Joint Detection and Pose Estimation Based on Images

by

Shuchun Wen

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

Master of Science

in

Control Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Pose estimation is widely used in our daily lives and there exist many algorithms that are applied for bringing convenience and improving efficiency in various fields. While images are the most common input for feature detection and matching, it is necessary to study their knowledge background and mathematical representations for conducting implementations. Techniques of joint extraction and classification have been introduced in detail with conducted simulations for each algorithm and they are applied together for joint detection and classification of a robotic arm.

Several common object tracking algorithms have been introduced and implemented, including their mandatory background and mathematical representations. Multiple filter-based algorithms are simulated and discussed for object tracking purposes, including the Kalman Filter (KF), Extended Kalman Filter (EKF), and Particle Filter. Reinforcement Learning (RL) has been implemented together with EKF working towards a trajectory tracking and motion planning problem for robotic arm accomplishing a pick-and-place task. Enhanced methods are tested in the thesis.

Acknowledgements

First, I would like to thank my supervisor Dr. Qing Zhao for offering me this wonderful opportunity to explore a field that I never would have dreamt about. This new journey has not been easy, but with her patience, generosity, continuous support, guidance and encouragement, I can pull myself together every time when I encounter research difficulties, to reconstruct strong faith in completing the research and to grow better and better.

Second, I would like to thank my cousin Jinyu Zhao and sister Ping Guan for their unconditional love, kindness, patience, care, and tolerance over the past three years. I would also thank my parents, family and friends for all their love, support, understanding and encouragement.

Last but not least, I would like to thank my Father for his love and everything he has brought to my life.

Table of Contents

List of Tables

List of Figures

List of Symbols

Constants

α	Learning rate
δ	Dirac delta
γ	Discount factor
μ	Mean
A_k	State transition matrix
B_k	Control matrix
E	Expectation
f	Process nonlinear vector function
G	Gaussian operator
h	Observation nonlinear vector function
H_k	Transformation matrix
I_c	Intrinsic camera matrix
J	Jacobian
K_k	Kalman gain
P_k	Error covariance matrix
Q_k	Process noise covariance matrix
Q_k	Process noise covariance
R	Measurement noise covariance matrix
r	Reward
u_k	Control input
v_k	Measurement noise
Var	Variance

w_k	Process noise
x_k	State vector
z_k	Observation vector
$\mathbf{t}$	Translation vector
dB	Decibels

Abbreviations

BRIEF Binary Robust Independent Elementary Features.

CNN Convolutional Neural Network.

D Dimension.

DH Denavit-Hartenberg.

DNN Deep Neural Networks.

DoF Degrees of Freedom.

DoG Difference of Gaussians.

EKF Extended Kalman Filter.

FAST Features from Accelerated Segment Test.

KF Kalman Filter.

MDPs Markov Decision Processes.

ML Machine Learning.

NN Neural Networks.

ODE Ordinary Differential Equation.

ORB Oriented FAST and Rotated BRIEF.

PDF Probability Density Function.

PF Particle Filter.

RANSAC Random sample consensus.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

RLFJ rigid-link flexible-joint.

RRT Rapidly-exploring Random Tree.

SARSA State-Action-Reward-State-Action.

SIFT Scale Invariant Feature Transform.

SURF Speeded-Up Robust Features.

TD Temporal Difference.

Chapter 1

Introduction

Pose estimation is a computer vision technique that infers the pose of a person or object in an image or video. It can be treated as the problem of determining the position and orientation of a camera relative to a given person or object. It is typically done by identifying, locating, and tracking a number of keypoints on a given object or person. For objects, they could be corners or other significant features; for humans, they could represent major joints, like elbows, knees, and wrists.

Most inanimate objects are rigid, and rigid pose estimation is the process of making predictions of those objects. A task like object detection also locates objects within an image. This localization consists of a bounding box encompassing the object, while pose estimation can predict the precise location of its keypoints. For robotic arms, the keypoints are mostly the joints.

Pose estimation has the potential to create a new wave of automated tools designed to measure the precision of human movement. In addition to tracking human movement and activity, pose estimation opens up applications in a range of areas, such as: augmented reality, animation, gaming, and robotics, which raises the demand and development of various algorithms in order to bring them in reality.

1.1 Motivation and Background

For certain applications, e.g. autonomous driving, online learning algorithms need to be performed since the environment is changing all the time while the system is operated. Hence it needs a fast and accurate way to detect, recognize and classify the objects in order to avoid obstacles on the road and plan for the best route for the user. In other words, this system needs to learn the new environment while running, process the information with speed and accuracy, and provide the optimal solution based on the user's need from the perspective of computer vision. This requires extensive data collection and preparation, which includes a massive image database of the environment in various scenarios, geometric transformations between the camera and world coordinates, and training for the algorithms to reach the best result.

As objective tracking is widely utilized, it is important yet challenging to develop more effective methods with improved performance. Although Machine Learning (ML) and Deep Neural Network (DNN) based approaches have gained the most attention recently, they have limitations when applied to online and real-time situations, due to their inherent high computation requirement. Therefore, many traditional approaches are still being utilized and combined with these DNN approaches, such as Kalman Filter (KF) for maximizing the use of model information [1], and Particle Filter (PF) for improving the precision of human movement tracking [2].

1.2 Literature Review

Object detection and pose estimation are some fundamental problems in computer vision. Recently, there have been many proposed methods to increase the accuracy of object detection and tracking and reduce the processing time of matching features between videos and images. These methods encompass a wide range of work from the more classical filter-based to the more recent machine learning and neural network-

based techniques which have extraordinary computational abilities [3][4][5].

In the last decade, several approaches were proposed and demonstrated by different researchers for detection and tracking [6] [7] [8]. Occlusion, insufficient training data, and depth ambiguity are still challenges in pose estimation [9]. To deal with object occlusion, clutter background, illumination changes, object transformation and object variation, various local features have been proposed and they are applied to many applications such as object detection and recognition, image matching, image stitching [10], object tracking, etc. A Scale Invariant Feature Transform (SIFT) feature detector and descriptor for object recognition was proposed [11].

SIFT is an algorithm to detect, describe, and match local features in images. It can extract keypoints from the image and detect objects by comparing and matching features [12]. It is widely used for it has the advantage of scale-invariant demonstrated by [13]. Research results show that SIFT is invariant on rotations, translations, and scaling and SIFT features have strong matching robustness for radiation transformation, perspective changes, illumination changes and noises by studying the theories of SIFT matching, using Euclid distance as similarity measurement of keypoints and using RANSAC (Random Sample Consensus) to eliminate mismatches [14]. The SIFT method emerges as a valuable mechanism for enhancing the safety of autonomous vehicles by ensuring the accuracy and reliability of information influencing autonomous decision-making processes [15].

This work then becomes the original inspiration for most of the local feature descriptors proposed later. Then, a Speeded-Up Robust Features (SURF) was reported as a more efficient substitution for SIFT by H. Bay, et al. [16] since it produces a descriptor with a smaller feature size and also speeds up the matching step. SURF method plays a vital role as it detects, extracts the robust features [17] and describes the detected feature for matching purposes [18][19].

The SURF algorithm mainly includes three steps: interest point detection, interest point descriptor and interest point matching [20]. SURF is not only used innovatively

to solve the problem of automatic target detection and tracking in an unstabilized video by auto-detecting the target of interest followed by tracking [21], but also used for watermarking authentication [22]. While SURF is known to be a powerful algorithm, it is computationally expensive and therefore time-consuming. It can be further improved in feature detection by linearizing the SURF detector in a way that its detection characteristics are preserved [23]. It is known that the Haar descriptor of the SURF algorithm can not make full use of the information around the neighborhood of the feature points found [23]. [20] resolves this issue by proposing an improved SURF algorithm.

For object recognition systems, the goal is to make descriptors faster to compute, and more compact while remaining the robustness. To address these requirements, many researchers attempted to build a lightweight descriptor based on a binary string. A Binary Robust Independent Elementary Features (BRIEF) descriptor which relies on a relatively small number of intensity difference tests to represent an image patch as a binary string was proposed [24]. It is an important result from a practical point of view because it shows that real-time matching performance can be achieved even on devices with limited computational power. However, it is very sensitive to in-plane rotation. E. Rublee, et al. [25] built Oriented Features from Accelerated Segment Test (FAST) and Rotated BRIEF (ORB) to make an orientation invariant feature based on the well-known FAST keypoint detector [26] and the BRIEF descriptor [24], which is rotation invariant and resistant to noise. FAST detects the keypoint based on the difference of intensity value on a circle surrounding a point and looks for continuous arc that are either darker or lighter than the point. It is designed to be very efficient for real-time applications [27].

Although SIFT has many merits, it can be time-consuming. [28] proposed an improved SIFT algorithm that can generate feature descriptors based on hierarchical regions and treat those regions differently. A major challenge in object tracking is the occlusion of the target object by other objects in the scene. [29] proposed a

target tracking method based on SIFT and Kalman Filter: SIFT for computing the location of the target, and Kalman Filter (KF) for optimizing the target location in order to correct the error of SIFT. They are also implemented to track occluded and non-occluded objects in the videos in an automatic object tracking system [30]. The objects in the image sequences can be identified with the help of invariant features extracted using the SIFT algorithm. Then the occluded objects can be tracked using Kalman Filter since it optimally estimates the position of the object in the current frame using the information obtained from the previous frame so that the overall performance in the event of occlusion can be improved.

Generally, the Kalman Filter is used to optimally estimate the variables of interest when they can not be measured directly, but only when an indirect measurement is available. It can also be applied in moving object detection and tracking [31]. An improved Kalman Filter is capable of reducing the computational effort of the algorithm while improving the accuracy of the multi-object tracking results which solves the problem of target loss during multi-object tracking [32]. Similarly, the Extended Kalman Filter (EKF) can also be used to treat nonlinearities. It is applied to estimate the actual positions and velocities of the detected objects [33], as well as to accurately estimate the state of the target [34].

EKF can also be combined with a temporal-difference (TD) learning algorithm to train the active fault detector [35] and a reinforcement learning (RL) based algorithm for solving a game model [34], as RL can be used to provide a superior method of creating desired behaviors for agents [36]. Furthermore, an EKF model can be used to identify moving objects in the scene and incorporate deep learning and RL models for detecting the type of vehicle [37]. An EKF-based algorithm for trajectory tracking is proposed in [38]. It is capable of predicting the position of moving objects in dynamic environments.

Neural Networks (NN) are one of the most popular algorithms in pose estimation and are widely used in complicated contexts, like human pose estimation [39][40] [41]

and multi-object tracking [42] as object tracking and detection are strongly interconnected and they can benefit each other. The NN-based approach has the advantage of handling large amounts of data efficiently, which alleviates people's burden to a certain extent by allowing them to generalize a system without reprogramming.

Convolutional Neural Network (CNN) is used for image classification and recognition to improve significant performance. It is trained with millions of images of different classes [43]. CNN is the learning method which exploits the spatial information of an image and learns the complex features automatically [44]. It can be used for object tracking with shift variant architecture, for which the features were learned during an online process, and the spatial and temporal features are considered using a pair of images instead of a single image [45]. [46] used a pre-trained CNN model for online tracking. The CNN is used after parameter tuning to adjust the appearance of the object in the scene and a probability map is created instead of creating labels.

Deep learning techniques are tested to be reliable for 2D human pose estimation [9]. As the features extracted by the deep learning method usually contain noise and outliers, [47] designed a complex deep CNN to generate deep feature descriptors. Since deep CNN can learn high-level image abstractions, it has been applied for feature detection and achieves satisfactory results in low-level feature detections [48]. The low-level features and deep features can be combined to improve the performance of image retrieval tasks by enhancing the connection between different feature maps [49].

1.3 Contribution and Organization

With the rapid growth of machine learning in recent years, many techniques and algorithms have been developed for pose estimation in order to improve the effectiveness and efficiency for complicated tasks in various fields. Traditional approaches have come along at first and they have been applied with various popular new algorithms for delivering competitive results. As there are many existing algorithms for conduct-

ing pose estimation task for different purposes, this thesis focuses on the ones that are most widely used and most suitable for robotic arms.

This work first provides the background knowledge and preliminaries for image-based pose estimation of robotic arms, such as the process of designing CNN for joint classification, Reinforcement Learning (RL) and the structure of a robotic arm. Kalman Filter, Extended Kalman Filter and Particle Filter are also introduced as the foundation for object tracking. A detailed description of pick-and-place platform is presented and applied as a 3D simulation environment.

The working mechanisms of object detection algorithms like SIFT and SURF, are explained in stages with conducted simulations along with the discussion of their advantages and disadvantages. Object tracking simulations with Kalman Filters and its extensions are provided and their results can serve as a base for conducting a more complicated task.

In the last part of the thesis, with several modifications of the pick-and-place environment, RL is applied with EKF for state prediction and update to achieve a better performance for object tracking, along with object detection algorithms being used as the preparation of the input for tracking.

In summary, Chapter 2 provides a knowledge base for pose estimation, including filters, camera intrinsics and extrinsics, the structure of CNN, an introduction to RL, the kinematics of robotic arm and the setup of the simulation environment. Chapter 3 includes all the preliminaries and simulation results for joint detection and classification. A complete simulation of joint tracking and motion planning is presented in Chapter 4 with the applications of all the methods presented in the previous chapters along with a reinforcement learning algorithm combined with the EKF designed for adaptive tracking. Chapter 5 concludes the thesis by summarizing the presented work and simulation results, as well as discussing more future research work and directions.

Chapter 2

Background and Preliminaries

This chapter introduces background knowledge for conducting image-based robotic arm pose estimation. The sections start with an introduction to filters, camera intrinsics and extrinsics, the process of designing a pre-trained Convolutional Neural Network (CNN) and its structure, Reinforcement Learning (RL), as well as robotic arm kinematics. The platform for a pick-and-place task is presented in the last section.

2.1 Filters

2.1.1 Kalman Filter (KF)

Kalman Filter (KF) uses the prior knowledge of the state to make a forward projection state or a prediction of the next state.

$$x_{k+1} = A_k x_k + B_k u_k + w_k, \tag{2.1}$$

A measurement model z_k that describes a relation between the state and measurement at the current step k always comes along with x_k.

$$z_k = H_k x_k + v_k, \tag{2.2}$$

where $A_k \in \mathbb{R}^{n \times n}$ is a state transition matrix relating the state at time step k to the state at time step $k+1$; $B_k \in \mathbb{R}^{n \times p}$ is a control input matrix for the input $u_k \in \mathbb{R}^{p \times 1}$. $H_k \in \mathbb{R}^{m \times n}$ is the measurement transformation matrix that transforms the state to

the measurement z_k. w_k is the process noise with covariance $Q_k \in \mathbb{R}^{n\times n}$, and v_k is the measurement noise with covariance $R \in \mathbb{R}^{m\times m}$, and they are statistically independent Gaussian.

