the patch extracted from I_t at the tracker's location in I_{t-1}, i.e. $\mathbf{I_t}(\mathbf{w}(\mathbf{x_0}, \mathbf{p_{t-1}}))$. Finally, the perturbation $\Delta\mathbf{p}_{nn}$ corresponding to this nearest neighbor is *inverted* and applied to $\mathbf{p_{t-1}}$ to obtain $\mathbf{p_t}$:

$$\mathbf{p_t} = \mathbf{p_{t-1}} \circ \Delta\mathbf{p}_{nn}^{-1} \tag{4.21}$$

Building the index is a very time consuming process (Fig. A.7) and so cannot be done on line during tracking. This constrains NN to use samples only from I_0. FLANN does allow more samples to be added to the index after it has been built without having to rebuild it. However, this functionality seems a bit buggy at present and leads to frequent crashes. The index also becomes unbalanced after a few additions which necessitates its rebuilding. This possibility has therefore been excluded in this work but might be considered in a future extension once these issues are resolved.

It can be seen that Eq. 4.21 is quite similar to Eq. 4.20 and indeed NN is closely analogous to ICLK as they both find the warp that aligns $\mathbf{I_0}$ with $\mathbf{I_t}(\mathbf{x_{t-1}})$ and use its inverse to update $\mathbf{p_{t-1}}$. There is an underlying assumption in this approach that, if a warp applied to $\mathbf{I_0}$ causes it to match a slightly warped version of $\mathbf{I_t}(\mathbf{x_t})$ (which $\mathbf{I_t}(\mathbf{x_{t-1}})$ is considered to be), then the inverse of this warp can be applied to the latter for the resultant patch to match $\mathbf{I_0}$ and thus be a good guess for $\mathbf{I_t}(\mathbf{x_t})$ itself. As this assumption is only valid when the appearance of $\mathbf{I_t}(\mathbf{x_t})$ is similar to $\mathbf{I_0}$, these SMs tend to fail when the object's appearance undergoes change due to factors like occlusion, motion blur or lighting variations. This is the price these SMs pay for the speed advantage they gain by avoiding online computations - re-extracting patches in case of NN and re-computing the gradient in case of ICLK.

There are several methods for looking up the best matching sample in the index. FLANN [156] offers two main methods - randomized kd tree search (**KDT**) [158, 159, 160] and hierarchical k-means tree search [161, 162, 163] (**HKMT**). KDT constructs a set of trees by splitting the data in half at each level along one of the dimensions chosen randomly from the 5 dimensions with the greatest variance. At search time, all the trees are traversed in parallel, with a common priority queue keeping track of the closest neighbor found so far. To ensure that the search is fast, only a fixed number of leaf nodes are examined in each tree to find an approximate nearest neighbor that is good enough to satisfy the required precision. Though this method is the fastest

one in practice, it has an important limitation in only being compatible with AMs where the distance measure $(-f)$ can be considered as a valid vector space distance since it needs to perform dimension wise splitting. This requirement is only satisfied by pixel wise measures like SSD (Sec. 5.1.1) and SPSS (Sec. 5.2.5) as well as by NCC (Sec. 5.2.1) with careful formulation of the distance vector (Sec. A.2.1.1).

HKMT, on the other hand, works with all AMs as it only requires its distance measure to be a metric distance, though this flexibility does come at the cost of slower performance than KDT, both while building and searching the index. Here, the tree is constructed by recursively splitting the data points at each level into k non overlapping regions using k-means clustering. While searching, this tree is traversed by choosing the branch in each level whose center is closest to the query point. Like KDT, HKMT too examines only as many leaf nodes as needed to achieve the specified precision.

There is another more recent method that uses graph search [164] instead of tree traversal. This is here referred to as graph based nearest neighbor search (**GNN**). This is not present in FLANN but has been adapted for tracking within MTF from the original source code provided by its authors. The graph is constructed by connecting each node - corresponding to one sample - to its k nearest neighbors in the dataset, where k is the user specified graph connectivity. The search is started from a random node in the graph and follows a greedy hill climbing procedure where the search moves to the neighbor with the smallest distance from the query in each step. The search stops either after a fixed number of steps or when the distance of the query from the current node becomes smaller than that from any of its neighbors in the graph. To benefit from the temporal coherence in a tracked image sequence, which ensures high correlation between nearby frames, this method is adapted for tracking by starting each search at the node corresponding to the nearest neighbor found in the previous search instead of using a random node each time. This facilitates faster convergence since the current query point is likely to be very similar to the previous one due to the aforementioned coherence.

Though GNN compares favorably against KDT and HKMT due to this tracking specific adaptation (Sec. A.2.1.1), it has the limitation of taking significantly longer to initialize since the graph requires several of the *true* nearest neighbors to be found for each of the samples in the set which can only be guaranteed using slow brute force search. A simple approach to alleviate this issue is to use the *approximate* nearest neighbors, provided by a fast FLANN search, for building the graph too. This method

is termed FLANN based GNN (**FGNN**) and has been found (Sec. A.2.1.1) to perform similar to GNN while being faster to initialize.

As will be seen in the next section, one of the main issues with direct sampling based stochastic SMs is that there exists no general method for estimating a distribution that can generate good samples. A solution proposed there is to use multiple distributions simultaneously and dynamically select the most appropriate one by its performance in recent frames. However, this approach is not as effective with NN since its samples are generated only once so there is no way to decide the proportion of samples to draw from each distribution. In the absence of prior information about expected motion, the most reasonable approach is to draw equal number of samples from each distribution but this was not found to be a significantly improvement over just using one distribution (Fig. A.8). Another method for combining multiple distributions is to use them in cascade to carry out a coarse to fine search [98]. In this arrangement, each layer in the cascade has an NN based tracker that uses a finer distribution - one that generates less perturbed samples - than the one before it. The output of each layer is then used as the starting point for the next one so that each layer successively refines the estimate. Cascades with 3 and 5 layers were tested in this work and found to be significantly better than single layer NN (Fig. A.6).

4.2.1.2 Particle Filter (PF)

PF is a sequential Monte Carlo (SMC) sampling method [165] that has been widely used for tracking especially since the condensation algorithm of Isard & Blake [166]. It uses sequential importance sampling to estimate and propagate a conditional probability distribution for the tracker state given the images as observations. This distribution is represented by a set of weighted samples or particles where each particle is a randomly generated candidate for the tracker state - typically drawn from a Gaussian distribution - and its weight is indicative of the likelihood of this being the true state. Within the proposed framework, the tracker state is represented by $\mathbf{p}$ while the likelihood is considered to be some increasing function of the similarity f of the corresponding patch with the template. Since, PF does not require f to be differentiable, it has been the SM of choice for several AMs that incorporate online learning of the template [145, 99, 146, 147, 101, 100, 148]. It has conventionally been used for low DOF tracking [166, 167, 168, 169, 170, 171] but there have been several instances of affine [99, 100, 101, 102, 103, 104, 105, 106] and, more recently, homography [1] based trackers too.

Unlike NN, PF generates new samples in each frame I_t rather than just once in I_0. This has the disadvantage of making it significantly slower and thus restricted to using fewer samples. On the other hand, it also allows PF to generate better samples by adapting them to reflect the actual motion of the tracked object in recent frames, thus taking advantage of the temporal coherence of this motion. One approach is to use an autoregressive dynamic model [172] instead of simple random walk used by NN. Other possibilities include simultaneous estimation of optimal sampling parameters along with $\mathbf{p}$ [173, 174, 106] or using the gradient of f to improve the estimation [105, 1]. However, each of these methods has its own limitations and, to the best of the author's knowledge, no sufficiently general method has yet been developed for generating good samples for $\mathbf{p}$, especially for high DOF SSMs. As mentioned earlier, most works use extensive hand tuning with a trial-and-error approach to get their PF based trackers to perform well with a given set of test sequences, often using different parameters for each sequence to help their tracker compare more favorably against the competition. This approach does not work in real tracking applications as it is usually impossible to predict the object motion there. Further, as the optimal sampling parameters depend strongly on the actual motion of the object in the sequence, any single set of these parameters is unlikely to work for all scenarios.

A possible solution to this problem attempted in this work is to use multiple Gaussian distributions for generating the samples instead of just one and dynamically adjust the proportion of total particles drawn from each distribution according to the performance of its samples in the recent past. This is accomplished by choosing the probability of drawing a particle from a specific distribution to be proportional to the average weight of all the particles generated by that distribution in the previous n frames (only $n = 1$ has been tested here). This probability is subject to a minimum threshold so that all distributions have a non zero chance of contributing some particles to the pool. This approach allows the tracker to take advantage of several distributions - each representing a specific kind of motion - to effectively draw samples from a Gaussian mixture model [175] that can hopefully represent real motion better than any of its constituent distributions. This can reduce the need to fine tune the sampler for each sequence and also perform better in the absence of prior information about the kind of motion the tracked object will undergo. A somewhat similar approach was also employed in [176]. It was found to work quite well with both the 8 DOF samplers used in this work. (Sec. A.2.2.1) and was also slightly better in general than the coarse to fine cascade approach mentioned in the last section (Sec.

A.2.2.5).

Apart from the sampling method, there are several other variables that affect the performance too. One of these is the method used for resampling the particles to avoid the degeneracy problem [166]. Several methods have been proposed for this crucial process [177, 178, 179]. Experiments conducted in the course of this study indicated that binary multinomial resampling offers a good compromise between speed and accuracy so this was used for all experiments. Another factor that affects the performance is the approach adopted for combining the resampled particles to estimate the optimal value of $\mathbf{p}$. The conventional approach used in SMC methods is to take the mean of these particles though this is rather tricky for high DOF SSMs like homography since these often do not have a well defined mean except in specific cases [180]. A simpler alternative is to consider the state corresponding to the highest weighted particle as the optimal $\mathbf{p}$ in each frame. This relatively trivial approach was found to perform quite well (Sec. A.2.2.4) when compared with the mean estimation method for $\mathbb{SL}(3)$ parameterization of homography used by Kwon et al.[1].

Another factor that appears to have a significant impact on the performance of PF is the measurement function used for converting f to the likelihood L_f which, when normalized, becomes the particle weight. Since, it measures probability, $0 \leq L_f \leq 1$ and thus the following expression has been used in this work:

$$L_f = \exp\left(-\alpha\left(\frac{f^* + \beta}{f} - 1\right)^2\right) \tag{4.22}$$

where $f^* = f(\mathbf{I_0}, \mathbf{I_0})$ is the maximum value of f while α and β are parameters that depend on the numerical range of f. These were tuned for each AM to provide optimal performance (Sec. A.3.1).

4.2.2 Indirect Sampling

SMs in this category are also known as robust regression methods in literature [181]. They generate samples for $\mathbf{p}$ indirectly by estimating these from two sets of corresponding points (i.e. image locations) produced by tracking each point between two consecutive frames in the sequence. This estimation is a form of model fitting that tries to find the $\mathbf{p}$ that can best explain the geometric transformation of points from the first to the second set. Though any tracker can be used for generating this point correspondence, simple 2 DOF LK type trackers are used in practice [33], so that the process becomes identical to gradient based optical flow estimation. There are three

main reasons for this. Firstly, these are some of the fastest trackers available and so better suited to tracking a large number of points in real time. Secondly, these trackers are not required to be very robust as the estimation process can simultaneously detect failed trackers as outliers while estimating the best fit samples. Finally, more robust OLTs are not suited for this task as frame-to-frame tracking does not benefit from model learning. Nevertheless, feature detection and matching has been used for offline applications of image registration like mosaicing [182].

Two popular methods in this category have been considered here - random sample consensus (**RANSAC**) [183] and least median of squares (**LMS**) [50]. These methods were chosen mainly due to the availability of their efficient implementations in OpenCV [18]. In both methods, a sample is generated by first choosing a random subset of the corresponding point pairs and then finding the warp which can be applied to the first of the point pairs to minimize the distance, in the least squares sense, of the resultant warped points from the second of the pairs. For all SSMs whose transformation can be expressed by a matrix multiplication, this is usually done using the direct linear transform (DLT) algorithm [184]. The warp thus generated is then applied to all of the points in the first set and the number of inliers is counted by comparing the warped points with their correspondences in the second set. A large number of such samples are generated, with the precise count being controlled either by a user specified maximum or till a good enough sample is obtained. The minimum number of point pairs needed for each subset depends on the DOF of the SSM - 1, 2,3 and 4 respectively for translation, similitude, affine and homography. Tests were conducted with up to 4 more points than the minimum for each SSM but showed that the subset size makes no difference to tracking performance.

RANSAC and LMS differ in the way they define the best fitting sample. RANSAC considers this to be the sample that maximizes the number of inliers. An inlier is defined here as a point pair where the distance, usually measured by the Euclidean norm, of the second point in the pair from the warped location of the first point is less than a user defined threshold. LMS, on the other hand, looks for the sample that minimizes the median of this distance over all point pairs. This gives LMS the advantage of not requiring a threshold to determine inliers. However, this comes at the cost of LMS working correctly only when at least half the points are inliers while RANSAC can handle any ratio of inliers even if its estimation does get less accurate as this ratio falls.

4.3 Composite Search

As mentioned before, both GD and stochastic SMs have their own strengths and weaknesses. The former are often more precise while being more prone to failure in the presence of false maxima due to appearance changes or when the starting point of search is outside the basin of convergence due to fast motion. The latter, though less prone to getting stuck in local maxima, depend largely on the number and quality of random samples for their performance since these determine the available search space. This, combined with the limited number of samples that can be used while maintaining real time speeds as well as the difficulty of generating good samples, causes these SMs to produce comparatively jittery and unstable results. A simple approach to simultaneously obtain better precision and robustness is to combine the best of both types of SMs by running a stochastic SM in *cascade* with a GD SM [98, 33] such that results from the former are used as starting search points for the latter. Results from the latter are in turn fed back to the former to provide it with a more precise starting location for the next frame. Three such combinations have been considered in this work as examples of hybrid or composite SMs - NN+ICLK (**NNIC**), PF+FCLK (**PFFC**) and LMS+ESM (**LMES**).

In addition to providing better starting points for the two layers, this cascade arrangement can also provide a useful cue to automatically detect tracking failure. This is based on the observation that when the composite tracker works correctly, the output of the GD method does not vary too much from that of the stochastic one since GD methods only converge correctly within a relatively narrow basin of convergence. This is why the GD step can be considered as performing a further refinement of the already approximately correct location found by the stochastic step. Therefore, a significant difference in the locations found by the two SMs - if the alignment error (Sec. 8.2) between them exceeds a threshold for instance - almost always indicates failure of the composite SM. The tracker can take advantage of this in two ways. Firstly, it can assume that the stochastic SM is more robust and the failure was due to the GD step, thus rejecting the output of the latter and considering that of the former as the correct one for this frame. This method was used in [33]. Secondly, it can maintain a buffer of last n frames along with the corresponding object locations so as to reinitialize itself n frames before the one where failure is detected, under the assumption that it was still working then. This way, if the tracker failed due to a change in the object's appearance, the updated template might help it to avoid failing

again after the reinitialization.

Finally, indirect sampling methods have yet another advantage in this context provided that the resolution of the grid whose intersection points form the corresponding point pairs (used as input to the robust estimation step) is same as the sampling resolution used by the GD method. In this case, a one-to-one correspondence exists between the point pairs and the pixel locations where the latter samples the patch. The optimal warp estimated by the robust method can then be used to generate a mask of inliers corresponding to this warp and passed to the GD method so the latter can use it to perform selective pixel integration (SPI) [153] by considering pixel values at only those sampled points that correspond to inliers. This is based on the assumption that outliers represent failures of the optical flow estimation process and the corresponding pixel values probably do not match the template well. Thus, rejecting these pixels while computing f and its derivatives in the GD step might provide better convergence [33].

4.4 Summary

This chapter described the various SMs considered in this thesis. These were divided into three main categories. The first category of methods was based on gradient descent search and included the four variants of the LK tracker - FCLK, ICLK, FALK and IALK - in addition to the newer ESM method. The second category was of stochastic search methods and was further divided into to sub categories based on the sampling approach adopted. There were two SMs in each subcategory - methods that perform direct sampling included NN and PF while those with indirect sampling comprised RANSAC and LMS. Finally, several composite SMs were proposed to combine both GD and stochastic methods in cascade and get the best of both worlds.

Chapter 5

Appearance Models

AM is the image similarity measure defined by f (Eq. 3.1) that the SM uses to determine if a candidate patch $\mathbf{I^c}$ is a good match for the template $\mathbf{I}^*$. These two patches are typically extracted from I_t and I_0 respectively, though the reverse is also possible as in NN (Sec. 4.2.1.1) and ICLK (Sec. 4.1). Also, the template $\mathbf{I}^*$ need not be fixed during tracking but can incorporate online learning [145, 146, 146] to adapt to more recent object appearance. Further, $\mathbf{I}^*$ is not constrained by definition to represent a single image patch in the conventional sense. It can be any suitable representation for the object including, for instance, the eigenbasis constructed using several tracked patches between $\mathbf{I_0}$ and $\mathbf{I_t}$ [99, 185] or statistical models of its shape and appearance [186, 187]. Finally, f can be parameterized in any arbitrary manner, in which case the SM optimizes its parameters ($\mathbf{p_a}$) too along with those of the SSM.

In order to work well in practice, f must bear a strong positive correlation with the probability of the two image patches being compared belonging to the same object, i.e. it must increase when this probability is higher and vice versa. Due to the large variety of changes that the image patch corresponding to an object can undergo in real world sequences, there is still no single AM that can work well under all scenarios. Several attempts have, however, been made to deal with specific challenges like illumination changes [84], motion blur [87], shadows, reflections [5] and occlusions [101].

Though the definition of AM is flexible enough to incorporate online learning and non-pixel based models of object appearance [99, 185, 186], such AMs are not compatible with all SMs considered in this study, mainly as not being differentiable, and so have not been included here. In fact, two such AMs - called Principal Components Analysis (PCA) and Deep Feature Maps (DFM) - have already been implemented within MTF at the time of this writing. The former uses online learning by con-

structing an eigenbasis from several tracked patches [99] while the latter employs offline learning by using a pre-trained deep convolutional neural network [188] to apply a feature transform to both $\mathbf{I_t}$ and $\mathbf{I_0}$. Both AMs are excluded, however, for the reason mentioned above. The scope of this work is restricted to AMs and SMs that can all be combined with each other and more specialized models have been deferred to future extensions. Another AM excluded here for a slightly different reason is Kullback-Leibler divergence (KLD). Though theoretically compatible with all SMs and SSMs, this was not found to perform well with higher DOF SSMs, even when computed over several corresponding sub patches to compensate for the lack of spatial information [49]. Finally, as mentioned in Sec. 3.1, the exploration of parameterization of f is restricted to ILMs [139, 84], where $\mathbf{I^c}$ is replaced with $g(\mathbf{I^c}, \mathbf{p_a})$ to account for differences in illumination between I_0 and I_t.

The various non-parameterized AMs included in this study are described next along with expressions for $\partial f/\partial \mathbf{I}$ and $\partial^2 f/\partial \mathbf{I}^2$ that are needed for Eqs. 4.2 and 4.3. Detailed derivations for these expressions are only provided for AMs that, to the best of the author's knowledge, have not been used with GD based SMs elsewhere in literature. The AMs are divided into two categories - those where f can be expressed as the squared ℓ^2 norm of the element wise difference between two vectors and those where this is not true. The former are termed as L2 models and the latter as robust models after [4]. Parameterization of AMs, as represented by ILMs, is described in a separate section that follows that of these aforementioned AMs.

In each of the following sections, **function plots** for the respective AMs are also shown to visually demonstrate their characteristics like radius of convergence and degree of illumination invariance. These were generated by applying horizontal and vertical translations in the range of $[-20, 20]$ pixels to the ground truth locations of specific frames in six sequences and evaluating f by using the translated patch as $\mathbf{I_t}$. The template $\mathbf{I_0}$ was extracted from the first frame in each sequence. Fig. 5.1 shows the frames that have been used. These were chosen from the four datasets (Sec. 8.1) to incorporate common tracking challenges like motion blur, occlusion and lighting changes. These plots have not been analyzed here since such a small sample is not enough to draw meaningful conclusions based on the shapes of these plots alone. These have instead been used as supporting evidence for the analysis in chapter 9 and the two appendices.

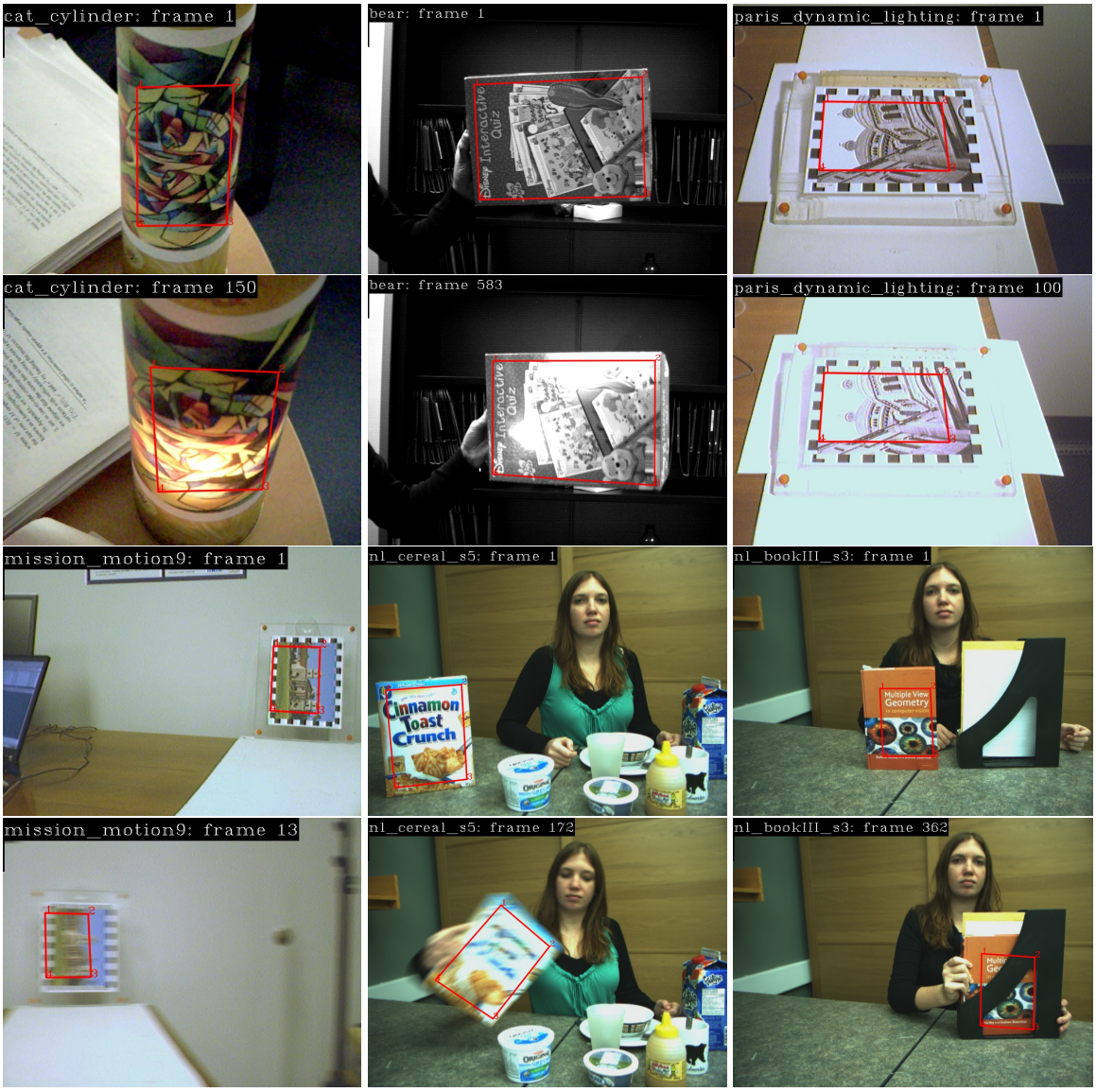


Figure 5.1: Frames used for generating the function plots for AMs. First and fourth rows show I_0 where the template is extracted while the second and fourth rows show I_t. Challenge present here - localized illumination change (`cat_cylinder`, `bear`), global illumination change (`paris_dynamic_lighting`), motion blur(`mission_motion9`, `nl_cereal_s5`) and occlusion (`nl_bookIII_s3`)

5.1 L2 models

This category includes AMs that can be defined according to the following expression:

$$f_{L2} = -\frac{1}{2}\|\hat{\mathbf{I}}_\mathbf{t} - \hat{\mathbf{I}}_\mathbf{0}\|^2 \tag{5.1}$$

where $\hat{\mathbf{I}}_\mathbf{t}, \hat{\mathbf{I}}_\mathbf{0} \in \mathbb{R}^N$ are derived from $\mathbf{I_0}$ and $\mathbf{I_t}$. The minus sign is added to convert the distance measure to one for similarity (which f is defined to be) and the multiplicative $\frac{1}{2}$ to avoid multiplication by 2 in the derivatives of f_{L2}.

The first and second order derivatives of f_{L2} w.r.t. $\hat{\mathbf{I}}_\mathbf{t}$, are given as:

$$\frac{\partial f_{L2}}{\partial \hat{\mathbf{I}}_\mathbf{t}} = \hat{\mathbf{I}}_\mathbf{0} - \hat{\mathbf{I}}_\mathbf{t} \tag{5.2}$$

$$\frac{\partial^2 f_{L2}}{\partial \hat{\mathbf{I}}_\mathbf{t}^2} = -\mathbb{I}_N \tag{5.3}$$

where $\mathbb{I}_N$ denotes the $N \times N$ identity matrix. To be accurate, the corresponding derivatives of f_{L2} w.r.t. $\mathbf{I_t}$ should employ chain rule as $\frac{\partial f_{L2}}{\partial \mathbf{I_t}} = \frac{\partial f_{L2}}{\partial \hat{\mathbf{I}}_\mathbf{t}} \frac{\partial \hat{\mathbf{I}}_\mathbf{t}}{\partial \mathbf{I_t}}$ and $\frac{\partial^2 f_{L2}}{\partial \mathbf{I_t}^2} = \frac{\partial \hat{\mathbf{I}}_\mathbf{t}}{\partial \mathbf{I_t}}^T \frac{\partial^2 f_{L2}}{\partial \hat{\mathbf{I}}_\mathbf{t}^2} \frac{\partial \hat{\mathbf{I}}_\mathbf{t}}{\partial \mathbf{I_t}} + \frac{\partial f_{L2}}{\partial \hat{\mathbf{I}}_\mathbf{t}} \frac{\partial^2 \hat{\mathbf{I}}_\mathbf{t}}{\partial \mathbf{I_t}^2}$. However, in practice, these are often approximated using Eqs. 5.2 and 5.3 respectively either because the relation between $\hat{\mathbf{I}}_\mathbf{t}$ and $\mathbf{I_t}$ is not differentiable (Sec. 5.1.2 - 5.1.4) or the derivatives are too costly to compute in real time compared to the resultant gain in accuracy (Sec. 5.1.5).

5.1.1 Sum of Squared Differences (SSD)

SSD is the simplest and most commonly used AM in literature [16, 22, 137, 25, 76] where $\hat{\mathbf{I}}_\mathbf{0} = \mathbf{I_0}$ and $\hat{\mathbf{I}}_\mathbf{t} = \mathbf{I_t}$ so that Eqs. 5.2 and 5.3 provide accurate expressions for $\frac{\partial f_{ssd}}{\partial \mathbf{I_t}}$ and $\frac{\partial^2 f_{ssd}}{\partial \mathbf{I_t}^2}$. SSD is especially popular with SMs based on GD due to its relatively wide radius of convergence (Fig. 5.2) and the ease of computing its derivatives. However, its simplicity also makes it vulnerable to providing false matches when the object's appearance undergoes changes. This has led to the development of several improved AMs, some of which follow here.

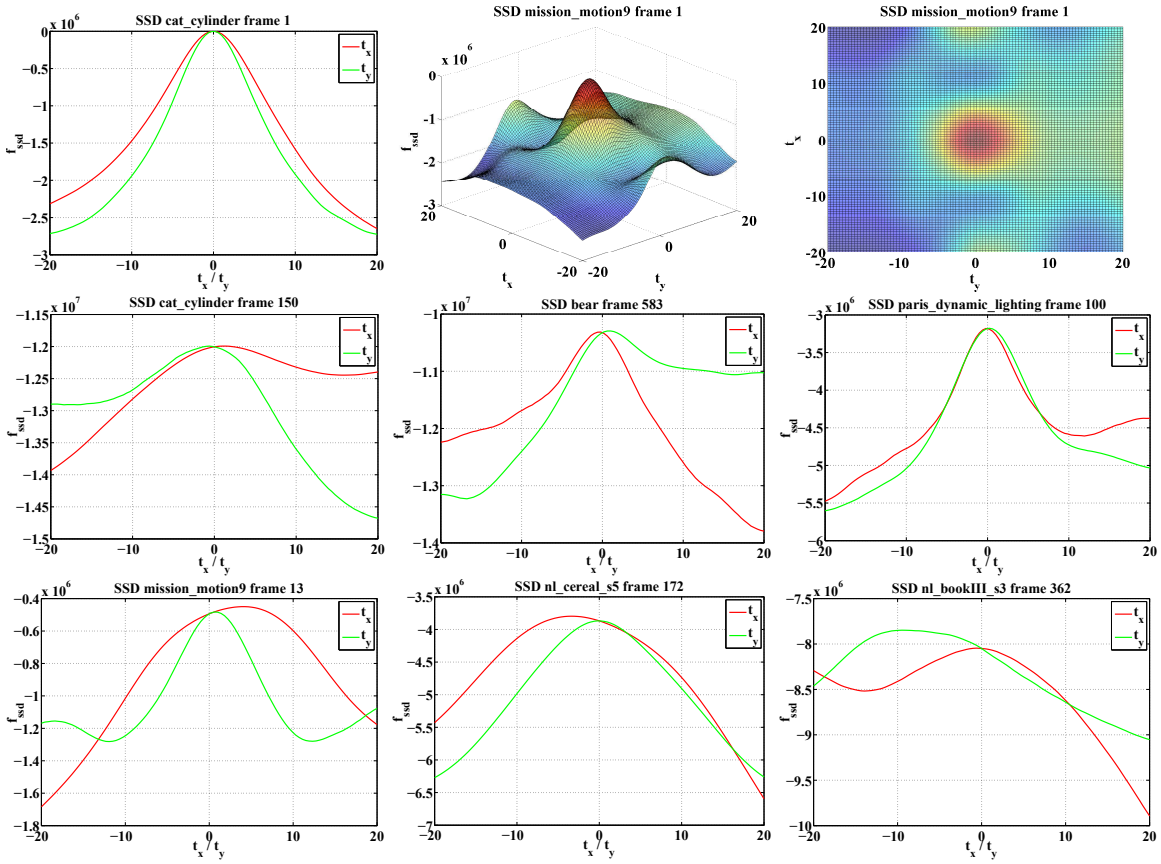


Figure 5.2: f_{ssd} plotted against x and y translation.

5.1.2 Sum of Conditional Variance (SCV)

SCV was originally introduced for multi modal medical image registration [189] but was adapted for RBT by Richa et al. [95] as an improvement to SSD for better handling of illumination changes without using ILMs. It has since been used for visual servoing [97] and non rigid object tracking [96] too. SCV uses $\hat{\mathbf{I}}_\mathbf{t} = \mathbf{I}_\mathbf{t}$ and computes $\hat{\mathbf{I}}_\mathbf{0}$ using the following expression:

$$\hat{\mathbf{I}}_\mathbf{0} = \mathbb{E}(\mathbf{I}_\mathbf{t}|\mathbf{I}_\mathbf{0}) \tag{5.4}$$

where $\mathbb{E}$ is the expectation operator which is defined as:

$$\hat{\mathbf{I}}_\mathbf{0}(\mathbf{x}_{k0}) = \mathbb{E}(I_{kt}|I_{k0} = j) = \frac{\sum_i^{L-1} i \cdot P_{t0}(i, I_{k0})}{\sum_i^{L-1} P_{t0}(i, I_{k0})} \tag{5.5}$$

$\forall\ 1 \leq k \leq N$. Here, L denotes the number of gray levels in the images (typically $L = 256$) and $j \in [0, L-1]$. It should be recalled from Sec. 3.1 that $I_{kt} = \mathbf{I}_\mathbf{t}(\hat{\mathbf{x}}_{kt}) =$

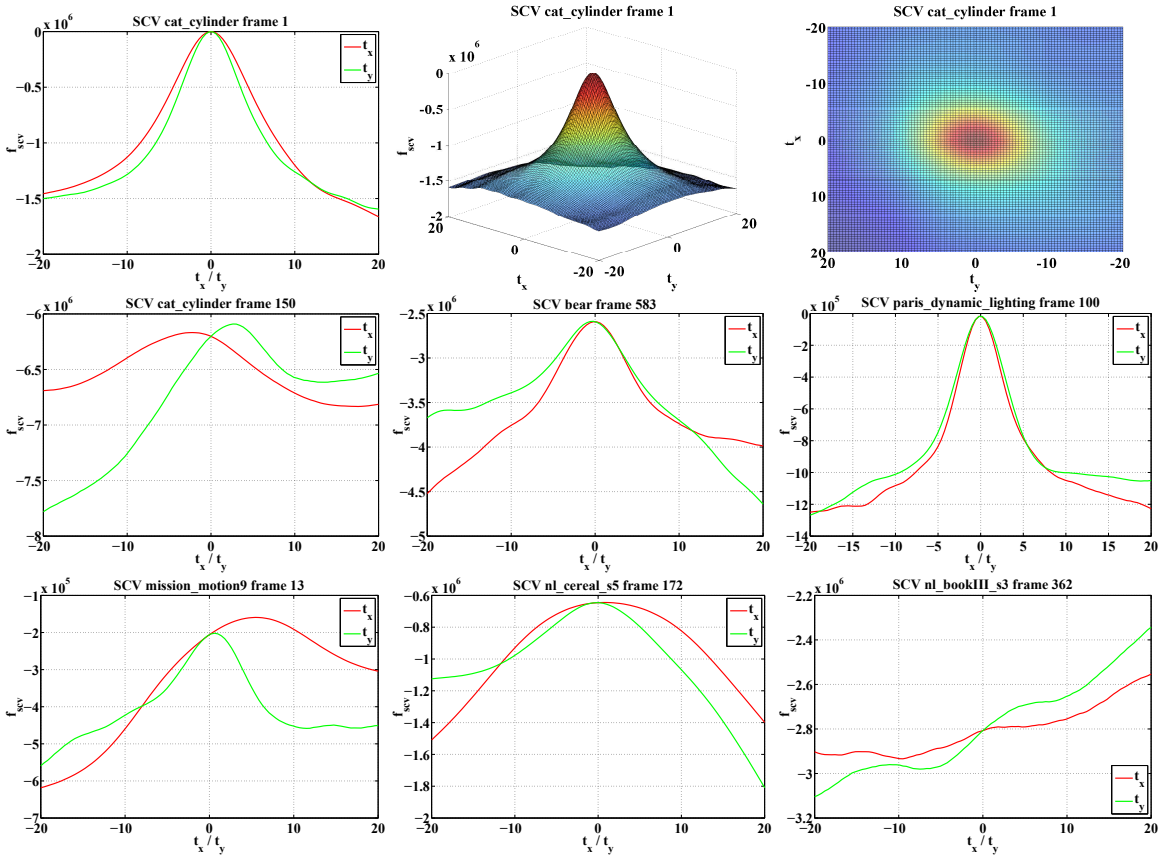


Figure 5.3: f_{scv} plotted against x and y translation.

$I_t(\hat{x}_{kt}, \hat{y}_{kt})$ and $I_{k0} = \mathbf{I_0}(\mathbf{x}_{k0}) = I_0(x_{k0}, y_{k0})$ are the k^{th} pixel values in $\mathbf{I_t}$ and $\mathbf{I_0}$ respectively. Further, $P \in \mathbb{R}^{L \times L}$ is the joint intensity distribution between $\mathbf{I_t}$ and $\mathbf{I_0}$, each of whose entries $P_{t0}(i, j)$ is the probability of co-occurrence of $\mathbf{I_t} = i$ and $\mathbf{I_0} = j$. This is approximated using the normalized joint histogram which is computed as:

$$P_{t0}(i, j) = \frac{1}{N} \sum_{k=1}^{N} \psi(i - I_{kt}) \psi(j - I_{k0}) \tag{5.6}$$

where ψ is the histogram window or membership function which determines the contribution of each pixel to each bin in the histogram. In its simplest form, ψ is the Kronecker delta function given by

$$\psi_{kronecker}(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

In order to allow for non integral values in $\mathbf{I_0}/\mathbf{I_t}$, a slightly modified version of $\psi_{kronecker}$ is used here where each pixel contributes only to the bin that is *nearest*

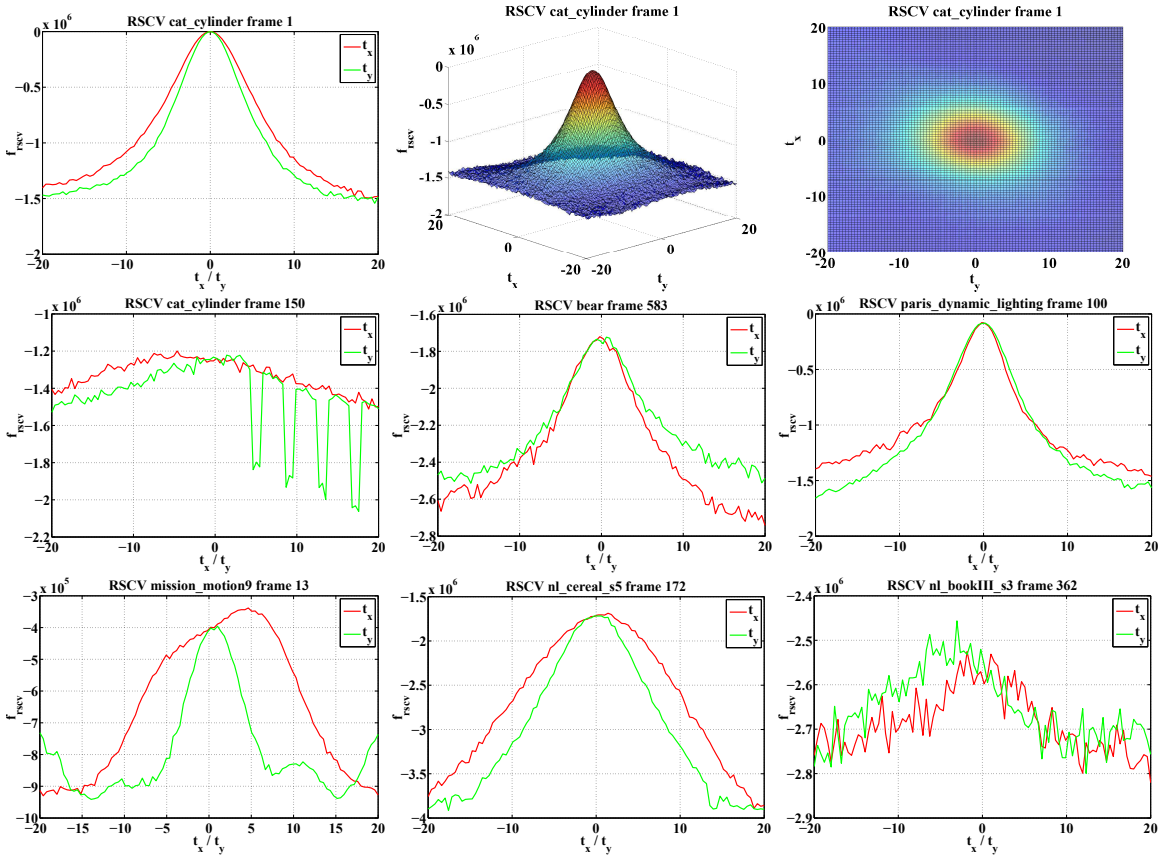


Figure 5.4: f_{rscv} plotted against x and y translation.

to its value:

$$\psi_{nearest}(s) = \begin{cases} 1 & \text{if } -0.5 < s \leq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

With this formulation, ψ, as a piecewise function, is not differentiable w.r.t. pixel values in $\mathbf{I_0}$ and $\mathbf{I_t}$ and therefore nor are P_{t0} and $\mathbb{E}$. Though ψ can be made differentiable by using Parzen density estimation with ψ as a smooth B-Spline function [190, 191, 92], but differentiating such functions is too computationally expensive to be applied here. SCV thus uses the approximate expressions in Eqs. 5.2 and 5.3 for computing $\frac{\partial f_{scv}}{\partial \mathbf{I_t}}$ and $\frac{\partial^2 f_{scv}}{\partial {\mathbf{I_t}}^2}$ respectively.

5.1.3 Reversed Sum of Conditional Variance (RSCV)

This was introduced by Dick et al. [98] as an adaptation for SCV for its efficient use with inverse SMs like NN whre candidate patches are extracted from I_0 rather than

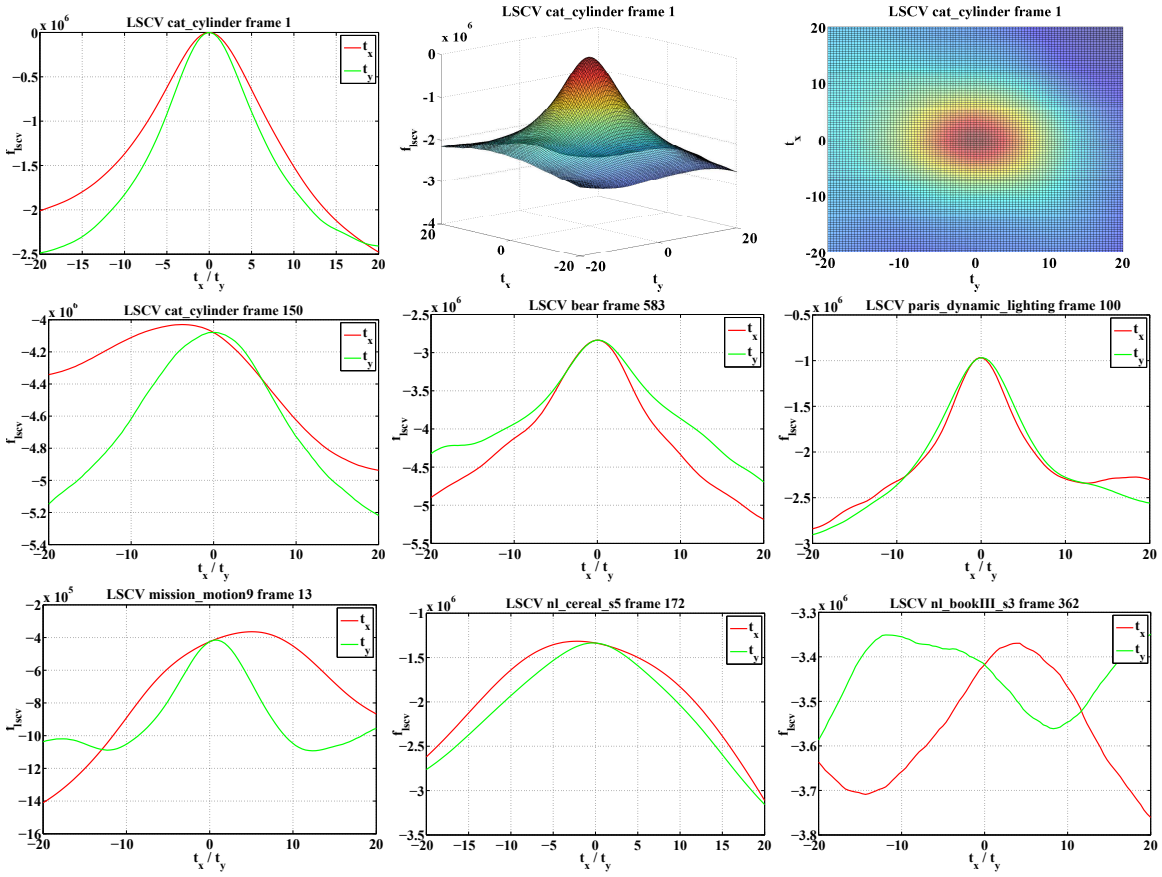


Figure 5.5: f_{lscv} plotted against x and y translation.

I_t. It only differs from SCV in using $\hat{\mathbf{I}}_\mathbf{0} = \mathbf{I_0}$ and computing $\hat{\mathbf{I}}_\mathbf{t}$ as:

$$\hat{\mathbf{I}}_\mathbf{t} = \mathbb{E}(\mathbf{I_0}|\mathbf{I_t}) \tag{5.9}$$

with

$$\mathbb{E}(I_{k0}|I_{kt} = i) = \frac{\sum_j^{L-1} j \cdot P_{t0}(I_{kt}, j)}{\sum_j^{L-1} P_{t0}(I_{kt}, j)} \tag{5.10}$$

The derivatives $\frac{\partial f_{rscv}}{\partial \mathbf{I_t}}$ and $\frac{\partial^2 f_{rscv}}{\partial \mathbf{I_t}^2}$ are computed using Eqs. 5.2 and 5.3 as for SCV.

5.1.4 Localized Sum of Conditional Variance (LSCV)

This was introduced in [5] as an extension to SCV for improved handling of localized illumination variations including reflections. LSCV creates K joint histograms $P_{t0(u,v)}$ from corresponding sub regions in $\mathbf{I_0}$ and $\mathbf{I_t}$. Here, $u \in [1, K_u], v \in [1, K_v]$ with K_u

and K_v being the number of possibly overlapping horizontal and vertical sub regions so that $K = K_u \times K_v$. Each of these histograms is then used to generate a different mapped template $\hat{\mathbf{I}}_{\mathbf{0(u,v)}}$ using Eqs. 5.4 and 5.5. Each pixel in the final mapped template $\hat{\mathbf{I}}_{\mathbf{0}}$ is computed as a weighted average of the corresponding pixels in these templates such that the weight is inversely proportional to the distance of that point from the center of the sub region, that is,

$$\hat{\mathbf{I}}_{\mathbf{0}}(\mathbf{x}_{k0}) = \sum_{1}^{K_u}\sum_{1}^{K_v} \frac{\hat{\mathbf{I}}_{\mathbf{0(u,v)}}(\mathbf{x}_{k0})}{\|\mathbf{x}_{k0} - \mathbf{c}_{(u,v)}\|} \tag{5.11}$$

where $\mathbf{c}_{(u,v)}$ is the center of the subregion (u, v).

Another difference in LSCV compared to SCV is that it uses *global affine* mapping (Eqs. 5.12, 5.13), instead of the pixel wise mapping in Eq. 5.4, in order to add constraints to this process to avoid the permutation invariance property of the latter [5] that reduces the radius of convergence:

$$\hat{\mathbf{I}}_{\mathbf{0}(u,v)} = a^*\mathbf{I_t} + b^* \tag{5.12}$$

with a^*, b^* estimated using least squares optimization as

$$a^*, b^* = \underset{a,b}{\operatorname{argmin}} \sum_{k=1}^{N} \|P_{t0(u,v)}(i, I_{kt}) - (aI_{kt} + b)\| \tag{5.13}$$

A reversed version of LSCV, analogous to RSCV, is also present in MTF but is not included in this study as it does not add anything useful to the analysis.

5.1.5 Zero mean Normalized Cross Correlation (ZNCC)

This AM is inspired from the work of Ruthotto [192] which showed that maximizing the NCC (Sec. 5.2.1) between $\mathbf{I_0}$ and $\mathbf{I_t}$ is equivalent to minimizing the SSD between the z-score [193] normalized versions of these patches. Each of these normalized patches is computed by subtracting the mean of the respective patch from the original pixel values in the patch followed by dividing its standard deviation. Therefore, $\hat{\mathbf{I}}_{\mathbf{0}}$ and $\hat{\mathbf{I}}_{\mathbf{t}}$ are given as:

$$\hat{\mathbf{I}}_{\mathbf{0}} = \frac{\mathbf{I_0} - \mu_0}{\sigma_0} \tag{5.14}$$

$$\hat{\mathbf{I}}_{\mathbf{t}} = \frac{\mathbf{I_t} - \mu_t}{\sigma_t} \tag{5.15}$$

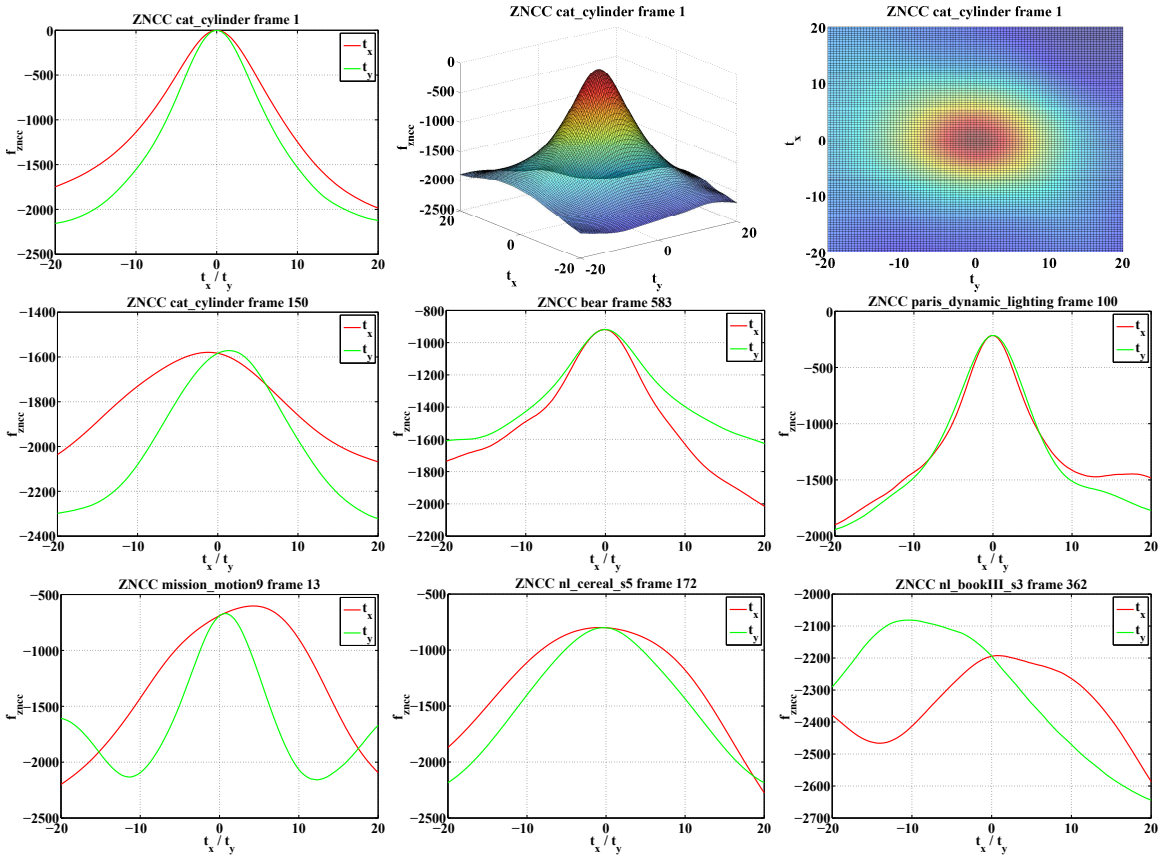


Figure 5.6: f_{zncc} plotted against x and y translation.

where μ_t and σ_t are the mean and standard deviation of $\mathbf{I_t}$ given as:

$$\mu_t = \frac{1}{N}\sum_{k=1}^{N} I_{kt} \tag{5.16}$$

$$\sigma_t = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(I_{kt} - \mu_t)^2} \tag{5.17}$$

and likewise for $\mathbf{I_0}$. Similar to the previous L2 AMs, this one too uses Eqs. 5.2 and 5.3 as approximations to $\dfrac{\partial f_{zncc}}{\partial \mathbf{I_t}}$ and $\dfrac{\partial^2 f_{zncc}}{\partial \mathbf{I_t}^2}$ respectively.

5.2 Robust Models

This category includes all AMs that do not fit into the previous one. Most of these were introduced [4] as more robust alternatives to SSD for dealing with appearance

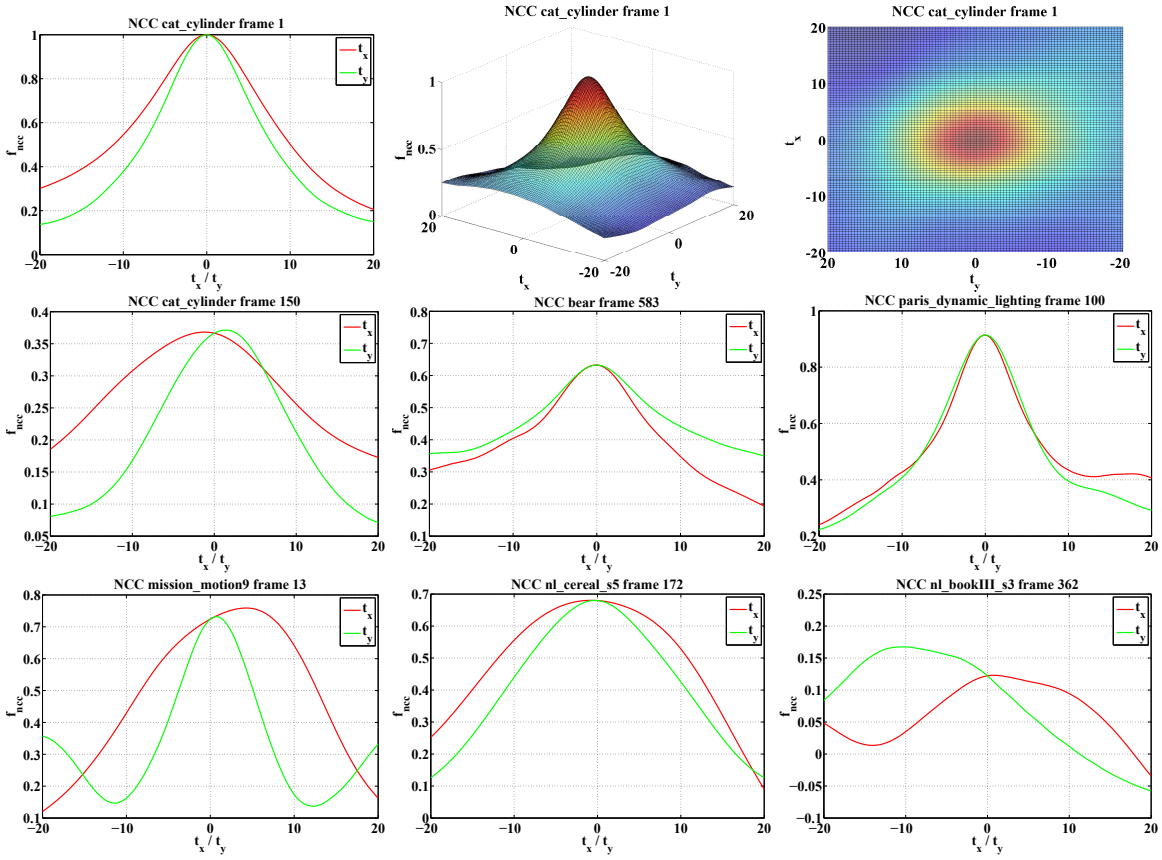


Figure 5.7: f_{ncc} plotted against x and y translation.

variations and hence are so named.

5.2.1 Normalized Cross Correlation (NCC)

NCC is another classical image similarity measure like SSD that has been extensively used for image matching [194, 195, 196] and registration [197, 192] as well as for visual tracking [198, 199, 200]. It has only been applied for RBT relatively recently though [3, 4, 1]. The main advantage of NCC over SSD is that it is invariant to affine illumination changes. Its formulation is similar to ZNCC except that it maximizes the inner product between the two normalized patches rather than minimizing the squared difference between them. It is thus defined as:

$$f_{ncc} = \frac{\mathbf{I_0} - \mu_0}{\sigma_0} \cdot \frac{\mathbf{I_t} - \mu_t}{\sigma_t} = \frac{\bar{\mathbf{I}}_\mathbf{0}}{\|\bar{\mathbf{I}}_\mathbf{0}\|} \cdot \frac{\bar{\mathbf{I}}_\mathbf{t}}{\|\bar{\mathbf{I}}_\mathbf{t}\|} \tag{5.18}$$

where μ_t and σ_t are the mean and standard deviation of $\mathbf{I_t}$, as given in Eqs. 5.14 and 5.15, while $\bar{\mathbf{I}}_\mathbf{t} = \mathbf{I_t} - \mu_t$ denotes the mean centered version of $\mathbf{I_t}$. The Jacobian and

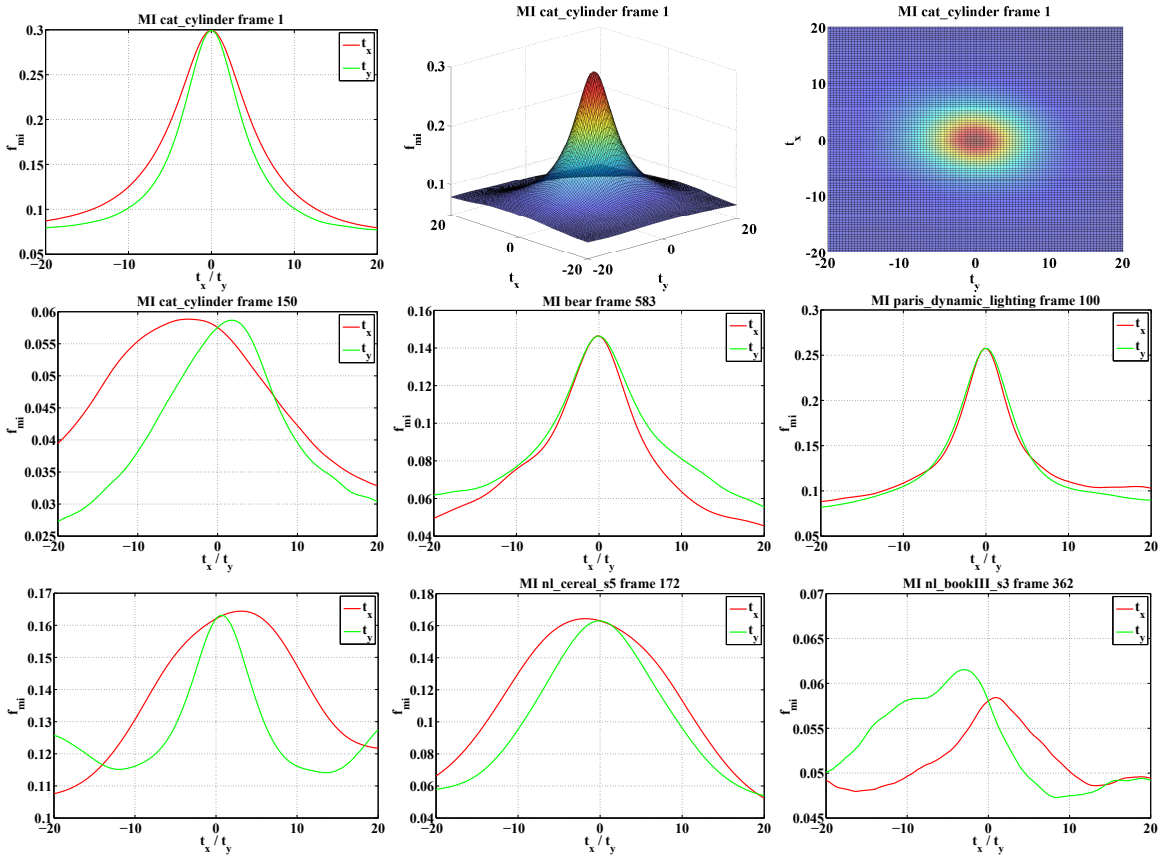


Figure 5.8: f_{mi} plotted against x and y translation.

Hessian for NCC are given as [3]:

$$\frac{\partial f_{ncc}}{\partial \mathbf{I_t}} = \frac{1}{\|\bar{\mathbf{I}}_\mathbf{t}\|}\left(\frac{\bar{\mathbf{I}}_\mathbf{0}}{\|\bar{\mathbf{I}}_\mathbf{0}\|} - f_{ncc}\frac{\bar{\mathbf{I}}_\mathbf{t}}{\|\bar{\mathbf{I}}_\mathbf{t}\|}\right) \tag{5.19}$$

$$\frac{\partial^2 f_{ncc}}{\partial {\mathbf{I_t}}^2} = -\frac{1}{\|\bar{\mathbf{I}}_\mathbf{t}\|^2}\left(f_{ncc} + \frac{\bar{\mathbf{I}}_\mathbf{0}\bar{\mathbf{I}}_\mathbf{t}^T}{\|\bar{\mathbf{I}}_\mathbf{0}\|\|\bar{\mathbf{I}}_\mathbf{t}\|} + \frac{\bar{\mathbf{I}}_\mathbf{t}\bar{\mathbf{I}}_\mathbf{0}^T}{\|\bar{\mathbf{I}}_\mathbf{t}\|\|\bar{\mathbf{I}}_\mathbf{0}\|} - 3\frac{\bar{\mathbf{I}}_\mathbf{t}\bar{\mathbf{I}}_\mathbf{t}^T}{\|\bar{\mathbf{I}}_\mathbf{t}\|^2}\right) \tag{5.20}$$

For $\hat{\mathbf{H}}_{self}$ (Eqs. 4.13, 4.12), the expression in Eq. 5.20 simplifies to:

$$\frac{\partial^2 f_{ncc}(\mathbf{I_t}, \mathbf{I_t})}{\partial {\mathbf{I_t}}^2} = \frac{1}{\|\bar{\mathbf{I}}_\mathbf{t}\|^2}\left(\frac{\bar{\mathbf{I}}_\mathbf{t}\bar{\mathbf{I}}_\mathbf{t}^T}{\|\bar{\mathbf{I}}_\mathbf{t}\|^2} - 1\right) \tag{5.21}$$

5.2.2 Mutual Information (MI)

MI is an information theoretic similarity metric that has been widely used for medical image registration [190, 191, 201, 202], especially when multi modal images are

involved [203, 204], since it is invariant to changes of modality. Inspired by its success in this context, it has also been applied for RBT [90, 92, 4], mainly using ICLK SM due to its computationally expensive derivatives, though formulations for other GD SMs have been introduced too [93]. MI is defined as:

$$f_{mi} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} P_{t0}(i,j)\log\left(\frac{P_{t0}(i,j)}{P_t(i)P_0(j)}\right) \tag{5.22}$$

where P_{t0} is the joint histogram of $\mathbf{I_t}$ and $\mathbf{I_0}$, defined as in Eq. 5.6, and L is the number of bins. P_t and P_0 are the respective marginal histograms that are given as:

$$P_t(i) = \sum_{j=0}^{L-1} P_{t0}(i,j) \tag{5.23}$$

$$P_0(j) = \sum_{i=0}^{L-1} P_{t0}(i,j) \tag{5.24}$$

for $i, j \in [0, L-1]$. A significant difference in the computation of P_{t0} for MI, compared to SCV and its variants, is that ψ here is a smooth cubic B-Spline function (Eq. 5.25) so that f_{mi} is differentiable w.r.t. $\mathbf{I_t}$ and $\mathbf{I_0}$.

$$\psi_{bspline}(s) = \begin{cases} \frac{1}{6}(2+s)^3 & \text{if } -2 \le s \le -1 \\ \frac{1}{6}(4-6s^2-3s^3) & \text{if } -1 < s \le 0 \\ \frac{1}{6}(4-6s^2+3s^3) & \text{if } 0 < s \le 1 \\ \frac{1}{6}(2-s)^3 & \text{if } 1 < s \le 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

Through a process known as Parzen density estimation [205], this function is used to approximate a Gaussian distribution so the contribution of each pixel can be smoothly distributed to four neighboring bins. Another difference here is that L is typically much smaller than 256 and the pixel values in the patches are scaled accordingly. $L = 8$ is most commonly used though results presented later (Sec. A.3.2) will show that higher values up to 24 can perform better with specific SMs.

The Jacobian of MI is given as [93]:

$$\frac{\partial f_{mi}}{\partial \mathbf{I_t}} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} \frac{\partial P_{t0}(i,j)}{\partial \mathbf{I_t}}\left(1 + \log\frac{P_{t0}(i,j)}{P_t(i)}\right) \tag{5.26}$$

where $\frac{\partial P_{t0}(i,j)}{\partial \mathbf{I_t}}$ is the $1 \times N$ derivative of $P_{t0}(i,j)$ w.r.t. $\mathbf{I_t}$ whose k^{th} entry is given as:

$$\frac{\partial P_{t0}(i,j)}{\partial I_{kt}} = \frac{1}{N} \frac{\partial \psi(i - I_{kt})}{\partial I_{kt}} \psi(j - I_{k0}) \tag{5.27}$$

with:

$$\frac{\partial \psi(s)}{\partial s} = \frac{\partial \psi_{bspline}(s)}{\partial s} = \begin{cases} \frac{1}{2}(2+s)^2 & \text{if } -2 \leq s \leq -1 \\ \frac{1}{6}(4 - 12s - 9s^2) & \text{if } -1 < s \leq 0 \\ \frac{1}{6}(4 - 12s + 9s^2) & \text{if } 0 < s \leq 1 \\ \frac{1}{2}(2-s)^2 & \text{if } 1 < s \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.28}$$

MI Hessian is given as [93]:

$$\begin{aligned} \frac{\partial^2 f_{mi}}{\partial {\mathbf{I_t}}^2} = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \Bigg[& \frac{\partial P_{t0}(i,j)}{\partial \mathbf{I_t}}^T \frac{\partial P_{t0}(i,j)}{\partial \mathbf{I_t}} \left(\frac{1}{P_{t0}(i,j)} - \frac{1}{P_t(i)} \right) \\ & + \frac{\partial^2 P_{t0}(i,j)}{\partial {\mathbf{I_t}}^2} \left(1 + \log \frac{P_{t0}(i,j)}{P_t(i)} \right) \Bigg] \end{aligned} \tag{5.29}$$

where the $N \times N$ Hessian $\frac{\partial^2 P_{t0}(i,j)}{\partial {\mathbf{I_t}}^2}$ is a diagonal matrix whose k^{th} diagonal entry is given as:

$$\frac{\partial^2 P_{t0}(i,j)}{\partial I_{kt}^2} = \frac{1}{N} \frac{\partial^2 \psi(i - I_{kt})}{\partial I_{kt}^2} \psi(j - I_{k0}) \tag{5.30}$$

with:

$$\frac{\partial^2 \psi(s)}{\partial s^2} = \frac{\partial^2 \psi_{bspline}(s)}{\partial s^2} = \begin{cases} 2 + s & \text{if } -2 \leq s \leq -1 \\ -2 - 3s & \text{if } -1 < s \leq 0 \\ 3s - 2 & \text{if } 0 < s \leq 1 \\ 2 - s & \text{if } 1 < s \leq 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.31}$$

The expression for $\frac{\partial^2 f_{mi}(\mathbf{I_t}, \mathbf{I_t})}{\partial {\mathbf{I_t}}^2}$ needed for $\hat{\mathbf{H}}_{self}$ is identical to Eq. 5.29 except that the various histograms and their derivatives are computed by setting $\mathbf{I_0} = \mathbf{I_t}$.

$$\begin{aligned} \frac{\partial^2 f_{mi}(\mathbf{I_t}, \mathbf{I_t})}{\partial {\mathbf{I_t}}^2} = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \Bigg[& \frac{\partial P_{tt}(i,j)}{\partial \mathbf{I_t}}^T \frac{\partial P_{tt}(i,j)}{\partial \mathbf{I_t}} \left(\frac{1}{P_{tt}(i,j)} - \frac{1}{P_t(i)} \right) \\ & + \frac{\partial^2 P_{tt}(i,j)}{\partial {\mathbf{I_t}}^2} \left(1 + \log \frac{P_{t0}(i,j)}{P_t(i)} \right) \Bigg] \end{aligned} \tag{5.32}$$

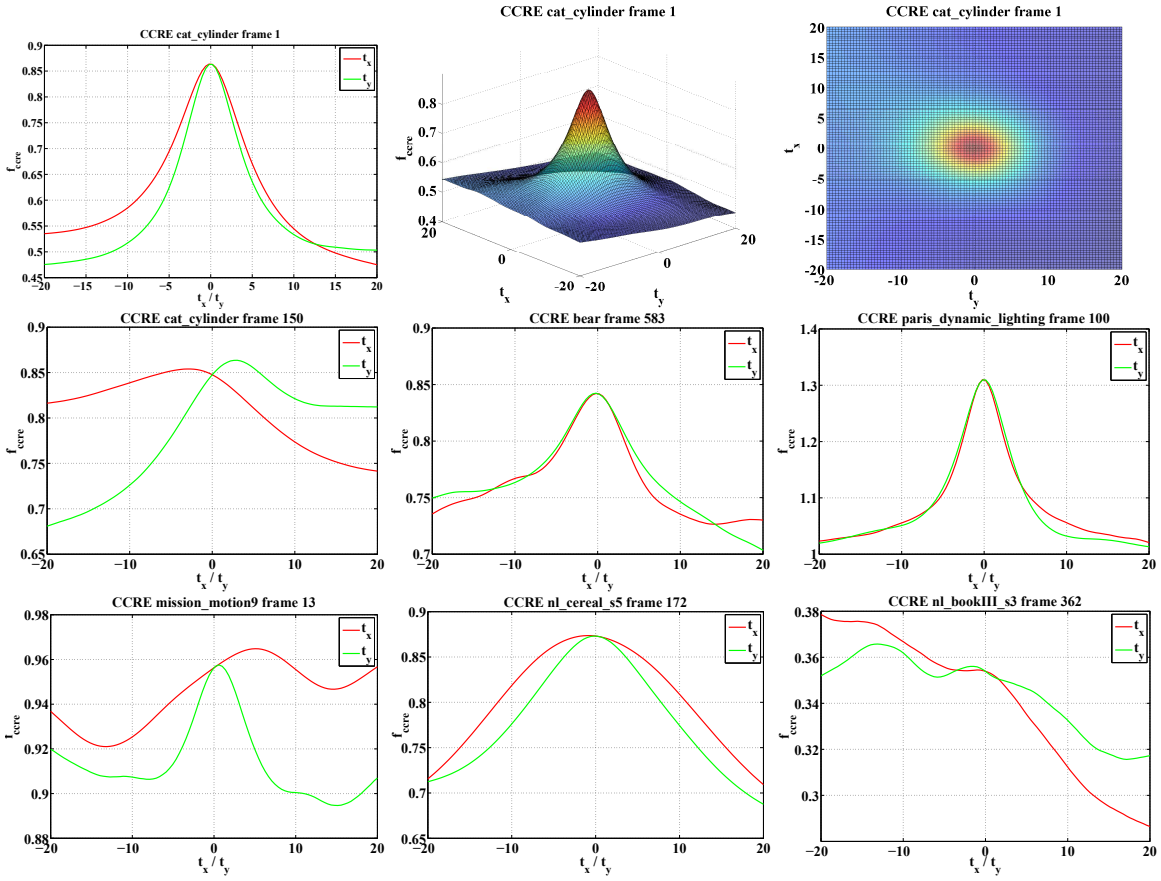


Figure 5.9: f_{ccre} plotted against x and y translation.

with

$$P_{tt}(i,j) = \frac{1}{N}\sum_{k=1}^{N}\psi(i - I_{kt})\psi(j - I_{kt}) \tag{5.33}$$

$$\frac{\partial P_{tt}(i,j)}{\partial I_{kt}} = \frac{1}{N}\sum_{k=1}^{N}\frac{\partial\psi(i - I_{kt})}{\partial I_{kt}}\psi(j - I_{kt}) \tag{5.34}$$

$$\frac{\partial^2 P_{tt}(i,j)}{\partial I_{kt}^2} = \frac{1}{N}\sum_{k=1}^{N}\frac{\partial^2\psi(i - I_{kt})}{\partial I_{kt}^2}\psi(j - I_{kt}) \tag{5.35}$$

5.2.3 Cross Cumulative Residual Entropy (CCRE)

CCRE [206, 207] is another AM based on information theory and is very similar to MI - the only difference between the two is that CCRE uses *cumulative* joint and

marginal histograms instead of P_{t0} and P_t. CCRE has been shown to offer several advantages over MI [208] including a larger convergence region, smoother function surface and higher range of values which should all allow for more numerically stable optimization. It has been used successfully for image registration [208, 209, 210], but, to the best of the author's knowledge, has never been applied for RBT even though the relevant formulations for GD based SMs have been reported [4].

CCRE is formulated as:

$$f_{ccre} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1} P^*_{t0}(i,j)\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right) \tag{5.36}$$

where the cumulative joint and marginal histograms P^*_{t0}, P^*_t are given as [208]:

$$P^*_{t0}(i,j) = \sum_{i=0}^{L-1} P_{t0}(i,j) = \frac{1}{N}\sum_{k=1}^{N}\psi^*(i-I_{kt})\psi(j-I_{k0}) \tag{5.37}$$

and

$$P^*_t(i) = \sum_{j=0}^{L-1} P^*_{t0}(i,j) \tag{5.38}$$

with $\psi^*(s)$ denoting the cumulative cubic B-Spline function that is identical to the quintic B-Spline:

$$\psi^*(s) = \int_z^\infty \psi(z)dz = \begin{cases} 1 & \text{if } s < -2 \\ 1 - \frac{(2+s)^4}{24} & \text{if } -2 \leq s \leq -1 \\ \frac{1}{2} - \frac{2s}{3} + \frac{s^3}{3} + \frac{s^4}{8} & \text{if } -1 < s \leq 0 \\ \frac{1}{2} - \frac{2s}{3} + \frac{s^3}{3} - + \frac{s^4}{8} & \text{if } 0 < s \leq 1 \\ \frac{(s-2)^4}{24} & \text{if } 1 < s < 2 \\ 0 & \text{otherwise} \end{cases} \tag{5.39}$$

Further, the Jacobian of CCRE is given as [208, 4]:

$$\frac{\partial f_{ccre}}{\partial \mathbf{I_t}} = \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\left[\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_t}}\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right)\right] \tag{5.40}$$

where the k^{th} element of $\dfrac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_t}}$ is computed as:

$$\frac{\partial P^*_{t0}(i,j)}{\partial I_{kt}} = \frac{1}{N}\frac{\partial \psi^*(i-I_{kt})}{\partial I_{kt}}\psi(j-I_{k0}) \tag{5.41}$$

with $\frac{\partial \psi^*(i - I_{kt})}{\partial I_{kt}} = -\psi(i - I_{kt})$.

