Towards Practical Uncalibrated Visual Servoing

by

Oscar Alejandro Ramirez

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

Master of Science

Department of Computing Science
University of Alberta

Abstract

When controlling dynamic systems such as robots a big challenge lies in defining how the desired actions will be accomplished. In industrial settings automation has been possible due to the structured and predictable environments. The repeatability of the tasks makes it viable to manually automate them. Moving to dynamic and unstructured environments, such as human settings and the outdoors, makes such approaches impractical.

Uncalibrated Visual Servoing (UVS) presents a viable approach to facilitate robot control and task definition in unstructured environments. UVS can be applied either in autonomous systems, or as a direct interface for users to manage the robot. Tasks are defined through visual features directly in image space. By estimating the full non-parametric image Jacobian no a-priori models or camera calibration is required. Real-world adoption of UVS has been slow despite over forty years of research.

This work presents a minimalistic framework and accompanying software library for UVS. The goal is to enable users to create a variety of visual servoing systems using low complexity control interfaces that easily interact with visual tracking systems to produce a complete environment able to drive robot control. Our library, ROS-UVS,

has now been used with several robots for a variety of tasks such as pick and place with a WAM manipulator and quadrotor assisted flight control. Our implementation, developed within the ROS framework, has proven to be flexible, robust and easy to use and integrate across multiple robots and control interfaces. A simulation environment is also available allowing users to try out our system. Finally, through the use of ROS-UVS we explore the performance of UVS both in simulation and with a physical robot. Characterizing the behaviour of UVS will help facilitate adoption for new users and serves to showcase the features and applications of our library.

Preface

This thesis is an original work by Oscar Alejandro Ramirez. Three referenced works in Chapter 1 of this thesis have been previously published. First the work presented in: "VIBI: Assistive Vision-Based Interface for Robot Manipulation" was performed in a collaboration with Kinova Robotics and published in the 2015 International Conference on Robotics and Automation. I was responsible for the robot control while Camilo Perez developed the visual detection of objects.

The work in "Small Object Manipulation in 3D Perception Robotic Systems Using Visual Servoing" was published at the 2014 International Conference on Intelligent Robots and Systems. Finally, research done with Mona Gridseth was published at the International Conference on Robotics and Automation in 2016. In this work, titled, "ViTa: Visual Task Specification Interface for Manipulation with Uncalibrated Visual Servoing" I developed the library used to perform the Uncalibrated Visual Servoing used by the visual interface developed by Mona Gridseth.

Acknowledgements

To my parents, and friends. I can not thank you all enough for the constant support and help you have all given me. I hope I can somehow pay it back to you as I am deeply in your debt.

I consider myself lucky to have had the opportuinty to work with an amazing group of people in our research group. Camilo, Mona, Diego, Sepehr, Masood and everyone else that I've shared the robotics lab with, thank you for putting up with my grumbling. We got the chance to share this adventure together, and it is not something I will soon forget.

I would also like to thank my supervisor, Martin Jagersand, for sharing so much knowledge with me and teaching me so much. This extends to all the faculty in the department. I have enjoyed my time here and I have learned much more than I could have imagined in topics ranging from machine learning, computer vision, robotics, and myself.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DOF	*Degrees of Freedom*
HRI	*Human Robot Interface*
IBVS	*Image Based Visual Servoing*
PBVS	*Position Based Visual Servoing*
ROS	*Robot Operating System [1]*
UVS	*Uncalibrated Visual Servoing*
VS	*Visual Servoing*

Chapter 1

Introduction

1.1 Motivation

As robots move towards domestic settings from industrial environments many challenges arise. Robot control is not an easy task. In current research a big push is taking place to try and automate systems within unstructured environments [2]. Understanding how to control robotic systems, in order to achieve generic, common place tasks, is still not a solved problem.

Figure 1.1: UofA's WAM robot at the Amazon Picking Challenge Seattle, 2015

Robots first thrived in industrial settings, where structured environments facilitate

the tasks at hand. Their repetitive nature makes programming specific motions, for a given task financially viable. This was first seen in the automotive industry for example. In [2] Kemp et al. identify key aspects that make robot interactions difficult. For instance the presence of humans along other autonomous actors such as pets or other robots taking action can interfere with the robot's task or change the environment. Further more the environment is usually built and matched to human bodies and capabilities and not those of robots. The range of challenges in these environments encompass everything from control, perception, and design to name a few.

A great new example in industry of the shift towards unstructured environments can be seen in the Amazon Picking Challenge [3]. Here contestants were expected to develop a robotic system capable of replacing humans performing semi-structured picking tasks. The goal being, to identify and relocate items from a shelf, into a plastic bin, in order to fulfill online orders. The shelf in this case had a variety of objects. Non rigid structures as well as objects with high specularity challenged visual and grasping systems alike. Object position or orientation was not predetermined; giving the challenge more unstructured characteristics. Our system is shown in Figure 1.1

The results of the Amazon Picking Challenge showcase how, even top researchers in the field, struggle to make robust systems for these kinds of tasks [4]. Robotic control in fully unstructured environments is still a challenging domain. However solutions that have the human-in-the-loop have been demonstrated to be reliable. In [5] Leeper et al. show how human-in-the loop robotic systems can be used to handle complex tasks in unstructured environments when performing grasping. Remote operation in assistive home robotic systems was also demonstrated in [6] and [7] where robots served as surrogates and helpers manually controlled remotely. Humans are also able to handle shifting levels of autonomy which makes gradual integration a possibility. Using a teleoperation interface in [8] Muszynski et al present users with the option to shift between different levels of autonomous control. With this in mind, the focus of our work is in trying to breach this gap between structured and unstructured environments for robots through the use of Uncalibrated Visual Servo-

ing (UVS) which we can use in both autonomous systems and through user defined tasks.

Visual Servoing approaches the control of a robot through the use of visual input. In the traditional implementation, camera images are used to define visual features and a visual task that the robot can then complete. Although Visual Servoing has been researched for over fourty years it has failed to see adoption in practice. Challenges include camera calibration, lack of, or difficulty integrating reliable real-time visual trackers and a lack of simple control interfaces to which robots can be connected.

1.2 Summary of Work

Before working through the details of Visual Servoing a summary of the work done is presented. These publications illustrate the road that motivated the development of ROS-UVS.

First, in collaboration with Kinova Robotics, we developed a system to assist users in the operation of Kinova's Jaco arm [9]. Developed for upper body disabled, Kinova's Jaco arm mounts directly on the user's wheelchair. Their disabilities accentuate the difficulties of tele-operating a robotic arm and our system was shown to facilitate the use of the robot.

A 3D vision system was used to ease the positioning of the robotic arm with respect to a target object as shown in Figure 1.2. Users need only click on the desired object on an image and the robot will perform the required motion. Our work, "VIBI: Assistive Vision-Based Interface for Robot Manipulation" [10] was published at ICRA 2015. Grasping however, was not fully automated. In our research we faced similar problems to the authors in [11] where a 3D depth sensor was used for coarse positioning, but manual control had to be used for grasping. The precision of the RGBD sensor, as well as the difficulties of precise calibration came into effect. Making the task challenging for general applications. Based on the feedback received by Kinova it is clear that users prefer systems that work reliably with manual intervention to a system that is

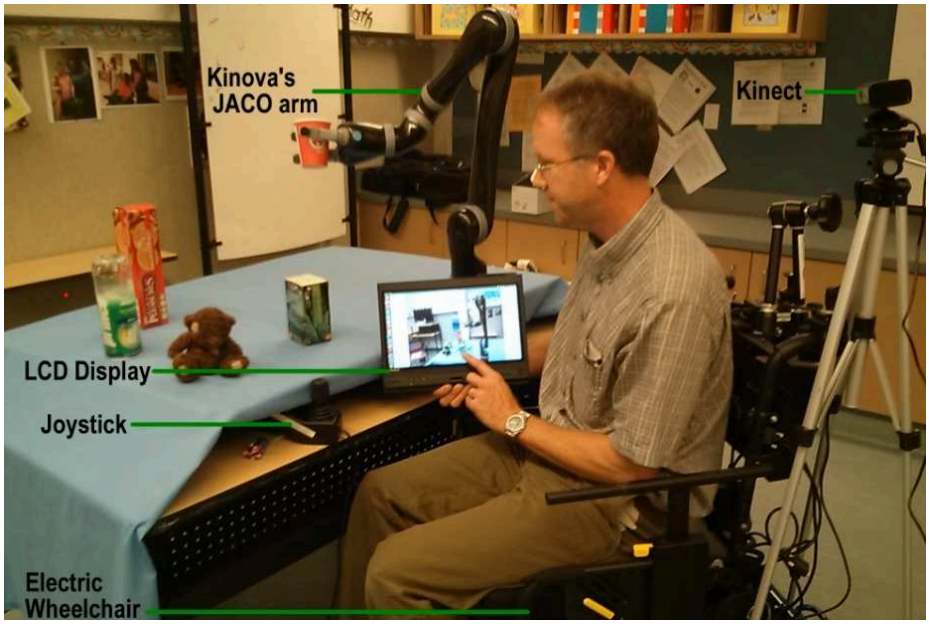


Figure 1.2: Vision-based User Interface to a 6DOF robot arm and hand. By pointing and selecting in a 2D video image of the scene, the user can point to objects, select grasp types and execute robot actions.

autonomous but not reliable.

Having faced the problems using an RGBD sensor, we explored in more detail the challenges that arise when performing small object manipulation. Our paper: *"Small Object Manipulation in 3D Perception Robotic Systems Using Visual Servoing"* [12], outlined five major factors we dealt with. The restrictions on detection range and resolution of affordable depth sensors limit what can be detected. End effector capabilities and sensor localization inhibit how objects can be manipulated. And finally calibration and robot control further affect precision and manipulation potential.

Uncalibrated Visual Servoing was presented as a way for users to address the challenges of calibrated sensors and control. Sample fine manipulation scenarios were then demonstrated such as threading a line through a fishing lure as is shown in Figure 1.3. Although a very challenging task to perform, even with Uncalibrated Visual Servoing, we were able to perform it in practice.

Finally, *ROS-UVS: A Minimalistic ROS Library for Visual Constraint Minimiza-*

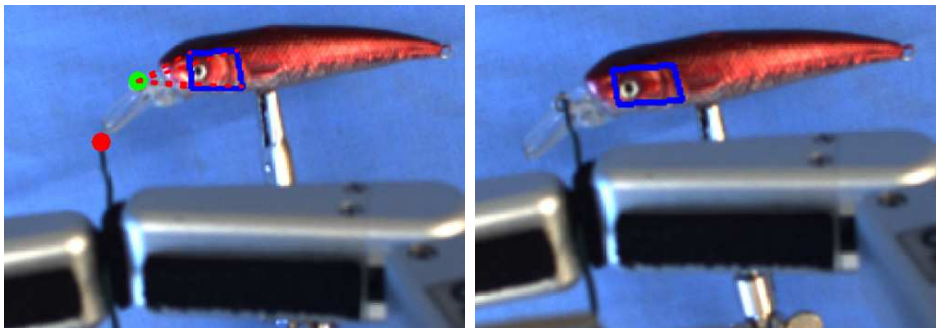


Figure 1.3: Eye in hand view as the fishing lure is threaded through the use of Visual Servoing. This showcases the ability to perform fine object manipulation with visual servoing

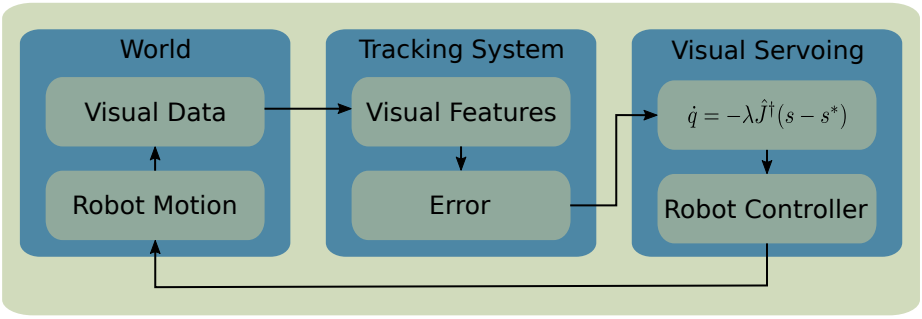


Figure 1.4: ROS-UVS System Overview: Given some visual data from the world the tracking system produces visual features. From the visual features an error is computed which is used by the visual servoing to feed the robot controller.

tion through Uncalibrated Visual Servoing was developed with the knowledge acquired. Focusing on the robot control aspect of the servoing, allows for great abstraction and division of labour making this library very flexible. As shown in Figure 1.4, the control loop for Visual Servoing consists in taking world visual data, tracking points of interest and generating a visual error which is then used to drive the robot to complete a task. In our work we focus on facilitating this interaction with a robot system and on the effects of the tracking and visual system on the robot servoing.

Using ROS-UVS to control a robot, "ViTa: Visual Task Specification Interface for Manipulation with Uncalibrated Visual Servoing" [13] was also built in our research group. This allowed us to explore the feasibility of Visual Servoing as a real world solution for robot control with a human-in-the-loop. At the same time, I was able to

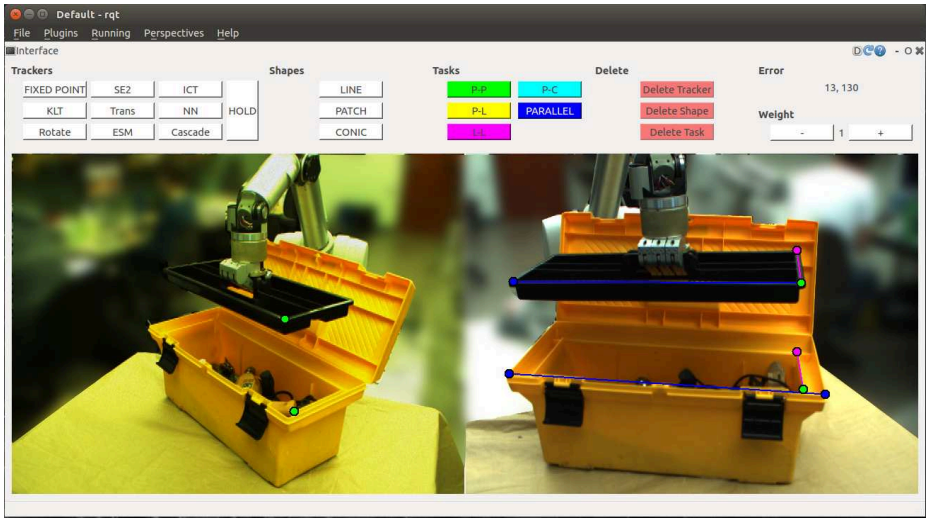


Figure 1.5: Using the Visual Task Specification Interface users are able to easily initialize trackers and define visual geometric constrains that help direct the robot to a given goal.

have a first test user of ROS-UVS which helped improve the usability of the system.

As seen in Figure 1.5 users are presented with views of the scene from one or more cameras. Using these a variety of trackers can be initialized. Through the trackers and visual features, tasks are defined using geometric visual constrains that form the error to be minimized.

1.3 Contributions

Visual Servoing (VS) has been researched for over fourty years, but real-world adoption has been slow. Challenges include camera calibration, lack of, or difficulty, integrating reliable real-time visual trackers and a lack of simple control interfaces through which robots can be controlled. Recent work has begun to show how VS can be utilized in practice to accomplish tasks through visual task specifications [14].