The equations of Kalman Filter are divided into two groups. The time update equations (or the predictor equations) are responsible for projecting forward the current state and error covariance estimates to obtain the priori estimates for the next time step. The measurement update equations (or the corrector equations) are responsible for improving the posteriori estimate by incorporating a new measurement into the priori estimate. The goal is to minimizing the trace of the error covariance matrix.

1. Predictor equations

   A priori state estimate ${\hat{x}_{k+1}}^-$ is predicted by using the state dynamic equation that projects one step in time:

   $$ {\hat{x}_{k+1}}^- = A\hat{x}_k + Bu_k. \tag{2.3} $$

   The error covariance matrix ${P_{k+1}}^-$ is predicted by:

   $$ {P_{k+1}}^- = AP_k^- A^T + Q_k, \tag{2.4} $$

   where ${P_k}^-$ is the previous estimated error covariance matrix with

   $$ P_k = E\langle e_k, e_k^T \rangle, \tag{2.5} $$

   where e_k is the estimation error $e_k = x_k - \hat{x}_k$ and Q_k is the process noise covariance.

2. Corrector equations

   The Kalman gain K_k can be computed as follows:

   $$ K_k = {P_k}^- {H_k}^T (H_k {P_k}^- {H_k}^T + R_k)^{-1}, \tag{2.6} $$

where R_k is the measurement error covariance.

The measurement residual is the difference between the actual measurement z_k and the previous estimated measurement $H_k\hat{x}_k{}^-$. The update of the predicted state estimate $\hat{x}_k$ is proceeded by the summation of the priori projected state estimate $\hat{x}_k{}^-$ to the product of the Kalman gain and the measurement residual:

$$\hat{x}_k = \hat{x}_k{}^- + K_k(z_k - H_k\hat{x}_k{}^-) \tag{2.7}$$

With calculated $\hat{x}_k$, the filter calculates the updated error covariance P_k, which will be used in the next time step:

$$P_k = (I - K_kH_k)P_k{}^-. \tag{2.8}$$

2.1.2 Extended Kalman Filter (EKF)

While most of the real-world systems are nonlinear, Extended Kalman Filter (EKF) has been applied with adapted techniques from multivariate Taylor series expansions to linearize a model about a working point. EKF can start with a continuous ordinary differential equation (ODE):

$$\dot{x}_k = f(x_{k-1}, u_k, w_k), \tag{2.9}$$

which is then linearized and discretized.

$$z_k = h(x_k, v_k), \tag{2.10}$$

where w_k and v_k are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance Q_k and R_k respectively. u_k is the control input.

Function f is used to compute the predicted state from the previous estimate and function h is used to compute the predicted measurement from the predicted state. However, they cannot be applied to the covariance directly. Instead, a matrix of partial derivatives (the Jacobian) is computed. At each time step, the Jacobian

is evaluated with current predicted states. This process essentially linearizes the nonlinear function around the current estimate. By assuming the nonlinearities in the dynamic and observation model, functions f and h can be expanded in Taylor Series and proceed the approximations and predictions.

The process starts with an initialization of x_0:

$$x_0 = \mu_0 = E[x_0], \tag{2.11}$$

with error covariance P_0, where μ_0 is the mean, and x_0 represents the initial optimal estimate.

By assuming an optimal estimate x_{k-1} with P_{k-1} covariance at time $k-1$, the predicted state estimate is:

$$\hat{x}_k \approx f(x_{k-1}), \tag{2.12}$$

where $f(x_{k-1})$ is acquired by expanding the process nonlinear vector funtion f in Taylor Series about x_{k-1}, and predicted covariance estimate is:

$$P_k{}^- = J_f(x_{k-1})P_{k-1}J_f^T(x_{k-1}) + Q_{k-1}, \tag{2.13}$$

with

$$J_f = \frac{\partial f}{\partial x}. \tag{2.14}$$

The equations of corrector is given below. Kalman gain is:

$$K_k = P_k{}^- J_h^T(x_k)(J_h(x_k)P_k{}^- J_h^T(x_k) + R_k)^{-1}, \tag{2.15}$$

with

$$J_h = \frac{\partial h}{\partial x}. \tag{2.16}$$

The state estimate is:

$$\hat{x}_k \approx x_k + K_k(z_k - J_h(x_k)\hat{x}_k{}^-), \tag{2.17}$$

and the posterior covariance of the new estimate is:

$$P_k = (I - K_k J_h(x_k))P_k{}^-, \tag{2.18}$$

with R_k as the measurement noise covariance matrix.

2.1.3 Particle Filter (PF)

The key idea of particle filtering is to represent the posterior probability density function by a set of discrete samples known as particles. Each particle represents a hypothesis of the state and it is randomly drawn from the prior density. After a particle is drawn, it is then propagated according to the transition model. Each propagated particle is verified by a weight assignment using the likelihood model.

The posterior probability density function is constructed recursively by the set of weighted random samples $x_k^{(i)}$, $w_k^{(i)}$; $i = 1, ..., N$ where N is the total number of particles. At each time k, the particle filtering algorithm repeats a two-stage procedure, prediction and update:

1. For prediction, each particle $x_k^{(i)}$ evolves independently according to the state model, including the addition of random noise in order to simulate the unknown disturbance. The step yields an approximation of the prior probability density function:
$$p(x_k) \approx \frac{1}{N} \sum_{i=1}^{N} \delta(x_k - x_k^{(i)}), \tag{2.19}$$
where $\delta(\cdot)$ denotes the Dirac delta function. The Dirac delta function $\delta(a)$ is zero everywhere except for a, and its integral is equal to 1.

2. For update, the weight of each particle is evaluated based on the latest measurement according to the measurement model (likelihood model). The posterior probability density function at time k in the form of a discrete approximation can be written as:
$$p(x_k|z_{1:k}) \approx \sum_{i=1}^{N} w_k^{(i)} \delta(x_k - x_k^{(i)}), \tag{2.20}$$
and the sum of the weight set equals to 1, with z represents the measured sequence in the equation.

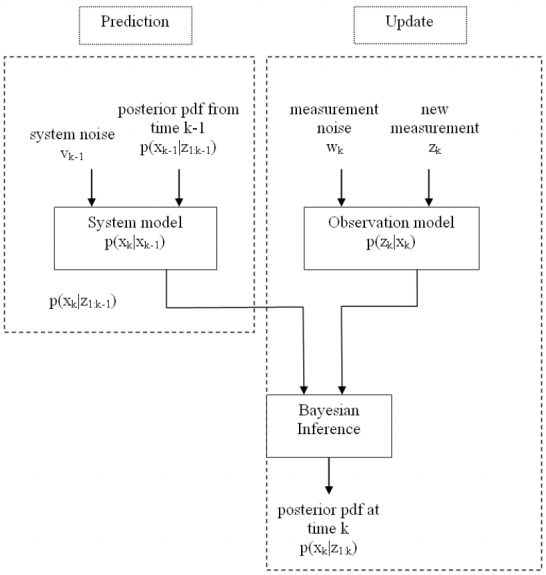


Figure 2.1: One iteration of prediction and update, extracted from [50]

2.2 Camera Intrinsics and Extrinsics

Intrinsics and extrinsics are parameters of a camera. Extrinsics describe the location of the camera in the 3D world, while intrinsics are parameters inside the camera.

1. Extrinsic camera matrix

   The camera's extrinsic matrix describes the camera's location, and what direction it is pointing. It has two components: a rotation matrix, R, and a translation vector $\mathbf{t}$.

   The extrinsic matrix takes the form of a rigid transformation matrix: a 3×3 rotation matrix in the left block, and 3×1 translation column vector in the right:

$$\Big[R|\mathbf{t}\Big] = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3}|\mathbf{t}_1 \\ r_{2,1} & r_{2,2} & r_{2,3}|\mathbf{t}_2 \\ r_{3,1} & r_{3,2} & r_{3,3}|\mathbf{t}_3 \end{bmatrix} \tag{2.21}$$

   It is common to see a version of this matrix with extra row of (0, 0, 0, 1) added to the bottom. This makes the matrix square, which allows for further decomposition into a rotation followed by translation.

2. Intrinsic camera matrix

   The intrinsic matrix I_c transforms 3D camera coordinates to 2D homogeneous image coordinates, by assuming that the projection is modeled by an ideal pinhole camera,

$$I_c = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.22}$$

   with s encodes any possible skew between the sensor axes due to the sensor not being mounted perpendicular to the optical axis and (c_x, c_y) denotes the optical center expressed in pixel coordinates. f_x and f_y represent focal lengths, i.e. the distance between the pinhole and the film (also known as the image plane). The focal length is measured in pixels. In a true pinhole camera, f_x and f_y have the same value.

   By using a single focal length and an aspect ratio that describes the amount of deviation from a perfectly square pixel, the camera geometry (focal length) from distortion (aspect ratio) can be separated.

   There are infinitely many pinhole cameras that can produce the same image. The intrinsic matrix is only concerned with the relationship between camera coordinates and image coordinates, so the absolute camera dimensions are irrelevant. Using pixel units for focal length and principal point offset allows for representing the relative dimensions of the camera, which is the film's position relative to its size in pixels. This suggests the intrinsic camera transformation is invariant to a uniform scaling of the camera geometry, and the invariance can be captured by representing dimensions in pixel units.

2.3 Convolutional Neural Network (CNN)

The Convolutional Neural Network (CNN) is a deep learning algorithm commonly used to recognize patterns in image data. It basically consists of convolutional layers,

pooling layers and fully-connected layers. A simple convolutional neural network for deep learning classification has been created and trained within Matlab, which involves the following steps:

1. Load Image Data

   Load the sample image data. The data are divided into training and validation data sets. For example, 30% of the training data are set aside to be used as validation data and observations to the training and validation sets are randomly allocated.

2. Define Network Architecture

   The size of the images in the input layer of the network and the number of classes in the fully-connected layer are specified before the classification layer. The structure of the layers is given in Fig. 2.2.

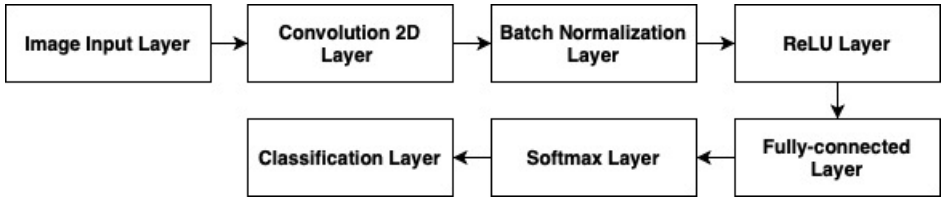


Figure 2.2: CNN Network Architecture

   (a) Convolution 2D Layer

   The convolution kernel in the convolutional layer is the key to the automatic extraction of image features by CNN. Different convolution kernels can extract different features. The calculation formula for the convolutional layer feature extraction is:

$$A = \sigma(W^T \cdot X + b), \tag{2.23}$$

   where $\sigma(\cdot)$ is the activation function, with input X, weights W, and bias b. The filter size defines the size of the local region of the feature matrix, which can be set as 3, 5, or 7. The number of filters is set to be 32,

which corresponds to the number of neurons in the convolutional layer that connect to the same region in the input.

Stride is the step size for traversing the input vertically and horizontally. They are both equal to 1, same as the factor for dilated convolution.

In order to let the output have the same size as the input when the stride equals 1, the size of padding is applied to input borders vertically and horizontally, specified as a vector $[a\ b]$ of two nonnegative integers, where a is the padding applied to the top and bottom of the input data and b is the padding applied to the left and right. They are set to be 0.

(b) Batch Normalization Layer

Batch normalization layer normalizes the activations and gradients propagating through a neural network, acrossing all the observations for each channel independently. It is used between convolutional layer and nonlinearities to speed up neural network training and reduce the sensitivity to neural network initialization.

(c) Rectified Linear Unit (ReLU) Layer

Rectified Linear Unit (ReLU) layer performs a threshold operation max(0, x) to each element of the input, thresholding at zero.

(d) Fully-connected Layer

A fully-connected layer combines all the features learned by the previous layers across the image to identify the larger patterns. It multiplies the input by a weight matrix and then adds a bias vector. The last fully-connected layer combines the features to classify the images. The input size is automatically determined during training, and the output size is equal to the number of classes in the target data.

The Glorot initializer is applied as the function to initialize the weights, which independently samples from a uniform distribution with zero mean

and variance 2/(input size + output size). The function values to initialize biases are zeros.

(e) Softmax Layer

The softmax activation function normalizes the output of the fully-connected layer. The output of the softmax layer consists of positive numbers that sum to one, which can then be used as classification probabilities by the classification layer. It is defined as:

$$S(x_i) = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}, \tag{2.24}$$

where x is an input vector to the softmax function S, x_i is the ith element of the input vector, and n represents the number of classes (possible outcomes).

(f) Classification Layer

It computes the cross-entropy loss for classification and weighted classification tasks with mutually exclusive classes. The layer infers the number of classes from the output size of the previous layer. Unweighted cross-entropy loss is applied for class weights. The classes of the output layer are automatically set at training time.

3. Train the Network

During the training process, options need to be specified, e.g. validation frequency, maximum of epoch, and evaluation metrics. The training is for joints classification. The classification accuracy is the percentage of correctly predicted labels.

2.4 Reinforcement Learning (RL)

Reinforcement Learning (RL) is a type of machine learning technique that is based on the agent's learning strategies to maximize returns or achieve specific goals while

exploring and interacting with the environment by trial and error using feedback from its actions and experiences. In general, it consists of an agent and the interaction environment. An RL agent may include one or more of the following components:

1. Policy: is the agent's behaviour function. It maps from state to action.

2. Value function: is a prediction of future reward. It is used to evaluate the goodness of each state, and therefore select between actions.

3. Model: is the agent's representation of the environment.

The agent sends actions to the environment, and then the environment moves to a new state and generates rewards, which are sent back to the agent for the next step. The state represents the observation of the environment, and the rewards evaluate the outcome of the current action. The objective of RL is to learn the optimal policy for the agent to take the best action at each step. When interacting with the environment, an agent always follows a certain behavior pattern during the entire procedure.

Sarsa(λ) Algorithm is implemented in the thesis as it can update the λ step before getting the reward, which enables the agent to learn how to get the maximum reward more efficiently. The process of the Sarsa(λ) algorithm is shown in Algorithm 1.

Based on the algorithm, a Q-value table has been built to save the state s, all actions a that will be taken, and $Q(s, a)$. In each round, the first state and action are randomly initialized and execute action to get the reward r and the new state s'. Then use ϵ-greedy to select action a' from the Q-value table based on the current state s'. The value functions $Q(s, a)$ are updated for the state s and corresponding action a of the current sequence [51]. α is the learning rate, γ is the discount factor, and $r + \gamma * Q(s', a')$ is the estimated return for the next state-action pair.