Finally, the $(k,l)^{th}$ element of the CCRE Hessian is given as [4]:

$$\frac{\partial^2 f_{ccre}}{\partial I_{kt} \partial I_{lt}} = \sum_{i=0}^{L-1} \sum_{j=0}^{L-1} \left[\frac{\partial^2 P^*_{t0}(i,j)}{\partial I_{kt} \partial I_{lt}} \log \left(\frac{P^*_{t0}(i,j)}{P^*_t(i) P_0(j)} \right) + \frac{1}{P^*_{t0}(i,j)} \frac{\partial P^*_{t0}(i,j)}{\partial I_{kt}} \frac{\partial P^*_{t0}(i,j)}{\partial I_{lt}} - \frac{1}{P^*_t(i)} \frac{\partial P^*_{t0}(i,j)}{\partial I_{lt}} \sum_{m=0}^{L-1} \left(\frac{\partial P^*_{t0}(i,m)}{\partial I_{kt}} \right) \right] \quad (5.42)$$

where

$$\frac{\partial^2 P^*_{t0}(i,j)}{\partial I_{kt} \partial I_{lt}} = \frac{1}{N} \frac{\partial^2 \psi^*(i - I_{kt})}{\partial I_{kt} \partial I_{lt}} \psi(j - I_{k0}) \quad (5.43)$$

with $\frac{\partial^2 \psi^*(i - I_{kt})}{\partial I_{kt}^2} = \frac{\partial \psi(i - I_{kt})}{\partial I_{kt}}$ and $\frac{\partial^2 \psi^*(i - I_{kt})}{\partial I_{kt} \partial I_{lt}} = 0$ for $k \neq l$. As for MI, the self Hessian $\frac{\partial^2 f_{ccre}(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I_t}^2}$ is formulated as in Eq. 5.42 except that the various histograms and their derivatives are computed after substituting $\mathbf{I_0} = \mathbf{I_t}$.

5.2.3.1 Inverse Derivatives

It can be noted from Eq. 5.36 that, unlike MI, CCRE is not a symmetrical measure, i.e. $f_{ccre}(\mathbf{I_0}, \mathbf{I_t}) \neq f_{ccre}(\mathbf{I_t}, \mathbf{I_0})$. This has important implications for computing the inverse derivatives $\frac{\partial f_{ccre}}{\partial \mathbf{I_0}}$ and $\frac{\partial^2 f_{ccre}}{\partial \mathbf{I_0}^2}$ needed by ICLK and ESM (Sec. 4.1) since these cannot be obtained by simply switching the roles of $\mathbf{I_0}$ and $\mathbf{I_t}$ in Eqs. 5.40 and 5.42. One solution is to change the form of Eq. 5.36 itself by interchanging the roles of $\mathbf{I_0}$ and $\mathbf{I_t}$ and the resultant form is termed as inverse CCRE (ICCRE):

$$f_{iccre} = \sum_{i,j} P^*_{0t}(i,j) \log \left(\frac{P^*_{0t}(i,j)}{P^*_0(i) P_t(j)} \right) \quad (5.44)$$

with

$$P^*_{0t}(i,j) = \sum_{i=0}^{L-1} P_{0t}(i,j) = \frac{1}{N} \sum_{k=1}^{N} \psi^*(i - I_{k0}) \psi(j - I_{kt}) \quad (5.45)$$

and

$$P^*_0(i) = \sum_{j=0}^{L-1} P^*_{0t}(i,j) \quad (5.46)$$

Though this formulation allows Eqs. 5.40 and 5.42 to be used for computing the inverse derivatives and also makes more sense from the perspective of searching I_0 for the optimal warp, experiments indicated that ICLK and ESM do not work as well with it as the one in Eq. 5.36. A similar approach was proposed in [207] where the mean of f_{ccre} and f_{iccre} was taken to obtain a symmetrical version of CCRE. This latter was not explored in the current work both due to the poor performance of f_{iccre} and the significantly more computational load involved in using it. As a result, f_{ccre} was used for ICLK and ESM too and its derivatives w.r.t. $\mathbf{I_0}$ are presented next.

$$\begin{aligned}\frac{\partial f_{ccre}}{\partial \mathbf{I_0}} &= \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\left[\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right)\right.\\ &\left.+ P^*_{t0}(i,j)\frac{P^*_t(i)P_0(j)}{P^*_{t0}(i,j)}\frac{1}{P^*_t(i)P_0(j)}\left(\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}} - \frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\frac{\partial P_0(j)}{\partial \mathbf{I_0}}P^*_t(i)\right)\right]\\ &= \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\left[\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}\left(1+\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right)\right) - \frac{P^*_{t0}(i,j)}{P_0(j)}\frac{\partial P_0(j)}{\partial \mathbf{I_0}}\right] \end{aligned} \tag{5.47}$$

$$\begin{aligned}\frac{\partial^2 f_{ccre}}{\partial \mathbf{I_0}^2} &= \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\left[\left\{\frac{\partial^2 P^*_{t0}(i,j)}{\partial \mathbf{I_0}^2}\left(\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right)+1\right) + \frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}\left(\frac{\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}}{P^*_{t0}(i,j)} - \frac{\frac{\partial P_0(j)}{\partial \mathbf{I_0}}}{P_0(j)}\right)\right\}\right.\\ &\left.- \left\{\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}\frac{\frac{\partial P_0(j)}{\partial \mathbf{I_0}}}{P_0(j)} + P^*_{t0}(i,j)\left(\frac{\frac{\partial^2 P_0(j)}{\partial \mathbf{I_0}^2}}{P_0(j)} - \left(\frac{\frac{\partial P_0(j)}{\partial \mathbf{I_0}}}{P_0(j)}\right)^2\right)\right\}\right]\\ &= \sum_{i=0}^{L-1}\sum_{j=0}^{L-1}\left[\frac{\partial^2 P^*_{t0}(i,j)}{\partial \mathbf{I_0}^2}\left(\log\left(\frac{P^*_{t0}(i,j)}{P^*_t(i)P_0(j)}\right)+1\right) + \frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}\left(\frac{\frac{\partial P^*_{t0}(i,j)}{\partial \mathbf{I_0}}}{P^*_{t0}(i,j)} - 2\frac{\frac{\partial P_0(j)}{\partial \mathbf{I_0}}}{P_0(j)}\right)\right.\\ &\left.- P^*_{t0}(i,j)\left(\frac{\frac{\partial^2 P_0(j)}{\partial \mathbf{I_0}^2}}{P_0(j)} - \left(\frac{\frac{\partial P_0(j)}{\partial \mathbf{I_0}}}{P_0(j)}\right)^2\right)\right] \end{aligned} \tag{5.48}$$

5.2.4 Structural Similarity (SSIM)

SSIM [116] is a popular image quality measure for evaluating the loss in fidelity incurred by image compression algorithms like JPEG by comparing the compressed image with the original one. It has been very popular in this domain since it closely mirrors the approach adopted by the human visual system to subjectively evaluate the quality of an image. Since it measures the information loss in the compressed image - essentially a slightly distorted or damaged version of the original - it also makes a

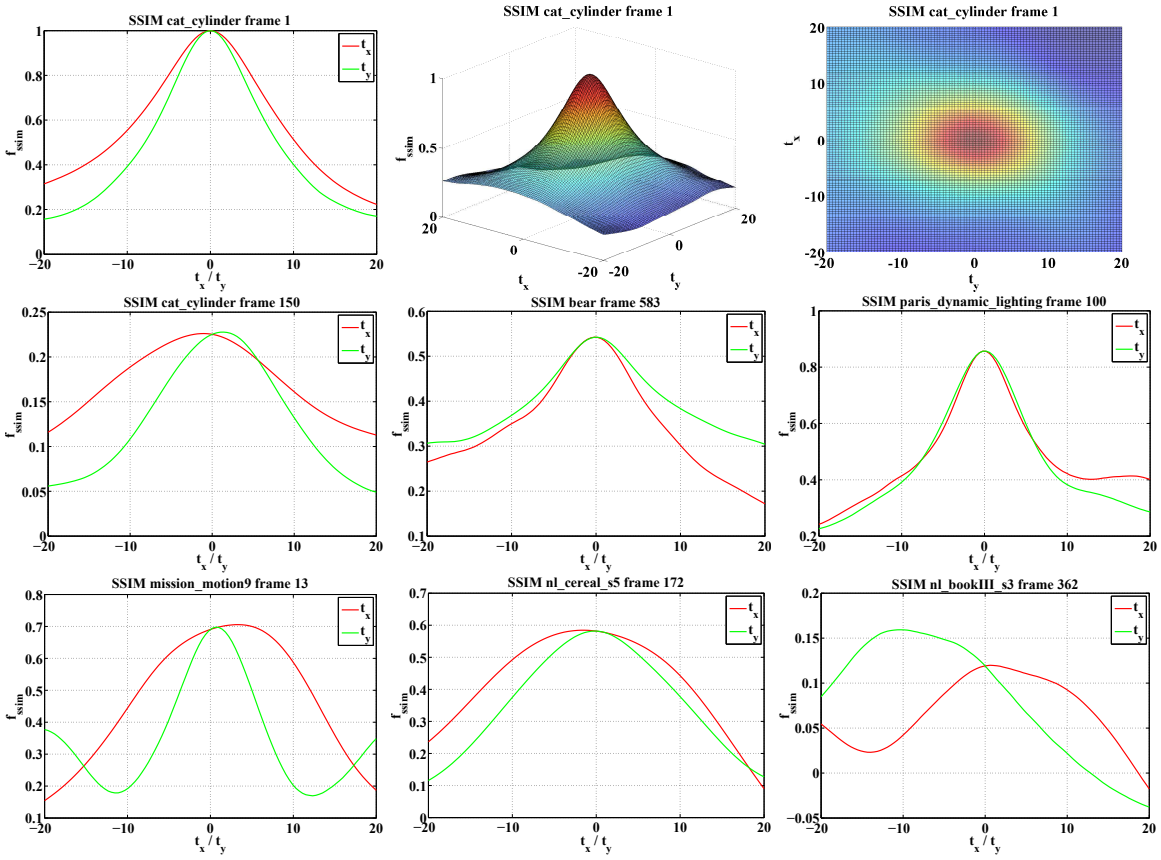


Figure 5.10: f_{ssim} plotted against x and y translation.

suitable metric for comparing candidate warped patches with the template to find the one with the minimum loss and thus most likely to represent the same object. Further, it has been designed to capture the perceptual similarity of images and is known to be robust to illumination and contrast changes [116]. It is reasonable, therefore, to expect it to perform well for tracking under challenging conditions. As such, it has indeed been used for tracking before with particle filters [117, 121], gradient ascent [119] and hybrid methods [118]. All of these have been imprecise trackers, however, estimating low DOF motion limited to translation and scaling of the target patch. To the best of the author's knowledge, no attempt has thus far been made to use SSIM for high DOF RBT within the LK framework [16].

SSIM is defined as a product of 3 components:

$$f_{ssim} = \left(\frac{2\mu_t\mu_0 + C_1}{\mu_t^2 + \mu_0^2 + C_1}\right)^{\alpha} \left(\frac{2\sigma_t\sigma_0 + C_2}{\sigma_t^2 + \sigma_0^2 + C_2}\right)^{\beta} \left(\frac{\sigma_{t0} + C_3}{\sigma_t\sigma_0 + C_3}\right)^{\gamma} \tag{5.49}$$

where μ_t is the mean and σ_t is the sample standard deviation of $\mathbf{I_t}$ while σ_{t0} is the

sample *covariance* between $\mathbf{I_t}$ and $\mathbf{I_0}$. The three components of f_{ssim} from left to right are respectively used for luminance, contrast and structure comparison between the two patches. The positive constants α, β, γ are used to assign relative weights to these components while C_1, C_2, C_3 are added to ensure their numerical stability with small denominators. Here, as in most practical implementations [116, 117, 118, 119, 121], it is assumed that $\alpha = \beta = \gamma = 1$ and $C_3 = \dfrac{C_2}{2}$ so that Eq. 5.49 simplifies to:

$$f_{ssim} = \frac{(2\mu_t\mu_0 + C_1)\,(2\sigma_{t0} + C_2)}{(\mu_t^2 + \mu_0^2 + C_1)\,(\sigma_t^2 + \sigma_0^2 + C_2)} \tag{5.50}$$

For clarity and brevity in the subsequent expressions, SSIM is expressed in a simplified form as:

$$f_{ssim} = \frac{ab}{cd} \tag{5.51}$$

with $a = 2\mu_t\mu_0 + C_1$, $b = 2\sigma_{t0} + C_2$, $c = \mu_t^2 + \mu_0^2 + C_1$ and $d = \sigma_t^2 + \sigma_0^2 + C_2$. Further, as in Eq. 5.18, $\bar{\mathbf{I}}_\mathbf{t}$ refers to a mean normalized version of $\mathbf{I_t}$ so that $\bar{\mathbf{I}}_\mathbf{t} = \mathbf{I_t} - \mu_t$ and $\bar{I}_{kt} = \bar{\mathbf{I}}_\mathbf{t}(\hat{\mathbf{x}}_{kt}) = \bar{\mathbf{I}}_\mathbf{t}(\hat{\mathbf{x}}_{kt}) - \mu_t$ with $\sum_{k=1}^{N} \bar{I}_{kt} = 0$. Differentiating Eq. 5.51 w.r.t. $\mathbf{I_t}$ gives:

$$\frac{\partial f_{ssim}}{\partial \mathbf{I_t}} = \frac{1}{cd}\left[(\mathbf{a}'b + \mathbf{b}'a) - f_{ssim}(\mathbf{c}'d + \mathbf{d}'c)\right] \tag{5.52}$$

with

$$\mathbf{a}' = \frac{\partial a}{\partial \mathbf{I_t}} = \frac{2\mu_0}{N}\frac{\partial \sum_{k=1}^{N} I_{kt}}{\partial \mathbf{I_t}} = \frac{2\mu_0}{N} \tag{5.53}$$

$$\mathbf{b}' = \frac{\partial b}{\partial \mathbf{I_t}} = \frac{2}{N-1}\frac{\partial \sum_{i=1}^{N} \bar{I}_{kt}\bar{I}_{k0}}{\partial \mathbf{I_t}} = \frac{2\bar{\mathbf{I}}_\mathbf{0}}{N-1} \tag{5.54}$$

$$\mathbf{c}' = \frac{\partial c}{\partial \mathbf{I_t}} = \frac{2\mu_t}{N}\frac{\partial \sum_{k=1}^{N} I_{kt}}{\partial \mathbf{I_t}} = \frac{2\mu_t}{N} \tag{5.55}$$

$$\mathbf{d}' = \frac{\partial d}{\partial \mathbf{I_t}} = \frac{1}{N-1}\frac{\partial \sum_{k=1}^{N} (\bar{I}_{kt})^2}{\partial \mathbf{I_t}} = \frac{2\bar{\mathbf{I}}_\mathbf{t}}{N-1} \tag{5.56}$$

The last equality in Eq. 5.54 follows since $\forall j \in \{1..N\}$, $\dfrac{\partial \sum_{k=1}^{N} \bar{I}_{kt}\bar{I}_{k0}}{\partial I_{jt}} = \bar{I}_{j0} - \dfrac{1}{N}\sum_{i=1}^{N} \bar{I}_{kt} = \bar{I}_{j0}$. Similar reasoning holds for Eq. 5.56 too. Substituting Eqs. 5.53 - 5.56 in Eq.

5.52 gives:

$$\frac{\partial f_{ssim}}{\partial \mathbf{I_t}} = \frac{2}{cd}\left[\left(\frac{\mu_0 b}{N} + \frac{a\bar{\mathbf{I}}_\mathbf{0}}{N-1}\right) - f_{ssim}\left(\frac{\mu_t d}{N} + \frac{c\bar{\mathbf{I}}_\mathbf{t}}{N-1}\right)\right] \tag{5.57}$$

Referring to f_{ssim} as f and $\frac{\partial f_{ssim}}{\partial \mathbf{I_t}}$ as $\mathbf{f}'$ for brevity and letting $f'_A = \frac{b\mu_0 - fd\mu_t}{N}$, $\mathbf{f'_B} = \frac{a\bar{\mathbf{I}}_\mathbf{0} - fc\bar{\mathbf{I}}_\mathbf{t}}{N-1}$ and $f'_C = \frac{cd}{2}$, Eq. 5.57 can be rewritten as:

$$\mathbf{f}' = \frac{f'_A + \mathbf{f'_B}}{f'_C} \tag{5.58}$$

Differentiating Eq. 5.58 w.r.t. $\mathbf{I_t}$ gives:

$$\frac{\partial^2 f_{ssim}}{\partial \mathbf{I_t}^2} = \frac{1}{f'_C}\left(\frac{\partial f'_A}{\partial \mathbf{I_t}} + \frac{\partial \mathbf{f'_B}}{\partial \mathbf{I_t}} - \mathbf{f}'^T \frac{\partial f'_C}{\partial \mathbf{I_t}}\right) \tag{5.59}$$

with

$$\frac{\partial f'_A}{\partial \mathbf{I_t}} = \frac{1}{N}\left[\mu_0 \mathbf{b}' - \mu_t(d\mathbf{f}' + f\mathbf{d}') - \frac{fd}{N}\right] \tag{5.60}$$

$$\frac{\partial \mathbf{f'_B}}{\partial \mathbf{I_t}} = \frac{1}{N-1}\left(\bar{\mathbf{I}}_\mathbf{0}\mathbf{a}' - \bar{\mathbf{I}}_\mathbf{t}(c\mathbf{f}' + f\mathbf{c}') - fc\mathbb{I}_N\right) \tag{5.61}$$

$$\frac{\partial f'_C}{\partial \mathbf{I_t}} = \frac{d\mathbf{c}' + c\mathbf{d}'}{2} = \frac{1}{2}\left(\frac{\mu_t d}{N} + \frac{c\bar{\mathbf{I}}_\mathbf{t}}{N-1}\right) \tag{5.62}$$

where $\mathbb{I}_N$ is the $N \times N$ identity matrix and it may be recalled from Sec. 3.1 that the addition of the $1 \times N$ vector $\frac{\partial f'_A}{\partial \mathbf{I_t}}$ with the $N \times N$ matrix $\frac{\partial \mathbf{f'_B}}{\partial \mathbf{I_t}}$ is to be performed row-wise.

Substituting Eqs. 5.60, 5.61, 5.62 in Eq. 5.59 and simplifying gives:

$$\frac{\partial^2 f_{ssim}}{\partial \mathbf{I_t}^2} = \frac{2}{cd}\left[\frac{1}{N}\left(\frac{4\left(\mu_0\bar{\mathbf{I}}_\mathbf{0} - \mu_t f\bar{\mathbf{I}}_\mathbf{t}\right)}{N-1} - \frac{3\mu_t d}{2}\mathbf{f}' - \frac{fd}{N}\right) - \frac{c}{N-1}\left(\frac{3}{2}\mathbf{f}'^T\bar{\mathbf{I}}_\mathbf{t} + f\mathbb{I}_N\right)\right] \tag{5.63}$$

Setting $\mathbf{I_0} = \mathbf{I_t}$ for computing $\hat{\mathbf{H}}_{self}$ makes $\mu_t = \mu_0$, $\mathbf{a}' = \mathbf{c}'$, $\mathbf{b}' = \mathbf{d}'$, $f = 1$ and $\mathbf{f}' = 0$ so that Eq. 5.63 simplifies to:

$$\frac{\partial^2 f_{ssim}(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I_t}^2} = \frac{-2}{c_{self} d_{self}}\left[\frac{d_{self}}{N^2} + \frac{c_{self}}{N-1}\mathbb{I}_N\right] \tag{5.64}$$

with $c_{self} = 2\mu_t^2 + C_1$ and $d_{self} = 2\sigma_t^2 + C_2$.

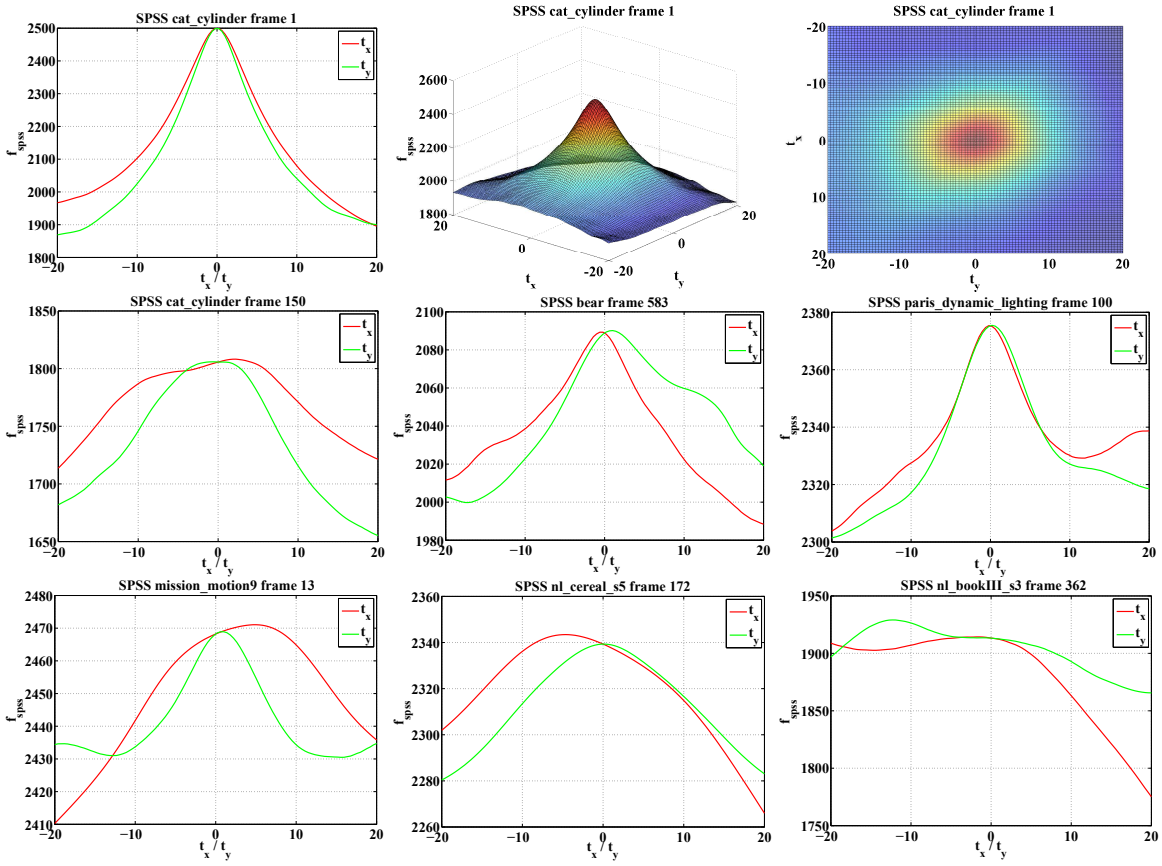


Figure 5.11: f_{spss} plotted against x and y translation.

5.2.5 Sum of Pixelwise Structural Similarity (SPSS)

In the formulation described so far, SSIM has been computed over the *entire* patch - i.e. μ_t, σ_t and σ_{t0} have been computed over all N pixels in $\mathbf{I_t}$ and $\mathbf{I_0}$. In its original form [116], however, the expression in Eq. 5.50 was applied to several corresponding *sub windows* within the two patches - for instance 8×8 or 11×11 sub windows that are moved pixel-by-pixel over the entire patch - and the *mean* of all resultant scores was taken as the overall similarity measure. For tracking applications, such an approach is not only impracticable from speed perspective, it presents another issue for GD based SMs - presence of insufficient texture in these small sub windows may cause Eq. 4.1 to become ill posed if $\mathbf{J}$ and $\mathbf{H}$ are computed for each sub window and then averaged.

As a result, the previous section considered only one end of the spectrum of variation in the size and number of sub windows - a single sub window of the same

size as the patch. Now, if the *other* end of the spectrum is considered - N sub windows of size 1×1 each - then a different AM is obtained that may provide some idea about the effect of window wise operations while also being much simpler and possibly faster. The resultant model is termed as **Sum of Pixelwise Structural Similarity** or **SPSS**. When considered pixel wise, σ_t and σ_{t0} become null while μ_t becomes equal to the pixel value itself so that Eq. 5.50 simplifies to:

$$f_{spss} = \sum_{k=1}^{N} \frac{2I_{kt}I_{k0} + C_1}{I_{kt}^2 + I_{k0}^2 + C_1} \tag{5.65}$$

Similar to SSD, contributions from different pixels to f_{spss} are independent of each other so that each entry of $\partial f_{spss}/\partial \mathbf{I_t}$ has contribution only from the corresponding pixel. This also holds true for each entry of the principal diagonal of $\partial^2 f_{spss}/\partial \mathbf{I_t}^2$ (which is a diagonal matrix). Denoting the contributions of the k^{th} pixel to f_{spss}, $\partial f_{spss}/\partial \mathbf{I_t}$ and principal diagonal of $\partial^2 f_{spss}/\partial \mathbf{I_t}^2$ respectively as f_k, f'_k and f''_k, gives:

$$f'_k = \frac{\partial f_{spss}}{\partial I_{kt}} = \frac{2(I_{k0} - I_{kt}f_k)}{I_{kt}^2 + I_{k0}^2 + C_1} \tag{5.66}$$

$$f''_k = \frac{\partial^2 f_{spss}}{\partial I_{kt}^2} = \frac{-2(f_k + 3I_{kt}f'_k)}{I_{kt}^2 + I_{0t}^2 + C_1} \tag{5.67}$$

$$f''_k(\mathbf{I_t}, \mathbf{I_t}) = \frac{\partial^2 f_{spss}(\mathbf{I_t}, \mathbf{I_t})}{\partial I_{kt}^2} = \frac{-2}{2I_{kt}^2 + C_1} \tag{5.68}$$

5.2.6 Ratio Image Uniformity (RIU)

RIU was originally introduced [107] for registration of PET images though it has also been widely used [110] for MR images [108, 109]. To the best of the author's knowledge, however, it has never been used for RBT. Also known as the variance of intensity ratios (VIR), it defines the similarity between two images as the uniformity of the image constructed by taking the pixel wise ratio between them, which is maximized by minimizing the variance of this image. An extended version of RIU, called Partitioned Intensity Uniformity (PIU) in [110], has also been proposed [108] but has not been considered here as it is much more involved and computationally expensive to obtain its derivatives.

RIU is formulated as:

$$f_{riu} = -\frac{\|\mathbf{R_{t0}} - \bar{R}_{t0}\|^2}{\bar{R}_{t0}} = -\frac{\sum_{k=1}^{N}(R_{kt0} - \bar{R}_{t0})^2}{\bar{R}_{t0}} \tag{5.69}$$

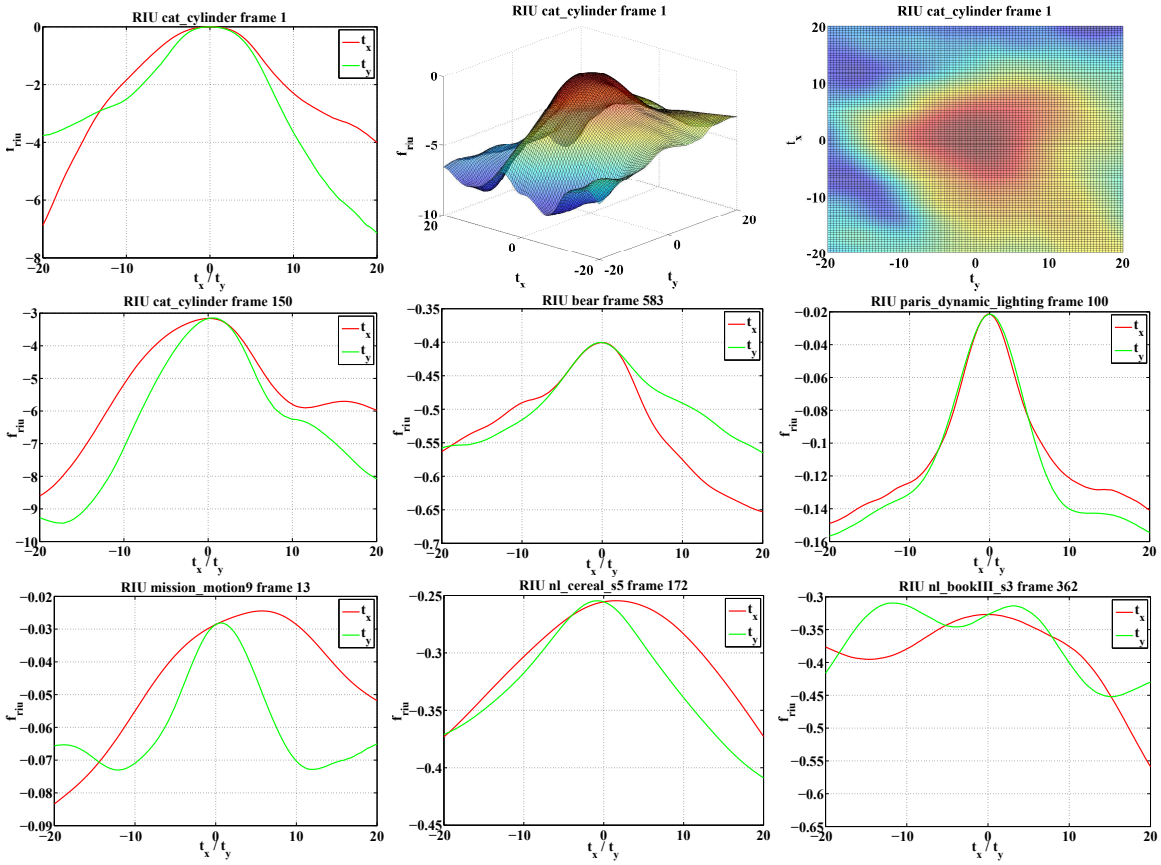


Figure 5.12: f_{riu} plotted against x and y translation.

where

$$\mathbf{R_{t0}} = \frac{\mathbf{I_t} + C}{\mathbf{I_0} + C} \tag{5.70}$$

is the ratio patch, $R_{kt0} = \dfrac{I_{kt} + C}{I_{k0} + C}$ is its k^{th} element, $\bar{R}_{t0} = \sum_{k=1}^{N} R_{kt0}$, is its mean and C is a scalar constant added to provide numerical stability with small values in $\mathbf{I_0}$. Assuming $C = 1$ and differentiating Eq. 5.69 w.r.t. $\mathbf{I_t}$ gives:

$$\frac{\partial f_{riu}}{\partial \mathbf{I_t}} = \frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}} \frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} \tag{5.71}$$

with

$$\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} = \text{diag}\left(\frac{1}{\mathbf{I_0} + 1}\right) \tag{5.72}$$

and the k^{th} element of $\frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}$ given as:

$$\begin{aligned}
\frac{\partial f_{riu}}{\partial R_{kt0}} &= -\frac{1}{N\bar{R}_{t0}}\left[\frac{\partial}{\partial R_{kt0}}\left(\sum_{k=1}^{N}(R_{kt0}-\bar{R}_{t0})^2\right) - f_{riu}\frac{\partial \bar{R}_{t0}}{\partial R_{kt0}}\right] \\
&= -\frac{1}{N\bar{R}_{t0}}\left[2(R_{kt0}-\bar{R}_{t0})\left(1-\frac{1}{N}\right) - \frac{1}{N}\sum_{j=1,j\neq k}^{N}(R_{jt0}-\bar{R}_{t0}) - \frac{f_{riu}}{N}\right] \\
&= -\frac{1}{N\bar{R}_{t0}}\left[2(R_{kt0}-\bar{R}_{t0}) - \frac{1}{N}\sum_{j=1}^{N}(R_{jt0}-\bar{R}_{t0}) - \frac{f_{riu}}{N}\right] \\
&= \frac{1}{N\bar{R}_{t0}}\left[\frac{f_{riu}}{N} - 2(R_{kt0}-\bar{R}_{t0})\right] \\
&= \frac{1}{N}\left[\frac{f_{riu}}{N\bar{R}_{t0}} - 2\left(\frac{R_{kt0}}{\bar{R}_{t0}}-1\right)\right]
\end{aligned} \tag{5.73}$$

where diag(K) refers to a diagonal matrix with K as the principal diagonal.

Substituting Eqs. 5.72 and 5.73 in Eq. 5.71 gives the k^{th} element of $\frac{\partial f_{riu}}{\partial \mathbf{I_t}}$ as:

$$\frac{\partial f_{riu}}{\partial I_{kt}} = \frac{1}{N\bar{R}_{t0}(I_{k0}+1)}\left[\frac{f_{riu}}{N} - 2(R_{kt0}-\bar{R}_{t0})\right] \tag{5.74}$$

Differentiating Eq. 5.71 w.r.t. $\mathbf{I_t}$ gives:

$$\frac{\partial^2 f_{riu}}{\partial \mathbf{I_t}^2} = \frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}}^T \frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2}\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} + \frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}\frac{\partial^2 \mathbf{R_{t0}}}{\partial \mathbf{I_t}^2} = \frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}}^T \frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2}\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} \tag{5.75}$$

since $\frac{\partial^2 \mathbf{R_{t0}}}{\partial \mathbf{I_t}^2}$ is null. It follows from Eq. 5.73 that:

$$\frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}} = \frac{1}{N}\left[\frac{f_{riu}}{N\bar{R}_{t0}} - 2\left(\frac{\mathbf{R_{t0}}}{\bar{R}_{t0}}-1\right)\right] \tag{5.76}$$

Differentiating Eq. 5.76 w.r.t. $\mathbf{R_{t0}}$ gives:

$$\frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2} = \frac{1}{N}\left[\frac{1}{N}\frac{\partial}{\partial \mathbf{R_{t0}}}\left(\frac{f_{riu}}{\bar{R}_{t0}}\right) - 2\frac{\partial}{\partial \mathbf{R_{t0}}}\left(\frac{\mathbf{R_{t0}}}{\bar{R}_{t0}}\right)\right] \tag{5.77}$$

Here,

$$\frac{\partial}{\partial \mathbf{R_{t0}}}\left(\frac{f_{riu}}{\bar{R}_{t0}}\right) = \frac{\frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}} - \frac{f_{riu}}{N\bar{R}_{t0}}}{\bar{R}_{t0}} = \frac{\frac{f_{riu}}{N\bar{R}_{t0}}\left(\frac{1}{N}-1\right) - 2\frac{\mathbf{R_{t0}}}{N\bar{R}_{t0}} - \frac{2}{N}}{\bar{R}_{t0}} \tag{5.78}$$

and

$$\frac{\partial}{\partial \mathbf{R_{t0}}}\left(\frac{\mathbf{R_{t0}}}{\bar{R}_{t0}}\right) = \frac{\mathbb{I}_N - \frac{\mathbf{B}}{N}}{\bar{R}_{t0}} = \frac{\mathbb{I}_N - \frac{\mathbf{R_{t0}}}{N\bar{R}_{t0}}}{\bar{R}_{t0}} \tag{5.79}$$

Substituting Eqs. 5.78 and 5.79 in Eq. 5.77 gives:

$$\begin{aligned}\frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2} &= \frac{1}{N}\left[\frac{\frac{f_{riu}}{N\bar{R}_{t0}}\left(\frac{1}{N}-1\right) - 2\frac{\mathbf{R_{t0}}}{N\bar{R}_{t0}} - \frac{2}{N}}{N\bar{R}_{t0}} - 2\frac{\mathbb{I}_N - \frac{\mathbf{R_{t0}}}{N\bar{R}_{t0}}}{\bar{R}_{t0}}\right] \\ &= \frac{1}{N^2\bar{R}_{t0}}\left[\frac{f_{riu}}{N\bar{R}_{t0}}\left(\frac{1}{N}-1\right) - 2\frac{\mathbf{R_{t0}}}{N\bar{R}_{t0}} - \frac{2}{N} - 2N\mathbb{I}_N + \frac{2\mathbf{R_{t0}}}{\bar{R}_{t0}}\right] \\ &= \frac{1}{N^2\bar{R}_{t0}}\left[\frac{N-1}{N\bar{R}_{t0}}\left(2\mathbf{R_{t0}} - \frac{f_{riu}}{N}\right) - 2\left(N\mathbb{I}_N + \frac{1}{N}\right)\right]\end{aligned} \tag{5.80}$$

Further, substituting Eq. 5.80 in Eq. 5.75 gives:

$$\frac{\partial^2 f_{riu}}{\partial \mathbf{I_t}^2} = \frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}}^T \frac{1}{N^2\bar{R}_{t0}}\left[\frac{N-1}{N\bar{R}_{t0}}\left(2\mathbf{R_{t0}} - \frac{f_{riu}}{N}\right) - 2\left(N\mathbb{I}_N + \frac{1}{N}\right)\right]\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} \tag{5.81}$$

Finally, setting $\mathbf{I_0} = \mathbf{I_t}$ for the self Hessian simplifies Eq. 5.81 to:

$$\frac{\partial^2 f_{riu}(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I_t}^2} = -\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}}^T \frac{2}{N^2}\left[N\mathbb{I}_N - \frac{N-2}{N}\right]\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_t}} \tag{5.82}$$

5.2.6.1 Inverse Derivatives

Similar to CCRE, RIU is not a symmetrical measure and therefore its inverse derivatives are also presented here. Differentiating Eq. 5.69 w.r.t. $\mathbf{I_0}$ gives:

$$\frac{\partial f_{riu}}{\partial \mathbf{I_0}} = \frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_0}} \tag{5.83}$$

with $\frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}$ given by Eq. 5.76 and:

$$\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_0}} = \text{diag}\left(\frac{-\mathbf{R_{t0}}}{\mathbf{I_0}+1}\right) \tag{5.84}$$

Further differentiating Eq. 5.83 w.r.t. $\mathbf{I_0}$ gives:

$$\frac{\partial^2 f_{riu}}{\partial \mathbf{I_0}^2} = \frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_0}}^T \frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2}\frac{\partial \mathbf{R_{t0}}}{\partial \mathbf{I_0}} + \frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}\frac{\partial^2 \mathbf{R_{t0}}}{\partial \mathbf{I_0}^2} \tag{5.85}$$

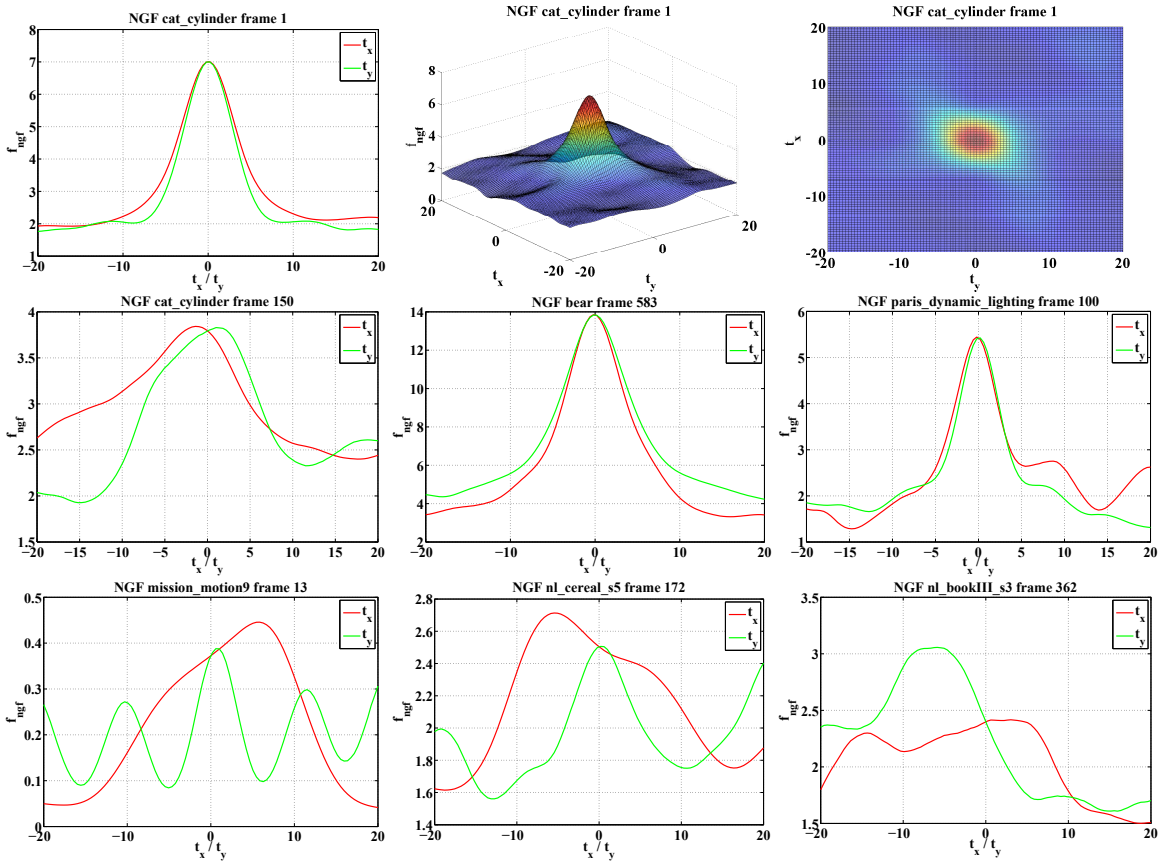


Figure 5.13: f_{ngf} plotted against x and y translation.

where $\frac{\partial^2 f_{riu}}{\partial \mathbf{R_{t0}}^2}$ is given by Eq. 5.80 and $\frac{\partial^2 \mathbf{R_{t0}}}{\partial \mathbf{I_0}^2}$ is a very sparse $N \times N \times N$ tensor with only N non zero entries on its principal diagonal given as:

$$\frac{\partial^2 R_{kt0}}{\partial I_{k0}^2} = \frac{2R_{kt0}}{(1+I_{k0})^2} \tag{5.86}$$

It follows that the second term in Eq. 5.85 is a diagonal matrix whose main diagonal is given by the element wise or Hadamard product of the two $1 \times N$ vectors $\frac{\partial f_{riu}}{\partial \mathbf{R_{t0}}}$ and $\frac{2\mathbf{R_{t0}}}{(1+\mathbf{I_0})^2}$.

5.2.7 Normalized Gradient Fields (NGF)

NGF was introduced by Haber & Modersitzki [111, 112] as an alternative to MI for registration of multi modal images. It is based on the observation that, irrespective of differences in their modality or the illumination conditions under which they were

captured, registered images of the same object will have edges in similar spatial locations. Therefore, a gradient based similarity metric would be more robust to these differences than pixel based measures like SSD. Though MI is also known to be very effective for multi modal registration, NGF offers several advantages including a simpler and easier to interpret formulation and a wider basin of convergence. It has been widely used for image registration in both medical imaging [113, 144] and airborne surveillance [211] applications and its popularity has led to the development of several efficient and parallel implementations [212, 114, 213, 214] to offset its relatively computationally expensive formulation.

Since edges are marked by rapid changes in intensity levels, spatial gradients can be used as simple and fairly reliable indicators for the location and orientation of edges. However, gradients also measure the *magnitude* of intensity change, or equivalently the strength of these edges, which can be misleading when there exist differences of modality or illumination conditions between the two patches being compared. A simple approach to avoid this undesirable information is to divide the spatial gradients by their magnitude to normalize them. As a result, NGF measures the similarity between two patches by the pixel wise alignment between the 2D unit vectors corresponding to the horizontal and vertical spatial gradients of the patches. Two vectors can be aligned by minimizing the angle between them. This in turn can be achieved by either minimizing the sine or maximizing the cosine of this angle, corresponding respectively to the cross and dot product between the vectors. Since the former is more costly to compute, it is the latter that is usually used in practice [144, 212, 114, 214], including the current work.

NGF is thus formulated as:

$$f_{ngf} = \sum_{k=1}^{N} \frac{\|\hat{\nabla}\mathbf{I_0}(\mathbf{x}_{k0}) \cdot \hat{\nabla}\mathbf{I_t}(\mathbf{x}_{kt})\|^2}{(\|\hat{\nabla}\mathbf{I_0}(\mathbf{x}_{k0})\|^2 + \eta^2)(\|\hat{\nabla}\mathbf{I_t}(\mathbf{x}_{kt})\|^2 + \eta^2)} \tag{5.87}$$

where the scalar η is used to reduce the impact of noise by considering only those intensity changes as edges where $\|\hat{\nabla}\mathbf{I}(\mathbf{x}_k)\| > \eta$. This work uses $\eta = 100$ after [144], from whose accompanying C code the implementation of NGF in MTF has been adapted. The estimated gradient $\hat{\nabla}\mathbf{I}$ used here is computed in a slightly different way than the gradient $\nabla\mathbf{I}$ used in Sec. 4.1 (e.g. Eqs. 4.2 and 4.3) so that f_{ngf} is differentiable w.r.t. $\mathbf{I_0}$ and $\mathbf{I_t}$. Recalling from Sec. 3.1 that $\mathbf{x_t}$ is populated in row

major order, the k^{th} element of $\hat{\nabla}\mathbf{I_t}$ is computed as:

$$\hat{\nabla}\mathbf{I_t}(\mathbf{x}_{kt}) = \hat{\nabla}\mathbf{I}_{kt} = [\hat{\nabla}_x I_{kt}, \hat{\nabla}_y I_{kt}] = \left[\frac{\mathbf{I_t}(\mathbf{x}_{u^+t}) - \mathbf{I_t}(\mathbf{x}_{u^-t})}{2}, \frac{\mathbf{I_t}(\mathbf{x}_{v^+t}) - \mathbf{I_t}(\mathbf{x}_{v^-t})}{2}\right] \tag{5.88}$$

with

$$(u^+, u^-) = \begin{cases} (k+1, k) & \text{if } k_x = 1 \\ (k, k-1) & \text{if } k_x = N_x \\ (k+1, k-1) & \text{otherwise} \end{cases} \tag{5.89}$$

and

$$(v^+, v^-) = \begin{cases} (k+N_x, k) & \text{if } k_y = 1 \\ (k, k-N_x) & \text{if } k_y = N_y \\ (k+N_x, k-N_x) & \text{otherwise} \end{cases} \tag{5.90}$$

for $1 \leq k = k_x + (k_y - 1)N_x \leq N$, $1 \leq k_x \leq N_x$ and $1 \leq k_y \leq N_y$.

The expression in Eq. 5.87 can be rewritten as the squared ℓ^2 norm of a residual that measures the degree of alignment between the gradients of the two patches:

$$f_{ngf} = \|\mathbf{r}\|^2 = \|\mathbf{r_1} \odot \mathbf{r_2}\|^2 \tag{5.91}$$

where $\mathbf{r}, \mathbf{r_1}, \mathbf{r_2} \in \mathbb{R}^N$ with

$$\mathbf{r_1}(k) = r_{k1} = \hat{\nabla}\mathbf{I}_{k0} \cdot \hat{\nabla}\mathbf{I}_{kt} \tag{5.92}$$

$$\mathbf{r_2}(k) = r_{k2} = \frac{1}{\sqrt{\|\hat{\nabla}\mathbf{I}_{k0}\|^2 + \eta^2}\sqrt{\|\hat{\nabla}\mathbf{I}_{kt}\|^2 + \eta^2}} \tag{5.93}$$

and $\odot$ denoting the Hadamard product so that $\mathbf{r}(k) = r_k = r_{k1}r_{k2}$. Further, defining $\mathbf{F_x}, \mathbf{F_y} \in \mathbb{R}^N$ as:

$$\mathbf{F_x}(k) = F_{kx} = \frac{r_{k2}}{2}\left(\hat{\nabla}_x I_{k0} - \frac{r_{k1}\hat{\nabla}_x I_{kt}}{\|\hat{\nabla}\mathbf{I}_{kt}\|^2 + \eta^2}\right) \tag{5.94}$$

$$\mathbf{F_y}(k) = F_{ky} = \frac{r_{k2}}{2}\left(\hat{\nabla}_y I_{k0} - \frac{r_{k1}\hat{\nabla}_y I_{kt}}{\|\hat{\nabla}\mathbf{I}_{kt}\|^2 + \eta^2}\right) \tag{5.95}$$

the k^{th} element of $\frac{\partial f_{ngf}}{\partial \mathbf{I_t}}$ is given as:

$$\frac{\partial f_{ngf}}{\partial \mathbf{I}_{kt}} = \begin{cases} m_1 - r_{k+N_x} F_{(k+N_x)y} & \text{if } k_y = 1 \\ m_2 + r_{k-N_x} F_{(k-N_x)y} & \text{if } k_y = N_y \\ m_3 + r_{k-N_x} F_{(k-N_x)y} - r_{k+N_x} F_{(k+N_x)y} & \text{otherwise} \end{cases} \tag{5.96}$$

with

$$m_1 = \begin{cases} -r_k(F_{kx} + F_{ky}) - r_{k+1} F_{(k+1)x} & \text{if } k_x = 1 \\ r_k(F_{kx} - F_{ky}) + r_{k-1} F_{(k-1)x} & \text{if } k_x = N_x \\ r_{k-1} F_{(k-1)x} - r_k F_{ky} - r_{k+1} F_{(k+1)x} & \text{otherwise} \end{cases} \tag{5.97}$$

$$m_2 = \begin{cases} -r_k(F_{kx} + F_{ky}) - r_{k+1} F_{(k+1)x} & \text{if } k_x = 1 \\ r_k(F_{kx} + F_{ky}) + r_{k-1} F_{(k-1)x} & \text{if } k_x = N_x \\ r_{k-1} F_{(k-1)x} + r_k F_{ky} - r_{k+1} F_{(k+1)x} & \text{otherwise} \end{cases} \tag{5.98}$$

$$m_3 = \begin{cases} -r_k F_{kx} - r_{k+1} F_{(k+1)x} & \text{if } k_x = 1 \\ r_{k-1} F_{(k-1)x} + r_k F_{kx} & \text{if } k_x = N_x \\ r_{k-1} F_{(k-1)x} - r_{k+1} F_{(k+1)x} & \text{otherwise} \end{cases} \tag{5.99}$$

It can be seen from Eqs. 5.94 - 5.99 that $\frac{\partial f_{ngf}}{\partial \mathbf{I_t}}$ has a complex piecewise form and it follows that $\frac{\partial^2 f_{ngf}}{\partial \mathbf{I_t}^2}$ is even more cumbersome to compute. Since this is not used for any GD method anyway, its formulations are omitted here and only that for the self Hessian is reported. Following Modersitzki [144], from where all of the above as well as the following expressions are adapted, the self Hessian is approximated by neglecting the second order derivative of the residual $\mathbf{r}$ similar to the GN Hessian (Eq. 4.11). As mentioned before, $\mathbf{H}_{gn}$ is identical to $\mathbf{H}_{self}$ for SSD and it is assumed here, as in [144], that the SSD like formulation of Eq. 5.91 will ensure that this approximation is good enough in practice:

$$\frac{\partial^2 f_{ngf}(\mathbf{I_t}, \mathbf{I_t})}{\partial \mathbf{I_t}^2} \approx -\frac{\partial \mathbf{r}}{\partial \mathbf{I_t}}^T \frac{\partial \mathbf{r}}{\partial \mathbf{I_t}} \tag{5.100}$$

with $\frac{\partial \mathbf{r}}{\partial \mathbf{I_t}}$ being a very sparse $N \times N$ matrix where the values and locations of non zero elements in the k^{th} column depend on k_x and k_y and are deducible from Eqs. 5.96

- 5.99. It can be observed that each term in Eq. 5.96, after substituting m_1, m_2, m_3 from Eqs. 5.97 - 5.99, can be expressed as a *sum of products* of one element from $\mathbf{r}$ and one from $\mathbf{F_x}/\mathbf{F_y}$ (or a sum thereof). For each case of k_x, k_y in these equations, the k^{th} column of $\dfrac{\partial \mathbf{r}}{\partial \mathbf{I_t}}$ contains non zero entries at all indices where elements from $\mathbf{r}$ are present in the corresponding term while the values of these entries are equal to the elements from $\mathbf{F_x}/\mathbf{F_y}$ that these are multiplied with. For example, if $k_x = N_x$ and $k_y = 1$, $\dfrac{\partial f_{ngf}}{\partial \mathbf{I}_{kt}} = r_k(F_{kx} - F_{ky}) + r_{k-1}F_{(k-1)x} - r_{k+N_x}F_{(k+N_x)y}$. Therefore, the k^{th} column of $\dfrac{\partial \mathbf{r}}{\partial \mathbf{I_t}}$, with $k = k_x + (k_y - 1)N_x$, contains $F_{kx} - F_{ky}, F_{(k-1)x}$ and $-F_{(k+N_x)y}$ in rows $k, k-1$ and $k+N_x$ respectively.

5.3 Illumination Models

ILMs are used to account for changes in illumination conditions between the capture times of I_0 and I_t by replacing $\mathbf{I_t}$ in f with $\mathbf{g}(\mathbf{I_t}, \mathbf{p_a}) : \mathbb{R}^N \times \mathbb{R}^A \mapsto \mathbb{R}^N$. In order to register $\mathbf{I_t}$ with $\mathbf{I_0}$, the SM will then optimize over the A photometric parameters $\mathbf{p_a}$ along with the S warping parameters $\mathbf{p_s}$ so that the search space becomes $S + A$ dimensional. Though this optimization can be performed both simultaneously and piecewise, only the former case is considered here as the latter was shown to perform worse in [70].

Most physical models of illumination represent the overall pixel intensity at each image location as being the result of diffuse, specular and ambient reflections [84, 85]:

$$I_t(\mathbf{x}_k) = I_{dt}(\mathbf{x}_k) + I_{st}(\mathbf{x}_k) + I_{at} \tag{5.101}$$

As indicated in Eq. 5.101, I_{dt} and I_{st} vary over the image while I_a can be assumed to be constant. The differences in intensity between $\mathbf{I_t}$ and $\mathbf{I_0}$ due to lighting changes can thus be captured as:

$$\mathbf{I_t}(\mathbf{x}_k) = \alpha_k \mathbf{I_{d0}}(\mathbf{x}_k) + \eta_k \mathbf{I_{s0}}(\mathbf{x}_k) + \beta \tag{5.102}$$

where $\boldsymbol{\alpha}, \boldsymbol{\eta} \in \mathbb{R}^N$ respectively measure the difference in diffuse and specular conditions while β accounts for ambient or global changes. Since estimating all $2N$ unknowns in $\boldsymbol{\alpha}, \boldsymbol{\eta}$ will lead to an overdetermined system, all ILMs in literature make the further assumption that the surface being imaged is perfectly Lambertian so that $\boldsymbol{\eta} = \mathbf{0}$ and $\mathbf{I_{d0}} = \mathbf{I_0}$. It may be mentioned here that a slightly different approach to modeling

illumination changes was proposed in [22]. A set of training images of the object taken under varying lighting conditions were first used to compute an eigenbasis $\mathbf{B}$. Differences in illumination between $\mathbf{I_0}$ and $\mathbf{I_t}$ were then compensated for by adding a linear combination of the basis vectors to $\mathbf{I_t}$, i.e. $\mathbf{g}(\mathbf{I_t}, \mathbf{p_a}) = \mathbf{I_t} + \mathbf{p_a} \odot \mathbf{B}$ where each element of $\mathbf{p_a}$ is multiplied with the basis vector in the corresponding row of $\mathbf{B}$ followed by element wise summation of the resulting scaled vectors. $\mathbf{B}$ also included $\mathbf{I_t}$ along with a vector of all ones to account for simple affine illumination changes as in GB ILM (Sec. 5.3.1) so that this method can be regarded as a generalization of the latter. As it requires an off line training stage to compute $\mathbf{B}$, however, it has not been considered here. An on line variant of this approach can also be formulated by constructing $\mathbf{B}$ incrementally from tracked patches [99] but this has been deferred to future extensions of this work.

Within the framework proposed in this thesis, ILM is posited as a specific case of a parameterized AM though this also happens to be the only form of parameterization in literature that the author is aware of. As a result, the presence of ILM within the AM is completely transparent to the SM. Further, since the replacement of $\mathbf{I_t}$ with $\mathbf{g}$ is equally applicable to all AMs, ILM is conceptualized as an independent module that can be combined with any AM, though this study only considers combinations with SSD as these are the only ones implemented in MTF yet. As mentioned in Sec. 4.1, the breakdown of GD based trackers into the 3 modules using chain rule requires the AM to compute terms involving f and $\mathbf{I_t}$ like $\partial f/\partial \mathbf{I_t}$ and $\partial^2 f/\partial {\mathbf{I_t}}^2$ which in turn are used to compute $\partial f/\partial \mathbf{p}$ and $\partial^2 f/\partial \mathbf{p}^2$, also within the AM. When the AM contains an ILM, these terms can be broken down further using chain rule as:

$$\frac{\partial f}{\partial \mathbf{I_t}} = \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{I_t}} \tag{5.103}$$

$$\frac{\partial^2 f}{\partial {\mathbf{I_t}}^2} = \frac{\partial \mathbf{g}}{\partial \mathbf{I_t}}^T \frac{\partial^2 f}{\partial \mathbf{g}^2} \frac{\partial \mathbf{g}}{\partial \mathbf{I_t}} + \frac{\partial f}{\partial \mathbf{g}} \frac{\partial^2 \mathbf{g}}{\partial {\mathbf{I_t}}^2} \tag{5.104}$$

$$\frac{\partial f}{\partial \mathbf{p_a}} = \frac{\partial f}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{p_a}} \tag{5.105}$$

$$\frac{\partial^2 f}{\partial {\mathbf{p_a}}^2} = \frac{\partial \mathbf{g}}{\partial \mathbf{p_a}}^T \frac{\partial^2 f}{\partial \mathbf{g}^2} \frac{\partial \mathbf{g}}{\partial \mathbf{p_a}} + \frac{\partial f}{\partial \mathbf{g}} \frac{\partial^2 \mathbf{g}}{\partial {\mathbf{p_a}}^2} \tag{5.106}$$

In addition to these expressions, the joint optimization of $\mathbf{p_a}$ and $\mathbf{p_s}$ also requires $\frac{\partial^2 f}{\partial \mathbf{p_s} \partial \mathbf{p_a}}$ to be computed as part of the $(S+A) \times (S+A)$ Hessian $\frac{\partial^2 f}{\partial \mathbf{p}^2}$. This is given as:

$$\frac{\partial^2 f}{\partial \mathbf{p_a} \partial \mathbf{p_s}} = \frac{\partial}{\partial \mathbf{p_s}} \left(\frac{\partial f}{\partial \mathbf{p_a}} \right) = \frac{\partial^2 f}{\partial \mathbf{I_t} \partial \mathbf{p_a}} \frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}} \tag{5.107}$$

with

$$\frac{\partial^2 f}{\partial \mathbf{I_t} \partial \mathbf{p_a}} = \frac{\partial \mathbf{g}}{\partial \mathbf{I_t}}^T \frac{\partial^2 f}{\partial \mathbf{g}^2} \frac{\partial \mathbf{g}}{\partial \mathbf{p_a}} + \frac{\partial f}{\partial \mathbf{g}} \frac{\partial^2 \mathbf{g}}{\partial \mathbf{I_t} \partial \mathbf{p_a}} \tag{5.108}$$

The terms in Eqs. 5.103 - 5.106 and 5.108 involving $\mathbf{g}$ and $\mathbf{p_a}$ are computed by the ILM within the proposed framework and the relevant expressions for the three ILMs considered in this work are thus presented next. It may be mentioned here that the second terms in Eqs. 5.104, 5.106 and 5.108 need to be discarded to use these derivatives with $\hat{\mathbf{H}}_{self}$.

5.3.1 Gain & Bias (GB)

This is the simplest ILM [65, 70, 138, 139, 215] that, in addition to considering the surface as perfectly Lambertian, makes the further assumption that the entire surface has constant reflectance so that $\alpha_k = \alpha \ \forall k$. As a result, $\mathbf{p_a} = [\alpha, \beta]$ and $A = 2$ for this ILM. The multiplicative factor α is called gain and the additive factor β is the bias. GB is thus formulated as:

$$\mathbf{g_{gb}} = (1 + \alpha)\mathbf{I_t} + \beta \tag{5.109}$$

Due to its simple formulation, derivatives of $\mathbf{g_{gb}}$ w.r.t. $\mathbf{p_a}$ and $\mathbf{I_t}$ are straightforward to compute and given as:

$$\frac{\partial \mathbf{g_{gb}}}{\partial \mathbf{I_t}} = (1 + \alpha)\mathbb{I}_N \tag{5.110}$$

$$\frac{\partial \mathbf{g_{gb}}}{\partial \alpha} = \mathbf{I_t}, \frac{\partial \mathbf{g_{gb}}}{\partial \beta} = 1 \tag{5.111}$$

$$\frac{\partial^2 \mathbf{g_{gb}}}{\partial \mathbf{I_t}^2} = 0 \tag{5.112}$$

$$\frac{\partial^2 \mathbf{g_{gb}}}{\partial \mathbf{p_a}^2} = 0 \tag{5.113}$$

$$\frac{\partial^2 \mathbf{g}}{\partial \mathbf{I_t} \partial \alpha} = \mathbb{I}_N, \frac{\partial^2 \mathbf{g}}{\partial \mathbf{I_t} \partial \beta} = 0 \tag{5.114}$$

5.3.2 Piecewise Gain & Bias (PGB)

This was introduced [84] as an improvement over GB based on the observation that the constant reflectance assumption made in the latter rarely holds true in practice over the entire object surface though it is approximately valid over local subregions of sufficiently small sizes. Therefore, a better approximation to Eq. 5.102 would be to have piecewise constant gain, i.e. the tracked region is divided into a set of non overlapping sub regions and a constant gain is used for each. The bias remains fixed for the entire region. Extensions to PGB for use with color images [77, 85] and omni directional cameras [140] have also been reported in literature though not considered here.

The size of $\mathbf{p_a}$ here depends on the number of subregions used. For instance, if a 3×3 grid of subregions is used then $A = 3^2 + 1 = 10$. Similarly, $A = 26$ for a 5×5 grid of subregions. More sub regions than the latter are typically not used in practice as this would require too many parameters to be estimated and a 5×5 grid is also enough to give sufficiently small sub regions of size 10×10 with the commonly used sampling resolution of 50×50.

Assuming an $R_x \times R_y$ grid of subregions, $\mathbf{p_a} = [\alpha_1, ..., \alpha_R, \beta] \in \mathbb{R}^{R+1}$ with $R = R_x R_y$. The size of each subregion is denoted as $\Delta u \times \Delta v = \frac{N_x}{R_x} \times \frac{N_y}{R_y}$. PGB is then given as:

$$\mathbf{g_{pgb}} = (1 + \tilde{\mathbf{I}}_{\boldsymbol{\alpha}}) \odot \mathbf{I_t} + \beta \tag{5.115}$$

where $\tilde{\mathbf{I}}_{\boldsymbol{\alpha}} \in \mathbb{R}^N$ and its k^{th} element is given as:

$$\tilde{I}_{k\alpha} = \alpha_j \tag{5.116}$$

if j^{th} subregion is the one that contains $\mathbf{x}_{kt}$, that is,

$$(j_x - 1)\Delta u < k_x \leq j_x \Delta u \tag{5.117}$$

and

$$(j_y - 1)\Delta v < k_y \leq j_y \Delta v \tag{5.118}$$

with $1 \leq j = j_x + (j_y - 1)R_x \leq R, 1 \leq j_x \leq R_x, 1 \leq j_y \leq R_y$ and $k = k_x + (k_y - 1)N_x$ as in Eq. 5.90.

The derivatives of $\mathbf{g_{pgb}}$ are similar to those for $\mathbf{g_{gb}}$ except that the terms involving $\boldsymbol{\alpha} = [\alpha_1, ..., \alpha_R]$ have to be computed in a piecewise manner by applying Eqs. 5.110 - 5.113 separately for each subregion. For instance, the k^{th} element of $\frac{\partial \mathbf{g_{pgb}}}{\partial \alpha_j}$, $\forall j \in [1, R]$, is given as:

$$\frac{\partial \mathbf{g_{pgb}}(k)}{\partial \alpha_j} = \begin{cases} I_{kt} & \text{if } \mathbf{x}_{kt} \in j^{th} \text{ subregion} \\ 0 & \text{otherwise} \end{cases} \tag{5.119}$$

$\frac{\partial^2 \mathbf{g_{pgb}}}{\partial \mathbf{I_t}^2}$ and $\frac{\partial^2 \mathbf{g_{pgb}}}{\partial \mathbf{p_a}^2}$ are null too while the k^{th} elements of the principal diagonals of $\frac{\partial \mathbf{g_{pgb}}}{\partial \mathbf{I_t}}$ and $\frac{\partial^2 \mathbf{g}}{\partial \mathbf{I_t} \partial \alpha_j}$ are respectively given as:

$$\frac{\partial \mathbf{g_{pgb}}(k)}{\partial I_{kt}} = 1 + \tilde{I}_{k\alpha} \tag{5.120}$$

and

$$\frac{\partial^2 \mathbf{g_{pgb}}}{\partial I_{kt} \partial \alpha_j} = \begin{cases} 1 & \text{if } \mathbf{x}_{kt} \in j^{th} \text{ subregion} \\ 0 & \text{otherwise} \end{cases} \tag{5.121}$$

5.3.3 Radial Basis Function (RBF)

This was introduced along with PGB [84] and has been mentioned in all subsequent works [77, 85, 140] where the latter has been applied. It has, however, not been used for any experimental tests since PGB has sparse derivatives that are computationally less expensive to compute. RBF represents the variation of gain over the patch by a smooth surface rather than a set of discrete values. It can alternatively be seen as an extension of PGB where interpolation is used to vary the gain smoothly between subregions rather than changing it abruptly at their boundaries. In its original formulation, the RBF surface [216] was approximated by a thin plate spline (TPS) function with a grid of control points corresponding to the sub regions of PGB.

In this work, a slightly simplified version of this has been used where the gain at each pixel in the patch is computed as a weighted average of the gains at the control points that are chosen to be the centroids of the respective subregions. The weight assigned to the pixel for each control point is inversely proportional to the distance of the pixel from that point. Assuming that (c_{jx}, c_{jy}) denotes the j^{th} control point (or equivalently the centroid of the j^{th} subregion) with $c_{jx} = \frac{(2j_x - 1)\Delta u + 1}{2}$ and

$c_{jy} = \dfrac{(2j_y - 1)\Delta v + 1}{2}$, the weight of the k^{th} pixel w.r.t. the j^{th} control point is given as:

$$W_{jk} = \frac{1}{1 + (k_x - c_{jx})^2 + (k_y - c_{jy})^2} \tag{5.122}$$

and the overall gain at this pixel is computed as:

$$\tilde{I}_{k\alpha} = \frac{\sum_{j=1}^{R} W_{jk}\alpha_j}{\sum_{j=1}^{R} W_{jk}} \tag{5.123}$$

RBF is then formulated similar to PGB as:

$$\mathbf{g_{rbf}} = (1 + \tilde{\mathbf{I}}_{\boldsymbol{\alpha}}) \odot \mathbf{I_t} + \beta \tag{5.124}$$

The derivatives of RBF are similar to PGB too except that the gain now varies smoothly rather than in a piecewise manner.

$$\frac{\partial \mathbf{g_{rbf}}(k)}{\partial \alpha_j} = W_{jk} I_{kt} \tag{5.125}$$

$$\frac{\partial \mathbf{g_{rbf}}(k)}{\partial I_{kt}} = 1 + \tilde{I}_{k\alpha} \tag{5.126}$$

$$\frac{\partial^2 \mathbf{g_{rbf}}}{\partial I_{kt} \partial \alpha_j} = W_{jk} \tag{5.127}$$

with $\dfrac{\partial^2 \mathbf{g_{rbf}}}{\partial \mathbf{I_t}^2}$ and $\dfrac{\partial^2 \mathbf{g_{rbf}}}{\partial \mathbf{p_a}^2}$ being null too.

5.4 Summary

This chapter presented details of the various AMs considered in this study. These were divided into two categories - L2 and robust models. The former category included 5 AMs - SSD, SCV, RSCV, LSCV and ZNCC while the latter comprised 7 AMs - NCC, MI, CCRE, SSIM, SPSS, RIU and NGF. Expressions for the first and second order derivatives of these AMs were presented with detailed derivations provided where these have not been published elsewhere in literature. In addition, the concept of ILM was introduced as a simple way to add parameterization to these AMs to make them more robust to lighting variations, reflections and shadows. Three ILMs were described - GB, PGB and RBF.

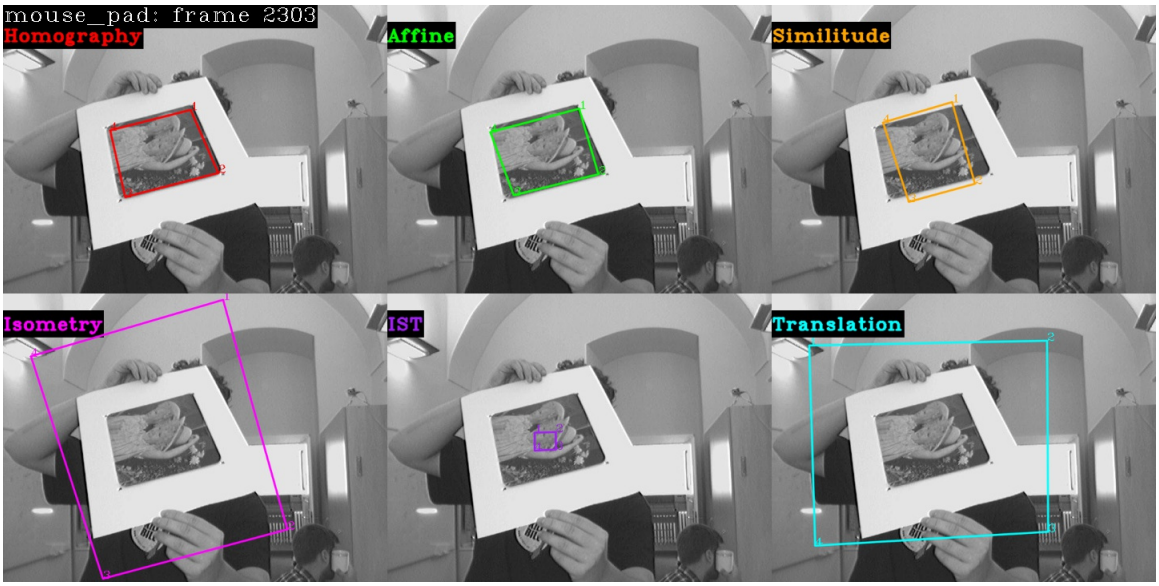
Chapter 6

State Space Models

SSM is the warping function $\mathbf{w}$ that represents the deformations that the tracked patch can undergo. It therefore defines the set of allowable image motions of the corresponding object and can be used to place constraints on the search space of $\mathbf{p_s}$ to make the optimization more robust or efficient. Besides the DOF of allowed motion, SSM also includes the actual parameterization of $\mathbf{w}$. For instance, homography using $\mathbb{SL}(3)$ [76, 1] Lie group parameterization is considered as a different SSM from the more conventional variant [16, 25] that uses actual entries of the corresponding matrix.

This work considers seven different SSMs, out of which five are from the standard hierarchy of geometrical transformations [184, 15] - translation, isometry, similitude, affine and homography and thus differ in their DOF. The remaining two SSMs, however, are alternative parameterizations of homography - $\mathbb{SL}(3)$ [76, 1] and corner based (using x,y coordinates of the bounding box corners) - and so have identical DOF. A related 3 DOF SSM with isotropic scaling and translation (IST) that is commonly used with OLTs is implemented in MTF too but not tested here due to the difficulty of generating reliable low DOF ground truth for this SSM (Fig. 6.1). A 4 DOF variant of IST with anisotropic scaling and translation (AST) has not been tested for the same reason. More complex SSMs have also been proposed in literature to handle non rigid or deformable objects including thin plate splines (TPS) [217, 96], basis splines [218] and quadric surfaces [219] of which the first two have already been partially implemented within MTF. However, these are not tested here either, both to limit the scope of this study and because the datasets used here only feature rigid object motion and so are ill suited to evaluate such SSMs.. Several extensions to $\mathbf{w}$ proposed in literature like incorporation of 3D pose [125] and camera parameters [133] are likewise excluded.

(a)



(b)

Figure 6.1: Two frames from a sequence in LinTrack dataset (Sec. 8.1) demonstrating the difference in the accuracy with which different SSMs can track an object undergoing complex deformations. Bounding boxes represent the low DOF ground truth generated for each SSM using least squares optimization (Sec. 9.3). Their corners are numbered clockwise from the top left to demonstrate rotation better. (a) Frame 1 showing the initial location where a tracker is initialized (b) Frame 2303 showing significant change in scale along with in-plane and out-of-plane rotations of the object. It can be seen that higher DOF SSMs can track the object pose more precisely. The issue encountered while generating low DOF ground truth for IST in the presence of in-plane rotation is also evident - scaling down the bounding box to an unreasonable extent corresponds to the least squares solution when the object undergoes significant rotation.

The main reason for using an SSM with higher DOF is to achieve more precise alignment with the target patch so that complex motions of the object like out of plane rotations can be tracked accurately. This follows from the fact that transforms that are higher up in the hierarchy [184, sec. 2.4] can better approximate the projective transformation process that captures the relative motion between the camera and the object in the 3D world into the 2D images. However, there are several issues with having to estimate more parameters:

- The search process becomes more likely to either diverge or end up in a local optimum, causing the tracker to be less stable and more likely to lose track. This is a well known phenomenon with GD based SMs [62] whose higher DOF variants are usually less robust.

- Iterative SMs take longer to converge, thus making the tracker slower. For GD based SMs, this is exacerbated by the usually higher cost of computing the gradients of $\mathbf{w}$.

- Generating good samples for stochastic SMs becomes more challenging due to complex dependencies between the different elements of $\mathbf{p_s}$. Also, more samples are needed to cover the search space.

It may be noted that this sub module differs from the other two in that it does not admit new methods in the conventional sense and may even be viewed as a part of the SM since the two are often closely intertwined in practical implementations. However, though the SSMs used in this work are limited to the standard hierarchy of geometric transformations, more complex models also exist even if their application domain is somewhat limited. It is also theoretically possible to impose novel constraints on the search space that can significantly decrease the convergence time while still producing sufficiently accurate results. The fact that such a constraint will be an important contribution in its own right justifies the use of SSM as a separate sub module within the proposed framework to motivate further research in this direction.

Brief descriptions for the SSMs considered here will be provided next with expressions for $\mathbf{w}$ and its first and second order derivatives that are needed for GD methods (Sec. 4.1). Before proceeding with these, however, some notation is introduced to make the subsequent formulations easier.

In its general form, $\mathbf{w}$ takes all N points in the sampling grid $\mathbf{x}$ as input and outputs the warped location of the entire grid, i.e. $\mathbf{w} : \mathbb{R}^{2\times N} \times \mathbb{R}^{S} \mapsto \mathbb{R}^{2\times N}$. This

implies that the warped location of any given grid point might depend on the locations of other points. However, all SSMs considered here are rigid transformations so that the geometric transform defined by $\mathbf{w}$ is identical for each point in $\mathbf{x}$ and independent of all other points. It follows that $\mathbf{w}$ and its derivatives can be completely specified by expressing these for a single point and $\mathbf{w}$ can thus be treated as a function of this point as its input and its warped location as its output, i.e. $\mathbf{w} : \mathbb{R}^{2\times 1} \times \mathbb{R}^{S} \mapsto \mathbb{R}^{2\times 1}$.