In this work we present a minimalistic framework for Uncalibrated Visual Servoing (UVS) with the hope that we can alleviate some of the issues that are holding back adoption. One of the main roadblocks is the time and effort required to develop the extensive software for video tracking and visual servo control required for real

time execution of tasks. Fortunately there are a few good options that allow fast prototyping of with Visual Servoing [15, 16]. Arguably one of the most complete libraries is the Visual Servoing platform (VISP) [16].

As pointed out by the VISP authors, approaches such as a learning process of the interaction matrix are not currently integrated within their library. In particular, an implementation of Uncalibrated Visual Servoing is missing.

In this work we focus on enabling users to create simple control interfaces that easily interact with tracking systems to produce a complete working environment. Our Visual Servoing library, ROS-UVS, has now been used with several kinds of robots for a variety of tasks such as pick and place with a WAM [17] manipulator and quadrotor assisted flight control. Our implementation, developed within the ROS framework, has proven to be flexible, robust and easy to use and integrate across multiple robots. A simulation environment is also available allowing users to quickly try out our system, both through ROS and Matlab.

Additionally, the convergence of visual servoing is studied. This is done in simulation as well as on our lab's WAM robotic manipulator. The effects of camera placement and types of control for the robot are studied with respect to the convergence and accuracy of the resulting motions. Readers of this work can also acquire some of the intuition as to how different control approaches affect robot motion, and how to best tune the available parameters facilitating adoption for new users.

1.4 Thesis Outline

Visual Servoing background and related work are reviewed in Chapter 2. Chapter 3 focuses on Task Specification and how the constrains for the servoing tasks are formally defined. Chapter 4 presents ROS-UVS, our Uncalibrated Visual Servoing library, along with the motivating design choices that were taken and an outline of the main features available. In Chapter 5 the convergence of Visual Servoing is explored, both in a simulation environment, and on a real robot, alongside the control strategies and how they affect convergence. Finally Chapter 6 presents the

conclusions and future work.

Chapter 2

Visual Servoing

The main objective of Visual Servoing is to minimize a visual error [18, 19, 16]. In this chapter we present the background of Visual Servoing, and its mathematical derivation. The two main types of Visual Servoing: Position Based Visual Servoing (PBVS) and Image Based Visual Servoing (IBVS) are presented. Finally we delve into Uncalibrated Visual Servoing which is a core component in this thesis.

2.1 Background

The term Visual Servoing first appeared in 1979. Introduced by Hill and Park in "*Real time control of a robot with a mobile camera*" [20] visual servoing distinguished itself from other vision approaches by offering a closed loop control system. The idea of using cameras to improve task accuracy through a visual feedback loop was presented before that by Shirai and Inoue [21]. The use of a closed loop control system versus a 'look' and 'move' system provided the turning point into Visual Servoing.

The task of Visual Servoing is to control or command a robot to interact with its environment using vision. Tasks are defined through the use of visual features which are obtained from one or more cameras. Jang et al. [22] provided a formal definition of visual features as image functionals. In practice any visual feature that can be unambiguously identified from multiple views of the environment can be used to define tasks. The feasibility of task definition and its decidability were then further investigated by Dodds et al. in [23]. Chapter 3 of this thesis delves deeper into the

definition of tasks and how to use them to perform actions with a robot.

After its initial appearance in the 1970s visual servoing saw only a small amount of progress. It wasn't until the 1990s that research and publications in the area saw a marked increase. In "A tutorial on visual servo control" [18] Hutchinson et al. attribute the increase in research in the area to the increase in computing power in personal computers. The ability to process image data at a quick enough rate was critical in the servoing of robots.

Although image processing began to happen at a fast enough rate the accuracy and robustness of visual trackers still required a lot of time to progress. Currently visual trackers are reaching a point where their accuracy and robustness have improved significantly. Tracking over significant periods of time and challenging visual conditions is now robust and accurate enough to use in unstructured environments. This presents an opportunity for another leap forward in the area.

2.1.1 Camera Configuration

Figure 2.1: Sample camera placement. **Left:** In the eye-in-hand configuration the camera is attached rigidly to the WAM. **Right:** The eye-to-hand camera, placed on a tripod, is to the side of the robot providing a complete scene overview.

There are two main configurations for the placement of cameras as shown in Figure 2.1. In the *eye-in-hand* configuration the camera is rigidly attached to the robot. This configuration allows the camera to be relocated and for it to be placed closer to the objects of interest as the robot moves. Having a closer view facilitates fine object manipulation. With a closer view there are more pixels covering a smaller area thus

giving visual systems a higher image resolution of the region of interest to work on. With this configuration motions have to be performed carefully as it is easy to lose view of the section of the scene we are interested in as the camera covers a smaller region of it.

When placing the camera in an *eye-to-hand* configuration both the robot and the scene will usually be visible. This view simplifies tasks such as object identification and localization as the camera acquires an overview of the complete scene. By placing the camera in a stationary position the robot has more freedom of motion as it can more easily move without causing the region of interest to leave the camera's field of view.

Considerations should also be taken when a human-in-the-loop is present. Driessen et al. [24] presented a collaborative controller for the Manus arm with an eye-in-hand camera configuration. The system was used to allow users to select objects and reach towards them. A notable result from their study was that an eye-in-hand camera does not provide a natural point of view for human interaction. Later in [25] Tsui et al. reported that in their prototype interface trials, fixed camera views outperformed a moving camera for object selection.

Each camera configuration has benefits that should be weighted when designing a vision system. The task at hand, along with the control method to be used should both influence the chosen camera setup. If a user will be mostly in charge, then perhaps an eye-to-hand configuration would be preferable to improve usability and make it easier for the user.

2.2 Visual Servoing Derivation

In this section we will work through the mathematical derivation of Visual Servoing. A great external reference for this topic by Chaumette et al., "Visual servo control. I. Basic approaches" [19], is also worth reviewing. Here we expand on their derivations and highlight details relevant to our interest in Uncalibrated Visual Servoing.

As stated above, the main objective of Visual Servoing is to minimize a visual

error which can be defined as follows:

$$\boldsymbol{e}(t) = \boldsymbol{s} - \boldsymbol{s}^* \tag{2.1}$$

Where $\boldsymbol{s}$ is a vector denoting some features, e.g. the position of some object, or image coordinates of tracked points, lines or other measures, and $\boldsymbol{s^*}$ is the desired goal configuration of these features. Suppose we are using an *eye-in-hand* configuration as seen in Figure 5.1. Then the spatial velocity of the camera is defined as $\mathbf{v}_c = (\boldsymbol{v}_c, \boldsymbol{\omega}_c)$, that is a linear and an angular velocity. The interaction between the camera's motion and the rate of change of the tracked features can then be linked:

$$\dot{\boldsymbol{s}} = \boldsymbol{L}_s \mathbf{v}_c \tag{2.2}$$

This gives rise to the term *interaction matrix*, $\boldsymbol{L}_s$, also known as the *feature Jacobian*. This matrix relates how the tracked features change with respect to the camera's linear and angular velocities. A similar formulation is then used to model the relationship of image error to camera motion:

$$\dot{\boldsymbol{e}} = \boldsymbol{L_e} \mathbf{v}_c \tag{2.3}$$

The only difference being that while $\boldsymbol{L_s}$ relates camera motion to tracked features, now $\boldsymbol{L_e}$ relates camera motion to the defined error. From equation (2.3) we can then find control velocities that will result in an exponential decrease of the error. This is usually referred to as the Visual Seroving control law as it is dictates the robot's motion.

$$\mathbf{v}_c = -\lambda \boldsymbol{L}_{\boldsymbol{e}}^{\dagger} \boldsymbol{e} \tag{2.4}$$

Where $\boldsymbol{L}_{\boldsymbol{e}}^{\dagger} \in \mathbb{R}^{d \times 2k}$ is the Moore-Penrose pseudoinverse of $\boldsymbol{L_e}$ such that $\boldsymbol{L}_{\boldsymbol{e}}^{\dagger} = (\boldsymbol{L}_{\boldsymbol{e}}^T \boldsymbol{L_e})^{-1} \boldsymbol{L}_e^T$. Here, d is the number of degrees of freedom (DOF) that are being controlled, and k the number of tracked features. Finally, λ is a parameter, or gain, used to control the final robot velocities as the minimization takes place. We also refer to λ as a step size as it determines how much the robot will move on each iteration. The use of the pseudoinverse is needed as the interaction matrix is not necessarily a square matrix.

2.3 Types of Visual Servoing

There are two main ways to approach Visual Servoing [18]. They mainly differ on how the tracked visual features are defined giving different interaction matrices. In Position Based Visual Servoing (PBVS) the tracked features are used to calculate 3D information about their location. To do this PBVS uses a calibrated camera and a 3D geometric model of the features of the object to reconstruct the relative pose of the camera with respect to the object. This means that for PBVS we required both a calibrated camera and a-priori models of the objects with which we will interact; making it inadequate for unstructured environments.

On the other hand, Image Based Visual Servoing (IBVS) directly utilizes image features to perform the robot control. In it's classical implementation however the analytic form of the image Jacobian is often used to drive the robot control. This requires intrinsic camera parameters which still impose a calibration requirement. Finally, Uncalibrated Visual Servoing (UVS) estimates the non-parametric image Jacobian through robot motions avoiding all calibration or model requirements [26, 27]. In the following sections we will look at how these features are defined, and how they affect the interaction matrices.

2.3.1 Position Based Visual Servoing (PBVS)

In Position Based Visual Servoing, 3D information is retrieved through the tracked feature points. If we assume an *eye-in-hand* configuration then the error can be defined to be:

$$s = ({}^{c^*}\boldsymbol{t}_c, \theta\boldsymbol{u}) \tag{2.5}$$

Where ${}^{c^*}\boldsymbol{t}_c$ and $\theta\boldsymbol{u}$ represent the translation and rotation of the current camera frame defined as $\mathcal{F}_c$ relative to the desired camera frame position $\mathcal{F}_{c^*}$. Setting the frame $\mathcal{F}_{c^*}$ as the reference frame for the system simplifies the derivation as we then induce $\boldsymbol{s}^* = 0$ and $\boldsymbol{e} = \boldsymbol{s}$. The interaction matrix for the error then decouples the translational and rotational motions:

$$\boldsymbol{L_e} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{L_{\theta u}} \end{bmatrix} \tag{2.6}$$

Where $\boldsymbol{L_{\theta u}}$ is defined as:

$$\boldsymbol{L_{\theta u}} = \boldsymbol{I}_3 - \frac{\theta}{2}[\boldsymbol{u}]_\times + \left(1 - \frac{sinc\,\theta}{sinc^2\,\frac{\theta}{2}}\right)[\boldsymbol{u}]_\times^2 \tag{2.7}$$

Here $sinc\ x$ is the sinus cardinal defined such that $x\ sinc\ x = sin\ x$ and $sinc\ 0 = 1$. This interaction matrix gives rise to a simple control scheme:

$$\begin{cases} \boldsymbol{v}_c = -\lambda \boldsymbol{R}^{Tc^*} \boldsymbol{t}_c \\ \boldsymbol{\omega}_c = -\lambda \theta \boldsymbol{u} \end{cases} \tag{2.8}$$

The final control law gives rise to some interesting motion consequences. With this formulation the camera's movement results in a straight line when the system is perfectly calibrated. However there are several complications that require special attention. A perfectly calibrated system is not realistic in practice. Users will also require a way to model the target objects for the servoing, and 3D information of the tracked features needs to be generated from image data which will again be highly dependant on the calibration.

2.3.2 Image Based Visual Servoing (IBVS)

Now we will cover the derivation of the *interaction matrix* for IBVS with two goals in mind. First to create a better understanding of the dynamics of the system. And most importantly, to motivate the use of the Uncalibrated Visual Servoing approach. A result of the derivation of the interaction matrix is that we require knowledge of the distance between the camera and the visual features. Since we are working with simple 2D camera images, we do not directly have this information.

To derive the interaction matrix we begin by modeling how image features behave. Image features are defined in image space, and thus are 2D point projections

with coordinates $\boldsymbol{x} = (x, y)$. Using a simple pinhole camera model we can find the projection to this point from 3D space by:

$$\begin{cases} x = X/Z = (u - c_u)/f\alpha \\ y = Y/Z = (v - c_v)/f\alpha \end{cases} \tag{2.9}$$

Where (u, v) are the pixel coordinates of the point in the image, c_u and c_v are the coordinates of the principal point, f the focal length, and finally α the ratio of the pixel dimensions. Taking the time derivative of equation (2.9) we get:

$$\begin{cases} \dot{x} = \dot{X}/Z - X\dot{Z}/Z^2 = (\dot{X} - x\dot{Z}/Z) \\ \dot{y} = \dot{Y}/Z - Y\dot{Z}/Z^2 = (\dot{Y} - y\dot{Z}/Z) \end{cases} \tag{2.10}$$

This describes the motion of the visual features in the image. Studying the derivation we see we need to know how the 3D point $\boldsymbol{X} = (X, Y, Z)$ is moving with respect to the image plane, i.e. how the camera is moving. Previously we defined the spatial velocity of the camera as $\mathbf{v}_c = (\boldsymbol{v}_c, \boldsymbol{\omega}_c)$. Expanding this we see how the 3D point moves with respect to the camera motion through:

$$\dot{\boldsymbol{X}} = -\boldsymbol{v}_c - \boldsymbol{\omega}_c \times \boldsymbol{X} \Leftrightarrow \begin{cases} \dot{X} = -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -v_y - \omega_z X + \omega_x Z \\ \dot{Z} = -v_z - \omega_x Y + \omega_y X \end{cases} \tag{2.11}$$

Finally combining (2.10) and (2.11) we get:

$$\begin{cases} \dot{x} = -v_x/Z + xv_z/Z + xy\omega_x - (1 + x^2)\omega_y + y\omega_z \\ \dot{y} = -v_y/Z + yv_z/Z + (1 + y^2)\omega_x - xy\omega_y - x\omega_z \end{cases} \tag{2.12}$$

Which we can re-write as:

$$\dot{\boldsymbol{x}} = \boldsymbol{L}_{\boldsymbol{x}} \mathbf{v}_c \tag{2.13}$$

This allows us to solve for the interaction matrix $\boldsymbol{L}_{\boldsymbol{x}}$

$$\boldsymbol{L_x} = \begin{bmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & x \end{bmatrix} \tag{2.14}$$

As a result of the analytical solution we see that in order to follow the control law we require to know the depth Z for every feature point, which is not directly known as we are only using 2D camera images. Having defined this background knowledge on visual servoing we can move on to Uncalibrated Visual Servoing and how it is implemented in order to avoid the required depth information.