Algorithm 1 Sarsa(λ) algorithm

1. Initialize $Q(s, a)$ arbitrarily
2. Repeat for each episode:
   (a) Initialize s
   (b) Choose a from s using policy derived from Q (ϵ-greedy)
   (c) Repeat for each step of episode
      i. Take action a, observe r, s'
      ii. Choose a' from s' using policy derived from Q (ϵ-greedy)
      iii. $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
      iv. $s \leftarrow s'$; $a \leftarrow a'$
   (d) until s is terminal

2.5 Robotic Arm Kinematics

1. Dynamics

   The robotic arm for our simulation is based on Kinova Jaco robotic arm, which is a light-weight robot composed of six inter-linked segments. Through the controller, the robot can be moved in three-dimensional space and grasp or release objects with the gripper. The simulations are conducted in Matlab simulink environment. Figure 2.3 provides the structure and components of the applied robotic arm. The n-link rigid-link flexible-joint (RLFJ) robot dynamics can be expressed as:

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + G(q) = u, \tag{2.25}$$

   by assuming where q, $\dot{q}$, $\ddot{q} \in \mathbb{R}^n$ represent the position, velocity, and acceleration of the joints respectively. $M(q) \in \mathbb{R}^{n \times n}$ is the symmetric and positive definite inertia matrix, $C(q, \dot{q})\dot{q} \in \mathbb{R}^{n \times n}$ denotes the Coriolis and centrifugal vector, $G(q)$ $\in \mathbb{R}^n$ is the gravity vector and $u \in \mathbb{R}^n$ is the joint torque with $n = 6$.

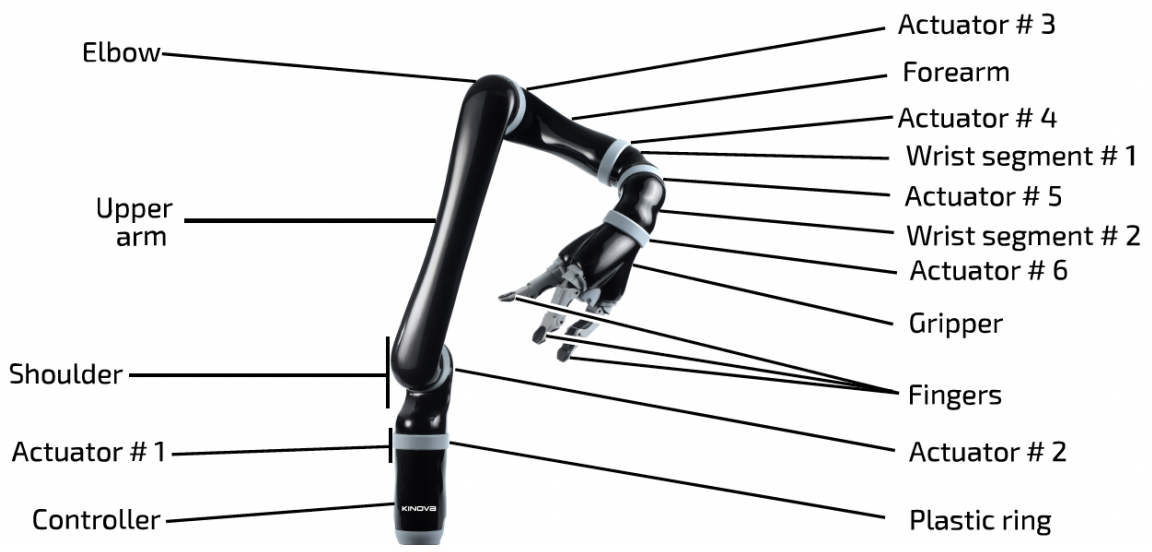


Figure 2.3: Components of Kinova Jaco Assistive Robotics Arm, extracted from [52]

2. Kinematics

The pose of each joint in the robot's kinematic chain is represented by homogeneous transformation. A Denavit-Hartenberg (DH) transformation matrix has been applied to describe the relationship between two adjacent links in the robotic arm. It is defined by link length, link twist, link offset, and joint angle. A 4×4 homogeneous transformation matrix T describes the position and orientation of a joint in space:

$$T = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cos\alpha_n & \sin\theta_n \sin\alpha_n & r_n \cos\theta_n \\ \sin\theta_n & \cos\theta_n \cos\alpha_n & -\cos\theta_n \cos\alpha_n & r_n \sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.26}$$

θ_n is the rotation around the direction of the joint axis. d_n is the sliding motion along the joint axis. α_n and r_n define the physical dimensions of the link in terms of the angle measured around and distance measured along the axis that is parallel to the common normal.

The Jacobian matrix is used to compute the joint velocities. It relates the

velocity of the end-effector to the velocity of the joints as follows:

$$V_e = J(q) * V_q \tag{2.27}$$

where V_e is the velocity of the end-effector, $J(q)$ is the Jacobian matrix evaluated at the current joint angles q, and V_q is the velocity of the robot's joints. The Jacobian matrix for the robotic arm is a 6×6 matrix that consists of both linear and angular velocity components. The linear velocity component represents the rate of change of the end effector's position with respect to the joint angles, while the angular velocity component represents the rate of change of the end effector's orientation with respect to the joint angles.

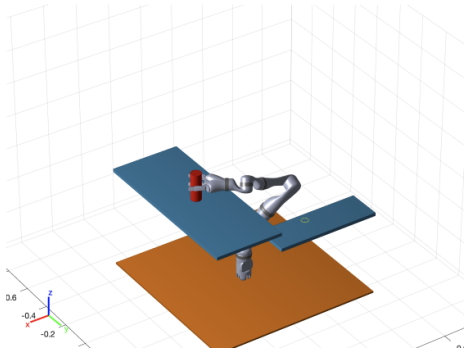
2.6 Pick-and-Place Platform

A pick-and-place platform has been constructed for a Kinova Jaco Assistive Robotics Arm for trajectory tracking and motion planning within Matlab as our simulation environment. The environment consists of one Kinova Jaco robotic arm, a ground floor, two table tops in different heights locating above the floor, and a red cylinder can on the upper table top. In order to improve the complexity and difficulty for simulation, a green cylinder can in different size is later set on the upper table top, together with the one in red. In addition, a blue sphere will appear in the environment in the middle of the mission as an unknown obstacle for interfering the movement of the robotic arm. The radiuses, height, length, width and their positions are shown in the configuration table.

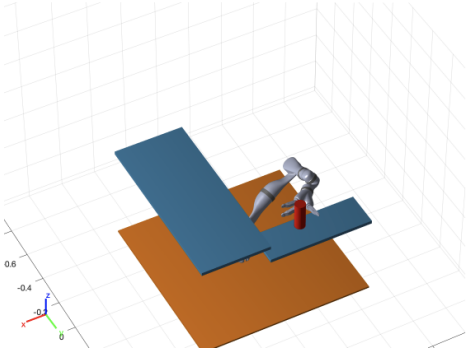
The starting point (initial position of the gripper) of the robotic arm is set as a point between the table tops. The target position can be set by the user, for example, in our experiment, it is set to be [-0.15, 0.35, 0.51]. It needs to pass through the lower table top to pick the red cylinder can, and then place it to the target point on the lower table top without touching anything in the environment.

Objects	Size	Pose
Floor	(1, 1, 0.01)	N/A
Tabletop 1	(0.4, 1, 0.02)	[0.3, 0, 0.6]
Tabletop 2	(0.6, 0.2, 0.02)	[-0.2, 0.4, 0.5]
Red cylinder can	(0.03, 0.16)	[0.3, 0.0, 0.7]
Green cylinder can	(0.03, 0.1)	[0.3, 0.2, 0.7]
Blue sphere	(0.06)	[0.15, 0.1, 0.7]

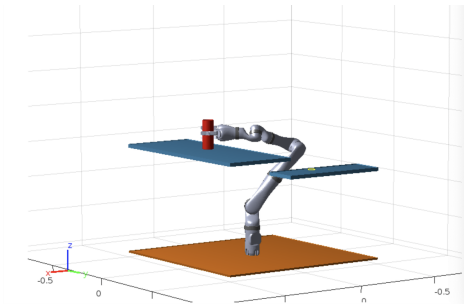
Table 2.1: Configurations of the environment setup



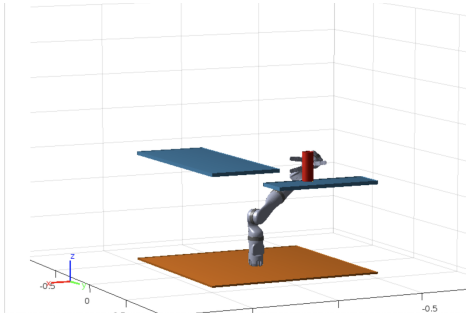
(a) Starting position of the red cylinder (bird's eye view)



(b) Target position of the red cylinder (bird's eye view)



(c) Starting position of the red cylinder (front view)



(d) Target position of the red cylinder (front view)

Figure 2.4: The pick-and-place platform from different perspectives

Chapter 3

Joint Detection and Classification

In this chapter, two joint detection and matching approaches, SIFT (Scale Invariant Feature Transform) and SURF (Speeded-Up Robust Features) are introduced in detail in the first section. The process of joint classification with CNN (Convolutional Neural Network) is illustrated in section 3.2. All the algorithms are validated by using sample image data of a multi-robot-arm platform through Matlab, and the results of validation as well as the implementation of SIFT and SURF on the pick-and-place platform are presented in the last section, with comparisons and discussions.

3.1 SIFT and SURF-based Joint Detection

There exist many approaches for feature detection and matching in the image processing area. In this work, we focus on the SIFT and SURF methods, which are known to be invariant to image transformations. Other methods have also been studied and applied extensively, for corner and edge detection, etc. For example, the Harris Corner Detector, it is commonly used to extract corners and infer features of an image, similar to the FAST (Features from Accelerated Segment Test) method [53]. BRIEF (Binary Robust Independent Elementary Features) approach is very fast both in building and matching features and easily outperforms other fast descriptors such as SURF and SIFT. In this method, image patches need to be converted to binary feature vectors, and each keypoint is described by a feature vector which is a

128-dimensional vector [24]. The ORB (Oriented FAST and Rotated BRIEF) method builds on the well-known FAST keypoint detector and the BRIEF descriptor. Both techniques are attractive because of their good performance and low cost, but they are generally more computationally expensive.

3.1.1 SIFT

The SIFT algorithm detects interest points in a scale-invariant way, as extrema in the response of the convolution of the image with a difference of Gaussians (DoG) function [54]. It is applied for joint detection and matching by the following main steps which are shown in Figure 3.1:

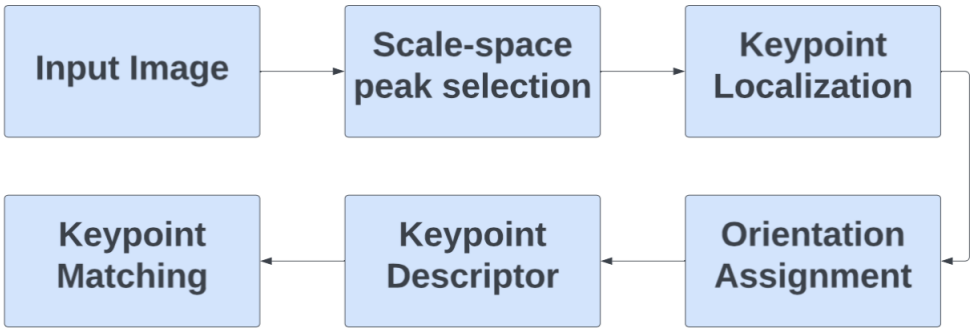


Figure 3.1: Main steps of the SIFT

1. Scale-space peak selection

   (a) Scale-space

   The scale-space of the image is a function $L(x, y, \sigma)$ (σ is the scaling parameter) that is produced from the convolution of a Gaussian kernel at different scales with the input image. Scale spaces are usually implemented as image pyramids. Scale-space is separated into octaves and each octave has 3 layers. Several octaves of the original image are generated, and the image size of each octave is half of the previous one. As most parts of a robotic arm are in the same color, it is difficult to use color as a differentiation.

(b) Blurring

Blurring refers to the convolution of the Gaussian operator and the image. This particular expression is applied to each pixel. It is written as:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y), \tag{3.1}$$

$$G(x, y, \sigma) = \frac{1}{2\Pi\sigma^2} \exp\left\{-\frac{(x^2 + y^2)}{2\sigma^2}\right\}, \tag{3.2}$$

where G is the Gaussian blur operator and I represents an image. σ is the scale parameter and x and y are the location coordinates. The result is a blurred image.

(c) DoG (difference of Gaussians)

Blurred images are also used to generate another set of images, the DoG, which is the difference of Gaussian blurring of an image with two different scale parameters, i.e. σ and $k\sigma$. This process needs to repeat for every octave of the image in the Gaussian pyramid.

(d) Finding keypoints

Once the scale spaces are generated and used to calculate the DoGs, the DoGs are used to calculate the Laplacian of Gaussian approximations that are scale invariant. Every pixel in the image is compared with its 8 neighbors and 9 pixels in the previous scale and the next scale. A total of 26 checks are made for each one. If it is a local extrema, then it is a potential keypoint, the best one represented in that scale. Keypoints like joints and gripper are found in this step.

2. Keypoint localization

Intensities are checked for low contrast features, as some of the keypoints generated may not have enough contrast, or they all lie along the edge. Taylor series expansion is applied to scale space to get a more accurate location of extrema.

If the intensity at an extrema is less than a predetermined threshold value, it will be rejected.

3. Orientation assignment

   The legitimate keypoints are obtained from the previous step and the scale at which keypoint was detected, which is the same as the scale of the blurred image. Therefore, the scale invariance is achieved. In order to make each keypoint rotation invariant, an orientation to each keypoint is assigned.

   A neighborhood is taken around the keypoint location depending on the scale, and the gradient magnitude and direction are calculated in that region. An orientation histogram with 36 bins covering 360 degrees is created. The amount added to the bin is proportional to the magnitude of the gradient at that point. After this has been done for all the pixels around the keypoint, the histogram has a peak.

   The highest peak in the histogram is taken and any peak above 80% is also considered for calculating the orientation. It creates keypoints with the same location and scale in different directions.

4. Keypoint descriptor

   A descriptor is used to compute the local image region about each joint that is highly distinctive and invariant to variations, e.g. changes in viewpoint and illumination.

   A 16×16 window around the keypoint is taken. It is divided into 16 sub-blocks of 4×4 size. For each sub-block, an 8-bin orientation histogram is created, so 4×4 descriptors over 16×16 sample arrays are used. Totally $4 \times 4 \times 8$ directions correspond to 128 bin values. It is represented as a feature vector to form the keypoint descriptor.

5. Keypoint matching

   Finally, joints are extracted and saved as separate images. They are matched

with the corresponding original input images by identifying the nearest neighbors.

3.1.2 SURF

Although both SIFT and SURF can detect and extract features, SURF mainly differs from SIFT in the following aspects:

1. It is based on Haar wavelets instead of derivative approximations in an image pyramid;

2. The interest points constitute approximations of scale-space extrema of the determinant of Hessian matrix instead of Laplacian operator;

3. Entries in the feature vector are computed as sums and absolute sums of the first-order derivative, instead of histograms of coarsely quantized gradient directions.