As this assumption greatly simplifies the relevant formulations, it is adopted in the following sections where expressions are provided for a single point $\mathbf{x}_k = [x_k, y_k]^T$. with the corresponding warped location also denoted as $\mathbf{x}'_k = [x'_k, y'_k]^T$ for brevity. Further, all of these transformations can be produced by multiplying $\mathbf{x}_k$, written in homogeneous coordinates [184], with a 3×3 matrix. This matrix is denoted as $\mathbf{G} : \mathbb{R}^{S} \mapsto \mathbb{R}^{3\times 3}$ so that $\mathbf{w}(\mathbf{x}_k, \mathbf{p_s}) \equiv \mathbf{h^{-1}}(\mathbf{G}(\mathbf{p_s})\mathbf{h}(\mathbf{x}_k))$ where $\mathbf{h} : \mathbb{R}^{2\times 1} \mapsto \mathbb{R}^{3\times 1}$ and $\mathbf{h^{-1}} : \mathbb{R}^{3\times 1} \mapsto \mathbb{R}^{2\times 1}$ map a pixel location from standard to homogeneous coordinates and vice versa :

$$\mathbf{h}([x, y]^T) = [x, y, 1]^T \tag{6.1}$$

$$\mathbf{h^{-1}}([x, y, z]^T) = [x/z, y/z]^T \tag{6.2}$$

For brevity, the verbose expression $\mathbf{h^{-1}}(\mathbf{G}(\mathbf{p_s})\mathbf{h}(\mathbf{x}_k))$ will be replaced with $\mathbf{G}\mathbf{x}_k$ to denote multiplication in homogeneous space when such notation does not create any semantic ambiguity. This notation will also be extended to deal with the entire grid $\mathbf{x}$ rather than a single point, i.e. $\mathbf{G}\mathbf{x}$ with $\mathbf{x} \in \mathbb{R}^{2\times N}$ will produce a $2 \times N$ warped grid where homogeneous multiplication has been performed for each point.

6.1 Translation

This is the simplest SSM that can only track 2 DOF motion of the image patch. Assuming $\mathbf{p_s} = [t_x, t_y]$, it is given as:

$$\mathbf{w_{trans}}(\mathbf{x}_k, \mathbf{p_s}) = \begin{bmatrix} x_k + t_x \\ y_k + t_y \end{bmatrix} \tag{6.3}$$

and

$$\mathbf{G_{trans}}(\mathbf{p_s}) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

so that t_x and t_y are the translations in the x and y direction. Due to its simple formulation, its derivatives are straightforward to compute:

$$\frac{\partial \mathbf{w_{trans}}}{\partial \mathbf{p_s}} = \frac{\partial \mathbf{w_{trans}}}{\partial \mathbf{x}_k} = \mathbb{I}_2 \tag{6.5}$$

$$\frac{\partial^2 \mathbf{w_{trans}}}{\partial \mathbf{p_s}^2} = \frac{\partial^2 \mathbf{w_{trans}}}{\partial \mathbf{x}_k^2} = 0 \tag{6.6}$$

6.2 Isometry

Also known as Euclidean or rigid transformation, isometry is a 3 DOF SSM that includes rotation in addition to translation. Denoting the angle of rotation in radians as θ, $\mathbf{p_s} = [t_x, t_y, \theta]$ and isometry transform is given as:

$$\mathbf{w_{iso}}(\mathbf{x}_k, \mathbf{p_s}) = \begin{bmatrix} x_k \cos\theta - y_k \sin\theta + t_x \\ x_k \sin\theta + y_k \cos\theta + t_y \end{bmatrix} \tag{6.7}$$

and

$$\mathbf{G_{iso}}(\mathbf{p_s}) = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.8}$$

Its derivatives are computed as:

$$\frac{\partial \mathbf{w_{iso}}}{\partial \mathbf{p_s}} = \begin{bmatrix} 1 & 0 & -x_k \sin\theta - y_k \cos\theta \\ 0 & 1 & x_k \cos\theta - y_k \sin\theta \end{bmatrix} \tag{6.9}$$

$$\frac{\partial \mathbf{w_{iso}}}{\partial \mathbf{x}_k} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{6.10}$$

$$\frac{\partial^2 \mathbf{w_{iso}}}{\partial t_x^2} = \frac{\partial^2 \mathbf{w_{iso}}}{\partial t_y^2} = 0, \frac{\partial^2 \mathbf{w_{iso}}}{\partial \theta^2} = \begin{bmatrix} 0 & 0 & -x_k \cos\theta + y_k \sin\theta \\ 0 & 0 & -x_k \sin\theta + y_k \cos\theta \end{bmatrix} \tag{6.11}$$

$$\frac{\partial^2 \mathbf{w_{sim}}}{\partial \mathbf{x}_k^2} = 0 \tag{6.12}$$

6.3 Similitude

This SSM extends isometry by providing isotropic scaling in addition to translation and rotation, thus being able to track 4 DOF motion. Though this can be parameterized using the rotation angle and the scaling factor directly, such parameterization would lead to more computationally expensive derivatives, similar to isometry, due to the non linearity of sine and cosine functions. Therefore, as in [15], the terms involving rotation and scaling are instead combined to express similitude as a linear function of $\mathbf{x}_k$ and $\mathbf{p_s} = [t_x, t_y, a, b]$:

$$\mathbf{w_{sim}}(\mathbf{x}_k, \mathbf{p_s}) = \begin{bmatrix} x_k(1+a) - y_k b + t_x \\ x_k b + y_k(1+a) + t_y \end{bmatrix} \tag{6.13}$$

and

$$\mathbf{G_{sim}}(\mathbf{p_s}) = \begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.14}$$

Its derivatives are given as:

$$\frac{\partial \mathbf{w_{sim}}}{\partial \mathbf{p_s}} = \begin{bmatrix} 1 & 0 & x_k & -y_k \\ 0 & 1 & y_k & x_k \end{bmatrix} \tag{6.15}$$

$$\frac{\partial \mathbf{w_{sim}}}{\partial \mathbf{x}_k} = \begin{bmatrix} 1+a & -b \\ b & 1+a \end{bmatrix} \tag{6.16}$$

$$\frac{\partial^2 \mathbf{w_{sim}}}{\partial {\mathbf{p_s}}^2} = \frac{\partial^2 \mathbf{w_{sim}}}{\partial \mathbf{x}_k^2} = 0 \tag{6.17}$$

6.4 Affine

This extends similitude further by adding two more DOFs - one is for anisotropic scaling and the other can either be seen as a skewing/shearing factor [62] or as the direction along which the scaling is carried out [184]. The latter parameterization has been popular for PF based affine trackers [147, 101, 100, 148] due to the relative ease of estimating the standard deviations required for the Gaussian samples to represent the expected object motion. In this work, however, only the direct parameterization of affine using actual entries of $\mathbf{G}$ is considered so that $\mathbf{w}$ is linear in $\mathbf{x}_k$ and $\mathbf{p_s}$ and its

derivatives are consequently easier to compute. Incidentally, affine is also the highest DOF SSM that can be expressed as a linear function of $\mathbf{x}_k$ and $\mathbf{p_s}$.

With $\mathbf{p_s} = [t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11}]$, affine transformation is given as:

$$\mathbf{w_{aff}}(\mathbf{x}_k, \mathbf{p_s}) = \begin{bmatrix} x_k(1 + a_{00}) + y_k a_{01} + t_x \\ x_k a_{10} + y_k(1 + a_{11}) + t_y \end{bmatrix} \tag{6.18}$$

and

$$\mathbf{G_{aff}}(\mathbf{p_s}) = \begin{bmatrix} 1 + a_{00} & a_{01} & t_x \\ a_{10} & 1 + a_{11} & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6.19}$$

Differentiating $\mathbf{w_{aff}}$ w.r.t. $\mathbf{p_s}$ and $\mathbf{x}_k$ gives:

$$\frac{\partial \mathbf{w_{aff}}}{\partial \mathbf{p_s}} = \begin{bmatrix} 1 & 0 & x_k & y_k & 0 & 0 \\ 0 & 1 & 0 & 0 & x_k & y_k \end{bmatrix} \tag{6.20}$$

$$\frac{\partial \mathbf{w_{aff}}}{\partial \mathbf{x}_k} = \begin{bmatrix} 1 + a_{00} & a_{01} \\ a_{10} & 1 + a_{11} \end{bmatrix} \tag{6.21}$$

$$\frac{\partial^2 \mathbf{w_{aff}}}{\partial {\mathbf{p_s}}^2} = \frac{\partial^2 \mathbf{w_{aff}}}{\partial \mathbf{x}_k^2} = 0 \tag{6.22}$$

6.5 Homography

This is the full 8 DOF SSM that can accurately represent the deformations produced in the image patch corresponding to a planar object by relative motion between the object and the camera. Since a camera uses the perspective projection process to represent the 3D world in 2D images, this SSM is also known in literature as planar projective transformation [15, 184]. As mentioned before, there are several ways to parameterize this SSM but this section considers the simplest one where actual entries of $\mathbf{G}$ are used. Two other parameterizations are dealt with in the subsequent sections.

Homography is a linear mapping in homogeneous coordinates but not so in standard ones since the conversion from the former to the latter (Eq. 6.2) is non linear. Assuming $\mathbf{p_s} = [p_1, p_2, ..., p_8]$ and $D = p_7 x_k + p_8 y_k + 1$, this transformation is given as:

$$\mathbf{w_{hom}}(\mathbf{x}_k, \mathbf{p_s}) = \frac{1}{D} \begin{bmatrix} (1 + p_1)x_k + p_2 y_k + p_3 \\ p_4 x_k + (1 + p_5)y_k + p_6 \end{bmatrix} \tag{6.23}$$

and

$$\mathbf{G_{hom}}(\mathbf{p_s}) = \begin{bmatrix} 1+p_1 & p_2 & p_3 \\ p_4 & 1+p_5 & p_6 \\ p_7 & p_8 & 1 \end{bmatrix} \tag{6.24}$$

First and second order derivatives of $\mathbf{w_{hom}}$ are computed as:

$$\frac{\partial \mathbf{w_{hom}}}{\partial \mathbf{p_s}} = \frac{1}{D} \begin{bmatrix} x_k & y_k & 1 & 0 & 0 & 0 & -x'_k x_k & -x'_k y_k \\ 0 & 0 & 0 & x_k & y_k & 1 & -y'_k x_k & -y'_k y_k \end{bmatrix} \tag{6.25}$$

$$\frac{\partial \mathbf{w_{hom}}}{\partial \mathbf{x}_k} = \frac{1}{D} \begin{bmatrix} 1+p_1-p_7x'_k & p_2-p_8x'_k \\ p_4-p_7x'_y & 1+p_5-p_8x'_y \end{bmatrix} \tag{6.26}$$

$$\frac{\partial^2 \mathbf{w_{hom}}(1)}{\partial {\mathbf{p_s}}^2} = \frac{1}{D^2} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -x_k^2 & -x_k y_k \\ 0 & 0 & 0 & 0 & 0 & 0 & -x_k y_k & -y_k^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x_k & -y_k \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -x_k^2 & -x_k y_k & -x_k & 0 & 0 & 0 & 2x'_k x_k^2 & 2x'_k x_k y_k \\ -x_k y_k & -y_k^2 & -y_k & 0 & 0 & 0 & 2x'_k x_k y_k & 2x'_k y_k^2 \end{bmatrix}$$

$$\frac{\partial^2 \mathbf{w_{hom}}(2)}{\partial {\mathbf{p_s}}^2} = \frac{1}{D^2} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x_k^2 & -x_k y_k \\ 0 & 0 & 0 & 0 & 0 & 0 & -x_k y_k & -y_k^2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -x_k & -y_k \\ 0 & 0 & 0 & -x_k^2 & -x_k y_k & -x_k & 2y'_k x_k^2 & 2y'_k x_k y_k \\ 0 & 0 & 0 & -x_k y_k & -y_k^2 & -y_k & 2y'_k x_k y_k & 2y'_k y_k^2 \end{bmatrix} \tag{6.27}$$

$$\frac{\partial^2 \mathbf{w_{hom}}(1)}{\partial \mathbf{x}_k^2} = -\frac{1}{D^2} \begin{bmatrix} 2p_7(1+p_1-p_7x'_k) & p_8(1+p_1-p_7x'_k)+p_7(p_2-p_8x'_k) \\ p_8(1+p_1-p_7x'_k)+p_7(p_2-p_8x'_k) & 2p_8(p_2-p_8x'_k) \end{bmatrix}$$

$$\frac{\partial^2 \mathbf{w_{hom}}(2)}{\partial \mathbf{x}_k^2} = -\frac{1}{D^2} \begin{bmatrix} 2p_7(p_4-p_7y'_k) & p_8(p_4-p_7y'_k)+p_7(1+p_5-p_8y'_k) \\ p_8(p_4-p_7y'_k)+p_7(1+p_5-p_8y'_k) & 2p_8(1+p_5-p_8y'_k) \end{bmatrix} \tag{6.28}$$

where it may be recalled that $\mathbf{w_{hom}}(\mathbf{x}_k, \mathbf{p_s})$ is denoted as $\mathbf{x}'_k = [x'_k, y'_k]^T$ for brevity.

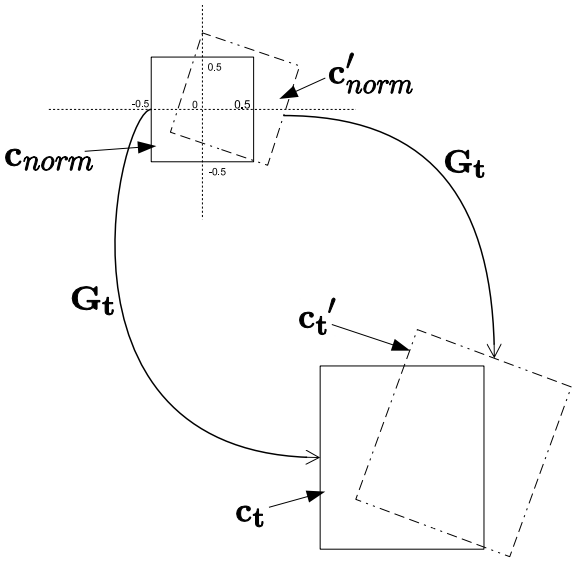


Figure 6.2: Producing perturbed samples for homography

6.5.1 Stochastic Sample Generation

As mentioned before, the author is not aware of any general method for estimating the Gaussian distribution needed to generate stochastic samples for high DOF SSMs that can represent any given kind of motion well. The matrix based parameterization used by this SSM makes this task even more challenging due to the large differences in the amount of warping - as measured by the Euclidean distance between the corners of the original and warped bounding boxes - produced by similar numerical perturbations in different components of $\mathbf{p_s}$. There also exist complex interdependencies between these components so that the perturbation in any given component needed to produce a certain degree of change in the bounding box depends on the actual values of the remaining components.

A simple way to circumvent this issue is to perturb the bounding box itself and then estimate the warp matrix $\mathbf{G_{hom}}$ (and thus the corresponding $\mathbf{p_s}$) needed to produce this warped bounding box using the direct linear transform (DLT) algorithm [184, sec. 4.1]. This is the method that has been adopted in this work to generate samples for this SSM with both NN and PF. Let $\mathbf{c_t}$ denote the bounding box corners corresponding to $\mathbf{x_t}$. Since the range of values in $\mathbf{c_t}$ may vary widely depending on the size and location of the object patch, the perturbation is instead applied to the normalized unit square centered at the origin, denoted as $\mathbf{c}_{norm}$. That is,

$\mathbf{c}_{norm} = \begin{bmatrix} -0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & -0.5 & 0.5 & 0.5 \end{bmatrix}$.

The perturbation of $\mathbf{c}_{norm}$ is carried out in two steps. First, the x,y coordinates of all its corners are perturbed independently using 8 values drawn from a single Gaussian distribution with standard deviation σ_d. Second, two more values are drawn from another distribution with standard deviation σ_t and added to the coordinates of all the corners to produce a consistent translation of the entire box. This second step is needed to better represent realistic scenarios where translation is the most common type of motion that the image patch undergoes. Denoting the corners thus produced as $\mathbf{c}'_{norm}$ and letting $\mathbf{G_t}$ be the warp matrix, computed using DLT, that can transform $\mathbf{c}_{norm}$ to $\mathbf{c_t}$, the perturbed version of $\mathbf{c_t}$ is given as:

$$\mathbf{c_t}' = \mathbf{G_t}\mathbf{c}'_{norm} \tag{6.29}$$

Fig. 6.2 gives a pictorial representation of this sampling process. Applying the perturbation to $\mathbf{c}_{norm}$ rather than $\mathbf{c_t}$ allows the same pair of distributions to produce more consistent warped samples with widely varying object sizes and locations, thus reducing the need to fine tune these for each tracking scenario. Some adjustments may still help, of course, if prior information is available about the expected motion.

6.6 SL3

This SSM utilizes an alternative parameterization of homography by choosing its transformation matrix $\mathbf{G}$ to belong to the special linear group of degree 3 or $\mathbb{SL}(3)$ [1, 73]. This is a Lie group [220] that includes all 3×3 matrices with unit determinant. Placing this constraint on $\mathbf{G}$ is one way of limiting its DOF to 8 - a necessary restriction since it is only defined up to a scaling factor and one that was imposed in the previous section by fixing its bottom right element to 1. Both resultant sets of matrices can represent all projective transformations and are in fact equivalent in the sense that a bijective mapping exists between the two. Any matrix in the latter set can be converted to the former by dividing it by the cube root of its determinant while the reverse mapping can be carried out by dividing the $\mathbb{SL}(3)$ matrix by its bottom right element.

Each Lie group has an associated Lie algebra [221, 220] that corresponds to the differential or tangential space at its identity element. Elements belonging to the Lie algebra can be mapped to the associated Lie group using the matrix exponential

transform (Eq. 6.30) while the reverse mapping can be performed using the matrix logarithm (Eq. 6.31).

$$\exp(\mathbf{X}) = \sum_{m=0}^{\infty} \frac{\mathbf{X}^m}{m!} \tag{6.30}$$

$$\log(\mathbf{X}) = \sum_{m=1}^{\infty} (-1)^{m+1} \frac{(\mathbf{X} - \mathbb{I})^m}{m} \tag{6.31}$$

The Lie algebra that corresponds to $\mathbb{SL}(3)$ is called $\mathfrak{sl}(3)$ and is composed of all 3×3 matrices with null trace. Any matrix in $\mathfrak{sl}(3)$ can be represented by the linear combination of matrices in its basis set. As a result, SL3 parameterizes a projective transformation by the coefficients of these basis matrices that will produce the $\mathfrak{sl}(3)$ matrix corresponding to $\mathbf{G}$ that represents this transformation. Denoting these basis matrices as $\mathbf{B} = [\mathbf{B_1}, \mathbf{B_2}, ..., \mathbf{B_8}]$ and assuming $\mathbf{p_s} = [p_1, ..., p_8]$, the transformation matrix is thus given as:

$$\mathbf{G_{sl3}}(\mathbf{p_s}) = \begin{bmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{bmatrix} = \exp\left(\sum_{j=1}^{8} p_j \mathbf{B_j}\right) \tag{6.32}$$

and the corresponding warping function becomes:

$$\mathbf{w_{sl3}}(\mathbf{x}_k, \mathbf{p_s}) = \mathbf{G_{sl3}}\mathbf{x}_k = \frac{1}{D} \begin{bmatrix} g_{00}x_k + g_{01}y_k + g_{02} \\ g_{10}x_k + g_{11}y_k + g_{12} \end{bmatrix} \tag{6.33}$$

where $D = g_{20}x_k + g_{21}y_k + g_{22}$. Following are the basis matrices used here [1]:

$$\mathbf{B_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B_2} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \mathbf{B_3} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B_4} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$\mathbf{B_5} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B_6} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{B_7} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \mathbf{B_8} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{6.34}$$

This is the only SSM that has been used with ESM in literature [73, 76] since the special properties of the exponential map are needed to theoretically justify the second order convergence of this SM. It has also been used for PF [1] since, unlike the parameterization in the previous section, individual elements of $\mathbf{p_s}$ here can be

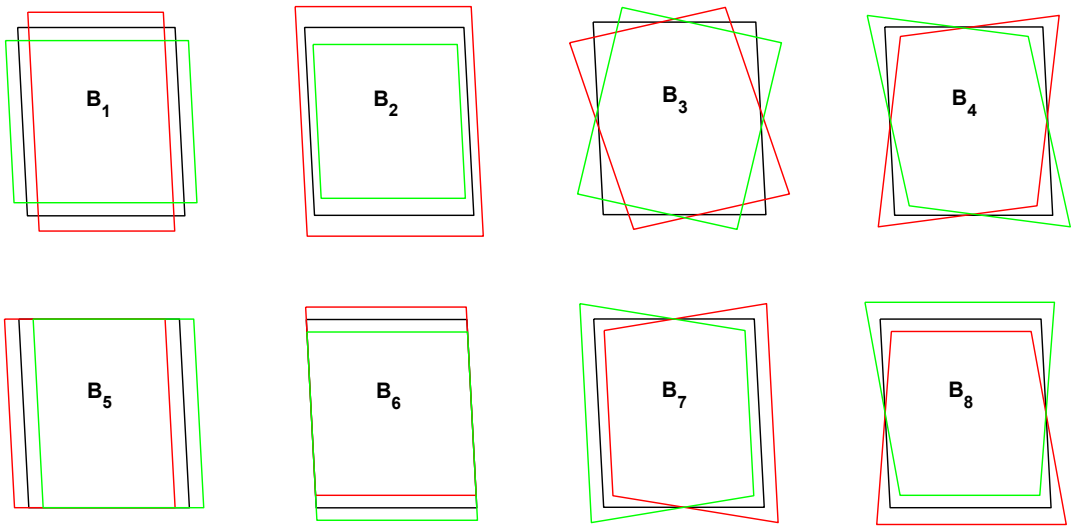


Figure 6.3: Transformations produced by SL3 corresponding to different $\mathfrak{sl}(3)$ basis matrices. The original bounding box is in black while the warped boxes produced by small positive and negative coefficients applied to the basis matrix are in red and green respectively. This figure is adapted from [1].

associated with specific deformations of the patch (Fig. 6.3). This in turn makes it easier to estimate Gaussian parameters needed to generate random samples that can represent the expected motion well. In addition, the distribution of SL3 parameters corresponding to projective transformations generated by random camera placements is much closer to the expected normal distribution [222] than that of the direct homography used in the previous section. As a result, direct perturbations to $\mathbf{p_s}$ using Gaussian parameters provided in [1] have also been used in this work to generate 8 DOF samples for PF as an alternative to corner based sampling described in Sec. 6.5.1. This method could not be used for NN, however, as the parameters in [1] are fine tuned for first order auto regression dynamic model that can only be applied to PF.

The first order derivatives of $\mathbf{w_{sl3}}$ are given as:

$$\frac{\partial \mathbf{w_{sl3}}}{\partial \mathbf{p_s}} = \frac{\partial \mathbf{w_{sl3}}}{\partial \mathbf{G_{sl3}}} \frac{\partial \mathbf{G_{sl3}}}{\partial \mathbf{p_s}} \tag{6.35}$$

where, after rearranging $\mathbf{G_{sl3}}$ into a 9×1 vector by flattening it in row major order,

$$\frac{\partial \mathbf{w_{sl3}}}{\partial \mathbf{G_{sl3}}} = \frac{1}{D} \begin{bmatrix} x_k & y_k & 1 & 0 & 0 & 0 & -x'_k x_k & -x'_k y_k & -x'_k \\ 0 & 0 & 0 & x_k & y_k & 1 & -y'_k x_k & -y'_k y_k & -y'_k \end{bmatrix} \tag{6.36}$$

and

$$\frac{\partial \mathbf{G_{sl3}}}{\partial \mathbf{p_s}} = \frac{\partial \exp(\mathbf{X})}{\partial \mathbf{X}} \frac{\partial \mathbf{X}}{\partial \mathbf{p_s}} \tag{6.37}$$

where $\mathbf{X} = \sum_{j=1}^{8}(p_j \mathbf{B_j}) \in \mathfrak{sl}(3)$, $\dfrac{\partial \mathbf{X}}{\partial \mathbf{p_s}}$ is a 9×8 matrix whose j^{th} column is formed by flattening $\mathbf{B_j}$ in row major order and $\dfrac{\partial \exp(\mathbf{X})}{\partial \mathbf{X}}$ is the derivative of the exponential map given as [223]:

$$\frac{\partial \exp(\mathbf{X})}{\partial \mathbf{X}} = \exp(\mathbf{X}) \left(\frac{\mathbb{I} - \exp(-\mathrm{ad}_{\mathbf{X}})}{\mathrm{ad}_{\mathbf{X}}} \right) = \exp(\mathbf{X}) \left(\sum_{j=0}^{\infty} \frac{(-1)^j}{(j+1)!} (\mathrm{ad}_{\mathbf{X}})^j \right) \tag{6.38}$$

where $\mathrm{ad}_{\mathbf{X}}$ is the adjoint action of the Lie algebra on itself. It is worth noting that, when evaluated at $\mathbf{p_s} = 0$, as needed for the compositional formulations of LK (Sec. 4.1), $\mathbf{X}$ becomes null and Eq. 6.38 evaluates to identity, thus eliminating the significant computational expense of evaluating it explicitly. This makes SL3 more suited to compositional formulations. It is also evident from Eqs. 6.37 and 6.38 that the second order derivative of $\mathbf{w_{sl3}}$ w.r.t. $\mathbf{p_s}$ will be very complex to express and compute and would involve evaluating the Hessian of the exponential map for which the author has been unable to find any expression in literature. It is therefore omitted here especially as it is not needed for computing $\hat{\mathbf{H}}_{self}$ for most of the AMs.

The derivatives of $\mathbf{w_{sl3}}$ w.r.t. $\mathbf{x}_k$, on the other hand, are much simpler to compute as they can be expressed directly in terms of the entries of $\mathbf{G_{sl3}}$ and thus follow trivially from Eqs. 6.26 and 6.28:

$$\frac{\partial \mathbf{w_{sl3}}}{\partial \mathbf{x}_k} = \frac{1}{D} \begin{bmatrix} g_{00} - g_{20}x'_k & g_{01} - g_{21}x'_k \\ g_{10} - g_{20}x'_y & g_{11} - g_{21}x'_y \end{bmatrix} \tag{6.39}$$

$$\begin{aligned} \frac{\partial^2 \mathbf{w_{sl3}}(1)}{\partial \mathbf{x}_k^2} &= -\frac{1}{D^2} \begin{bmatrix} 2g_{20}(g_{00} - g_{20}x'_k) & g_{21}(g_{00} - g_{20}x'_k) + g_{20}(g_{01} - g_{21}x'_k) \\ g_{21}(g_{00} - g_{20}x'_k) + g_{20}(g_{01} - g_{21}x'_k) & 2g_{21}(p_2 - g_{21}x'_k) \end{bmatrix} \\ \frac{\partial^2 \mathbf{w_{sl3}}(2)}{\partial \mathbf{x}_k^2} &= -\frac{1}{D^2} \begin{bmatrix} 2g_{20}(g_{10} - g_{20}y'_k) & g_{21}(g_{10} - g_{20}y'_k) + g_{20}(g_{11} - g_{21}y'_k) \\ g_{21}(g_{10} - g_{20}y'_k) + g_{20}(g_{11} - g_{21}y'_k) & 2g_{21}(g_{11} - g_{21}y'_k) \end{bmatrix} \end{aligned} \tag{6.40}$$

6.7 Corner Based Homography (CBH)

This SSM parameterizes homography through displacements in the x, y coordinates of the four corners of the bounding box of $\mathbf{x}$. Let these corners be denoted as $\mathbf{c} =$

$[\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4] \in \mathbb{R}^{2\times4}$ and a change therein as $\Delta\mathbf{c} = [\Delta\mathbf{c}_1, \Delta\mathbf{c}_2, \Delta\mathbf{c}_3, \Delta\mathbf{c}_4] \in \mathbb{R}^{2\times4}$ with $\mathbf{c}_j = [c_{jx}, c_{jy}]^T$ and $\Delta\mathbf{c}_j = [\Delta c_{jx}, \Delta c_{jy}]^T$. Further, let $\mathbf{G_{dlt}}(\mathbf{c}_{in}, \mathbf{c}_{out}) : \mathbb{R}^{2\times4} \times \mathbb{R}^{2\times4} \mapsto \mathbb{R}^{3\times3}$ refer to a function that uses least squares optimization to produce a 3×3 warp matrix that can be multiplied to corners $\mathbf{c}_{in}$ to generate warped corners whose Euclidean distance from $\mathbf{c}_{out}$ is minimized.

$$\mathbf{G_{dlt}}(\mathbf{c}_{in}, \mathbf{c}_{out}) = \underset{\mathbf{G}}{\operatorname{argmin}} \|\mathbf{G}\mathbf{c}_{in} - \mathbf{c}_{out}\|^2 \tag{6.41}$$

The parameter vector then becomes $\mathbf{p_s} = [\Delta c_{1x}, \Delta c_{1y}, ..., \Delta c_{4x}, \Delta c_{4y}]$ and the corresponding transformation matrix is given as:

$$\mathbf{G_{cbh}}(\mathbf{p_s}) = \begin{bmatrix} g_{00} & g_{01} & g_{02} \\ g_{10} & g_{11} & g_{12} \\ g_{20} & g_{21} & g_{22} \end{bmatrix} = \mathbf{G_{dlt}}(\mathbf{c}, \mathbf{c} + \Delta\mathbf{c}) \tag{6.42}$$

The warping function, as in the previous section, becomes:

$$\mathbf{w_{cbh}}(\mathbf{x}_k, \mathbf{p_s}) = \mathbf{G_{cbh}}\mathbf{x}_k = \frac{1}{D}\begin{bmatrix} g_{00}x_k + g_{01}y_k + g_{02} \\ g_{10}x_k + g_{11}y_k + g_{12} \end{bmatrix} \tag{6.43}$$

where $D = g_{20}x_k + g_{21}y_k + g_{22}$. The least squares problem in Eq. 6.41 is solved using the DLT algorithm [184, sec. 4.1] which involves performing the singular value decomposition of an 8×9 matrix constructed from $\mathbf{c}_{in}$ and $\mathbf{c}_{out}$. Since this process is not differentiable, the derivative of $\mathbf{w_{cbh}}$ w.r.t. $\mathbf{p_s}$ has to be computed numerically. Let $\mathbf{c}^{(i,j)+}$ and $\mathbf{c}^{(i,j)-}$ denote slightly displaced versions of $\mathbf{c}$, where the $(i,j)^{th}$ element thereof, for $1 \leq i \leq 2, 1 \leq j \leq 4$, has a small positive and negative displacement ϵ respectively. For instance, $\mathbf{c}^{(1,2)+} = \begin{bmatrix} c_{1x} & c_{2x}+\epsilon & c_{3x} & c_{4x} \\ c_{1y} & c_{2y} & c_{3y} & c_{4y} \end{bmatrix}$ and $\mathbf{c}^{(2,3)-} = \begin{bmatrix} c_{1x} & c_{2x} & c_{3x} & c_{4x} \\ c_{1y} & c_{2y} & c_{3y}-\epsilon & c_{4y} \end{bmatrix}$. Then, for $r = 2(p-1)+q$ and $t = 2(u-1)+v$, with $1 \leq r, t \leq 8, 1 \leq p, u \leq 2, 1 \leq q, v \leq 4$, the central finite difference method [149, sec. 8.1] gives:

$$\frac{\partial\mathbf{w_{cbh}}}{\partial p_{rs}} = \frac{\mathbf{G_{dlt}}(\mathbf{c}, \mathbf{c}^{(p,q)+})\mathbf{x}_k - \mathbf{G_{dlt}}(\mathbf{c}, \mathbf{c}^{(p,q)-})\mathbf{x}_k}{2\epsilon} \tag{6.44}$$

$$\begin{aligned} \frac{\partial^2\mathbf{w_{cbh}}}{\partial p_{ts}\partial p_{rs}} = \frac{1}{4\epsilon^2}\Big\{ &\big(\mathbf{G_{dlt}}((\mathbf{c}^{(i,j)+})^{(u,v)+})\mathbf{x}_k + \mathbf{G_{dlt}}((\mathbf{c}^{(i,j)-})^{(u,v)-})\mathbf{x}_k\big) \\ &- \big(\mathbf{G_{dlt}}((\mathbf{c}^{(i,j)+})^{(u,v)-})\mathbf{x}_k + \mathbf{G_{dlt}}((\mathbf{c}^{(i,j)-})^{(u,v)+})\mathbf{x}_k\big)\Big\} \end{aligned} \tag{6.45}$$

where $\mathbf{G_{dlt}}(\mathbf{c}, \mathbf{c}^{(i,j)+})$ has been denoted as $\mathbf{G_{dlt}}(\mathbf{c}^{(i,j)+})$ for brevity.

As for SL3, the derivatives w.r.t. $\mathbf{x}_k$ can be expressed directly in terms of the entries in $\mathbf{G_{cbh}}$:

$$\frac{\partial \mathbf{w_{cbh}}}{\partial \mathbf{x}_k} = \frac{1}{D}\begin{bmatrix} g_{00} - g_{20}x'_k & g_{01} - g_{21}x'_k \\ g_{10} - g_{20}x'_y & g_{11} - g_{21}x'_y \end{bmatrix} \tag{6.46}$$

$$\begin{aligned} \frac{\partial^2 \mathbf{w_{cbh}}(1)}{\partial \mathbf{x}_k^2} &= -\frac{1}{D^2}\begin{bmatrix} 2g_{20}(g_{00} - g_{20}x'_k) & g_{21}(g_{00} - g_{20}x'_k) + g_{20}(g_{01} - g_{21}x'_k) \\ g_{21}(g_{00} - g_{20}x'_k) + g_{20}(g_{01} - g_{21}x'_k) & 2g_{21}(p_2 - g_{21}x'_k) \end{bmatrix} \\ \frac{\partial^2 \mathbf{w_{cbh}}(2)}{\partial \mathbf{x}_k^2} &= -\frac{1}{D^2}\begin{bmatrix} 2g_{20}(g_{10} - g_{20}y'_k) & g_{21}(g_{10} - g_{20}y'_k) + g_{20}(g_{11} - g_{21}y'_k) \\ g_{21}(g_{10} - g_{20}y'_k) + g_{20}(g_{11} - g_{21}y'_k) & 2g_{21}(g_{11} - g_{21}y'_k) \end{bmatrix} \end{aligned} \tag{6.47}$$

6.8 Summary

This chapter presented details of the seven SSMs considered in this study. Five of these were subsets of the projective transformation - translation, isometry, similitude, affine and homography. The remaining two - SL3 and CHB - were alternative parameterizations of homography. Expressions were provided for first and second order derivatives of the warping functions of these SSMs w.r.t. both $\mathbf{p_s}$ and $\mathbf{x}$.

Chapter 7

System Design

This chapter presents details about the system design of MTF. A broad overview of how the different modules in MTF are organized and connected to each other is provided in Sec. 7.1. This is followed by descriptions of the two abstract classes corresponding to AM and SSM in Sec. 7.2 and 7.3 respectively. Sec. 7.4 presents pseudo codes for implementing several SMs as instances of using the functionality provided by these two classes to perform the optimization in different ways. Finally, Sec. 7.6 compares MTF with another existing system for RBT to motivate its suitability from a practical standpoint.

7.1 Overview

As shown in the class diagram in Fig. 7.1, MTF closely follows the decomposition described in the previous chapters and has three abstract base classes corresponding to the three sub modules - `SearchMethod`, `AppearanceModel` and `StateSpaceModel`. For brevity, these will henceforth be referred to as `SM`, `AM` and `SSM` respectively, with the font serving to distinguish the *classes* from the corresponding *concepts*. The component class `IlluminationModel` of `AM` will likewise be referred to as `ILM`. Of the three main classes, only `SM` is a generic class that is templated on specializations of the other two classes. A concrete tracker, defined as a particular combination of the three sub modules, thus corresponds to a subclass of `SM` that has been instantiated with subclasses of `AM` and `SSM`, with the former optionally parameterized through a subclass of `ILM` for improved handling of lighting changes.

It may be noted that `SM` itself derives from a non generic base class called `TrackerBase` for convenient creation and interfacing of objects corresponding to heterogeneous

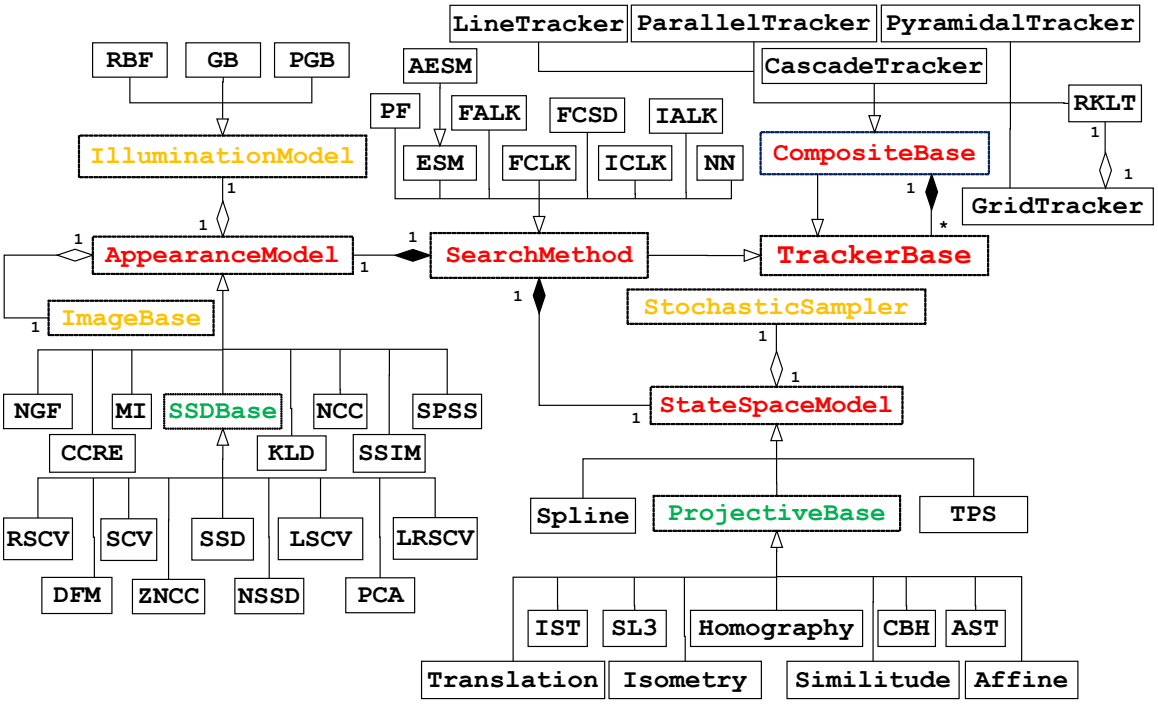


Figure 7.1: MTF Class Diagram showing all models currently implemented. Pure and partially abstract classes are respectively shown in red and green while concrete classes are in black. Classes that are sub parts of `AM` and `SSM` are in yellow. It may be noted that, though not shown here, most AMs also have separate multi channel variants that take RGB rather than gray scale images as input. Their names are formed by prefixing `MC` to those of their single channel counterparts - for example `MCSSD`, `MCMI` and `MCNCC`.

trackers, including those external to MTF, so that they can be run simultaneously and their results combined to create a tracker that is more robust than any of its components. Allowing a diverse variety of trackers to integrate seamlessly is one of the core design objectives of MTF and this is emphasized by having such trackers derive from a separate base class called `CompositeBase` that contains several instances of `TrackerBase` while also deriving from it. Since individual RBTs are well known to be prone to failures and more than three decades of research has failed to make significant improvements in this regard, the composite approach seems to be one of the more promising ones [33]. MTF has thus been designed to facilitate work in this direction. Five trackers have currently been implemented in this category:

- `GridTracker`: This corresponds to the stochastic SMs based on indirect sam-

pling described in Sec. 4.2.2. It uses a grid of (typically low DOF) trackers such that each tracks a different sub patch within the tracked object. An example is shown in Fig. 7.2 with a 5×5 grid of trackers. The results of these trackers are then combined by a robust estimator provided by the SSM to estimate the best fit warp that gives the overall location of the patch.

- `LineTracker`: This identifies straight lines in the object of interest and uses 2 DOF trackers to track multiple points on each line. The outputs of these trackers are then used to estimate the best fit line assuming that linearity of points is invariant under the warp that the object patch has undergone. Constraints between different lines, such as parallelism, can also be enforced to improve this estimation further. By resetting the trackers to their expected positions on these lines, any drift can be compensated for. This tracker does not work well yet and needs more work including better estimation method for best fit lines and improved constraints between different lines.

- `ParallelTracker`: This one runs multiple trackers in parallel to track the same patch and then combines their outputs to produce a more robust estimate of the tracked object's location. Many methods may be used to combine the locations produced by the different trackers, with a simple example being to take the mean of the bounding box corners. As mentioned in Sec. 4.3, these outputs can also be used to detect tracking failure by imposing a threshold on the amount by which the different locations might differ for tracking to be considered successful.

- `PyramidalTracker`: This one builds a Gaussian image pyramid [19] and then tracks each level of the pyramid through different instance of (usually) the same tracker such that the output of the tracker at level n, after appropriate scaling, is used as starting point for the one at level $n + 1$.

- `CascadeTracker`: This corresponds to the cascaded composite SMs described in Sec. 4.3. Similar to `ParallelTracker`, this too tracks the same patch using multiple trackers but here the output of the tracker at each layer of the cascade is used as the starting point for the tracker at the next layer with the last layer output fed back to the first one to make a closed loop system.

A particular SM in the proposed formulation is defined only by its objective - to find the $\mathbf{p}$ that maximizes the similarity measure defined by the AM. Thus, different

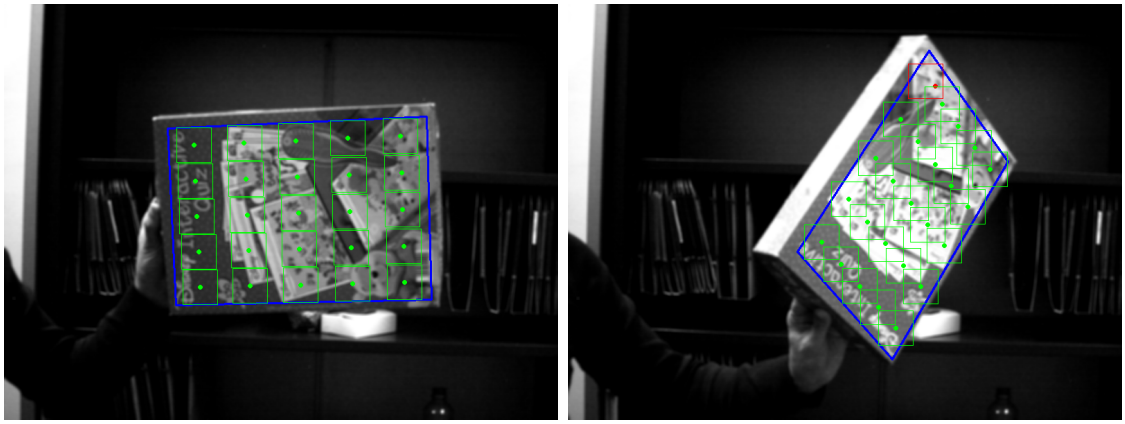


Figure 7.2: Two frames from beat sequence of PAMI dataset showing `GridTracker` with a 5×5 grid of trackers with each tracking 50×50 sub patch within the object of size 371×260. The red sub patch in the right frame shows an outlier detected by the robust estimation procedure while green ones are all inliers.

implementations of `SM` can cover a potentially wide range of methods that have little in common. As a result, `SM` is the least specific of the three main classes and only provides functions to initialize, update and reset the tracker along with accessors to obtain its current state. In fact, an SM is regarded in this framework simply as one way to *use* the methods provided by the other two sub modules to accomplish the above objective. The main idea behind this is to abstract out as much computation as possible from the SM to the AM and SSM so as to make for a general purpose tracker while also allowing aspects of the resultant functionality to be used by other SMs too. This design further encourages the conceptualization of an SM (or a tracker in general) in terms of what it has in common with and how it differs from other SMs (or trackers) as only the part unique to it needs to be implemented within the corresponding module in MTF. Therefore, this chapter describes only `AM` and `SSM` in detail while some specializations of `SM` currently available in MTF are presented as pseudo codes to exemplify the usage of the functionality described there to carry out the search in different ways.

Another consequence of this conceptual impreciseness of `SM` is that a specific SM may utilize only a small subset of the functionality provided by `AM`/`SSM`. For instance, GD type SMs do not use the random sampling functions of `SSM` and conversely, stochastic SMs do not use the differential functions required by the former. This has two further implications. Firstly, the functionality set out in `AM` and `SSM` is not fixed

but can change depending on the requirements of an SM, i.e. if a new SM is to be implemented that requires some functionality not present in the current specifications, the respective class can be extended to support it as long as such an extension makes logical sense within the definition of the corresponding sub module. Secondly, it is not necessary for all combinations of AMs and SSMs to support all SMs. For instance a similarity measure does not need to be differentiable to be a valid AM as long as it is understood that it cannot be used with SMs that require derivatives.

In the broadest sense, the division of functionality between `AM` and `SSM` described next can be understood as `AM` being responsible for everything to do with the image I, the sampled patch $\mathbf{I}(\mathbf{x})$ and the similarity f computed using it while `SSM` handles the actual *points* $\mathbf{x}$ at which the patch is sampled along with the warping function $\mathbf{w}$ that defines it in terms of the state parameters $\mathbf{p_s}$.

7.2 `AppearanceModel`

This class can be divided into three main parts where each is defined as a set of variables dependent on I_0 and I_t with a corresponding `initialize` and `update` function for each. The division is mainly conceptual and methods in different parts are free to interact with each other in practice. Table 7.1 presents a brief specification of some important methods in `AM`.

7.2.1 Image Operations

This part, abstracted into a separate class called `ImageBase`, handles all pixel level operations on the image I like extracting the patch $\mathbf{I}(\mathbf{x})$ and computing its numerical gradient $\nabla\mathbf{I}$ and Hessian $\nabla^2\mathbf{I}$ using sub pixel interpolation to handle non integral pixel locations. Currently, functions are implemented for bilinear, bicubic [224] and cubic BSpline [191, 225] interpolation along with the simple nearest neighbor method that may be used when speed is more important than accuracy.

Though `AM` bears a composition or "has a" relationship with `ImageBase`, in practice the latter is actually implemented as a base class of the former to maintain simplicity of the interface and allow a specializing class to efficiently override functions in both classes. Moreover, having a separate class for pixel related operations means that L2 models like SCV, RSCV and ZNCC (Sec. 5.1) that differ from SSD only in using a modified version of $\mathbf{I_0}$ or $\mathbf{I_t}$ - thus deriving from a common base class called `SSDBase`

Table 7.1: Specifications for important methods in `AM`

Function	Inputs	Output/Variable updated
updatePixVals	$\hat{\mathbf{x}}_\mathbf{t}$	$\mathbf{I_t}(\hat{\mathbf{x}}_\mathbf{t})$
updatePixGrad	$\hat{\mathbf{x}}_\mathbf{t}$	$\nabla\mathbf{I_t}(\hat{\mathbf{x}}_\mathbf{t})$
updatePixHess	$\hat{\mathbf{x}}_\mathbf{t}$	$\nabla^2\mathbf{I_t}(\hat{\mathbf{x}}_\mathbf{t})$
updateModel	$\mathbf{x_t}$	$\mathbf{I}^*$
updateSimilarity	None	$f(\mathbf{I_0}, \mathbf{I_t})$
updateInitGrad	None	$\frac{\partial f(\mathbf{I_0}, \mathbf{I_t})}{\partial \mathbf{I_0}}$ (Eq. 4.7)
updateCurrGrad	None	$\frac{\partial f(\mathbf{I_0}, \mathbf{I_t})}{\partial \mathbf{I_t}}$ (Eq. 4.5)
cmptInitJacobian	$\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}$	$\frac{\partial f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}}$ (Eq. 4.7)
cmptCurrJacobian	$\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}$	$\frac{\partial f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}}$ (Eqs. 4.4, 4.5)
cmptDifferenceOfJacobians	$\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}$	$\frac{\partial f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}} - \frac{\partial f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}}$ (Eq. 4.10)
cmptInitHessian*	$\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p_s}^2}$	$\frac{\partial^2 f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}^2}$ (Eq. 4.12)
cmptCurrHessian*	$\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$	$\frac{\partial^2 f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ (Eq. 4.13)
cmptSelfHessian*	$\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$	$\frac{\partial^2 f(\mathbf{I_t}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ (Eqs. 4.12, 4.13)
cmptSumOfHessians*	$\frac{\partial \mathbf{I_0}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_0}}{\partial \mathbf{p_s}^2}, \frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}, \frac{\partial^2 \mathbf{I_t}}{\partial \mathbf{p_s}^2}$	$\frac{\partial^2 f(\mathbf{I_0}(\mathbf{p}), \mathbf{I_t})}{\partial \mathbf{p}^2} + \frac{\partial^2 f(\mathbf{I_0}, \mathbf{I_t}(\mathbf{p}))}{\partial \mathbf{p}^2}$ (Eq. 4.18)

* All Hessian functions have overloaded variants that omit the second term in Eq. 4.3, as in Eq. 4.11, and so do not require $\frac{\partial^2 \mathbf{I}}{\partial \mathbf{p_s}^2}$ as input

- can implement the corresponding mapping entirely within the functions defined in `ImageBase` and so be combined easily with other AMs besides SSD. For instance, it is trivial to combine SCV with SSIM by simply replacing the `initializePixVals` and `updatePixVals` functions in the latter with those in the former.

7.2.2 Similarity Function

This is the core of `AM` and handles the computation of the similarity measure $f(\mathbf{I}^*, \mathbf{I^c}, \mathbf{p_a})$ and its derivatives $\partial f/\partial \mathbf{I}$ and $\partial^2 f/\partial \mathbf{I}^2$ w.r.t. both $\mathbf{I}^*$ and $\mathbf{I^c}$. It also provides interfacing functions to use inputs from `SSM` to compute the derivatives of f w.r.t. SSM parameters using the chain rule. As a notational convention, all interfacing functions, including those in `SSM`, are prefixed with `cmpt`.

The functionality specific to $\mathbf{p_a}$ is abstracted into the separate class `ILM` so it can be combined with any AM to add support for photometric parameters to it. This class provides functions to compute $g(\mathbf{I}, \mathbf{p_a})$ and its derivatives including $\partial g/\partial \mathbf{p_a}$, $\partial^2 g/\partial \mathbf{p_a}^2$, $\partial g/\partial \mathbf{I}$, $\partial^2 g/\partial \mathbf{I}^2$ and $\partial^2 g/\partial \mathbf{I} \partial \mathbf{p_a}$. These are called from within `AM` to compute the respective derivatives w.r.t. f so that the concept of ILM is transparent to

the SM. Specifications for the various functions in this class are provided in Table 7.2. It should be noted that `AM` is designed to support f with arbitrary $\mathbf{p_a}$ and `ILM` is a only a special case of this. `AM` also supports online learning to update the object's appearance, as present, for instance, in `PCA` [99] and provides a dedicated function named `updateModel` that can be called by the SM once per frame with the final estimate of the tracked object's location in that frame.

Since several of the functions in this part of `AM` involve common computations, there exist *transitive dependency* relationships between them, as depicted in Fig. 7.3, to avoid repeating these computations when multiple quantities are needed by the SM. What this means is that a function lower down in the dependency hierarchy may delegate part of its computations to any function higher up in the hierarchy so that the latter must be called *before* calling the former if correct results are to be expected. To further reduce unnecessary computations, an additional flag can be provided to `updateSimilarity` to specify that only that component of f that is needed by the derivative functions lower down in the dependence hierarchy is to be computed. This can be useful for GD based SMs that do not actually need f except when the LM formulation of $\hat{\mathbf{H}}_{\mathbf{self}}$ is used.

7.2.3 Distance Feature

This part is designed specifically to enable integration with the FLANN library [156] that is used by the NN based SM for all index types except GNN. It provides two main functions:

1. A feature transform $\mathbf{D}(\mathbf{I}^*) : \mathbb{R}^N \mapsto \mathbb{R}^K$ that maps the pixel values extracted from a patch $\mathbf{I}^*$ into a feature vector that contains the results of all computations in $f(\mathbf{I}^*, \mathbf{I^c})$ that depend only on $\mathbf{I}^*$ and likewise for $\mathbf{I^c}$. This transform is applied to all sampled patches during tracker initialization and only the resultant feature vectors are stored in the index. At runtime, it is applied to $\mathbf{I^c}$ and the feature vector is passed to the distance functor.

2. A highly optimized distance functor $f_D(\mathbf{D}(\mathbf{I}^*), \mathbf{D}(\mathbf{I^c})) : \mathbb{R}^K \times \mathbb{R}^K \mapsto \mathbb{R}$ that computes a measure of the distance or dissimilarity between $\mathbf{I}^*$ and $\mathbf{I^c}$ (typically the negative of $f(\mathbf{I}^*, \mathbf{I^c})$) given the distance features $\mathbf{D}(\mathbf{I}^*)$ and $\mathbf{D}(\mathbf{I^c})$ as inputs.

The main idea behind the design of these two components is to place as much computational load as possible on $\mathbf{D}$ so that the runtime speed of f_D is maximized, with

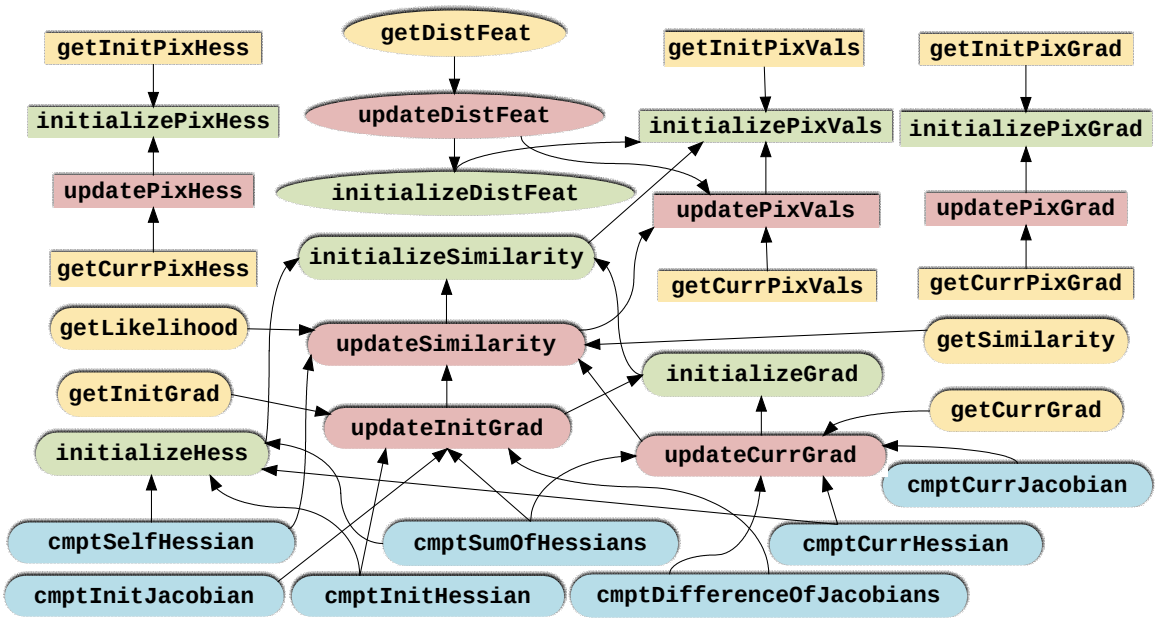


Figure 7.3: Dependency relationships between various functions in `AM`: an arrow pointing from A to B means that A depends on B. Color of a function box denotes its type - green: initializing; red: updating; blue: interfacing and yellow: accessor function. Shape of a function box represents the part of `AM` it belongs to - rectangle: Image Operations; rounded rectangle: Similarity Functions; ellipse: Distance Feature.

the premise that the former is called mostly during initialization when the sample dataset is to be built while the latter is called online to find the best matches for a candidate patch in the dataset. An optimal design may involve a trade off between the size K of the feature vector and the amount of computation performed in f_D.

For non symmetrical AMs, i.e. where $f(\mathbf{I}^*, \mathbf{I}^\mathbf{c}) \neq f(\mathbf{I}^\mathbf{c}, \mathbf{I}^*)$ (e.g. CCRE, RIU and SCV), the feature vector may also include an indicator flag so that f_D can determine which of its arguments corresponds to the $\mathbf{D}(\mathbf{I}^*)$ and which to $\mathbf{D}(\mathbf{I}^\mathbf{c})$. This is needed because FLANN does not specify the order in which the arguments will be passed to f_D and examination of its code showed that this order varies for each index type as well as for different calls to f_D within the same index.

7.3 `StateSpaceModel`

This class has a simpler internal state than `AM` and can be described by only three main variables at any time t - sampled grid points $\mathbf{x_t}$, corresponding corners $\mathbf{c_t}$ and

Table 7.2: Specifications for important methods in `ILM`.

Function	Inputs	Output
`update`	$\mathbf{p_a}, \Delta\mathbf{p_a}$	$\mathbf{p'_a} \mid \mathbf{g}(\mathbf{g}(\mathbf{I_t}, \mathbf{p_a}), \Delta\mathbf{p_a}) = \mathbf{g}(\mathbf{I_t}, \mathbf{p'_a})$
`apply`	$\mathbf{I_t}, \mathbf{p_a}$	$\mathbf{g}(\mathbf{I}, \mathbf{p_a})$
`invert`	$\mathbf{p_a}$	$\mathbf{p'_a} \mid \mathbf{g}(\mathbf{g}(\mathbf{I_t}, \mathbf{p_a}), \mathbf{p'_a}) = \mathbf{I_t}$
`cmptParamJacobian`	$\frac{\partial f}{\partial \mathbf{g}}, \mathbf{I_t}, \mathbf{p_a}$	$\frac{\partial f}{\partial \mathbf{p_a}}$ (Eq. 5.105)
`cmptPixJacobian`	$\frac{\partial f}{\partial \mathbf{g}}, \mathbf{I_t}, \mathbf{p_a}$	$\frac{\partial f}{\partial \mathbf{I_t}}$ (Eq. 5.103)
`cmptParamHessian`*	$\frac{\partial^2 f}{\partial \mathbf{g}^2}, \frac{\partial f}{\partial \mathbf{g}}, \mathbf{I_t}, \mathbf{p_a}$	$\frac{\partial^2 f}{\partial {\mathbf{p_a}}^2}$ (Eq. 5.106)
`cmptPixHessian`*	$\frac{\partial^2 f}{\partial \mathbf{g}^2}, \frac{\partial f}{\partial \mathbf{g}}, \mathbf{I_t}, \mathbf{p_a}$	$\frac{\partial^2 f}{\partial {\mathbf{I_t}}^2}$ (Eq. 5.104)
`cmptCrossHessian`*	$\frac{\partial^2 f}{\partial \mathbf{g}^2}, \frac{\partial f}{\partial \mathbf{g}}, \mathbf{I_t}, \mathbf{p_a}$	$\frac{\partial^2 f}{\partial \mathbf{I_t} \partial \mathbf{p_a}}$ (Eq. 5.108)

* All Hessian functions have overloaded variants that omit the second terms in respective expressions and so do not require $\frac{\partial f}{\partial \mathbf{g}}$ as input

state parameters $\mathbf{p_{st}}$. It may be noted (Fig. 7.1) that, though `SSM` is designed to support any arbitrary $\mathbf{w}$, most SSMs currently implemented are subsets of the planar projective transform and so derive from `ProjectiveBase` that abstracts out the functionality common to such transforms. Functions in `SSM` can be divided into two categories:

7.3.1 Warping Functions

This is the core of `SSM` and provides a function $\mathbf{w}$ to transform a regularly spaced grid of points $\mathbf{x_0}$ representing the target patch into a warped patch $\mathbf{x_t} = \mathbf{w}(\mathbf{x_0}, \mathbf{p_{st}})$ that captures the tracked object's motion/deformation in image space. It also allows for the compositional inverse of $\mathbf{w}$ to be computed (`invertState`) to support inverse SMs. Further, there are functions to compute the derivatives of $\mathbf{w}$ w.r.t. both $\mathbf{x}$ and $\mathbf{p_s}$ but, unlike `AM`, `SSM` does not store these as state variables; rather, their computation is implicit in the interfacing functions that compute $\partial\mathbf{I}/\partial\mathbf{p_s}$ and $\partial^2\mathbf{I}/\partial{\mathbf{p_s}}^2$ using chain rule. This design decision was made for reasons of efficiency since $\partial\mathbf{w}/\partial\mathbf{p_s}$ and $\partial\mathbf{w}/\partial\mathbf{x}$ are large and often very sparse tensors and computing these separately not only wastes a lot of memory but is also very computationally inefficient.

Finally, there are four ways to update the internal state: incrementally using additive (`additiveUpdate`) or compositional (`compositionalUpdate`) formulations (Sec. 4.1.3), or outright by providing either the state vector (`setState`) or the corresponding corners (`setCorners`) that define the current location of the patch. There are no complex dependencies in `SSM` - the correct performance of interfacing functions and

Table 7.3: Specifications for important methods in `SSM`.

Function	Inputs	Output/Result
`compositionalUpdate`	$\Delta\mathbf{p_s}$	$\mathbf{p_{st}} = \mathbf{p_s}' \mid \mathbf{w}(\mathbf{x}, \mathbf{p_s}') = \mathbf{w}(\mathbf{w}(\mathbf{x}, \Delta\mathbf{p_s}), \mathbf{p_{st}})$
`additiveUpdate`	$\Delta\mathbf{p_s}$	$\mathbf{p_{st}} = \mathbf{p_{st}} + \Delta\mathbf{p_s}$
`invertState`	$\mathbf{p_s}$	$\mathbf{p_s}' \mid \mathbf{w}(\mathbf{w}(\mathbf{x}, \mathbf{p_s}), \mathbf{p_s}') = \mathbf{x}$
`cmptPixJacobian`	$\nabla\mathbf{I_t}$	$\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_s}}\Big\vert_{\mathbf{p_s}=\mathbf{p_{st}}}$ (Eq. 4.4)
`cmptWarpedPixJacobian`	$\nabla\mathbf{I_t}$	$\frac{\partial \mathbf{I_t}(\mathbf{w})}{\partial \mathbf{p_s}}\Big\vert_{\mathbf{p_s}=\mathbf{p_{s0}}}$ (Eq. 4.5, 4.6)
`cmptApproxPixJacobian`	$\nabla\mathbf{I_0}$	$\frac{\partial \mathbf{I_t}}{\partial \mathbf{p_{st}}}$ (approx) (Eq. 4.8, 4.9)
`cmptPixHessian`	$\nabla\mathbf{I_t}, \nabla^2\mathbf{I_t}$	$\frac{\partial^2 \mathbf{I_t}}{\partial {\mathbf{p_s}}^2}\Big\vert_{\mathbf{p_s}=\mathbf{p_{st}}}$ (Eq. 4.14)
`cmptWarpedPixHessian`	$\nabla\mathbf{I_t}, \nabla^2\mathbf{I_t}$	$\frac{\partial^2 \mathbf{I_t}(\mathbf{w})}{\partial {\mathbf{p_s}}^2}\Big\vert_{\mathbf{p_s}=\mathbf{p_{s0}}}$ (Eq. 4.15, 4.16)
`cmptApproxPixHessian`	$\nabla\mathbf{I_0}, \nabla^2\mathbf{I_0}$	$\frac{\partial^2 \mathbf{I_t}}{\partial {\mathbf{p_{st}}}^2}$ (approx) (Eq. 4.17)
`applyWarpToPts`	$\mathbf{x}, \mathbf{p_s}$	$\mathbf{w}(\mathbf{x}, \mathbf{p_s})$
`composeWarps`	$\mathbf{p_{s1}}, \mathbf{p_{s2}}$	$\mathbf{p_{s12}} \mid \mathbf{w}(\mathbf{x}, \mathbf{p_{s12}}) = \mathbf{w}(\mathbf{w}(\mathbf{x}, \mathbf{p_{s1}}), \mathbf{p_{s2}})$
`getIdentityWarp`	None	$\mathbf{p_{sI}} \mid \mathbf{w}(\mathbf{x}, \mathbf{p_{sI}}) = \mathbf{x}$

accessors depends only on one of the update functions being called at every iteration. A separate dependency diagram is thus omitted here.

In addition to these state related functions, this part of `SSM` also provides several general utility functions that can be used independently of the SSM's current state. These include `applyWarpToCorners` and `applyWarpToPts` which can be used for transforming the provided grid points or its bounding corners according to the given warp, `composeWarps` that returns a single warp whose warping action is same as the composite action of the two provided warps and getIdentityWarp that returns the $\mathbf{p_s}$ corresponding to the identity transform. Table 7.3 lists the functional specifications for some important methods in this part.

7.3.2 Stochastic Sampler

This part is provided to support stochastic SMs and offers following functionality to this end:

1. generate small random incremental updates to $\mathbf{p_s}$ (`generatePerturbation`) by drawing these from a zero mean normal distribution with either user provided or heuristically estimated (`estimateStateSigma`) variance.

2. generate stochastic state samples using the given state transition model - currently random walk (`additiveRandomWalk`) and first order auto regression (`additiveAutoRegression1`) are supported. There are also `compositional` variants of these functions.

3. estimate the mean of a set of samples of $\mathbf{p_s}$ (`estimateMeanOfSamples`) where the definition of mean is dependent on the SSM itself.

4. estimate the best fit $\mathbf{p_s}$ from a set of original and warped point pairs (`estimateWarpFromPts`) using a robust method. Both methods described in Sec. 4.2.2 - RANSAC [33] and LMS [226] - are supported. Their implementations have been adapted for all the SSMs from the one for homography available in `findHomography` function of OpenCV. A simpler version called `estimateWarpFromCorners` is also provided to estimate the warp from a pair of corresponding corners - this is currently implemented using the DLT algorithm for all subclasses of `ProjectiveBase`.

7.4 Examples of Search Methods

This section presents pseudo codes for several SMs currently implemented in MTF to exemplify the usage of functions described in the previous sections. Following are some points and conventions to be noted:

- *am* and *ssm* respectively refer to instances of `AM` and `SSM` (or rather of specializations thereof)

- *am* has direct access to the latest image in the sequence so it is not passed explicitly in function calls - this is one of the design features of `AM` to avoid the overhead of passing the image repeatedly.

- several special cases like the optional use of the first order Hessian (Eq. 4.11), parameterization and online learning of AM and iterative form of the `update` function are demonstrated only for ICLK but should be obvious by analogy for other SMs too.

- v.head(h) and v.tail(t) in ICLK respectively refer to the first h and last t elements in the $h + t$ length vector v.

Algorithm 1 ICLK

```
1: function initialize(corners)
2:     ssm.initialize(corners)
3:     am.initializePixVals(ssm.getPts())
4:     am.initializePixGrad(ssm.getPts())
5:     am.initializeSimilarity()
6:     am.initializeGrad()
7:     am.initializeHess()
8:     dI0_dps ← ssm.cmptWarpedPixJacobian(am.getInitPixGrad())
9:     if use_first_order_hessian then
10:        d2f_dp2 ← am.cmptSelfHessian(dI0_dps)
11:    else
12:        am.initializePixHess(ssm.getPts())
13:        d2I0_dps2 ← ssm.cmptInitPixHessian(
               am.getInitPixHess(), am.getInitPixGrad())
14:        d2f_dp2 ← am.cmptSelfHessian(dI0_dps, d2I0_dps2)
15:    end if
16: end function
17: function update
18:    for i ← 1, max_iters do
19:        am.updatePixVals(ssm.getPts())
20:        am.updateSimilarity()
21:        am.updateInitGrad()
22:        df_dp ← am.cmptInitJacobian(dI0_dps)
23:        delta_p ← −d2f_dp2.inverse()∗df_dp
24:        delta_ps ← delta_p.head(ssm.getStateSize())
25:        delta_pa ← delta_p.tail(am.getStateSize())
26:        inv_delta_ps ←ssm.invertState(delta_ps)
27:        inv_delta_pa ←am.invertState(delta_pa)
28:        prev_corners←ssm.getCorners()
29:        ssm.compositionalUpdate(inv_delta_ps)
30:        am.update(inv_delta_pa)
31:        if ‖prev_corners − ssm.getCorners()‖² < ϵ then
32:            break
33:        end if
34:    end for
35:    am.updateModel(ssm.getPts())
36:    return ssm.getCorners()
37: end function
```

Algorithm 2 FCLK

```
1: function initialize(corners)
2:     lines 2-7 of Alg. 1
3:     am.initializePixHess(ssm.getPts())
4: end function
5: function update
6:     lines 19-20 of Alg. 1
7:     am.updateCurrGrad()
8:     am.updatePixGrad(ssm.getPts())
9:     am.updatePixHess(ssm.getPts())
10:    dIt_dps ← ssm.cmptWarpedPixJacobian(am.getCurrPixGrad())
11:    d2It_dps2 ← ssm.cmptWarpedPixHessian(
                   am.getCurrPixHess(), am.getCurrPixGrad())
12:    df_dp ←am.cmptCurrJacobian(dIt_dps)
13:    d2f_dp2 ←am.cmptSelfHessian(dIt_dps, d2It_dps2)
14:    delta_p ← −d2f_dp2.inverse()∗df_dp
15:    ssm.compositionalUpdate(delta_p)
16:    return ssm.getCorners()
17: end function
```

- different algorithms make extensive references to portions of each other not only to save space by avoiding redundancy but also to emphasize the parts they have in common.

- *flann* in NN is an instance of FLANN library [156] that can build an index from a set of samples and search it for a new candidate.

- variables used to store the results of computations are not described explicitly but their meanings should be clear from their names and context. For instance, *sample_dataset* and *ssm_perturbations* used in NN respectively refer to $n \times K$ and $n \times S$ matrices, each of whose rows contains the distance feature $\mathbf{D}$ (Sec. 7.2.3) and the SSM state $\mathbf{p_s}$ corresponding to one sample so that n = number of samples.

- only one state transition model is shown for PF though several others are available too (Sec. 7.3.2).

- Alg. 8 only shows the `GridTracker` component of LMS and RANSAC and the actual robust estimation using one of these algorithms is carried out in `estimateWarpFromPts` function of `SSM` (Sec. 7.3.2).

Algorithm 3 ESM

1: **function** initialize(*corners*)
2: lines 2-3 of Alg. 2
3: *dI0_dps* ← *ssm*.cmptWarpedPixJacobian(*am*.getInitPixGrad())
4: *d2f_dp2_0* ← *am*.cmptSelfHessian(*dI0_dps*, *d2I0_dps2*)
5: **end function**
6: **function** update
7: lines 6-11 of Alg. 2
8: *am*.updateInitGrad()
9: *df_dp*←*am*.cmptDifferenceOfJacobians(*dI0_dps*, *dIt_dps*)
10: *d2f_dp2_t* ←*am*.cmptSelfHessian(*dIt_dps*, *d2It_dps2*)
11: *d2f_dp2* ←*d2f_dp2_0* + *d2f_dp2_t*
12: lines 14-16 of Alg. 2
13: **end function**

Algorithm 4 IALK

1: **function** initialize(*corners*)
2: same as Alg. 2
3: **end function**
4: **function** update
5: lines 6-7 of Alg. 2
6: *dIt_dps* ← *ssm*.cmptApproxPixJacobian(*am*.getInitPixGrad())
7: *d2It_dps2* ← *ssm*.cmptApproxPixHessian(*am*.getInitPixHess(), *am*.getInitPixGrad())
8: lines 12-14 of Alg. 2
9: *ssm*.additiveUpdate(*delta_p*)
10: **return** *ssm*.getCorners()
11: **end function**

Algorithm 5 FALK

1: **function** initialize(*corners*)
2: same as Alg. 2
3: **end function**
4: **function** update
5: lines 6-9 of Alg. 2
6: *dIt_dps* ← *ssm*.cmptPixJacobian(*am*.getCurrPixGrad())
7: *d2It_dps2* ← *ssm*.cmptPixHessian(*am*.getCurrPixHess(), *am*.getCurrPixGrad())
8: lines 8-10 of Alg. 4
9: **end function**

- sampling resolution of *ssm* in Alg. 8 is set to be same as the grid resolution

Algorithm 6 NN

```
1: function initialize(corners)
2:     lines 2-3 of Alg. 1
3:     state_sigma← ssm.estimateStateSigma()
4:     ssm.initializeSampler(state_sigma)
5:     am.initializeDistFeat()
6:     for sample_id ← 1, no_of_samples do
7:         ssm_updates.row(sample_id) ← ssm.generatePerturbation()
8:         inv_update ←ssm.invertState(ssm_updates.row(sample_id))
9:         ssm.compositionalUpdate(inv_update)
10:        am.updatePixVals(ssm.getPts())
11:        am.updateDistFeat()
12:        sample_dataset.row(sample_id) ← am.getDistFeat()
13:        ssm.compositionalUpdate(ssm_updates.row(sample_id))
14:    end for
15:    flann.buildIndex(sample_dataset)
16: end function
17: function update
18:    am.updatePixVals(ssm.getPts())
19:    am.updateDistFeat()
20:    nn_sample_id ← flann.searchIndex(am.getDistFeat())
21:    ssm.compositionalUpdate(ssm_updates.row(nn_sample_id))
22:    return ssm.getCorners()
23: end function
```

and the function `getRegion`(c, s) in line 6 returns the corners of a rectangular region of size s with centroid c.

7.5 Use Cases

This section presents following use cases for MTF in C++ style pseudo code:

- Track an object in an image sequence using a simple (Alg. 9) and a composite (Alg. 10) tracker.
- Estimate the trajectory of a UAV within a large satellite image of an area from images it took while flying over that area (Alg. 11). Fig. 7.4 shows an example.
- Create an image mosaic in real time from a video sequence captured by a camera moving over different parts of the planar scene to be stitched (Alg. 12). Fig.

Algorithm 7 PF

```
1: function initialize(corners)
2:     lines 2-4 of Alg. 6
3:     am.initializeSimilarity()
4:     for particle_id ← 1, no_of_particles do
5:         particles[particle_id].state ← ssm.getState()
6:         particles[particle_id].weight ← 1/no_of_particles
7:     end for
8: end function
9: function update
10:    for particle_id ← 1, no_of_particles do
11:        particles[particle_id].state ← ssm.compositionalRandomWalk(
12:                    particles[particle_id].state)
13:        ssm.setState(particles[particle_id].state)
14:        am.updatePixVals(ssm.getPts())
15:        am.updateSimilarity()
16:        particles[particle_id].weight ←am.getLikelihood()
17:    end for
18:    normalize weights and resample the particles
19:    mean_state ← ssm.estimateMeanOfSamples(particles);
20:    ssm.setState(mean_state)
21:    return ssm.getCorners()
22: end function
```

7.5 shows an example.