2.3.3 Uncalibrated Visual Servoing in Practice

For a simple joint controller we can rewrite equation (2.4) as:

$$\dot{\boldsymbol{q}} = -\lambda \hat{J}^{\dagger} \boldsymbol{e} \tag{2.15}$$

Here, $\dot{\boldsymbol{q}}$ are joint velocities. This formulation has the advantage of requiring no knowledge of the camera placement. The interaction matrix is simply shown as J for the Jacobian:

$$J = \begin{bmatrix} \frac{\partial f_1(\boldsymbol{q})}{\partial q_1} & \cdots & \frac{\partial f_1(\boldsymbol{q})}{\partial q_m} \\ \vdots & & \vdots \\ \frac{\partial f_k(\boldsymbol{q})}{\partial q_1} & \cdots & \frac{\partial f_k(\boldsymbol{q})}{\partial q_m} \end{bmatrix} \tag{2.16}$$

Where $\boldsymbol{q}$ is a vector of joint angles, and $\boldsymbol{f}$ a vector containing image features. In calibrated systems this Jacobian is known a priori. Since our system is uncalibrated the Jacobian has to be determined before the error minimization can be done. To find an estimate of the Jacobian, $\hat{J}$, small exploratory motions [28] of each joint are performed and the error is observed to find a $\Delta \boldsymbol{e}$, that is, we will estimate the rate of change of the error with respect to the joint motions through finite differences. Using this change in the error vector, we estimate the partial derivative with respect to each joint:

$$\hat{J} = \begin{bmatrix} \begin{bmatrix} \vdots \\ \dfrac{\Delta \boldsymbol{e}_{q_1}}{\Delta q_1} \\ \vdots \end{bmatrix} & \cdots & \begin{bmatrix} \vdots \\ \dfrac{\Delta \boldsymbol{e}_{q_m}}{\Delta q_m} \\ \vdots \end{bmatrix} \end{bmatrix} \tag{2.17}$$

Here Δq_i is the scale of the motion of joint i performed to generate the error change $\Delta \boldsymbol{e}_{q_i}$. Having found the Jacobian we can then follow (2.15) to minimize the visual error. Finally in order to keep the Jacobian updated we can perform a Broyden update [27].

$$\hat{J}_{k+1} = \hat{J}_k + \alpha \frac{(\Delta \boldsymbol{e} - \hat{J}_k \Delta \boldsymbol{q}) \Delta \boldsymbol{q}^T}{\Delta \boldsymbol{q}^T \Delta \boldsymbol{q}} \tag{2.18}$$

Where α is a learning rate. After an initial Jacobian is estimated using (2.17) visual servoing proceeds using the control law, eq. (2.15). UVS differs from regular IBVS in that the Jacobian is continuously estimated using (2.18), instead of computed from an analytical expression of the camera geometry and robot kinematics. This method has been shown to improve performance and the robustness of the servoing to converge when used properly [27]. Since the Jacobian is only a local estimate for the motions of the system; as the robot changes pose the Jacobian will tend to deteriorate. Broyden's method allows us to update the Jacobian along with the robot's position.

Through the use of UVS we can then avoid the need for a-priori models and system calibration. These properties allow users to more easily place or move cameras around the robot's workspace. Eliminating the need for these steps makes it easier to adopt a visual system and requires less preparation before applying the visual control.

Chapter 3

Tasks Specification

In the context of Visual Servoing, task specification determines what the goal of the task at hand is. Through the specified task we can define an error. Using this error we then relate how robot motion changes the current world state with respect to the specified target. In the previous chapter we discussed how Visual Servoing is defined, and how a robot is driven through very simple control laws. In this chapter we will explore how the task specification is done in order to fully develop the control loop of Visual Servoing. Refering back to Figure 1.4 so far we have defined the panel on the right, Visual Servoing. Now we will see how a Tracking System can help us follow visual features that define the error to be minimized.

3.1 Visual Tracking

Visual tracking is the process of estimating the position of an object in images. Depending on the tracker, different aspects of an object's position, such as position and orientation, can be determined and followed through a series of camera images. Tracking algorithms will often offer different features. This leaves users with a choice where a trade off has to be made between complexity and information gathered. Figure 3.1 shows for instance a simple colour tracker known as Continuously Adaptive Mean Shift or CAMSHIFT [29]. This is a simple tracker that clusters points in a specific color space. It is simple to use and can process images quickly, however it mostly serves as a way to define a region of interest for a tracked object; it does not

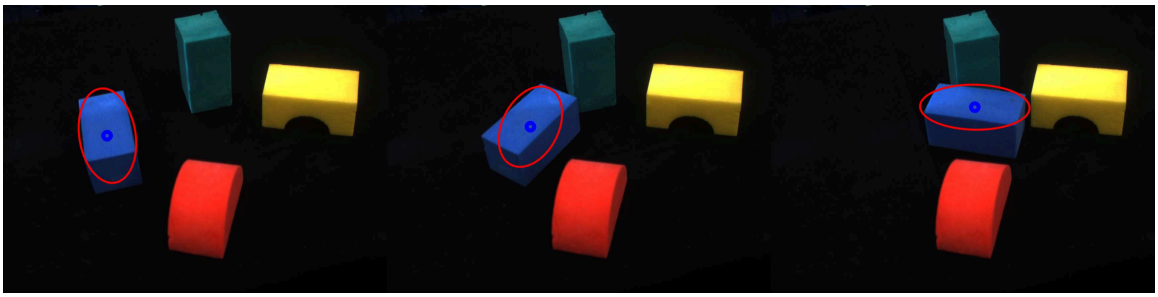


Figure 3.1: Sample visual tracker. Using OpenCV's CAMSHIFT tracker the color of the object is used to define a region of interest that can be followed as motion of the camera or of the object occurs.

provide a precise location for the tracked object.

As mentioned above there are many different types of visual trackers. For practical use alongside Visual Servoing trackers need to be robust, precise, and flexible. Since our goal with UVS is to allow simple control in unstructured environments, we need trackers that can perform under these conditions.

Tracking is still a big challenge in unstructured environments. Not only do the properties outlined above have to hold, but the tracker must also function in real time. For each image frame trackers must process the image, and output the position of the tracked region or object of interest. The time it takes to process an image limits the maximum effective camera frame rate we can use. In this case by effective camera frame rate we mean the rate at which we can fully process an image frame. If we can only process half the frames a camera is producing then, half the images are wasted.

There are two main advantages to performing the tracking quickly. In visual tracking it is usually assumed that the inter frame motion along with changes in intensity or light conditions will be small. In the case of robotics this can be helped by performing motions slowly. However, unstructured environments are dynamic and changes in lighting or the scene can affect tracker performance. By providing the tracking system with more frames at a higher rate rapid changes can be slowed down making the job of trackers easier as their base assumptions are held.

Having the ability to utilize higher frame rate cameras also facilitates the servoing

of the robot. As we use the results of the trackers to drive the motion of the robot, we need the processing time after an image is taken to be small. The commands that we generate through visual servoing are relative to the robot's position within the image processed. However while the image is being processed robot motion continues. Meaning that the longer it takes to process the image, the further away the robot will be from where the image was taken.

Recently, Roy et al. presented "Tracking Benchmark and Evaluation for Manipulation Tasks" [30]. This is a public dataset to evaluate trackers used for human and robot manipulation tasks. As specified by the authors, for these tasks both high DOF motion and high accuracy are needed. This dataset places a stringent convergence criteria of ± 1 pixel on the ground truths reported. In contrast previous benchmarks use a pixel threshold $t_p = 20$ pixels and low DOF motions [31]. This threshold is far too high for manipulation tasks. Given the state of the art in visual trackers, few sequences are fully tracked accurately, again highlighting the difficult task at hand. Similar to how computing power in personal computers gave researchers the ability to rapidly move the field of Visual Servoing forward, visual tracking performance reaching a new level of robustness and accuracy opens the door to new opportunities. Attention to benchmarks like this should be taken as visual trackers are the main tool that permit visual servoing tasks to be completed.

In addition to the challenge of tracking itself, actually acquiring trackers can also be non-trivial. Fortunately, some trackers are available in ROS. Implementations for trackers like Kalal's TLD [32], or more classical trackers like KLT [33] provided in ViSP [16] allow for real time tracking and are easy to integrate through ROS. Other tracking systems such as XVision [34] could also be integrated even if they are not available in ROS. Recently the Modular Tracking Framework was also released[1]. This library provides open-source implementations of many state-of-the-art registration based visual trackers that can be used within ROS through the `mtf_bridge` package also available at the library's website.

For our Visual Servoing application visual tracking is used to keep track of features

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

on objects of interest. As we will see in the next couple of sections, this will give us information about their motion and movement within the image. We will then use this information to define and complete tasks.

3.2 Visual Tasks

Visual Servoing with 2D cameras allows for higher accuracy than 3D depth sensors. Modern cameras are not only cheap, but also provide a higher resolution than equivalently priced 3D sensors. This can result in better accuracy even when performing full 3D depth triangulations with regular cameras as opposed to a Kinect sensor for instance [35]. Cameras also have the advantage of being able to capture images both from far away as well as at very close distances. Unlike depth sensors which normally have a defined range. Given the right tracker implementation sub-pixel accuracy can be given for visual features. Using this data the robot can perform precise movements.

Given these advantages for Image Based Visual Servoing, we are now left with the problem of how to properly use them to complete tasks. Chaumette et al. outline the classification and realization of vision based tasks in [36]. Then in 1999, Hespanha et. al. [37] addressed this problem by defining when it can be decided if a task has been completed.

We define a task through a *task function*:

$$T(\boldsymbol{f}) = 0 \tag{3.1}$$

Let the arm workspace be $\mathcal{W} \in SE(3)$, with a pair of stereo cameras that have a field of view $\mathcal{V}$. Typically $\mathcal{V}$ is either a subset of $\mathbb{R}^3$ or $\mathbb{P}^3$. Where $\mathbb{R}^m$ is the real linear space of m-dimensions, and $\mathbb{P}^m$ is the real projective space of one-dimensional subspaces of $\mathbb{R}^{m+1}$.

If we let $\boldsymbol{f}$ be the list of point features observed in the camera's field of view, then the task function is said to be *accomplished* if equation 3.1 holds on the basis of the observed image features. The list of features $\boldsymbol{f}$ is one of many possible lists of features in $\mathcal{F}$ which is called the *admissible features space*. This space is a mapping

from $\mathcal{F}$ into $\{0, 1\}$. The Value of 0 is given when a task is completed. For example, for a point to point task we define $\boldsymbol{f}$ to be:

$$\boldsymbol{f} \mapsto \begin{cases} 0 & \text{if } f_1 \text{ and } f_2 \text{ are the same point in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases} \tag{3.2}$$

This allows us to define very simple geometric constrains in order to specify tasks. Similar mappings can be performed for other simple tasks. This mapping however would assume that we have an accurate camera model. Since calibration error will always be present to some extent this approach is further extended to include an *encoding*, $E : \mathcal{Y}^T \to \mathbb{R}$. Where $\mathcal{Y}$ is the two camera joint image space.

This encoding is then used so that the task is accomplished when:

$$E(\boldsymbol{y}) = 0 \tag{3.3}$$

This is an extension that accounts for the error in the modeling of the visual system. Basically given a task equation 3.3 must hold when the task is completed. For the point to point case, using the difference between the visual feature coordinates suffices as it will only equal 0 when the points coincide.

3.2.1 Visual Task Specification

Now we will explore how to relate the mathematical definition of a task as shown above, with the image representation. The visually specified tasks provide two main functions. First they allow users to directly specify the task visually. Then, the task specification is used to verify whether the task was completed or not. There are some special cases where this can not be done. As shown by Dodds [23] cases where there is visual ambiguity in 2D images prevent us from always validating the task. A sample case is shown in Figure 3.2 where it appears the marker will be inserted into the cap even though it is not properly aligned.

In practice it is easiest to combine tasks in order to generate enough visual constraints so that a more complex action takes place. For example in Figure 3.3 a

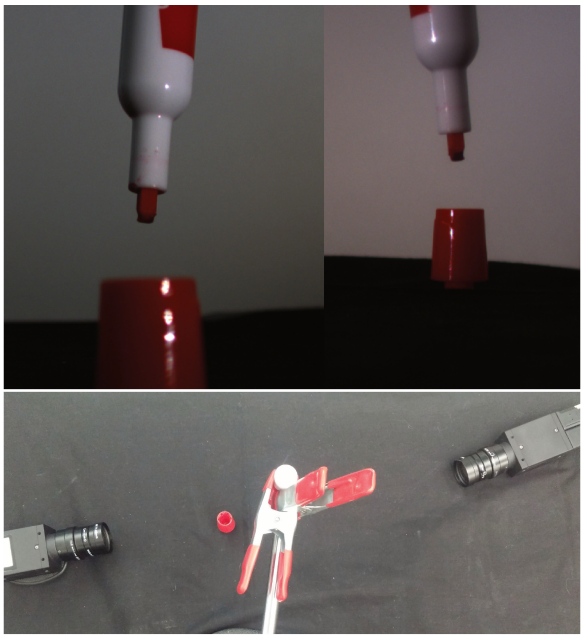


Figure 3.2: Given two camera views it appears as if the marker was being placed into the cap. The cap however is offset from the tip of the marker as can be seen from the overhead view.

combination of point to point tasks and one point to line task are used to the define the task that will grab the heat sink with the robot's hand. Depending on the activity different constrains can be used. In Figure 3.4, a parallel line constraint is used to align the memory card to the slot before placement. These figures also showcase our Visual Task Specification Interface [13]. By presenting users with an interface that resembles image editing software users are able to define geometric constrains and create visual task targets for visual servoing.

A classical example of task specification is shown in Figure 3.5. With this simple diagram we can better illustrate how to generate the proper error encoding for the task we want to accomplish. The goal is to align the wrench on the right to the bolt on the left. To do this we will use the visual features labeled $\boldsymbol{f}_1$ through $\boldsymbol{f}_8$. Suppose a two camera setup is in place. To identify features from each camera we will use $\boldsymbol{f}_{li}$ for features from the left camera, and $\boldsymbol{f}_{rj}$ for features from the right camera.

First to align the orientation of the wrench two point to line tasks are used. Using homogeneous coordinates, we let each feature $\boldsymbol{f} = [u, v, 1]^T$. Then, the line connecting

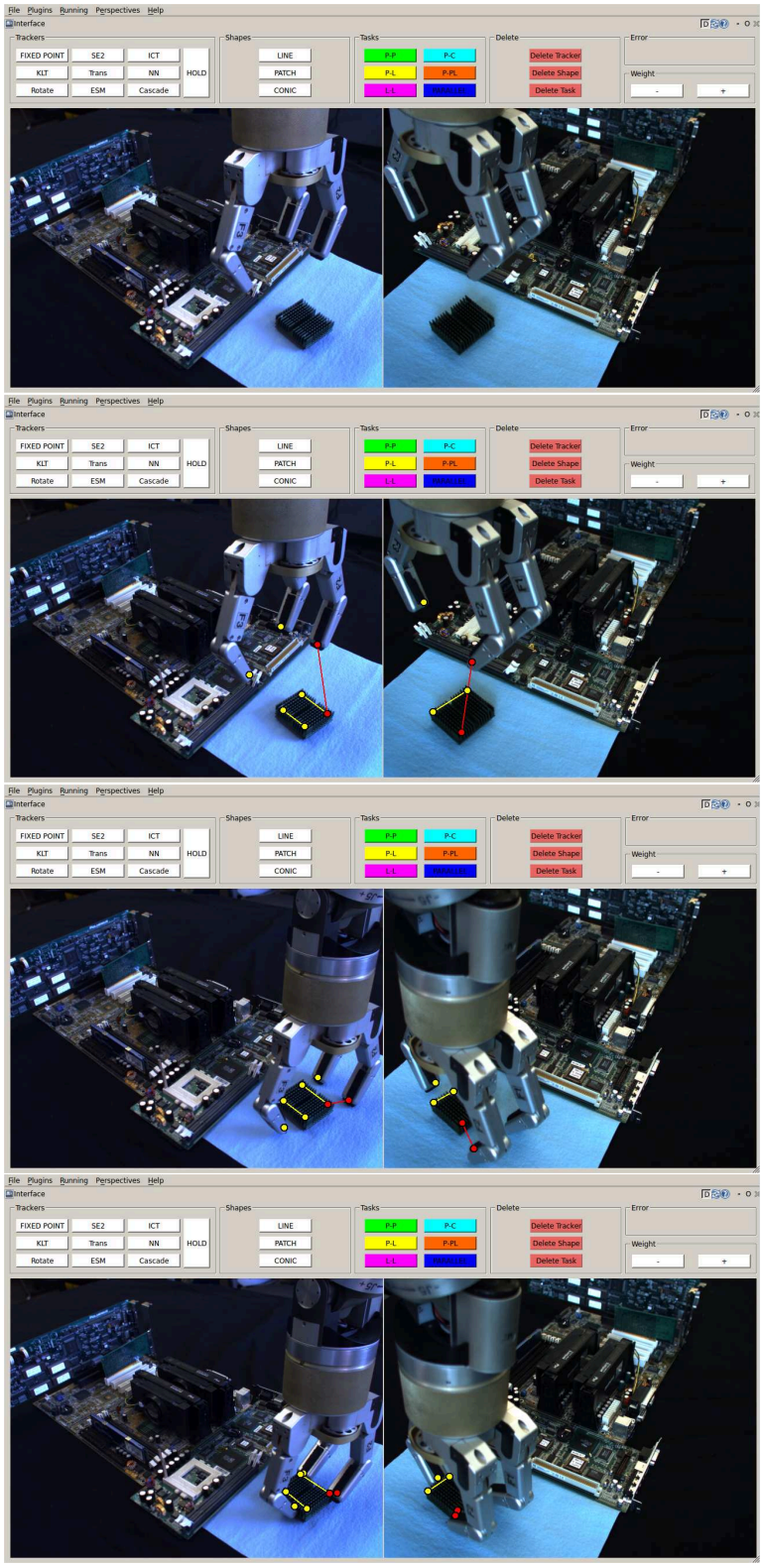


Figure 3.3: Sample task definition on a heat sink installation task. Yellow points show a point to line task while red points define a point to point task.