The detailed process of SURF presented in Figure 3.2 is illustrated below:

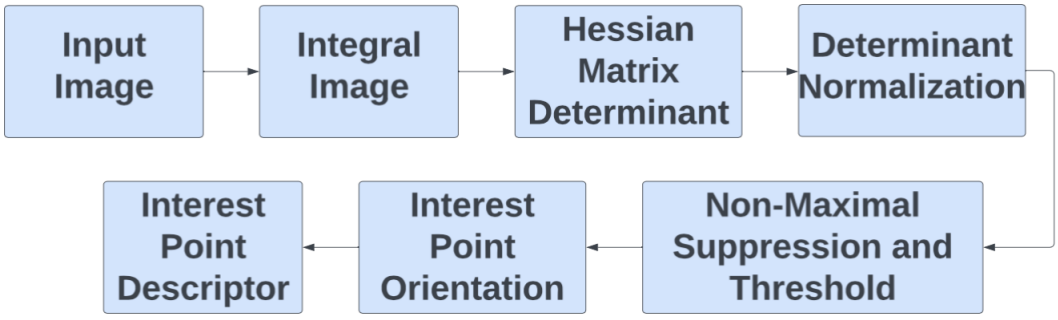


Figure 3.2: SURF Algorithm

1. Integral image

   The integral image is used as a quick and effective way of calculating the sum of pixel values and the average intensity within the given image. The entry of

an integral image $I_\Sigma(\boldsymbol{x})$ at location $\boldsymbol{x} = (x, y)^T$ represents the sum of all pixels in the input image I within the rectangular region formed by the origin and $\boldsymbol{x}$.

$$I_\Sigma(\boldsymbol{x}) = \sum_{i=0}^{i<=x} \sum_{j=0}^{j<=y} I(i, j) \tag{3.3}$$

2. Hessian matrix determinant and normalization

   SURF uses the Hessian matrix because of its good performance in computation time and accuracy. It relies on the determinant of the Hessian matrix for selecting the location and the scale. In order to adapt to any scale, the Gaussian kernel is used to filter the image. Given a point $\boldsymbol{x} = (x, y)$, Hessian matrix $H(\boldsymbol{x}, \sigma)$ in $\boldsymbol{x}$ at scale σ is defined as:

$$H(\boldsymbol{x}, \sigma) = \begin{bmatrix} L_{xx}(\boldsymbol{x}, \sigma) & L_{xy}(\boldsymbol{x}, \sigma) \\ L_{xy}(\boldsymbol{x}, \sigma) & L_{yy}(\boldsymbol{x}, \sigma) \end{bmatrix}, \tag{3.4}$$

   where $L_{xx}(\boldsymbol{x}, \sigma)$ is the convolution of the Gaussian second order derivative with image I in point $\boldsymbol{x}$, similarly for other representations.

3. Non-maximum suppression

   Non-maximum suppression is a technique used to eliminate duplicate detections and select the most relevant detected objects. The images are repeatedly smoothed with a Gaussian and subsequently sub-sampled. Due to the use of box filters and integral images, SURF applies such filters of any size at exactly the same sliding rate directly on the original image. In order to localize interest points in the image and over scales, a non-maximum suppression in a $3 \times 3 \times 3$ neighborhood is conducted. It helps reduce false positives and the computational complexity.

4. SURF descriptor

   The creation of the SURF descriptor takes place in two steps.

   (a) Interest point orientation

   In order to be invariant to rotation, SURF first calculates the Haar-wavelet

responses in x and y directions, and in a circular neighborhood of radius 6s around the keypoint, with $s = 4$, the scale at which the keypoint was detected. Also, the sampling step is scale-dependent and chosen to be s, and the wavelet responses are computed at that current scale s.

Calculate the sum of vertical and horizontal wavelet responses in a scanning area, then change the scanning orientation by adding $\pi/3$, and re-calculate, until the orientation with the largest sum value is found. This orientation is the main orientation of the feature descriptor.

(b) Interest point descriptor

In order to extract the descriptor, the first step consists of constructing a square region centered around the keypoint and oriented along the orientation acquired above. The size of this window is $20s$. Then the region is split up regularly into smaller 4×4 square sub-regions. For each sub-region, a few simple features are computed at 5×5 regularly spaced sample points. To increase the robustness towards geometric deformations and localization errors, the Haar-wavelet response in the horizontal and vertical directions (with filter size $2s$), dx and dy are first weighted with a Gaussian ($\sigma = 3.3s$) centered at the keypoint.

Then, the wavelet responses are summed up over each sub-region and form a first set of entries to the feature vector. In order to bring in information about the polarity of the intensity changes, the sum of the absolute values of the responses is extracted, $|dx|$ and $|dy|$. Hence, each sub-region has a four-dimensional descriptor vector v for its underlying intensity structure $V = (\Sigma dx, \Sigma dy, \Sigma|dx|, \Sigma|dy|)$. This results in a descriptor vector for all 4 $\times$ 4 sub-regions of length 64.

Images with a number of the strongest detected features that are in their corresponding circular neighborhoods are generated by SURF. The features are extracted and

matched with the target images, and located in the right position of the original images.

3.2 CNN-based Joint Classification

After the features are obtained from the SIFT and SURF, which are a set of images of the gripper and joints from different angles at different time stamps, CNN is used for classification of the features since it is robust to translation, rotation, and scaling invariance, and its ability to handle large amounts of data and achieve high accuracy. The images of extracted features, such as the images of joints shown on the left side of Figures 3.9, 3.11 and 3.13, are the input used for the training process. After the classification, they become the input for the filtering.

The images are divided into training and validation data sets, so that each category in the training set contains 750 images, and the validation set contains the remaining images. 30% of the training data are set aside to be used as validation data. Each image is $270 \times 270 \times 3$ pixels and there are 5 classes. The input to the convolution layer is 2D images, which are data with four dimensions corresponding to pixels in two spatial dimensions, the channels and the observations. The filter size is set to 3. The maximum number of epochs is set to be 5 for implementation and other settings are kept as default. The accuracy is the percentage of correctly predicted labels. In this case, more than 97% of the prediction match the true ones of the validation set.

Matching is completed by calculating the differences and making comparisons of the images between the data sets. As a result, images of each joint and gripper from different angles are classified and labelled for joints from top to bottom of the arm and the gripper respectively. This brings convenience for prediction with the Kalman Filter. If one joint is detected and classified, its coordinates can be easily acquired and then the poses of other joints can be determined by the robot kinematics and inverse kinematics, which speeds up the prediction step.

3.3 Results, Comparisons and Discussion

The joints of the robotic arm are the best choices for testing the joint detection and classification algorithms since they are revolute and rotational as the connections of the links of the arm. In this section, three algorithms are implemented for handling joint detection and classification.

3.3.1 Algorithm Validation

1. Joint detection with SIFT

   Simulation has been conducted using SIFT: 200 strongest features are extracted for the image of multiple robotic arms, which is extracted from [55].

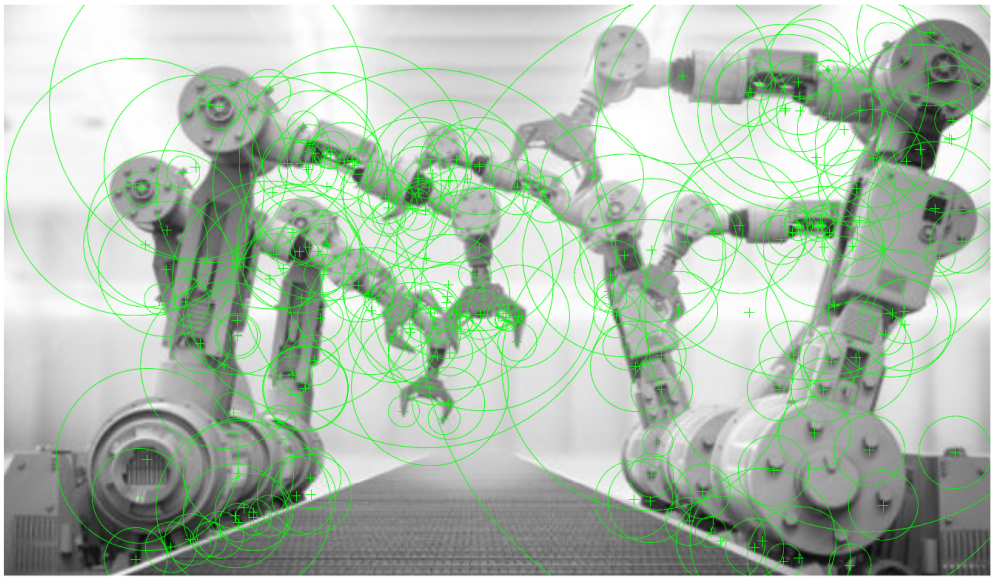


Figure 3.3: 200 Strongest features of the image retreived from [55] are extracted with SIFT

2. Joint detection with SURF

   This simulation is performed for detecting a specific object based on finding point correspondences between the reference and the target image. The reference image is the image of a joint, and the target image is the image of multiple

robotic arms.

The two imported images are transformed into gray scales at first. After that, 50 strongest feature points in the joint image and 200 strongest feature points in the arm image are detected respectively as shown in Figure 3.4.

Figure 3.4: 50 strongest feature points of joint and 200 strongest feature points of the image retreived from [55] are extracted with SURF

Then using SURF, the extracted features are matched between two images connected in yellow lines in Figure 3.5 and the joint is located in the right position of the target image, which is the red box shown in Figure 3.6.

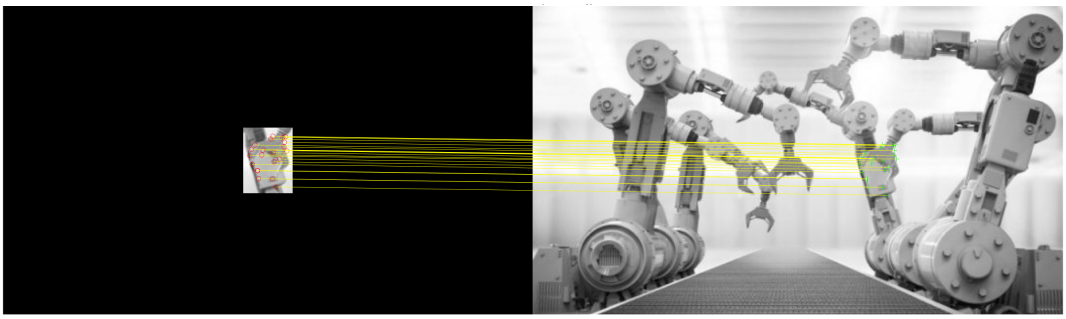


Figure 3.5: Validation result on SURF: features are matched between the image of 50 strongest feature points of joint and the image of 200 strongest feature points of the image retreived from [55]

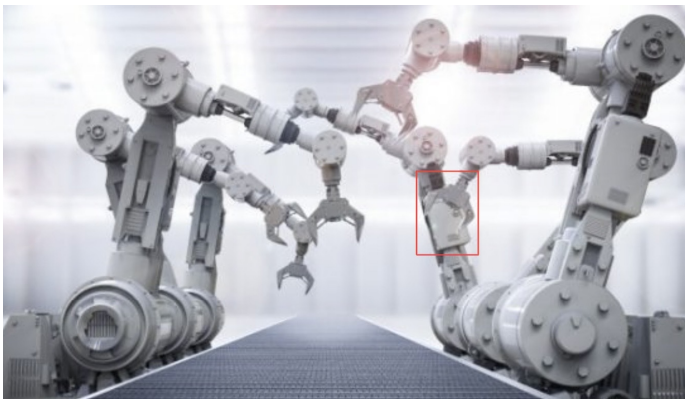


Figure 3.6: Joint location validation

Keypoints between two images are matched by identifying their nearest neighbors in SIFT, but in some cases, the second closest match may be very near to the first. It may happen due to noise or some other reasons. In that case, the ratio of closest distance to second closest distance is taken. If it is greater than 0.8, they are rejected. It eliminates around 90% of false matches while discards 5% correct matches.

For SURF, Gaussians are optimal for scale-space analysis, but in practice, they have to be discretized and cropped. This will lead to a loss in repeatability under image rotations around odd multiples of $\pi/4$. This weakness holds for Hessian-based detectors in general. The detectors perform well, and the slight decrease in performance does not outweigh the advantages of fast convolutions brought by the discretization and cropping. In SIFT, the descriptor is the 128 dimensional vector.

While both algorithms can provide satisfactory results, and are comparable in performance, SURF sometimes works faster than SIFT. This method of object detection works best for objects that exhibit non-repeating texture patterns, which give rise to unique feature matches. This technique is not likely to work well for uniformly-colored objects, or for objects containing repeating patterns.

In all, SURF is better than SIFT in rotation invariance, blurring and warp transform. SIFT is better than SURF for images in different scales.

3. Joint classification with CNN

   CNN has been applied for classifying the joints of the robotic arm with the validation data sets, which consists the images of the joints from different perspectives. The validation is performed by a pretrained CNN via Matlab.

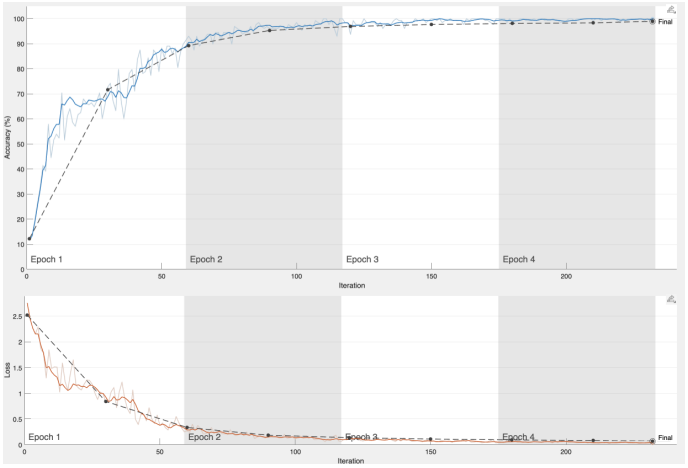


Figure 3.7: CNN Training Progress (validation for training and testing): (i) The upper diagram shows the accuracy of the training progress in percentage. Smoothed training is represented in solid blue line. Training is represented in solid line in light blue. (ii) The lower diagram shows the loss of the training progress. Smoothed training is represented in solid red line. Training is represented in solid line in light red. (iii) Validation is represented in dotted black line for both accuracy and loss.

As shown in the figure, the accuracy starts from 10% to 20%, then goes to around 70% after roughly 30 iterations, while the diagram of loss shows the consistency by starting at around 2.7 then decreases to around 0.8 after about 30 iterations. There are also fluctuations in the first epoch for both accuracy and loss, which suggest the process of learning in the first 50 iterations, then the lines become smooth until the training process ends. The validation accuracy is 98%, and max epoches are reached. For each epoch, the frequency of validation is 30.