Following are some supplementary details regarding variables and functions used in these algorithms that may assist the reader in understanding them better:

- MTF comes with an input module with wrappers for the image capturing functions in OpenCV, ViSP and XVision. Represented by the variable *input*, it is assumed to have been initialized with the appropriate source.
- Raw images acquired by the input module can optionally be passed to the preprocessing module that provides wrappers for OpenCV image filtering and conversion functions. `GaussianSmoothing` in Alg. 9 is an example.
- Though only five combinations of SM, AM and SSM are shown here, these can be replaced by virtually any combination of methods in Fig. 7.1.
- MTF also has a set of general utility functions for image and warping related operations, following of which have been used in Alg. 11 and 12:

Algorithm 8 LMS/RANSAC

```
1: function initialize(corners)
2:     sub_trackers ← vector of 2 DOF sub patch trackers
3:     ssm.initialize(corners)
4:     curr_pts ← ssm.getPts()
5:     for pt_id ← 1, no_of_pts do
6:         sub_patch_corners ← getRegion(curr_pts[pt_id], sub_patch_size)
7:         sub_trackers[pt_id].initialize(sub_patch_corners)
8:     end for
9: end function
10: function update
11:     prev_pts ← curr_pts
12:     for pt_id ← 1, no_of_pts do
13:         sub_trackers[pt_id].update()
14:         curr_pts[pt_id] ← getCentroid(sub_trackers[pt_id].getRegion())
15:     end for
16:     opt_warp ← ssm.estimateWarpFromPts(prev_pts, curr_pts)
17:     warped_corners ← ssm.applyWarpToCorners(ssm.getCorners(), opt_warp)
18:     ssm.setCorners(warped_corners)
19:     lines 4-8 of Alg. 8
20:     return ssm.getCorners()
21: end function
```

  - `getFrameCorners`(*image*) returns a 2×4 matrix containing the corners of *image* and is thus used when the entire image is to be considered as the tracked region.
  - `writePixelsToImage`(*patch*, *corners*, *size*) writes the pixel values in *patch* to an image with dimensions *size* within the region bounded by *corners*.

- *init_mos_location* in Alg. 12 is the user specified location of the first frame in the sequence within the mosaic image of size *mos_size*. This is typically the center of the mosaic though can be elsewhere depending on the actual sequence.

7.6 Performance

An existing system that is somewhat similar to MTF in functionality is the template tracker module of the Visual Servoing Platform (ViSP) library [24] that includes 4 SMs, 3 AMs and 6 SSMs though not all combinations work. While being functionally

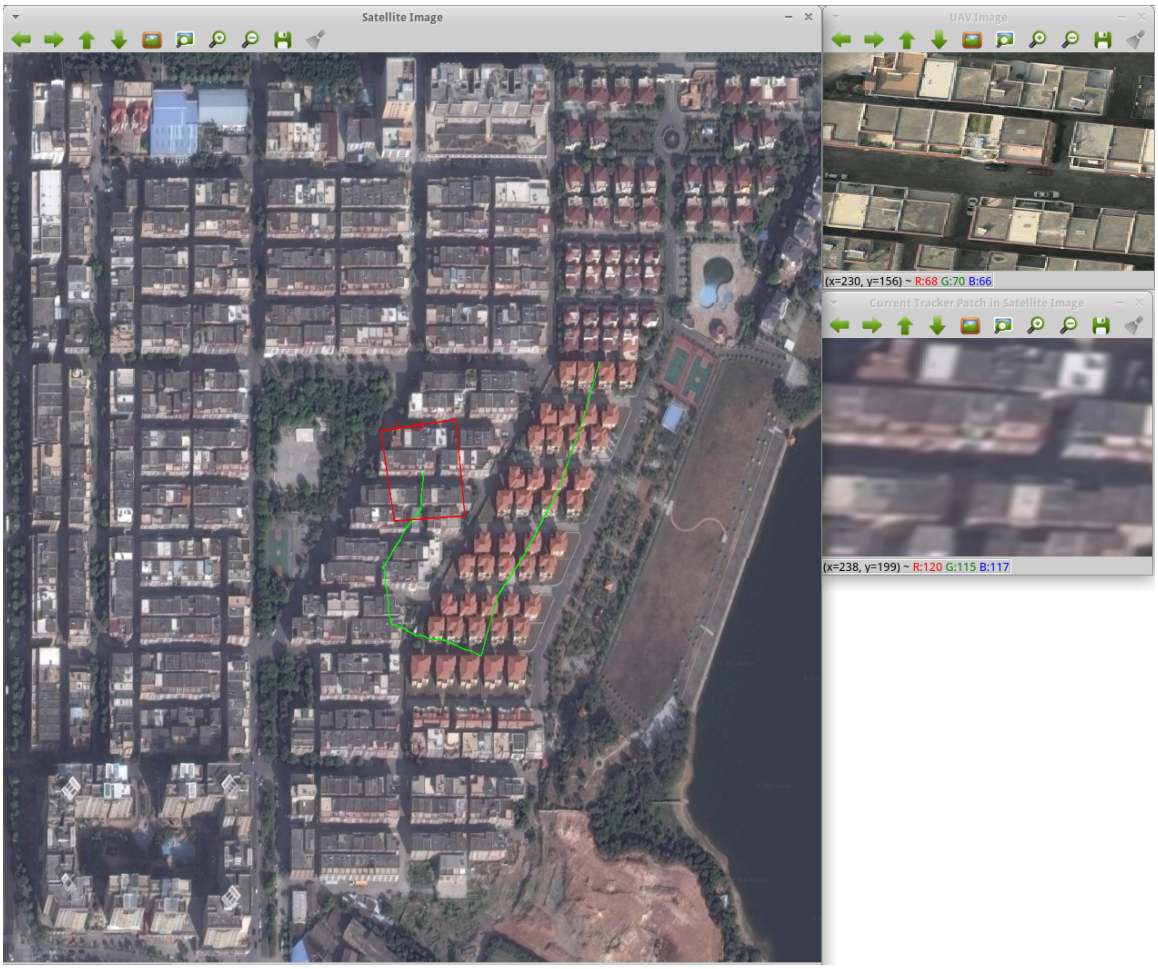


Figure 7.4: Sample screen shot taken while running the UAV trajectory estimation application. The large window on the left shows the satellite image scaled down to fit the screen. The trajectory estimated so far is marked in greenand the bounding box corresponding to the current tracker location shown in red. The top window on the right shows the current image from the sequence captured by the UAV while flying over the area in the satellite image. The bottom window shows the patch extracted from this image at the current tracker location. It is assumed here that the size of the UAV images is approximately same as that of the corresponding region in the satellite image. It can be seen that the two images, though of the same region, differ markedly in appearance as one was captured by the UAV camera and the other by the satellite. This is an example of multi modality tracking scenario that AMs like MI and NCC can handle well.

Algorithm 9 Object Tracking - Simple

```
1: using namespace mtf;
2: ICLK<SSD, Homography> tracker;
3: GaussianSmoothing pre_proc(input.getFrame(), tracker.inputType());
4: tracker.initialize(pre_proc.getFrame(), init_location);
5: while input.update() do
6:     pre_proc.update(input.getFrame());
7:     tracker.update(pre_proc.getFrame());
8:     new_location ← tracker.getRegion();
9: end while
```

Algorithm 10 Object Tracking - Composite

```
1: PF<ZNCC, Affine> tracker1;
2: FCLK<SSIM, SL3> tracker2;
3: vector<TrackerBase*> trackers = {&tracker1, &tracker2};
4: CascadeTracker tracker(trackers);
5: lines 3-9 of Alg. 9
```

Algorithm 11 UAV Trajectory Estimation in Satellite Image

```
1: ESM<MI, Similitude> tracker;
2: uav_img_corners←getFrameCorners(input.getFrame());
3: tracker.initialize(satellite_img, init_uav_location);
4: curr_uav_location←tracker.getRegion();
5: while input.update() do
6:     tracker.initialize(input.getFrame(), uav_img_corners);
7:     tracker.setRegion(curr_uav_location);
8:     tracker.update(satellite_img);
9:     curr_uav_location←tracker.getRegion();
10: end while
```

Algorithm 12 Online Image Mosaicing

```
1: FALK<MCNCC, Isometry> tracker;
2: mos_img ← writePixelsToImage(input.getFrame(), init_mos_location, mos_size);
3: mos_location ← init_mos_location;
4: while input.update() do
5:     temp_img ← writePixelsToImage(input.getFrame(), mos_location, mos_size);
6:     tracker.initialize(temp_img, mos_location);
7:     tracker.update(mos_img);
8:     mos_location ← tracker.getRegion();
9:     mos_img ← writePixelsToImage(input.getFrame(), mos_location, mos_size);
10: end while
```

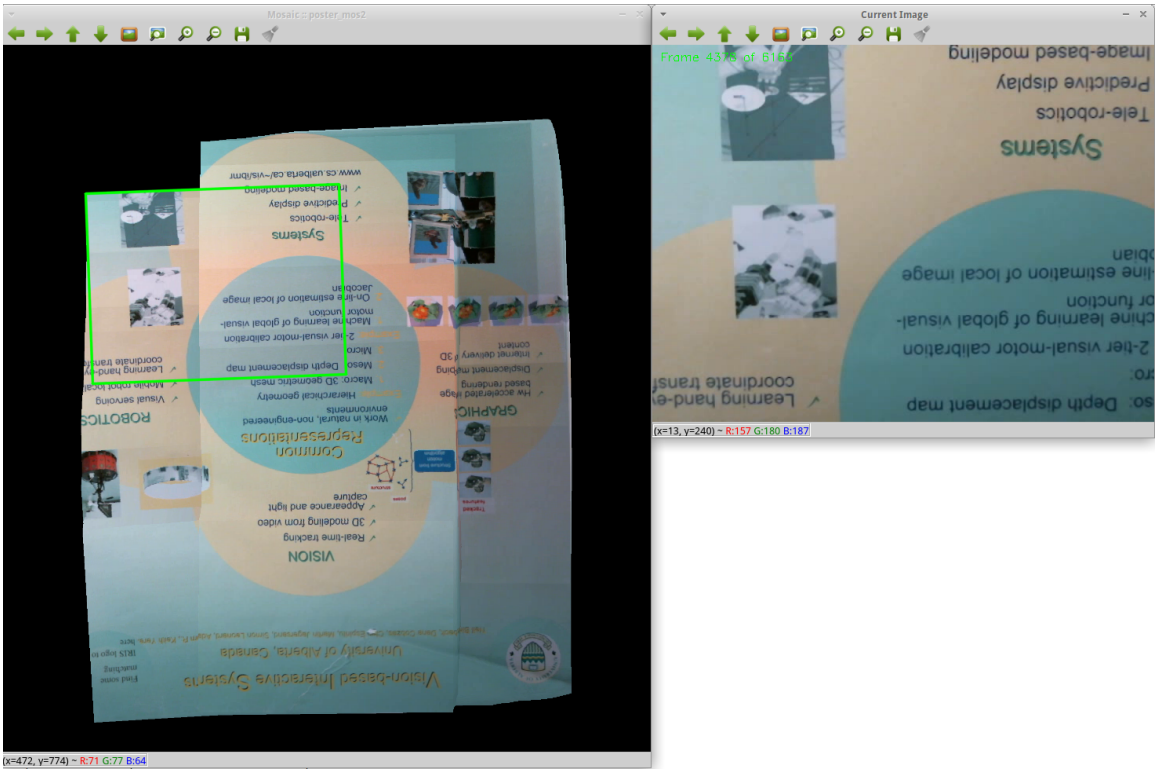


Figure 7.5: Sample screen shot taken while running the online mosaicing application. The large window on the left shows the mosaic constructed so far with the location of the current image shown in green. The window on the right shows the current image.

similar, at least for the methods present in ViSP, MTF offers several advantages over it. Firstly, SMs and AMs in ViSP are not implemented as independent modules, rather each combination of methods has its own separate class. This makes it difficult to add a new method for either of these sub modules and combine it with existing methods for the others. Secondly, MTF has several more AMs, one more GD based SM (IALK) as well as four stochastic SMs (Sec. 4.2). It also allows multiple SMs to be combined effortlessly to create novel composite SMs (Sec. 4.3). Similarly, MTF makes it equally easy to combine multiple AMs to create new composite AMs though these are outside the scope of this study

Lastly, and perhaps most importantly from a practical standpoint, MTF is significantly faster than ViSP. As shown in Fig. 7.6, MTF is usually more than an order of magnitude faster, with its speed being over 20 times higher than ViSP on average for all SSMs except affine where it is slightly lower at around 14 but still enough to make

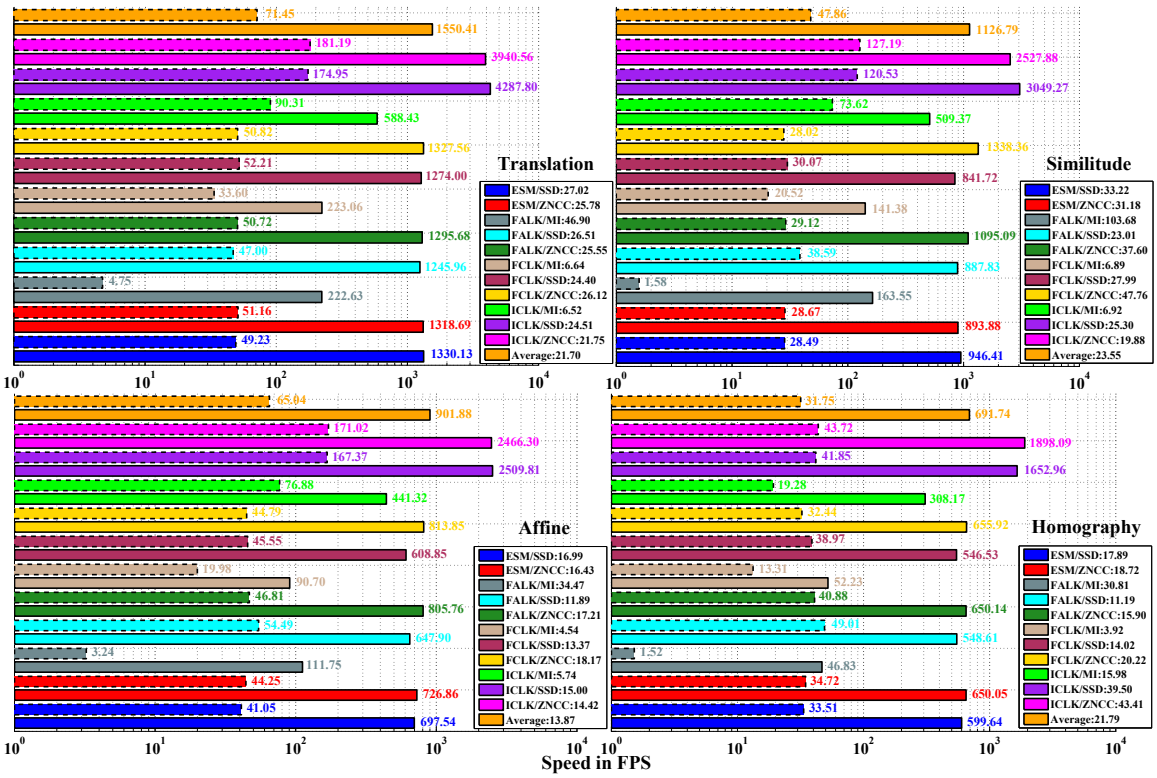


Figure 7.6: ViSP vs MTF average tracker speeds in FPS for all combinations of SMs and AMs supported by ViSP with 4 different SSMs. MTF and ViSP results are shown in **solid and dotted lines** respectively. Note that **logarithmic scaling** has been used on the x axis for better visibility of ViSP bars though the actual figures are also shown for additional clarity. **Speedup** provided by MTF - as measured by the ratio of MTF to ViSP speed - is shown in the legends. All results were generated on a 4 GHz Intel Core i7-4790K machine with 32 GB of RAM.

a significant impact in practice. This is mainly because it uses the Eigen library [227] for all mathematical computations and this is known to be one of the fastest [228]. It may be noted that MTF has not been carefully optimized and parallelized yet and the speed gain is likely to increase further once this is completed. Unlike systems like ViSP and XVision that use their own linear algebra subsystems, being based on a fast and open source library that is under active development affords MTF with the further advantage of automatically improving in speed and reliability whenever improvements are made to this library. This speed advantage is particularly significant in applications like visual servoing and SLAM where several different regions in the same video stream often need to be tracked simultaneously. In addition to

allowing them to run in real time, MTF offers another benefit for such multi tracking applications. For each feature or region of the scene, the tracker best matching the characteristics of that region needs to be selected for optimal performance since different combinations of methods are best suited for distinct scenarios [10]. This is much easier to accomplish using MTF than it would be by combining different executables, if any, published by the authors of the respective methods.

It may be mentioned here that the template tracking module of ViSP is notably buggy at present and only managed to complete about 70% of all the sequences used for testing. The results in Fig. 7.6 were thus produced by averaging over only these sequences. Also, ViSP does not allow arbitrary sampling resolutions to be used; instead the user must specify integral sampling ratios for both vertical and horizontal directions so that the sampling resolution in each dimension can only be such that the object size in the respective dimension is an integral multiple of the same. All possibilities for these sampling ratios were tested for each object and those that resulted in the total number of pixels being closest to 2500 were used to ensure fair comparison with the fixed 50×50 resolution used for the MTF trackers.

7.7 Summary

This chapter presented details of the system design used for implementing MTF in C++. This included an overview of the division of functionality between the three main abstract classes corresponding to the three modules into which RBT has been decomposed in this work. This was followed by detailed descriptions of two of these classes - `AM` and `SSM` - that provide well defined interfaces for all concrete classes that implement these. To account for the far greater variability in the possible approaches to optimization, `SM` was treated simply as a way to utilize the functionality in `AM` and `SSM` to search for the optimal warp and was thus described through pseudo codes of algorithms corresponding to various SMs. This was followed by three practical use cases for MTF, again in the form of algorithms, and a brief performance comparison with another popular library in this domain.

Chapter 8

Evaluation Methodology

This chapter provides details of the datasets - both real world and synthetic - used for evaluating the trackers as well as the criteria for assessing their performance. It also provides a summary of the configuration settings used for the various modules in the results reported in the next chapter.

8.1 Datasets

Following four publicly available datasets [1] have been used to analyze the trackers:

1. Tracking for Manipulation Tasks (**TMT**) [229] dataset: This contains videos of some common manipulation tasks performed at several speeds and under varying lighting conditions (Fig. 8.1). The tasks are divided into two categories - single and composite motion tasks. The first includes 8 tasks performed using 4 different objects - book, cereal box, juice carton and mug. Each of these tasks involves a particular type of motion including, for instance, in plane and out of plane translations and rotations. To provide varying levels of tracking difficulty, these tasks are performed under two lighting conditions and at six different speeds, including one where the speed increases in the course of the sequence, . Also, some of these are performed by a robotic arm in addition to a human to better represent the jerky movement associated with the former that is encountered in visual servoing applications. The second category includes more general tasks that involve performing multiple types of motion in a single complex task. Examples of tasks in this category include playing with a toy bus, reading and highlighting a newspaper and placing a letter in an envelope. All

[1] available at http://webdocs.cs.ualberta.ca/~vis/mtf/

sequences have RGB images and feature the eye-to-hand configuration where the camera is static and the object moves.

2. Visual Tracking Dataset provided by **UCSB** [230]: This includes 16 different motion patters including camera rotation, zooming and panning, gradual and sudden lighting changes, perspective distortion, 9 different speeds of translation and unconstrained motion. Each type of motion is recorded using 6 different planar textures for a total of 96 sequences. Unlike TMT, all of these have been recorded using the eye-in-hand configuration where the object is static and the camera moves, though, similar to TMT, they have RGB images too. UCSB is more challenging than TMT but also rather artificial (Fig. 8.3) as its sequences were created to represent specific challenges rather than realistic scenarios. Though this dataset presents a variety of challenges, the one that causes tracking failures most often is the motion blur induced by fast or abrupt camera movements in both the translation and the unconstrained motion sequences.

3. **LinTrack** dataset [231]: This includes 3 long sequences where a range of challenging unconstrained motions are performed using 3 objects - towel, phone and mouse pad (Fig. 8.4). Two of these sequences feature eye-to-hand configuration while the remaining one has eye-in-hand. These sequences are more realistic than those in UCSB but also more difficult to track. They do not feature any significant occlusions or illumination variations so the main challenges include extreme perspective distortions, large scale changes and motion blur caused by sudden movements. As in UCSB, the latter is most often the cause of tracking failures. The sheer length of these sequences also proved hard to cope with for most of the tested trackers. Unlike the previous two datasets, this one features only gray scale images.

4. **PAMI** dataset: This includes a collection of 28 challenging planar tracking sequences (Fig. 8.2) from several important works in literature [1, 84, 77, 85, 140, 232, 32]. It has been named after [1] from where several of the sequences originate. The sequences here vary widely in the level of tracking difficulty though overall, it seems slightly easier to track than UCSB for most trackers (appendix B). The main challenges here include localized illumination changes, significant occlusions and non planarity. This dataset contains both color and gray scale images and also includes both eye-in-hand and eye-to-hand scenarios.

Table 8.1: Summary of datasets used for evaluating trackers. Subsequences refer to the initialization of trackers at 10 different frames in each sequence (Sec. 8.2).

Dataset	Without Subsequences			With Subsequences		
	Sequences	**Total Frames**	**Trackable Frames[1]**	**Sub-sequences**	**Total Frames**	**Trackable Frames[1]**
TMT	109	70592	70483	1090	390470	389380
UCSB	96	6889	6793	960	41170	40210
LinTrack	3	12477	12474	30	68700	68670
PAMI	28	16511	16483	280	91400	91120
Total	**236**	**106469**	**106233**	**2360**	**591740**	**589380**

[1] The first frame in each sequence/subsequence, where the tracker is initialized, is not considered for evaluating the tracking performance

All of these datasets except PAMI have full pose (8 DOF) ground truth data made available by their creators, which makes them suitable for evaluating high precision trackers that are the principal subjects of this study. For PAMI, this data was generated using a combination of very high precision tracking and manual refinement [229]. Table 8.1 provides a summary of the number of sequences and frames in these datasets.

A popular dataset in this domain that is excluded here is the Metaio dataset [233]. It might have helped to make the results more comprehensive if testing could be done on this too but the tracker evaluation service is no longer offered by its creators and attempts to obtain its ground truth from them also proved unsuccessful.

8.1.1 Synthetic Datasets

In addition to the above real world sequences, testing has also been done on synthetic frames generated using the procedure described in [68] and [70] except that homography has been used instead of affine for warping the frames. Also, rather than generating all warped frames from a single source image, 25 different images drawn from the four datasets (Fig. 8.5) have been used so that varying object sizes and texture types are tested with to rule out any bias with these factors. The random homography warps were generated by perturbing the four corners of the original bounding box and estimating the warp matrix using the DLT algorithm. This is similar to the first step of the process described in Sec. 6.5.1 except that here the perturbations were applied to the original corners rather than the normalized ones.

Figure 8.1: Representative frames from sequences in TMT dataset

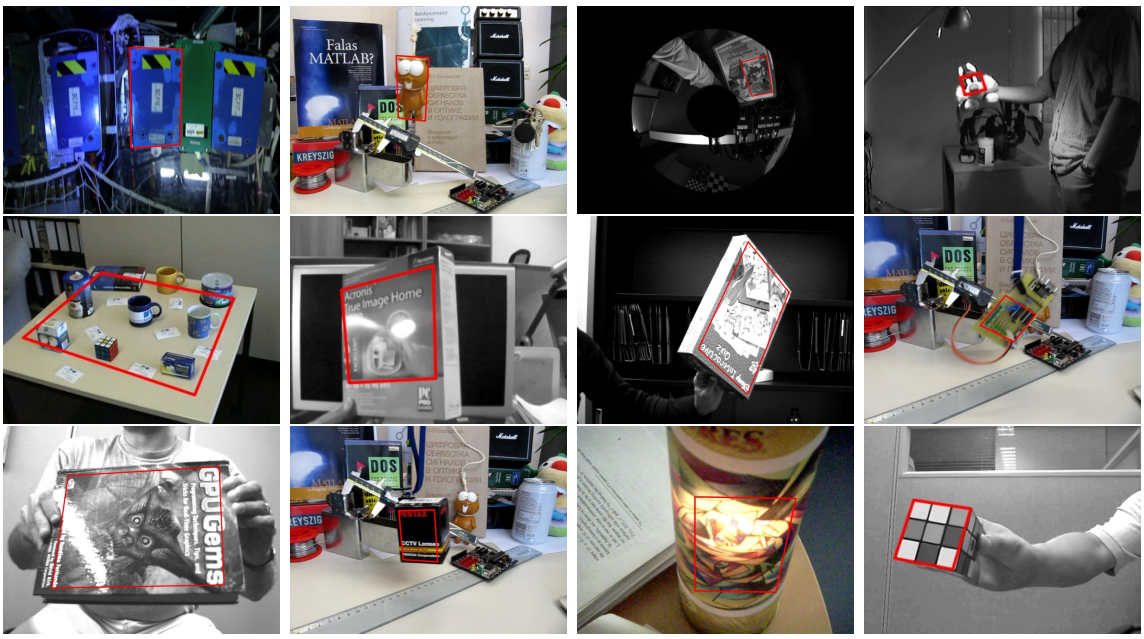


Figure 8.2: Representative frames from sequences in PAMI dataset

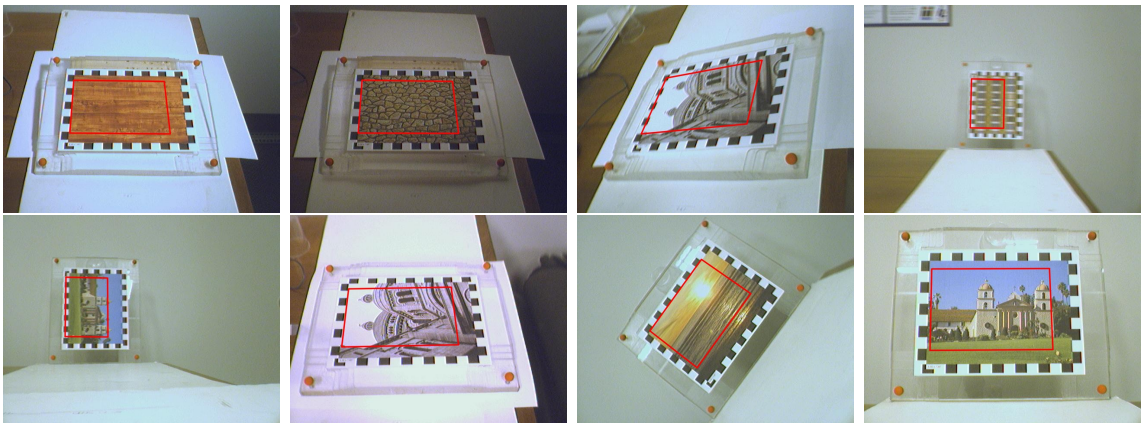


Figure 8.3: Representative frames from sequences in UCSB dataset

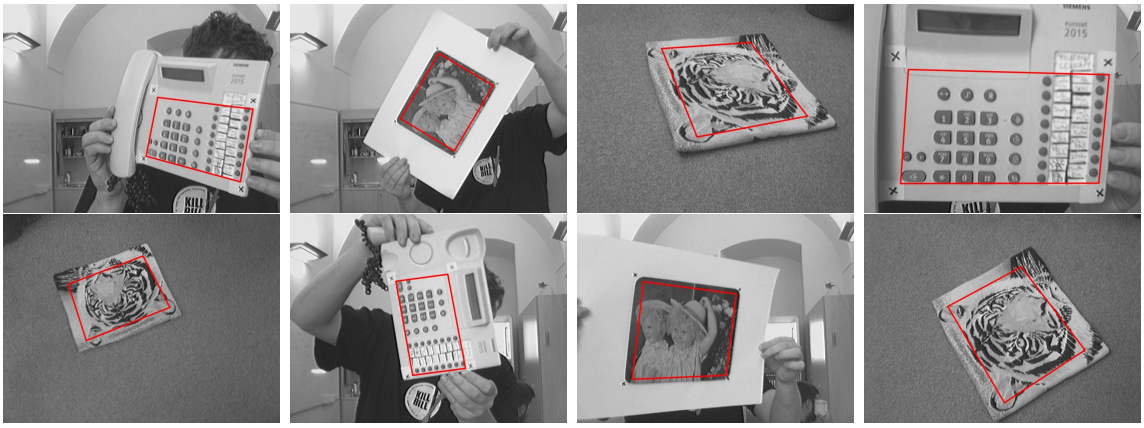


Figure 8.4: Representative frames from sequences in LinTrack dataset

Each of the 25 source images were used for generating 10 different sequences using perturbations drawn from zero mean Gaussian distributions with standard deviation or σ_{syn} varying from 1 to 10. Each such sequence has 400 frames so that there are 10000 frames corresponding to each σ_{syn}.

Two more sets of sequences with higher levels of tracking difficulty were generated to simulate real world conditions better. First one had additive Gaussian noise with $\sigma_{noise} = 10$ added to each pixel while the second one also had illumination changes generated using RBF ILM (Sec. 5.3.3) with a 3×3 grid of control points and perturbations to $\mathbf{p_a}$ drawn from a zero mean Gaussian distribution with $\sigma_{rbf} = 20$ (Fig. 8.6) for both gain and bias. Also, all frames were saved using a JPEG quality factor of 25 (out of 100) so that the resulting compression artifacts can make the sequences

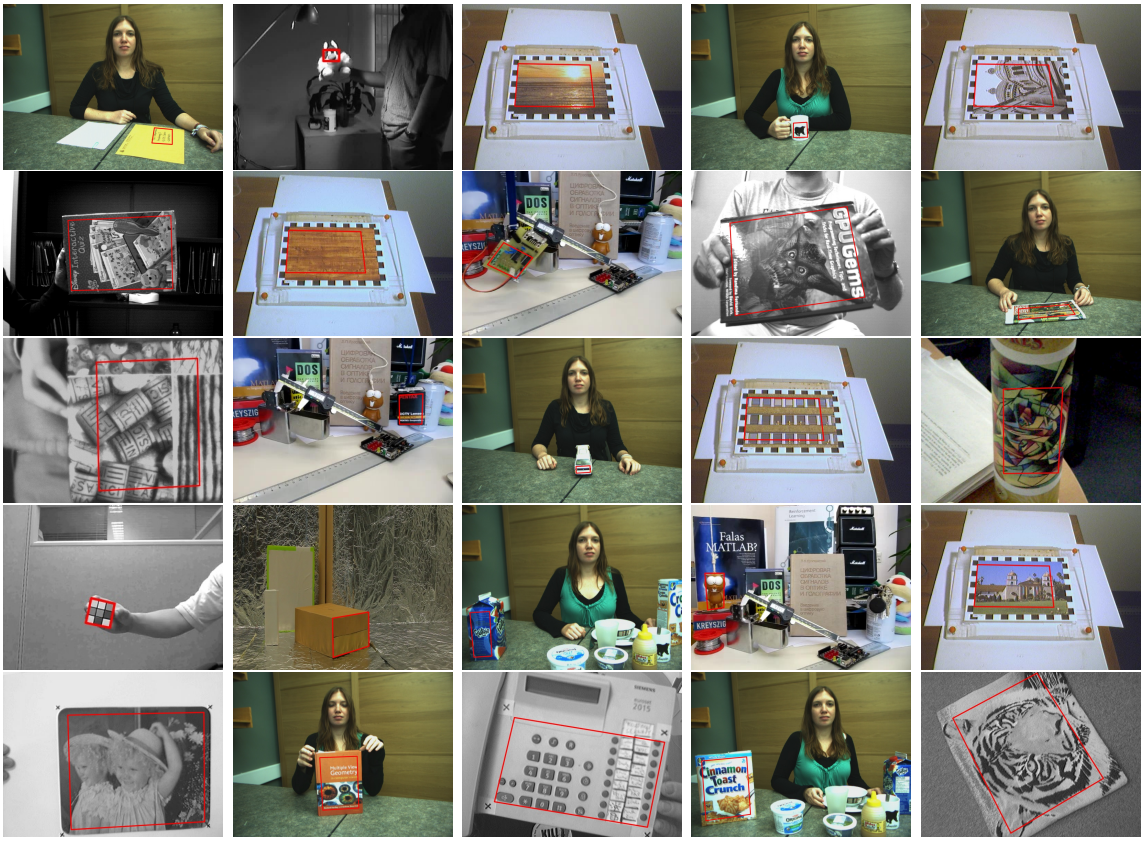


Figure 8.5: Frames used for generating the synthetic datasets

more realistic.

Finally, in order to test different SSMs, synthetic sequences were also generated with translation, isometry, similitude and affine warping using the same DLT based approach as homography but perturbing only 1, 2 or 3 points. The bottom left and bottom right corners along with the top center point were used for affine [25] and the top left and bottom right corners for similitude. Isometry warps were generated by first generating similitude warps and then dropping the scaling factor [15]. For translation, of course, perturbation was applied directly to the centroid and no DLT step was needed.

8.2 Performance Metric

Alignment Error (E_{AL}) [98] has been used as the metric to compare tracking result with the ground truth since it accounts for fine misalignments of pose better than

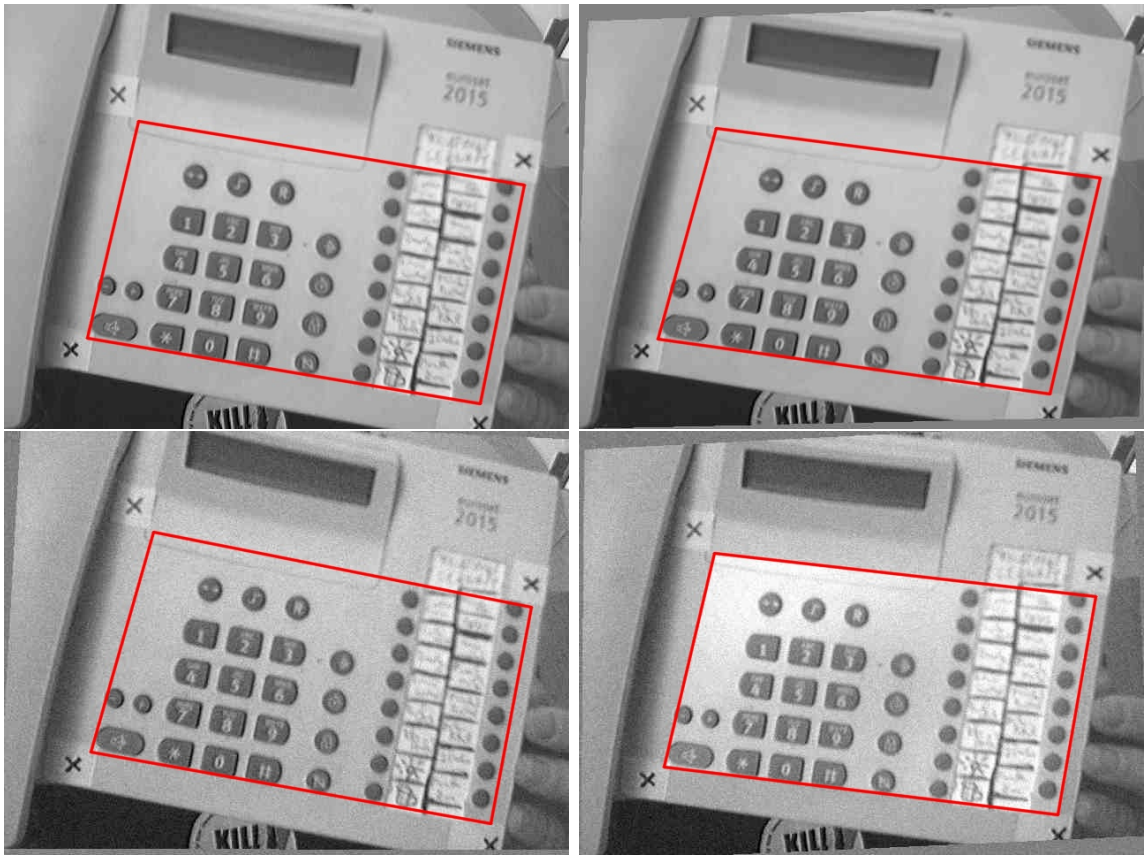


Figure 8.6: Representative frames from the three sets of synthetic sequences. Clockwise from top left - original frame, warped using $\sigma_{syn} = 10$, with additive Gaussian noise and with noise + illumination change generated by RBF. Note that the geometric warping and noise have been applied to the entire image while illumination change is restricted to the object of interest.

other measures like center location error and Jaccard index. It is defined as:

$$E_{AL} = \frac{1}{4}\sum_{k=1}^{4} \|\mathbf{c}_{k(track)} - \mathbf{c}_{k(gt)}\| \tag{8.1}$$

where $\mathbf{c}_{(track)} = [\mathbf{c}_{1(track)}, ..., \mathbf{c}_{4(track)}] \in \mathbb{R}^{2\times 4}$ and $\mathbf{c}_{(gt)}$ are the bounding box corners corresponding to the tracker's location and the ground truth respectively. The overall accuracy of a tracker is measured through its **success rate (SR)** which is defined as the fraction of total frames where E_{AL} is less than a threshold of t_p pixels. Formally, $SR = |S|/|F|$ where $S = \{f^i \in F : E^i_{AL} < t_p\}$, F is the set of all frames and E^i_{AL} is the error in the i^{th} frame f^i.

It is difficult to choose a single error threshold to define tracking success for the large range of scenarios present in the datasets. The SR is thus evaluated for several values of t_p ranging from 1 to 20 and the resulting SR vs. t_p plot is studied to get an overall idea of how precise and robust a tracker is. Values of $t_p < 1$ are not considered as they do not add any useful information to the plot but increase the range of y axis needed to show the curve, thus decreasing the separation between individual curves and making interpretation more difficult. Further, as there are far too many sequences to present results for each, an overall summary of performance is reported instead by averaging the SR over all the sequences in the four datasets. Results for individual datasets are also provided for some important cases in appendix B.

In order to better utilize frames that follow a tracker's first failure in any sequence, trackers are initialized at 10 different evenly spaced frames in each sequence. This also helps to test trackers with a greater variety of initial object pose and appearance rather than only those in the first frame in each sequence. As a result, the SR plots represent accumulated tracking performance over a total of $|F| = 589380$ frames, out of which 106233 are unique (Table 8.1). It should be mentioned here that, in order to process so many frames with the multitude of trackers in limited time, it is necessary to avoid wasting time in updating a tracker after it has clearly suffered an unrecoverable failure in a sequence. A heuristic measure has thus been adopted to detect such failures and stop the tracking once these are encountered. Denoting the image height and width as h and w respectively, an unrecoverable tracking failure is assumed to have occurred when E_{AL} exceeds $\sqrt{h^2 + w^2}$ or the maximum possible distance between two points in the image.

As an alternative measure for tracking robustness, reinitialization tests similar to those in the visual object tracking (VOT) challenge toolkit [8, 13] are also conducted. Here, a tracker is reinitialized after skipping 5 frames every time its E_{AL} exceeds 20 and the number of such failures is counted to get an estimate of the tracker's **failure rate (FR)**. Smaller values of FR indicate higher tracking robustness. Unlike FR, it is not possible to summarize all the information in an SR curve by a single number since different portions of the curve indicate distinct characteristics about the tracker's performance - SR values for smaller t_p indicate precision while those for higher t_p incline more towards robustness. However, the area under this curve, which is equivalent to the average SR for this range of t_p [234], can be taken to indicate the overall "goodness" of tracking performance that it represents. This value is therefore shown in the legends of the SR curves and is also used, along with FR, to rank

different methods in scatter plots (figs. 9.18, 9.25).

Finally, performance over synthetic datasets is measured by plotting the SR computed using $t_p = 2$ for σ_{syn} varying from 1 to 10. This is similar to the method employed in [25] except that $t_p = 1$ was used there but experiments seemed to suggest that $t_p = 2$ generates more meaningful results so this has been used here instead.

8.3 Configuration

A brief summary of the default settings used for the results reported in the next chapter now follows. Several variations on specific settings have also been tested whose details are provided in appendix A.

- A fixed sampling resolution of 50×50 was used irrespective of the tracked object's size. Bilinear interpolation was used for obtaining pixel values at non integral locations.
- Input images were converted to gray scale and smoothed using a Gaussian filter with a 5×5 kernel before being fed to the trackers.
- GD based SMs were allowed to perform a maximum of 30 iterations per frame but only as long as the ℓ^2 norm of the change in bounding box corners in each iteration exceeded 0.0001.
- NN and PF were only allowed a single iteration per frame.
- LM formulation of $\hat{\mathbf{H}}_{self}$ (Sec. 4.1.2.1) was used with GD based SMs for all AMs except RSCV, which performed better with GN (Fig. A.1) (excepting IALK).
- For stochastic and composite SMs based on indirect sampling like LMS and LMES (Sec. 4.2.2), a 10×10 grid of sub patches, each 25×25 pixels in size, was used and each sub patch was tracked by a 2 DOF tracker with a sampling resolution of 25×25.
- Both LMS and RANSAC were allowed a maximum of 1000 iterations and a re-projection threshold of 5 pixels was used for the latter to determine outliers.
- Two different stochastic samplers were used to generate 8 DOF samples for PF - homography sampler with normalized corner perturbations (Sec. 6.5.1)

Table 8.2: Standard deviation values used for homography sampler with normalized corner perturbations. Meanings of σ_d and σ_t are explained in Sec. 6.5.1.

Distribution	σ_d	σ_t
$\mathcal{N}_{hom(1)}$	0.01	0.015
$\mathcal{N}_{hom(2)}$	0.02	0.030
$\mathcal{N}_{hom(3)}$	0.03	0.045
$\mathcal{N}_{hom(4)}$	0.04	0.060
$\mathcal{N}_{hom(5)}$	0.05	0.075

Table 8.3: Standard deviation values used for the SL3 sampler; σ_k is used for perturbing p_k as defined in Sec. 6.6

Distribution	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6	σ_7	σ_8
$\mathcal{N}_{sl3(1)}$	0.015	0.015	0.06	0.015	3.5	3.5	0.0003	0.0003
$\mathcal{N}_{sl3(2)}$	0.015	0.015	0.04	0.015	5	5	0.0002	0.0002
$\mathcal{N}_{sl3(3)}$	0.015	0.015	0.05	0.015	2.5	2.5	0.0004	0.0004
$\mathcal{N}_{sl3(4)}$	0.01	0.01	0.04	0.01	2	2	0.0002	0.0002
$\mathcal{N}_{sl3(5)}$	0.01	0.01	0.03	0.01	10	10	0.0001	0.0001

and SL3 sampler with standard deviations provided in [1]. Five distributions were considered for both samplers along with the Gaussian mixture approach described in Sec. 4.2.1.2. Denoted as $\mathcal{N}_{hom(k)}$ and $\mathcal{N}_{sl3(k)}$ for $1 \leq k \leq 5$, these are given in tables 8.2 and 8.3 respectively. Their mixtures are respectively denoted as $\mathcal{N}_{hom(mix)}$ and $\mathcal{N}_{sl3(mix)}$. The former was used for all results in chapter 9.

- NN was run with 2000 samples generated using $\Sigma_{hom(4)}$. HKMT index type (Sec. 4.2.1.1) was used for searching as being compatible with all AMs and also better performing overall (Sec. A.2.1.1). Parameters for HKMT were left to their default settings in FLANN.

- PF was run with 500 particles using binary multinomial resampling method [177] and first order compositional auto regression dynamic model.

- Several details of the SL3 stochastic sampler, apart from the Gaussian distributions, were also taken from [1] including the algorithms for computing the sample mean and applying first order auto regression. However, some other innovations described there, such as optimal importance sampling and child/parent particles, were not used.

Table 8.4: Multiplicative (α) and additive (β) constants used for computing AM likelihood (Eq. 4.22)

AM	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	**AM**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$	**AM**	$\boldsymbol{\alpha}$	$\boldsymbol{\beta}$
SSD	5	0	SSIM	100	0	NGF	300	1
SCV	50	0	SPSS	1	0	CCRE	100	1
RSCV	50	0	RIU	1000	0	MI	1	0
LSCV	50	0	NCC	50	0	ZNCC	50	0

Table 8.5: Number of histogram bins used with MI and CCRE

	FCLK	ICLK	ESM	FALK	IALK	NN	PF	NNIC	PFFC
MI	10	10	10	10	10	24	24	12	12
CCRE	16	12	16	16	16	24	16	12	16

- The 3 layer cascaded configurations of NN and PF (NN3 in Fig. A.6 and PF3 in Fig. A.12) used $\mathcal{N}_{hom(4)}$, $\mathcal{N}_{hom(2)}$ and $\mathcal{N}_{hom(1)}$ respectively in their first, second and third layers. 5 layer NN (NN5 in Fig. A.6) used all 5 distributions in the reverse order, i.e. the first layer had $\mathcal{N}_{hom(5)}$ and the last one $\mathcal{N}_{hom(1)}$.

- The results comparing SSMs with PF and NN (Fig. 9.27) were generated using the same sampling technique as employed for generating the synthetic datasets (Sec. 8.1.1) except that, instead of using a single distribution, the Gaussian mixture technique (Sec. 4.2.1.2) was utilized to combine 5 distributions with σ varying from 1 to 5.

- Likelihood values for PF were computed using the α and β values given in Table 8.4. These were estimated by evaluating a range of values (Sec. A.3.1) and choosing the best performing one. Eq. 4.22 was used for computing likelihood for all AMs except ZNCC where a slightly modified version taken from [99] was used instead.

- Number of histogram bins for MI and CCRE used with the different SMs are given in Table 8.5. These were chosen experimentally too (Sec. A.3.2). All histogram bins were pre-seeded with 10 to avoid empty bins and the resultant numerical instability [190]. Also, the partition of unity constraint [93] was enforced by normalizing all pixel values to the range $[1, n_b - 2]$ where n_b is the number of bins.

- SCV, RSCV and LSCV were used with 64 histogram bins instead of the conventional 256 [95, 5] for stochastic SMs where speed is crucial. Experiments indicated that the two perform similarly (Fig. A.20) with the former being significantly faster to compute.
- As in [116], SSIM parameters were computed as $C_1 = (K_1 L)^2$ and $C_2 = (K_2 L)^2$ with $K_1 = 0.01$, $K_2 = 0.03$ and $L = 255$.
- OLTs (Sec. 9.3) were run using default settings provided by the respective authors in their C/C++ implementations which have been integrated into MTF.
- Speed tests were performed on a 4 GHz Intel Core i7-4790K machine with 32 GB of RAM.

8.4 Summary

This chapter presented details of the real world and synthetic datasets as well as the performance measures used for evaluating the trackers in this study. It also provided a summary of the main configuration settings used for the modules that constitute the trackers.

Chapter 9

Results and Analysis

The results presented in this chapter are divided into three sections corresponding to the three sub modules. In each of these sections, results are provided to evaluate the performance of all methods considered for the respective sub module against several combinations of methods for one or more of the other sub modules. Specifically, section 9.1 compares the performance of different AMs with each SM while section 9.2 presents mostly these same results but from the perspective of comparing different SMs with each AM. Both these sections only consider homography as the SSM since similar comparative results were obtained with other SSMs too. Section 9.3, on the other hand, compares different SSMs with each other using several combinations of AMs and SMs. This is also where 2 DOF RBTs are compared with OLTs.

This chapter only includes results for the optimal configuration of each module as specified in Sec. 8.3. Comparisons between other parameter settings have been deferred to appendix A. Results for individual datasets are likewise in appendix B and only the overall performance with all datasets combined is presented here. The first two sections in this chapter end with a **summary of important results** (Sec. 9.1.3 and 9.2.4) so readers not interested in the detailed analysis of individual methods can directly skip there. A concise summary of results from all three sections is also included in Sec. 10.1 for readers only interested in the most important conclusions.

9.1 Search Methods

9.1.1 Gradient Descent Search

Fig. 9.1 and 9.2 show the performance of GD SMs over the real world datasets while figs. 9.3 - 9.7 show it over the synthetic datasets. It may be recalled from Sec. 8.2

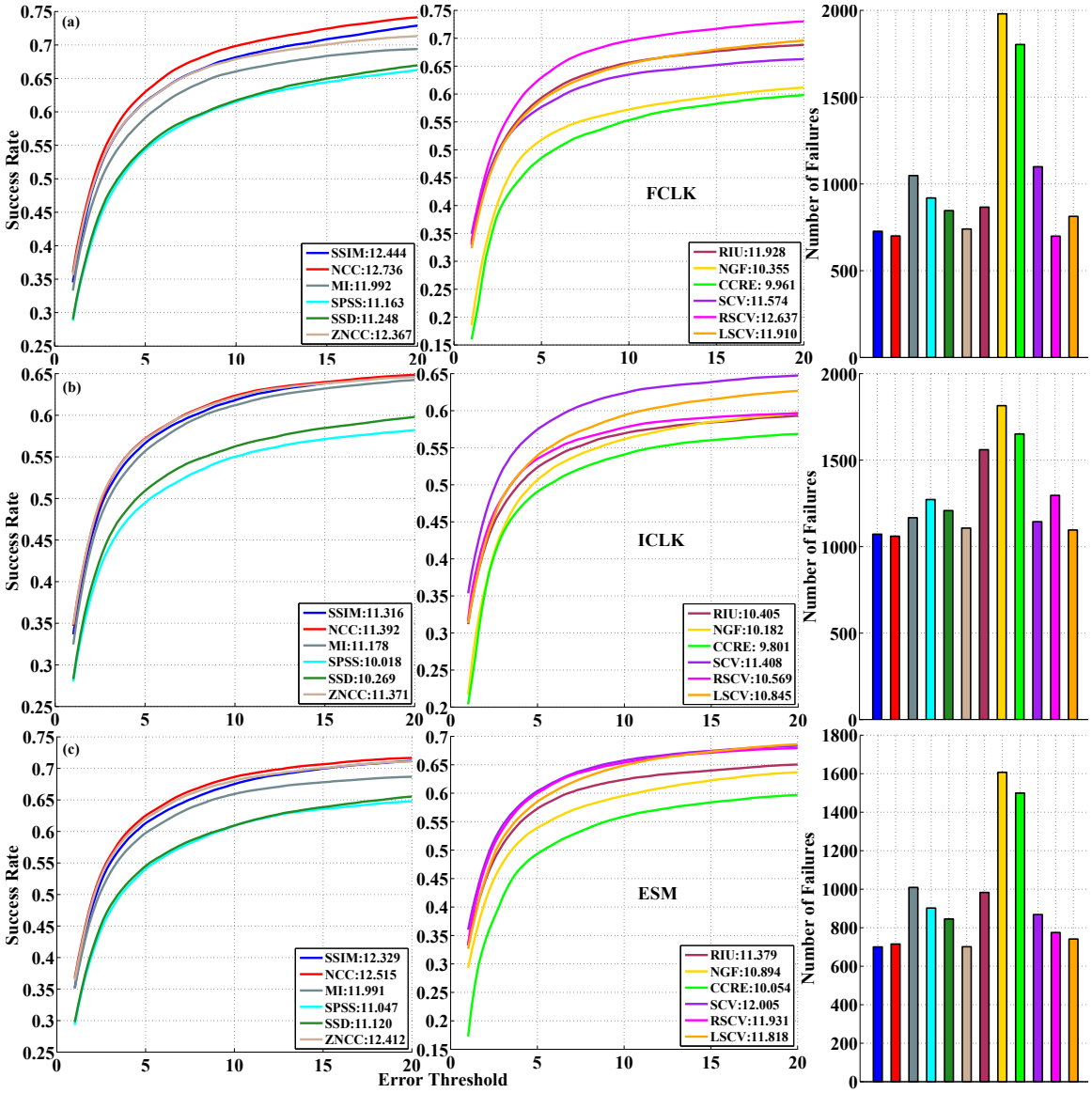


Figure 9.1: Performance of AMs with FCLK, ICLK and ESM

that the values shown in the legend of each SR plot are the areas under the respective curves.

Following are some observations that can be made from these results:

- FCLK, FALK and ESM perform best with NCC followed by RSCV, SSIM and ZNCC. The superiority of NCC is worth noting considering that it is a fairly simple and long known image similarity metric but still manages to outperform newer, more sophisticated and computationally expensive AMs like MI, CCRE

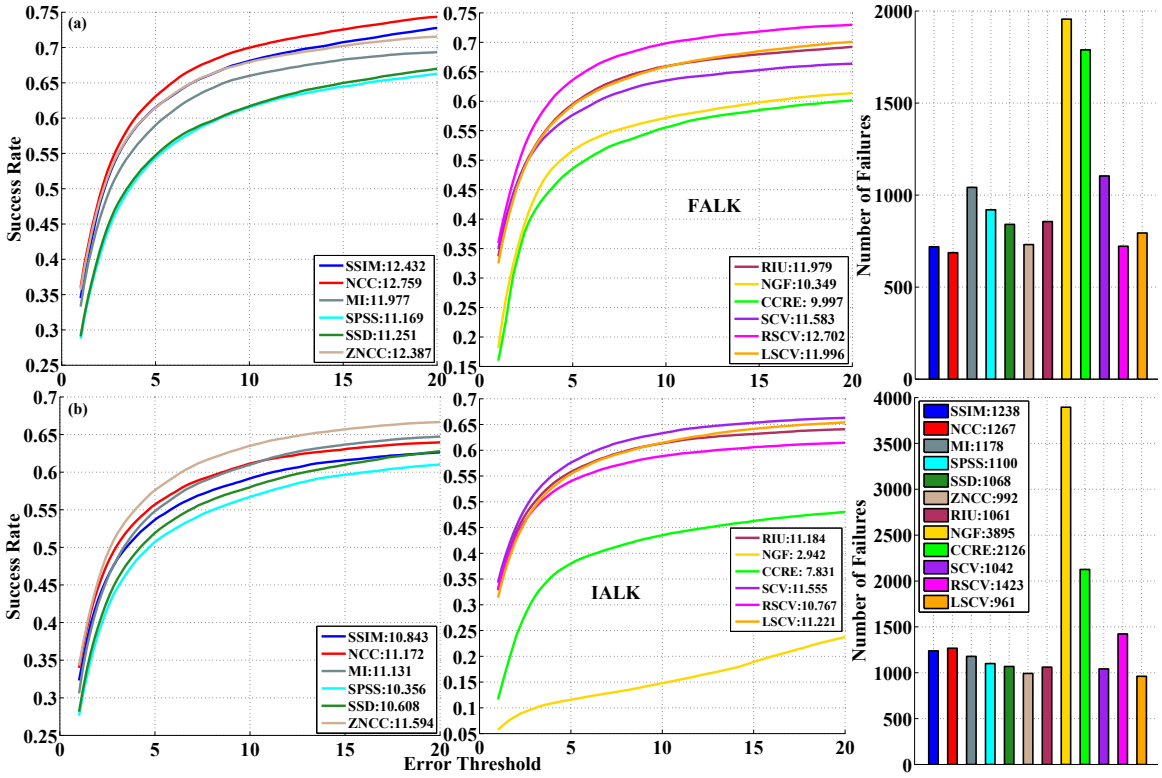


Figure 9.2: Performance of AMs with FALK and IALK

and NGF. The near identical performance of NCC and SSIM is probably to be expected though, as their functional forms too are quite similar.

- CCRE and NGF are the two worst performing AMs with all SMs. CCRE is slightly better in terms of FR and NGF with SR except with IALK which does not work at all with this AM. This is rather surprising as these are also some of the most complex and computationally expensive AMs. This can be best explained by the relatively narrow basin of convergence that these AMs provide (Fig. 5.9 and 5.13).

- It is worth noting that NGF was introduced as an improved alternative to MI that, owing to its squared residual formulation that resembles that of SSD, did not suffer from the latter's shortcomings including narrow convergence region [111]. However, these claims do not seem to hold in practice, at least for tracking.

- RSCV outperforms LSCV with FCLK, FALK and ESM while SCV does so

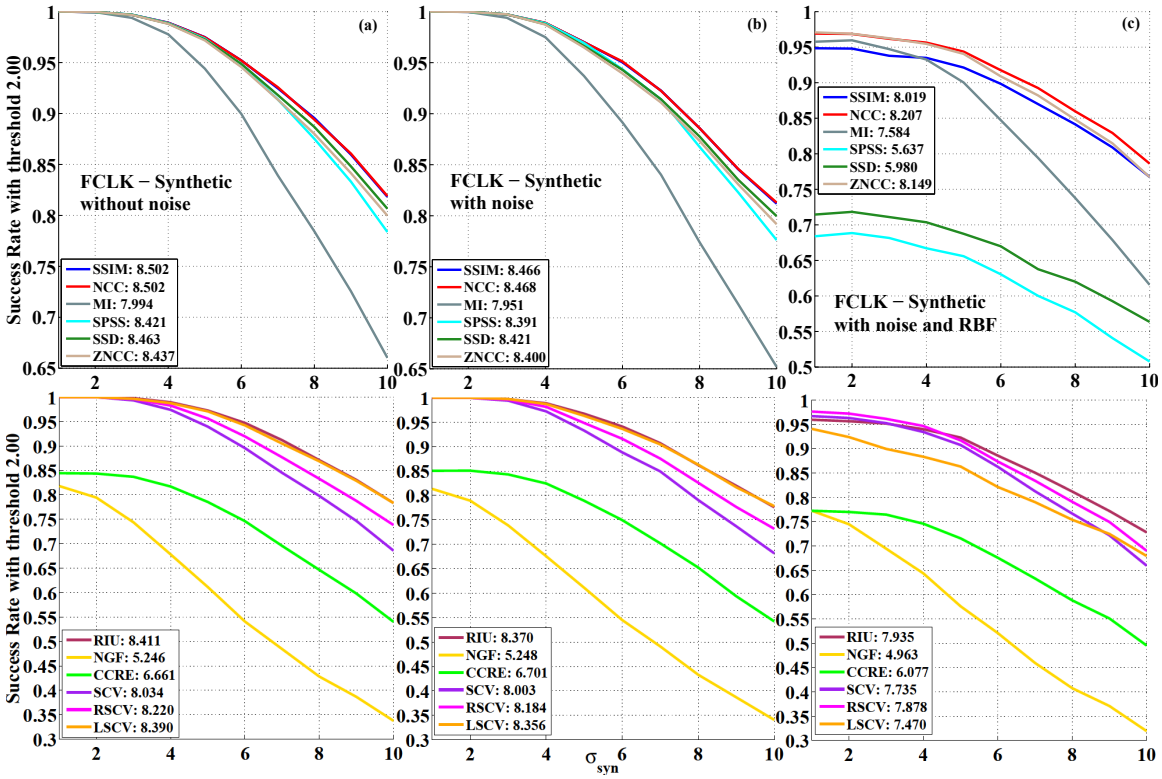


Figure 9.3: Performance of FCLK over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

with ICLK and IALK even though LSCV is a newer and improved version that supposedly handles localized illumination changes better. Also, this observation holds true for all datasets combined as well as for each individual one (figs. B.1 - B.3) including PAMI that does contain many sequences with such lighting changes.

- A clear role reversal can be observed between SCV and RSCV in their relative performance with the forward and inverse SMs - SCV outperforms RSCV with the inverse SMs by almost the same amount as it is outperformed by RSCV with the forward SMs. Also, the two AMs perform almost identically with ESM that combines both the forward and inverse methods (Sec. 4.1) - SCV is slightly better in SR and RSCV in FR. This can be explained by the fact that SCV replaces $\mathbf{I_0}$ with the expected patch $\mathbb{E}(\mathbf{I_t}|\mathbf{I_0})$ while RSCV replaces $\mathbf{I_t}$ with $\mathbb{E}(\mathbf{I_0}|\mathbf{I_t})$. It can be recalled from Sec. 5.1.2 and 5.1.3 that the Jacobians used by these AMs are only approximations since the expressions for the expectation

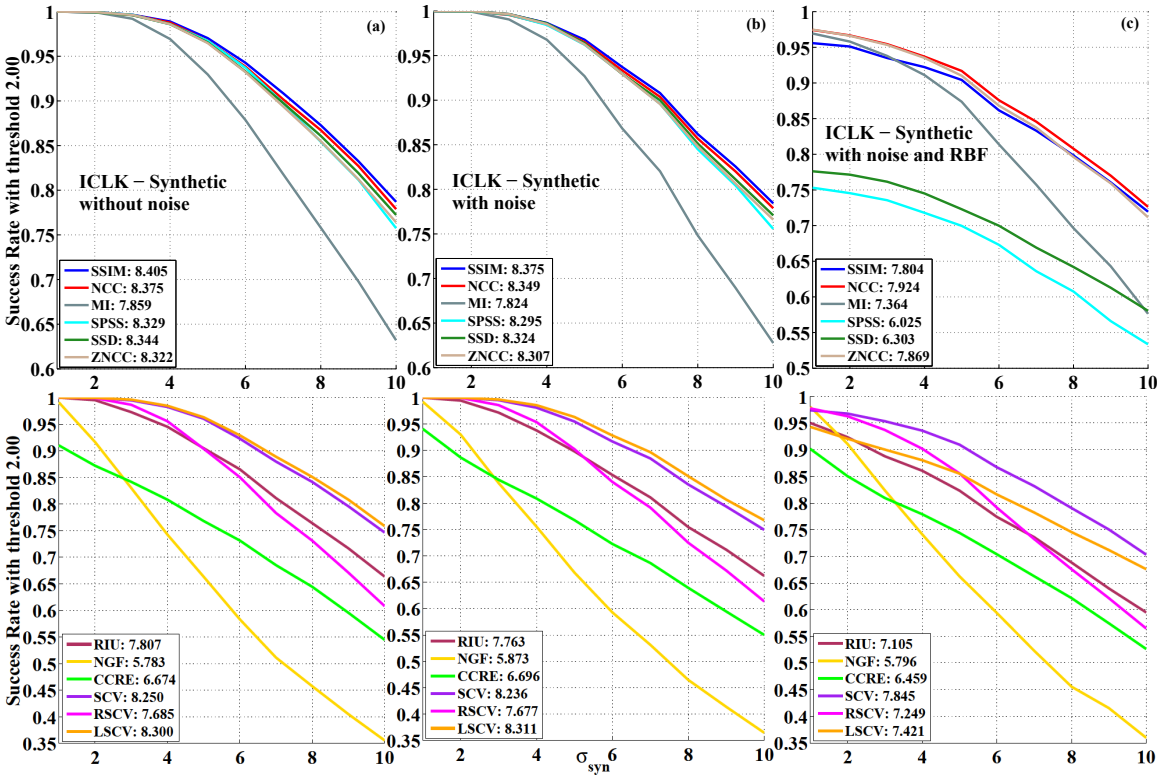


Figure 9.4: Performance of ICLK over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

operator (Eqs. 5.5, 5.10) are not differentiable. Combined with the fact that the inverse and forward SMs compute the image gradient ∇I from I_0 and I_t respectively, it seems that approximation in Eq. 5.2 is more accurate when the expectation operator is applied to the same image from which the gradient is computed.

- SPSS and SSD perform almost identically and only slightly better than CCRE and NGF which is to be expected as both are simple pixel wise measures.

- Though ZNCC does perform similarly to NCC, the latter seems to be slightly better overall, especially with the forward SMs. Assuming that minimizing the SSD between normalized patches is equivalent to maximizing the dot product between them [192], the opposite would be expected due to the wider convergence region of ZNCC. It would thus appear that this equivalence does not hold in practice.

- For synthetic datasets without illumination change (Fig. 9.3 - 9.6 (a),(b)), it

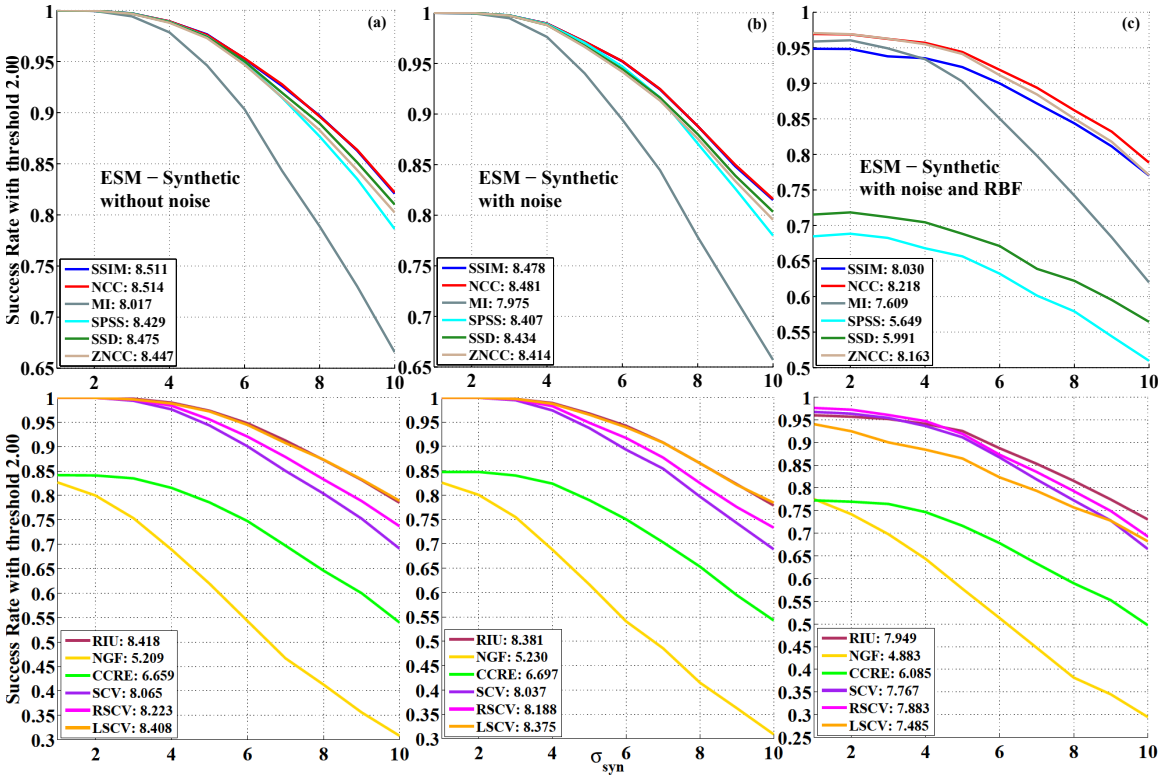


Figure 9.5: Performance of ESM over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

seems that the performance of AMs for all SMs except IALK depends mainly on their respective radii of convergence. As a result, simpler AMs like SSD and SPSS perform at par with more sophisticated ones like NCC and SSIM and the only ones that fare poorly here - MI, CCRE and NGF - are those that are known to be deficient in this respect (figs. 5.8, 5.9 and 5.13).

- The introduction of noise does not impact the performance of any AM significantly with any of the SMs, indicating that all of them, even SSD, are well capable of handling noise in real sequences.

- SCV and RSCV too perform notably worse than other AMs with three of the SMs - FCLK, FALK and ESM - while LSCV does not. This confirms the observation made in [5] that the affine mapping in LSCV (Eqs. 5.12, 5.13) helps to increase the radius of convergence compared to the non linear mapping in SCV and RSCV.

- The role reversal in the relative performance of SCV and RSCV between the

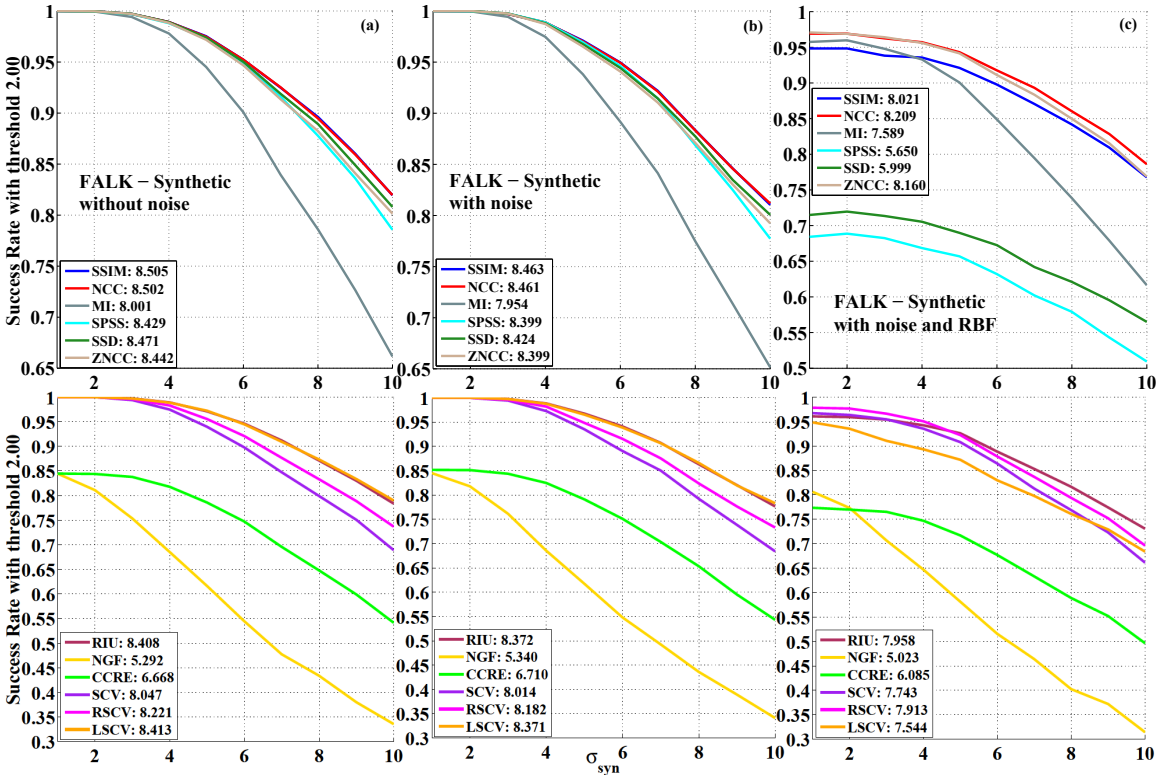


Figure 9.6: Performance of FALK over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

inverse and forward SMs is present in synthetic tests too.

- IALK seems to favor L2 models over robust ones in these tests (Fig. 9.7 (a), (b)). This behavior can be observed to a lesser extent in the real world datasets too as this is the only SM where ZNCC outperforms NCC and other robust AMs too perform relatively poorly with it. This might be due to the generalization of Gauss Newton method for robust AMs using $\hat{\mathbf{H}}_{self}$ being less valid with additive SMs [93] though this explanation is somewhat undermined by the fact that FALK performs just as well as FCLK with these AMs. It is possible that the combined effect of the approximation in Eq. 4.9 and the one implicit in $\hat{\mathbf{H}}_{self}$ is responsible for this poor performance.

- NCC, ZNCC and SSIM prove to be the best equipped AMs to handle synthetic illumination changes (Fig. 9.3 - 9.7 (c)) with FCLK, FALK and ESM. NGF is the worst one here too, followed closely by SSD and SPSS. MI and CCRE outperform the latter but, rather surprisingly, are still notably worse than the

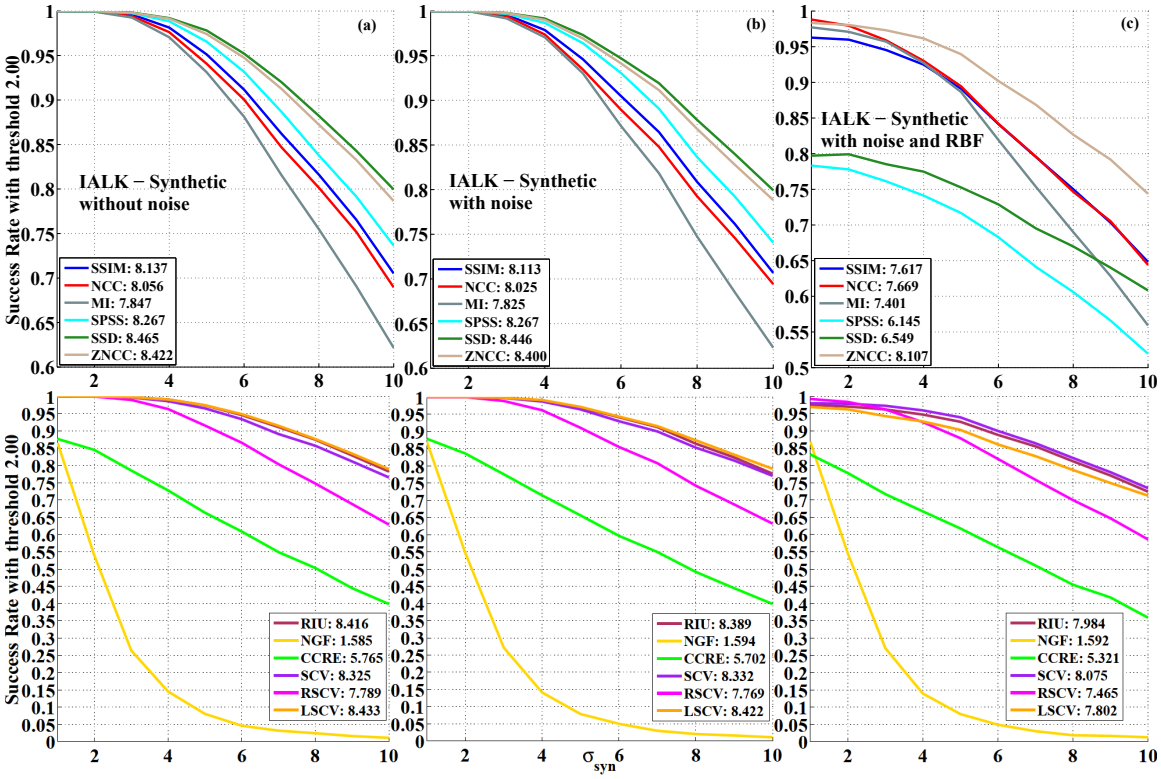


Figure 9.7: Performance of IALK over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

other AMs. For CCRE and NGF, it seems that the somewhat unstable nature of these AMs prevents them from getting high SR with such a small t_p. MI, on the other hand, shows a sharp decline after σ_{hom} exceeds 4 which can probably be attributed more to its narrow convergence region dominating its illumination invariance. Finally, LSCV fares slightly worse than SCV and RSCV here even though these sequences were specifically generated to have smoothly varying localized illumination changes.

- RIU perform surprisingly well on synthetic datasets - both with and without illumination changes - for all SMs except ICLK. It even works well with IALK and is the only robust AM to do so. This is rather at odds with its performance on real world datasets where it fared only marginally better than SSD.