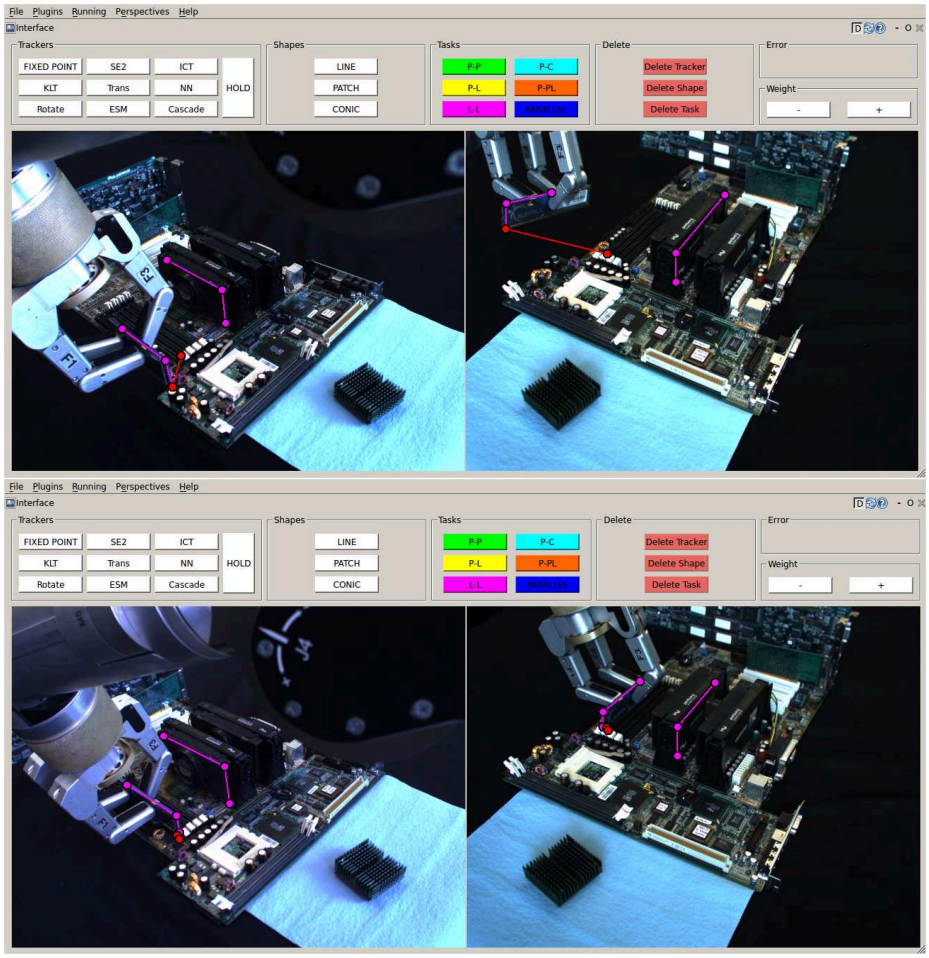


Figure 3.4: Sample task definition on a memory card installation task. Magenta lines represent parallel line tasks which aid in aligning the card. The red point represents a point to line task that moves the card to it's final destination.

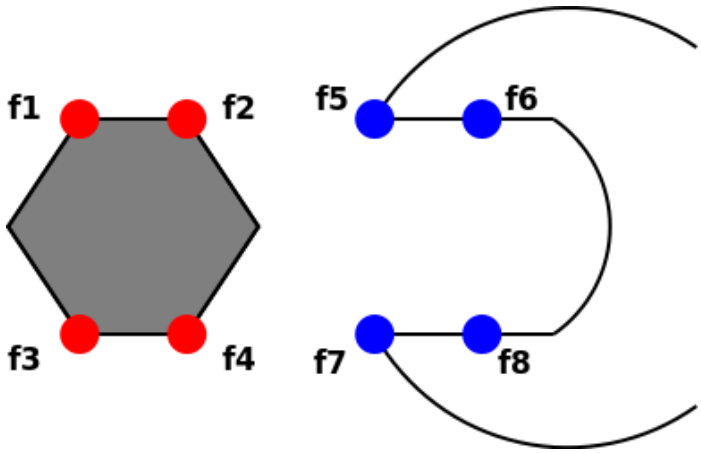


Figure 3.5: Here the goal is to align the wrench on the right to the bolt shown on the left. Using the tracked visual features labeled $\boldsymbol{f}_1$ through $\boldsymbol{f}_8$.

any two points, $\boldsymbol{f}_1, \boldsymbol{f}_2$ is defined as $\boldsymbol{f}_1 \times \boldsymbol{f}_2$. Using this we can define the lines on the top and bottom of the bolt to be: $\boldsymbol{f}_1 \times \boldsymbol{f}_2$ and $\boldsymbol{f}_3 \times \boldsymbol{f}_4$. Finally to align the wrench to these lines we need to define the point to line task.

To specify this task we need an encoding that will equal 0 when the task is completed so that we can satisfy equation 3.3. In this case we would like an equation that is equal to 0 when the point is incident to the line. Luckily in homogeneous coordinates this can be expressed with a dot product as when a line and point are incident then: $\boldsymbol{l} \cdot \boldsymbol{p} = \boldsymbol{0}$. Thus we define the first part of the task as:

$$\boldsymbol{e} = \begin{bmatrix} (\boldsymbol{f}_{l1} \times \boldsymbol{f}_{l2}) \cdot \boldsymbol{f}_{l5} \\ (\boldsymbol{f}_{l3} \times \boldsymbol{f}_{l4}) \cdot \boldsymbol{f}_{l7} \\ (\boldsymbol{f}_{r1} \times \boldsymbol{f}_{r2}) \cdot \boldsymbol{f}_{r5} \\ (\boldsymbol{f}_{r3} \times \boldsymbol{f}_{r4}) \cdot \boldsymbol{f}_{r7} \end{bmatrix} \tag{3.4}$$

This specifies the point to line tasks for both cameras. To specify the constraint that will move the wrench to the bolt we can use a point to point task. As we saw before, the point to point encoding is expressed through the difference in the visual features. Putting it all together we can define the whole task as:

$$\boldsymbol{e} = \begin{bmatrix} \boldsymbol{f}_{l2} - \boldsymbol{f}_{l6} \\ \boldsymbol{f}_{l4} - \boldsymbol{f}_{l8} \\ (\boldsymbol{f}_{l1} \times \boldsymbol{f}_{l2}) \cdot \boldsymbol{f}_{l5} \\ (\boldsymbol{f}_{l3} \times \boldsymbol{f}_{l4}) \cdot \boldsymbol{f}_{l7} \\ \boldsymbol{f}_{r2} - \boldsymbol{f}_{r6} \\ \boldsymbol{f}_{r4} - \boldsymbol{f}_{r8} \\ (\boldsymbol{f}_{r1} \times \boldsymbol{f}_{r2}) \cdot \boldsymbol{f}_{r5} \\ (\boldsymbol{f}_{r3} \times \boldsymbol{f}_{r4}) \cdot \boldsymbol{f}_{r7} \end{bmatrix} \tag{3.5}$$

Although this task appears to be fully defined, one problem can still arise. While testing this task in simulation we observed an unexpected solution. Given these constraints, the orientation of the wrench with respect to the bolt is not unique. Having points $\boldsymbol{f_2}, \boldsymbol{f_6}$ and $\boldsymbol{f_4}, \boldsymbol{f_8}$ coincide can be visually accomplished with $\boldsymbol{f_5}$ and $\boldsymbol{f_7}$ towards the left or the right. A video of this can be seen in the ROS-UVS website [2].

When visually defining tasks it is important to keep in mind what degrees of freedom have been constrained and which ones have not. Otherwise unexpected solutions can be found once the robot motion is performed as was the case with the wrench task.

3.3 Human In the Loop

In *"Bringing visual servoing into real world applications"* [38] our research group discussed how visual servoing can come together to allow users to control robots. The ultimate goal is to have robots aid us in our everyday lives or in the workplace. The challenge becomes greater due to the nature of unstructured and dynamic surroundings as we have discussed. The idea of having a *human-in-the-loop* means the human guides the robot without controlling it completely. In the case of visual servoing defining the task visually is an example of this.

A different approach presented by our group in *"Interactive Teleoperation Interface for Semi-autonomous Control of Robot Arms"* [39] showcases a system where Visual Servoing is used to aid at certain points during robot interaction. Using an RGBD

[2] `http://ugweb.cs.ualberta.ca/~vis/ros-uvs/videos.html`

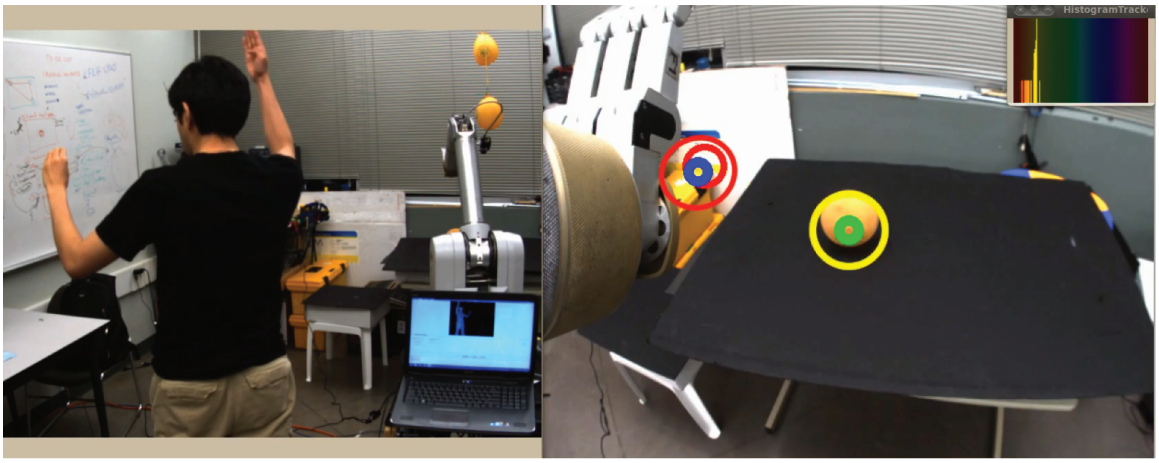


Figure 3.6: Users link the robot's position to their arm. Using this interface users perform large motions and initialize a visual servoing routine that performs the finer alignment.

sensor the arm of a user is tracked. Large motions are performed using a direct mapping from the user's arm to the robot. Through these gross motions users align an eye in hand camera to a target. Once aligned a tracker is initialized and visual servoing then finalizes a finer alignment. Figure 3.6 shows a user interacting with the system. On the right an eye-in-hand view is presented on which a visual servoing task is defined. The blue circle defines a point to point task with the green circle.

3.4 Summary

Overall task specification can be done in simple ways such as unique constraints like point to point or point to line. However more complex tasks such as the manipulation of electronics as shown in figures 3.3 and 3.4 require a combination of these primitives to be used. In [40] the authors suggest tasks should be divided into three main categories: transport, alignment, and fine manipulation. This simplifies the chain of required tasks users must define by dividing the task into manageable pieces.

Defining tasks this way is easy to do for a human operator. However to it is not clear how to automate complex tasks this way. In practice we have been able to initialize trackers automatically to perform simple transport and alignment stages, but

in general this has required prior knowledge of the manipulation task or the relevant objects in the scene. Automatically generating the context required to automate these tasks is still an area that requires exploring.

In this work we have focused our attention to the robot control and the effects of the visual features on the servoing process. Our goal is to facilitate the use of UVS for others. Having said that we can now move on and introduce our UVS library, ROS-UVS.

Chapter 4

ROS-UVS

In this chapter we present ROS-UVS, our Uncalibrated Visual Servoing library. More information can be found on the library's website[1] along with the source code, tutorials, videos, and documentation.

4.1 Feature Overview

In Visual Servoing one or more cameras track scene features and use them to control the motion of a robotic system [18, 19]. Although literature in this field is mature and extensive, relatively few works have directly addressed real-world applications [41]. One of the main roadblocks is the time and effort required to develop the extensive software for video tracking and visual servo control required for real time execution of tasks. Fortunately there are a few good options that allow fast prototyping for Visual Servoing [15, 16]. Arguably one of the most complete libraries is the Visual Servoing platform (ViSP) [16]. As pointed out by the ViSP authors, approaches such as a learning process of the interaction matrix are not currently integrated within their library. In particular an implementation of Uncalibrated Visual Servoing (UVS) is missing. This is a significant omission which, when closed, might help bring Visual Servoing into more real world scenarios. Although current vision sensors in robots can be calibrated with respect to the robotic platform space, a different approach consists of learning on-line how the vision sensors are related to and positioned with

[1] `http://ugweb.cs.ualberta.ca/~vis/ros-uvs`

respect to the robotic platform as was seen in chapter 2.

In this work we introduce ROS-UVS: a minimalistic library for visual constraint minimization through Uncalibrated Visual Servoing. ROS-UVS provides the following:

- An implementation of Uncalibrated Visual Servoing (UVS) performed through estimating the full non-parametric Jacobian, thereby avoiding all a-priori calibration.
- It isolates the core of Visual Servoing functionality in a minimalistic way, while being general in the types of robots, configurations, and tasks it can address.
- Only dependant on Eigen[2] which is a standard of ROS libraries [1].
- Easy interaction with numerous robotic systems through ROS.
- Rather than including extensive video, image processing and tracking functionality, through ROS it can access numerous up-to-date computer vision libraries.
- A fully connected simulation environment available through Matlab by using Corke's Robotics Toolbox [42]. The simulation environment allows users to quickly test the system, even if they do not have direct access to a robot.