3.3.2 Algorithm Implementation

After the SIFT, SURF and CNN algorithms for joint detection and classification are validated, they are applied to the pick-and-place platform. The workflow of joint detection and classification is shown in the figure below.

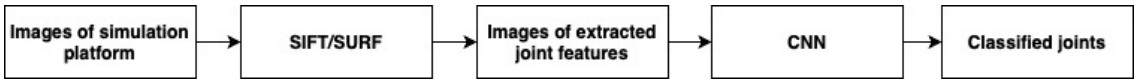


Figure 3.8: Workflow of Joint Detection and Classification Implementation

Videos have been recorded from various angles while the robotic arm conducting the pick-and-place task. Frames (images) are captured in each 0.1 seconds for every video as the original input for determining the points of interest. The images of the robotic arm conducting simulations with the platform consisting of one red cylinder and one green cylinder are the input for the implementation. They are obtained from the frames captured from the simulation videos. SURF is applied to generate extracted features, which can be used as the input for classification. The results of SURF implementation are shown in Figures 3.9, 3.11 and 3.13 with the images of joints on the left side. Each joint is extracted and the three strongest features are detected and matched to the right positions of the robotic arm. Figures 3.10, 3.12 and 3.14 provide corresponding satisfactory results for matching different joints between the extracted and the original input images. Each of the joints is located in the right position with respect to the various stages of the robotic arm conducting the pick-and-place task.

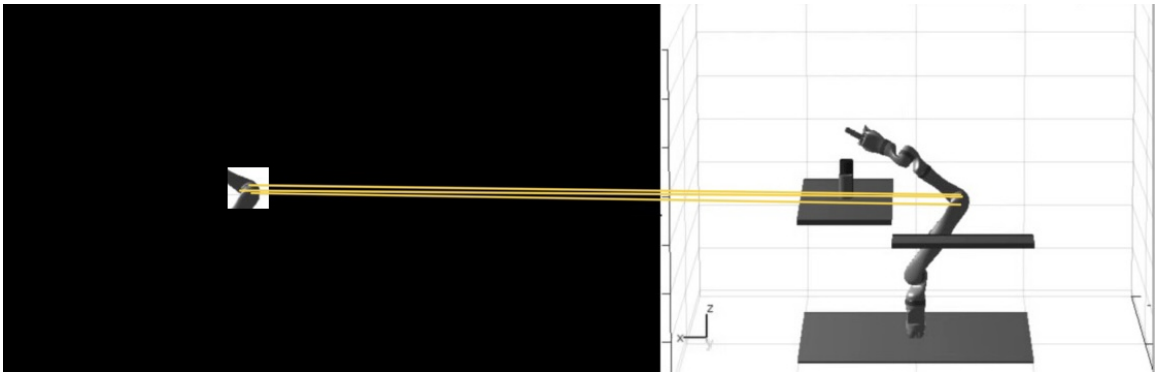


Figure 3.9: The pick-and-place benchmark model: matching features between the image of a joint and the image of the simulation platform - 1 (implementation)

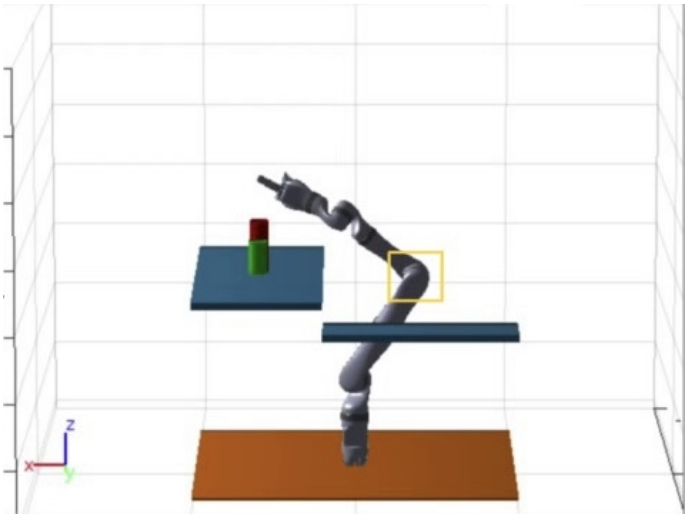


Figure 3.10: The pick-and-place benchmark model: locating joint in the corresponding position in the image of the simulation platform - 1 (implementation)

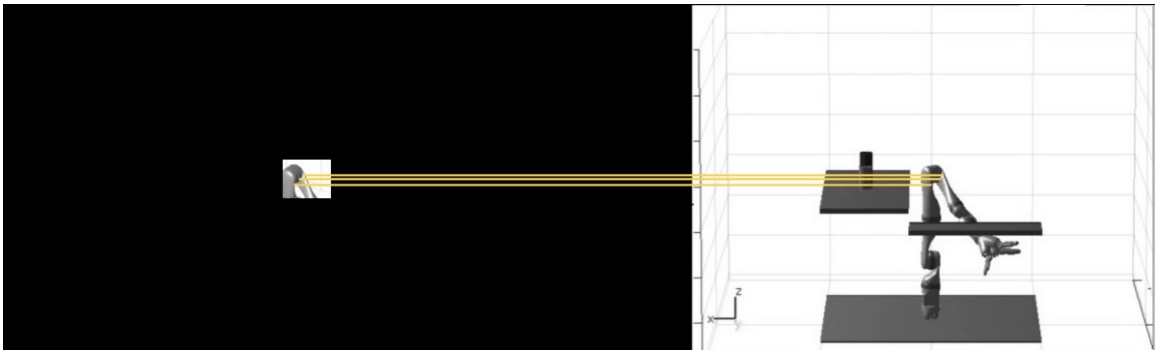


Figure 3.11: The pick-and-place benchmark model: matching features between the image of a joint and the image of the simulation platform - 2 (implementation)

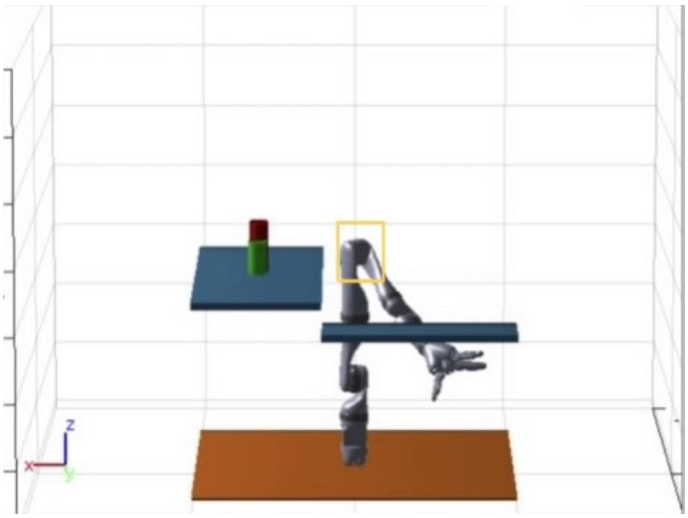


Figure 3.12: The pick-and-place benchmark model: locating joint in the corresponding position in the image of the simulation platform - 2 (implementation)

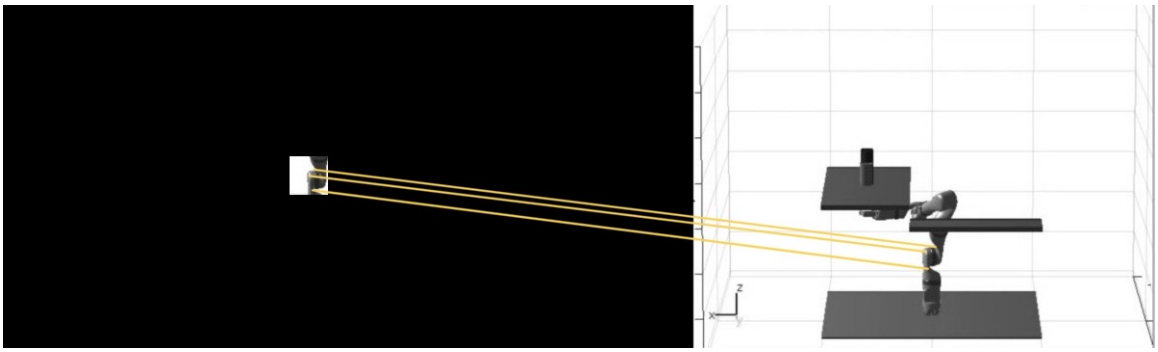


Figure 3.13: The pick-and-place benchmark model: matching features between the image of a joint and the image of the simulation platform - 3 (implementation)

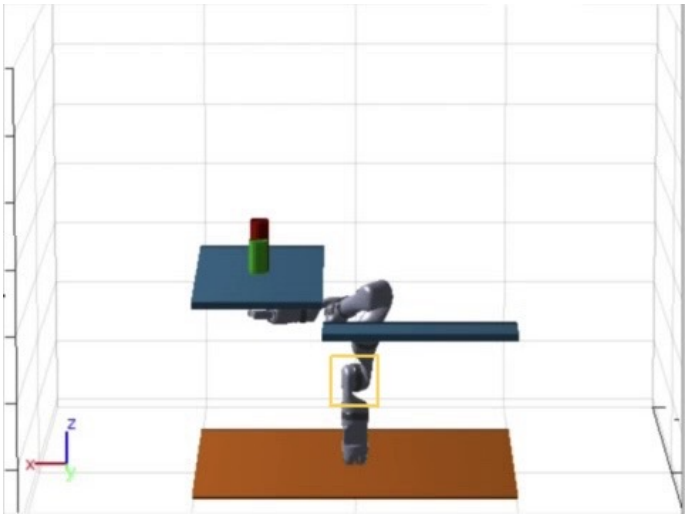


Figure 3.14: The pick-and-place benchmark model: locating joint in the corresponding position in the image of the simulation platform - 3 (implementation)

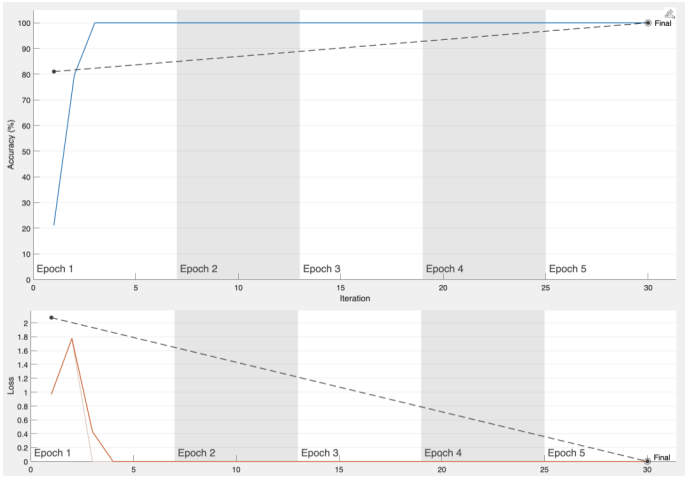


Figure 3.15: CNN Training Progress (implementation): (i) The upper diagram shows the accuracy of the training progress in percentage. Smoothed training is represented in a solid blue line. Training is represented in a solid line in light blue. (ii) The lower diagram shows the loss of the training progress. Smoothed training is represented in a solid red line. Training is represented in a solid line in light red. (iii) Validation is represented in a dotted black line for accuracy and loss.

As shown in Figure 3.15, the accuracy starts from a bit over 20% at first, then goes to 80% after 2 iterations, and increases to almost 100% after about 3 iterations. The diagram of loss shows consistency with the diagram of accuracy as well. It starts at around 1 then goes to 1.8 and quickly decreases to around 0.4 after about 3 iterations. Then the lines become smooth until the training process ends. The small number of iterations it takes to complete the classification and the time it spends to reach this high accuracy suggest the reliability and effectiveness of SIFT and SURF. The validation accuracy is 99%, and max epochs are reached. For each epoch, 6 iterations are completed.

Chapter 4

Trajectory Tracking and Motion Planning

After the features of joints have been extracted and classified, they can be applied for tracking the trajectory of the robotic arm with the help of many available tracking algorithms, including feature-based, filter-based, optimization-based, and more recently Reinforcement Learning (RL) approaches, etc. In this work, we mainly focus on the filter-based approach. Furthermore, we attempt to incorporate reinforcement learning in filter-based motion planning and tracking.

In the first section, the Extended Kalman Filter (EKF) and Particle Filter (PF) are designed and implemented for joint tracking and tested on the simulated pick-and-place benchmark model (shown in Figure 3.10) introduced in Chapter 2. The robotic arm used for joint tracking is a 6DoF (Degrees of Freedom) Kinova Jaco Assistive Robotics Arm. Its components and structure are introduced in Figure 2.3 and Figure 2.4 respectively. The initial trajectory of the pick-and-place is determined by the rapidly-exploring random tree (RRT)[56][57]. The whole process is recorded in video format with a length of 100 seconds, which will be used for pose estimation and motion tracking. The process of motion planning and tracking with EKF and RL is presented in the second section, along with the simulation results in a more complicated context.

4.1 Filter-based Trajectory Tracking

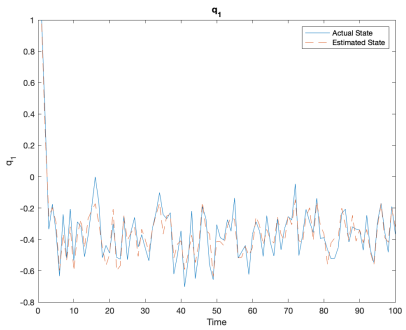
4.1.1 Extended Kalman Filter

With videos as the initial input, frames are captured every 0.1 seconds. The image data are loaded and read as pixels for joint detection with SURF and classification with CNN. The results are the input for prediction with Extended Kalman Filter (EKF). The information regarding the dynamics and kinematics of the robotic arm is applied by loading the setup built in the Matlab simulation platform.

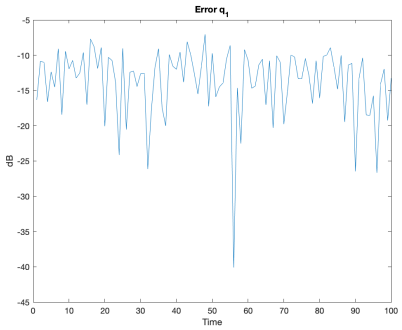
The results of EKF implementations are shown in Figure 4.1. The first column of figures shows the results for state estimation, while the second column of figures shows the results for the corresponding errors of states.

For the first column, the actual state is plotted in blue line and the estimated state is in dotted red. The value of q ranges from -0.8 to 1. The value of $\dot{q}_1$ ranges from -0.7 to 0.1. The value of $\dot{q}_2$ ranges from -0.1 to 0.8. For the second column, the range of q_1 estimation error is from -45dB to -5dB, the the range of $\dot{q}_1$ estimation error is from -30dB to -5dB, while the range of q_2 and $\dot{q}_2$ estimation error is from -35dB to -5dB.