- Both relative and absolute performance of AMs on synthetic datasets with ESM (Fig. 9.5) is similar to that with FCLK (Fig. 9.3). This indicates that the higher convergence rate of ESM that was claimed and theoretically proven to first order

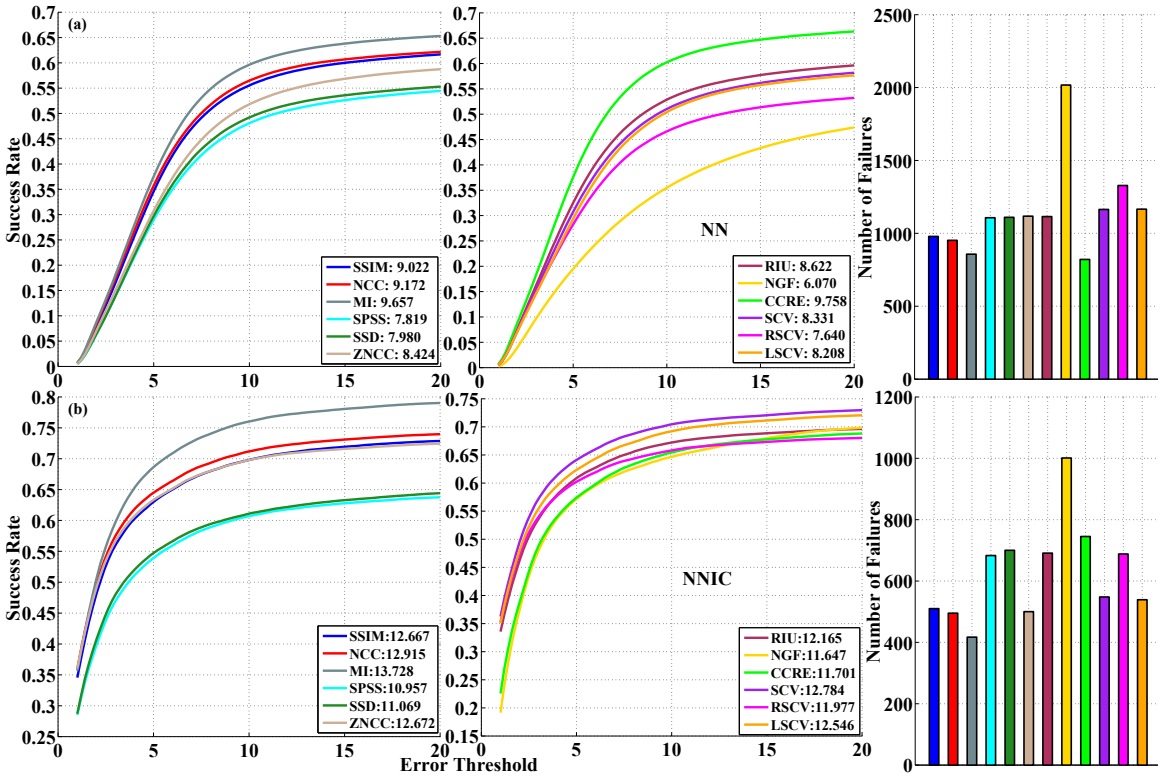


Figure 9.8: Performance of AMs with NN and NNIC

in [76] does not have much significance in practice even with synthetic data.

9.1.2 Stochastic and Composite Search

This section will present results for both stochastic SMs and composite ones created by combining these in cascade with GD based SMs (Sec. 4.3). Note that the synthetic plots for stochastic SMs have been generated using $t_p = 4$ pixels instead of the usual 2 to compensate for the inherently imprecise nature of these SMs.

9.1.2.1 NN

Fig. 9.8 shows the performance of NN and NNIC over the real world datasets while figs. 9.9 and 9.10 show it for the synthetic datasets. Following are some notable observations:

- CCRE is the best performing AM with NN followed by MI in real world tests (Fig. 9.8 (a)). This indicates that the poor performance of CCRE with GD

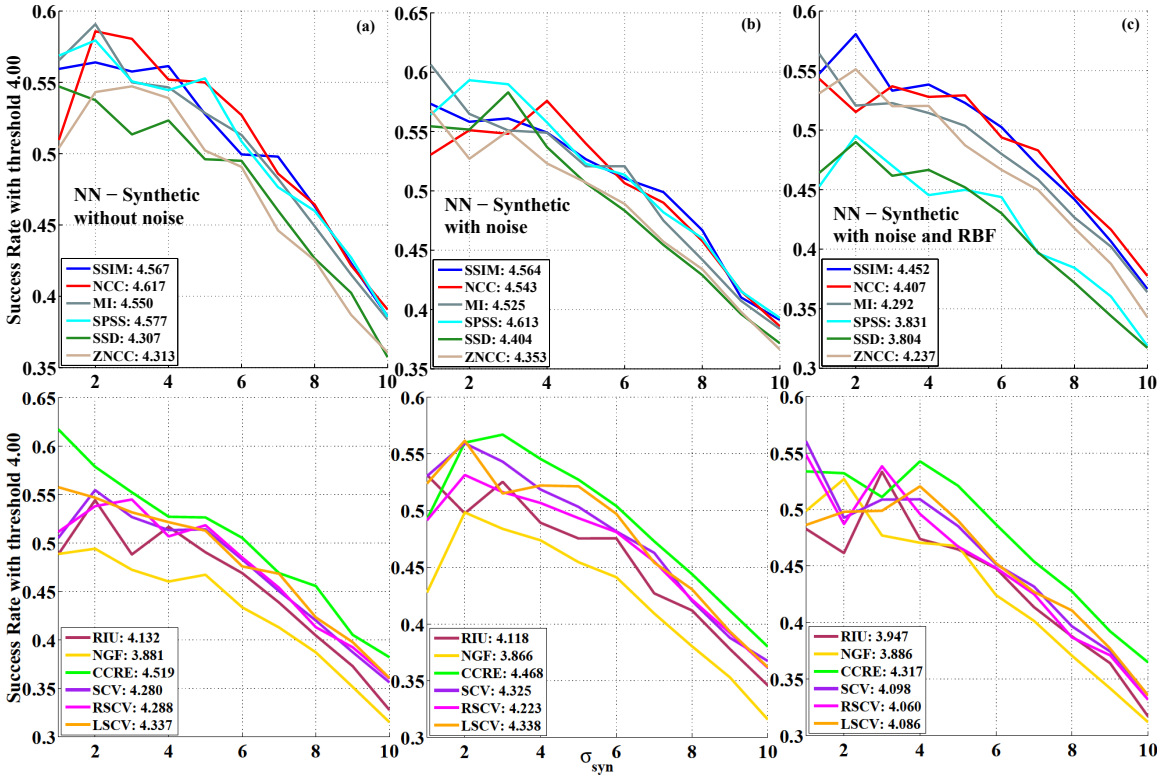


Figure 9.9: Performance of AMs using NN over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

based SMs is more due to its narrow basin of convergence rather than any inherent shortcoming in the similarity measure itself. The same applies to MI.

- When used with NNIC (Fig. 9.8 (b)), however, CCRE reverts to being one of the worst performing AMs. It seems that the radius of convergence of CCRE is too narrow for ICLK to converge even when a better starting point is provided by NN.

- MI does not appear to suffer from this drawback and so ends up being the best performing AM with NNIC and by a significant margin too. This is consistent with the better performance of MI with ICLK - apparently its radius of convergence is large enough for the coarse location provided by NN to lie within it more often than not.

- SSIM and NCC perform fairly well here too - being just behind MI and CCRE.

- NGF continues to be the worst performing AM with both NN and NNIC sug-

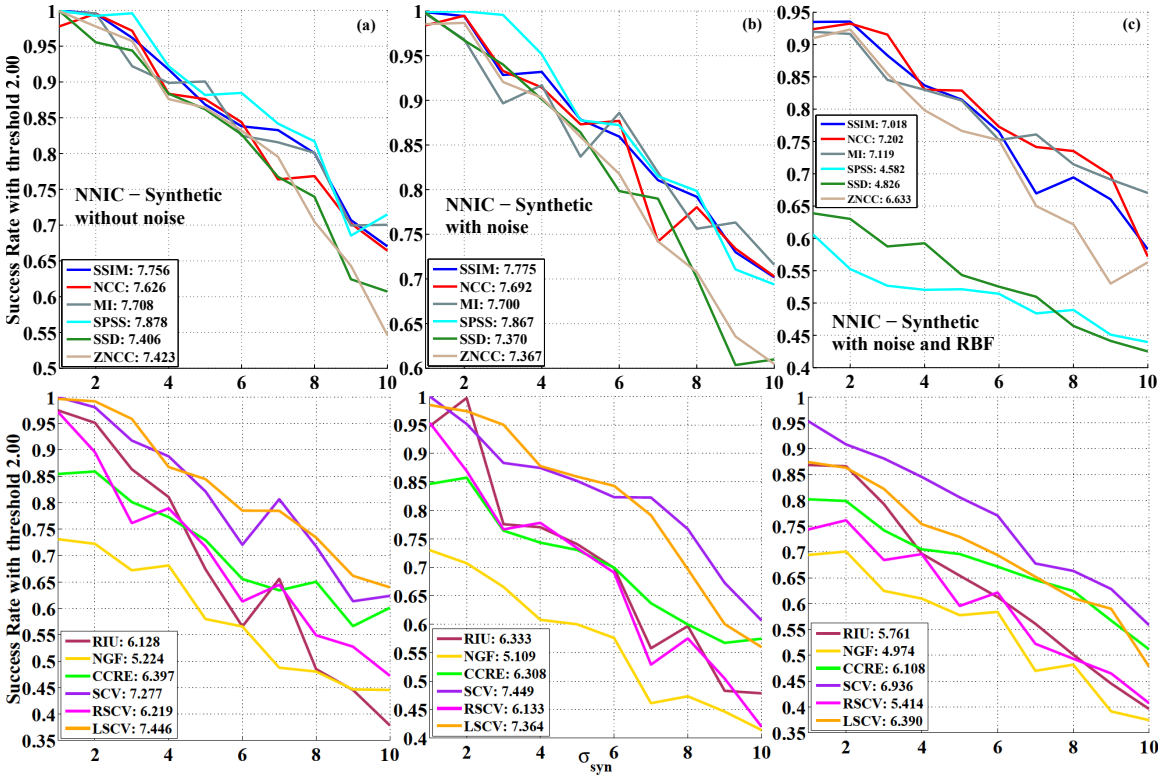


Figure 9.10: Performance of AMs using NNIC over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

gesting that, unlike MI and CCRE, this measure is intrinsically unsuitable for tracking applications.

- SCV outperforms RSCV with both NN and NNIC which is consistent with the observation made in the last section that it works well with SMs that search I_0 for the optimal warp.

- Relative performance between AMs in synthetic tests is similar to that on the real datasets though SSIM and NCC appear to have a slight edge here.

- The jagged nature of the synthetic plots for NN (Fig. 9.9) indicates the drawback of using a single distribution with stochastic SMs to track different types of motion. Since all samples were generated using $\mathcal{N}_{hom(4)}$, these agreed better with sequences corresponding to certain σ_{syn} values than others. This in turn caused NN to provide higher SR with these σ_{syn} even if they are larger in absolute terms than those that do not fit as well with the samples.

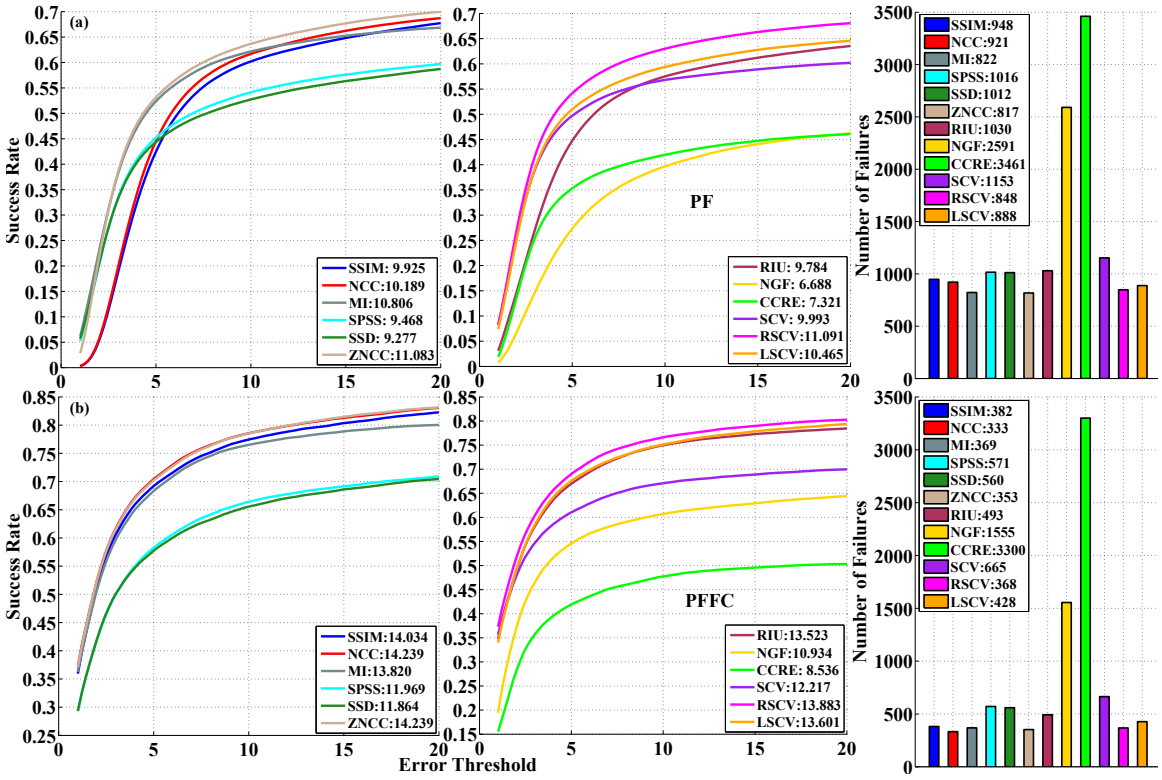


Figure 9.11: Performance of AMs with PF and PFFC

- NNIC exhibits uneven performance in synthetic tests too (Fig. 9.10) though the actual SR values are significantly higher here. This is a bit surprising as the ICLK layer would have been expected to compensate for the dependence of NN on the agreement between its samples and σ_{syn}. It seems, however, that NN does have a significant impact on the convergence rate of NNIC in the frame to frame tracking that was used for synthetic tests. This might partially be due to NN providing a *worse* starting point for some σ_{syn} values and thus causing ICLK to converge to a poorer optimum that it would have otherwise.

9.1.2.2 PF

Fig. 9.11 presents the performance of AMs with PF and PFFC over the real datasets while figs. 9.12 and 9.13 show it for the synthetic ones. It may be recalled from Sec. 8.3 that the homography sampler with mixture distribution was used for all results. Following observations can be made from these results:

- RSCV and ZNCC are the best performing AMs with PF (Fig. 9.11 (a)) followed

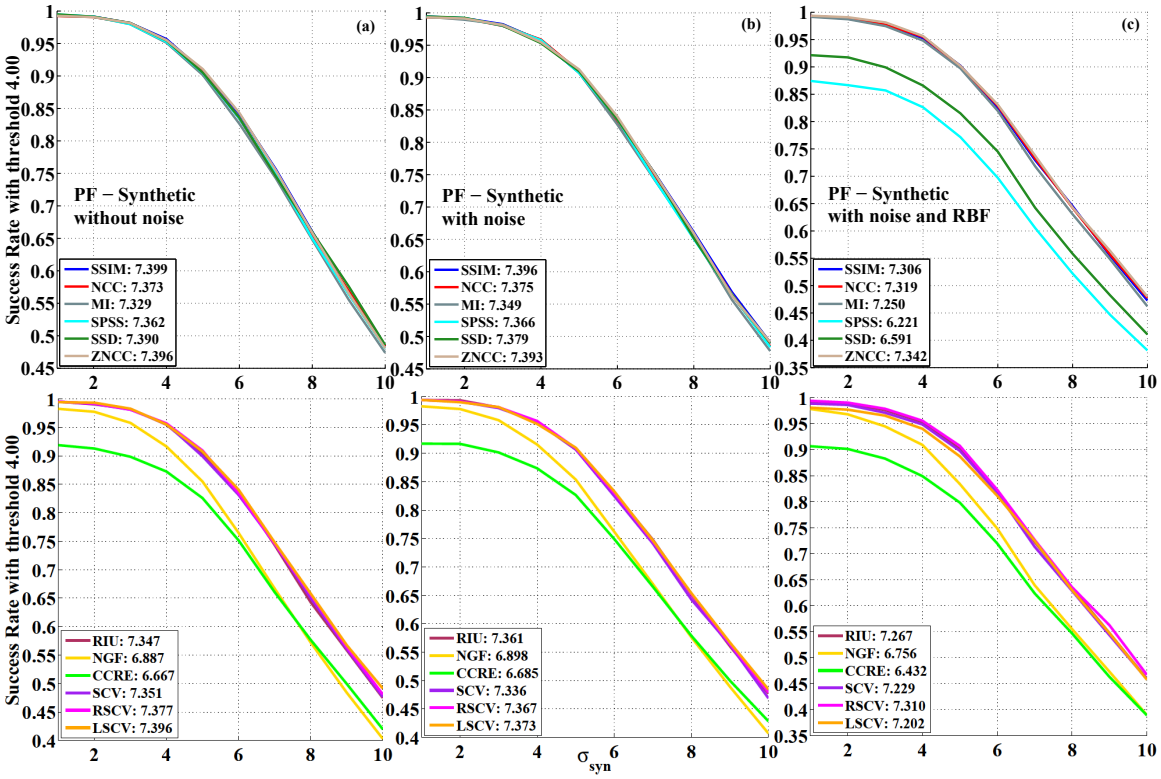


Figure 9.12: Performance of PF over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

by MI.

- RSCV continues the trend of performing well with forward SMs just as SCV had done with NN, thus reaffirming that the two should respectively be used with forward and inverse SMs.

- MI too continues to perform well with stochastic SMs to add weight to the hypothesis that its relatively poor performance with GD based SMs is mainly due to its narrow convergence region and not any inherent shortcoming as a similarity measure.

- ZNCC may partially owe its good performance to the fine tuned likelihood function and associated constants that were taken from [99]. As shown in Fig. A.3.1, these have a very significant impact on the performance of PF and it is possible that many of the AMs might be gotten to perform better by more extensive fine tuning.

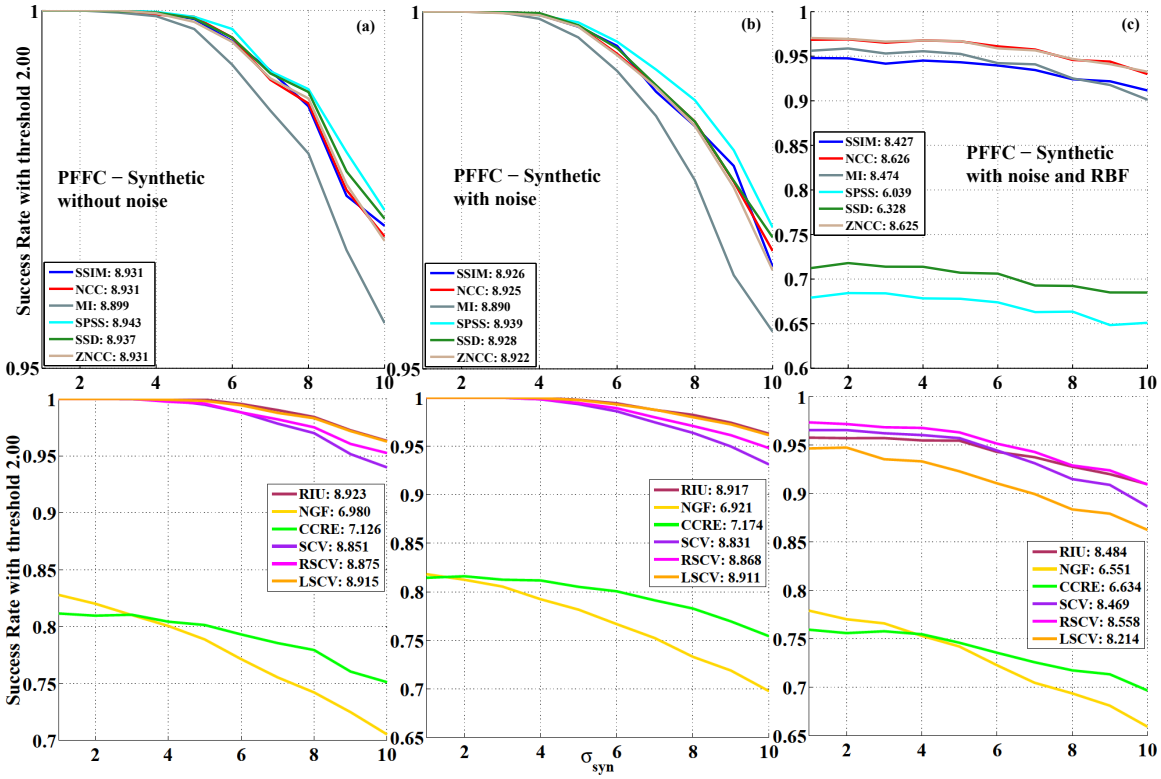


Figure 9.13: Performance of PFFC over synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

- The unexpectedly poor performance of CCRE in particular might be due to insufficiently optimized likelihood constants though the best ones that could be obtained given the time constraints were used.

- NGF is the worst performing AM with PF too which reiterates that it is not a suitable AM for tracking applications.

- NCC and SSIM manage to outperform RSCV and MI with PFFC (Fig. 9.11 (b)) though not by a significant margin, showing, as with NNIC, a slight domination of the FCLK layer in determining the performance of PFFC.

- CCRE performs even worse than NGF with PFFC which is consistent with its poorer performance with FCLK too. Since the PF layer does not perform well, its results provide even worse starting points for FCLK than the locations from the last frame. As a result CCRE actually performs worse with PFFC than FCLK (Fig. 9.1 (a)). This is one of the main downsides of cascade tracking -

poor quality results from the first layer can end up causing the second layer to perform even worse than it would by itself.

- PF does not exhibit the uneven character of NN in its synthetic results (Fig. 9.12) since the Gaussian mixture distribution $\mathcal{N}_{hom(mix)}$ used by it can generate good samples for both small and large motions. The use of auto regressive dynamic model also helps to gradually adapt samples to any given σ_{syn} as more frames from the corresponding sequence are processed.

- PF synthetic results show significantly smaller variation between AMs than all SMs considered so far. This is especially true for sequences without illumination change where NGF and CCRE are the only ones that stand out as being worse than the rest. This agrees with the hypothesis that the limiting factor in the performance of PF with many AMs is the likelihood function and the associated constants (Sec. A.3.1) rather than the AM itself.

- PFFC manages to achieve near constant performance over all σ_{syn} (Fig. 9.13). Note that the y axes in the first two plots in Fig. 9.13 only extend to 0.95 so that the decrease in SR with increasing σ_{syn} is much smaller that it appears. Even CCRE and NGF, though having less SR than others, show relatively constant plots. This demonstrates the great utility of providing GD based SMs with good starting points notwithstanding the drawback mentioned earlier.

- The relative performance between AMs on synthetic datasets is roughly the same as on the real ones except that SPSS appears to be both the best performer without illumination change and the worst one with it. The performance margin is very small though, so this disparity is probably not significant.

9.1.2.3 LMS

As seen in Sec. 4.2.2, the two stochastic SMs based on indirect sampling - LMS and RANSAC - share a significant portion of their approaches. Both methods involve estimating the corresponding locations of a grid of points between two consecutive frames by tracking a small sub patch around each, using random subsets of these points to estimate the best fit warp using DLT and finally retaining the warp that maximizes some measure of consistency when applied to all the points. They only differ in the criteria they adopt for evaluating this consistency. Both SMs also behave very similar in practice as far as the relative performance of AMs is concerned.

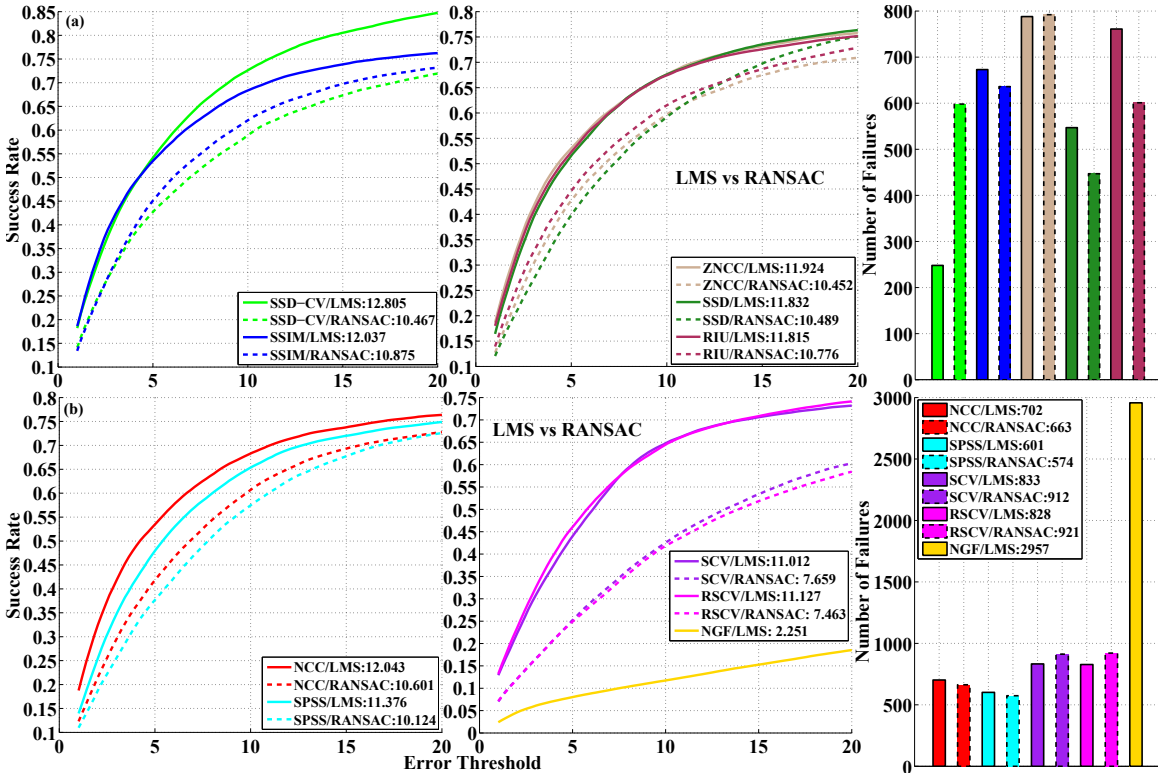


Figure 9.14: Performance of AMs with LMS and RANSAC

Their results are therefore not presented in separate sections; instead, they are first compared with each other and only the better one is used for the composite SM.

Further, it turns out that four of the AMs considered in this study - CCRE, MI, LSCV and NGF- are not suitable for tracking the grid of points, both because they are far too slow (Fig. 9.19) for a set of these to be run simultaneously in real time and their formulations do not work well with small sub patches. MI and CCRE need a large and well textured region to compute reasonably discriminative joint histograms and LSCV needs a large enough patch to compute multiple joint histograms from corresponding sub patches. NGF too requires more texture information for its gradient based approach to work. As a result, these AMs are not included here, though NGF is shown in Fig. 9.14 to demonstrate the poor performance of these AMs.

Finally, two different variants of SSD are considered here - the standard one implemented in MTF and one that is provided by the `calcOpticalFlowPyrLK` function of OpenCV [235] and the latter is referred to as **SSD-CV**. According to its documentation, this function implements a pyramidal version of the original LK optical flow algorithm [19] where the gradient is computed from the first image. It should

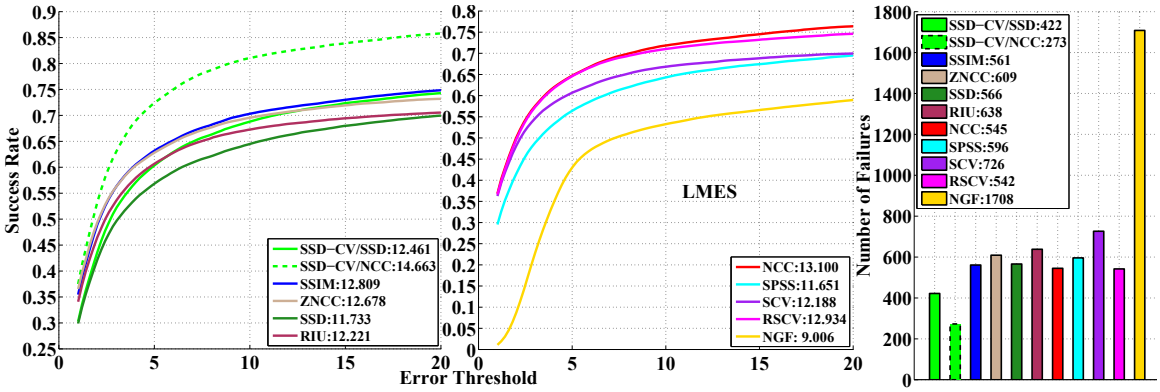


Figure 9.15: Performance of AMs with LMES. SSD-CV/SSD and SSD-CV/NCC refer to LMES trackers with SSD and NCC respectively being the AMs in the second layer and SSD-CV in the first one.

therefore be identical to a pyramidal MTF tracker that combines ICLK with SSD and translation. However, for some reason that has not been ascertained yet, the OpenCV version consistently outperformed this tracker as well as all other AMs combined not only with ICLK but FCLK and ESM (Sec. A.2.3.3). In fact, to the best of the author's knowledge, its performance establishes a new state of the art in high DOF tracking, at least over the tested datasets. It seems that the OpenCV function contains some optimizations not described in the paper in the way it implements its pyramidal tracking. However, time constraints, coupled with its very dense implementation that contains assembly level code in places, have prevented this hypothesis from being confirmed at the time of this writing.

Fig. 9.14 and 9.15 respectively present the performance LMS/RANSAC and LMES over the real datasets while figs. 9.16 and 9.17 do so for the synthetic ones. Following observations can be made therein:

- LMS performs notably better than RANSAC with all AMs in terms of SR. This agrees with the results obtained in the context of image mosaicing too [182]. LMS has the added advantage of not requiring any parameters to be tuned unlike RANSAC which needs a threshold to determine inliers. This might also be one of the reasons behind the latter's poorer performance as it is unlikely that a single threshold will be optimal under all scenarios. These results were generated using a re projection threshold of 5 pixels though experiments with thresholds up to 50 did not present any significant difference in performance. It is likely, however, that sequence specific settings will be needed to get optimal

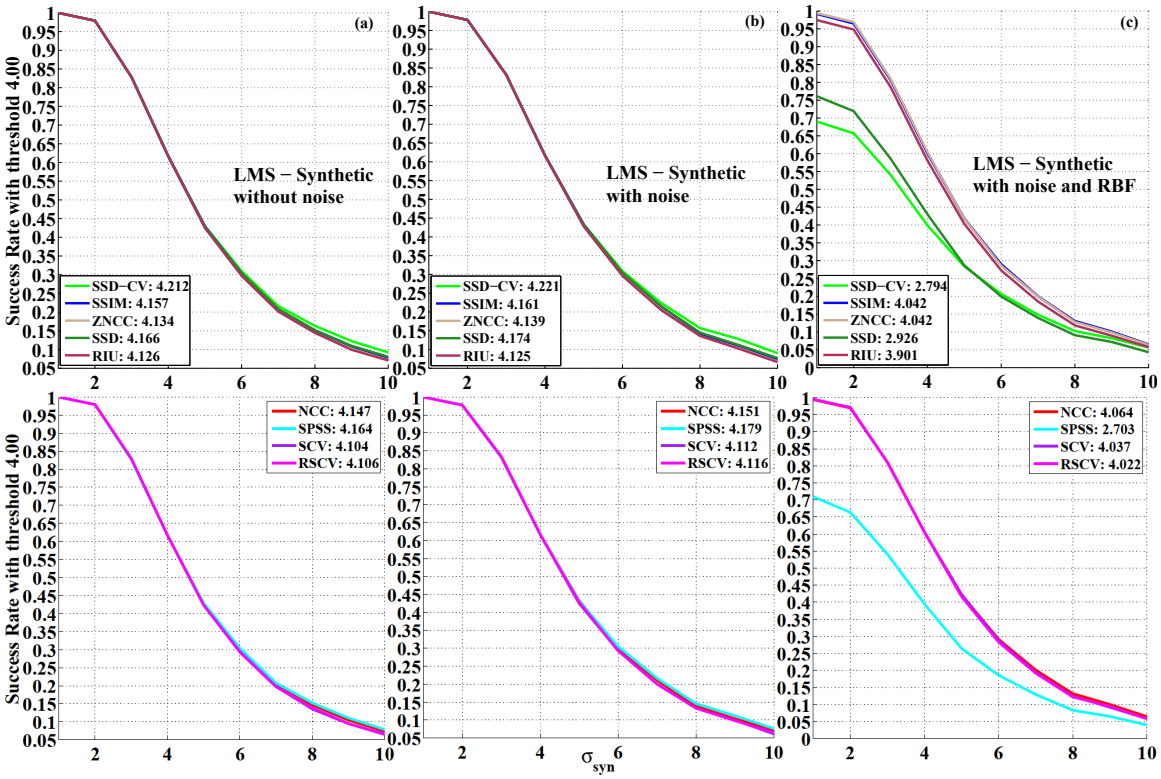


Figure 9.16: Performance LMS on synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

performance from RANSAC.

- RANSAC compares more favorably with LMS in terms of FR and is even slightly better with robust AMs as well as with SSD. In practice, this means that RANSAC gives somewhat less stable results than LMS - it moves around a bit about the object's actual location similar to the way direct sampling based stochastic SMs do - but does not actually lose track completely. LMS, while being more precise and stable as long as it tracks, is also slightly more prone to failure. The main reason for this is that RANSAC can work with any ratio of inliers while LMS only works correctly when at least half the points are inliers. Therefore, when the underlying patch trackers fail in challenging scenarios, RANSAC is slower to lose track and so can sometimes avoid complete failure if the cause of the challenge (occlusion, illumination change, etc.) is removed and the patch trackers recover before it has drifted off. LMS, on the other hand, loses track as soon as more than half the patch trackers have failed.

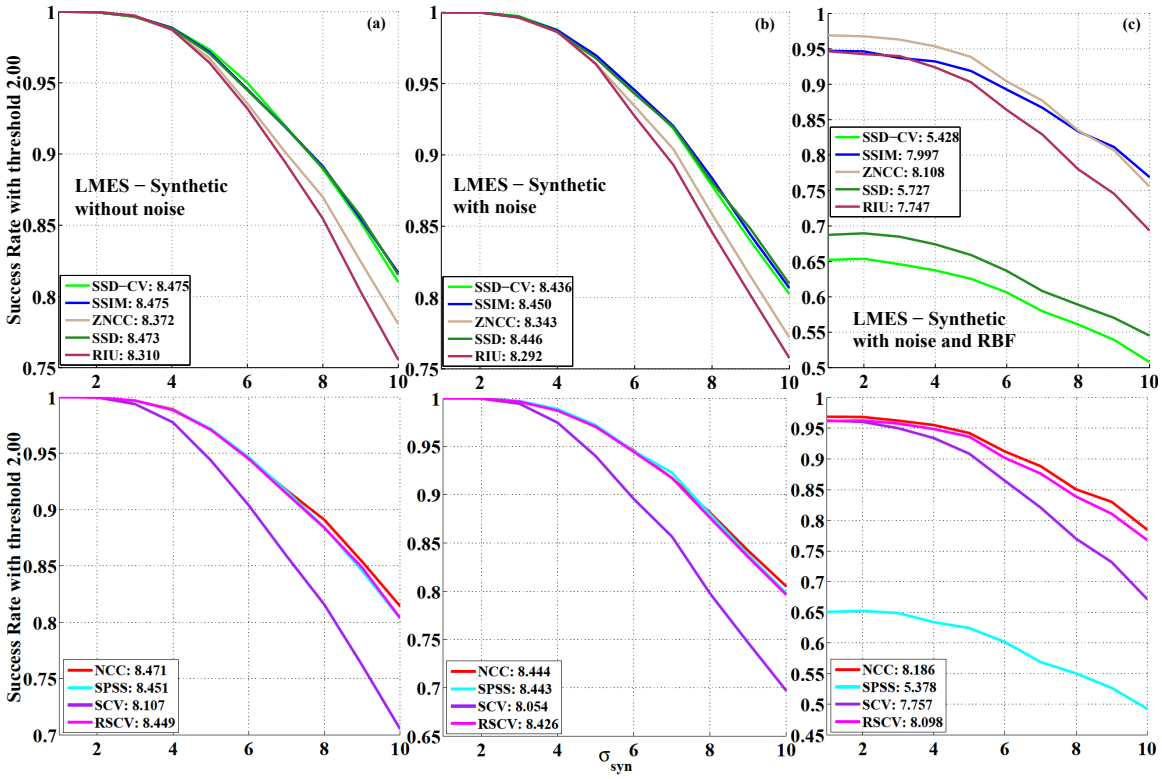


Figure 9.17: Performance LMES on synthetic datasets - (a) without noise (b) with noise and (c) with noise and illumination change

The case of SSD-CV is a bit harder to explain and has something to do with the reason for the superior performance of this AM that is not yet fully understood, as explained next.

- When used with LMS, SSD-CV is far better than other AMs especially in terms of robustness, having less than half the FR of the next best AM and significantly higher SR with larger t_p. As mentioned earlier, the reason for this difference between SSD and SSD-CV is not certain though results on individual datasets (Fig. B.9) seemed to indicate that the improvement is mainly confined to UCSB and LinTrack datasets where motion blur due to fast movement is the main cause of tracking failures. It seems likely therefore that the pyramidal implementation of SSD-CV is better than that in MTF since using the latter did not improve results significantly (Fig. A.14 (c)).

- LMS and RANSAC show comparatively little variation between the AMs and SSD is actually the best performer among the MTF AMs followed by SPSS.

This suggests that simpler AMs are more suitable for tracking the small sub patches which also makes more sense from the speed perspective as these are usually also the fastest. Their superior performance is probably because, when the tracked regions are so small and only need to be used to estimate 2 DOF motion between consecutive frames, simple pixel wise measures can make better use of whatever little texture information in available there and the illumination invariance of more complex AMs is not much help with such short term tracking.

- ZNCC, though also a pixel wise measure and one of the best performers with GD based SMs, turns out to be the worst one here after NGF. This reiterates that illumination invariance in the sub patch trackers does not benefit these SMs.

- LMES with SSD-CV in the first layer and ESM/NCC in the second one (termed SSD-CV/NCC in Fig. 9.14 (c)) is the best performing configuration of this SM. This is also the best performer among all tested trackers in this study in terms of SR though LMS with SSD-CV has slightly lower FR. It is also worth noting that replacing NCC with SSD in the second layer (SSD-CV/SSD in Fig. 9.14 (c)) leads to a significant drop in performance so that its SR becomes comparable to MTF AMs and actually worse than LMS with SSD-CV. There are several challenging frames with both fast motion and large illumination changes where LMS manages to find the correct location but ESM with SSD fails even when provided with this as the starting point. When used with NCC, however, ESM takes advantage of this AM's illumination invariance to converge correctly.

- LMS performs surprisingly poorly on synthetic datasets (Fig. 9.16). These tests were repeated several times but the poor performance remained consistent. Though the reason for this is not clear, it does point to the limitation of such tests as predictors of real world tracking performance.

- LMES performs much better than LMS on synthetic datasets (Fig. 9.17) but still not as good as PFFC. The relative performance of AMs is largely same as in ESM with the notable difference of SCV being significantly worse than RSCV - evidently LMES, unlike ESM, behaves more like a forward model than a mixed one.

9.1.3 Summary

Fig. 9.18 provides an overall summary of performance for all AMs with each SM so that the best AMs for each SM can be determined at a glance. As in the previous subsections, the overall performance of a tracker is measured by two numbers - the average SR for $1 \leq t_p \leq 20$, which is equivalent to the areas under the respective SR curves [234], and the number of failures or FR. These quantities are therefore on the x and y axes of the scatter plots respectively. Since low FR and high SR are desirable, the best trackers are near the bottom right corners of these plots while the worst ones are near the top left.

In practice, tracking accuracy and robustness alone are often not sufficient criteria for choosing the tracker for any given application. The tracker's speed is also important especially if it is to be a relatively small part of a larger system where its results will be used by higher level modules and its speed will thus be a bottleneck. This is true of many robotics applications like visual servoing and SLAM. As these are the principal application domains for RBT, Fig. 9.19 presents the speed results for all AMs and SMs as the final criterion for choosing the most appropriate tracker. The mean and standard deviations shown there were computed using the average speeds over all sequences and subsequences that the trackers were tested in. These speeds only reflect the time needed for the tracker to process each frame - the time spent in acquiring the frames and applying Gaussian filtering has been excluded.

It should be noted that the speed of any module is highly implementation dependent and many of the speeds shown there can be increased significantly by more optimized implementations. However, the relative speeds between different modules are still largely valid for evaluating their comparative suitability for any application as they have all been implemented with roughly the same degree of optimization. It should be kept in mind, however, that some modules like PF, MI, CCRE and NGF that are relatively slow here, are also more suitable for parallelization and their speeds can be increased manifold this way.

Following observations can be made from these results, some of them being reiterations from the previous subsections:

- NCC is the best performing AM with all GD based SMs except IALK, usually followed closely by SSIM and ZNCC. Considering that ZNCC is significantly faster than both NCC and SSIM, it might offer a better compromise between speed and accuracy than either.

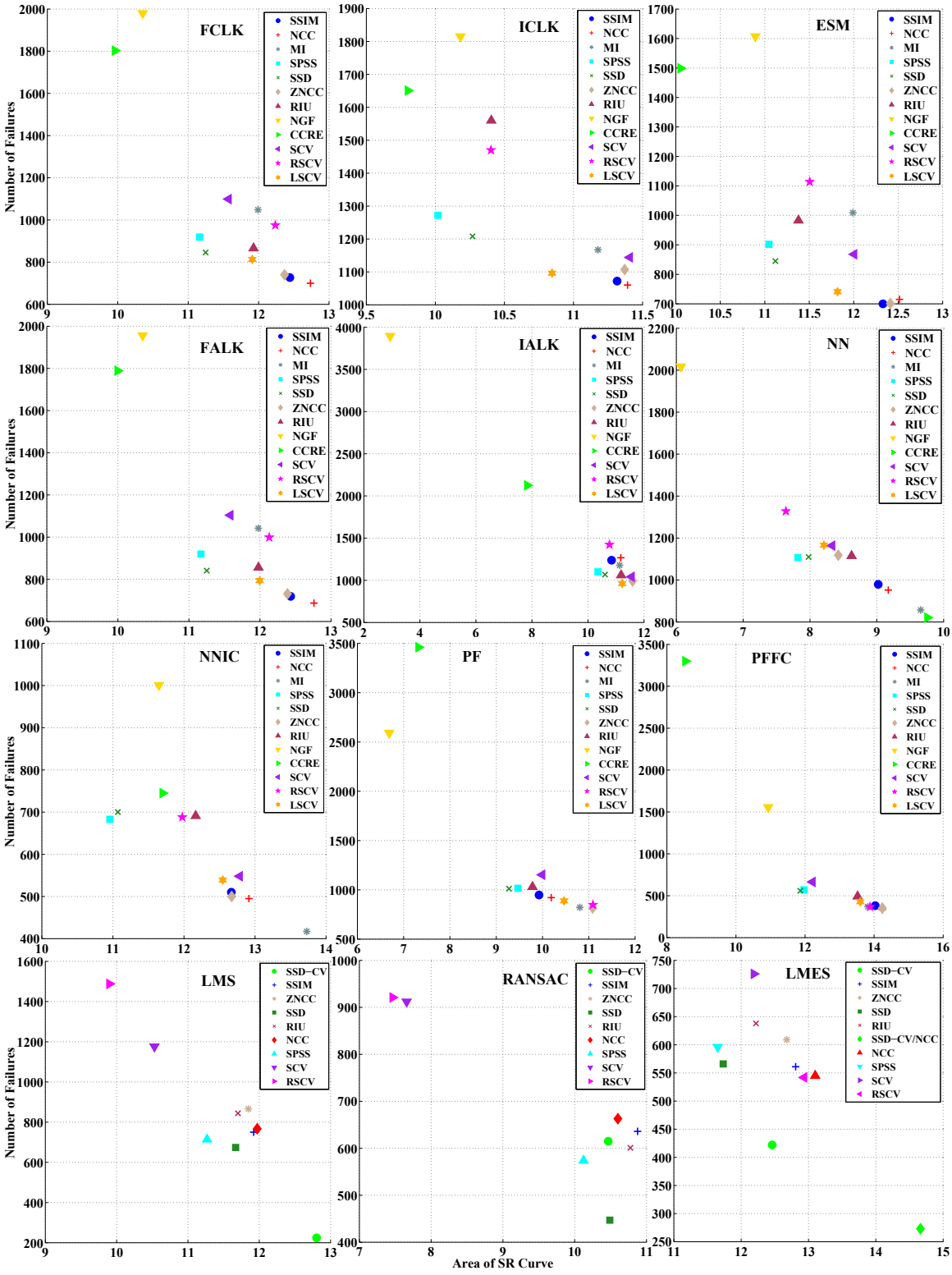


Figure 9.18: Performance summary for SMs. Best trackers are near the **bottom right** corner of each plot.

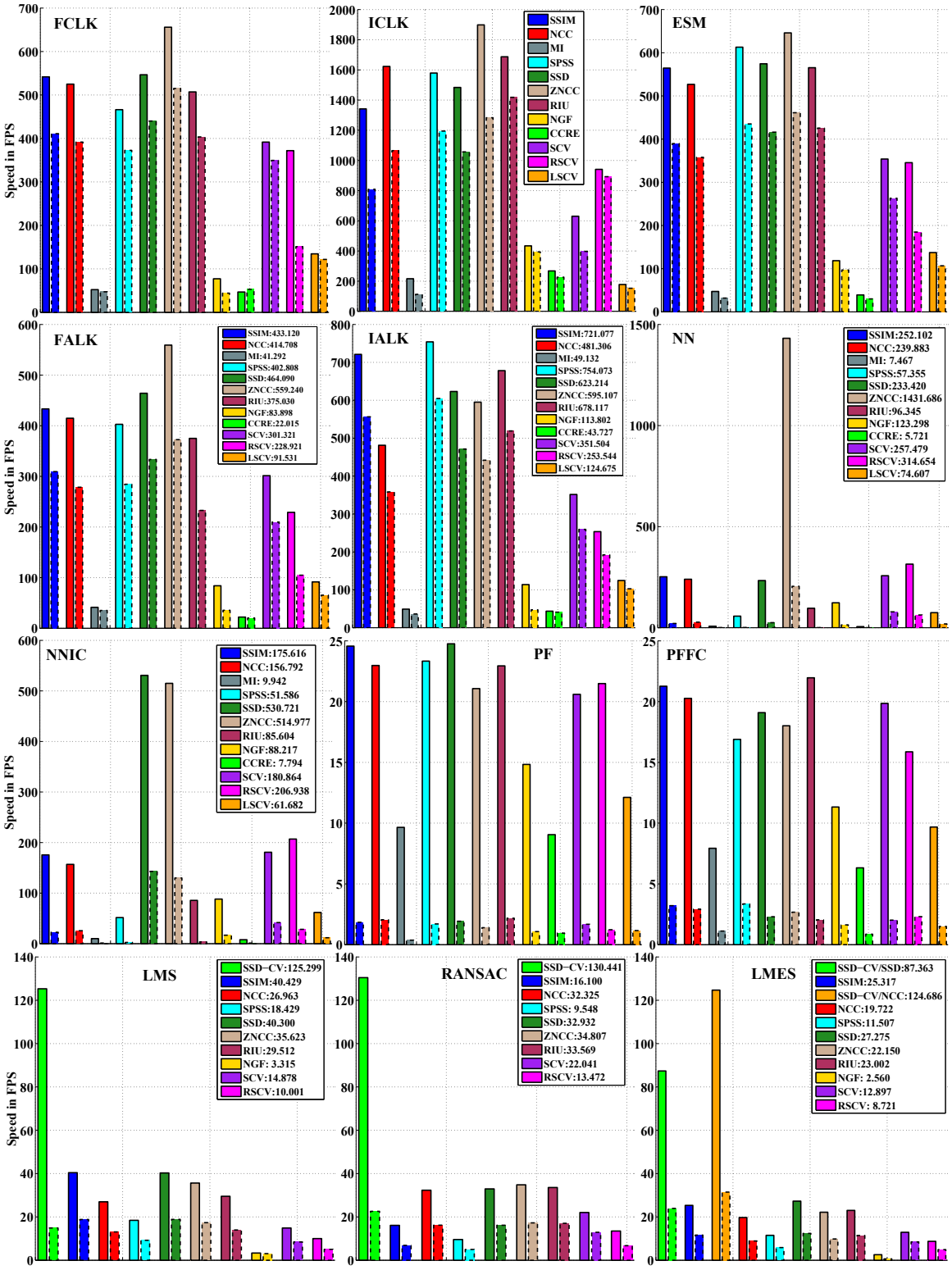


Figure 9.19: Speeds of AMs using different SMs with homography. Bars with solid and dotted outlines respectively show the means and standard deviations of speeds in frames per second (FPS) processed by the tracker. Legends show the means.

- It is more difficult to choose a clear winner with IALK as many of the AMs are clustered together near the bottom right - probably because IALK itself is the limiting factor here rather than the AMs. ZNCC is one of the best candidates here too, though, and, combined with the fact that it is also the fastest, this is perhaps the optimal choice with IALK too.

- CCRE and MI are the best AMs with NN while MI is the best one with NNIC. In terms of speed, though, ZNCC is by far the fastest, being over 5 times faster than the next one and almost 200 times over MI/CCRE. Along with SSD, it is also one of the fastest with NNIC. Since the number of samples has a significant impact on the performance of NN (Fig. A.6), it might perhaps be a better idea to use ZNCC with more samples rather than MI/CCRE with less, assuming that initialization time is not crucial to the application as this increases linearly with the number of samples.

- RSCV and ZNCC are the best AMs with PF while NCC and SSIM are so with PFFC while there is not much to choose between them in terms of speed.

- SSD-CV is the best AM, by a large margin, with LMS and LMES while SSD seems the best overall with RANSAC though NCC and SSIM do have higher SR. In terms of speed, SSD-CV is by far the fastest due to its highly optimized and parallelized implementation. It is worth noting that LMES with SSD-CV is actually faster with NCC in the second layer than SSD, probably due to the faster convergence of ESM with NCC.

- GD based SMs have much higher standard deviations than stochastic ones due to their iterative nature since the number of iterations actually run varies widely between frames.

- PF has the smallest standard deviations and also gives the truest indications of the respective computational complexities of the AMs with the constraint that Eigen optimizes certain matrix operations better than others.

- PF and PFFC are the slowest SMs but also highly parallelizable since the weight of each particle can be computed independently. Efficient parallelization of this SM is one of the extensions to MTF that are currently under development.

- SSIM is about as much faster than NCC on average as it is poorer in accuracy - the two aspects seem to compensate for each other well.

- SPSS is not significantly faster than SSIM with any of the SMs except perhaps ICLK even though it has lower computational complexity. For PF and GD based SMs, this is partly due to SSIM finding convergence in fewer iterations and partly due to the way Eigen optimizes matrix multiplications. Many of these are used for computing f_{ssim} and its derivatives while those of f_{spss} have to be computed pixel by pixel. The same holds for SSD too. For NN, SSIM turns out to be amenable to have more of its computations performed within the distance feature transform (Sec. 7.2.3) so that the functor f_D is significantly faster.

- IALK is not much faster than FALK with any of the AMs and is quite a bit slower than ICLK. This behavior was shown only by the LM formulation - the GN variant was significantly faster though still not as fast as ICLK. Also, SSIM is one of the fastest AMs here, ahead of simpler ones like SSD and ZNCC, probably because it needs fewer iterations to converge.

9.2 Appearance Models

This section only includes results for real world datasets to save space as the synthetic results did not offer any new insights not available in the previous section.

9.2.1 L2 Models

Fig. 9.20 and 9.21 present the results comparing all SMs with L2 models. The plots for SSD also include results for SSD-CV with LMS, RANSAC ad LMES. Following observations can be made from these results:

- The four variants of LK do not perform identically. Though FCLK and FALK are indeed evenly matched, both the inverse models are significantly worse than these and also different from each other - IALK is better with all AMs in terms of both FR and SR. It should be mentioned here that the situation is reversed when using the GN formulation (Sec. A.1.1) since IALK improves a lot more than ICLK on switching from GN to LM. It seems that the approximation of Eq. 4.9 in practice leads to poor step sizes for updating $\mathbf{p}$ and LM compensates

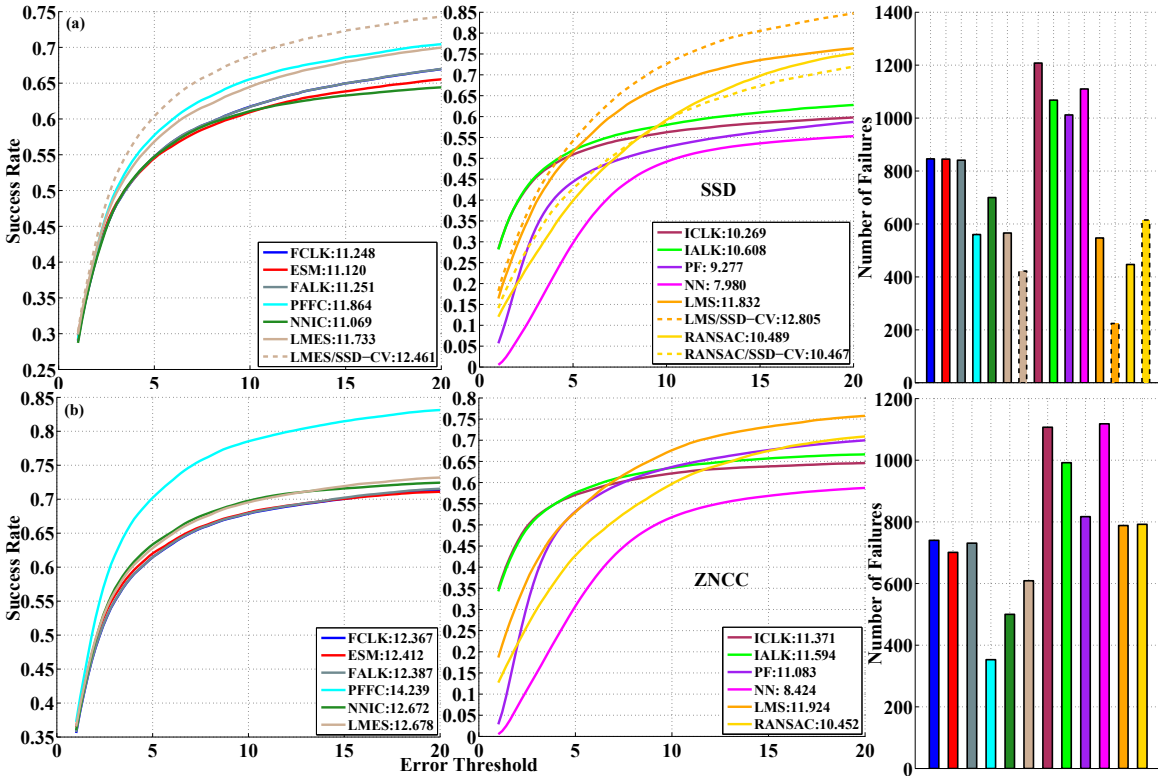


Figure 9.20: Performance of SMs with (a) SSD and (b) ZNCC

for this very well. This finding contradicts the equivalence between these variants that was reported in [25] and justified there using both theoretical analysis and experimental results. The latter, however, were only performed on synthetic images and even the former used several approximations. Therefore, it is perhaps not surprising that this supposed equivalence does not hold under real world conditions. A somewhat similar conclusion was also reached in [236] though in the context of image registration.

- ESM fails to outperform FCLK/FALK for any AM except SCV and even there it is probably the tendency of SCV to favor inverse models that is the main reason for ESM being better. This hypothesis is given more weight by the fact that ESM is just as much worse with RSCV as it is better with SCV. This fact too emerges in contradiction to the theoretical analysis in [76] where ESM was shown to have second order convergence and so should be better than first order methods like FCLK and FALK. It might be argued that the extended version of ESM [123, 3] used here might not possess the characteristics of the formulation

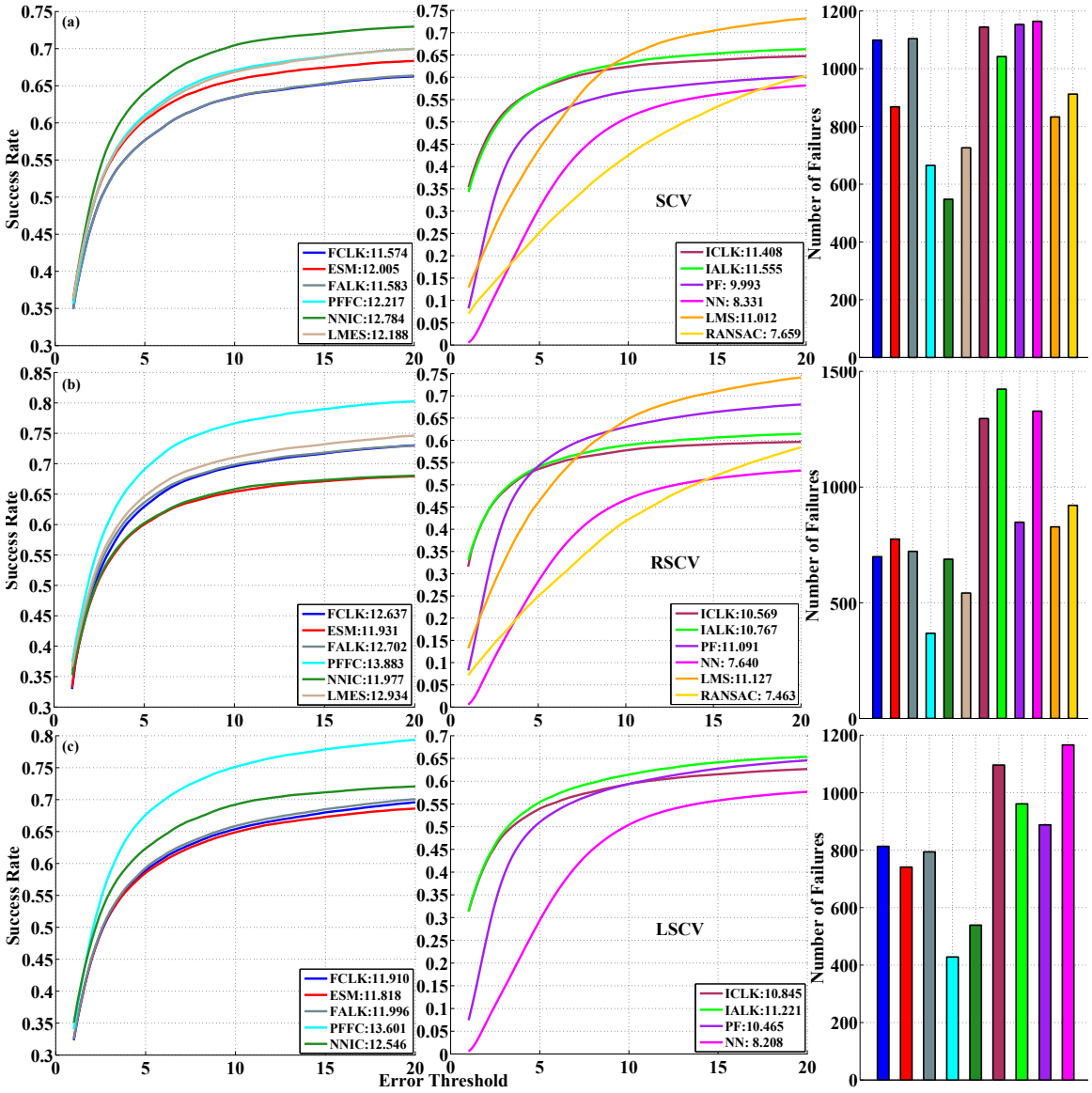


Figure 9.21: Performance of SMs with (a) SCV (b) RSCV and (c) LSCV

described in [76] but experiments were conducted with that exact formulation too and it was not found to perform significantly different from the one used here.

- The gain in performance with NNIC compared to ICLK is more marked with respect to FR than SR - the FR has been reduced by around half with all the AMs. The same holds true to a lesser extent with PFFC and LMES when compared respectively with FCLK and ESM. This is probably because the main

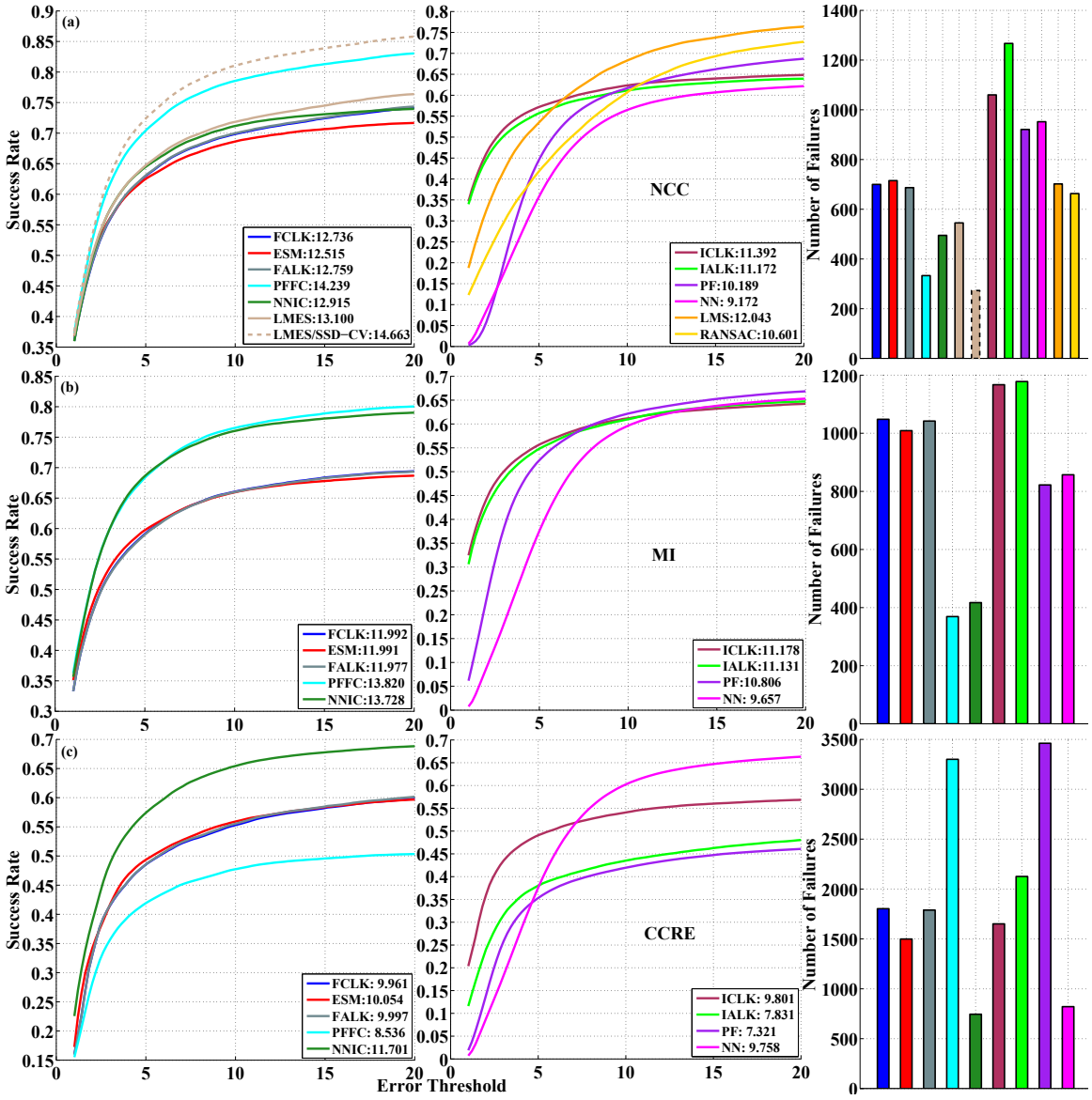


Figure 9.22: Performance of SMs with (a) NCC (b) MI and (c) CCRE

advantage of providing a better starting point for GD search is to make the tracker more robust by avoiding failures in scenarios like fast motion where the object's location from the last frame lies outside the region of convergence.

9.2.2 Robust Models

Fig. 9.22 and 9.23 present the results comparing all SMs with robust models. The plots for NCC also include results for LMES with SSD-CV in the first layer. Following

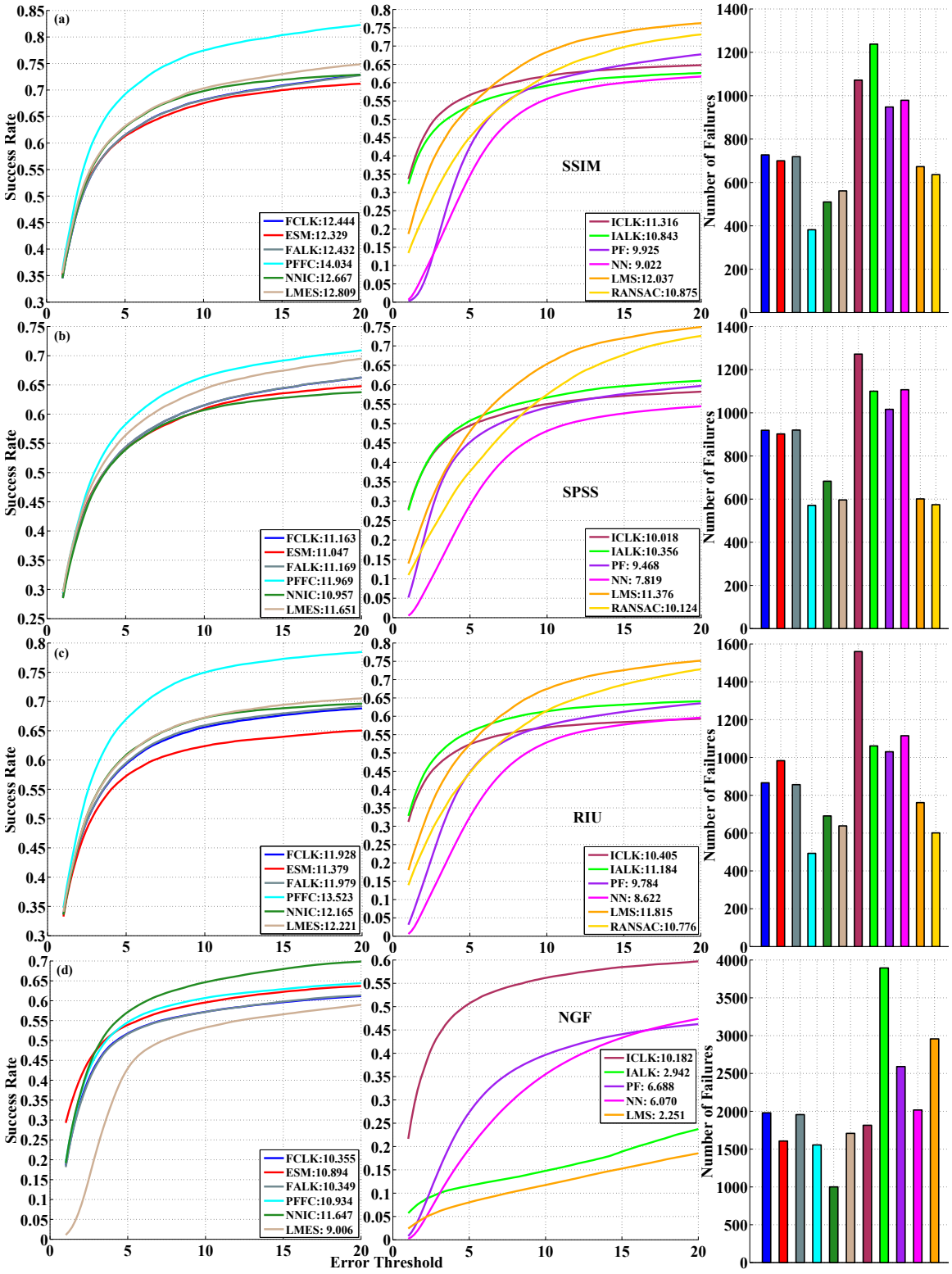


Figure 9.23: Performance of SMs with (a) SSIM (b) SPSS (c) RIU and (d) NGF

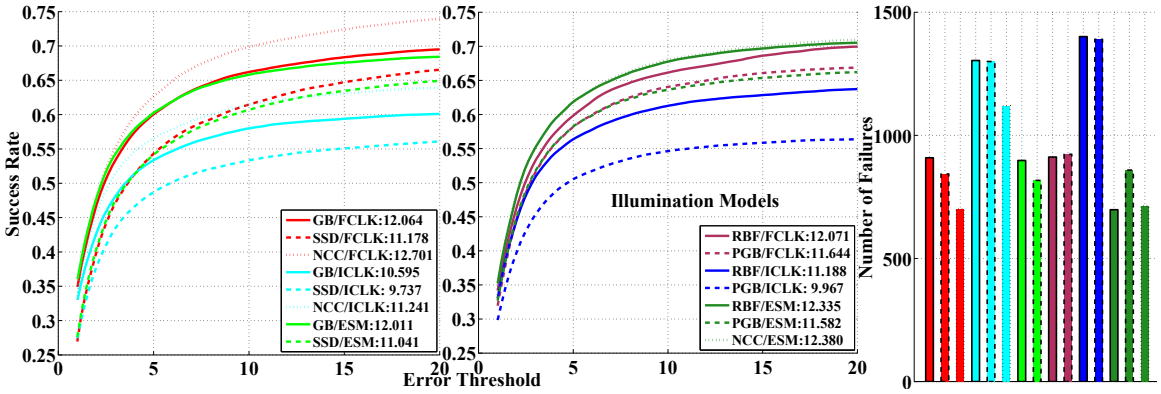


Figure 9.24: Performance of ILMs using FCLK, ICLK and ESM

are some observations from these results:

- The first two observations in the previous section - non equivalence of LK variants and ESM not outperforming FCLK/FALK - are true here too.
- ICLK fares somewhat better against IALK with robust AMs that with L2 ones - it is much better with CCRE and NGF and slightly better with MI, NCC and SSIM. RIU is the only AM where it is notably worse for some reason. This is consistent with the fact that the extension of GN method with $\hat{\mathbf{H}}_{self}$ does not make as much sense for additive SMs as compositional ones [93].
- The performance improvement provided by PFFC and NNIC over FCLK and ICLK respectively is more strongly marked here than with L2 models especially in terms of SR. In fact, PFFC with NCC nearly matches the new state of the art provided by LMES with SSD-CV and proves to be the second best tracker tested in this study.
- CCRE and NGF seem to favor inverse models as both ICLK and NN are overall better than FCLK and PF respectively - a trend that becomes even more noticeable when comparing NNIC with PFFC. This is somewhat surprising since, being the worst performing AMs and so presumably harder to optimize, these would be expected to work better with the more sophisticated forward models.

9.2.3 Illumination Models

Fig. 9.24 presents the results comparing all 3 ILMs with SSD using FCLK, ICLK and ESM. NCC is also shown for comparison as it too provides illumination invariance

like the ILMs. SSD is the only AM in MTF that currently supports ILMs so testing is limited to this AM. As the simplest AM, SSD is also likely to be more sensitive the addition of ILMs and so will demonstrate their impact better. A 3×3 grid of control points was used for RBF and a 3×3 grid of sub patches for PGB. Experiments conducted with resolutions ranging from 2×2 to 6×6 indicated that this works best though the impact of resolution is not significant. Following points can be observed:

- GB does improve performance over SSD with all three SMs but fails to match NCC with any of them.
- RBF improves further over GB, at least in terms of SR, and does manage to perform as well as NCC with both ICLK and ESM. In terms of FR, however, it outperforms GB only with ESM and has higher FR than both GB and SSD with the other two SMs. This is probably because the 10 extra parameters that need to be estimated make the search more likely to get stuck in local maxima, thus causing the tracker to fail.
- The significantly lower FR of RBF with ESM finally lends some credence to the supposedly superior convergence properties of this SM. In fact, ESM in general seems to handle higher DOF ILMs better than FCLK which in turn improves over ICLK. This seems to indicate that the advantage of using a more sophisticated SM becomes more prominent as the dimensionality of the search space increases, thus rendering the search more challenging.
- PGB performs worse than GB in terms of both FR and SR with all of the SMs. Its poor performance is most notable with ICLK, where it is even worse than SSD. It seems that the discontinuity between subregions that is inherent to this ILM does not represent realistic lighting changes well and the disadvantageous effect of the extra parameters dominates.