Our focus is on a design that is simple to use. To that effect it is fully integrated with with ROS [1]. It is reliable and incurs no extra dependencies on it's users. Our library is freely available at: `http://ugweb.cs.ualberta.ca/~vis/ros-uvs/`.

ROS-UVS is centered around the Visual Serviong process and accesses tracking details and robot functionality through ROS. The main flow of information is shown in Figure 1.4. First, tracking software is used to define visual features that identify the locations of interest. Given the visual features, users construct an error vector or encoding as shown in Chapter 3. This error vector is made in such a way that when the constraints defined are met, the desired task will be completed and the error

[2] `http://eigen.tuxfamily.org`

vector evaluates to 0. The control law defined in the Visual Servoing will then find the motions required to minimize the given error and these will be used by the robot controller to close the loop and perform the motion.

One limitation of Visual Servoing comes from the limited information that can be acquired from a single point of view. To counteract this, two or more cameras can be used. Given the multiple view points, constraints can be defined to complete most 3D tasks. Section 4.4 showcases several systems where we utilize our ROS-UVS system to complete a variety of tasks in 2D and 3D configurations. There is no limitation to the number of cameras that can be used. Adding more view points will help improve the stability and convergence. Examples of this are shown in Chapter 5. Since this system performs Uncalibrated Visual Servoing the use of multiple cameras is facilitated as they can simply be placed in the scene without needing to calibrate their location or intrinsic properties.

Regardless of camera configuration the use of ROS-UVS is the same. The ROS topic `/image_error` is used to report the error vector. For those unfamiliar with ROS many online tutorials are available in the ROS website.[3]. As a simple introduction ROS allows users to create connected programs which are referred to as nodes. ROS standardizes message prototypes that can be used to communicate between nodes. This is especially useful in robotics as many components usually have to come together to create a full robotic system.

ROS-UVS subscribes automatically to this topic through the Tracker Manager object and uses the reported error to estimate the interaction matrix and to generate the motions for the robot. Additionally ROS-UVS uses the reported errors to identify when a tracker has been lost. Although identifying drift is challenging and outside of the scope of this library, whenever a tracker stops publishing information the system is smart enough to stop all robot motion. This is essential as with a lost tracker the scale of the reported errors could change drastically resulting in unexpected robot motion. As an added safety provision limits on joint velocities and joint movements are also configurable by users. This can be done directly on the system while the

[3] `http://wiki.ros.org/ROS/Tutorials`

servoing is taking place. This is done by using ROS's dynamic reconfigure package [4].

4.2 System Overview

Our system is free and open-source and adheres to design principles outlined in other software libraries such as ROS. The result of our design is a standalone library that has no external dependencies. This allows for code extraction and reuse beyond its original intent. As outlined in [43] many robotics software projects contain implementations which could be reusable outside of the project, however due to limitations in the design it often becomes impractical to do so. By approaching Uncalibrated Visual Servoing as a general robot interaction we can abstract the robot controller and the image error tracking into individual modules.

In [44], the architecture for mobile robotic systems is evaluated. Although Visual Servoing is not necessarily dedicated to mobile robots, many of the same properties for a generic robotic system are desired. In their evaluation Kramer et al. suggest design for robotic systems should follow 7 main requirements. According to the authors systems should: (1) abstract robot hardware, (2) be extendible and scalable, (3) provide limited run-time overhead, (4) be modelled with actuator control, (5) follow good software practices, (6) provide tools and methods, and finally (7) be well documented. In our implementation we attempt to complete these requirements. Our system abstracts the robot through a robot controller that can be adapted to any robot. This can be extended and is ultimately a direct link to the actuators as these are what the defined control laws ultimately control. Having a minimalistic framework allows for minimal run-time overhead. The choice of Eigen as our linear algebra library also follows along these lines as its benchmarks show it as one of the best performing linear algebra libraries available [45]. The simulation environment provides simple prototyping and makes the system easy to test. Documentation and sample uses of the library have also been provided.

A minimalistic example for using ROS-UVS is presented in listing 1. For more

[4] `http://wiki.ros.org/dynamic_reconfigure`

details the most up to date and complete documentation of the code can be found at the library's website.[5].

```
#include <ros/ros.h>
#include "ros_uvs/tracker_manager.h"
#include "ros_uvs/ibvs.h"
#include "ros_uvs/wam_controller.h"

int main(int argc, char *argv[]) {
  ros::NodeHandle nh_("~");
  TrackerManager tm(nh_);
  WamController wam(robot_ns, nh_);
  IBVS ibvs(robot, tm);
  int dof = 4;
  ibvs.update_jacobian(robot, dof);
  int timeout = 10;    // seconds
  float lambda = 0.05; // step_size
  ibvs->converge(timeout, lambda);
}
```

Listing 1: Minimalistic UVS implementation leveraging ROS-UVS.

In typical ROS fashion a node is first initialized. This node is the main process by which the ROS system is defined. Three main objects then come together to form the Uncalibrated Visual Servoing system.

Users of our library need only create a small interface to the robot that will be controlled. In this interface a method to signal the robot motion will need to be implemented. This can be done through either joint position commands or velocity control. The UML diagram in Figure 4.1 illustrates how the data is managed and shows the three main classes relevant to the example listing.

First a *TrackerManager* is created. This system is the bridge between ROS-UVS and the visual tracking library of the user's choice. Through this bridge the defined visual error is fed into our system. The visual features are expected to be setup by the user into an error vector of their choosing. This is left as a user configuration as there is no restriction on what errors can be defined. They are task dependant and as such it is not practical to enforce a rigid structure around them. By leaving this

[5] http://ugweb.cs.ualberta.ca/~vis/ros-uvs/examples.html

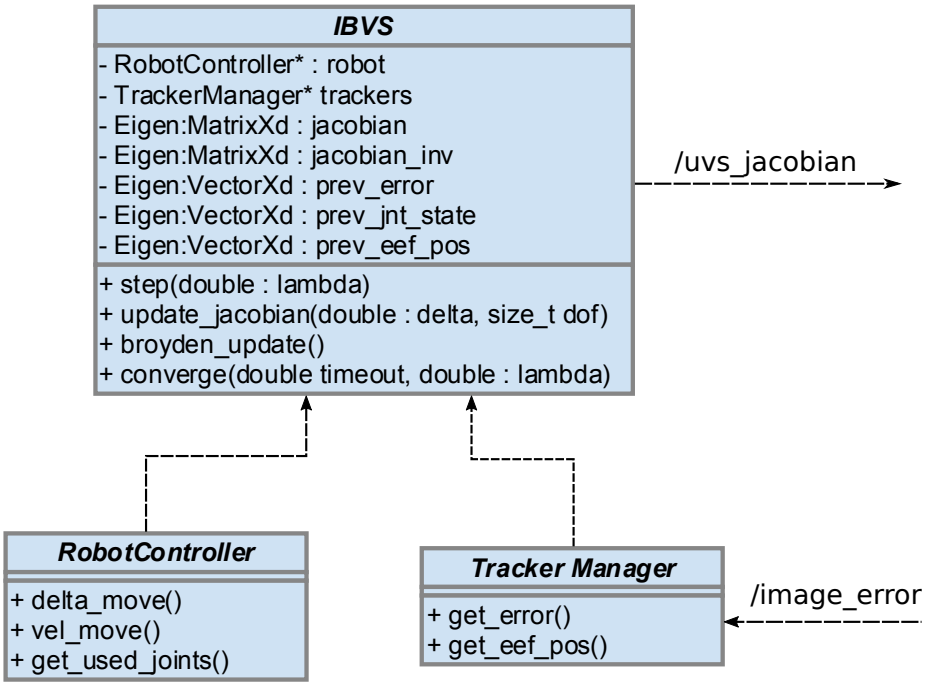


Figure 4.1: UML diagram overview. Three main classes are defined. IBVS implements Image Based Visual Servoing and publishes the calculated Jacobian out through the `uvs_jacobian` topic. Tracker manager handles the data published by trackers through the `/image_error` topic.

definition in the user's control we maintain greater flexibility. Once defined, the error vector is read through the *image_error* topic. To have a responsive system the error should be published with minimal delay after a frame is captured by the cameras.

Then, a *RobotController* is defined. This is the main interface to the robot to control. The controller provides the ability to make small robot motions with the `delta_move` method. This can be implemented through any available robot interface, for example both absolute movements or velocity controls will work. The only requirement is that the implemented control method causes the motions on the robot.

Finally, the Visual Servoing object is created. This object, `IBVS`, depends on both the *TrackerManager* and the *RobotController*. Using references to these objects the `IBVS` object can then direct the robot to perform the required motions and find the image Jacobian while at the same time accessing the error produced by the trackers. As an isolated module different servoing strategies can be easily implemented. In testing our system we have tried a variety of changes such as generating exploratory motions based on visual feature motion rather than direct joint angle motions. Creating modified servoing modules is straight forward and easy to do.

Once everything has been defined we can drive the system to attempt convergence with a given `timeout` and a step size `lambda`.

4.3 Simulation

Using Matlab's new Robotics System Toolbox [46] which connects to ROS, the same system calls and control directives are used for both simulation and real robots. Figure 4.2 shows a sample simulation of a Visual Servoing experiment.

In the simulation environment that we release with ROS-UVS we allow users to create simple configurations for visual servoing. Virtual Cameras are available following either a simple pinhole camera model or a full projective model.

The pinhole camera model as presented by Hartley and Zisserman [47] is shown in Figure 4.3. In general it is easy to think of the projection as a series of homogeneous coordinate matrix multiplications as shown in equation 4.1.

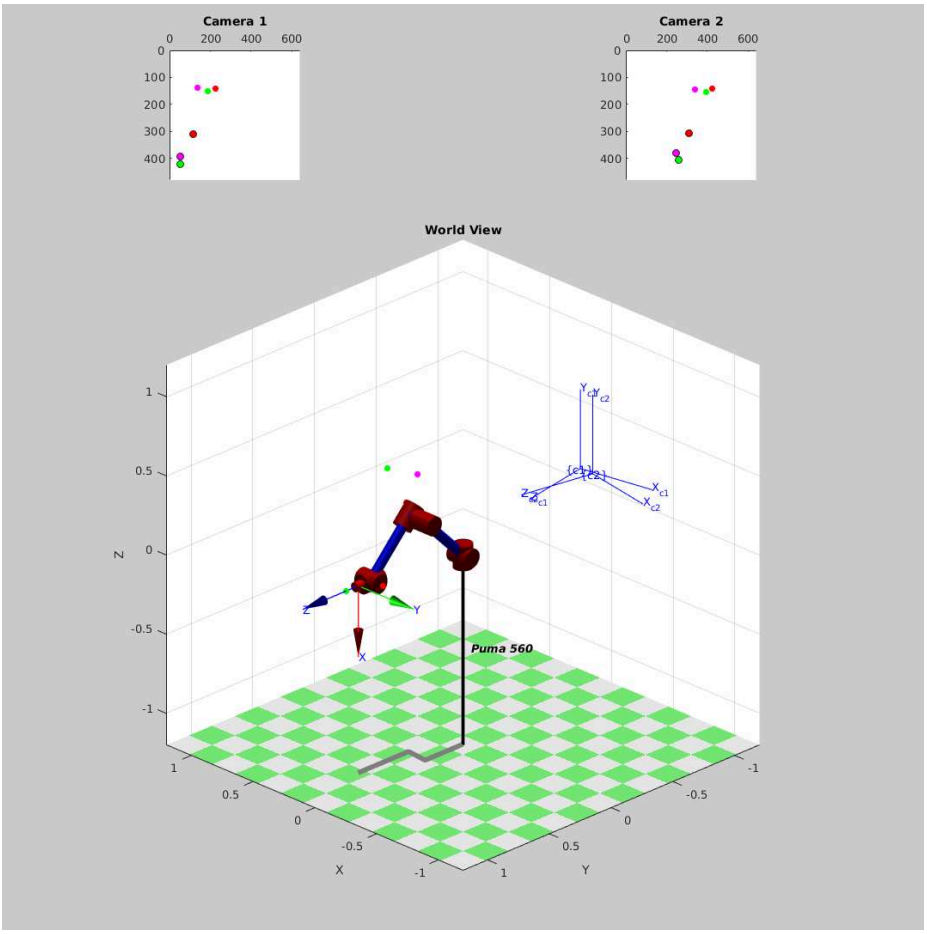


Figure 4.2: A sample simulation experiment. The robot has 3 points attached to it's end effector. Color coded target points are also visible. The two frames show the position of the virtual cameras. At the top the images as seen from these cameras are shown.

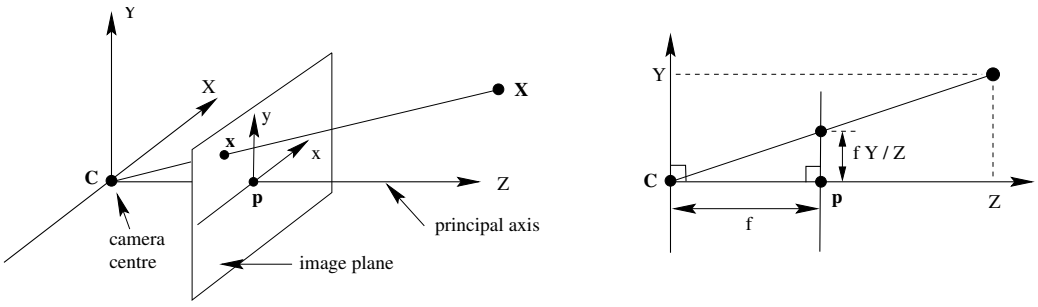


Figure 4.3: In the pinhole camera model the camera is shown with its center C viewing down the Z axis. The image camera plane is formed by the projection of points in 3D such as X onto the camera image.

$$\begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} \text{Intrinsic} \\ \text{Parameters} \end{bmatrix} \begin{bmatrix} \text{Projection} \\ \text{Model} \end{bmatrix} \begin{bmatrix} \text{Extrinsic} \\ \text{Parameters} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ T \end{bmatrix} \tag{4.1}$$

This series of matrix multiplications transform 3D points into their corresponding 2D image projection points. The *extrinsic parameters* describe how the camera is positioned with respect to the world coordinate frame. The *projection model* then projects the points to the image plane. This can be done in different ways. Orthographic projection will for instance give no scaling, while a weak perspective projection will give a linear approximation. Finally the intrinsic parameters convert the metric projection into pixel values.

Using this camera modeling the Matlab simulation allows users to create and place cameras in the simulated world and acquire the images as they would be seen by the cameras. These cameras points that are rigidly attached to the robot are used to simulate Visual Servoing. We use the camera projection model to follow features in three dimensions. By projecting them unto the image we generate the visual features used in the servoing.