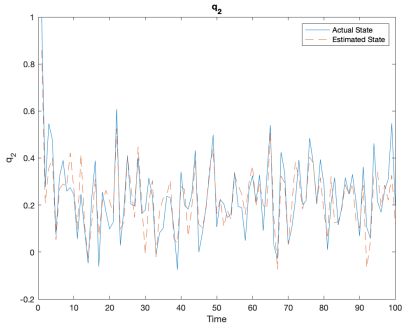
The fluctuations for estimation error suggest consistency to the actual movement with respect to the estimations for q_1 from time 55s to 60s and for q_2 from time 50s to 80s, while the directions of them moving the red cylinder have changed during these periods of time. The results of the simulations are satisfactory.



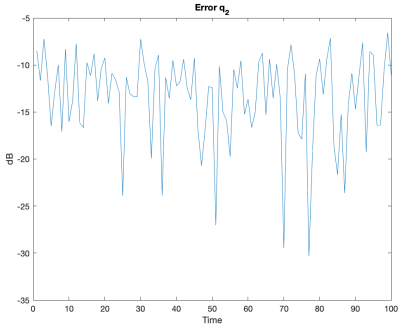
(a) q_1 estimation



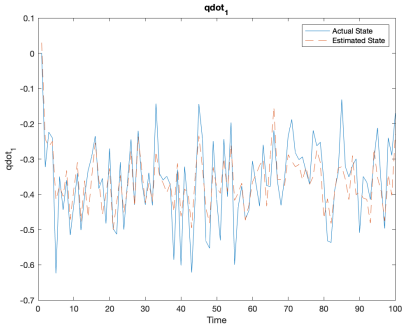
(b) q_1 estimation error



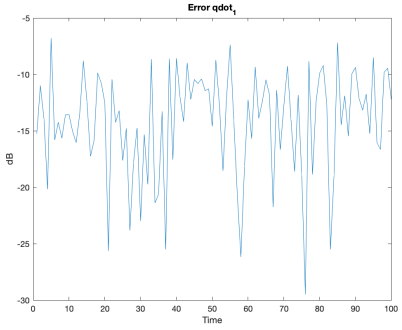
(c) q_2 estimation



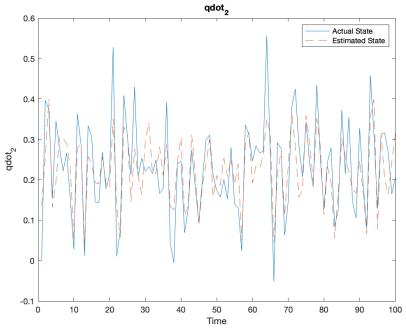
(d) q_2 estimation error



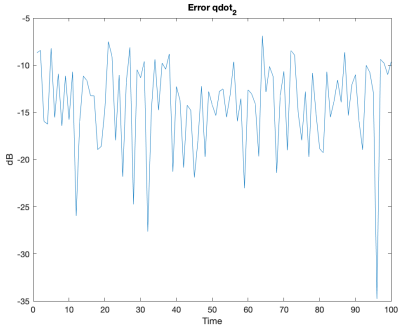
(e) $\dot{q}_1$ estimation



(f) $\dot{q}_1$ estimation error



(g) $\dot{q}_2$ estimation



(h) $\dot{q}_2$ estimation error

Figure 4.1: Results of EKF simulations

4.1.2 Particle Filter

The particle filter (PF) algorithm first constructs a temporary particle set X_k. This collection of particles represents the belief. It does this by taking each particle in the input set and further approximating them to the posterior distribution. The algorithm then generates a hypothetical state $x_k^{(i)}$ for time k based on particle $x_{k-1}^{(i)}$ and given input, and the weight is assigned to the corresponding belief.

The algorithm draws with replacement N particles from the temporary set $\bar{X}_t$. The probability of drawing each particle is given by its importance weight. Re-sampling transforms a particle set of N particles into another particle set of the same size. After a particle is drawn, it is propagated according to the transition model. Each propagated particle is verified by a weight assignment using the likelihood model. Generally, the algorithm consists of the following steps:

1. Initialize particles concentrated around the given initial joint state in the joint configuration space.

2. For each time step:

   (a) Use the joint torques and propagate the particles forward in time with a dynamic model that has Monte Carlo-like variations in parameters for each particle (resembling mutation in a genetic algorithm).

   (b) Calculate the forward kinematics of each particle and compare the result with the measured end effector pose.

   (c) Use the results from (d) to assign new probability weights to each particle.

   (d) Resample and normalize particle weights as needed.

The number of particles in our simulation of the pick-and-place platform with PF is 300 and the range of velocity is 0.5. For noise parameters, the variance of position and velocity are set to be 1.0 and 0.5 respectively. The random seed value is zero by

default and the velocity feature for the particles is considered. The quality of each particle is measured by comparing its new value to the target value. The weights are given to particles and a noise feature is added to the particle weights, then the particles are resampled. The sample set is updated at every time instant incorporating new data and resampling the set of state samples $x_k^{(i)}$.

$$x_k^{(i)} = p(x_k|z_{1:k}, u_{1:k}) \tag{4.1}$$

$$p(x_k|z_{1:k}) = cp(z_k|x_k)p(x_k|z_{1:k-1}) \tag{4.2}$$

For sampling from the prediction distribution $p(x_k|z_{1:k-1})$, the weight is

$$w(x) = cp(z_k|x_k), \tag{4.3}$$

with

$$c = \frac{1}{p(z_k|z_{1:k-1})}. \tag{4.4}$$

PF simulations have been conducted in the same environment setup as EKF. The residuals for the state estimation with robotic arm joints are shown in Figure 4.2. The overlap of the green concentrated particles and the area circled in red along with the errors of three joints ranges from 0 to 0.1 from Figure 4.4 also suggests the results are satisfactory.

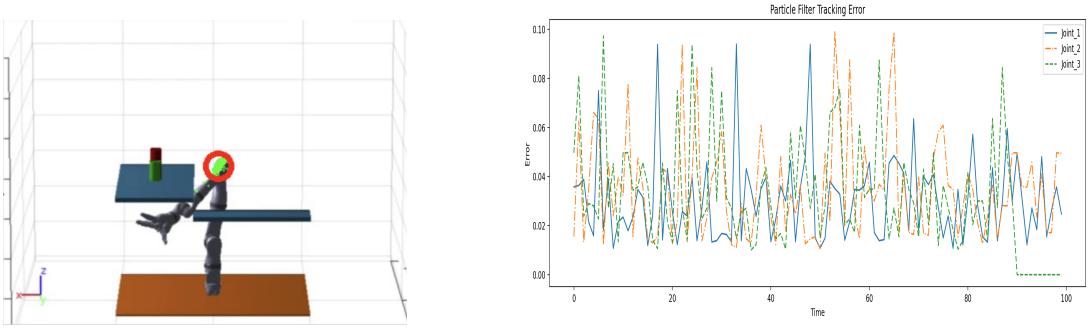


Figure 4.2: PF simulation with two cylinder cans: the concentrated particles overlap the area circled in red on the left, showing satisfactory results; the errors of three joints are plotted in blue, red, and green respectively on the right, the range of the errors is (0, 0.1)

4.1.3 Discussion

Unlike its linear counterpart, the EKF in general is not an optimal estimator. If the initial estimate of the state is wrong, or if the process is modelled incorrectly, the filter may quickly diverge, owing to its errors in the linear approximation. Another problem with EKF is that the estimated covariance matrix tends to underestimate the true covariance matrix.

The ability of the particle filter to represent multimodal belief states makes it well-suited to tackle the problem of estimating joint configurations, unlike tools such as the Kalman Filter, which requires beliefs to resemble Gaussian probability distributions. The filter represents arbitrary distributions non-parametrically with weighted clusters of particles. For Particle Filter is a recursive Bayesian state estimator that uses discrete particles to approximate the posterior distribution of an estimated state, it is useful for online state estimation when measurements and a system model that relates model states to the measurements are available.

If the system model is not accurate or not well-known, then Monte Carlo methods, especially Particle Filter, will be employed for estimation. Monte Carlo techniques predate the existence of the EKF but are more computationally expensive for any moderately dimensioned state space.

4.2 Reinforcement Learning-based Motion Tracking and Planning

For the pick-and-place task considered in this work, a state space is created to represent the configuration space (joint space) for motion planning. It samples feasible states for the robot arm. The grasp motion is planned using Rapidly-exploring random tree (RRT) with the customized state space and state validator objects. The start and goal configurations are specified by inverse kinematics for solving configurations based on the pose of the end effector.

The target position for the cylinder can on the other tabletop has been set and the found path is first smoothed through a recursive corner-cutting strategy. During the process of planning the move motion, the cylinder can level is kept at all time to avoid spill and an additional constraint is on the interim manipulator configurations. In this section, we consider a more complex scenario than the previous setup. A blue sphere would appear in the middle of the task, which interrupts the planned trajectory as the sphere blocks the original planned path of the robotic arm for placing the red cylinder can. To tackle this problem and achieve the obstacle avoidance, we have designed a scheme following the steps of the Reinforcement Learning (RL) algorithm SARSA (State-Action-Reward-State-Action).

Markov Decision Process

Since almost all RL problems can be formalized as Markov decision processes (MDPs), an environment represented by $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ is constructed with MDP. $\mathcal{S}$ is a finite set of states. The Markov state S_t, the coordinates of the joints, satisfies:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]. \tag{4.5}$$

$\mathcal{A}$ is a finite set of actions of the gripper moving horizontally (left or right) or vertically (up or down). Given action a the state transition probability $\mathcal{P}$ is defined by:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a], \tag{4.6}$$

where it defines the transition probabilities from all states s to all successor states s'. $\mathcal{R}$ is a reward function, which adds points if the distance between the gripper and the target has been reduced, or deducts points if the distance has not been reduced. No points will be assigned if the position of the gripper remains unchanged.

$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1}|S_t = s, A_t = a], \tag{4.7}$$

and γ is a discount factor, $\gamma \in [0, 1]$. The return G_t is the total discounted reward from time step t.

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.8}$$

The discount is the present value of future rewards. The value of receiving reward R after $k + 1$ time steps is $\gamma^k R$.

A policy π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s], \tag{4.9}$$

it fully defines the behaviour of an agent. The applied policy is ϵ-greedy. The state-value function $v_\pi(s)$ is the expected return starting from state s, and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]. \tag{4.10}$$

The action-value function $q_\pi(s, a)$ is the expected return starting from state s, taking action a and following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]. \tag{4.11}$$

4.2.1 Integration of Extended Kalman Filter and SARSA

With videos as the initial input, frames are captured every 0.1 seconds and each frame needs to go through the same workflow as shown in Figure 4.3.

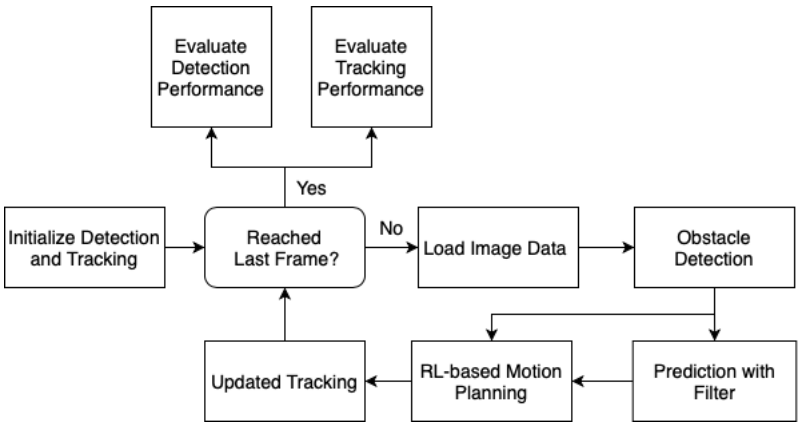


Figure 4.3: Workflow of RL-based Motion Tracking and Planning

The first step is to check if the last frame has been reached. If the last frame is reached, the performances of detection and tracking will be evaluated. As the process proceeds, if it is not the last frame, image data needs to be loaded and read. The data are the coordinates of the joints and their relative positions in the simulation platform. The coordinates are acquired by the pixel values of extracted features, which are the results of joint detection with SURF and classification with CNN. Through the transformation, the coordinates are obtained, and the possible locations of joints in the next moment are predicted. In addition, the data helps detect an unknown object when one appears. The results are the input for prediction with Extended Kalman Filter (EKF).

An agent is controlled by its policy (function), which takes the states (observations) as the input and generates the action the agent needs to take at the current step. The environment is the pick-and-place platform of a robotic arm with two cylinder cans where a blue sphere would appear in the middle of the task as an unexpected interference. The agent is the robotic arm and actions are moving vertically (up or down) or horizontally (right or left).

It is known that the RRT is usually performed to generate pre-determined trajectories at the beginning of a task. When there are unexpected changes in the environment, such as the appearance of an unexpected object interrupting the planned trajectory, it will be difficult to run RRT in real time to generate a new route as it is time-consuming. In our scheme, RRT is used once at the beginning as the initial condition is chosen based on the predetermined trajectory. When the blue sphere appears and is detected, Reinforcement Learning (RL) is implemented to find a new route to avoid the sphere object and complete the place task.

Workflow Illustration

The motion of the robotic arm is predicted. The system state equation for the dynamics of the joint pixels in the 2D frame can be expressed as:

$$x_k = A_k x_{k-1} + w_k. \tag{4.12}$$

x_k is the state variable of the system, and the gain matrix $A_k \in \mathbb{R}^{n\times n}$ is the system state transition matrix from the state of $k-1$ at the previous moment to the state of k at the current moment. The random variable w_k is the process noise with the Gaussian distribution.

The equation below defines the system observation variable as z_k, H_k is the observation matrix, and the random variable v_k is the observation noise, $v_k \sim \mathcal{N}(0, R)$, where R is the covariance matrix of v_k.

$$z_k = H_k x_k + v_k. \tag{4.13}$$

The workflow of the filter starts from the initial estimation of the covariance matrix and the initial estimation of state variables. If the state at the previous moment is x_{k-1} and the error covariance matrix associated with it is P_{k-1}, the prediction error covariance matrix is calculated first. $\mathbf{Q}$ is the covariance matrix.

$$\hat{x}_k^{\,-} = A_k \hat{x}_{k-1}, \tag{4.14}$$

$$P_k^- = A_k P_{k-1} A_k^T + \mathbf{Q}. \tag{4.15}$$

The Kalman gain can be calculated,

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R)^{-1}. \tag{4.16}$$

RL is applied in the updating step, which means that after the observations are obtained and the prediction step is finished, a reward is used for updating the policy. Based on the algorithm, a Q-value distance-based table has been built for calculating the maximum expected future rewards for action at each state. In each round, the first state and action of the robot are randomly initialized and execute action to get the reward points and the new state. After the robot takes action to obtain the reward, Sarsa(λ) updates the λ step before getting the reward. The value range of λ is [0, 1]. ϵ-greedy is applied to select action from the Q-value table based on the current state. The value functions $Q(s, a)$ are updated for the state and its corresponding action of the current sequence [51]. α is set to be 0.3 and discount factor γ is set to be 0.95. Combining the system observation value z_k with the predicted value $\hat{x}_k^-$, the posterior estimate can be calculated, which is the estimate of the optimal current state.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{4.17}$$

The error covariance of posterior estimates is updated by the equation:

$$P_k = (I - K_k H) P_k^- \tag{4.18}$$

After the policy is updated, it takes the current state as the input to issue new actions, which is the gripper moving vertically (up or down) or horizontally (right or left) to the next time step conducting prediction. The predicted coordinates of the joint location can be derived from the above formulas.