9.2.4 Summary

Fig. 9.25 provides an overall summary of performance for all SMs with each AM. Speed results have been omitted here to save space. Following are some points to be noted:

- PFFC is the best performing SM with most AMs especially if SSD-CV is excluded from consideration. There are three exceptions to this - SCV, CCRE and

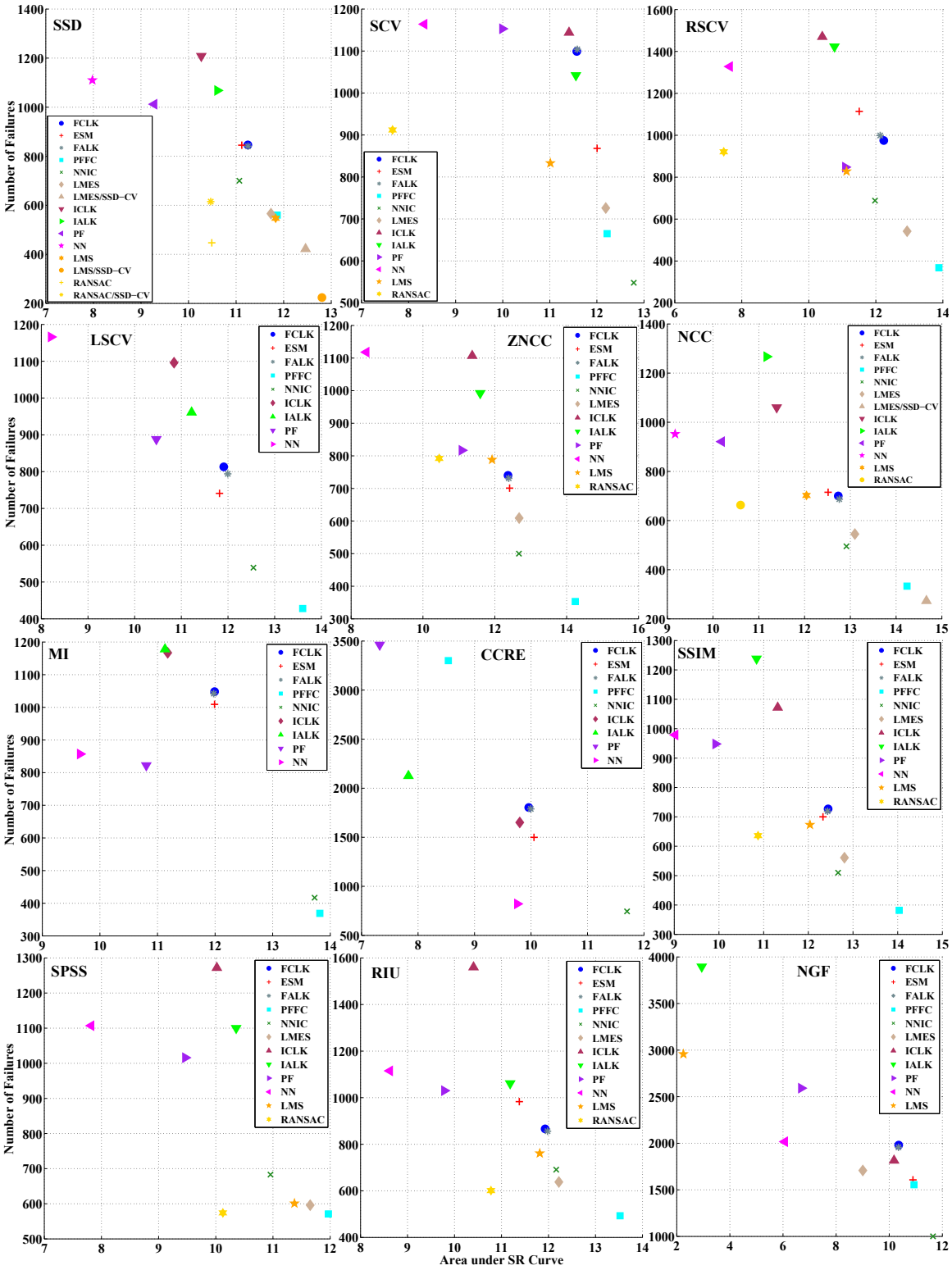


Figure 9.25: Performance summary for SMs. Best trackers are near the **bottom right** corner of each plot.

NGF - and they all perform best with NNIC. This is probably not unexpected as all of these have been observed to favor inverse SMs in general.

- If SSD-CV is included, LMS and LMES are the best performers with SSD and NCC respectively.
- ICLK, IALK and NN are among the worst performing SMs with most AMs. The only notable exception is CCRE where PF and PFFC are the worst. This is probably due to poor likelihood values rather than any flaw in these SMs though.
- The fact that almost all the best performing SMs are composite ones proves the efficacy of this approach and suggests that future research efforts be directed towards improving these. There does not appear to be much scope for improving individual SMs as evidenced, for instance, by the failure of ESM, widely considered the state of the art in GD based tracking, to outperform FALK that was introduced 23 years before it.
- Conversely, the fact that all the worst performing SMs are inverse models, and thus the fastest ones, confirms that speed comes at the cost of accuracy and proofs to the contrary [67] do not hold in practice.

9.3 State Space Models

The results presented here follow a slightly different format from the last two sections due to the difference in the motivations for using a low DOF SSM:

- Estimating lower DOF motion reduces the dimensionality of the search space of $\mathbf{p}$. This decreases the probability of the search process getting stuck in a local optimum and thus makes the tracker more robust.
- Lower DOF SSMs tend to be faster with GD based SMs since their Jacobians are less expensive to compute.

In order to examine the first point more closely, synthetic datasets were generated with 2, 3, 4 and 6 DOF warping and containing both noise and RBF illumination changes to make them more challenging. These were used for testing all the SSMs and the corresponding results are given in Fig. 9.26. Results are only given for NCC

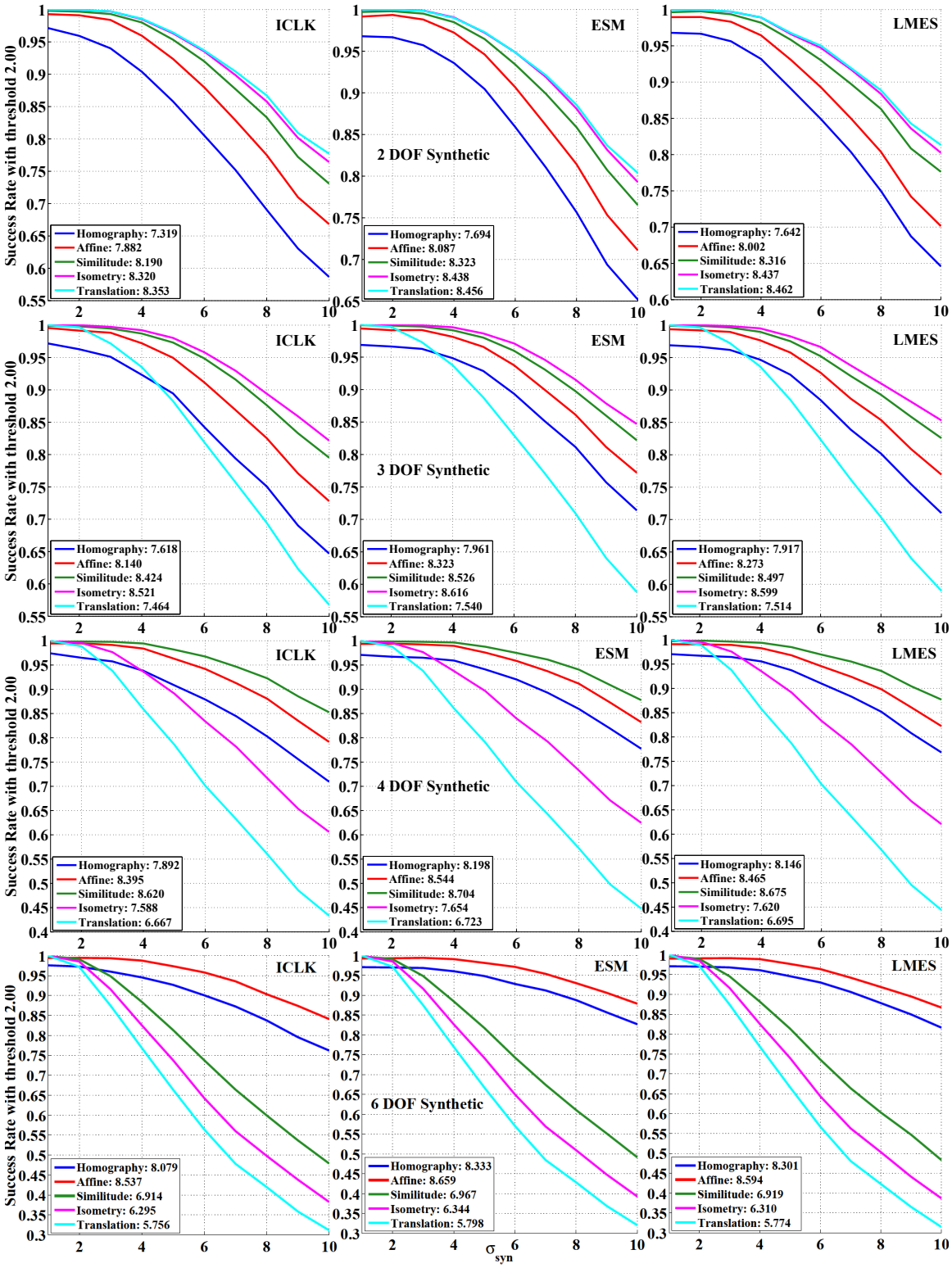


Figure 9.26: Performance of different SSMs with FCLK and NCC on synthetic datasets generated using 2, 3, 4 and 6 DOF warping and containing both noise and illumination change.

with three SMs - ESM, ICLK and LMES - though several other combinations were also tested and all of them provided similar results. Following points can be noted:

- As expected, lower DOF SSMs do indeed perform better on all of these datasets as long as the warp being estimated does not exceed their capabilities. The best performing SSM on each dataset is thus the one that corresponds to the DOF of the warping used for generating that dataset and performance decreases monotonically as DOF of the SSM increases.

- Once the DOF of the dataset exceeds that of the SSM, however, there is a sharp decrease in its performance, as would also be expected, and the performance trend also reverses - now it is the higher DOF SSMs that perform better.

- Increasing the sophistication of the SM involved has practically no effect on the results. This indicates that improved convergence properties or having a better starting point do not help a tracker with higher DOF SSM work as well as one with a lower DOF SSM in finding a warp corresponding more closely with the latter.

- The performance of higher DOF warps actually improves as the DOF of the synthetic dataset increases. Affine and homography, for instance, work best with 6 DOF datasets followed by 4, 3 and 2 DOF ones. This is harder to explain since any low DOF warp is a subset of all higher DOF warps. In other words, a 2, 3 or 4 DOF warp is also a valid 6 DOF warp so that there seems no obvious reason why a 6 DOF SSM should work better with the latter. It might, however, indicate that, for any tracking scenario, using the SSM with the least DOF that can accurately estimate the warping involved in that scenario, is to be recommended.

The SSMs were also tested on the real datasets to assess the practical significance of the synthetic results. There was an important difference in the evaluation methodology compared to the last two sections in that lower DOF ground truths were used for measuring the tracking error in order to make the evaluations fair. These were generated for each SSM by using least squares optimization to find the warping parameters that, when applied to the initial object location, produce a warped location whose alignment error E_{AL} with respect to the full 8 DOF ground truth is as small as it is possible to achieve given the constraints of that SSM. In most cases,

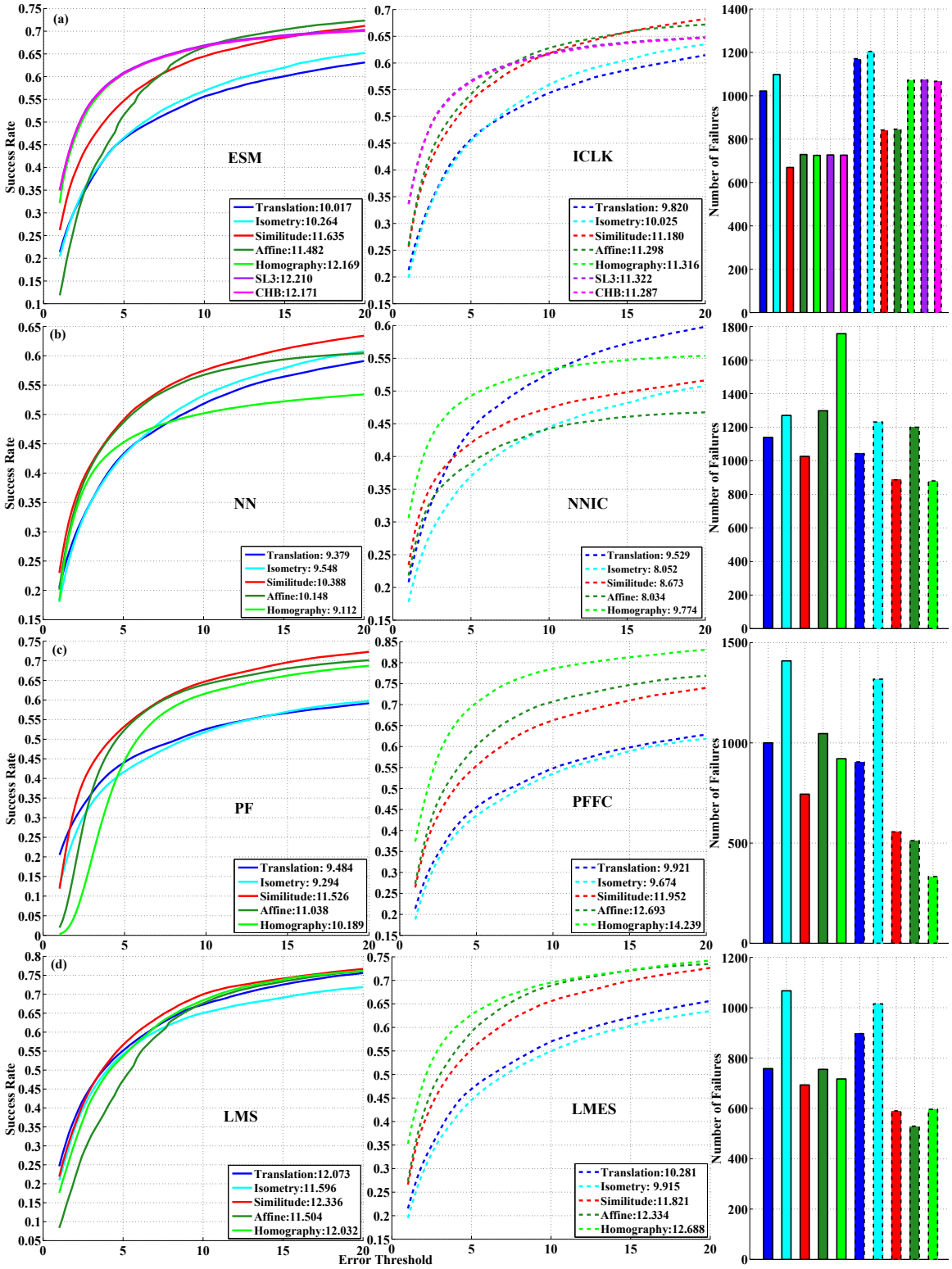


Figure 9.27: Performance of SSMs using SSIM with different SMs. Results were generated using optimized low DOF ground truth for each SSM.

the ground truth locations thus generated represent the best possible performance that can theoretically be achieved by any tracker that uses that SSM. In some rare cases, however, the resulting corners were quite unexpected so they were also visually inspected and any that appeared unreasonable were manually corrected. Fig. 6.1 shows some examples of low DOF ground truths.

Fig. 9.27 presents the results of these tests for SSIM with 8 SMs. ESM and ICLK results also include the two alternate parameterizations of homography (Sec. 6.6, 6.7). It should be noted that the results for NN with homography are not the same as those in the previous sections because a different sampler with mixture distribution has been used here (Sec. 8.3). Since the performance of NN depends largely on the quality of samples, the same sampling technique has to be used for all SSMs to ensure fairness while comparing them. The homography sampler used by NN in the previous sections (Sec. 6.5.1) cannot be used with lower DOF SSMs due to their inability to accurately transform the normalized unit square into the object bounding box in image coordinates. As a result, the sampling technique employed for generating the synthetic datasets has been used instead, though now in a mixture distribution form.

Following are some interesting observations from these results:

- All three parameterizations of homography have practically identical performance with both ESM and ICLK. This indicates that the theoretical justification given in [76] for using ESM with SL3 has little practical significance.

- Unlike the synthetic tests, lower DOF SSMs do not perform better with any GD or composite SM. On the contrary, all of these SMs, except perhaps NNIC, generally show improvement in performance on increasing the DOF, at least in terms of SR. The only partial exceptions with GD are affine and similitude which do outperform homography for higher values of t_p.

- Homography is by far the best with NNIC too but the remaining SSMs do not exhibit any well defined pattern.

- The increased robustness of low DOF SSMs with GD based SMs is at least partially apparent in the fact that their curves approach those of homography as t_p increases. Thus, though they may not be as precise as homography, they do tend to be more resistant to complete drift. In fact, a general trend apparent in the SR plots of high DOF SSMs, not only in Fig. 9.27 but also others seen earlier, is that their SR does not continue to increase over the entire range of t_p

but flattens out after a certain point (often for $t_p < 10$). Lower DOF SSMs, on the other hand, tend to have more of an upward slope throughout. This results from the fact that high DOF trackers follow the object very precisely as long as they are working but once they fail, they do so quite abruptly rather that drifting off gradually.

- Stochastic SMs do seem to be more favorable to lower DOF SSMs, particularly similitude and affine. Both of these outperform homography and the former is slightly the better one so that the expected behavior of performance improvement with decrease in DOF does hold for these 3 SSMs. Translation and isometry, however, are notably worse than all of these SSMs.

- A clear disparity exists between the behaviors of stochastic SMs, especially PF and LMS, and the corresponding composite ones, where the latter seem to favor higher DOF SSMs. It seems that having a better starting point helps the GD layer more with higher DOF SSMs. In other words, with the starting search point sufficiently close to the optimum, the advantage of smaller search space that lower DOF SSMs offer seems to lose its significance.

- Similtiude has the lowest FR as well as the highest SR for larger t_p with all SMs except PFFC and LMES, which, as observed in the last point, seem to favor higher DOF models. Similitude seems to have just enough DOFs for most practical scenarios, as exemplified in Fig. 6.1 for instance, while still being sufficiently constrained to provide significantly more robust performance than affine and homography.

- LMS shows the least variation between SSMs in terms of both SR and FR. Isometry is the only SSM that is noticeably worse, especially in FR, possibly because the DLT algorithm is not exactly valid for it due to its non linear parameterization. The approximate approach of estimating similitude and then dropping the scaling factor has to be used to obtain isometry warp using this method [15].

- The overall conclusion seems to be that constraining the DOF only helps to improve performance as long as the resulting SSM is flexible enough to provide a reasonably close approximation of the true warp so that the corresponding patch resembles the actual object well enough for a clear optimum to exist at

this approximate location too. If the SSM is too constrained, its patch will either include too much of the background or too little of the object patch whenever the object undergoes complex motion. This in turn makes the optimum less marked and so more difficult to find, especially for GD based SMs that depend on the existence of a strong slope near the optimum to converge correctly. As mentioned before, similitude exemplifies this conclusion well and seems like the recommended SSM to use unless the object is expected to undergo unusually complex motions like large out of plain rotations.

To examine the second reason for using low DOF SSMs, speed comparisons between them were also made and the results are shown in Fig. 9.27. Following points can be noted:

- Significant differences in speed exist only for GD based SMs, especially ESM and FCLK where the Jacobian has to be recomputed in each frame.
- Isometry has little advantage over similitude and is even slightly slower with ESM. This is because of its non linear parametrization that needs relatively costly sine and cosine computations to be performed to obtain both $\mathbf{w_{iso}}$ and its derivatives.
- CBH is predictably slower than standard homography since its gradient has to be computed using the slow numerical technique (Sec. 6.7). Somewhat surprisingly, however, SL3 is actually slightly faster even though it has to compute the matrix exponential to update $\mathbf{p_s}$ in each iteration. The reason for this is not clear though it might be that SL3 occasionally finds convergence in fewer iterations.
- Translation is the only SSM that seems notably faster with NN and PF. This is probably because the DLT algorithm does not need to be run by the sampler to convert point perturbations into samples of $\mathbf{p_s}$.
- LMS provides almost identical speeds with all SSMs indicating that the robust estimation step is not the dominant factor in determining the speed of this SM.

Before this section is concluded, it is noted that limiting the DOF makes RBTs directly comparable to OLTs as these too work in low DOF search spaces. As a result, 2 DOF RBTs were also compared with nine state of the art OLTs described

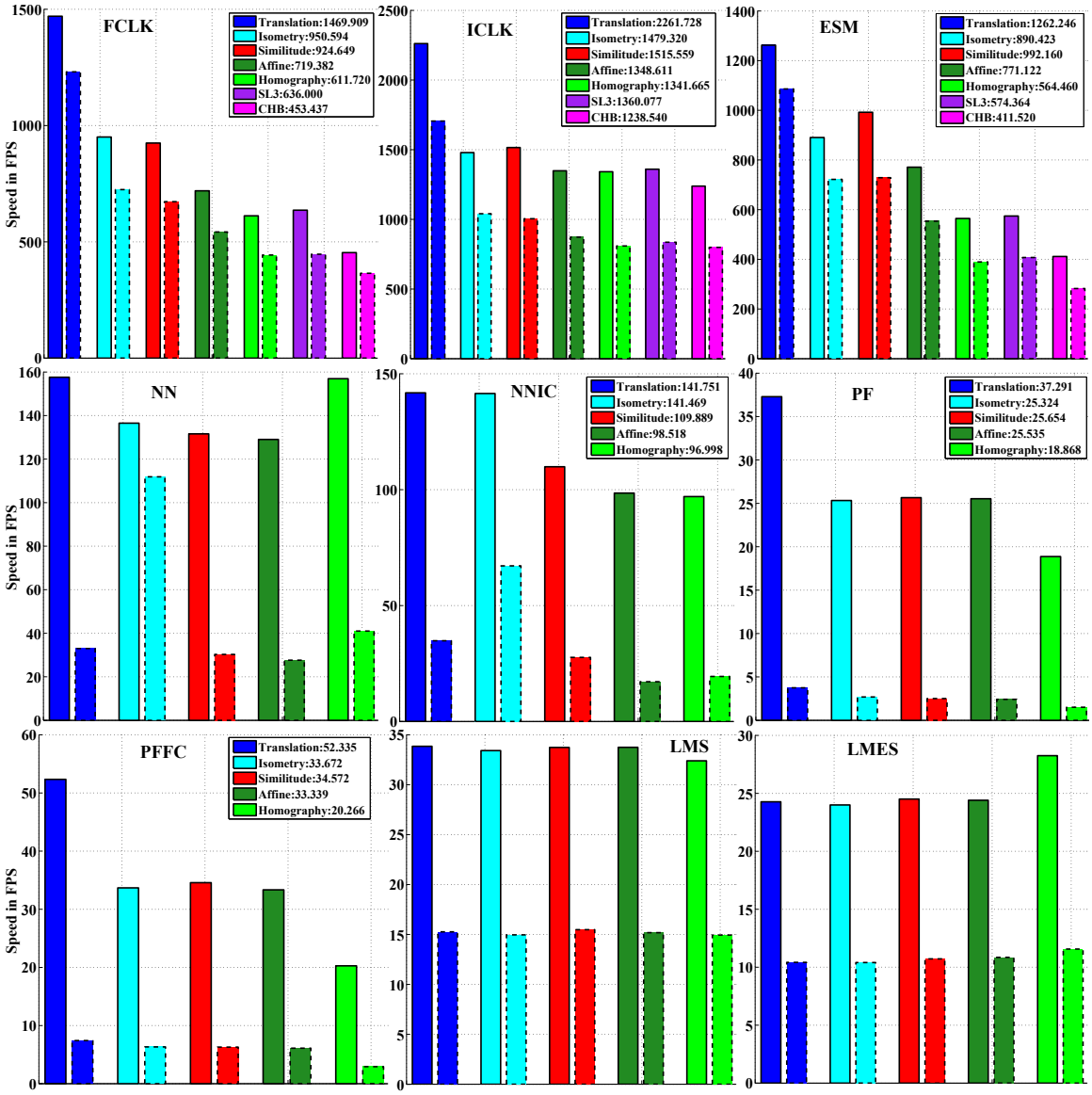


Figure 9.28: Speeds of SSMs with SSIM and different SMs.

in Sec. 2.1. The original C++ implementations of these trackers have been used after fully integrating them into MTF. This not only makes it easy to reproduce the results presented here and but also makes it reasonable to compare the speeds of these trackers with RBTs since slower speed is one of the main reasons why OLTs are often not used in robotics applications.

In addition to the four datasets used so far, the OLTs were also tested on the VOT 2016 dataset [13] along with the state of the art in 2 DOF RBT as represented by LMS and LMES. Since the ground truth annotations in this dataset are not very

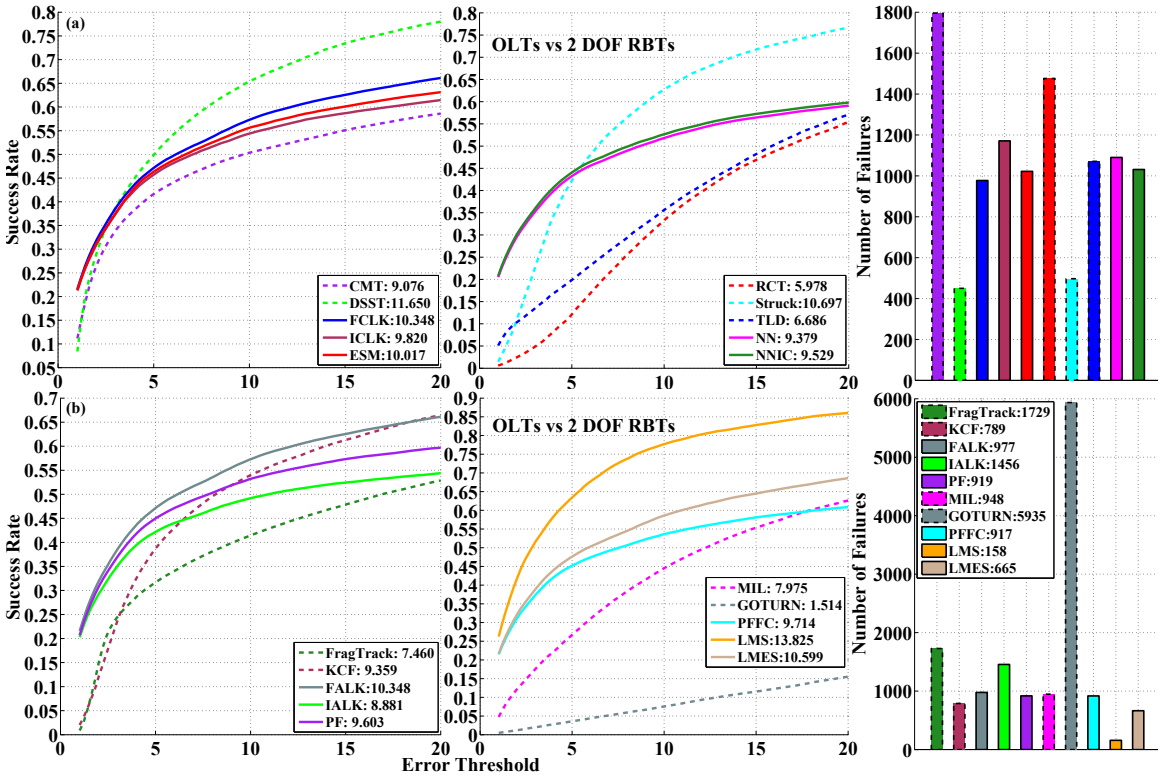


Figure 9.29: Comparing OLTs with SMs using translation SSM. SSIM was used as the AM for all SMs except LMS where SSD-CV was used to demonstrate the state of the art in 2DOF RBT. OLTs are shown with dotted lines and RBTs with solid ones. All results were generated using 2 DOF ground truth.

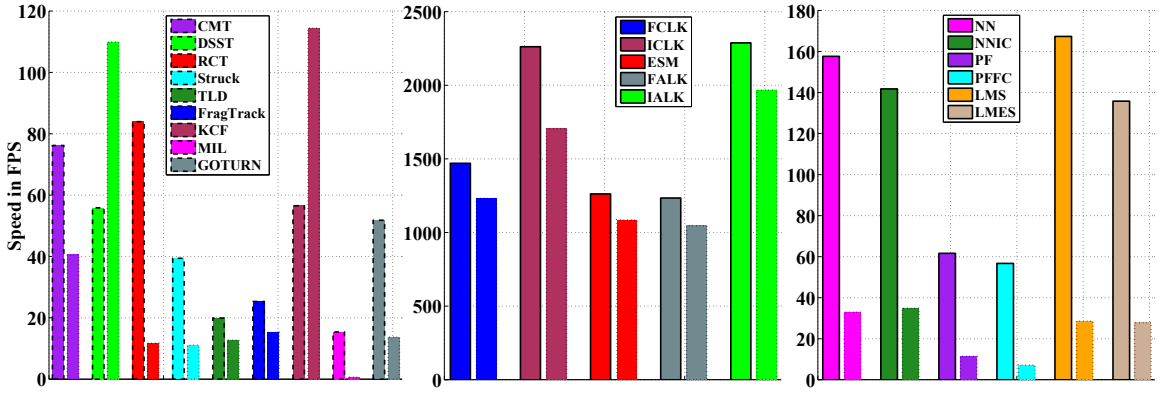


Figure 9.30: Speeds OLTs and SMs with SSIM and translation. Solid/dashed and dotted lines respectively show the means and standard deviations of speeds in frames per second (FPS) processed by the trackers.

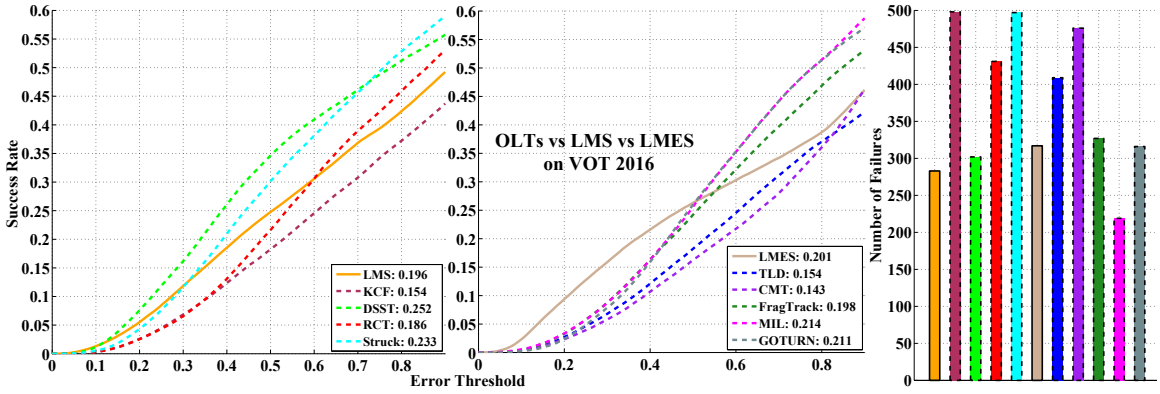


Figure 9.31: Comparing OLTs with state of the art RBTs on VOT 2016 dataset using Jaccard error

precise - indeed in most sequences it is not even possible to precisely represent the object pose with a bounding box - the Jaccard error has been used instead of E_{AL} for measuring the tracking accuracy. This is equivalent to the inverse of the Jaccard index or region overlap [237] and defined as:

$$E_{JAC} = 1 - \frac{|\mathbf{r}_{(track)} \cap \mathbf{r}_{(gt)}|}{|\mathbf{r}_{(track)} \cup \mathbf{r}_{(gt)}|} \tag{9.1}$$

where $\mathbf{r}_{(track)}$ and $\mathbf{r}_{(gt)}$ are the image regions representing the tracker output and the ground truth respectively. Since E_{JAC} varies only from 0 to 1, a reinitialization threshold of 0.9 was used for the FR plots.

Fig. 9.29 shows the performance of all SMs with translation compared against the OLTs while Fig. 9.30 shows the respective speeds. Fig. 9.31 shows the results over VOT 2016 dataset. Following are some interesting observations from these results:

- As expected, all the OLTs have low SR for smaller t_p since they are less precise in general [8]. What is more interesting, however, is that none of these trackers, with the exception of Struck and DSST, managed to outperform even the simplest of RBTs like ICLK and NN. Even DSST and Struck fall well behind the state of the art in 2 DOF RBT as represented by LMS with SSD-CV. These results clearly demonstrate the unsuitability of OLTs for the kind of tasks present in these datasets which represent the scenarios involved in robotics applications like visual servoing fairly well.

- The speed comparisons in Fig. 9.30 show another reason why OLTs are not suitable for tracking scenarios where speed is crucial - they are 10 to 20 times

slower than RBTs except PF/PFFC for which an efficient implementation is not yet available. It may be noted that DSST and KCF have much higher standard deviations than other trackers since their speeds depend strongly on the size of the initial bounding box and so varied widely between sequences. However, the mean figures do provide a good idea of the general performance that can be expected from these trackers. It is not surprising that tracking based SLAM systems like SVO [238] use RBTs as they may need to track hundreds to thousands of patches per frame - a feat that is clearly impossible with all but the fastest of RBTs.

- The superiority of DSST and Struck over other OLTs is consistent with results published elsewhere [8].

- GOTURN is by far the worst performer though considered to be the state of the art in OLT [53]. This points to a fundamental difference in the types of challenges involved in these two domains of tracking and shows why designing trackers to perform well in one domain usually ends up compromising their performance in the other.

- FALK and FCLK give identical performance which is to be expected as the two formulations are identical for translation.

- LMS is far better than LMES since, as seen while analyzing Fig. 9.27, the GD layer is likely to fail with lower DOF SSMs like translation even when the first layer finds the optimal location if the corresponding patch does not represent the object's appearance well.

- NNIC and PFFC perform no better than NN and PF respectively. This lack of difference between stochastic and composite SMs can be attributed to the former not finding very good locations to begin with.

- LMS and LMES perform surprisingly well on VOT 2016 (Fig. 9.31) - they are easily comparable to most of the OLTs and are in fact outperformed only by DSST and Struck for $t_p < 0.5$ which includes most cases where the tracking result is actually close enough to the object to be useful for many applications. In terms of FR too, LMS is outperformed only by MIL.

- In contrast to Fig. 9.29, GOTURN is one of the better trackers here and is mostly at par with MIL. The latter turns out to be the best tracker in terms of FR though DSST and Struck are better with SR.
- FragTrack is one of the better performing trackers and easily outperforms several of the newer and far more sophisticated trackers like KCF, CMT and TLD even though it does not even employ any online learning.

9.4 Summary

This chapter presented results comparing different methods for implementing each module in the proposed decomposition with several methods for the other two modules. Detailed analysis of these results yielded several new insights about these methods along with discovering a new state of the art in this domain of visual tracking.

Chapter 10

Conclusions and Future Work

10.1 Conclusions

This thesis presented a novel method for decomposing registration based trackers into three sub modules - SM, AM and SSM - respectively comprising the optimization method, the similarity metric and the warping function. It also adapted three image similarity measures - SSIM, RIU and NGF - as new AMs for high DOF tracking. A simpler and faster variant of SSIM called SPSS was also introduced. These four AMs, along with eight existing ones - SSD, ZNCC, SCV, RSCV, LSCV, NCC, MI, CCRE - were tested comprehensively with twelve different SMs.

The SMs were divided into three categories - gradient based SMs including FCLK, ICLK, ESM, FALK and IALK, stochastic SMs including NN, PF, LMS and RANSAC and composite SMs created by running a stochastic and a gradient based SM in cascade. Composite SMs were shown to combine the advantages of both of their components to create a tracker that is more robust than the gradient based component while also being more precise than the stochastic one. Three composite SMs were considered - NNIC, PFFC and LMES created respectively by combining NN with ICLK, PF wiih FCLK and LMS with ESM. The last two were introduced here for the first time and shown to provide a new state of the art in high DOF tracking.

These AMs and SMs were combined with seven different SSMs - translation, isometry, similitude, affine, homography, SL3 and CHB - though detailed testing was mostly done using homography since high precision tracking is the main focus of this work. Nevertheless, the SSMs were also compared with each other as well as with state of the art online learning and detection based trackers to better understand the impact of this sub module on tracking robustness and accuracy.

Trackers were tested using both synthetic sequences and four publicly available real world datasets called TMT, UCSB, LinTrack and PAMI. These datasets provided over 100000 frames to ensure statistical validity of the results. This was further aided by tracking each sequence from multiple starting points to fully test against all challenges it offers, thus effectively increasing the frames to almost 600000. Detailed analysis of the results was performed to gain several novel and interesting insights about the strengths and weaknesses of these methods. This included many observation about existing methods that were missing from their original papers.

Following are some of the more significant findings:

- LMES with SSD-CV in the first layer and NCC in the second is the best performing tracker overall and establishes a new state of the art in high DOF tracking.
- PFFC is the best SM with most AMs implemented within MTF and is only slightly worse than the above tracker.
- NNIC is the best SM with three of the AMs - SCV, CCRE and NGF - that consistently favored inverse SMs over forward ones.
- The fact that all three of the above are composite SMs indicates that this is a promising direction for future research in this domain.
- The four variants of LK are not equivalent in practice and ESM is not better than FCLK This shows that the theoretical proofs in [25] and [73] respectively have little practical significance.
- NCC is the best AM with most SMs though SSIM is usually quite similar and also much better than SPSS which is only at par with SSD.
- RIU performs fairly well and is comparable to SCV/RSCV on average. It is particularly well suited to scenarios containing both illumination changes and fast motion.
- NGF is one of the worst performing AMs and does not stand up to the theoretical advantages it supposedly offers over MI [111].
- CCRE and MI perform much better with stochastic SMs than GD ones. This is most marked with NN and NNIC where these are respectively the best AMs by large margins.

- ILMs do improve over SSD but none are significantly better than NCC.
- Low DOF SSMs only outperform higher DOF ones as long as the actual warp to be estimated does not significantly exceed their capabilities.
- Similitude is the most robust SSM with most SMs as well as the most precise one with stochastic SMs, thus providing the best compromise between these two metrics.
- The three parameterizations of homography perform identically with all SMs.
- OLTs track with much lower precision than even 2 DOF RBTs and none can outperform the best of the RBTs in terms of robustness either. They are also 10-30 times slower than the faster RBTs.

Finally, and most importantly, a modular and highly extensible open source framework for RBT called MTF was introduced to demonstrate the practical viability of the proposed decomposition. MTF follows this decomposition closely through extensive use of generic programming to provide a convenient interface where new methods for any of the sub modules can be plugged in and combined with existing methods with minimum effort. With its unified architecture, MTF is specifically designed to serve as a robust experimental platform for combining diverse trackers to create novel composite trackers that can overcome the shortcomings of their constituents. Further, its highly efficient C++ implementation ensures that it can also be a practical solution for robotics and augmented reality applications that need fast and precise tracking. To this end, MTF has been designed to integrate well with popular libraries like ROS, OpenCV and ViSP so it can be easily used with existing as well as future projects. Finally, having all trackers tested here, including OLTs, implemented within MTF makes it easy to reproduce the results.

10.2 Future Work

MTF is still a work in progress and offers several promising avenues of future extensions for each of the sub modules. This includes the creation of novel composite SMs especially those that, like LMS and RANSAC, run a large number of relatively simple trackers simultaneously and combine their outputs to deduce the state of the tracked patch with greater precision and robustness than any single tracker can possibly provide. A partially implemented work in this direction is the line tracker (Sec. 7.1)

that does not work well yet and needs improvements including a more robust method for estimating best fit lines and better constrains between different lines. This is a simple example of using known geometrical constraints to detect and correct failing trackers though other more complex ones can also be imagined.

One of the most potentially beneficial ways to improve AMs is the incorporation of online learning to update the template. As mentioned in Sec. 7.2.2, MTF is designed to support this and two related modules - `DFM` [188] and `PCA` [99] - already exist that respectively utilize offline and online learning. They are both rather simple and out dated, however, and more sophisticated learning methods need to be implemented, especially those utilizing deep neural networks that have become popular in recent years. Another aspect of AMs that needs work is the extension of ILMs with better parameterization that can account for other sources of variations in the appearance of the object patch such as motion blur and occlusion. Further, support for SPI has only been implemented for SSD and NCC yet and needs to be added to other AMs too. Finally, the AMs need to be adapted to handle multi channel images since, as stated in Sec. A.3.3, the variants that exist currently merely concatenate pixels from all channels into a single vector that is then processed as if originating from a single channel image. These need to be improved to better utilize channel specific information. A related extension is the ability to handle depth information from 3D cameras like Kinect whose increasing ubiquity may make this the easiest way to improve tracking performance.

SSMs can be improved by using motion learning from annotated sequences to generate better stochastic samples for SSMs. It has been seen in Sec. 9.1.2 that the performance of stochastic SMs, as well as the corresponding composite ones, depends largely on the quality of samples so any progress in this direction should definitely be worthwhile. Addition of non rigid SSMs that can go beyond the basic planar projective transforms will also a useful extension though its application domain is somewhat limited. Two such SSMs - `Spline` and `TPS` - are already partially implemented so completing these would be the first step. SSMs that support 3D motion estimation are also needed to complement the above mentioned extension of AMs with depth information processing.

Several improvements are also needed in the implementations of existing methods to make them one practically useful. For instance, slower methods like PF, MI, CCRE, NGF and the grid tracker need to be efficiently parallelized for both multi core CPU and GPU configurations.

Bibliography

[1] J. Kwon, H. S. Lee, F. C. Park, and K. M. Lee, "A geometric particle filter for template-based visual tracking," *Pattern Analysis and Machine Intelligence (PAMI), IEEE Transactions on*, vol. 36, no. 4, pp. 625–643, 2014.

[2] G. Silveira and E. Malis, "Unified Direct Visual Tracking of Rigid and Deformable Surfaces Under Generic Illumination Changes in Grayscale and Color Images," *International Journal of Computer Vision*, vol. 89, no. 1, pp. 84–105, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11263-010-0324-z

[3] G. G. Scandaroli, M. Meilland, and R. Richa, "Improving NCC-based Direct Visual Tracking," in *ECCV*. Springer, 2012, pp. 442–455.

[4] G. H. Rogerio Richa, Raphael Sznitman, "Robust Similarity Measures for Gradient-based Direct Visual Tracking," CIRL, Tech. Rep., June 2012.

[5] R. Richa, M. Souza, G. Scandaroli, E. Comunello, and A. von Wangenheim, "Direct visual tracking under extreme illumination variations using the sum of conditional variance," in *Image Processing (ICIP), 2014 IEEE International Conference on*, Oct 2014, pp. 373–377.

[6] H. Yang, L. Shao, F. Zheng, L. Wang, and Z. Song, "Recent advances and trends in visual tracking: A review," *Neurocomputing*, vol. 74, no. 18, pp. 3823–3831, 2011.

[7] Y. Wu, J. Lim, and M.-H. Yang, "Online Object Tracking: A Benchmark," in *CVPR*, June 2013, pp. 2411–2418.

[8] M. J. L. A. Kristan, Matej *et al.*, "The Visual Object Tracking VOT2015 Challenge Results," in *Proceedings of the IEEE ICCV Workshops*, 2015, pp. 1–23.

[9] Y. Wu, J. Lim, and M.-H. Yang, "Object tracking benchmark," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1834–1848, 2015.

[10] A. Singh, A. Roy, X. Zhang, and M. Jagersand, "Modular Decomposition and Analysis of Registration based Trackers," in *CRV*, June 2016.

[11] A. W. Smeulders, D. M. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: An experimental survey," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 36, no. 7, pp. 1442–1468, 2014.

[12] Q. Liu, X. Zhao, and Z. Hou, "Survey of single-target visual tracking methods based on online learning," *IET Computer Vision*, vol. 8, no. 5, pp. 419–428, October 2014.

[13] M. Kristan, A. Leonardis, J. Matas, M. Felsberg *et al.*, *The Visual Object Tracking VOT2016 Challenge Results*. Cham: Springer International Publishing, 2016, pp. 777–823.

[14] N. Wang, J. Shi, D.-Y. Yeung, and J. Jia, "Understanding and diagnosing visual tracking systems," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3101–3109.

[15] R. Szeliski, "Image alignment and stitching: A tutorial," *Foundations and Trends® in Computer Graphics and Vision*, vol. 2, no. 1, pp. 1–104, 2006.

[16] B. D. Lucas and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," in *7th International Joint Conference on Artificial intelligence*, vol. 2, 1981, pp. 674–679.

[17] A. Singh, M. Siam, and M. Jagersand, "Unifying Registration based Tracking: A Case Study with Structural Similarity," 2017, arXiv:1607.04673 [cs.CV], accepted in WACV 2017.

[18] G. Bradski, "OpenCV," *Dr. Dobb's Journal of Software Tools*, 2000.

[19] J.-Y. Bouguet, "Pyramidal Implementation of the Lucas Kanade Feature Tracker: Description of the Algorithm," Intel Corporation Microprocessor Research Labs, Tech. Rep., 2000.

[20] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[21] G. D. Hager and K. Toyama, "Xvision: A portable substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, no. 1, pp. 23–37, 1998.

[22] G. D. Hager and P. N. Belhumeur, "Efficient Region Tracking With Parametric Models of Geometry and Illumination," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 10, pp. 1025–1039, October 1998.

[23] A. Harris and J. Conrad, "Survey of popular robotics simulators, frameworks, and toolkits," in *Southeastcon, 2011 Proceedings of IEEE*, March 2011, pp. 243–249.

[24] E. Marchand, F. Spindler, and F. Chaumette, "ViSP for visual servoing: a generic software platform with a wide class of robot control skills," *Robotics Automation Magazine, IEEE*, vol. 12, no. 4, pp. 40–52, Dec 2005.

[25] S. Baker and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework," *IJCV*, vol. 56, no. 3, pp. 221–255, Feb 2004.

[26] A. Yilmaz, O. Javed, and M. Shah, "Object Tracking: A Survey," *ACM Computing Surveys*, vol. 38, no. 4, Dec 2006. [Online]. Available: http://doi.acm.org/10.1145/1177352.1177355

[27] M. Danelljan, G. Hager, F. S. Khan, and M. Felsberg, "Discriminative Scale Space Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[28] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2544–2550.

[29] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[30] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "Exploiting the circulant structure of tracking-by-detection with kernels," in *European conference on computer vision*. Springer, 2012, pp. 702–715.

[31] ——, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[32] G. Nebehay and R. Pflugfelder, "Clustering of Static-Adaptive correspondences for deformable object tracking," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2784–2791.

[33] X. Zhang, A. Singh, and M. Jagersand, "RKLT: 8 DOF real-time robust video tracking combing coarse RANSAC features and accurate fast template registration," in *CRV*, 2015, pp. 70–77.

[34] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[35] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," in *Pattern Recognition (ICPR), 2010 20th International Conference on*, Aug 2010, pp. 2756–2759.

[36] S. Hare, A. Saffari, and P. H. Torr, "Struck: Structured output tracking with kernels," in *2011 International Conference on Computer Vision*. IEEE, 2011, pp. 263–270.

[37] S. Hare, S. Golodetz, A. Saffari, V. Vineet, M. M. Cheng, S. L. Hicks, and P. H. S. Torr, "Struck: Structured Output Tracking with Kernels," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 2096–2109, Oct 2016.

[38] M. B. Blaschko and C. H. Lampert, "Learning to localize objects with structured output regression," in *European conference on computer vision*. Springer, 2008, pp. 2–15.

[39] A. Bordes, L. Bottou, P. Gallinari, and J. Weston, "Solving multiclass support vector machines with LaRank," in *Proceedings of the 24th international conference on Machine learning*. ACM, 2007, pp. 89–96.

[40] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 2009, pp. 983–990.

[41] ——, "Robust object tracking with online multiple instance learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 8, pp. 1619–1632, 2011.

[42] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial intelligence*, vol. 89, no. 1, pp. 31–71, 1997.

[43] N. C. Oza and S. Russell, "Online ensemble learning," in *AAAI/IAAI*, 2000, p. 1109.

[44] M. G. H. Grabner and H. Bischof, "Real-time Tracking via On-line Boosting," in *British Machine Vision Conference*, vol. 1, 2006, pp. 47–56.

[45] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, #jul# 2012. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2011.239

[46] K. Zhang, L. Zhang, and M.-H. Yang, "Real-time compressive tracking," in *European Conference on Computer Vision.* Springer, 2012, pp. 864–877.

[47] D. L. Donoho, "Compressed sensing," *IEEE Transactions on information theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[48] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, vol. 1. IEEE, 2001, pp. I–511.

[49] A. Adam, E. Rivlin, and I. Shimshoni, "Robust fragments-based tracking using the integral histogram," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1. IEEE, 2006, pp. 798–805.

[50] P. J. Rousseeuw, "Least median of squares regression," *Journal of the American statistical association*, vol. 79, no. 388, pp. 871–880, 1984.

[51] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[52] F. Porikli, "Integral histogram: A fast way to extract histograms in cartesian spaces," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 829–836.

[53] D. Held, S. Thrun, and S. Savarese, *Learning to Track at 100 FPS with Deep Regression Networks*. Cham: Springer International Publishing, 2016, pp. 749–765.

[54] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 2014, pp. 675–678.

[55] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[56] C. Tomasi and T. Kanade, "Detection and Tracking of Point Features," Carnegie Mellon University, Tech. Rep. CMU-CS-91-132, April 1991.

[57] J. Shi and C. Tomasi, "Good Features to Track," in *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994*, June 1994, pp. 593 – 600.

[58] T. Tommasini, A. Fusiello, E. Trucco, and V. Roberto, "Making good features track better," in *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*. IEEE, 1998, pp. 178–183.

[59] Z. Zivkovic and F. van der Heijden, "Better features to track by estimating the tracking convergence region," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, vol. 2. IEEE, 2002, pp. 635–638.

[60] P. Anandan, "A computational framework and an algorithm for the measurement of visual motion," *International Journal of Computer Vision*, vol. 2, no. 3, pp. 283–310, 1989.

[61] J.-M. Odobez and P. Bouthemy, "Robust multiresolution estimation of parametric motion models," *Journal of visual communication and image representation*, vol. 6, no. 4, pp. 348–365, 1995.

[62] J.-Y. Bouguet, "Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker: Description of the algorithm," Intel Corporation Microprocessor Research Labs, Tech. Rep., 2001.

[63] M. Gleicher, "Projective registration with difference decomposition," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, Jun 1997, pp. 331–337.

[64] G. D. Hager and P. N. Belhumeur, "Real-time tracking of image regions with changes in geometry and illumination," in *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96, 1996 IEEE Computer Society Conference on.* IEEE, 1996, pp. 403–410.

[65] H. Jin, P. Favaro, and S. Soatto, "Real-Time Feature Tracking and Outlier Rejection with Changes in Illumination." in *ICCV*, 2001, pp. 684–689.

[66] F. Jurie and M. Dhome, "Hyperplane approximation for template matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 996–1000, 2002.

[67] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *CVPR*, vol. 1, 2001, pp. 1090–1097.

[68] ——, "Lucas-Kanade 20 Years On: A Unifying Framework: Part 1," *International Journal of Computer Vision*, vol. 56, pp. 221–255, 2002.

[69] S. Baker, T. I. Ralph Gross, and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework: Part 2," Carnegie Mellon University, Tech. Rep. CMU-RI-TR-03-01, 2003.

[70] S. Baker, R. Gross, and I. Matthews, "Lucas-Kanade 20 Years On: A Unifying Framework: Part 3," Carnegie Mellon University, Tech. Rep. CMU-RI-TR-03-35, 2003.

[71] ——, "Lucas-Kanade 20 Years On: A Unifying Framework: Part 4," Carnegie Mellon University, Tech. Rep. CMU-RI-TR-04-14, 2003.

[72] S. Baker, R. Patil, G. Cheung, and I. Matthews, "Lucas-kanade 20 years on: Part 5," Robotics Institute, Carnegie Mellon University, Tech. Rep., 2004.

[73] S. Benhimane and E. Malis, "Real-time image-based tracking of planes using efficient second-order minimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, Sept 2004, pp. 943–948 vol.1.

[74] E. Malis, "Improving vision-based control using efficient second-order minimization techniques," in *IEEE International Conference on Robotics and Automation*, vol. 2, April 2004, pp. 1843–1848 Vol.2.

[75] E. Malis and S. Benhimane, "A unified approach to visual tracking and servoing," *Robotics and Autonomous Systems*, vol. 52, no. 1, pp. 39–52, 2005.

[76] S. Benhimane and E. Malis, "Homography-based 2D Visual Tracking and Servoing," *Int. J. Rob. Res.*, vol. 26, no. 7, pp. 661–676, July 2007.

[77] G. Silveira and E. Malis, "Visual servoing from robust direct color image registration," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 5450–5455.

[78] O. Tahri and Y. Mezouar, "On visual servoing based on efficient second order minimization," *Robotics and Autonomous Systems*, vol. 58, no. 5, pp. 712–719, 2010.

[79] G. Silveira, E. Malis, and P. Rives, "An efficient direct method for improving visual SLAM," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 4090–4095.

[80] ——, "An efficient direct approach to visual SLAM," *IEEE transactions on robotics*, vol. 24, no. 5, pp. 969–979, 2008.

[81] Y. Park, V. Lepetit, and W. Woo, "Handling motion-blur in 3d tracking and rendering for augmented reality," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 9, pp. 1449–1459, 2012.

[82] C. Mei, S. Benhimane, E. Malis, and P. Rives, "Efficient homography-based tracking and 3-D reconstruction for single-viewpoint sensors," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1352–1364, 2008.

[83] S. Benhimane and E. Malis, "Integration of Euclidean constraints in template based visual tracking of piecewise-planar scenes," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, Oct 2006, pp. 1218–1223.

[84] G. Silveira and E. Malis, "Real-time visual tracking under arbitrary illumination changes," in *CVPR. IEEE Conference on*, 2007, pp. 1–6.

[85] ——, "Unified direct visual tracking of rigid and deformable surfaces under generic illumination changes in grayscale and color images," *International journal of computer vision*, vol. 89, no. 1, pp. 84–105, 2010.

[86] C. Mei, S. Benhimane, E. Malis, and P. Rives, "Constrained Multiple Planar Template Tracking for Central Catadioptric Cameras," in *BMVC*, 2006, pp. 619–628.

[87] Y. Park, V. Lepetit, and W. Woo, "ESM-Blur: Handling & rendering blur in 3D tracking and augmentation," in *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2009, pp. 163–166.

[88] E. Ito, T. Okatani, and K. Deguchi, "Accurate and robust planar tracking based on a model of image sampling and reconstruction process," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 1–8.

[89] Y. Keller and A. Averbuch, "Fast motion estimation using bidirectional gradient methods," *IEEE Transactions on Image Processing*, vol. 13, no. 8, pp. 1042–1054, 2004.

[90] N. Dowson and R. Bowden, "Mutual Information for Lucas-Kanade Tracking (MILK): An Inverse Compositional Formulation," *PAMI*, vol. 30, no. 1, pp.

180–185, Jan 2008. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2007.70757

[91] G. Panin and A. Knoll, "Mutual Information-Based 3D Object Tracking," *International Journal of Computer Vision*, vol. 78, no. 1, pp. 107–118, 2008. [Online]. Available: http://dx.doi.org/10.1007/s11263-007-0083-7

[92] A. Dame and E. Marchand, "Accurate real-time tracking using mutual information," in *IEEE International Symposium on Mixed and Augmented Reality*, 2010, pp. 47–56.

[93] A. Dame, "A unified direct approach for visual servoing and visual tracking using mutual information," Ph.D. dissertation, University of Rennes, 2010.

[94] A. Dame and E. Marchand, "Second-Order Optimization of Mutual Information for Real-Time Image Registration," *Image Processing, IEEE Transactions on*, vol. 21, no. 9, pp. 4190–4203, Sept 2012.

[95] R. Richa, R. Sznitman, R. Taylor, and G. Hager, "Visual tracking using the sum of conditional variance," in *IROS*, Sept 2011, pp. 2953–2958.

[96] B. Delabarre and E. Marchand, "Dense non-rigid visual tracking with a robust similarity function," in *2014 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2014, pp. 4942–4946.

[97] ——, "Visual servoing using the sum of conditional variance," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 1689–1694.

[98] T. Dick, C. Perez, M. Jagersand, and A. Shademan, "Realtime Registration-Based Tracking via Approximate Nearest Neighbour Search," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, June 2013.

[99] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, "Incremental Learning for Robust Visual Tracking," *IJCV*, vol. 77, no. 1-3, pp. 125–141, May 2008.

[100] C. Bao, Y. Wu, H. Ling, and H. Ji, "Real time robust l1 tracker using accelerated proximal gradient approach," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1830–1837.

[101] X. Mei, H. Ling, Y. Wu, E. Blasch, and L. Bai, "Minimum Error Bounded Efficient L1 Tracker with Occlusion Detection," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, June, pp. 1257–1264.

[102] X. Zhang, W. Hu, S. Maybank, and X. Li, "Graph based discriminative learning for robust and efficient object tracking," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[103] Y. Liu, G. Li, and Z. Shi, "Covariance tracking via geometric particle filtering," *EURASIP Journal on Advances in Signal Processing*, vol. 2010, no. 1, p. 1, 2010.

[104] S. Liu, T. Fang, S. Chen, H. Tong, C. Yuan, and Z. Chen, "Particle filter with affine transformation for multiple key points tracking," in *Transactions on Edutainment VIII*. Springer, 2012, pp. 112–126.

[105] J. Kwon, K. M. Lee, and F. C. Park, "Visual tracking via geometric particle filtering on the affine group with optimal importance functions," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 991–998.

[106] M. Li, T. Tan, W. Chen, and K. Huang, "Efficient object tracking by incremental self-tuning particle filtering on the affine group," *IEEE Transactions on Image Processing*, vol. 21, no. 3, pp. 1298–1313, 2012.

[107] R. P. Woods, S. R. Cherry, and J. C. Mazziotta, "Rapid automated algorithm for aligning and reslicing PET images," *J. Comput. Assist. Tomogr.*, vol. 16, no. 4, pp. 620–633, July 1992.

[108] R. P. Woods, J. C. Mazziotta, S. R. Cherry *et al.*, "MRI-PET registration with automated algorithm," *Journal of computer assisted tomography*, vol. 17, no. 4, pp. 536–546, 1993.

[109] R. P. Woods, S. T. Grafton, C. J. Holmes, S. R. Cherry, and J. C. Mazziotta, "Automated image registration: I. General methods and intrasubject, intramodality validation," *Journal of computer assisted tomography*, vol. 22, no. 1, pp. 139–152, 1998.

[110] J. V. . Hajnal, D. J. . Hawkes, and D. L. . G. . Hill, Eds., *Medical Image Registration*, ser. Biomedical Engineering. CRC Press, June 2001, ch. Registration Methodology: Concepts and AIgorithms, pp. 39–70.

[111] E. Haber and J. Modersitzki, "Beyond mutual information: A simple and robust alternative," in *Bildverarbeitung für die Medizin 2005*. Springer, 2005, pp. 350–354.

[112] ——, "Intensity gradient based registration and fusion of multi-modal images," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2006, pp. 726–733.

[113] E. Hodneland, A. Lundervold, J. Rørvik, and A. Z. Munthe-Kaas, "Normalized gradient fields for nonlinear motion correction of DCE-MRI time series," *Computerized Medical Imaging and Graphics*, vol. 38, no. 3, pp. 202–210, 2014.

[114] L. König and J. Rühaak, "A fast and accurate parallel algorithm for non-linear image registration using normalized gradient fields," in *2014 IEEE 11th international symposium on biomedical imaging (ISBI)*. IEEE, 2014, pp. 580–583.

[115] Z. Wang and A. C. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, March 2002.

[116] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *Image Processing, IEEE Transactions on*, vol. 13, no. 4, pp. 600–612, April 2004.

[117] A. Łoza, L. Mihaylova, D. Bull, and N. Canagarajah, "Structural similarity-based object tracking in multimodality surveillance videos," *Machine Vision and Applications*, vol. 20, no. 2, pp. 71–83, 2009.

[118] A. Loza, F. Wang, M. A. Patricio, J. García, and J. M. Molina, "Optimised Particle Filter Approaches to Object Tracking in Video Sequences," in *Methods and Models in Artificial and Natural Computation*. Springer, 2009, pp. 486–495.

[119] A. Loza, F. Wang, J. Yang, and L. Mihaylova, "Video object tracking with differential Structural SIMilarity index," in *ICASSP, IEEE International Conference on*, May 2011, pp. 1405–1408.

[120] ——, *Structural Information Approaches to Object Tracking in Video Sequences*. INTECH Open Access Publisher, 2011.

[121] X. Wang, C. Ning, A. Shi, and G. Lv, "An improved similarity measure in particle filters for robust object tracking," in *Image and Signal Processing (CISP), 2013 6th International Congress on*, vol. 1. IEEE, 2013, pp. 46–50.

[122] R. Brooks and T. Arbel, "Generalizing inverse compositional image alignment," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 2. IEEE, 2006, pp. 1200–1203.

[123] ——, "Generalizing Inverse Compositional and ESM Image Alignment," *IJCV*, vol. 87, no. 3, pp. 191–212, May 2010.

[124] D. Schreiber, "Generalizing the Lucas–Kanade algorithm for histogram-based tracking," *Pattern Recognition Letters*, vol. 29, no. 7, pp. 852–861, 2008.

[125] D. Cobzas, M. Jagersand, and P. Sturm, "3D SSD tracking with estimated 3D planes," *Image and Vision Computing*, vol. 27, no. 1, pp. 69–79, 2009.

[126] S. Lucey, R. Navarathna, A. B. Ashraf, and S. Sridharan, "Fourier lucas-kanade algorithm," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 6, pp. 1383–1396, 2013.

[127] M. Lourenço and J. P. Barreto, "Tracking feature points in uncalibrated images with radial distortion," in *European Conference on Computer Vision*. Springer, 2012, pp. 1–14.

[128] Y. Niu, Z. Xu, and X. Che, "Dynamically Removing False Features in Pyramidal Lucas-Kanade Registration," *IEEE Transactions on Image Processing*, vol. 23, no. 8, pp. 3535–3544, 2014.

[129] Y. Keller and A. Averbuch, "Global parametric image alignment via high-order approximation," *Computer Vision and Image Understanding*, vol. 109, no. 3, pp. 244–259, 2008.

[130] B. Amberg, A. Blake, and T. Vetter, "On compositional image alignment, with an application to active appearance models," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1714–1721.

[131] E. Antonakos, J. Alabort-i Medina, G. Tzimiropoulos, and S. P. Zafeiriou, "Feature-based lucas–kanade and active appearance models," *IEEE Transactions on Image Processing*, vol. 24, no. 9, pp. 2617–2632, 2015.

[132] S. Oron, A. Bar-Hille, and S. Avidan, "Extended lucas-kanade tracking," in *European Conference on Computer Vision*. Springer, 2014, pp. 142–156.

[133] M. Trummer, J. Denzler, and C. Munkelt, "Guided KLT Tracking Using Camera Parameters in Consideration of Uncertainty," in *International Conference on Computer Vision and Computer Graphics*. Springer, 2008, pp. 252–261.

[134] P. Li and Q. Wang, "Robust registration-based tracking by sparse representation with model update," in *Asian Conference on Computer Vision*. Springer, 2012, pp. 205–216.

[135] B. Poling, G. Lerman, and A. Szlam, "Better feature tracking through subspace constraints," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3454–3461.

[136] E. Muñoz, P. Márquez-Neila, and L. Baumela, "Rationalizing Efficient Compositional Image Alignment," *International Journal of Computer Vision*, vol. 112, no. 3, pp. 354–372, 2015.

[137] H.-Y. Shum and R. Szeliski, "Construction of Panoramic Image Mosaics with Global and Local Alignment," *IJCV*, vol. 36, no. 2, pp. 101–130.

[138] A. Bartoli, "Direct image registration with gain and bias," *Contributions au recalage dimages et à la reconstruction 3D de scènes rigides et déformables*, p. 4, 2006.

[139] ——, "Groupwise geometric and photometric direct image registration," *TPAMI*, vol. 30, no. 12, pp. 2098–2108, 2008.

[140] G. Silveira, "Photogeometric Direct Visual Tracking for Central Omnidirectional Cameras," *Journal of Mathematical Imaging and Vision*, vol. 48, no. 72, October 2014.

[141] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[142] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[143] O. F. Nikos Paragios, Yunmei Chen, *Handbook of Mathematical Models in Computer Vision*. Springer, 2006.

[144] J. Modersitzki, *FAIR: Flexible Algorithms for Image Registration*, 2009.

[145] A. D. Jepson, D. J. Fleet, and T. F. El-Maraghi, "Robust online appearance models for visual tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 10, pp. 1296–1311, 2003.

[146] H. Firouzi and H. Najjaran, "Adaptive on-line similarity measure for direct visual tracking," *Image and Vision Computing*, vol. 32, no. 4, pp. 227 – 236, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0262885614000195

[147] X. Mei and H. Ling, "Robust Visual Tracking using L1 Minimization," in *IEEE 12th International Conference on Computer Vision*, September 2009, pp. 1436 – 1443.

[148] X. Mei, H. Ling, Y. Wu, E. P. Blasch, and L. Bai, "Efficient minimum error bounded particle resampling L1 tracker with occlusion detection," *IEEE Transactions on Image Processing*, vol. 22, no. 7, pp. 2661–2675, 2013.

[149] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.

[150] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 4th ed. John Wiley & Sons, 2013.

[151] S. Damas, O. Cordn, and J. Santamara, "Medical Image Registration Using Evolutionary Computation: An Experimental Survey," *IEEE Computational Intelligence Magazine*, vol. 6, no. 4, pp. 26–42, Nov 2011.

[152] R. Mégret, J.-B. Authesserre, and Y. Berthoumieu, "The bi-directional framework for unifying parametric image alignment approaches," in *European Conference on Computer Vision*. Springer, 2008, pp. 400–411.

[153] F. Dellaert and R. Collins, "Fast image-based tracking by selective pixel integration," in *Proceedings of the ICCV Workshop on Frame-Rate Vision*, 1999, pp. 1–22.

[154] S. Benhimane, A. Ladikos, V. Lepetit, and N. Navab, "Linear and quadratic subsets for template-based tracking," in *CVPR*. IEEE, 2007, pp. 1–6.

[155] C. Kelley, *Iterative Methods for Optimization*. Society for industrial and Applied Mathematics, 1995.

[156] M. Muja and D. G. Lowe, "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." *VISAPP (1)*, vol. 2, pp. 331–340, 2009.

[157] S. Gu, Y. Zheng, and C. Tomasi, "Efficient visual object tracking with online nearest neighbor classifier," in *Computer vision–ACCV 2010*. Springer, 2010, pp. 271–282.

[158] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[159] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*. IEEE, 1997, pp. 1000–1006.

[160] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[161] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-nearest neighbors," *IEEE transactions on computers*, vol. 100, no. 7, pp. 750–753, 1975.

[162] B. Leibe, K. Mikolajczyk, and B. Schiele, "Efficient Clustering and Matching for Object Class Recognition," in *BMVC*, 2006, pp. 789–798.

[163] K. Mikolajczyk and J. Matas, "Improving descriptors for fast tree matching by optimal linear projection," in *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 2007, pp. 1–8.

[164] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, p. 1312.

[165] A. Doucet, N. de Freitas, and N. Gordon, Eds., *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[166] M. Isard and A. Blake, "CONDENSATION - conditional density propagation for visual tracking," *International Journal of Computer Vision*, vol. 29, pp. 5–28, 1998.

[167] Z. Khan, T. Balch, and F. Dellaert, "A Rao-Blackwellized particle filter for EigenTracking," in *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 2, June 2004, pp. II–980–II–986 Vol.2.

[168] S. K. Zhou, R. Chellappa, and B. Moghaddam, "Visual tracking and recognition using appearance-adaptive models in particle filters," *IEEE Transactions on Image Processing*, vol. 13, no. 11, pp. 1491–1506, 2004.

[169] C. Yang, R. Duraiswami, and L. Davis, "Fast multiple object tracking via a hierarchical particle filter," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 1. IEEE, 2005, pp. 212–219.

[170] J. Wang, X. Chen, and W. Gao, "Online selecting discriminative tracking features using particle filter," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2. IEEE, 2005, pp. 1037–1042.

[171] K. Hotta, "Adaptive weighting of local classifiers by particle filters for robust tracking," *Pattern Recognition*, vol. 42, no. 5, pp. 619–628, 2009.

[172] J. Xavier and J. H. Manton, "On the generalization of AR processes to Riemannian manifolds," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 5. IEEE, 2006, pp. V–V.

[173] J. Liu and M. West, "Combined parameter and state estimation in simulation-based filtering," in *Sequential Monte Carlo methods in practice*. Springer, 2001, pp. 197–223.

[174] J. Kwon and F. C. Park, "Visual tracking via particle filtering on the affine group," *The International Journal of Robotics Research*, 2009.

[175] J.-M. Marin, K. Mengersen, and C. P. Robert, "Bayesian modelling and inference on mixtures of distributions," *Handbook of statistics*, vol. 25, pp. 459–507, 2005.

[176] M. Isard and A. Blake, "A mixed-state condensation tracker with automatic model-switching," in *Computer Vision, 1998. Sixth International Conference on.* IEEE, 1998, pp. 107–112.

[177] M. Bolić, P. M. Djurić, and S. Hong, "Resampling algorithms for particle filters: A computational complexity perspective," *EURASIP Journal on Applied Signal Processing*, vol. 2004, pp. 2267–2277, 2004.

[178] R. Douc and O. Cappé, "Comparison of resampling schemes for particle filtering," in *ISPA 2005. Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, 2005.* IEEE, 2005, pp. 64–69.

[179] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *Nonlinear Statistical Signal Processing Workshop, 2006 IEEE.* IEEE, 2006, pp. 79–82.

[180] E. Begelfor and M. Werman, "How to put probabilities on homographies," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1666–1670, 2005.

[181] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection.* New York: Wiley, 1987.

[182] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International journal of computer vision*, vol. 74, no. 1, pp. 59–73, 2007.

[183] M. A. Fischler and R. C. Bolles, "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, #jun# 1981. [Online]. Available: http://doi.acm.org/10.1145/358669.358692

[184] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, March 2004.

[185] J. Gonzalez-Mora, N. Guil, E. L. Zapata, and F. De La Torre, "Efficient image alignment using linear appearance models," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 2230–2237.

[186] T. F. Cootes, G. J. Edwards, and C. J. Taylor, "Active appearance models," in *European conference on computer vision*. Springer, 1998, pp. 484–498.

[187] X. Gao, Y. Su, X. Li, and D. Tao, "A review of active appearance models," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 40, no. 2, pp. 145–158, 2010.

[188] M. Siam, "CNN Based Appearance Model with Approximate Nearest Neigbour Search," University of Alberta, Tech. Rep., 2015. [Online]. Available: http://webdocs.cs.ualberta.ca/~vis/mtf/dfm_report.pdf

[189] M. R. Pickering, A. A. Muhit, J. M. Scarvell, and P. N. Smith, "A new multi-modal similarity measure for fast gradient-based 2d-3d image registration," in *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 2009, pp. 5821–5824.

[190] P. Viola and W. M. Wells III, "Alignment by maximization of mutual information," *International journal of computer vision*, vol. 24, no. 2, pp. 137–154, 1997.

[191] P. Thevenaz and M. Unser, "Optimization of mutual information for multiresolution image registration," *Image Processing, IEEE Transactions on*, vol. 9, no. 12, pp. 2083–2099, Dec 2000.

[192] L. Ruthotto, "Mass-preserving registration of medical images," *German Diploma Thesis (Mathematics), Institute for Computational and Applied Mathematics, University of Münster*, 2010.

[193] A. Jain, K. Nandakumar, and A. Ross, "Score normalization in multimodal biometric systems," *Pattern recognition*, vol. 38, no. 12, pp. 2270–2285, 2005.

[194] F. Zhao, Q. Huang, and W. Gao, "Image matching by normalized cross-correlation," in *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, vol. 2. IEEE, 2006, pp. II–II.

[195] K. Briechle and U. D. Hanebeck, "Template matching using fast normalized cross correlation," in *Aerospace/Defense Sensing, Simulation, and Controls*. International Society for Optics and Photonics, 2001, pp. 95–102.

[196] M. Mori and K. Kashino, "Fast template matching based on normalized cross correlation using adaptive block partitioning and initial threshold estimation," in *Multimedia (ISM), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 196–203.

[197] M. Irani and P. Anandan, "Robust multi-sensor image alignment," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 959–966.

[198] P. Sebastian and Y. V. Voon, "Tracking using normalized cross correlation and color space," in *Intelligent and Advanced Systems, 2007. ICIAS 2007. International Conference on*. IEEE, 2007, pp. 770–774.

[199] L. Sun and Z. Mao, "An improved Normalized Cross Correlation algorithm for object tracking," in *IEEE 10th INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING PROCEEDINGS*. IEEE, 2010, pp. 1267–1270.

[200] J. Santner, C. Leistner, A. Saffari, T. Pock, and H. Bischof, "PROST: Parallel robust online simple tracking," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 723–730.

[201] C. Fookes and M. Bennamoun, "The use of mutual information for rigid medical image registration: a review," in *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, vol. 4. IEEE, 2002, pp. 6–pp.

[202] J. P. Pluim, J. A. Maintz, and M. A. Viergever, "Mutual-information-based registration of medical images: a survey," *IEEE transactions on medical imaging*, vol. 22, no. 8, pp. 986–1004, 2003.

[203] A. Collignon, F. Maes, D. Delaere, D. Vandermeulen, P. Suetens, and G. Marchal, "Automated multi-modality image registration based on information theory," in *Information processing in medical imaging*, vol. 3, no. 6, 1995, pp. 263–274.

[204] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, "Multimodality image registration by maximization of mutual information," *IEEE transactions on Medical Imaging*, vol. 16, no. 2, pp. 187–198, 1997.

[205] E. Parzen, "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[206] F. Wang, B. C. Vemuri, M. Rao, and Y. Chen, "Cumulative residual entropy, a new measure of information & its application to image alignment," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on.* IEEE, 2003, pp. 548–553.

[207] M. Rao, Y. Chen, B. C. Vemuri, and F. Wang, "Cumulative residual entropy: a new measure of information," *IEEE transactions on Information Theory*, vol. 50, no. 6, pp. 1220–1228, 2004.

[208] F. Wang and B. C. Vemuri, "Non-rigid multi-modal image registration using cross-cumulative residual entropy," *IJCV*, vol. 74, no. 2, pp. 201–215, 2007.

[209] M. Hasan, M. Pickering, A. Robles-Kelly, J. Zhou, and X. Jia, "Regisration of hyperspectral and trichromatic images via cross cumulative residual entropy maximisation," in *2010 IEEE International Conference on Image Processing.* IEEE, 2010, pp. 2329–2332.

[210] M. Hasan, M. R. Pickering, and X. Jia, "Robust automatic registration of multimodal satellite images using CCRE with partial volume interpolation," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 10, pp. 4050–4061, 2012.

[211] J. Lee, X. Cai, C.-B. Schönlieb, and D. A. Coomes, "Nonparametric image registration of airborne LiDAR, hyperspectral and photographic imagery of wooded landscapes," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 11, pp. 6073–6084, 2015.

[212] J. Rühaak, L. König, M. Hallmann, N. Papenberg, S. Heldmann, H. Schumacher, and B. Fischer, "A fully parallel algorithm for multimodal image registration using normalized gradient fields," in *2013 IEEE 10th International Symposium on Biomedical Imaging.* IEEE, 2013, pp. 572–575.

[213] L. König, A. Derksen, M. Hallmann, and N. Papenberg, "Parallel and memory efficient multimodal image registration for radiotherapy using normalized gradient fields," in *2015 IEEE 12th international symposium on biomedical imaging (ISBI)*. IEEE, 2015, pp. 734–738.

[214] F. Tramnitzke, J. Rühaak, L. König, J. Modersitzki, and H. Köstler, "GPU based affine linear image registration using normalized gradient fields," in *Proc. Seventh International Workshop on High Performance Computing for Biomedical Image Analysis (HPC-MICCAI), Boston, MA, USA*, 2014.