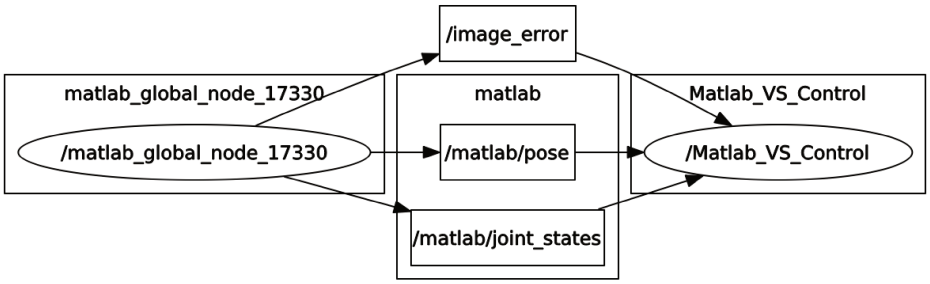


Figure 4.4: ROS graph of active nodes during Matlab Simulation

The simulated world creates a ROS node as seen in Figure 4.4. The cameras and robot are placed in the simulated world. An error function is then defined relating the tracked points from the simulated camera images and their corresponding targets. The output error is published through the `image_error` topic as in a real robot system. To make data recording simpler the robot's state is also published through the

```
% Initialize robot
mdl_puma560;
robot = p560;
robot.tool = rt2tr(eye(3), [0 0 0.1]')
initial_pose = [1.5708 0.7854 3.1416 0 0.7854 0];
target_pose  = [1 1.9635 2.7489 0 0.7854 -pi/4];
target_T = robot.fkine(target_pose);

% Define structure of tracked points with respect to the
% robot's tool
eef_point_displacement = [0.00  0.00  0.00;
                          0.00 -0.10  0.10;
                          0.10 -0.10 -0.10;
                          1.00  1.00  1.00;];

% Target points are transformed to the target robot pose
target_points = target_T * eef_point_displacement;

% Create cameras
camera_rotation = rt2tr(rotx(pi/2) * roty(180, 'deg') * rotz(pi);
c1 = ProjCamera(camera_rotation, [ 0.05, -1, 0.0]'), 'c1');
c2 = ProjCamera(camera_rotation, [-0.05, -1, 0.0]'), 'c2');
cameras = [c1;c2];

% Create point to point error function
e = @(eef, target) target - eef;

W = World(cameras, robot, initial_pose, target_points,
          eef_point_displacement, pi/180, e, true);

ts = rossvcserver('/matlab/joint_move', 'visual_servoing/JointMove',
                  @W.joint_move)
```

Listing 2: Sample world setup for simulation experiment in Matlab.

`/matlab/pose` and `matlab/joint_states` topics. These then connect to a controller node which the user can direct to perform the simulation.

Sample experiments are released and documented on the library website and also used to investigate convergence behaviour and how camera placement affects it. The results of these experiments are presented in Chapter 5. A sample world configuration matching that of Figure 4.2 is shown in listing 2.

Configuring a simulation world is straight forward. Users first define the world structure. This includes the robot to be used, the cameras and their pose with respect to the robot, the features to track on the robot and the target pose for the robot. The error function must also be defined with respect to the tracked features. Once defined the world is created and the ROS topics are created.

4.4 Sample Robot Use Cases

Our ROS-UVS system has been used in several projects within our lab. In our experience this system has shown to be flexible and easy to adapt to different applications. Initial testing was done on two WAM arms using 4 and 7 degrees of freedom (DOF). Later an Android interface to control the arms was developed. Virtual control points were also used to drive the motion of the robot in a compliant control system through the use of virtual springs. An interface into an ARDrone was also implemented. Some other use cases are also discussed in this section. Table 4.1 outlines the number of DOF controlled by UVS, the type of error that is defined, the number of cameras used, and the type of interaction of the implemented systems. Interaction type is determined by whether the user is able to dynamically modify the target point for the servoing, or if the target will remain static once the user defines it for the first time.

4.4.1 Visual Task Specification Interface

Visually specifying high-level tasks by combining a set of geometric constraints in an intuitive way is challenging. In this system we address this challenge by presenting

System	Controlled DOF	Error	# Cameras	Interaction
Visual Task Specification Interface	User Defined	User Defined	2	Static
Android Mobile	2	point-to-point	1	Dynamic
Virtual Control Point	3	point-to-point	2	Static
Amazon Picking Challenge	3	point-to-point	2	Automatic
Quadrotor Assisted Flight Control	1	angular	1	Static
3D point minimization	3	3D point-to-point	1 RGB-D	Static
Gesture Based Control	3	point-to-point	1	Static

Table 4.1: Overview of ROS-UVS use case properties. Controlled DOF are driven directly by ROS-UVS. The number of cameras alongside the error type determine the kind of motion and tasks that can be completed. Finally the interaction type determines if the user can dynamically change the servoing target or if it remains static once initialized.

the user with an image-editor like visual interface where they can specify the task. First, users initialize trackers on the relevant objects. Then, by selecting the defined trackers, geometric constraints such as point-to-point, or point-to-line task are defined. These constraints produce the required error measure that is then given to the ROS-UVS library. Using this error a Jacobian relating the robot motions to the geometric constraints is learned. Using Visual Servoing the robot movements are performed and the specified constraints are minimized. As can be seen in Figure 4.5 the initialized trackers are indicated with the coloured points. Once the Visual Servoing is performed the error is minimized.

This system allows us to explore how different constraints affect the robot's motion and how users plan in order to complete tasks. More analysis in this area is needed and it is part of future work.

4.4.2 Android Mobile Interface

In this system the user's touch location is tied directly to the Uncalibrated Visual Servoing target. This provides an intuitive interaction as the robot will follow the user's finger. Development within a mobile system such as an Android device is now simple as many tools are provided by the ROS environment. Having the ROS-UVS library integrated in the same ecosystem only helps to further expand it.

Figure 4.6 showcases the use of the mobile interface. A top-down view of the

Figure 4.5: **Top:** User defines geometric constraints visually. Green points indicate point-to-point constraints. Magenta indicates a line-to-line constraint, and finally blue is used for parallel constraints. **Bottom:** Once the constraints have been specified the Visual Servoing takes place minimizing the error of the defined goals.

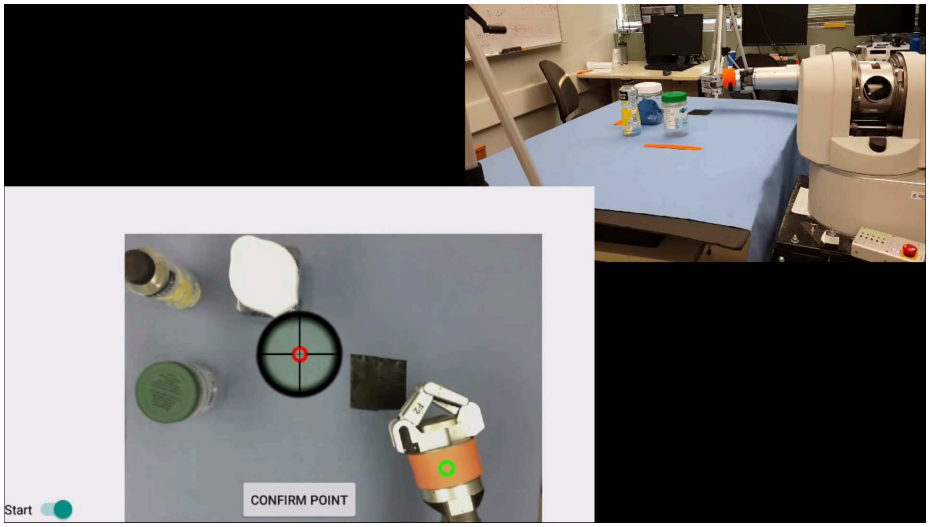


Figure 4.6: **Left:** Mobile platform view. Green circle displays tracked center of the end effector. Crosshairs indicate the user's defined target location.

robot's workspace is displayed to the user. Through touch interactions the user can initialize a tracker on the end effector and activate the servoing. In this system the need for tunable settings became apparent. Learning rate for Broyden updates, maximum joint velocities, and step size are some of the main attributes that require tuning to present the user with a natural system to use. ROS-UVS presents an easy to use API allowing users to quickly develop a system and configure user interfaces where settings can be altered while the system is running through ROS's dynamic reconfigure system. This facilitates parameter tuning and experimentation as changes can be done live.

Using these features we were able to create a user interface that was responsive and created predictable motions for the robotic arm without requiring a camera calibration process. A great advantage of this system was that we required no registration to be performed with the camera. We compared several other control modes including a gamepad and a calibrated joint control system. Not having to worry about the robot or camera moving made the visual interface easier to quickly setup and test.

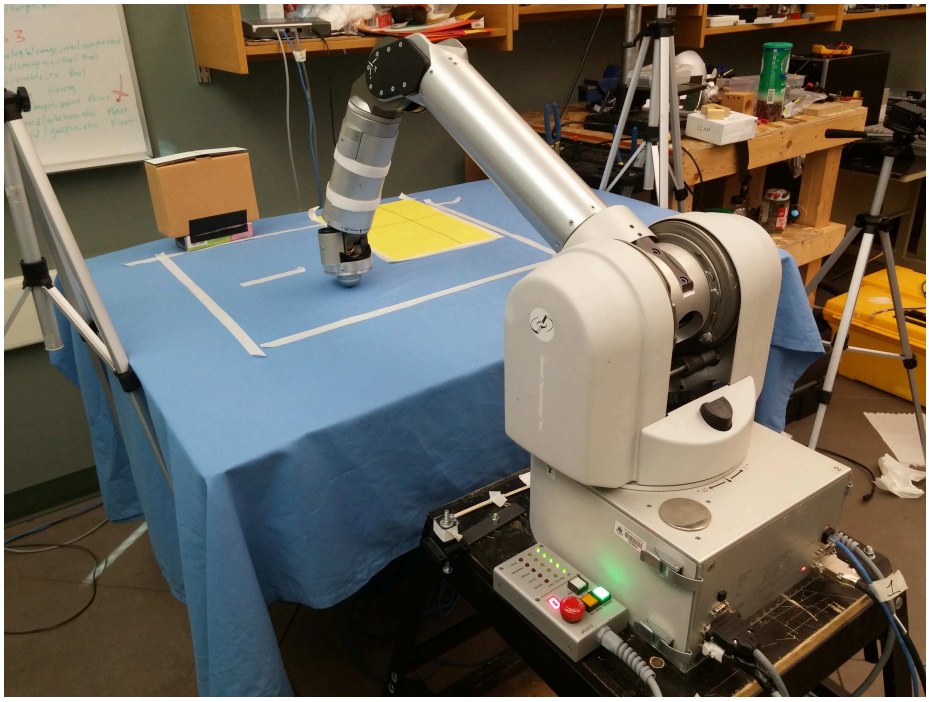


Figure 4.7: Using a virtual spring center as a virtual control point we are able to servo the arm while maintaining contact with the table top surface.

4.4.3 Virtual Control Points

This example describes the application of UVS for a compliant system in contact with the environment. To this end, a virtual spring is added. Assuming a linear spring model, the force applied by the end effector is:

$$\boldsymbol{F} = \boldsymbol{K}(\boldsymbol{x_d} - \boldsymbol{x}) \tag{4.2}$$

where $\boldsymbol{K} = diag(K_x, K_y, K_z)$ is a stiffness matrix, $\boldsymbol{x} = (x, y, z)^T$ is the Cartesian position of the end effector and $\boldsymbol{x_d} = (x_d, y_d, z_d)^T$ is the centre of the virtual spring. In order to maintain the contact with the environment z_d is set to be $20cm$ below the end effector position Z. This generates a constant force in negative Z-direction. Given the computed Cartesian force, the commanded torque to the robot would be $\boldsymbol{\tau} = J_r^T \boldsymbol{F}$, where J_r is the robot Jacobian.

UVS is then used to move the robot's end effector towards a goal. To do this, we slightly modify the control law such that the interaction matrix J relates the

Cartesian space of the virtual spring with the image space as follows:

$$J = \begin{bmatrix} \frac{\partial f(\boldsymbol{x_d})}{\partial x_d} & \frac{\partial f(\boldsymbol{x_d})}{\partial y_d} & \frac{\partial f(\boldsymbol{x_d})}{\partial z_d} \end{bmatrix} \tag{4.3}$$

and then $\dot{\boldsymbol{x}}_{\boldsymbol{d}} = -\lambda \hat{J}^{\dagger} \boldsymbol{e}$. The commanded torque to the robot would be $\boldsymbol{\tau} = J_r^T \boldsymbol{F}$, where J_r is the robot Jacobian. Using this configuration allowed us to generate a visual control system where we could safely generate contacts between a solid table and the robot.

The advantage here is that we are able to use the whole UVS framework and by changing the control output of our robot controller to be in the Cartesian space of a virtual point at the end effector, rather than the individual robot joints we can create a new way to interact with the robot. The exploratory motions we generate to estimate the robot Jacobian are general and can be used in such a way directly.

4.4.4 Amazon Picking Challenge (APC)

Our research group participated in the APC [3] where contestants were asked to grasp a variety of objects from a shelf. The ROS-UVS system was used to perform grasp alignment. Figure 4.8 illustrates an example grasp. SURF [48] features were learned for the set of objects to be grasped. Using these visual features, objects are identified as they come into view.

A tracker is then automatically initialized and the grasp alignment is done through UVS. The error is defined as a point to point geometric constraint from the object to the center of the robot hand. Once the error is within a threshold the servoing is stopped and the fingers on the robotic hand are closed.

Although the task was defined by simple point to point visual error the positioning of the arm after performing the Uncalibrated Visual Servoing was precise enough to finalize the task. In this example a great feature is that the system worked autonomously and successfully performed the grasp. A downside is that we required a-priori information to automate the instantiation of the tracking system.

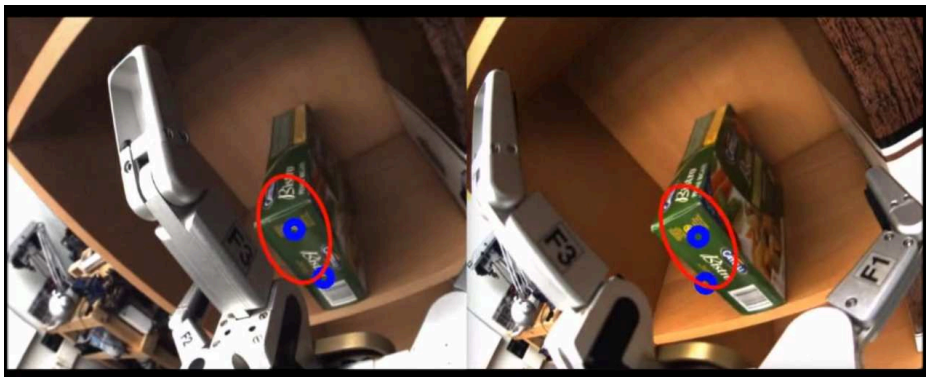


Figure 4.8: Grasping done via Uncalibrated Visual Servoing. A colour tracker is initialized to track the position of the green box. A target is defined in between the hand's fingers. Once a threshold is reached the hand closes completing the grasp.

4.4.5 Quadrotor Assisted Flight Control

Assisted control for the flight of an ARDrone, a commercially available quadrotor, was developed with the aid of ROS-UVS. The teleoperation system is designed to aid users in the task of following a line by having the drone automatically adjust the yaw while in flight to align with the target.

A sample scenario of this can be seen in Figure 4.9. Utilizing the bottom facing camera of an ARDrone we perform line tracking by segmenting the yellow line. The angle offset of the line to the vertical is then used to calculate the error for the visual servoing. Teleoperation is performed where the user takes control of the roll, pitch and elevation degrees of freedom while the UVS is performed for the drone's yaw. This use case was also automated and users were could turn the piloting assist on when required.

The system allowed users to divide the drone's control by taking over a single degree of freedom. This approach can be used to facilitate tasks for users while at the same time leaving them with an overall control of the robot.