4.2.2 Results and Discussion

Simulations have been conducted for Extended Kalman Filter and Reinforcement Learning on the same platform as the one introduced in the previous section but in two scenarios: one with a blue sphere appears in the middle of the process, while the other one does not. The tracking results are shown by plotting the actual and estimated states and the mean squared error.

1. Simulation results with no blue sphere

   The tracking results with no blue sphere in the environment are shown in Figure 4.4 with the green dots representing the tracked points of the movement of joints on the left. On the right side, the errors of the three joints on the X, Y and Z axes are presented in different colors respectively. The range of errors for all the joints is between -0.04 and 0.04.

   The left side of Figure 4.5, shows the differences between pairs of coordinates; the mean squared error (MSE) of the simulation ranges from around 0.05 to 0.3 is shown on the right. The differences in the coordinates are larger compared to the other diagram of differences. The tracked points are rather large for the initial process of learning the new environment, but the results are better after the process, which is shown in the diagram of MSE.

   Figure 4.6 presents the points of tracking with two cylinder cans in the simulation environment. The blue dots are the points of prediction named Data 1, while Data 2 shown in the red squares are the results of the measurements.

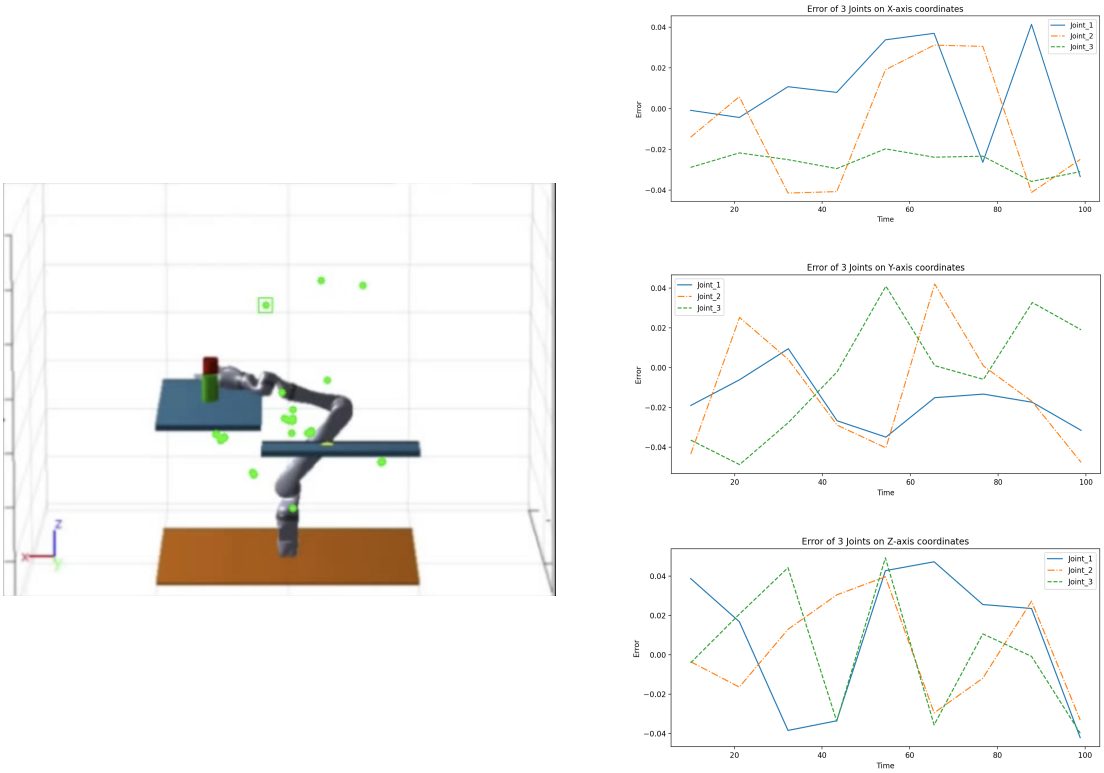


Figure 4.4: EKF and RL Tracking Performance (with no blue sphere): the figure on the left shows the points of tracked joints, which are in green; the errors of coordinates for each of the three joints on X, Y, and Z axes are shown in the three plots on the right in blue, red and green lines correspondingly.

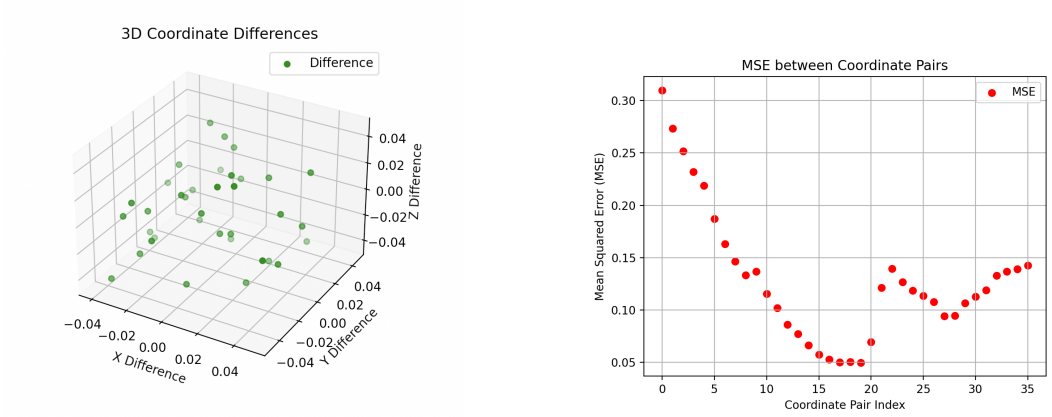


Figure 4.5: 3D coordinate differences and MSE between coordinate pairs (with no blue sphere)

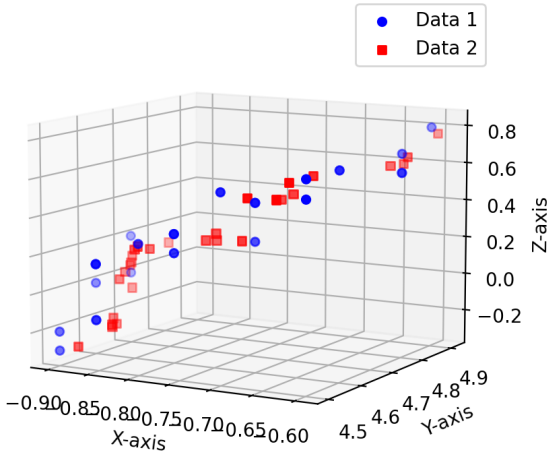


Figure 4.6: Tracked points with no blue sphere: Data 1 shown in blue dots represents the results of prediction; Data 2 shown in red squares represents the results of actual measurements

2. Simulation results with blue sphere

   Figure 4.7 presents the simulation platform with the blue sphere that blocks the planned trajectory of the robotic arm. The pose of the blue sphere is [0.15, 0.1, 0.7].

   The figure of the differences between pairs of coordinates is shown on the left side of Figure 4.8, while the MSE of the simulation is shown on the right side. Both diagrams suggest the results are satisfactory and justified by the small range of values of the differences and MSE.

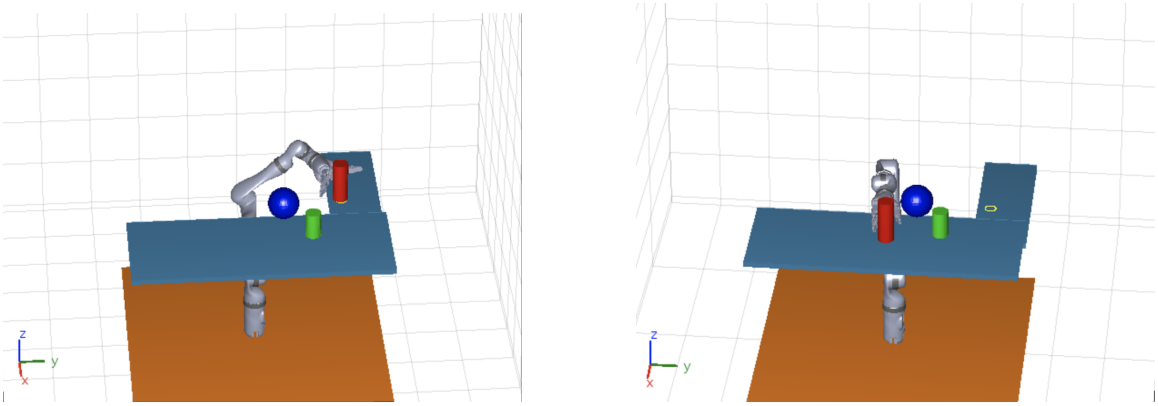


Figure 4.7: Simulation platform with blue sphere

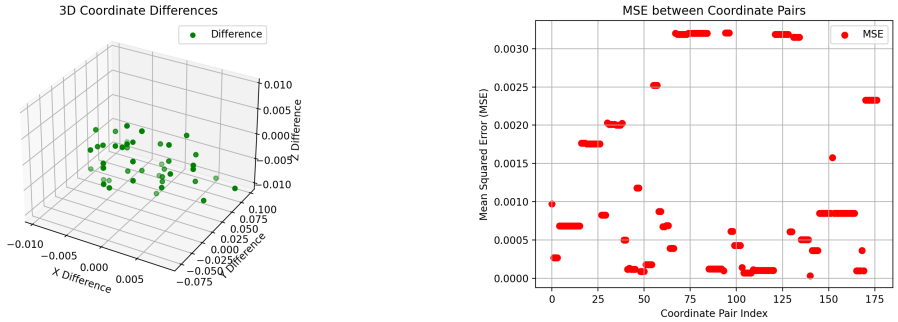


Figure 4.8: 3D coordinate differences and MSE between coordinate pairs (with a blue sphere)

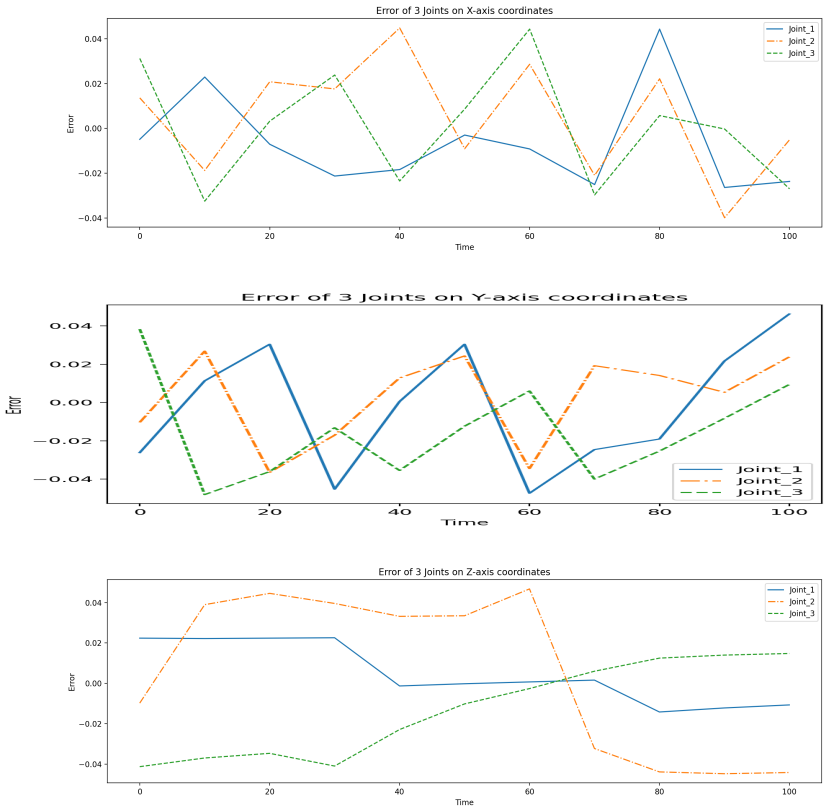


Figure 4.9: EKF and RL Tracking Performance (with a blue sphere): the errors of coordinates for each of the three joints on the X, Y, and Z axes are shown in blue, red and green lines respectively.

The errors of the three joints on the X, Y and Z axes are presented in different colors in Figure 4.9. The range of errors for all the joints is between -0.04 and 0.04.

In Figure 4.10, Data 1 represents the predicted positions, which are shown in blue dots, and Data 2 represents the actual positions, which are shown in red squares. As shown in the figure, the algorithm provides satisfactory and consistent tracking results.

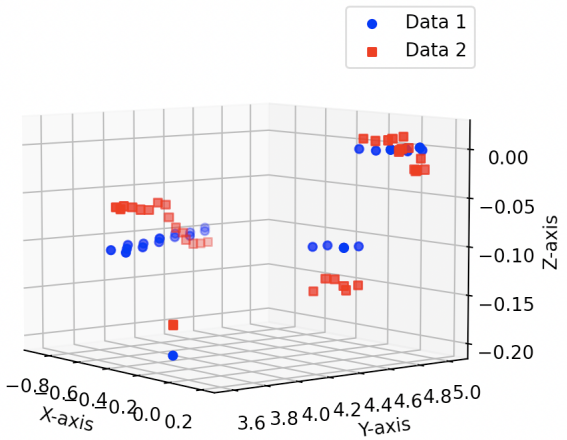


Figure 4.10: Tracked points with sphere: Data 1 shows in blue dots which represent the results of prediction; Data 2 shows in red squares which represent the results of actual measurements

3. Discussion

   From the implementations of EKF and RL, the simulation results are satisfactory. The range of MSE is much smaller compared to the ones without applying RL for simulations without the blue sphere. Because the green cylinder can be an obstacle in the initial setup, RL can easily learn its existence from the beginning. However, the blue sphere is set to appear in the environment in the middle of the process until the task has been finished, which interferes with the learning process and extends the time for making trajectory planning decisions. Although the interference of the newly added obstacle has brought challenges in searching for the best route, the results of the mean squared error and the differences between each pair of coordinates suggest the effectiveness of the combined application.