[215] M. Ebrahimi, A. Lausch, and A. L. Martel, "A Gauss–Newton approach to joint image registration and intensity correction," *Computer methods and programs in biomedicine*, vol. 112, no. 3, pp. 398–406, 2013.

[216] J. C. Carr, W. R. Fright, and R. K. Beatson, "Surface interpolation with radial basis functions for medical imaging," *IEEE transactions on medical imaging*, vol. 16, no. 1, pp. 96–107, 1997.

[217] F. L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *PAMI, IEEE Transactions on*, no. 6, pp. 567–585, 1989.

[218] R. Szeliski and J. Coughlan, "Spline-based image registration," *IJCV*, vol. 22, no. 3, pp. 199–218, 1997.

[219] A. Shashua and Y. Wexler, "Q-warping: Direct computation of quadratic reference surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 8, pp. 920–925, 2001.

[220] A. K. Jr, *An Introduction to Lie Groups and Lie Algebras*, ser. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2008.

[221] B. C. Hall, "An elementary introduction to groups and representations," *arXiv preprint math-ph/0005032*, 2000.

[222] E. Begelfor and M. Werman, "How to put probabilities on homographies," *IEEE transactions on pattern analysis and machine intelligence*, vol. 27, no. 10, pp. 1666–1670, 2005.

[223] J. Gallier and J. Quaintance, *Notes on Differential Geometry and Lie Groups*. Unpublished, available at http://www.seas.upenn.edu/ jean/diffgeom.pdf, November 2016, ch. 17: The Derivative of exp and Dynkins Formula, pp. 499–502.

[224] D. Lancaster, "A review of some image pixel interpolation algorithms," *copyright c2007 as Guru Gram*, vol. 32, 2007.

[225] D. Eberly, "Least-Squares Fitting of Data with B-Spline Surfaces, Geometric Tools, LLC," 2008.

[226] P. J. Rousseeuw, "Least Median of Squares Regression," *J. Am. Stat. Assoc.*, vol. 79, no. 388, pp. 871–880, 1984.

[227] "Eigen: A C++ template library for linear algebra," http://eigen.tuxfamily.org, accessed: 2017-02-04.

[228] "Eigen Benchmark," http://eigen.tuxfamily.org/index.php?title=Benchmark, accessed: 2017-02-04.

[229] A. Roy, X. Zhang, N. Wolleb, C. Perez Quintero, and M. Jagersand, "Tracking benchmark and evaluation for manipulation tasks," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 2448–2453.

[230] S. Gauglitz, T. Höllerer, and M. Turk, "Evaluation of interest point detectors and feature descriptors for visual tracking," *International journal of computer vision*, vol. 94, no. 3, pp. 335–360, 2011.

[231] K. Zimmermann, J. Matas, and T. Svoboda, "Tracking by an optimal sequence of linear predictors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 4, pp. 677–692, 2009.

[232] A. Crivellaro and V. Lepetit, "Robust 3d tracking with descriptor fields," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014, pp. 3414–3421.

[233] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab, "A dataset and evaluation methodology for template-based tracking algorithms," in *SMAR*. IEEE, 2009, pp. 145–151.

[234] L. Čehovin, M. Kristan, and A. Leonardis, "Is my new tracker really better than yours?" in *IEEE Winter Conference on Applications of Computer Vision*. IEEE, 2014, pp. 540–547.

[235] "Motion Analysis and Object Tracking," http://docs.opencv.org/2.4/modules/video/doc/motion_analysis_and_object_tracking.html, accessed: 2016-12-16.

[236] P. Lucey, S. Lucey, M. Cox, S. Sridharan, and J. Cohn, "Comparing object alignment algorithms with appearance variation: Forward-additive vs inverse-composition," in *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*. IEEE, 2008, pp. 337–342.

[237] L. Čehovin, A. Leonardis, and M. Kristan, "Visual object tracking performance measures revisited," *IEEE Transactions on Image Processing*, vol. 25, no. 3, pp. 1261–1274, 2016.

[238] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 15–22.

Appendix A

Experiments with Parameters

This appendix presents the results of experiments conducted to determine the optimal parameter values that were used for the results reported in chapter 9.

A.1 Gradient Descent Search

Following parameters have been tested to fine tune the performance of SMs in this category:

A.1.1 Hessian Type

This section compares the LM formulation of $\hat{\mathbf{H}}_{self}$ (Sec. 4.1.2.1) with the simpler GN approach. Fig. A.1 presents the results for FCLK and ICLK while Fig. A.2 does so for IALK. ESM and FALK gave similar results as FCLK and so have been omitted. Following observations can be made there:

- LM and GN are quite similar for most AMs with FCLK (Fig. A.1 (a), (b)) though LM seems slightly better overall. It is notably better with MI, ZNCC and NGF while GN is only so with RSCV. This agrees well with the synthetic results reported in [25] though they were only for SSD.

- This largely holds with ICLK too (Fig. A.1 (c), (d)), though several AMs - SSD, SPSS, ZNCC and NGF - show more significant improvement here. The large increase in SR with NGF for smaller t_p is particularly noteworthy as being indicative of the significantly greater stability of this AM with LM due to the latter helping to compensate for the approximate $\hat{\mathbf{H}}_{self}$ that has been used here (Sec. 5.2.7). However, LM simultaneously exhibits higher FR as well as lower

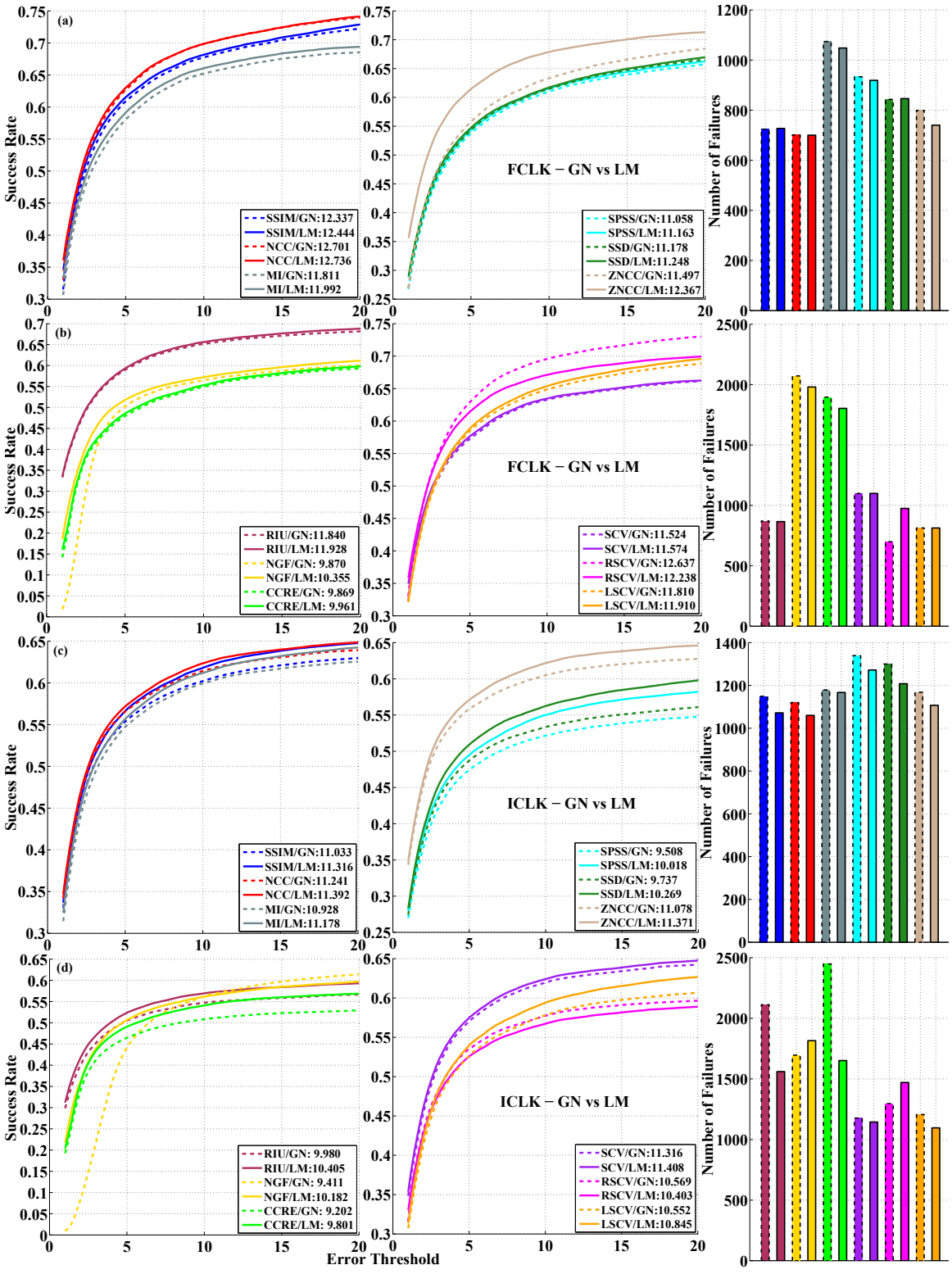


Figure A.1: Comparing GN and LM formulations of $\hat{\mathbf{H}}_{self}$ with FCLK and ICLK

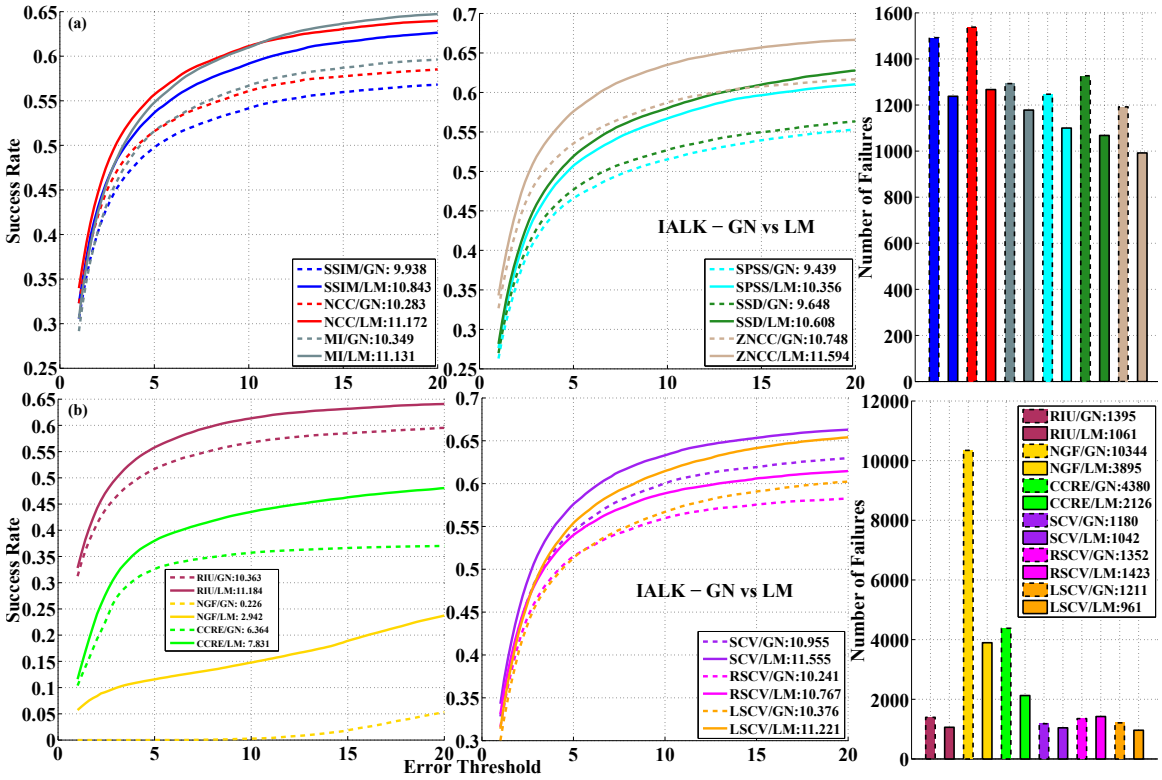


Figure A.2: Comparing GN and LM formulations of $\hat{\mathbf{H}}_{self}$ with IALK

SR for larger t_p, both indicative of a decrease in robustness. This is probably due to the somewhat high occurrence rate of false maxima in its function surface (Fig. 5.13) which can guide updates to the LM δ in the wrong direction, thus leading to failures in challenging scenarios.

- This is probably also the reason why RSCV performs so poorly with LM as its surface is noticeably craggier (Fig. 5.4) than all other AMs.

- IALK gains a lot more from using LM than all other SMs and actually ends up outperforming ICLK with most AMs. It would seem that the main reason for its poor performance with GN is that the approximation in Eq. 4.9 leads to poor step sizes and this is compensated well by LM.

A.1.2 Maximum Iterations

This is the maximum number of iterations that these SMs were allowed to run per frame subject to the condition that the L2 norm of the change in tracker location

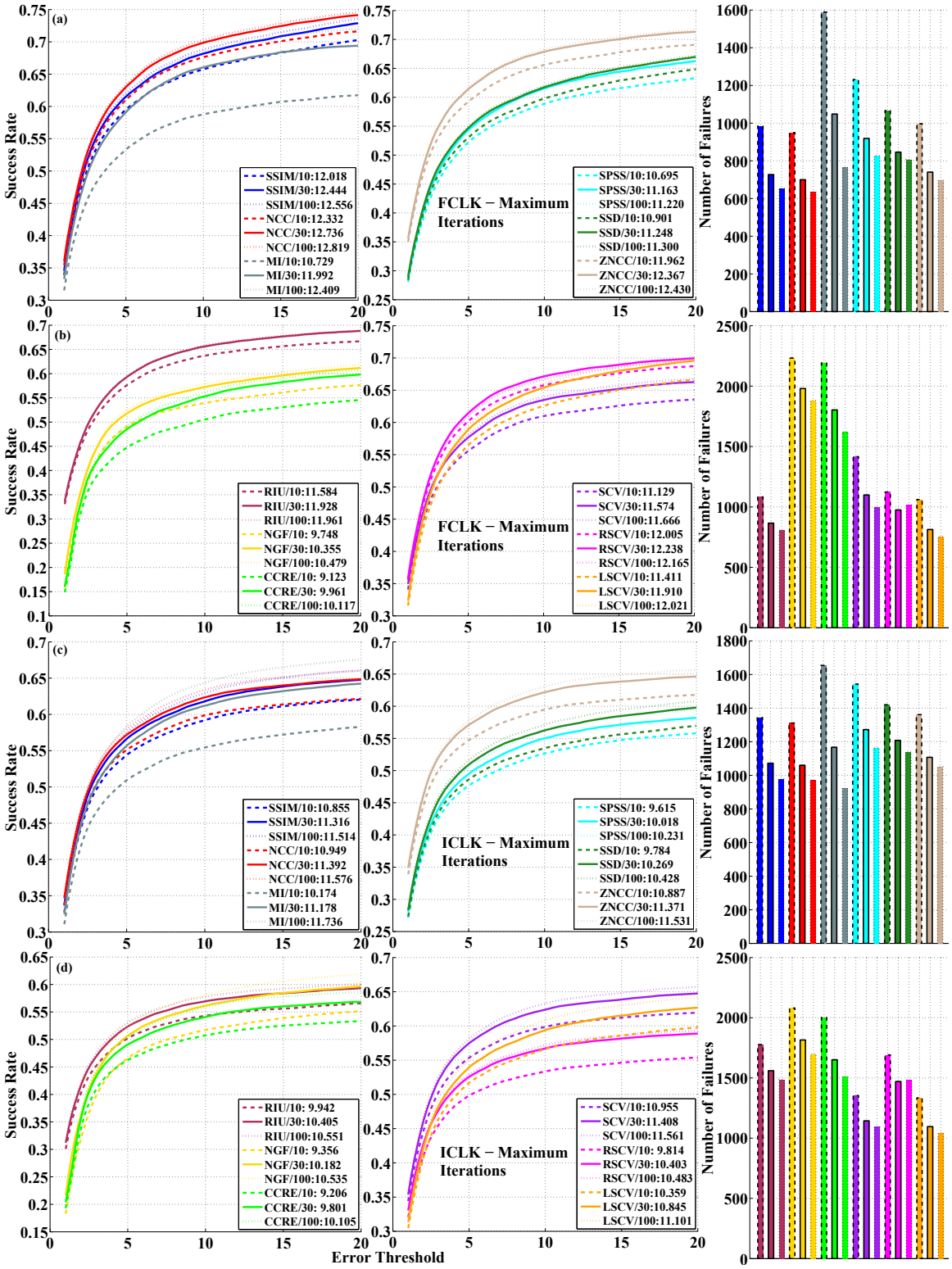


Figure A.3: Effect of maximum iterations on FCLK and ICLK. Dashed, solid, and dotted lines respectively show 10, 30 and 100 iterations.

between consecutive iterations exceeded 10^{-4}. Fig. A.3 shows the effect of this on different AMs using FCLK and ICLK. The remaining SMs have been omitted as ESM and FALK behaved similarly to FCLK while IALK was similar to ICLK. Following observations can be made:

- FCLK seems to benefit somewhat more than ICLK on increasing the iterations from 10 to 30 which is rather counter intuitive since ICLK, with it constant Hessian, would be expected to need more iterations to converge. However, the absolute performance of FCLK with 10 iterations is better than that of ICLK with 30, and even 100, so it seems that more iterations do not help ICLK much with finding convergence.

- Allowing more iterations than 30 only benefit FCLK significantly with MI and ZNCC as far as the SR is concerned though the FR does decrease with most AMs. This latter too is most significant for MI whose FR decreases by over 20%.

- Contrary to the first point, ICLK seems to improve more than FCLK on increasing the iterations from 30 to 100. When combined with the earlier point, this indicates that ICLK does indeed benefit from more iterations, as expected, but also converges more *slowly* so that the gain in going from 10 to 30 is less marked than with FCLK. Also, the decrease in FR is more marked than increase in SR indicating that more iterations help mainly to improve robustness and not precision.

- Though not shown here, ESM gave almost identical results as FCLK which belies its supposedly faster second order convergence.

A.1.3 Sampling Resolution

As described in Sec. 3.1, this determines the number of pixels used for sampling the object patch to be tracked. Increasing the resolution makes the function surface of f smoother and thus helps these SMs converge better by reducing false maxima resulting from sampling artifacts.. Fig. A.4 shows the impact of this parameter on FCLK and ICLK. Remaining SMs have been excluded for the same reason as in the last section. Following observations can be made:

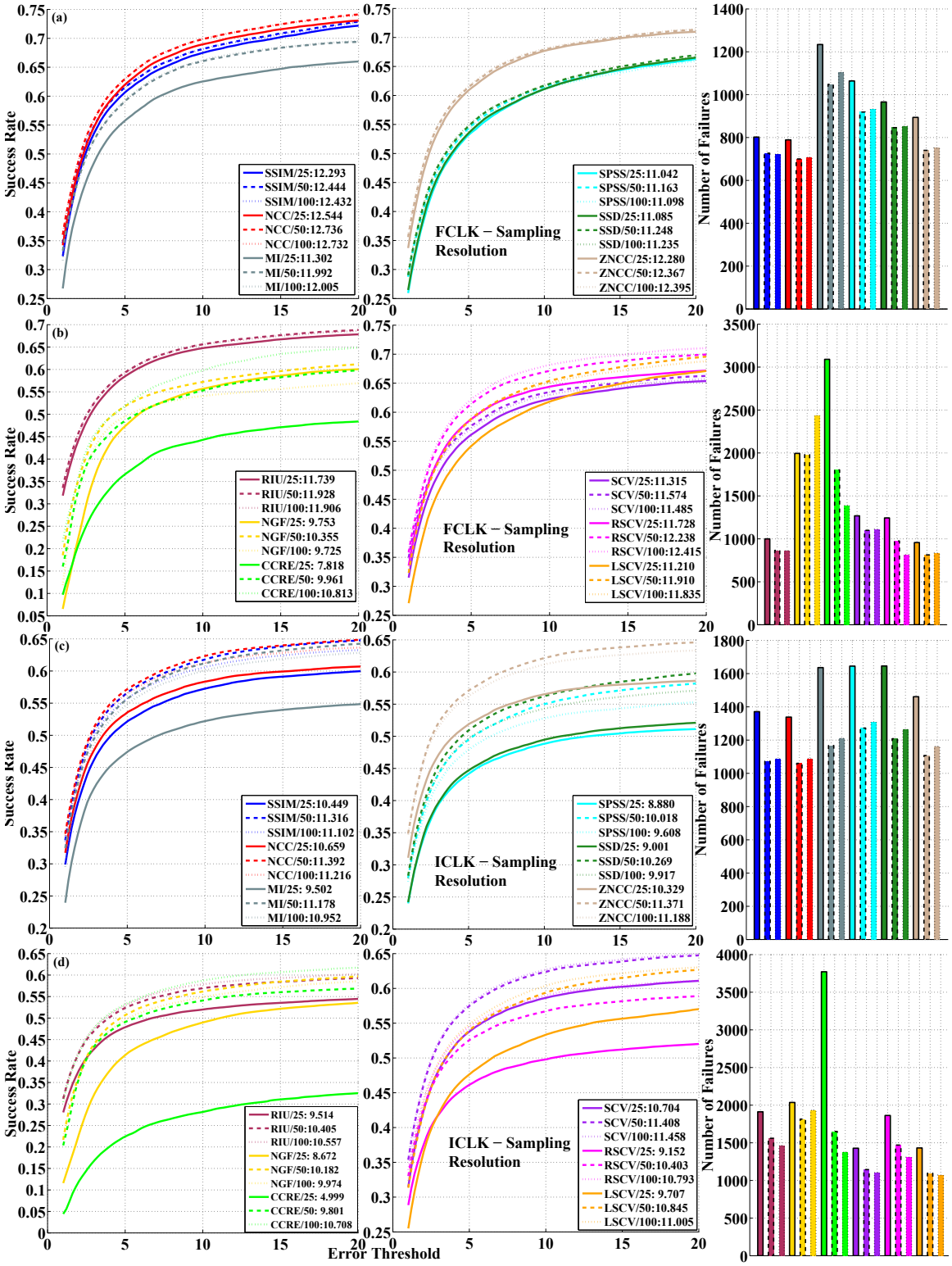


Figure A.4: Effect of sampling resolution on FCLK and ICLK. Solid, dashed and dotted lines respectively represent 25×25, 50×50 and 100×100 resolutions.

- CCRE is the only AM that improves significantly on increasing the sampling resolution from 50×50 to 100×100 and also shows the largest improvement on going from 25×25 to 50×50. It appears that the cumulative joint histograms used by it benefit from having more pixels to work with. This is also indicated by the fact that other AMs based on joint histograms - MI, SCV RSCV and LSCV - show comparatively greater improvement on going from 25×25 to 50×50 than others.

- NGF performs worse, in terms of both SR and FR, with 100×100 than both 25×25 and 50×50 even though denser sampling would be expected to make the estimated gradients more accurate. It should be noted, however, that the constant 2 used in the denominator of estimated gradients (Eq. 5.88) implies that the neighboring sampled points are separated by a 1 pixel gap. This can make the estimation less accurate when the patch size is less than the sampling resolution which is true with 100×100 for many objects in the datasets.

- As with iterations, most AMs show significantly greater improvement on increasing the sampling resolution with ICLK than with FCLK. This probably indicates a greater dependence of ICLK on the smoothness of f to find convergence.

A.1.4 Pyramidal Tracking

This refers to the use of a Gaussian image pyramid to carry out tracking in a coarse to fine manner [62] by first estimating a rough location in a downsized and blurred image and then refining it in successively larger and sharper images. Fig. A.5 shows the impact of pyramidal tracking on FCLK though other SMs gave similar results so this can be taken as indicative of its effect in general. These results were generated using 3 layers of Gaussian pyramid with a scaling factor of 0.5 between consecutive layers. A sampling resolution of 100×100 was used in the first layer though scaling was also applied to this so that the second and third layers used 50×50 and 25×25 respectively. Experiments were also conducted using 50×50 in the first layer but worse results were obtained with most AMs probably because the resultant 12×12 resolution in the last layer is too low.

It can be seen that pyramidal tracking does not improve SR with any AM except RSCV and actually causes it to decrease with several of them. However, there does

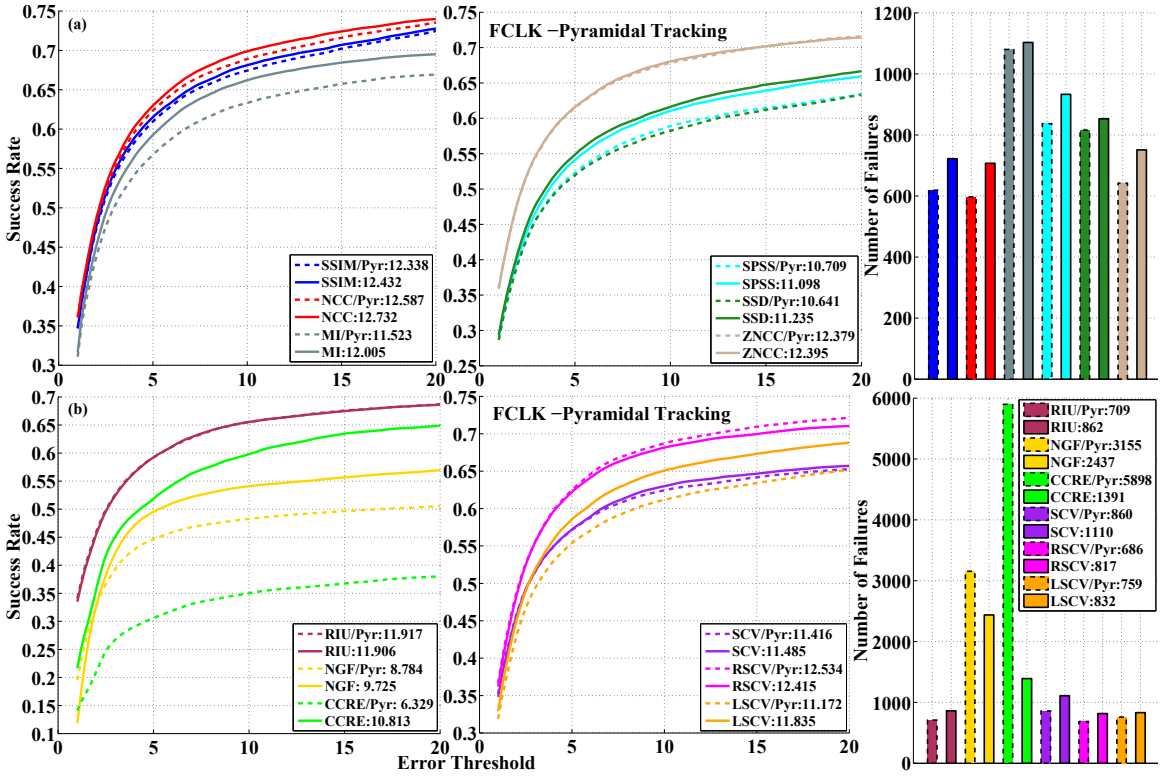


Figure A.5: Effect of pyramidal tracking on AMs with FCLK

seem to be a slight improvement in FR with all but CCRE and NGF. This is rather surprising as this technique has been widely considered as a good strategy to make tracking perform better [19, 62, 15, 93].

A.2 Stochastic and Composite Search

A.2.1 NN

Following parameters have been tested for NN:

A.2.1.1 Index type

As described in Sec. 4.2.1.1, NN search can be performed using several index types including those in FLANN - HKMT and KDT - and those implemented within MTF - GNN and FGNN. KDT is only compatible with AMs whose distance measure f_D can be computed by accumulating the results of computations performed using individual elements of the distance feature vector $\mathbf{D}(\mathbf{I})$ (Sec. 7.2.3). This is true for pixel wise

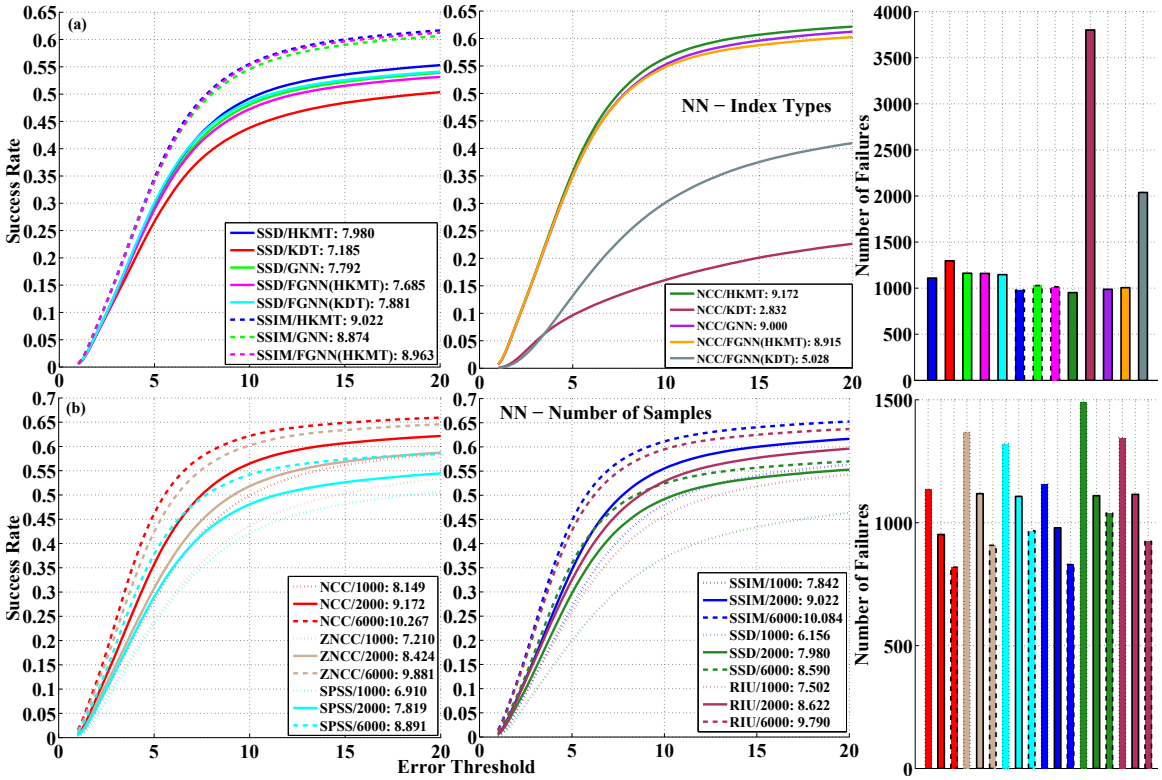


Figure A.6: Effect of (a) index types and (b) number of samples on NN.

measures including all L2 models and SPSS using $\mathbf{D}(\mathbf{I}) = \mathbf{I}$. NCC too can be made compatible by using $\mathbf{D}(\mathbf{I}) = \dfrac{\mathbf{I} - \mu_I}{\sigma_I}$ where μ_I and σ_I are respectively the mean and standard deviation of $\mathbf{I}$. Then, $f_{ncc}(\mathbf{I}^*, \mathbf{I}^\mathbf{c}) = -\mathbf{D}(\mathbf{I}^*) \cdot \mathbf{D}(\mathbf{I}^\mathbf{c})$ which can be computed by summing up the negative products of corresponding elements in the feature vectors. For AMs that support KDT, FGNN too can be formulated using both KDT and HKMT.

Fig. A.6 (a) presents the results comparing the index types with SSD, SSIM and NCC. SSD can be taken as representative of L2 models and NCC/SSIM for the robust ones. Since initialization time and runtime speed are also important criteria for choosing the index type, these results are also reported in Fig. A.7. Note that the total initialization time for NN also includes the time needed to build the dataset of samples which is excluded here as being independent of the index type. Following observations can be made:

- HKMT is the best performer with all the AMs in terms of both SR and FR though GNN and FGNN are only slightly worse.

- GNN and FGNN perform almost identically which indicates that using the exact nearest neighbors while building the graph is not crucial to its performance and approximate ones work just as well.
- With SSD, KDT is only marginally worse than the other index types which might be an acceptable trade off for increase in speed since it is over four times faster than GNN which in turn is four times faster than HKMT.
- With NCC, however, KDT performs very poorly and does not work at all. Though not shown here, this behavior also holds with SPSS so it seems that KDT only works well with L2 models.
- GNN is significantly slower to initialize than both HKMT and KDT which is to be expected as the graph is very computationally expensive to build. FGNN is not much faster either except when used with KDT which, as seen above, only works well with L2 models.
- FGNN is faster than GNN at runtime with both NCC and SSIM - only marginally with the former but over 3 times faster with the latter. This is unexpected as FLANN search is only used while building the graph and might indicate that GNN can actually converge faster when the graph is built using approximate nearest neighbors. It should be noted, however, that FGNN is slower than GNN with SSD so this advantage seems limited to robust AMs.
- GNN is over four times faster than HKMT with SSD but nearly as much slower with NCC and SSIM. Coupled with the previous point, this suggests that graph search finds convergence much faster with SSD than with the other two AMs.
- HKMT takes over 10 times longer to initialize with SSD than SSIM and NCC though the absolute times are probably small enough to not matter in practice. However, when combined with its slower runtime speed, especially as compared to GNN, this might indicate that HKMT is not as well suited to SSD as SSIM and NCC.
- To summarize, SSD (and other L2 models) should be used with KDT when speed or initialization time is crucial and GNN/FGNN when accuracy is more important. SSIM and NCC (and other robust AMs), on the other hand, work best with HKMT.

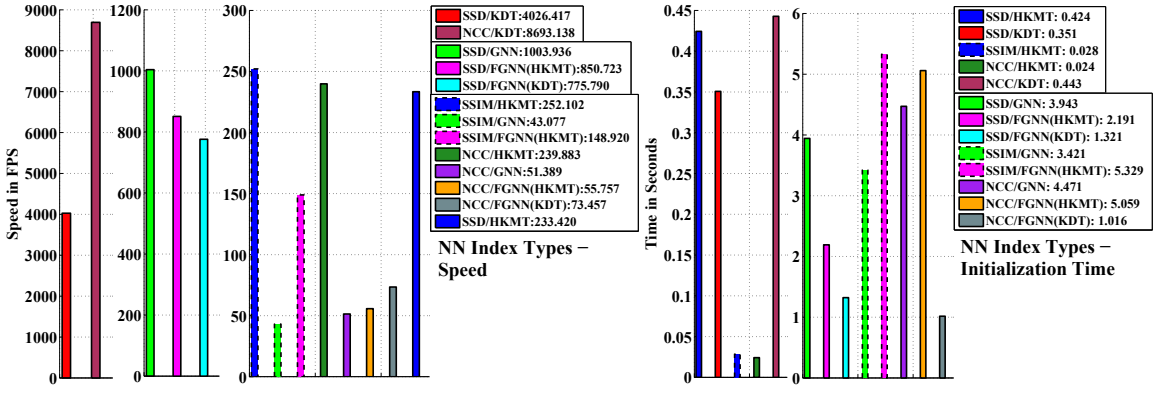


Figure A.7: Speeds and initialization times of NN index types. Only mean values are shown.

A.2.1.2 Number of Samples

The performance of stochastic SMs depends largely on the number and quality of samples used. NN in particular is very sensitive to these two variables since its samples are generated only once. The quality thereof depends on how well the distribution matches the actual motion and is hard to improve without prior knowledge of this motion. The sample count, however, is only limited by the available computational power and is the simplest way to improve the performance of these SMs. It is therefore obviously a good idea to use as many samples as possible and not much information can be obtained by examining the effect of this variable. However, its results are included here for completeness since it plays such an important role in determining the performance of this SM.

Fig. A.6 (b) shows the results with 1000, 2000 and 6000 samples. As expected, all AMs benefit from having more samples in terms of both SR and FR. The degree of improvement also seems fairly similar for all of them except perhaps SSD which does not gain much by the increase from 2000 to 6000 samples. This might be due to the limited discriminative ability of SSD itself forming a bottleneck.

A.2.1.3 Distribution Type

Fig. A.8 presents the results of using the mixture distribution with NN with equal number of samples dawn from each constituent distribution. It can be seen that the mixture method works quite well even without the stochastic distribution selection process of PF. The gain in performance is much less marked here though and mostly limited to a gain in precision.

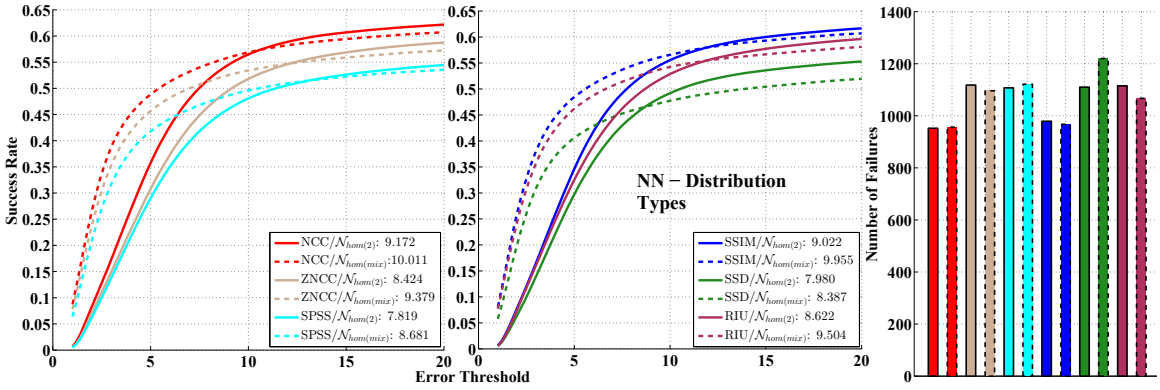


Figure A.8: Performance of NN with single and mixture distributions.

A.2.1.4 Number of Layers

As described in Sec. 4.2.1.1, one way to generate better quality samples is to run multiple layers of NN in cascade, each with a different distribution, performing a coarse to fine search. Two such arrangements with 3 and 5 layers were tested, referred to as NN3 and NN5 respectively. NNIC was also tested with the former. Note that the total number of samples used for all of these configurations were same as for NN and were distributed evenly between the layers. Fig. A.9 presents the results of these tests. Results for NN with 6000 samples are also included to demonstrate the comparative importance of the number and quality of samples. Following observations can be made:

- NN3 performs much better than NN with all the AMs. In fact, it performs even better than single layer NN with 6000 samples even though the latter is 3 times slower in practice. This proves that the quality of samples is more important than their number. This makes sense since all samples from a given distribution tend to cluster together in a small subregion of the search space so that having more samples merely allows this subregion to be searched better. Samples from multiple distributions, however, lie in different subregions so that even if each one is less densely sampled, the overall search space is better represented.

- NN5 does not improve significantly over NN3 with any of the AMs and is even slightly worse with SPSS and RIU. It may be recalled from Sec. 8.3 that the overall range of perturbations produced by the distributions in the first and last layers is similar for NN3 and NN5. The main advantage offered by the latter is thus in providing a smaller step size in the change of distribution from

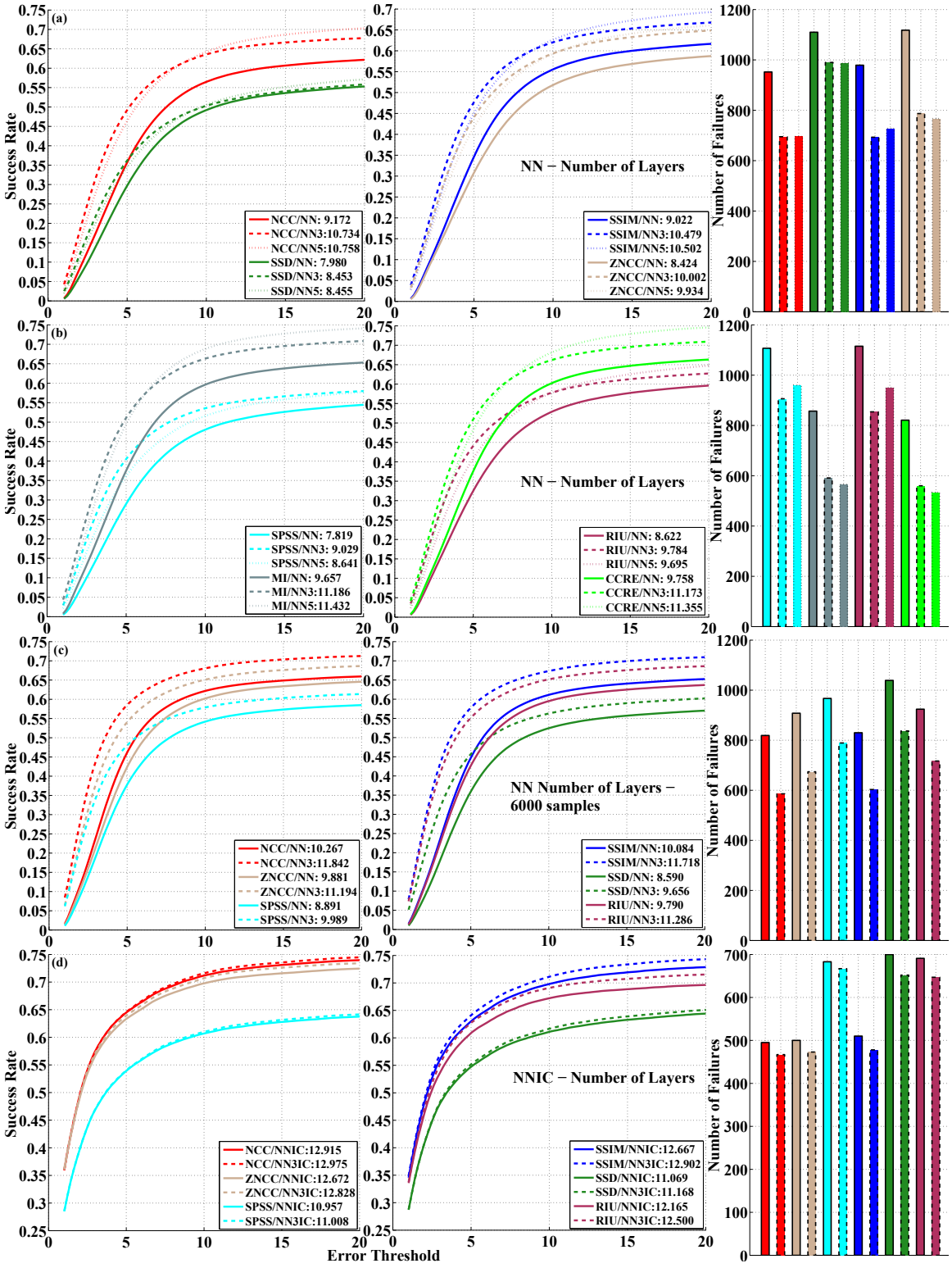


Figure A.9: Effect of number of layers on NN and NNIC. NN3 and NN5 respectively refer to 3 and 5 layers while NN3IC refers to a 4 layer cascade tracker with NN3 + ICLK.

each layer to the next which should allow the successive refinement process to be smoother and less likely to end up in false optima. It seems, however, that the decrease in the number of samples per layer offsets this advantage and the overall solution found ends up being of similar quality.

- The gain in performance with NN3 is just as strongly marked with 6000 samples (Fig. A.9 (c)) as with 2000 which indicates that the advantage of covering each subregion more densely does not diminish when more sub regions are being covered. This is also suggested by the fact that the improvement in NN3 itself by going from 2000 to 6000 samples is as large as with NN.

- NNIC hardly benefits from having 3 layers of NN (Fig. A.9 (d)), thus indicating that the solutions found by NN3 are not significantly better from the perspective of allowing ICLK to find convergence where it fails to do so with NN.

A.2.2 PF

Following variables have been tested for PF and PFFC:

A.2.2.1 Sampler Type

Two different 8 DOF samplers have been tested in this work (Sec. 8.3) - one for homography that uses normalized corner based sampling (Sec. 6.5.1) and the other for SL3 that directly generates samples for $\mathbf{p_s}$. The former is inspired from [98] while the latter is taken unchanged from [1]. The mixture distribution approach has been used for both samplers and they are thus denoted as $\mathcal{N}_{hom(mix)}$ and $\mathcal{N}_{sl3(mix)}$. Fig. A.11 presents results comparing these. Following points can be noted:

- For all AMs except CCRE and SCV, $\mathcal{N}_{hom(mix)}$ has higher SR for smaller t_p while $\mathcal{N}_{sl3(mix)}$ is better with larger t_p. The latter also has lower FR so this seems to be the more robust of the two though also less precise. The distributions used for this sampler were fine tuned for tracking sequences in the LinTrack dataset as well as several in PAMI and, as a result, this sampler performs much better on these sequences. As these are also some of the hardest sequences to track and often lead to tracking failures, the higher robustness of this sampler is to be expected. None of the distributions were designed for tracking fine motion, however, which explains its lower precision.

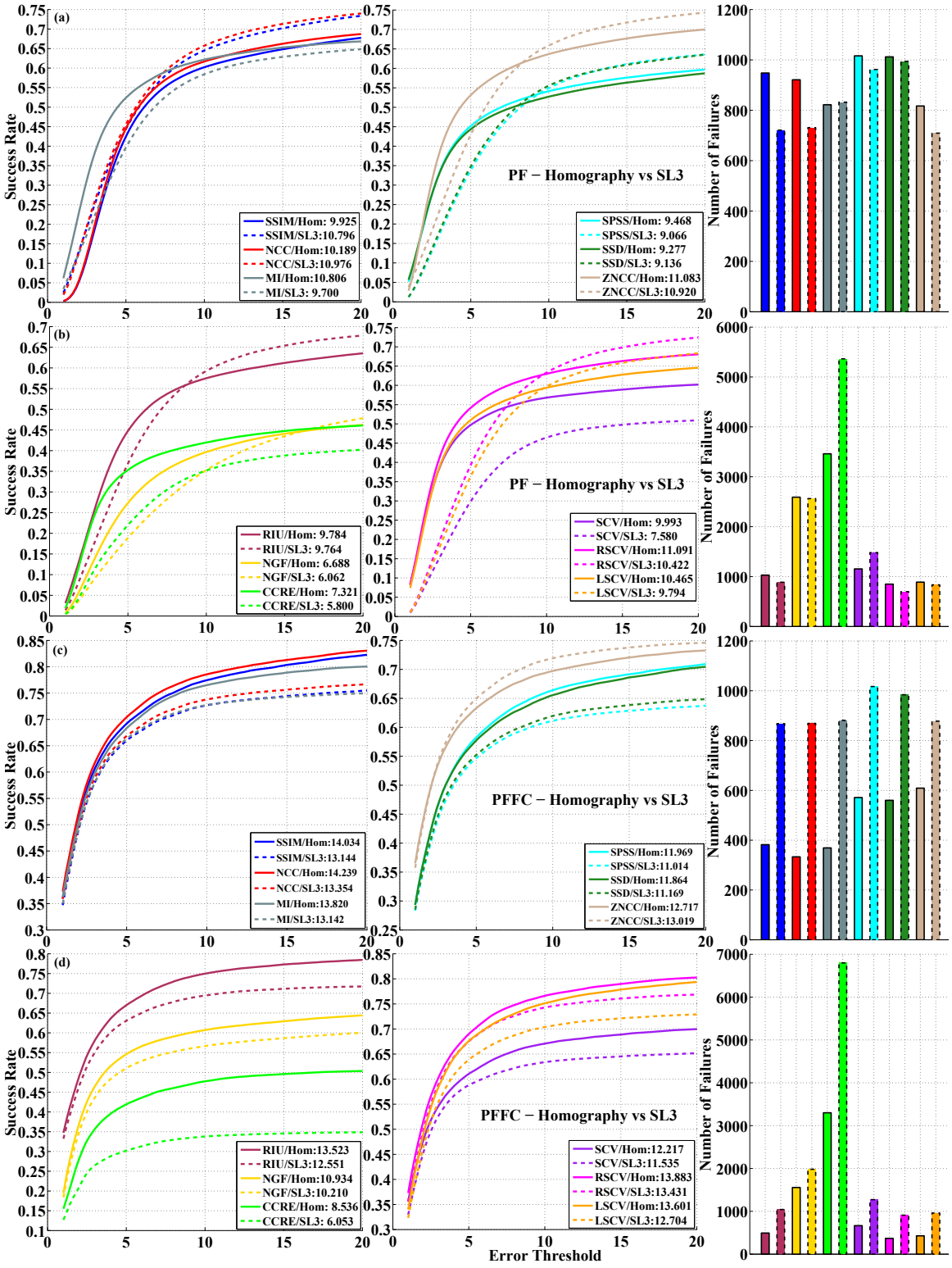


Figure A.10: Comparing Homography and SL3 samplers with PF and PFFC

- For CCRE and SCV, $\mathcal{N}_{hom(mix)}$ has much higher SR for all values of t_p as well as significantly lower FR.
- $\mathcal{N}_{hom(mix)}$ was chosen as the default because of its greater consistency as well for having the higher average SR and fairly similar FR with most AMs.
- Notwithstanding their close performance with PF, $\mathcal{N}_{hom(mix)}$ performs *much* better the $\mathcal{N}_{sl3(mix)}$ with PFFC in terms of both SR and FR. It seems that FCLK values precision more than robustness in the starting points that help it converge better. This was another reason for choosing $\mathcal{N}_{hom(mix)}$ as the default.

A.2.2.2 Number of Particles

Fig. A.11 (a) presents results for 250, 500 and 1000 particles. As expected, performance improves with more particles though the gain seems somewhat less marked here than with NN. This is partly because of the mixture distribution and partly because new particles are generated in each frame, both of which can allow more of the search space to be covered with fewer particles.

A.2.2.3 Distribution Type

This section compares the performance of individual distributions in tables 8.2 and 8.3 with the respective mixture distributions. Fig. A.11 (b) and (c) present the respective results. It can be seen that the mixture distributions for both samplers perform roughly as good as the best of their constituents for each value of t_p so that their SR curves almost form an enveloping upper bound for those of the individual distributions. This proves that the stochastic mixture technique proposed in Sec. 4.2.1.2 is indeed effective in choosing the best distribution for each scenario.

A.2.2.4 Optimum Estimation Method

This refers to the method used for combining the resampled particles to estimate the optimum state. The iterative mean estimation method for SL3 sampler [1, algorithm 1] is the only one known to the author so this was compared with the simpler alternative of choosing the state of the particle with the maximum weight as the optimum. Fig. A.11 (d) presents the results. It can be seen that choosing the maximum weighted particle provides similar performance as the mean estimation method and

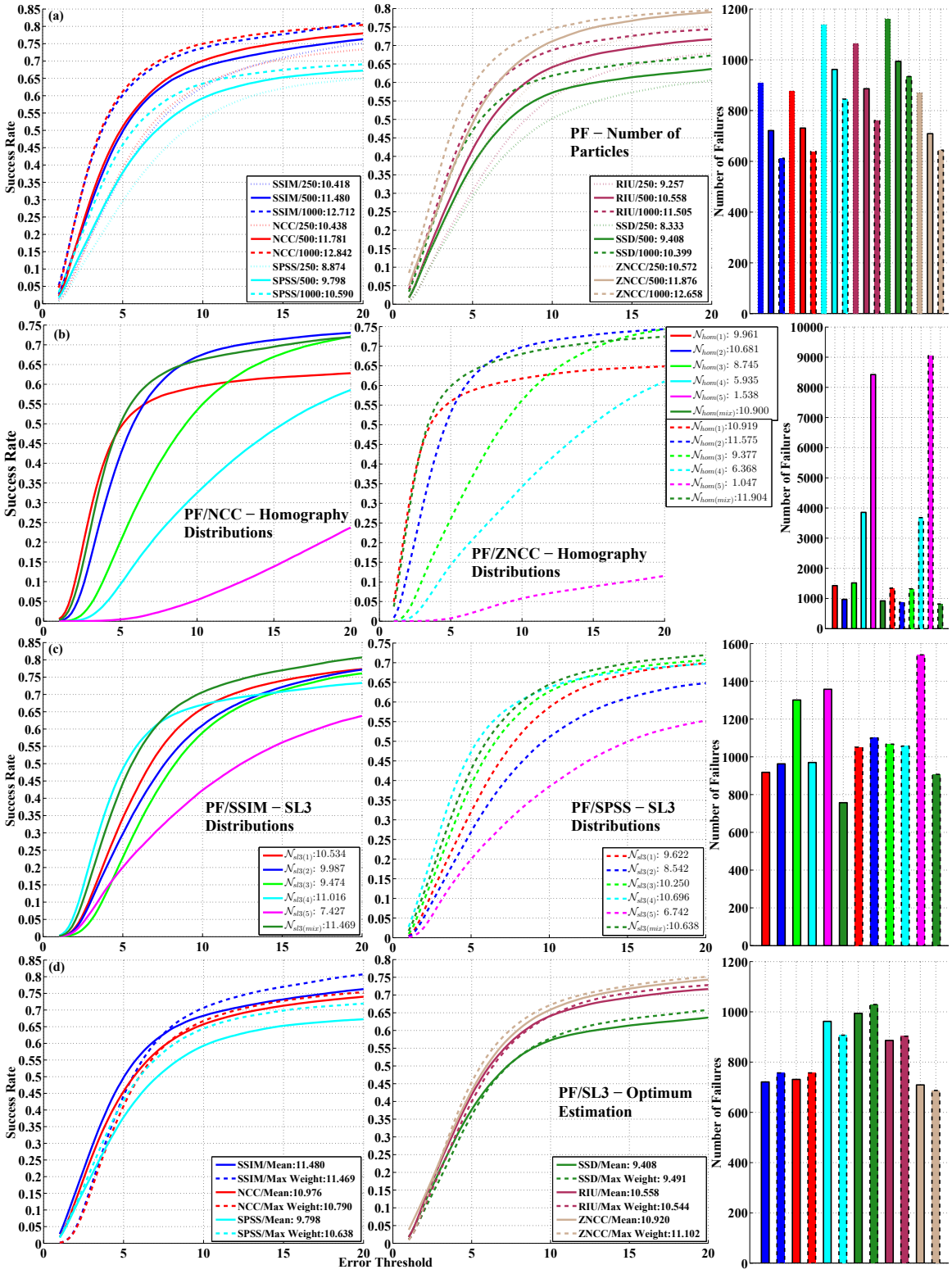


Figure A.11: Effect of different variables on the performance of PF: (a) number of particles (b) homography distributions (Table 8.2) (c) SL3 distributions (Table 8.3) (d) optimum estimation method. All results were generated without subsequences.

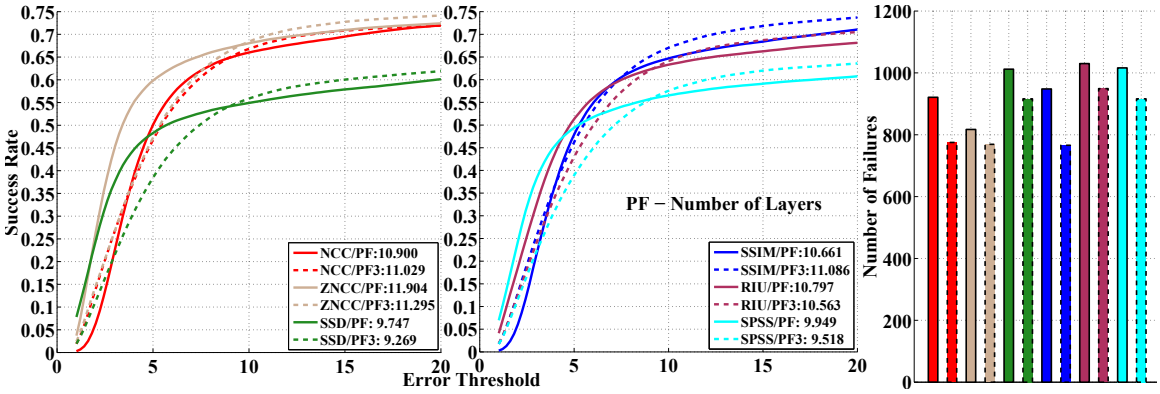


Figure A.12: Effect of number of layers on PF. PF3 refers to a 3 layer cascade.

is even better with most AMs for larger t_p. As this is also faster than the latter, this seems like the better alternative.

A.2.2.5 Number of Layers

Fig. A.12 presents the results of 3 layer PF with $\mathcal{N}_{hom(4)}$, $\mathcal{N}_{hom(2)}$ and $\mathcal{N}_{hom(1)}$ in individual layers compared against single layer PF with $\mathcal{N}_{hom(mix)}$. The former does seem to be more robust, having lower FR as well as higher SR for larger t_p though the difference is much smaller than with NN. Also, its average SR is lower with four of the AMs and very similar with the remaining two. This, coupled with its significantly lower precision with 3 of the AMs, rules it out as the clear winner. It can be concluded that the mixture approach is overall just as effective as the cascade one, with the former having a slight edge in precision and the latter in robustness.

A.2.2.6 Automatic Reinitialization

This section evaluates the performance of the automatic failure detection and reinitialization technique described in Sec. 4.3 as one of the ways to take advantage of running multiple trackers in cascade. Fig. A.13 presents the results for both PFFC and NNIC. Following observations can be made:

- The benefit of auto reinitialization is mainly confined to SR which is surprising since the main idea behind its design is to reduce the number of failures by automatically detecting them.
- The gain in SR is more strongly marked with NNIC than PFFC. Also, it is the poorer performing AMs like SSD and SPSS that benefit most with both SMs.

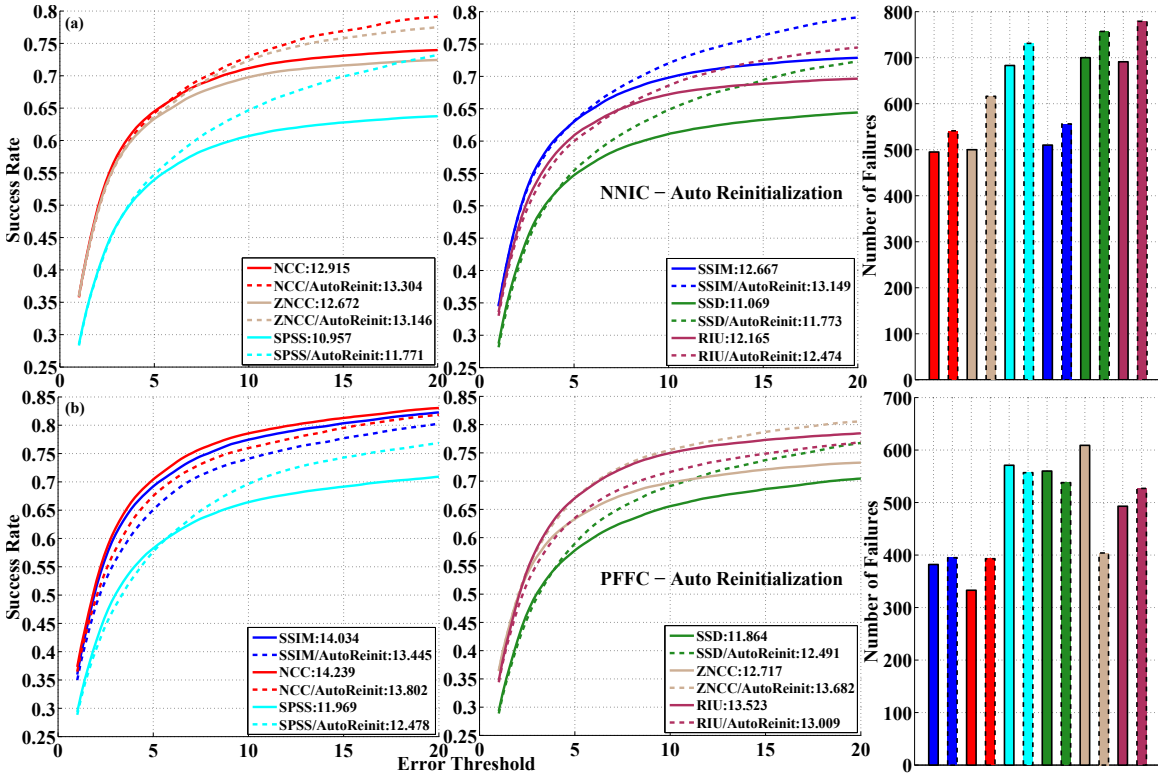


Figure A.13: Effect of auto reinitialization (Sec. 4.3) on NNIC and PFFC

In fact, NCC and SSIM actually show a decrease in SR with PFFC. This seems to indicate that auto reinitialization helps more with less robust trackers which also makes sense as these are the ones more likely to fail, thus providing the reinitialization module with more opportunities to detect and correct these.

A.2.3 LMS

Following parameters have been tested here:

A.2.3.1 Grid Resolution

This determines the number of point correspondences used by the robust estimation step and thus also the number of trackers needed for generating these. It would intuitively seem that having more point correspondences would help to find a better solution but, as shown in Fig. A.14 (a), increasing the resolution beyond 10×10 does not produce any improvements in performance. Results are only shown for SSD and NCC but also hold for remaining AMs. Since the speed of this SM varies inversely

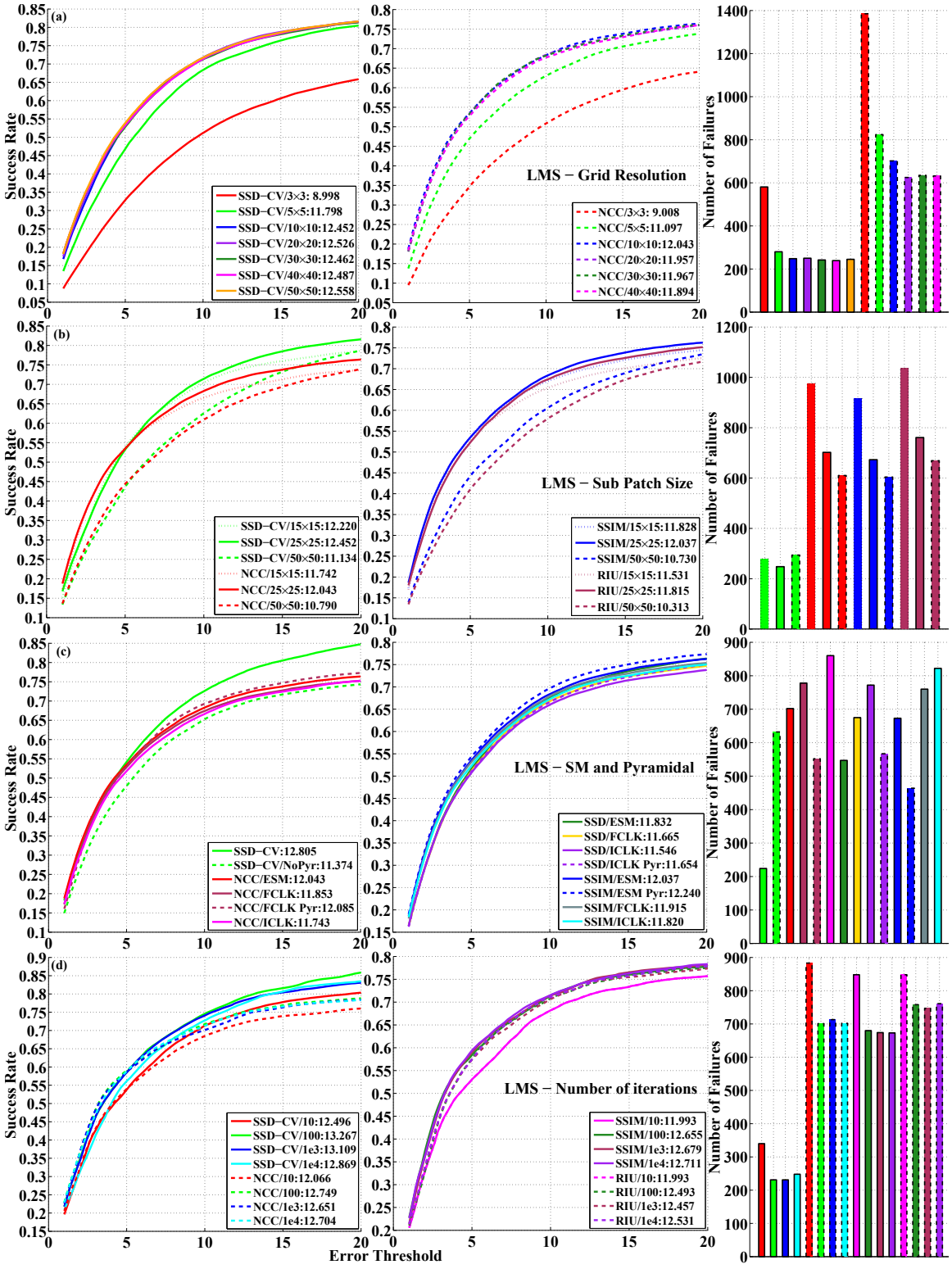


Figure A.14: Effect of (a) grid resolution, (b) sub patch size, (c) search method and (d) number of iterations on LMS

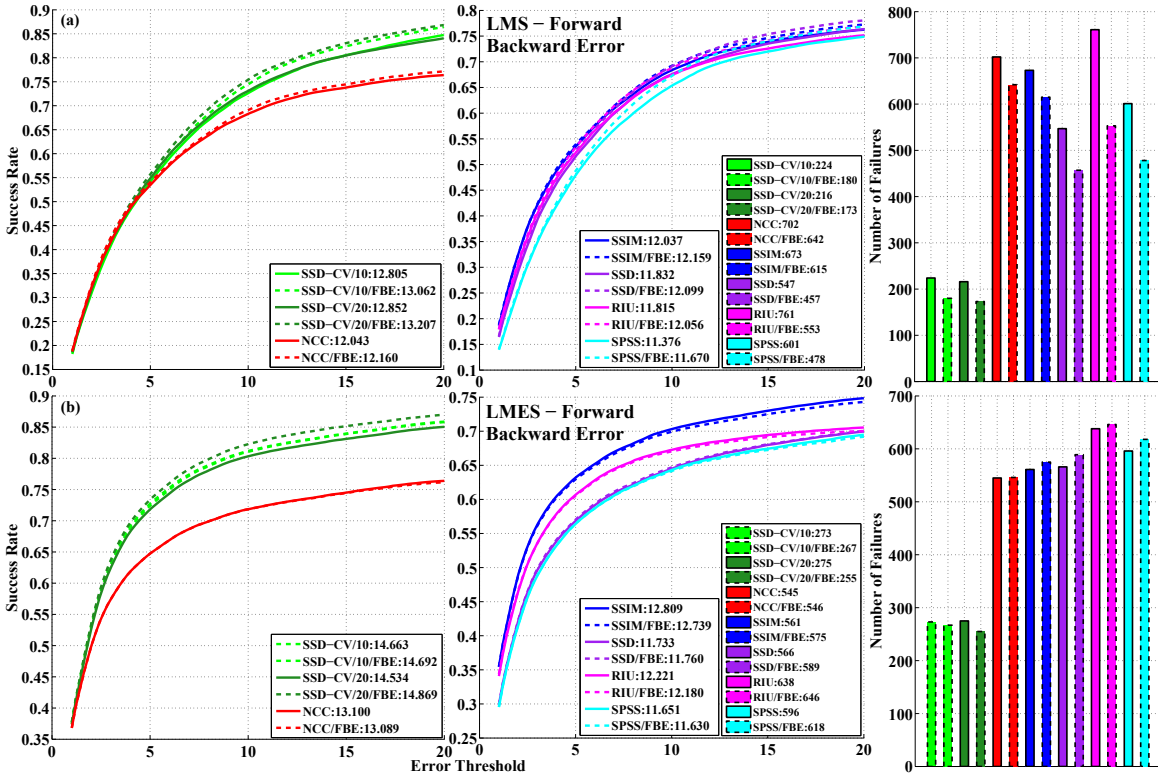


Figure A.15: Impact of failure detection with forward backward error on LMS and LMES. SSD-CV results are shown for both 10×10 and 20×20 grid resolutions.

as the square of the grid resolution, the lowest possible value is to be recommended and thus 10×10 seems like the best choice.

A.2.3.2 Sub patch Size

This is the size of the sub patch around each point in the grid from where pixels are used to track that point. In this work, this is also equal to the sampling resolution used by the corresponding tracker. It was suggested in [19] that this should be larger than the actual motion of the point between consecutive frames. While a 10×10 patch might be enough to satisfy this criterion, it has been seen in Sec. A.1.3 that GD type SMs do perform better with more pixels. It would therefore be expected that larger patch sizes should work just as well if not better. As shown in Fig. A.14 (b), this is indeed so with sizes up to 25×25 but increasing beyond that leads to a decrease in performance.

It should be kept in mind that, in smaller objects, portions of the sub patches lying near the boundaries of the object patch are more likely with larger sub patch

sizes to extend outside the object into the background and thus be more prone to failure. In general, larger patch sizes are beneficial as long as the object is large enough to accommodate them. This happens to be not true for a majority of objects in the tested datasets, which therefore perform worse with 50×50.

A.2.3.3 Search Method

This is the SM used by the sub patch trackers and also includes the use of pyramidal tracking. AMs in MTF were tested with the three compositional SMs, both with and without pyramidal tracking, while SSD-CV was only tested for the impact of the latter as it does not provide any way to change the SM. Results of these tests are given in Fig. A.14 (c). Following observations can be made:

- SSD-CV performs significantly worse without pyramids and is in fact even worse than SSD. This seems to confirm the hypothesis that the reason for the superiority of SSD-CV over MTF AMs has something to do with the way its pyramidal tracking is implemented.
- ESM is slightly better than FCLK which in turn outperforms ICLK, though the difference is much smaller in terms of SR than FR.
- Pyramidal tracking improves performance too but, as with SMs, the improvement is more marked with FR than SR. It is also not as significant as that for SSD-CV so that none of the MTF trackers manage to match SSD-CV even with pyramids.

A.2.3.4 Number of Iterations

This is the maximum number of iterations that the robust estimation step is allowed to perform. Since this decides the number of candidate warps that will be evaluated to find the one with maximum consistency, it is, in a sense, equivalent the number of samples/particles used by NN/PF. It would therefore be expected that allowing more iterations would help to find a better solution with a consequent improvement in performance. It turns out, however, that in practice the number of iterations actually used by both LMS and RANSAC rarely exceeds 10 and almost never goes over 100. The results in Fig. A.14 (d) confirm this as showing marginal improvement from 10 to 100 but none beyond that.

A.2.3.5 Forward Backward Error

This method was used in the median flow tracker [35] to automatically detect tracking failures by comparing the trajectories obtained by tracking forwards and backwards in time. In the present context, this can be used to detect points where the tracking (or optical flow estimation) has failed *before* these are passed to the robust estimation step. The points are first tracked from the previous to the current frame and then, starting from the resulting locations of this forward tracking, they are tracked back to the previous frame. If the tracker at a given point succeeded, the final location of the backwards tracking step would be very similar to the starting location of the forward step. As a result, all points where the L2 norm of the difference between the two locations exceeds a threshold can be declared as failures and excluded from the robust estimation step. This method can serve to complement the outlier detection provided by the latter.

Fig. A.15 shows the impact of forward backward error estimation with LMS and LMES. An error threshold of 1 was used for these results but thresholds of 0.5 and 2 were also tested and gave similar results. It can be seen that SSD-CV shows greater improvement than the MTF AMs which actually show a slight decrease in performance with LMES - evidently this method works better when the optical flow estimation itself is more accurate. In challenging situations, this method can lead to a large fraction of points getting excluded from the robust estimation step. It would therefore be expected that having more points to begin with might improve the probability that enough points are left for this step. Results do validate this to a certain extent as the improvements in SSD-CV are slightly more marked with 20×20 grid than 10×10 especially in terms of SR and more so for LMES than LMS. Though small, these gains are still significant as improving a tracker that is already the best one tested in this work.

A.2.3.6 Failure Detection and SPI

As described in Sec. 4.3, these are two further ways to take advantage of cascade tracking. Failure detection is similar to the one in automatic reinitialization (Sec. A.2.2.6) except that, instead of reinitializing the tracker, the output of the GD layer is discarded and that of the LMS layer is used instead under the assumption that it is more robust. SPI uses the outliers detected by the robust estimator to turn off pixels in the GD layer, assuming that outliers correspond to difficult to track patches which

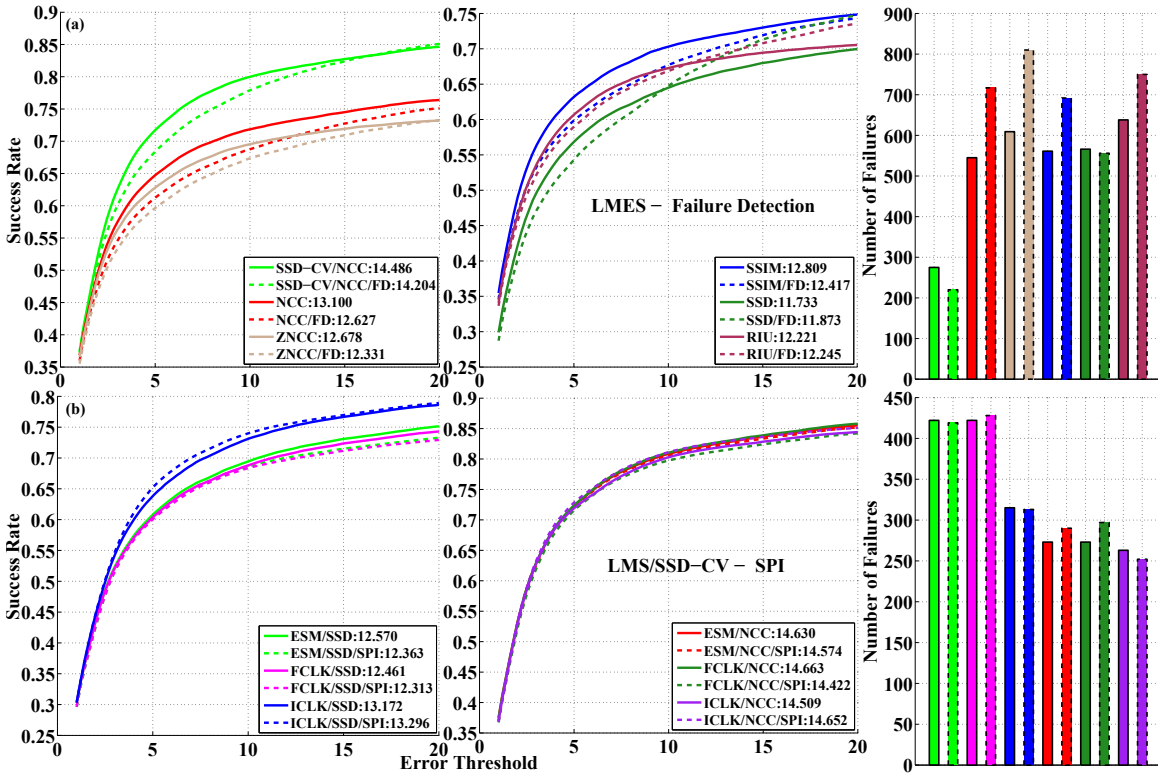


Figure A.16: Effect of (a) failure detection using LMES and (b) SPI using LMS with SSD-CV in the first layer and various SMs/AMs in the second layer.

would probably cause the GD tracker to fail too. This can only work when the LMS grid resolution is equal to the sampling resolution of the GD tracker. Since SSD-CV is the only AM fast enough to run LMS with a 50×50 grid, SPI has only been tested with this. All three compositional GD SMs have been tested in the second layer though only with SSD and NCC as these are the only AMs in MTF that currently support SPI.