Figure 4.9: Line alignment with UVS controlling the drone's yaw position. Once UVS is active users manually control roll and pitch to position the drone accordingly.

4.4.6 Other Uses

Two other relevant systems use ROS-UVS. First a 3D minimization where the error tracking is performed in 3D. Using an RGB-D sensor, such as Microsoft's Kinect [49], a 3D location is tracked and the error is formulated as a point-to-point constraint explicitly in 3D. Learning the interaction matrix with this 3D error the robot is able to perform UVS in the same way as with any of the other applications presented in this chapter. Notice that common RGB-D manipulation systems require a calibration step in order to make rigid body transformations between the robot reference frame and the sensor's reference frame. With our implementation this is not required as the relationship is learned automatically when the Jacobian is estimated. This use case is another example of the flexibility of using virtual control points through the visual servoing.

Finally a gesture based system was also developed. Through this system users are able to initialize trackers which are used to perform UVS. These two final use cases are of interest as they are not common uses of UVS. In this case the flexibility of ROS-UVS made it possible to generalize its use into these peculiar scenarios.

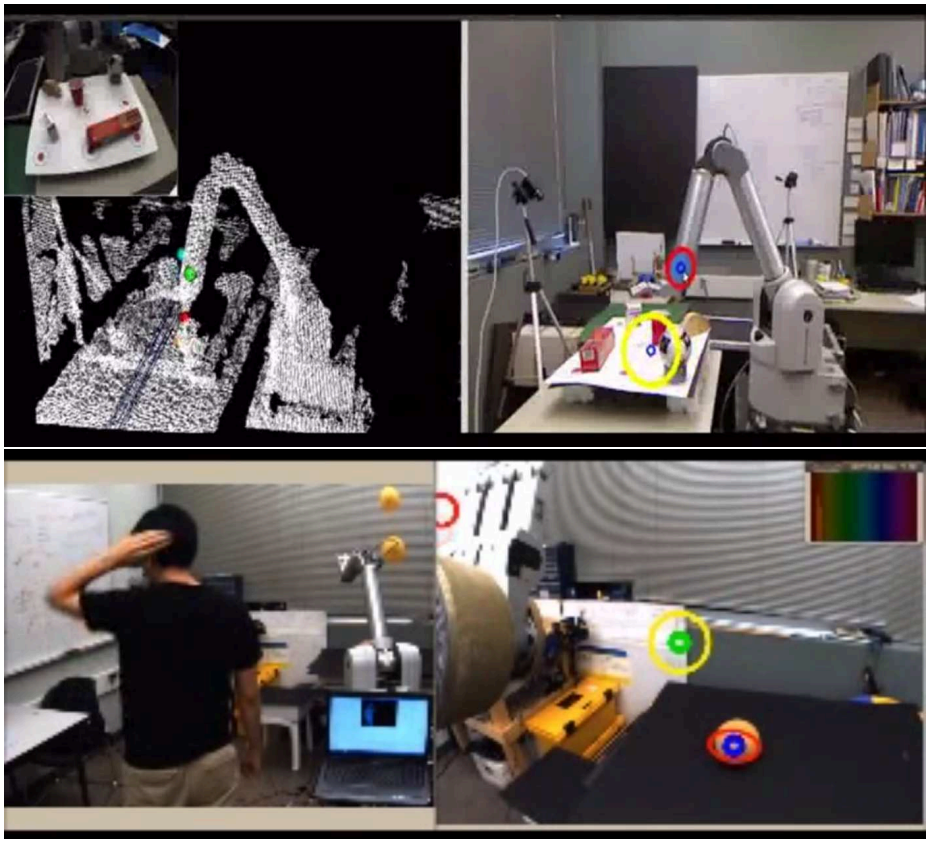


Figure 4.10: **Top:** Using a Kinect 3D points are tracked and a point to point in 3D space is defined. The arm then moves to minimize the tracked error. **Bottom:** Utilizing body tracking a 3D cursor is created. Through gestures users are able to initialize trackers and define a point to point task.

Chapter 5

Visual Servoing Convergence and Control Strategies

This chapter demonstrates Uncalibrated Visual Servoing in practice and summarizes many of the lessons learned while using our library. With the use of ROS-UVS we showcase and study the behaviour of UVS. Section 5.1 presents three prominent challenges we have faced with UVS; mainly the problems of camera positioning, noise in the visual trackers, and finally how to apply the control laws on the physical robot. In section 5.2 we then present several experiments tailored to investigate and highlight the challenges described. Finally in section 5.3 we conclude by reviewing the lessons learned through these experiments and our use of ROS-UVS.

5.1 Uncalibrated Visual Servoing Challenges

With Uncalibrated Visual Servoing our goal is to control the motion of a robot with the aid of image data. Utilizing cameras we generate constraints that limit and or guide the robot's movements. There are many things that affect how well the servoing progresses. In this section we first explore the challenges in the placement of cameras and task definition. Then, we quickly take a look at the noise present in visual trackers when operating on common objects that we manipulate. Finally, the roadblocks faced when controlling our robot are discussed along with the approaches we used to work around them.

5.1.1 Task Definition and Camera Placement

One of the main concerns when performing Uncalibrated Visual Servoing relates to how define the visual features being tracked. These features dictate how the servoing will be performed. To address this we need to consider what task is being defined, and how the cameras are placed.

If the visual features are not defined properly, several problems arise. For instance, one common problem is not properly constraining the target pose. The number of degrees of freedom that are controlled must be fully constrained by the visual features, otherwise, the resulting robot motion and the final pose can vary from the expected result. An example of this was shown for the wrench task in Figure 3.5. The total number of DOF that can be controlled with VS is directly linked to the unique constraints in the error vector. These constraints can however overlap. In general knowing off hand what degrees of freedom have been constrained is not straight forward and users must acquire the knowledge and intuition to determine how a desired task could be defined. It is useful if users can learn how to think of the object and robot motions with their projective motions in mind.

Camera positioning and the chosen visual features also present a challenge. Placing cameras so that the task at hand is visible, both when being defined, and completed, is not always possible. Occlusion during the motion of the robot can cause trackers to fail. Since the servoing performs a closed loop control, the desired motion of the robot can not be calculated as specified until the trackers are reinitialized. When a tracker fails the servoing is stopped. If the task has been overspecified this could be handled better by identifying what rows in the error vector have been affected and acting accordingly. This is left for future work.

The two prominent camera positions used are either *eye-in-hand*, or *eye-to-hand*. An example for each of these can be seen in figure 5.1. In *eye-in-hand* the cameras are positioned such that they can get an overview of the scene. With an *eye-to-hand* configuration the cameras are mounted rigidly on the robot and the motion of the robot will cause the camera pose to change. Where possible, the camera configuration

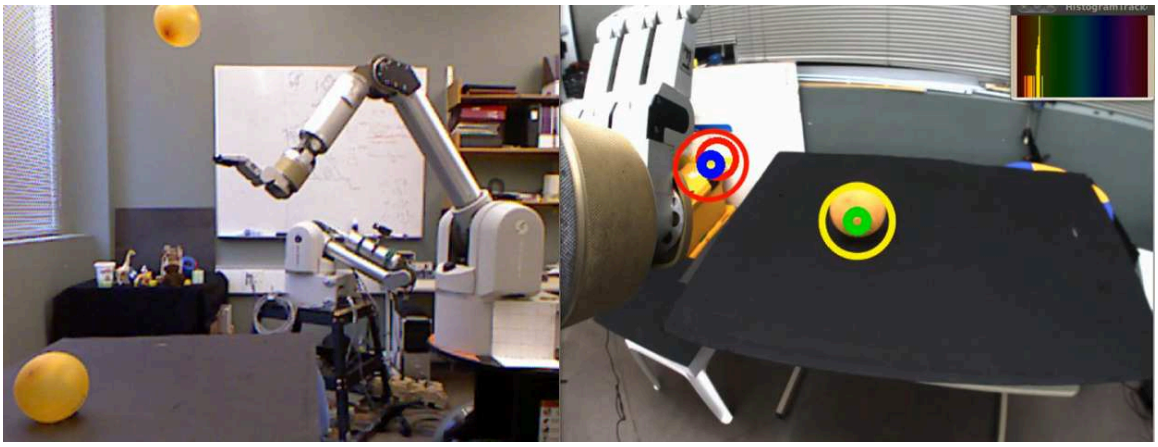


Figure 5.1: Examples of camera placement **Left:** Eye-to-hand camera. **Right:** Eye-in-hand camera.

should take into account the task at hand in order to facilitate its completion.

An example task can be seen in Figure 5.2. Here the *eye-in-hand* camera configuration was selected. The rigid motion of the camera with respect to the robot facilitates the task specification. The center of the gripper can be easily defined as a static point on the images and the only required tracker is placed on the straw. Configuring the cameras and the robot correctly makes difficult tasks such as picking a small thin object like the straw feasible. This is one example of a fine object manipulation task that we were able to perform through Visual Servoing that was not possible with low priced, off the shelf, 3D sensors [12]. As outlined in our paper the point cloud reported by the Kinect sensor used was not able to see the straw.

Task definition is also constrained by the camera positioning. As the robot moves, maintaining the visual features within the camera's field of view can be challenging. First the robot will perform motions to derive the image Jacobian. As the next couple of sections show, the system becomes more stable when these motions result in larger changes of the visual features. A balance must then be found to allow the motions of the arm to be large while maintaining the visual features in the camera's field of view. Several approaches have been studied to try and solve this problem, but they usually require camera calibration as presented in [50], or path planning as done in [51].

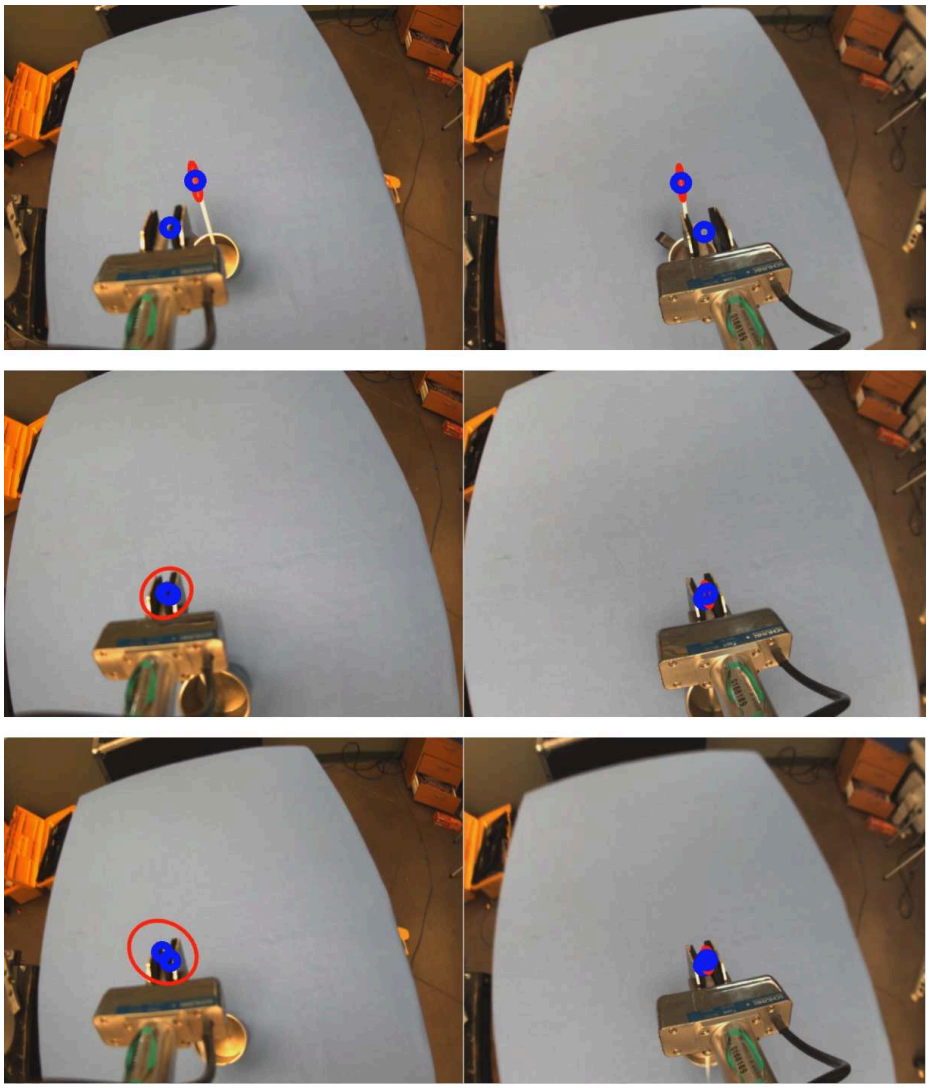


Figure 5.2: Two camera views are shown on the left and right columns. The straw is tracked in both cameras, a point to point error function between the two blue rings on each camera is formulated. When both errors are minimized the gripper is in position to grasp the straw.

5.1.2 Characterizing Visual Tracker Error

Having discussed some of the issues and challenges with camera placement we now turn our attention to the visual trackers used for VS. Understanding their behaviour can help us better simulate them and determine their effects on the servoing. As the robot moves the tracked features will change which can cause trackers to drift or fail. Unfortunately there is no quick and easy solution to this problem other than wait for better trackers to be developed. As users become more experienced with the behaviour of visual trackers and the robot motions, it becomes easier to identify good candidates for tracking targets.

Figure 5.3: Sample objects used to measure noise in a sub-pixel accurate tracker

Since we are dependant on tracker performance it is crucial we understand how they behave and what their effect is on the servoing. To get an estimate of the normal noise present on a tracker we tested several objects. We performed the tracking using three different available trackers: (1) Approximate Nearest Neighbour Search [52], (2) ESM [53], and (3) Inverse Compositional [54]. As shown in figure 5.3 we initialized trackers of different sizes on several objects and recorded the points reported.

After initializing the trackers we moved the objects in order to simulate the motions that they will encounter in a typical manipulation task. Having moved the objects we recorded the published feature points over 10 seconds and observed their distribution. A sample distribution of the reported points for the playing card is

shown in figure 5.4. The standard deviation found for the objects we tried ranged between 0.03 to 0.43 pixels when the tracking had not failed or drifted.

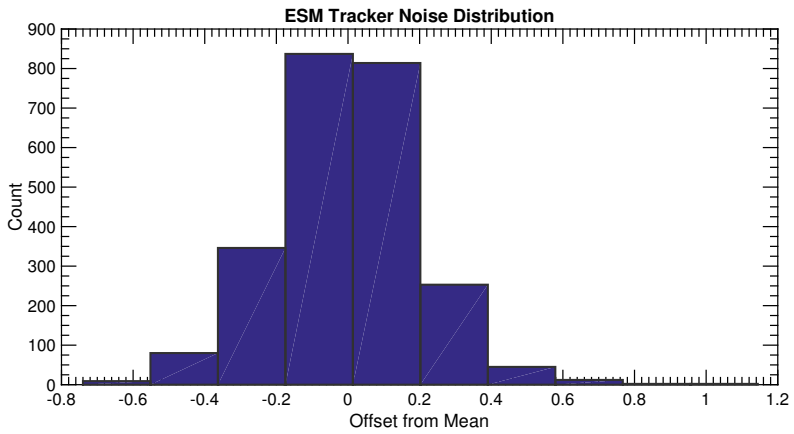


Figure 5.4: Sample distribution of the reported points for the Approximate Nearest Neighbour tracker initialized on a playing card.