   The learning category of Sarsa is on-policy. The process of the algorithm is to update the values in the Q table continuously, and then judge what action to take in a certain state based on the new values. In state s, after the action is executed according to the policy of ϵ-greedy and reaches the next state s', the

method used to update the Q-value of (s, a) at this time still uses the policy of ϵ-greedy and takes $Q(s, a)$, each episode and every step of each episode performs the ϵ-greedy exploration [36].

Since the time step in each episode in the Sarsa algorithm adopts the strategy of ϵ-greedy, it cannot guarantee that the agent can search every position in the space in an episode, but it visits and records each position of the space through the continuous increase of the episode, so the algorithm converges slowly [51].

Chapter 5

Conclusion and Future Work

5.1 Conclusion

One of the contributions of this thesis is to provide a process for image-based pose estimation with a robotic arm as a research subject. This thesis has presented and experimented with several popular object detection and tracking algorithms, which provide sufficient information for readers to understand and implement for various purposes. Another contribution is that the thesis combined the Extended Kalman Filter and Reinforcement Learning for composing the state estimation and tracking in a motion planning problem.

In Chapter 3, feature extraction, matching, and classification of images are introduced and implemented with SIFT, SURF and CNN. SURF is more robust than SIFT, and CNN is tested to be reliable for classifying the joints of the robotic arm. After the features are extracted and classified, the coordinates of the points of interest can be used as input for filter-based algorithms to conduct tracking missions. As illustrated in Chapter 4, EKF and PF can be applied for tracking, and each of them has its advantages for different purposes respectively. With the motion planning platform of the robotic arm conducting pick and place of the red cylinder, Reinforcement Learning (RL) for state update has been explored instead of the update step for Extended Kalman Filter, and implemented by adding a new obstacle which interferes with the original planned trajectory.

5.2 Future Work

As many algorithms can detect and extract features, for this particular robotic arm is our primary research subject, all the methods introduced in the thesis are selected and implemented towards the best result for this specific object. In other words, by changing to other algorithms or objects, the results may not be as satisfactory as the presented ones.

Conducting feature detection and extraction with BRIEF or ORB and tracking the movement of collaborative robotic arms or robotic arms that have more degrees of freedom conducting the same task in the same introduced framework could bring more challenges and less satisfactory results, for the structures would be more complicated which could bring some problems to feature extraction and errors in matrix transformations calculations.

As images and videos are the input for our simulations with no sensors involved, their clarity and the precision of the extracted pixel values could influence the results as well. One could work towards exploring better methods for improving the readability of the images with more input images and videos from different perspectives, which could lead to better results in more complicated scenarios.

More implementations can be conducted with other combinations of algorithms for the motion planning problem. RL has been applied together only with the Extended Kalman Filter in the thesis, further implementations with other filters, like the Cubature Kalman Filter, can be explored. Extended Kalman Filter can be applied with other machine learning algorithms as well for tracking objects.

Bibliography

[1] T. Wen, J. Liu, B. Cai, and C. Roberts, "High-precision state estimator design for the state of gaussian linear systems based on deep neural network kalman filter," *IEEE Sensors Journal*, vol. 23, no. 24, pp. 31 337–31 344, 2023. DOI: 10.1109/JSEN.2023.3329491.

[2] R. Zhou, M. Tang, Z. Gong, and M. Hao, "Freetrack: Device-free human tracking with deep neural networks and particle filtering," *IEEE Systems Journal*, vol. 14, no. 2, pp. 2990–3000, 2020. DOI: 10.1109/JSYST.2019.2921554.

[3] M. Ariza-Sentís, S. Vélez, R. Martínez-Peña, H. Baja, and J. Valente, "Object detection and tracking in precision farming: A systematic review," *Computers and Electronics in Agriculture*, vol. 219, p. 108 757, 2024, ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2024.108757. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169924001480.

[4] A. Kumar, R. Vohra, R. Jain, M. Li, C. Gan, and D. K. Jain, "Correlation filter based single object tracking: A review," *Information Fusion*, vol. 112, p. 102 562, 2024, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2024.102562. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253524003403.

[5] J. Kaur and W. Singh, "A systematic review of object detection from images using deep learning," *Multimedia Tools and Applications*, vol. 83, no. 4, pp. 12 253–12 338, Jan. 2024.

[6] A. Elgammal, R. Duraiswami, D. Harwood, and L. Davis, "Background and foreground modeling using nonparametric kernel density estimation for visual surveillance," *Proceedings of the IEEE*, vol. 90, no. 7, pp. 1151–1163, 2002. DOI: 10.1109/JPROC.2002.801448.

[7] S. Tang, M. Andriluka, and B. Schiele, "Detection and tracking of occluded people," *International Journal of Computer Vision*, vol. 110, no. 1, pp. 58–69, Oct. 2014.

[8] M. Sadik, S. Moussa, W. El-Sayed, and Z. Fayed, "Vehicles detection and tracking in advanced automated driving systems: Limitations and challenges," *International Journal of Intelligent Computing and Information Sciences*, pp. 1–16, Jul. 2022. DOI: 10.21608/ijicis.2022.117646.1158.

[9] C. Zheng *et al.*, "Deep learning-based human pose estimation: A survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–37, 2023.

[10] J. Zhang and Y. Xiu, "Image stitching based on human visual system and sift algorithm," *The Visual Computer*, vol. 40, no. 1, pp. 427–439, 2024.

[11] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[12] X. LI, L. JIAO, Y. LIU, and C. MA, "A video stabilization method based on improved sift," *Computer and Modernization*, no. 06, p. 43, 2024.

[13] F. Guo, J. Yang, Y. Chen, and B. Yao, "Research on image detection and matching based on sift features," in *2018 3rd International Conference on Control and Robotics Engineering (ICCRE)*, 2018, pp. 130–134. DOI: 10.1109/ICCRE.2018.8376448.

[14] A. Watts and D. Harvey, "Robust and optimal alignment of high-dimensional data using maximum likelihood estimation through a random sample consensus framework," *International Journal of Data Science and Analytics*, Jan. 2024.

[15] M. Anshari, M. N. Almunawar, M. Masri, N. L. Fitriyani, and M. Syafrudin, "Autonomous vehicle safety through the sift method: A conceptual analysis," *Information*, vol. 15, no. 6, p. 357, 2024.

[16] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417, ISBN: 978-3-540-33833-8.

[17] U. Diaa, "A deep learning model to inspect image forgery on surf keypoints of slic segmented regions," *Engineering, Technology & Applied Science Research*, vol. 14, no. 1, pp. 12 549–12 555, 2024.

[18] M. S. Patel, N. M. Patel, and M. S. Holia, "Feature based multi-view image registration using surf," in *2015 International Symposium on Advanced Computing and Communication (ISACC)*, 2015, pp. 213–218. DOI: 10.1109/ISACC.2015.7377344.

[19] D. Zhou and D. Hu, "A robust object tracking algorithm based on surf," in *2013 International Conference on Wireless Communications and Signal Processing*, 2013, pp. 1–5. DOI: 10.1109/WCSP.2013.6677270.

[20] H. Li, T. Xu, J. Li, and L. Zhang, "Face recognition based on improved surf," in *2013 Third International Conference on Intelligent System Design and Engineering Applications*, 2013, pp. 755–758. DOI: 10.1109/ISDEA.2012.179.

[21] K. Verma, D Ghosh, and A. Kumar, "Visual tracking in unstabilized real time videos using SURF," *Journal of Ambient Intelligence and Humanized Computing*, vol. 15, no. 1, pp. 809–827, Jan. 2024.

[22] D. Awasthi and V. K. Srivastava, "Robust, imperceptible and optimized watermarking of dicom image using schur decomposition, lwt-dct-svd and its authentication using surf," *Multimedia Tools and Applications*, vol. 82, no. 11, pp. 16 555–16 589, 2023.

[23] F. Schweiger, G. Schroth, R. Huitl, Y. Latif, and E. Steinbach, "Speeded-up surf: Design of an efficient multiscale feature detector," in *2013 IEEE International Conference on Image Processing*, 2013, pp. 3475–3478. DOI: 10.1109/ICIP.2013.6738717.

[24] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792, ISBN: 978-3-642-15561-1.

[25] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: 10.1109/ICCV.2011.6126544.

[26] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443, ISBN: 978-3-540-33833-8.

[27] D. Phan, C.-M. Oh, S.-H. Kim, I.-S. Na, and C.-W. Lee, "Object recognition by combining binary local invariant features and color histogram," in *2013 2nd IAPR Asian Conference on Pattern Recognition*, 2013, pp. 466–470. DOI: 10.1109/ACPR.2013.103.

[28] X. Zhou, K. Wang, and J. Fu, "A method of sift simplifying and matching algorithm improvement," in *2016 International Conference on Industrial Informatics - Computing Technology, Intelligent Technology, Industrial Information Integration (ICIICII)*, 2016, pp. 73–77. DOI: 10.1109/ICIICII.2016.0029.

[29] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. PP, pp. 1–1, Jul. 2019. DOI: 10.1109/LRA.2019.2931199.

[30] P. Mirunalini, S. M. Jaisakthi, and R. Sujana, "Tracking of object in occluded and non-occluded environment using sift and kalman filter," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, 2017, pp. 1290–1295. DOI: 10.1109/TENCON.2017.8228056.

[31] P. R. Gunjal, B. R. Gunjal, H. A. Shinde, S. M. Vanam, and S. S. Aher, "Moving object tracking using kalman filter," in *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, 2018, pp. 544–547. DOI: 10.1109/ICACCT.2018.8529402.

[32] X. Cai, Y. Wu, S. Liu, H. Xie, and H. Sun, "Multi-object tracking using kalman filter and historical trajectory correction for surveillance videos," Jan. 2024. DOI: 10.21203/rs.3.rs-3849387/v1.

[33] F. Wael, "A comprehensive vehicle-detection-and-tracking technique for autonomous driving," *International Journal of Computing and Digital Systems*, vol. 9, no. 4, pp. 567–580, 2020.

[34] L. Xue, B. Ma, J. Liu, C. Mu, and D. C. Wunsch, "Extended kalman filter based resilient formation tracking control of multiple unmanned vehicles via game-theoretical reinforcement learning," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 3, pp. 2307–2318, 2023. DOI: 10.1109/TIV.2023.3237790.

[35] J. Škach and I. Punčochář, "Input design for fault detection using extended kalman filter and reinforcement learning," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7302–7307, 2017.

[36] A. Seekircher, S. Abeyruwan, and U. Visser, "Accurate ball tracking with extended kalman filters as a prerequisite for a high-level behavior with reinforcement learning," in *The 6th Workshop on Humanoid Soccer Robots at Humanoid Conference, Bled (Slovenia)*, 2011.

[37] L. Fu, Q. Zhang, and S. Tian, "Real-time video surveillance on highways using combination of extended kalman filter and deep reinforcement learning," *Heliyon*, vol. 10, no. 5, 2024.

[38] T. Omeragić and J. Velagić, "Tracking of moving objects based on extended kalman filter," in *2020 International Symposium ELMAR*, 2020, pp. 137–140. DOI: 10.1109/ELMAR49956.2020.9219021.

[39] T. Xiao, S. Li, B. Wang, L. Lin, and X. Wang, "Joint detection and identification feature learning for person search," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3376–3385, 2017. DOI: 10.1109/CVPR.2017.360.

[40] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653–1660. DOI: 10.1109/CVPR.2014.214.

[41] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 472–487, ISBN: 978-3-030-01231-1.

[42] H. Kieritz, W. Hübner, and M. Arens, "Joint detection and online multi-object tracking," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1540–15 408, 2018. DOI: 10.1109/CVPRW.2018.00195.

[43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, 84–90, 2017, ISSN: 0001-0782. DOI: 10.1145/3065386. [Online]. Available: https://doi.org/10.1145/3065386.

[44] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2012.

[45] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010. DOI: 10.1109/TNN.2010.2066286.

[46] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung, *Transferring rich feature hierarchies for robust visual tracking*, 2015. arXiv: 1501.04587 [cs.CV].

[47] Y. Liao *et al.*, "Feature matching and position matching between optical and sar with local deep feature descriptor," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 15, pp. 448–462, 2022. DOI: 10.1109/JSTARS.2021.3134676.

[48] W. Liu, S. Liao, W. Ren, W. Hu, and Y. Yu, "High-level semantic feature detection: A new perspective for pedestrian detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5187–5196.

[49] Z. Lu, G.-H. Liu, F. Lu, and B.-J. Zhang, "Image retrieval using dual-weighted deep feature descriptor," *International Journal of Machine Learning and Cybernetics*, vol. 14, no. 3, pp. 643–653, 2023.

[50] N. Ki and E. Delp, "New models for real-time tracking using particle filtering," vol. 7257, Jan. 2009. DOI: 10.1117/12.807311.

[51] Y. Zhang, Y. Hu, X. Hu, and B. Xing, "Path planning for mobile robot based on rgb-d slam and pedestrian trajectory prediction," in *2020 4th Annual International Conference on Data Science and Business Analytics (ICDSBA)*, 2020, pp. 341–346. DOI: 10.1109/ICDSBA51020.2020.00094.

[52] Kinova Assistive, *Jaco assistive robotic arm user guide (en)*, 2023. [Online]. Available: https://assistive.kinovarobotics.com/uploads/EN-UG-007-Jaco-Assistive-robot-user-guide-r06.1.pdf.

[53] D. Viswanathan, "Features from accelerated segment test (fast)," 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:17031649.

[54] J. Cruz-Mota, I. Bogdanova, B. Paquier, M. Bierlaire, and J.-P. Thiran, "Scale invariant feature transform on the sphere: Theory and applications," *International journal of computer vision*, vol. 98, pp. 217–241, 2012.

[55] MARKET PROSPECTS, *Smart manufacturing: Robotic arm vision now and tomorrow*, 2023. [Online]. Available: https://www.market-prospects.com/articles/robotic-arm-vision-now-and-tomorrow.

[56] D. Song, B. Zhao, and L. Tang, "A tracking algorithm based on sift and kalman filter," Aug. 2012. DOI: 10.2991/iccasm.2012.400.

[57] D. Berenson, S. S. Srinivasa, D. Ferguson, A. Collet, and J. J. Kuffner, "Manipulation planning with workspace goal regions," in *2009 IEEE International Conference on Robotics and Automation*, 2009, pp. 618–624. DOI: 10.1109/ROBOT.2009.5152401.

[58] T. Lee *et al.*, "Camera-to-robot pose estimation from a single image," May 2020, pp. 9426–9432. DOI: 10.1109/ICRA40945.2020.9196596.

[59] T. Lindeberg, "Feature detection with automatic scale selection," *International journal of computer vision*, vol. 30, pp. 79–116, 1998.

[60] G. Golluccio, G. Gillini, A. Marino, and G. Antonelli, "Robot dynamics identification: A reproducible comparison with experiments on the kinova jaco," *IEEE Robotics Automation Magazine*, vol. 28, no. 3, pp. 128–140, 2021. DOI: 10.1109/MRA.2020.3004149.