Results for both techniques are in Fig. A.16. It can be seen that failure detection does not improve performance either in terms of SR or FR. On the contrary, it is notably worse with most AMs except perhaps SSD and RIU where it has higher SR for larger t_p. SPI too has no significant impact on the performance of any of the combinations of AMs and SMs.

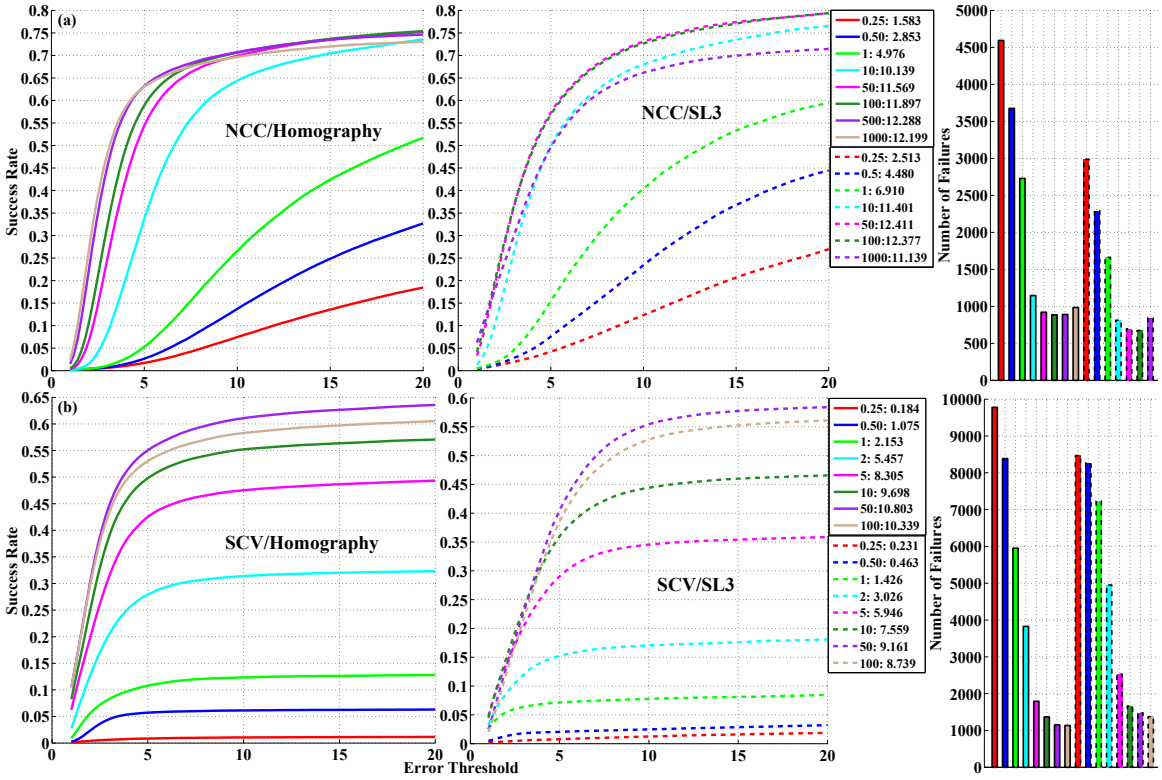


Figure A.17: Effect of likelihood α on the performance of NCC and SCV with PF using both homography and SL3 samplers. All results have been generated without subsequences.

A.3 Appearance Models

This sections presents results for experiments conducted to determine the optimum likelihood constants for all AMs as well as the effect of number of histogram bins on SCV, MI and CCRE. It also shows the benefit of using RGB images.

A.3.1 Likelihood Constants

Fig. A.17 and A.18 present likelihood results for most of the AMs. The α value for ZNCC was taken from [99] while those for RSCV and LSCV were set to be same as SCV since they have a similar range of f. Tests were conducted with both homography and SL3 samplers and, as seen with NCC (Fig. A.18 (a)), SCV (Fig. A.18 (b)) and NGF (Fig. A.17 (d)), they were found to perform similarly, so results for only one sampler are given for the remaining AMs. There is not much to be observed from these plots except the great impact that α can have on the performance of PF. This in

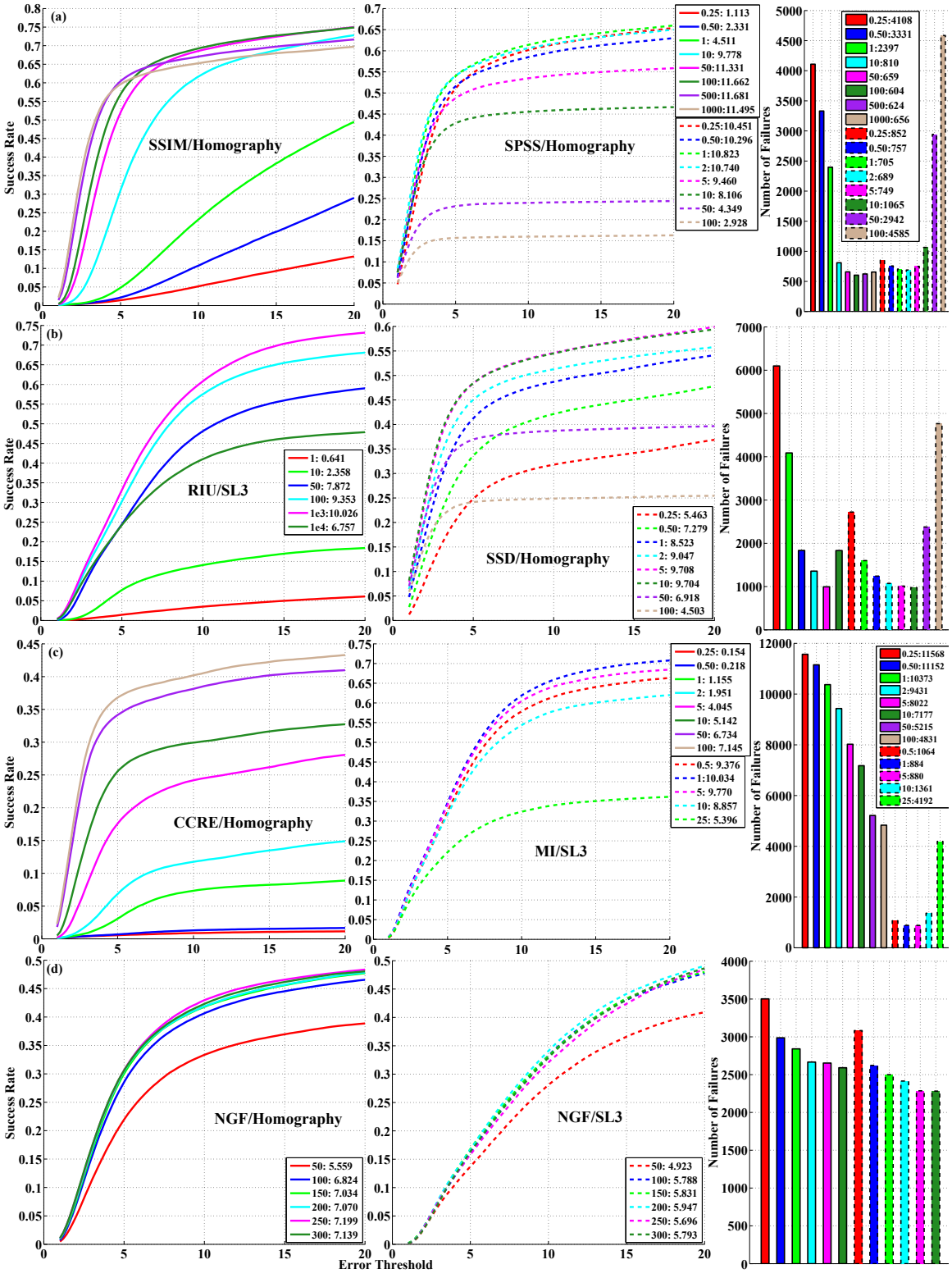


Figure A.18: Effect of likelihood α on the performance of AMs with PF. The additive factor β is 0 for all AMs except CCRE and NGF where it is 1. All results have been generated without subsequences.

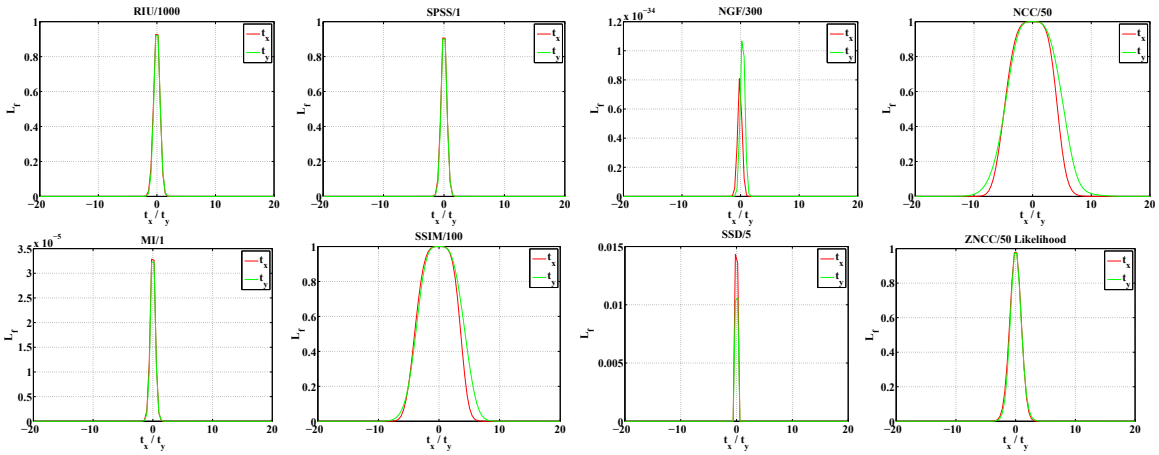


Figure A.19: L_f with optimal α plotted against x and y translations

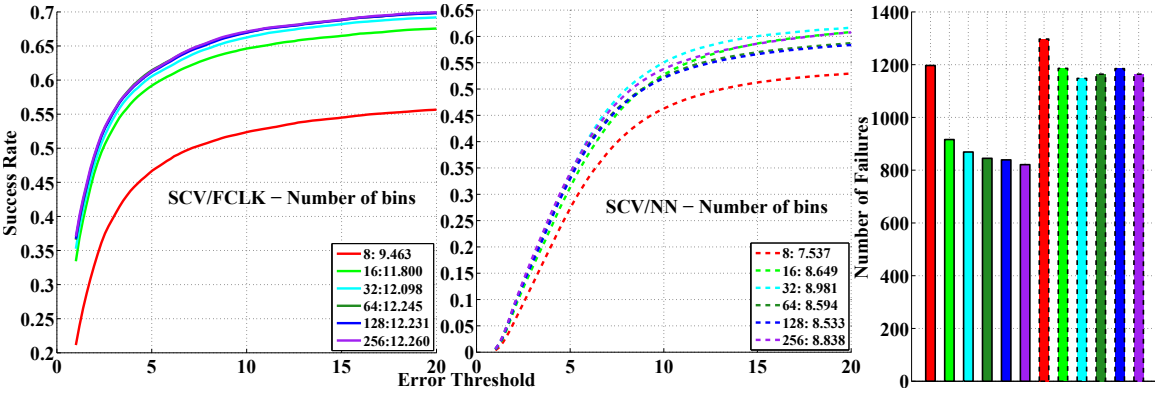


Figure A.20: Impact of the number of histogram bins on the performance of SCV with FCLK and NN. All results were generated without subsequences.

turn suggests that more exhaustive experiments might be able to provide better values than used here, especially for AMs like CCRE and NGF that performed so poorly with this SM. Attempts were made to understand the possible theoretical reasons behind these results by comparing the likelihood function surfaces for different α values. However, no definite patterns were apparent except that PF seems to prefer likelihood functions with relatively sharp optima as shown in Fig. A.19.

A.3.2 Number of Histogram Bins

Fig. A.20 and A.21 present the histogram bin results for SCV, MI and CCRE. SMs excluded here were found to perform similar to one of the included ones from the same category - for instance FCLK and PF gave similar results with CCRE as ESM

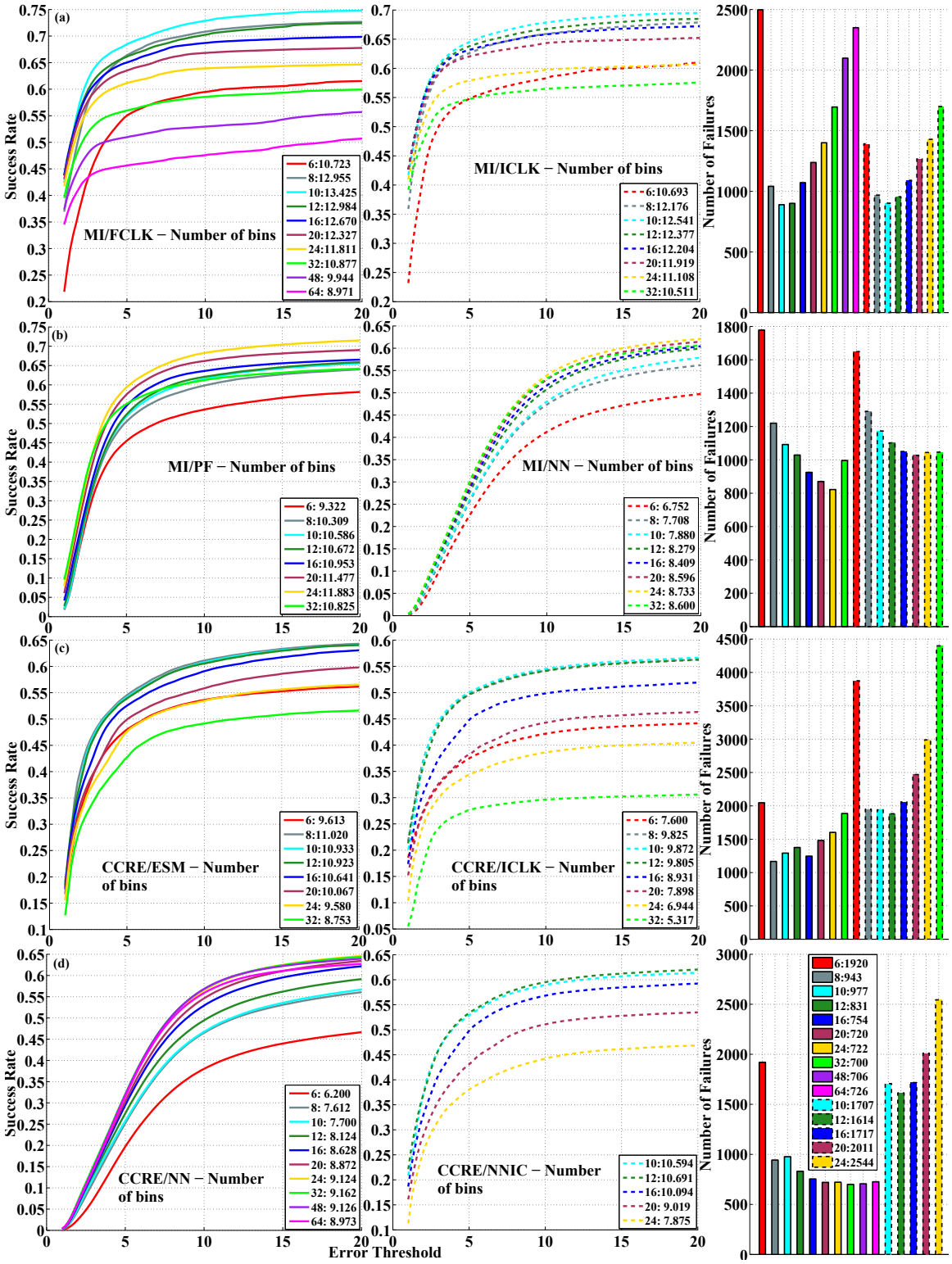


Figure A.21: Impact of the number of histogram bins on the performance of MI and CCRE with various SMs. All results were generated without subsequences.

and NN respectively. Both PF and NN results are given for MI to demonstrate this. Following observations can be made:

- SCV does not improve significantly on using more than 64 bins (Fig. A.20). Though FCLK does show a slight decrease in FR with 128 and 256 bins, it is definitely not worth the significant decrease in speed it leads to, particularly when 100 of these have to be run simultaneously in LMS.
- MI shows significant difference in behavior between the SMs - FCLK, ICLK and NN respectively work best using 10, 16 and 24 bins (Fig. A.21 (a)). This proves that the convention of using 8 bins that is usually adopted in literature [203, 190, 90, 92] is not justified in practice and only SM specific experiments can determine the optimal.
- CCRE too exhibits similar behavior as MI and performs best using 16, 12 and 32 bins respectively with ESM, ICLK and NN (Fig. A.21 (a)).
- Performance of NNIC with CCRE seems to be determined mostly by the ICLK layer as this too works best with 12 bins. Though not shown for MI, a similar behavior was seen there too.
- The large performance differences produced by varying the bins shows the great impact that this often neglected parameter can have on both MI and CCRE.

A.3.3 Multi Channel Imagery

This section evaluates the impact of using RGB images instead of gray scale ones. The support for multi channel images in MTF is currently limited to simple concatenation of pixel values from all channels into a single vector that represents the patch, i.e. $\mathbf{I_t}$ has 3 times the number of pixel values with RGB than with gray scale images (e.g. 7500 at a sampling resolution of 50×50).

Results are presented in figs. A.22 and A.23. LinTtack dataset was not included for generating these results since it only contains gray scale images (Sec. 8.1). FCLK and FALK gave similar results as ESM while IALK was similar to ICLK so these SMs have been excluded. PF and PFFC have also been omitted as the increase in the size of $\mathbf{I_t}$ also alters the range of values of f for most AMs. This in turn requires the likelihood constants to be readjusted for optimal performance and makes direct comparison with the gray scale variant difficult. Following are some observations:

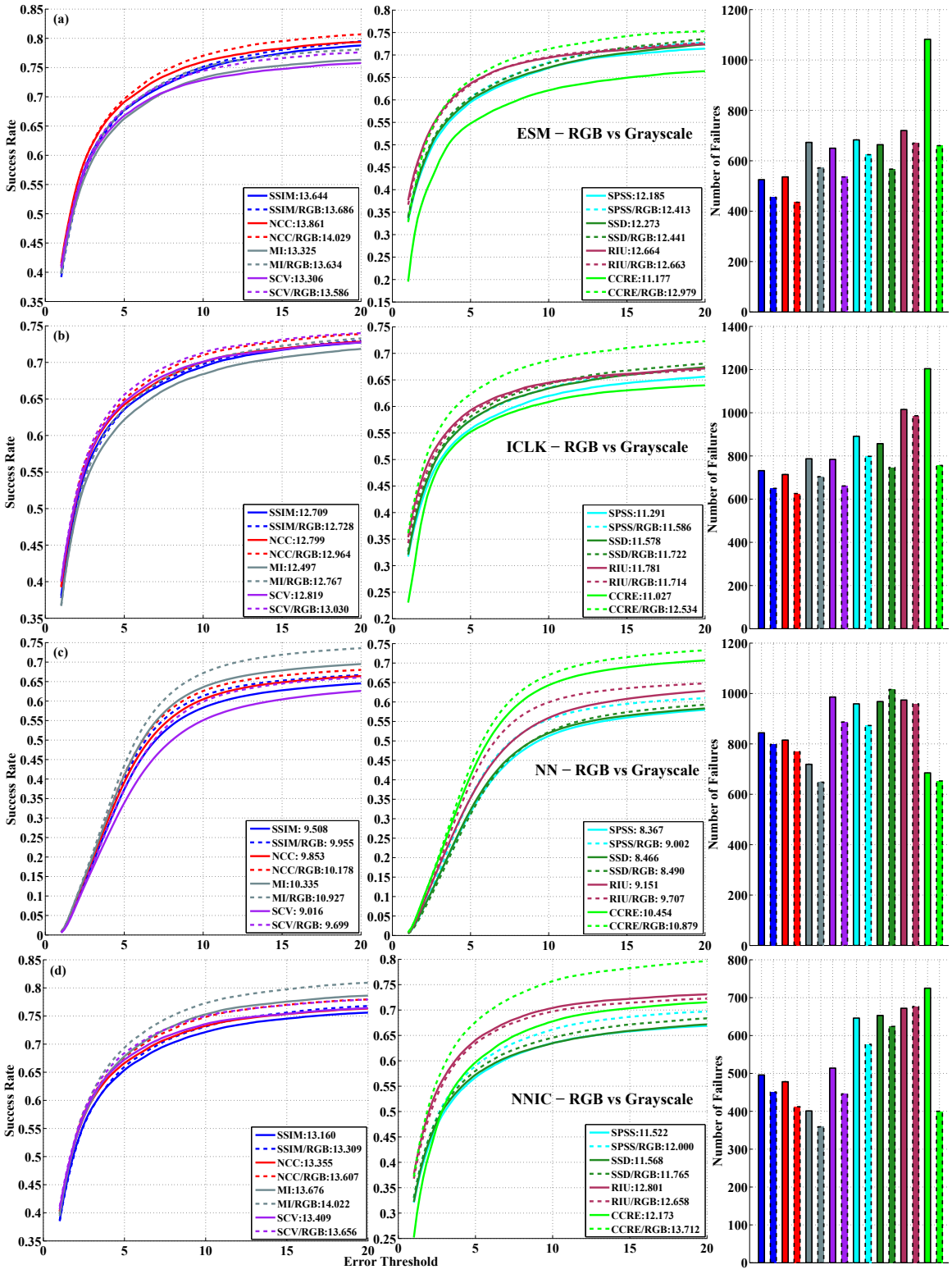


Figure A.22: Performance of AMs using RGB images with (a) ESM (b) ICLK (c) NN and (d) NNIC .

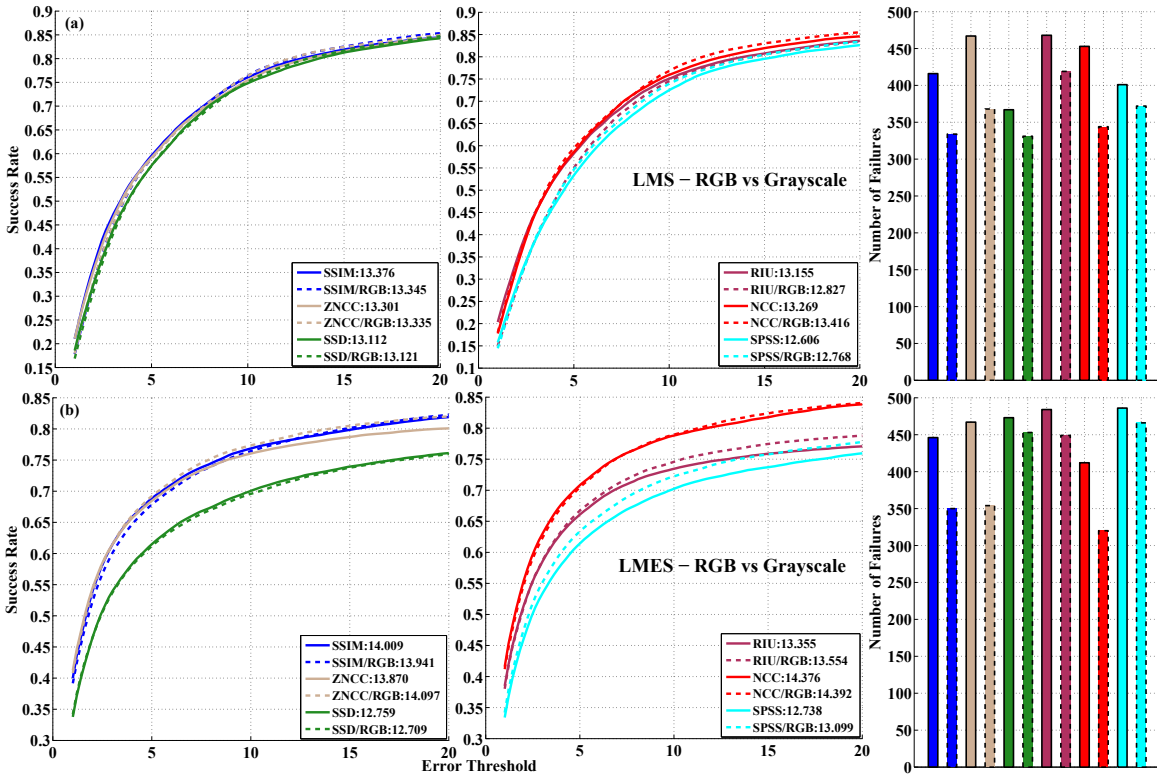


Figure A.23: Performance of AMs using RGB images with (a) LMS and (b) LMES.

- Most AMs do benefit from the use of RGB images though the improvement is most strongly marked in AMs that employ joint histograms - MI, CCRE and SCV. This is consistent with the results in Sec. A.1.3 where these same AMs benefited most from higher sampling resolutions.

- The improvement cannot be attributed entirely to the extra pixels, hoverer, since RGB images seem to produce a larger gain than 100×100 resolution even though the latter provides more pixels. This indicates that the extra discriminative ability afforded by color information does have practical benefits.

- The improvement in CCRE with GD based SMs is particularly noteworthy - it actually becomes ones of the better AMs and almost at par with MI.

- Decrease in FR is more strongly marked than increase in SR for most SMs except perhaps NN where the reverse seems to hold. NN is also the SM where the improvements is most consistent across the AMs.

Appendix B

Results on Individual Datasets

This appendix presents results for all SMs and ILMs over individual datasets to highlight the differences in performance ensuing from the varied challenges that these offer. Results for AMs have been excluded to save space as they did not provide any insights not present in the SM results.

B.1 Search Methods

Fig. B.1 - B.10 present results for all SMs except FALK that has been excluded as being very simlar to FCLK. PF results B.7 also include plots for the SL3 sampler to demonstrate the impact of using fine tuned distributions. Following are some noteworthy observations from these results:

- All trackers perform best on TMT which is the easiest dataset to track as offering no significant illumination changes and very few sequences featuring other challenges like motion blur and occlusion. This is followed by PAMI and UCSB which are increasingly harder while LinTrack proved most difficult to track by far. This latter is mainly because of the very long lengths of its sequences which cause a tracker to be penalized heavily if it loses track near the beginning at one of the several frames that feature motion blur due to jerky motion. This is indicated by the comparatively sudden plateauing of the SR curves of all GD based SMs at around $t_p \approx 4$ unlike the other datasets where this process is smoother. This happens when the trackers fail suddenly at a certain frame rather than drifting off gradually as the sequence gets more challenging.
- RIU is one of the best performers on UCSB with FCLK (Fig. B.1 (b)) and

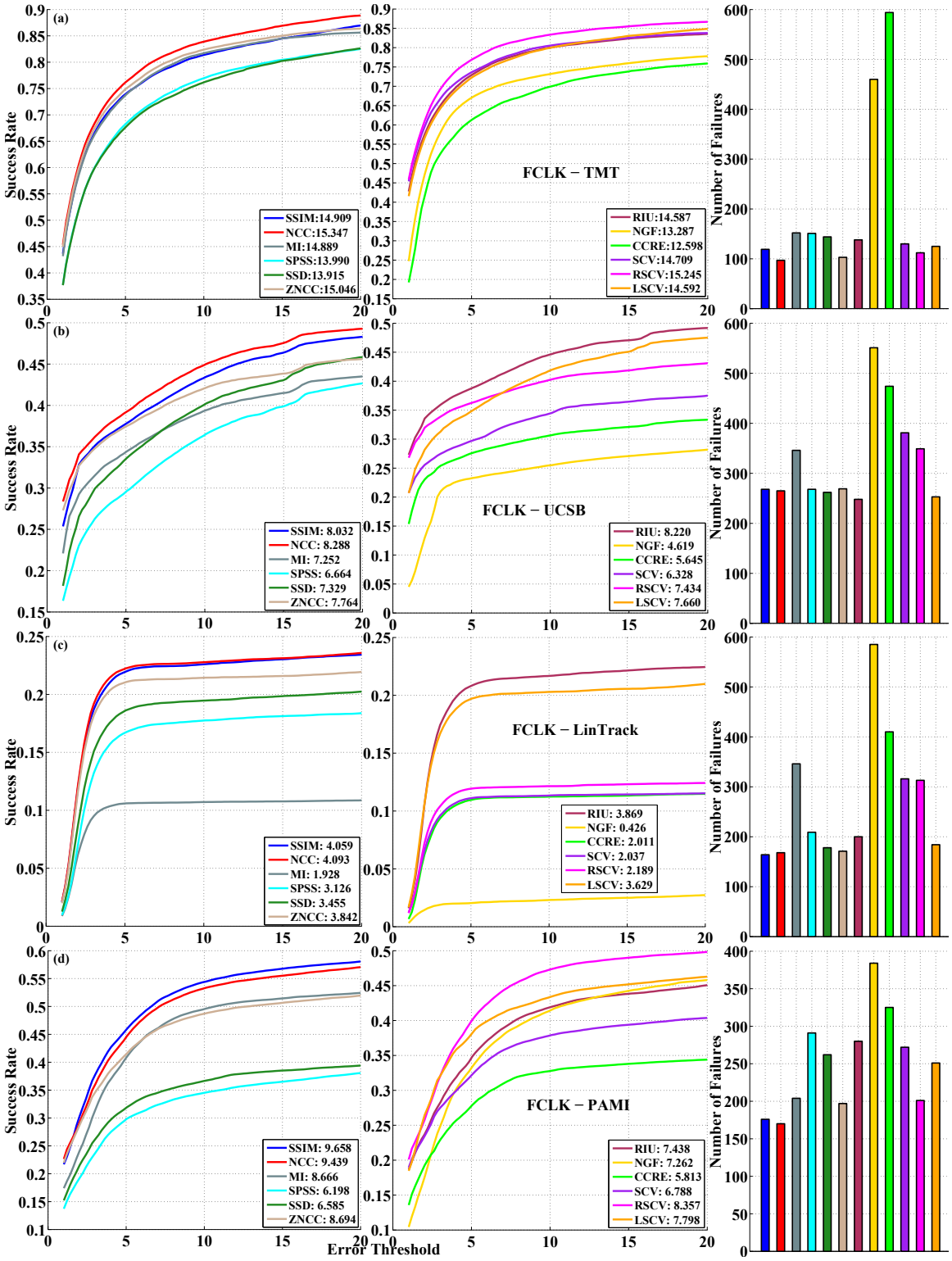


Figure B.1: Performance of different AMs with LM formulation of FCLK on individual datasets

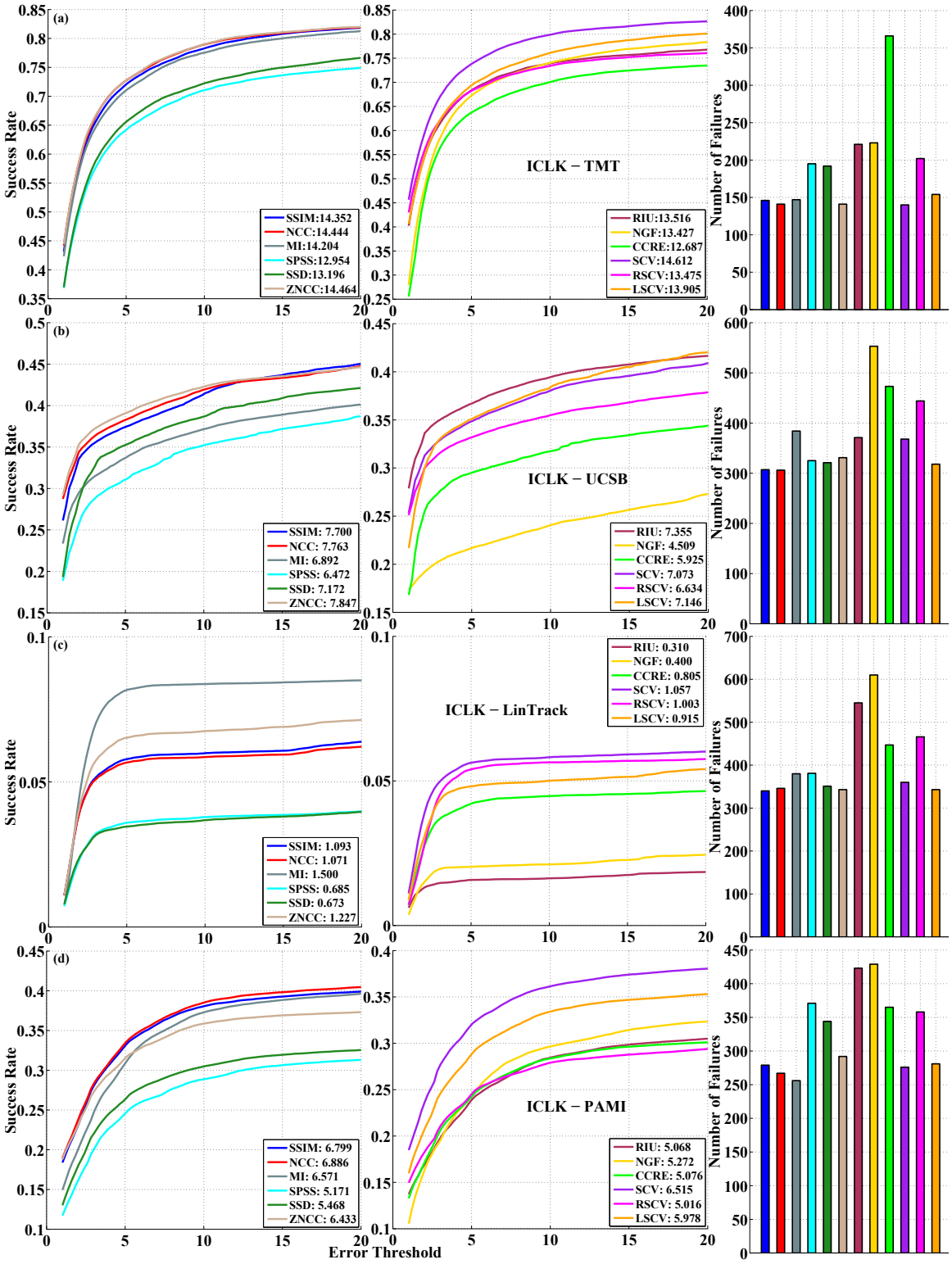


Figure B.2: Performance of different AMs with LM formulation of ICLK on individual datasets

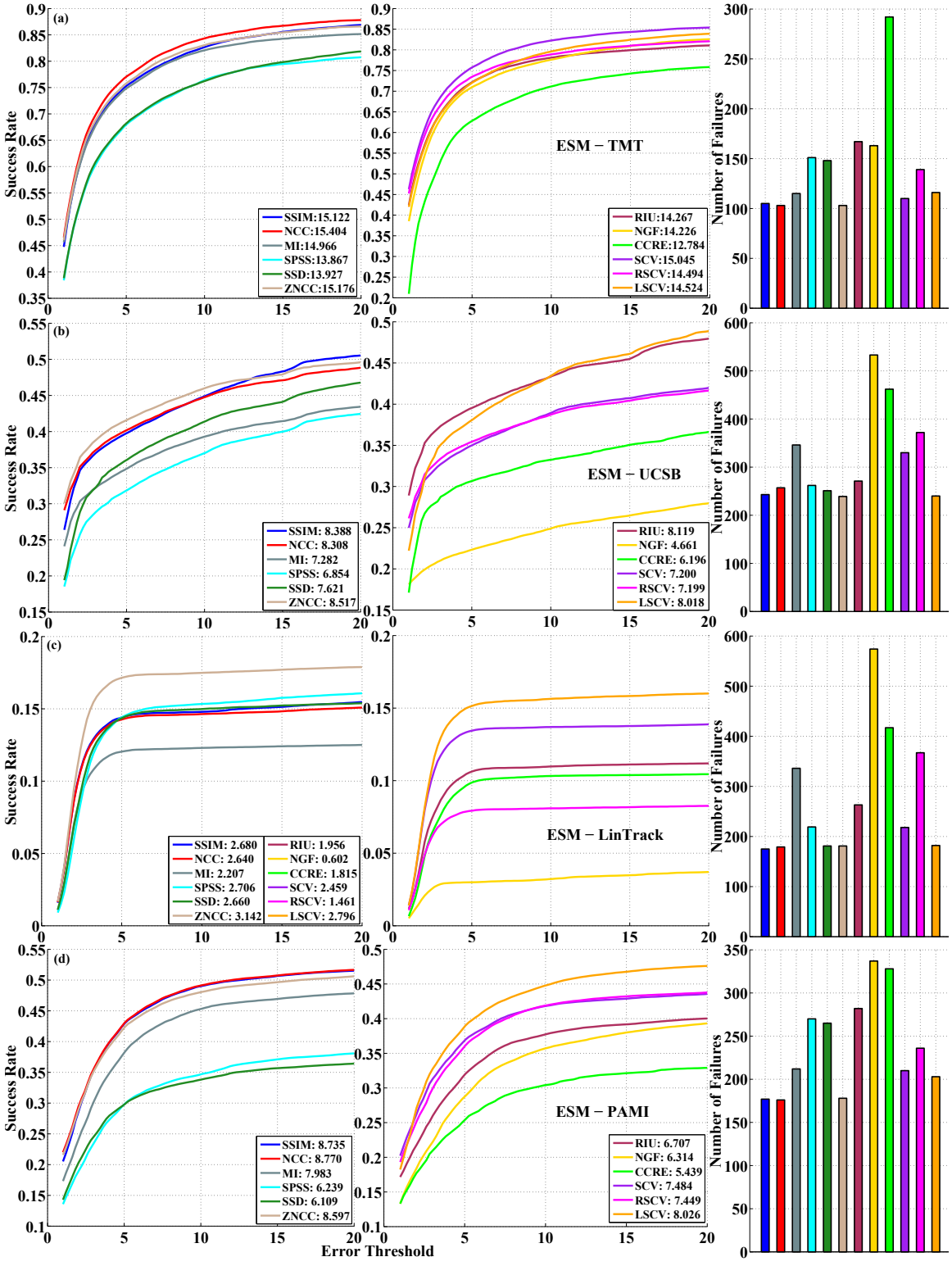


Figure B.3: Performance of LM formulation of ESM over individual datasets

IALK (Fig. B.4 (b)) and one of the better AMs with ICLK and ESM too. Its relatively wide convergence region and invariance to global illumination changes (Fig. 5.12) evidently help it with the main challenges in this dataset.

- CCRE and NGF perform much better on TMT with ESM and ICLK than FCLK though not on the remaining datasets. This may indicate that, like SCV, these two AMs favor inverse SMs too, though this is less evident with these AMs due to their significantly poorer performance in general.

- LSCV performs better on PAMI with ESM (Fig. B.3 (d)) than with any other SM. This might be partially due to the fact that PAMI contains several sequences with localized intensity changes of the sort that LSCV was designed to handle well. Also as several of these sequences were created for testing ESM with ILMs [84, 77, 85, 140], they may have been chosen by the respective authors to work well with this SM.

- NGF is one of the best performing AMs on PAMI with NNIC (Fig. B.6 (d)) in terms of SR though its FR is comparatively higher, as is usual with this AM.

- NN performs better on LinTrack (Fig. B.5 (c)) than all other GD and stochastic SMs. This is hard to explain as NN is generally the least robust SM against fast motion which is the primary challenge in this dataset. It is also worth noting that SSIM and RIU are the best performing AMs here, ahead of both MI and CCRE that are the best in all remaining datasets.

- NNIC with MI is one of the best performing trackers on LinTrack (Fig. B.6 (b)) along with LMES and PFFC. MI is also much better than all other AMs here. This serves as a good demonstration of the great and sometimes unexpected performance improvements that can be achieved by combining diverse trackers into a composite one. Each of the constituents of this tracker - NN, ICLK and MI - are susceptible to failure in the presence of fast motion, yet their combination creates a tracker that excels on sequences offering precisely this challenge.

- PF performs much better on LinTrack with $\mathcal{N}_{sl3(mix)}$ than $\mathcal{N}_{hom(mix)}$ (Fig. B.7 (c)). This is also true to a lesser extent for UCSB but not for TMT and PAMI. This shows the great impact that fine tuning the distribution for a specific sequence can have on the performance of this SM. The SL3 distributions

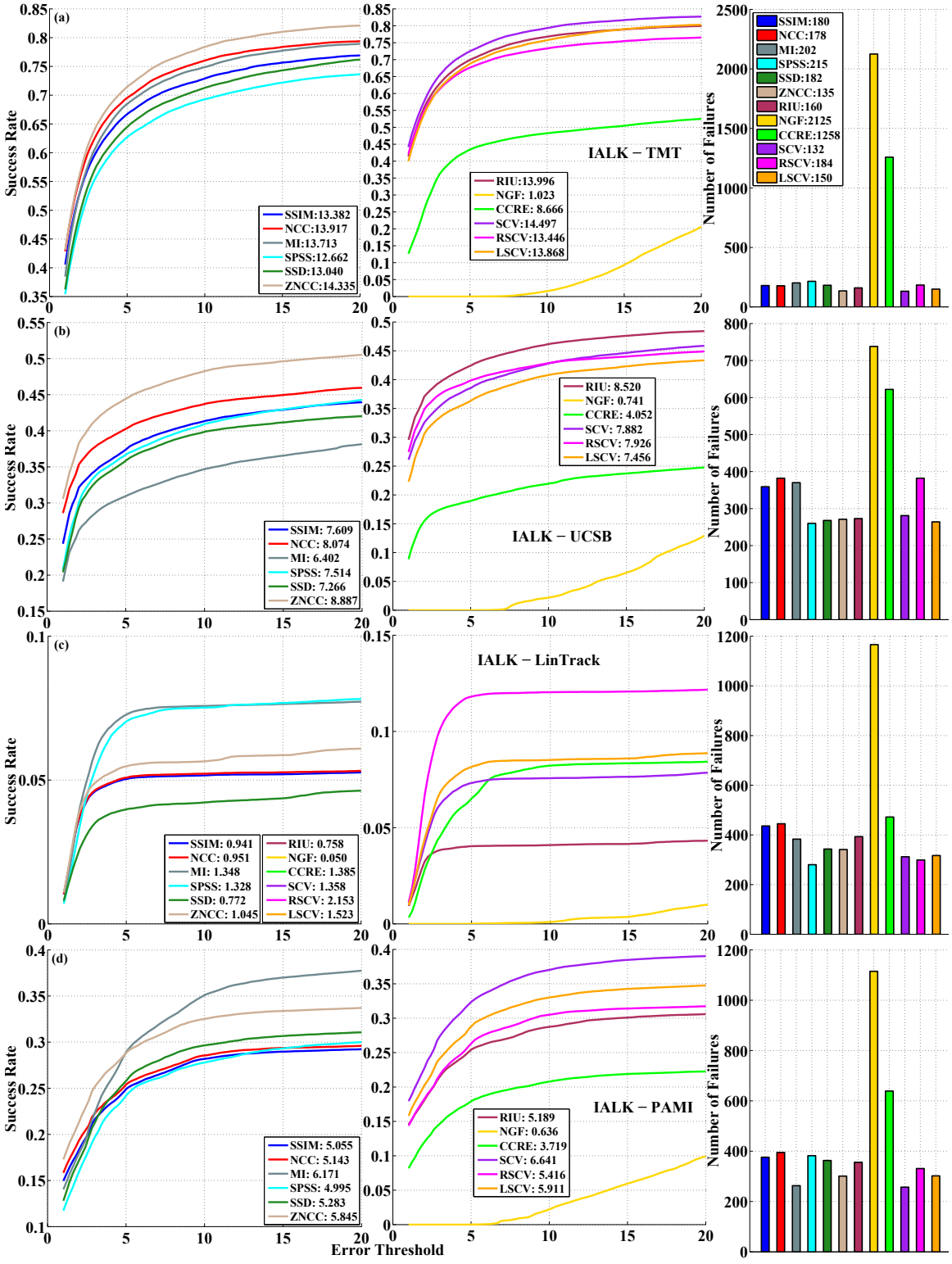


Figure B.4: Performance of IALK with LM Hessian on different datasets

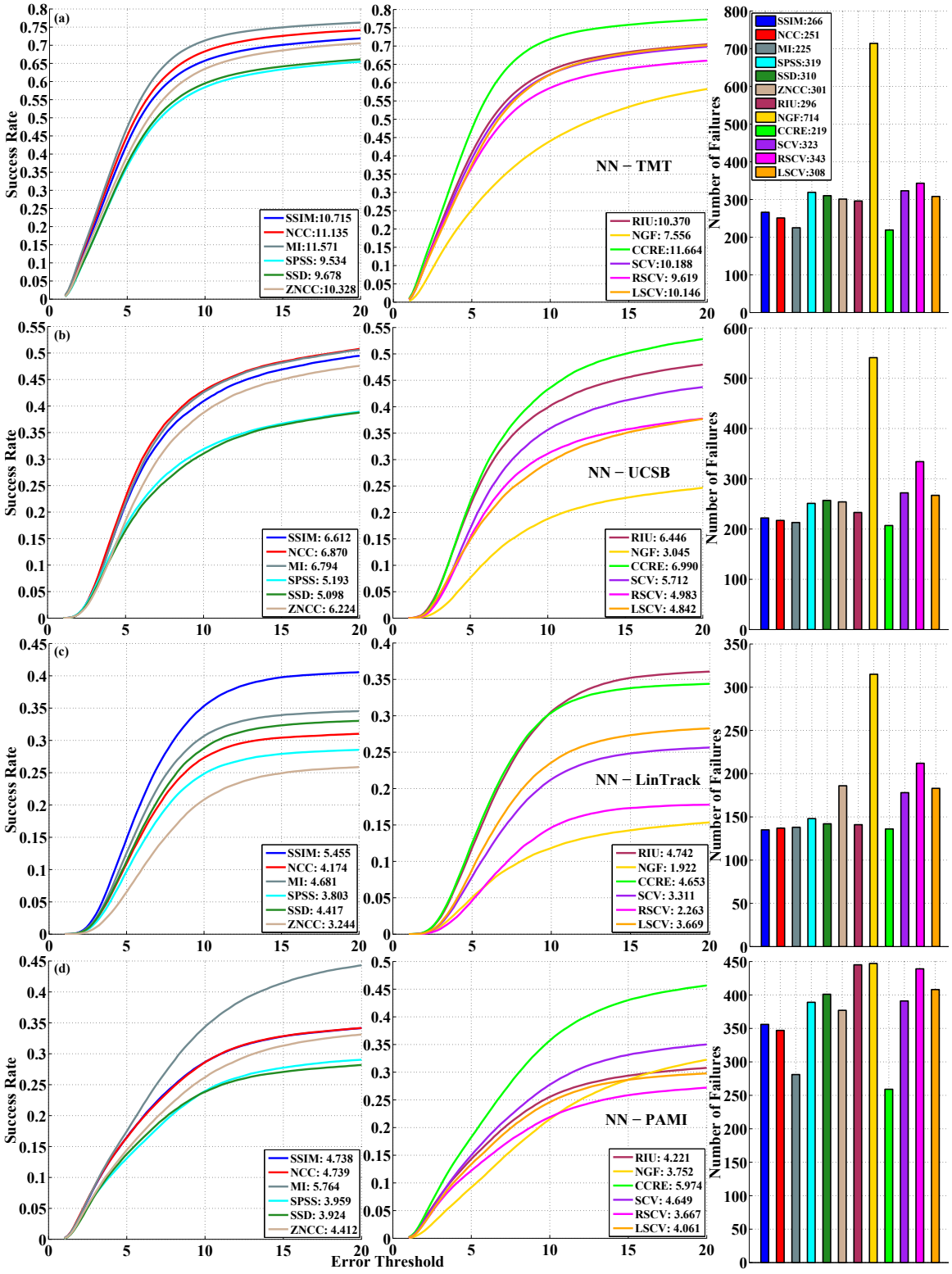


Figure B.5: Performance of AMs with NN on individual datasets

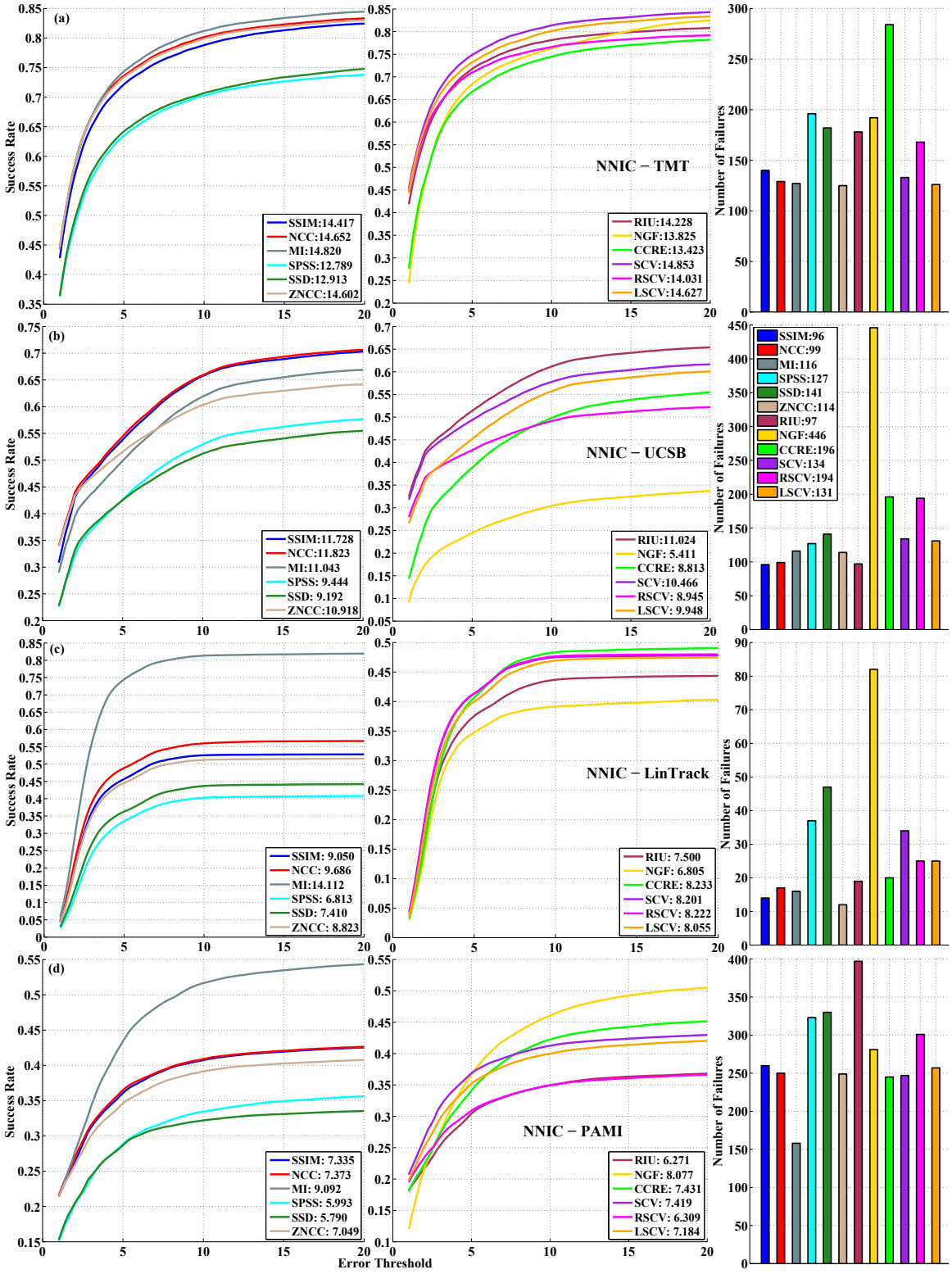


Figure B.6: Performance of AMs with NNIC on individual datasets

used here (Table 8.3) were designed specifically for sequences in the LinTrack dataset [1] among a few others and so have a significant advantage over the general purpose homography distributions. UCSB too offers many of the same challenges as LinTrack so these seem to work well there too. As expected, though, this advantage is non existent in general scenarios as represented by the other two datasets.

- PFFC works best on UCSB dataset with RIU (Fig. B.8 (b)) which shows again that RIU has a distinct advantage with scenarios containing both fast motion and global illumination changes. Further, SSIM is the best AM with LinTrack and the second best with UCSB showing that this too can outperform the generally better NCC in challenging scenarios. Finally, MI is the best AM on PAMI which may be indicative of its ability to handle localized illumination changes better.

- The advantage of SSD-CV over MTF AMs with LMS is clearly limited only to UCSB and LinTrack datasets (Fig. B.9 (b, c)). As fast motion is the major challenge in both datasets, this adds more weight to the hypothesis that it is some difference in the implementation of pyramidal tracking that is the underlying cause of this advantage.

- The nature of the SR curve of LMS/SSD-CV on both these datasets and particularly LinTrack, where it increases almost linearly with t_p, suggests that this tracker does not actually track the objects precisely at all. It manages to remain in the vicinity of the object without losing track completely but does not find the actual pose that is needed for many applications. This is of course a well known disadvantage of stochastic SMs and serves as a further motivation for using these as parts of composite SMs rather than by themselves.

- The related fact that LMS performs so poorly on these datasets with all other AMs indicates the strong dependence of this SM on the performance of the underlying sub patch trackers. It may also be noted that SSD and SPSS are actually the most robust among MTF AMs on these datasets. This seems to indicate that these trackers handle fast motion better with simpler pixel wise AMs which may partly be due to their wider convergence regions. Also, the smallness of their patches and the fact that they only track between consecutive frames means that the illumination invariance properties of more sophisticated

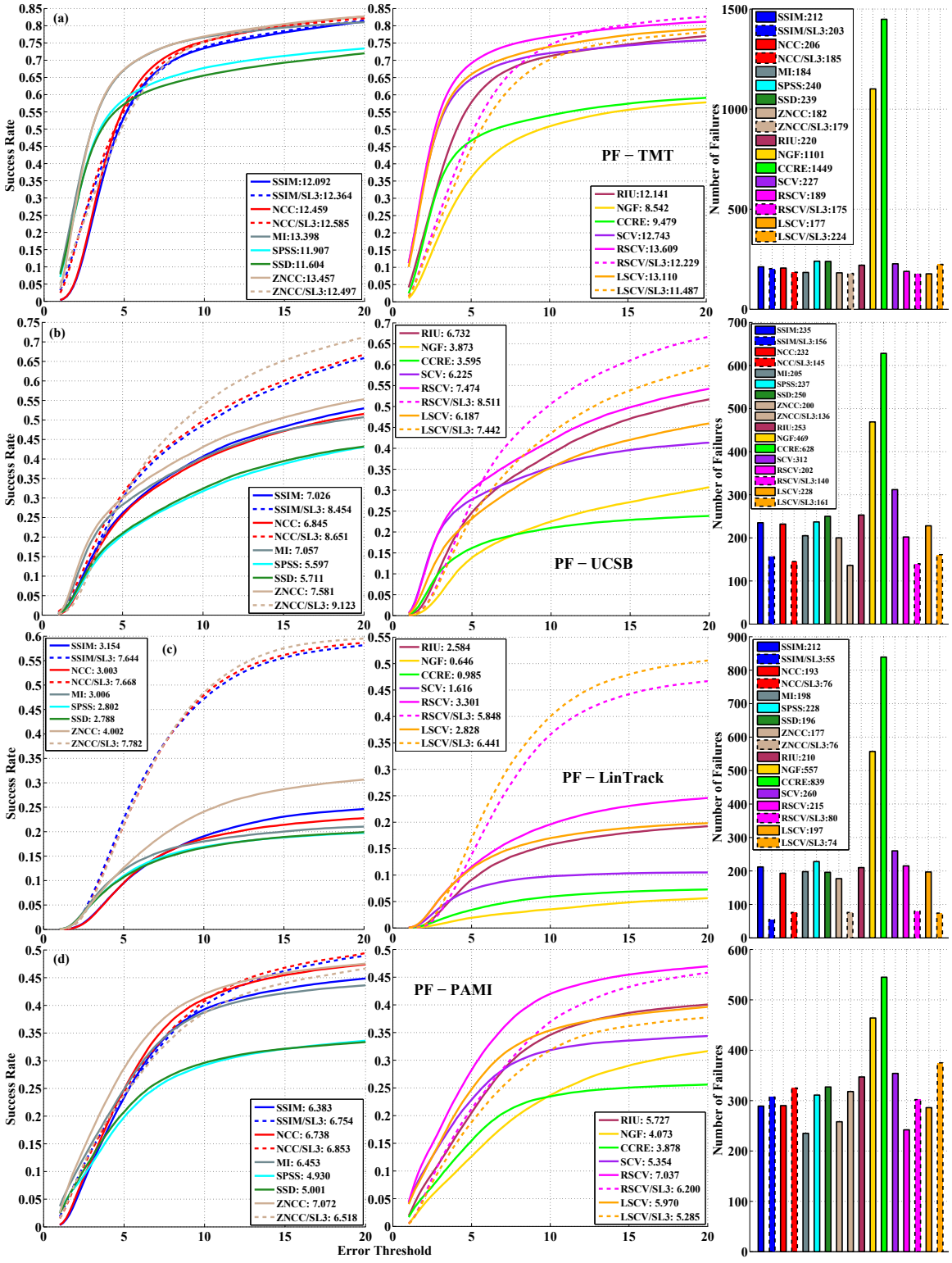


Figure B.7: Performance of AMs with PF on individual datasets

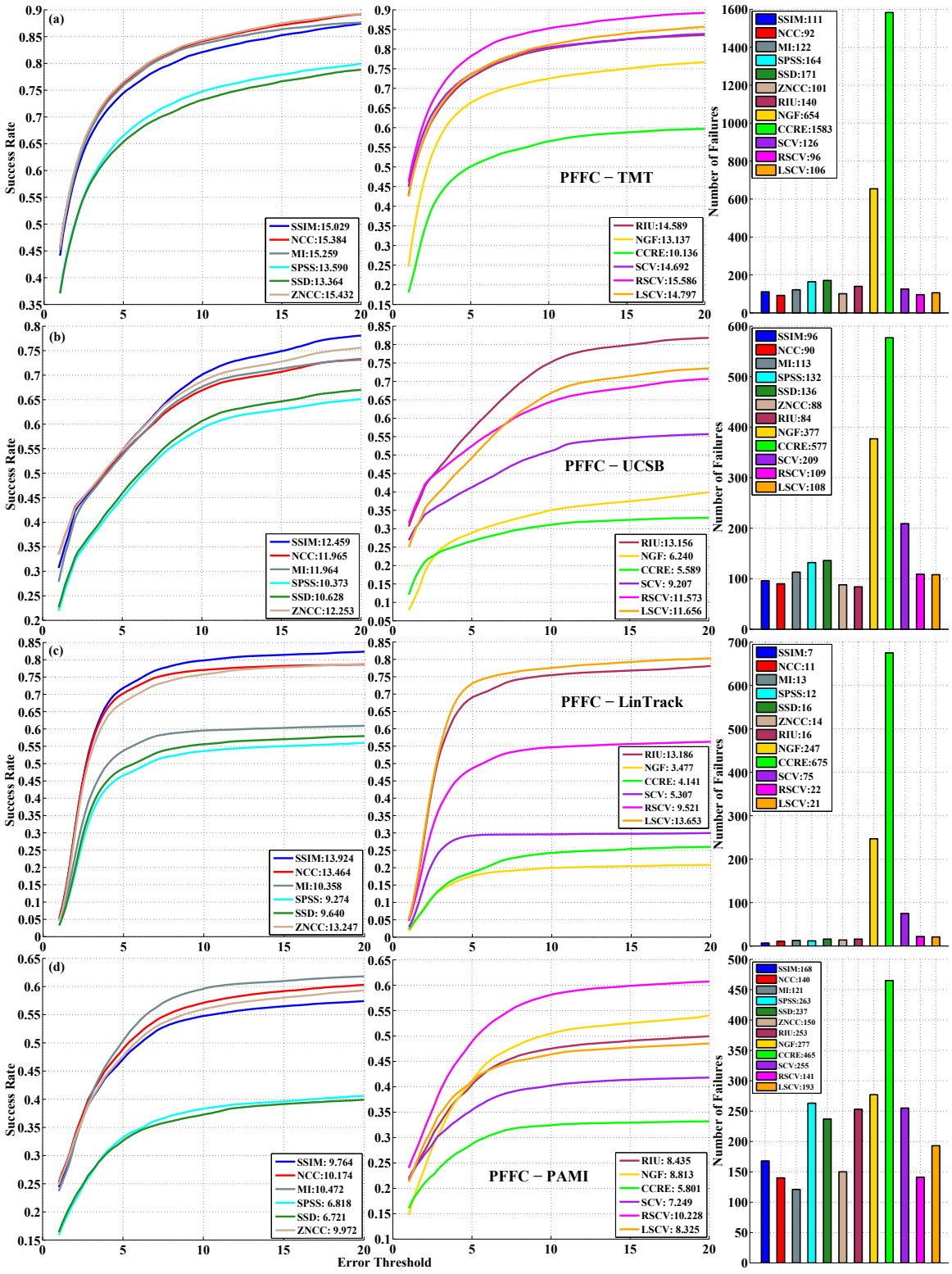


Figure B.8: Performance of AMs with PFFC on individual datasets

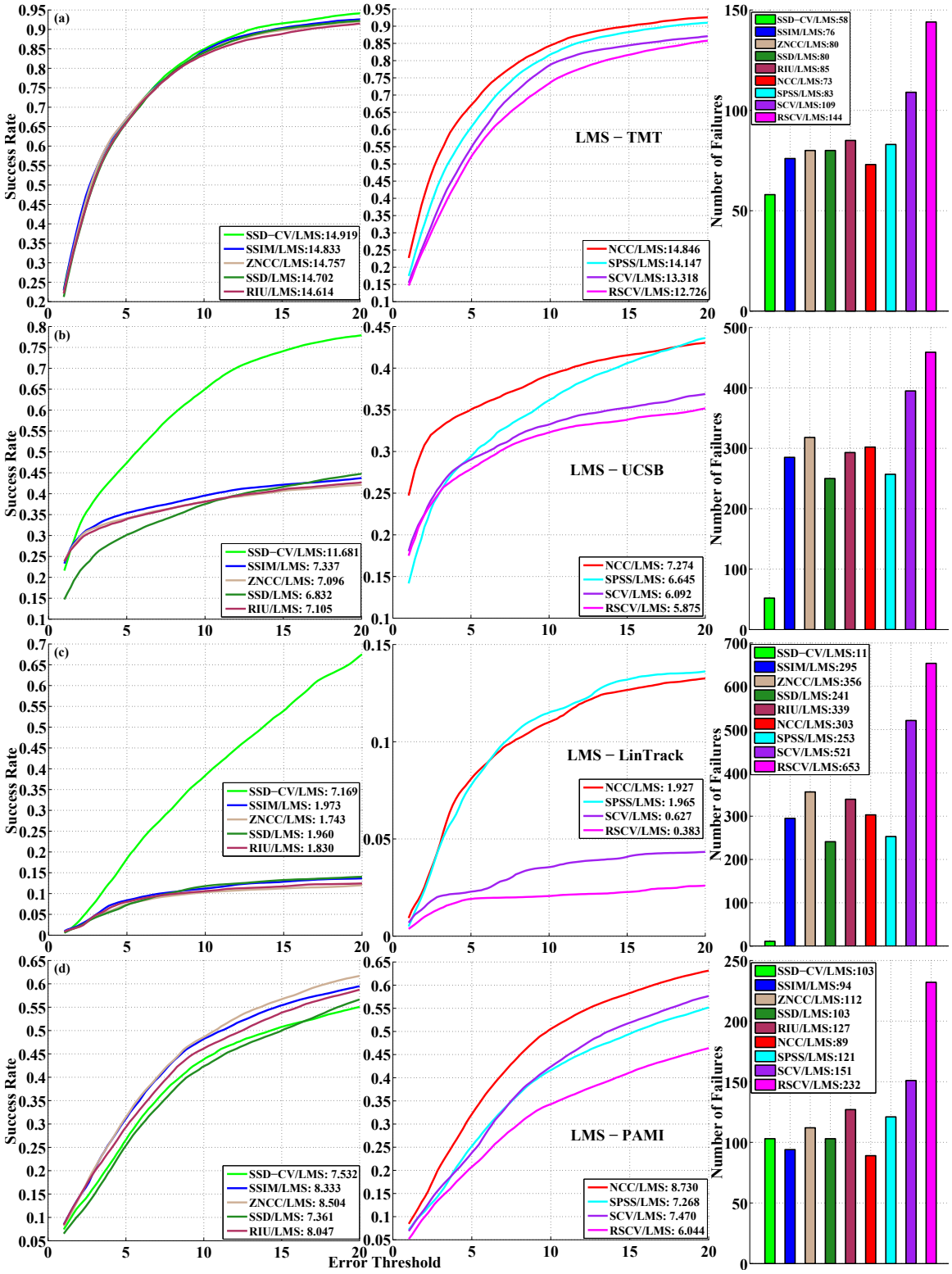


Figure B.9: Performance of AMs with LMS on individual datasets.

AMs are of little help, especially as they come at the cost of reduced convergence radius.

- Among MTF AMs, SSD is the best performer with LMES on UCSB and LinTrack (Fig. B.10 (b), (c)). This is followed by SPSS, at least in terms of robustness. In conjunction with the superiority of SSD-CV, this confirms the previous observation regarding the suitability of simpler pixel wise measures for the sub patch trackers in LMS.

- LMES with SSD-CV works much better on LinTrack and marginally better on UCSB when NCC is used in the second layer. This indicates that, unlike the sub patch trackers in LMS, the full patch tracker in this layer does benefit from NCC even in the absence of significant illumination changes. As NCC is also the best AM on both TMT and PAMI, its advantage in general tracking scenarios is evident too.

B.2 Illumination Models

Fig. B.11 presents results comparing ILMs with SSD and NCC over individual datasets. Following are some observations:

- Relative performance of the ILMs on individual datasets is mostly consistent with their overall performance (Fig. 9.24) both with respect to each other and to the two AMs.

- RBF with ESM does manage to outperform NCC on PAMI though only by a small margin. Considering that PAMI contains a large number of sequences with localized illumination changes designed specifically for testing ILMs, it would have been expected to perform better. It seems, however, that NCC can handle these challenges almost as well as RBF.

- As in the combined results (Fig. 9.24), RBF performs more favorably against NCC with ESM than FCLK and ICLK on each of the four datasets too. This strengthens the supposition that ESM can handle higher DOF ILMs better than the other two SMs.

- RBF appears to outperform NCC on LinTrack but all models perform so poorly there that such a lead probably is of little significance.

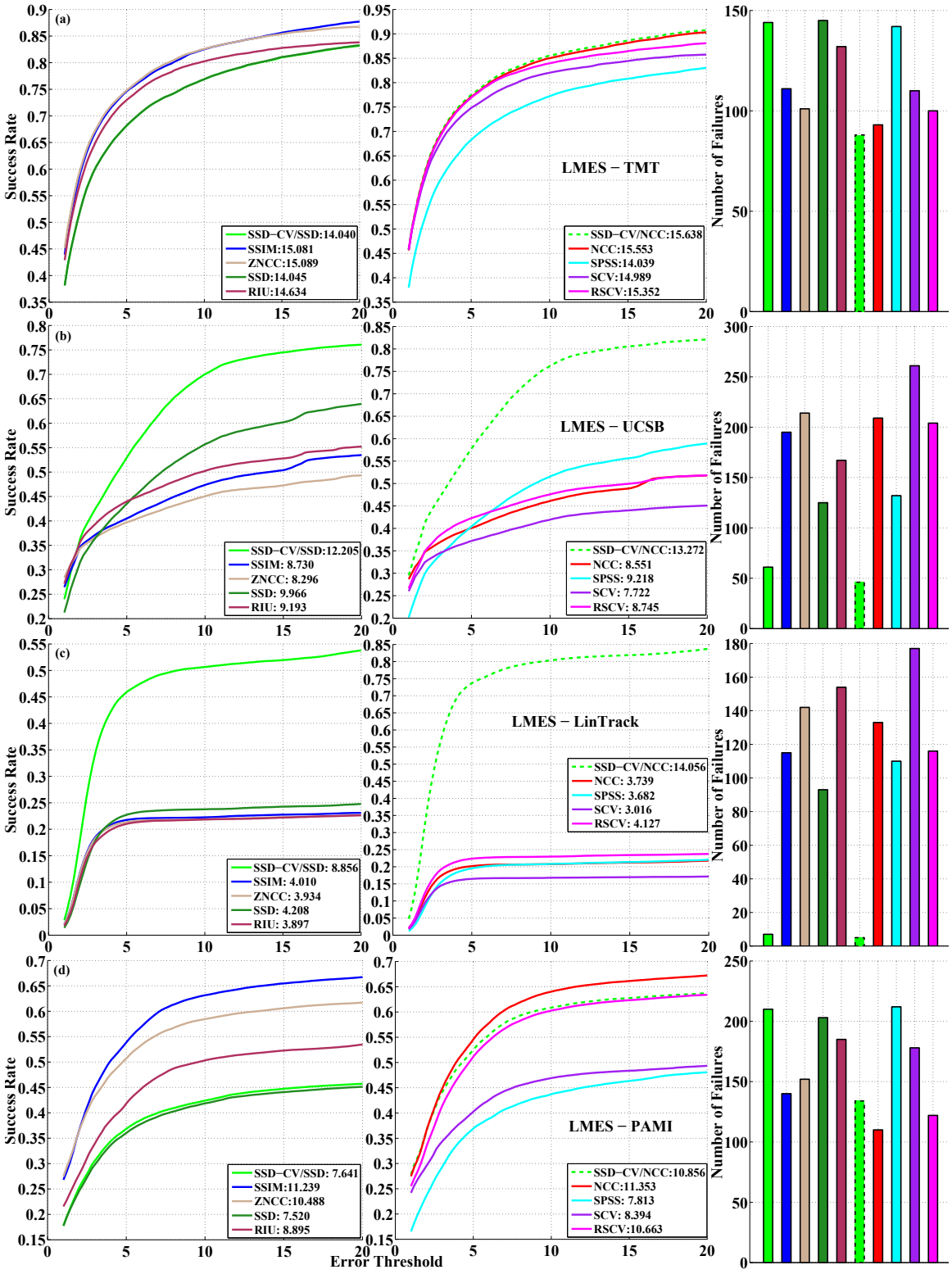


Figure B.10: Performance of AMs with LMES on individual datasets.

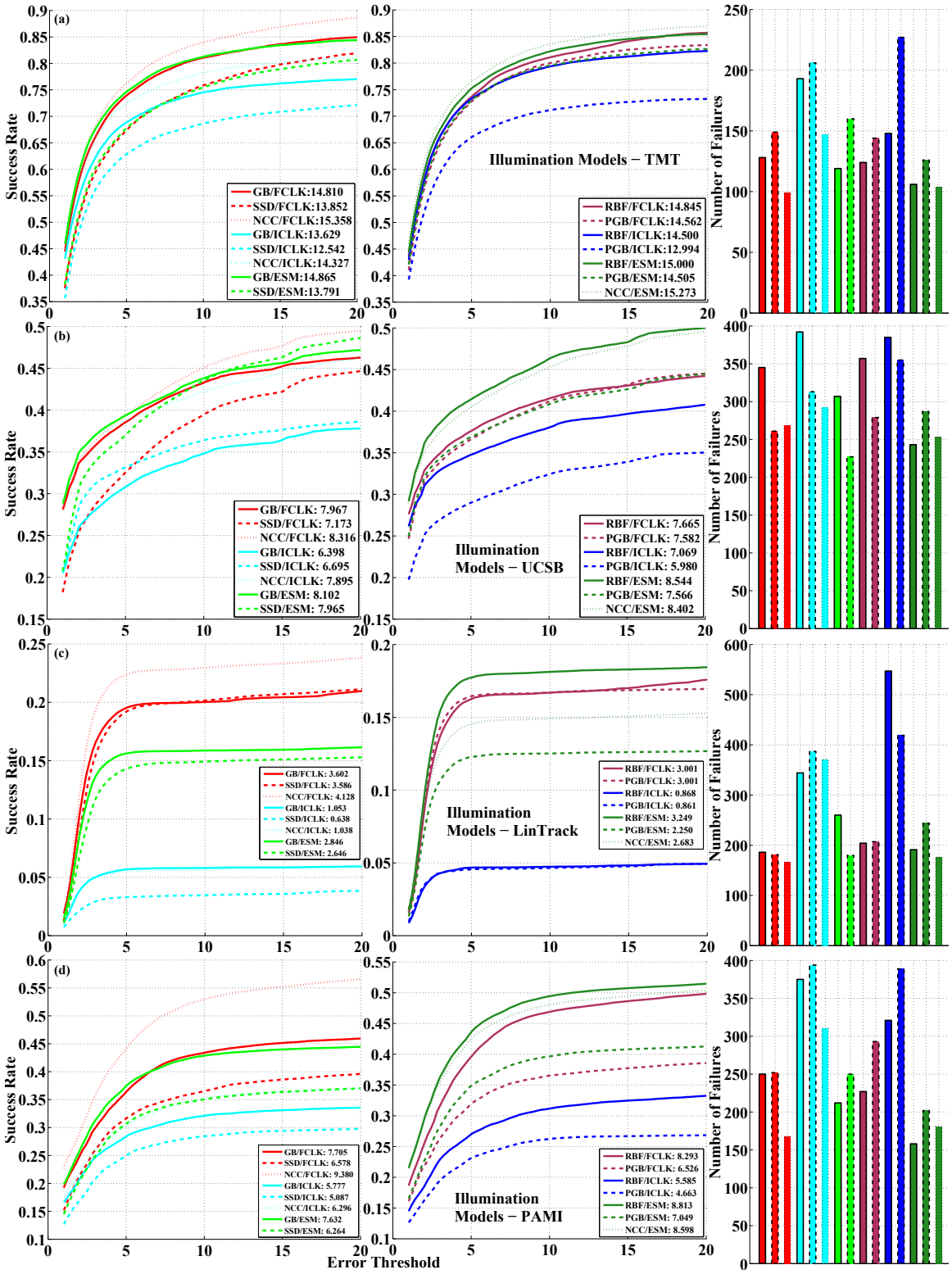


Figure B.11: Performance of ILMs on individual datasets