These numbers give us a good estimate of the best case scenario with trackers that are easily available. The trackers tested here produce sub-pixel accurate feature points. Depending on the implementation details, trackers can sometimes report integer values corresponding to the tracked pixel. In section 5.2 we use these values to compare the effects of the noise present in these trackers.

5.1.3 Robot Control

In Chapter 2 we derived the control law used for Image Based Visual Servoing. As shown in equation 2.15, the joint velocities, $\dot{\boldsymbol{q}}$, are used to drive the robot towards minimizing the visual error. One aspect of this control law that is not often discussed in literature is its direct relationship with a direct joint control approach.

To better understand this relationship we can refer to the system overview of ROS-UVS in Figure 1.4. The thing to note is how the update of the control law is propagated to the robot. In practice, on each new image frame received by the cameras, the error and the resulting joint velocities are calculated. This means that the robot's joint velocities are updated at the same frequency as the camera frame rate as long as the visual trackers can process the image data.

Given a deterministic update rate dictated by the camera we can find an equivalent joint position control to be:

$$\dot{\boldsymbol{q}} = -\lambda \hat{J}^{\dagger} \boldsymbol{e} = \Delta \boldsymbol{q} r \tag{5.1}$$

Where r is the inverse of the camera frame rate. For example, for a 30 fps camera, $r = 1/30$.

It is important to realize this relationship exists when using Visual Servoing. Although in practice they are not exactly equivalent, due to the dynamics of the robots. For example, whenever there is a velocity change, the robot will first have to accelerate to that velocity, meaning the whole $\Delta \boldsymbol{q}$ distance will not be traveled. In practice they behave similarly and the use of joint control can be advantageous in certain scenarios. In our experience the control of the robot through joint motion commands has been more intuitive and safer to use.

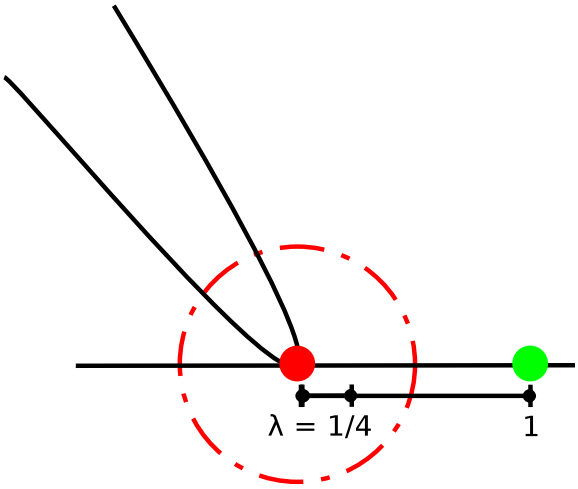


Figure 5.5: Small motions are restricted by an area of tolerance within the robot's controller. When faced with a goal that is very close using a step size λ that is too small might reduce the directed robot motion into the minimum tolerance resulting in no motion.

The specifics of the robot being used can also affect how reliable the system is. For example, in the WAM control, commanded joint movements result in no motion when they are sufficiently small. As is shown in Figure 5.5, when performing Visual

Seroving this situation can come into effect. Suppose the robot is trying to move from its current position at the red circle with a target located at the green circle. Assume also that we have a perfect interaction matrix such that the distance between these points would be covered by one iteration of our control law when using a $\lambda = 1$. Using a λ of $1/4$ would then result in a small joint motion that is scaled down and lies within the motion tolerance. In this case depending on the implementation, Visual Servoing will keep trying to converge to no avail as no robot movement is being done, or a limit on the number of time steps will be reached. In practice this increases the minimum error of the final robot position to be TOL/λ where TOL refers to the robot's tolerance.

Dealing with this control issue is not straight forward. In practice this issue will be highly dependant on the robot that is in use and how it is controlled. Note that a similar problem was encountered on the WAM when dealing with velocity control. Given the model of the robot's dynamics, the force applied to accelerate it to a given velocity was not always appropriate when commanding slow velocities. This problem was further complicated by the fact that the minimum joint velocities were not constant for different joint angles.

Increasing the λ parameter or gain can help reduce the error for both control methods. In some cases it can also cause the system to be less stable and diverge as shown in the next section. This is especially the case if the gain is set high when the residual error is also still high as big motions are created with this. As λ is increased the robot will can begin oscillating about the target, and if the set gain is too high the robot will then diverge.

5.2 Uncalibrated Visual Servoing Simulation and Experiments

In this section we take the challenges presented above and present experiments that explore their effects and give us insight into how to best handle them. First we explore camera positioning by creating a Matlab simulation environment using ROS-UVS

and compare the results of performing UVS across different configurations. Then we delve into the effects of tracker noise on the accuracy and convergence of the servoing. Finally we present repeatability experiments performed on our two WAM robotic manipulators.

5.2.1 Camera Rotation Experiment

In order to visualize the effects of camera placement several experiments were done in simulation. First, we define a world with a Puma 560 robot configured as specified by Corke in [55]. Three feature points were created and rigidly attached to the end effector. These define the visual features to be tracked on the robot. A matching set of points were placed 0.32 meters away to define a target for the UVS. Using these visual features we define a point to point task for each of them. With these constraints we generate a well defined task for the robot that covers all degrees of freedom. Virtual cameras were placed 1.0 meter behind the target position and they were offset by 30cm. A visualization of the world can be seen in figure 5.6.

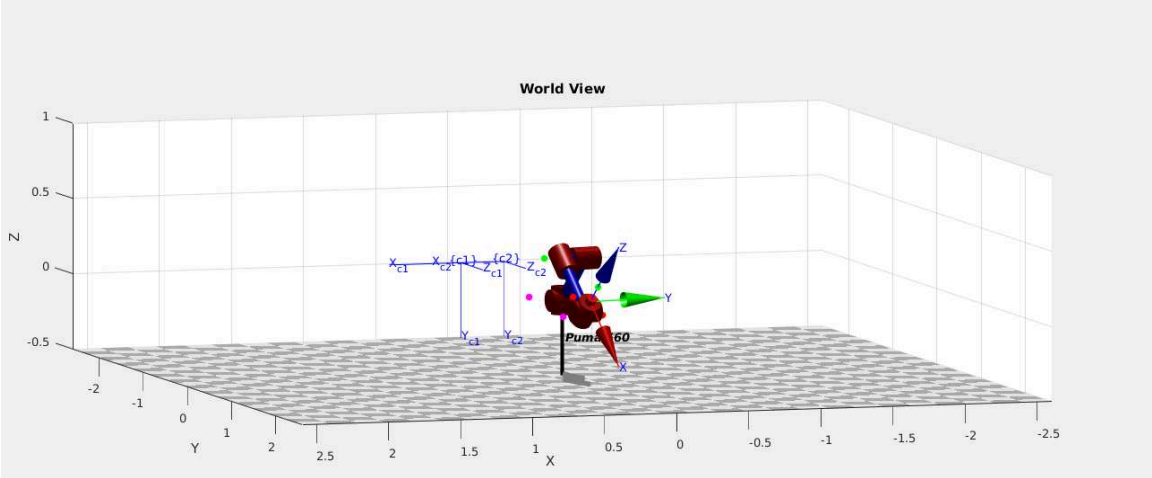


Figure 5.6: Puma 560 placed in a Matlab simulation. Three points are used to define a point to point error for Visual Servoing. Two cameras represented by the blue reference frames were placed behind the robot to observe the task.

Once configured, this simulation world was used to perform Uncalibrated Visual Servoing for 50 time steps while controlling the 6 robot joints. In order to compare

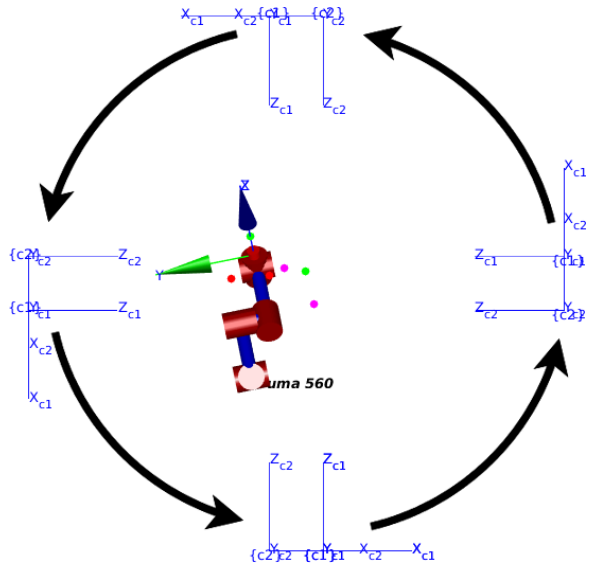


Figure 5.7: The cameras are rotated together around the target over 1 degree increments. UVS is performed for 50 time steps at each location. Here the cameras are represented by a simple reference frame with the camera center of projection down the Z axis.

the performance across multiple camera placements the camera position was rotated about the target robot location by 1 degree increments about the Z axis as shown in figure 5.7. This was then repeated while changing two parameters. First the joint motions when calculating the Jacobian were varied. Simulations were performed using motions of 2, 4, 8, and 10 degrees for each joint to calculate the Jacobian. For each of these the λ parameter was tested at $0.25, 0.50, 0.75$ and 1.00. The experiment was performed using three different camera models. First a camera that reports a precision of up to a pixel was used, this was simulated by rounding the value calculated from the camera projection. Then a camera reporting a perfect visual tracker, and finally a camera where we introduced Gaussian noise with a standard deviation of 0.2 pixels and a mean of 0.

The results of these simulations can be found in tables 5.1, 5.2, and 5.3. Here we report the mean image error and 3D errors, as well as the mean number of steps it took to reduce the image error to be below 10 pixels. In this context we report image error and 3D error as the sum of absolute values of the errors for the three

tracked visual features. When reporting the mean error values only cases which were not identified as divergent were used.

To identify diverging cases we looked at the image error and flagged simulations where the image error increased by over 100 pixels between iterations. A manual inspection of the flagged cases revealed this to be a good approximation to identify diverging cases. In some situations the Visual Servoing managed to recover, but this only occurred after several random motions resulted in the end effector moving close to the target. In a real world application this random motion would present a risk to the robot and its environment and would not be acceptable. Users would stop the robot long before it managed to recover.

From the data gathered several conclusions can be made. First, as can be seen by the mean image error and the mean 3D error, when converging there is little difference in the final position error. Second, when looking at the rate of convergence as λ increases the speed of convergence increases. This holds over all camera configurations.

We can also look at the stability of of the Visual Servoing with regards to the number of instances that diverged. When performing small motions while calculating the Jacobian the motion of the visual features in the images becomes smaller. This results in a more unstable system as can be seen when $\Delta = 2$, we can see iterations diverging at all values of λ. The pixel camera model generates the most cases that diverge, followed by the Gaussian noise, and finally having a perfect tracking system is the most stable. Although this is expected, it is surprising how much effect even a small amount of noise can have on the stability of the system.

Similarly, when $\lambda = 1.0$ we see a significant increase in the number of iterations that diverge. This relates back to equation 5.1. As λ is increased we cause bigger robot motions. This would be equivalent to having a slower camera frame rate. What causes the decrease in stability relates to the fact that when a motion occurs the likelihood that it will overshoot increases. This can cause an oscillation around the target, or if the overshoot is too large then the arm will move away from the target and diverge.

Looking at a specific example, we can further explore how rotating the cameras around the target affects performance. Detailed statistics are shown in Figure 5.8 for the case when $\Delta = 6$ and $\lambda = 0.5$ and a pixel camera is used. These graphs make it easy to visualize the relationship between properties of the camera positioning. The number of steps required to reduce the image error below 10 pixels, as shown by the *Steps to converge* graph, is mirrored by the condition number of the Jacobian. Given a higher condition number we expect the sensitivity to errors to increase. By using the pixel rounded camera model a lot of noise is introduced in the virtual tracker output. After having performed these experiments the true extent of its effects became apparent. Using trackers like this greatly affects the performance of Visual Servoing. When compared to the perfect tracker we can see that we get errors as high as 1cm, while the perfect tracker manages to reduce the 3D error well below 1mm for most camera poses.

As can be seen by the results, the behaviour of Visual Servoing when using perfect trackers with sub-pixel accuracy is much more stable. The image error was reduced to below 10 pixels in all cases within 40 steps. The image error was further reduced to almost 0 in most cases within 20 steps. Similarly the 3D error for most cases was under a tenth of a millimeter. Given more time to converge all camera positions would have converged to sub-pixel camera errors. The importance of accurate visual trackers can not be emphasised enough.

When estimating the Jacobian with small motions, introducing Gaussian noise increased the number of instances that diverged when compared to the perfect tracker. The system became most unstable in this case as the relative error between the change in pixel value when estimating the Jacobian is greater. The introduction of noise caused the Jacobian to change significantly making the system more unstable.

Surprisingly when the motions to calculate the Jacobian where larger, the number of iterations that diverged went down when compared to the perfect tracker. A possible explanation for this is that as the noise was introduced into the Jacobian, the perceived utility for each joint changed and it was averaged out. The error that was introduced made it so that joints don't dominate the motion as much. This made

the system less likely to diverge as smaller steps were taken.

As a conclusion, we can see that special care should be taken with regards to the tracker used. By analyzing the visual tracker an estimate of the pixel noise should be evaluated in the conditions in which the UVS will be performed. Knowing the noise of the visual trackers, one can then estimate the image motion required when generating the Jacobian to keep the relative error within some bounds so that a minimum precision can be reached.

Control Parameters	Mean Image Error	Mean 3D Error	Mean Steps to <10px	Iterations Diverged
Jacobian $\Delta = 2$				
$\lambda = 0.25$	8.151261	0.007660	8.900662	3
$\lambda = 0.50$	2.484507	0.004600	4.525526	5
$\lambda = 0.75$	2.544413	0.003703	2.966767	11
$\lambda = 1.00$	5.115854	0.005070	2.653979	32
Jacobian $\Delta = 4$				
$\lambda = 0.25$	5.445682	0.005613	8.813953	1
$\lambda = 0.50$	1.242340	0.003169	4.640805	1
$\lambda = 0.75$	1.500000	0.003176	3.184971	4
$\lambda = 1.00$	2.800000	0.003586	2.752613	45
Jacobian $\Delta = 6$				
$\lambda = 0.25$	5.433333	0.005976	8.834983	0
$\lambda = 0.50$	0.938889	0.002876	4.658046	0
$\lambda = 0.75$	0.902778	0.002811	3.207977	0
$\lambda = 1.00$	3.875776	0.004209	2.843636	38
Jacobian $\Delta = 8$				
$\lambda = 0.25$	6.216667	0.005973	8.840532	0
$\lambda = 0.50$	1.047222	0.002960	4.662824	0
$\lambda = 0.75$	0.986111	0.003000	3.323864	0
$\lambda = 1.00$	3.531250	0.004253	2.925000	40
Jacobian $\Delta = 10$				
$\lambda = 0.25$	5.952778	0.006031	8.817881	0
$\lambda = 0.50$	0.941667	0.002883	4.691429	0
$\lambda = 0.75$	0.994444	0.002940	3.292264	0
$\lambda = 1.00$	3.088608	0.003802	2.945055	44

Table 5.1: Resulting error convergence properties from rotating the cameras around a point to point task. For these simulations the simulated trackers reported integer pixel precision by rounding to the nearest pixel.

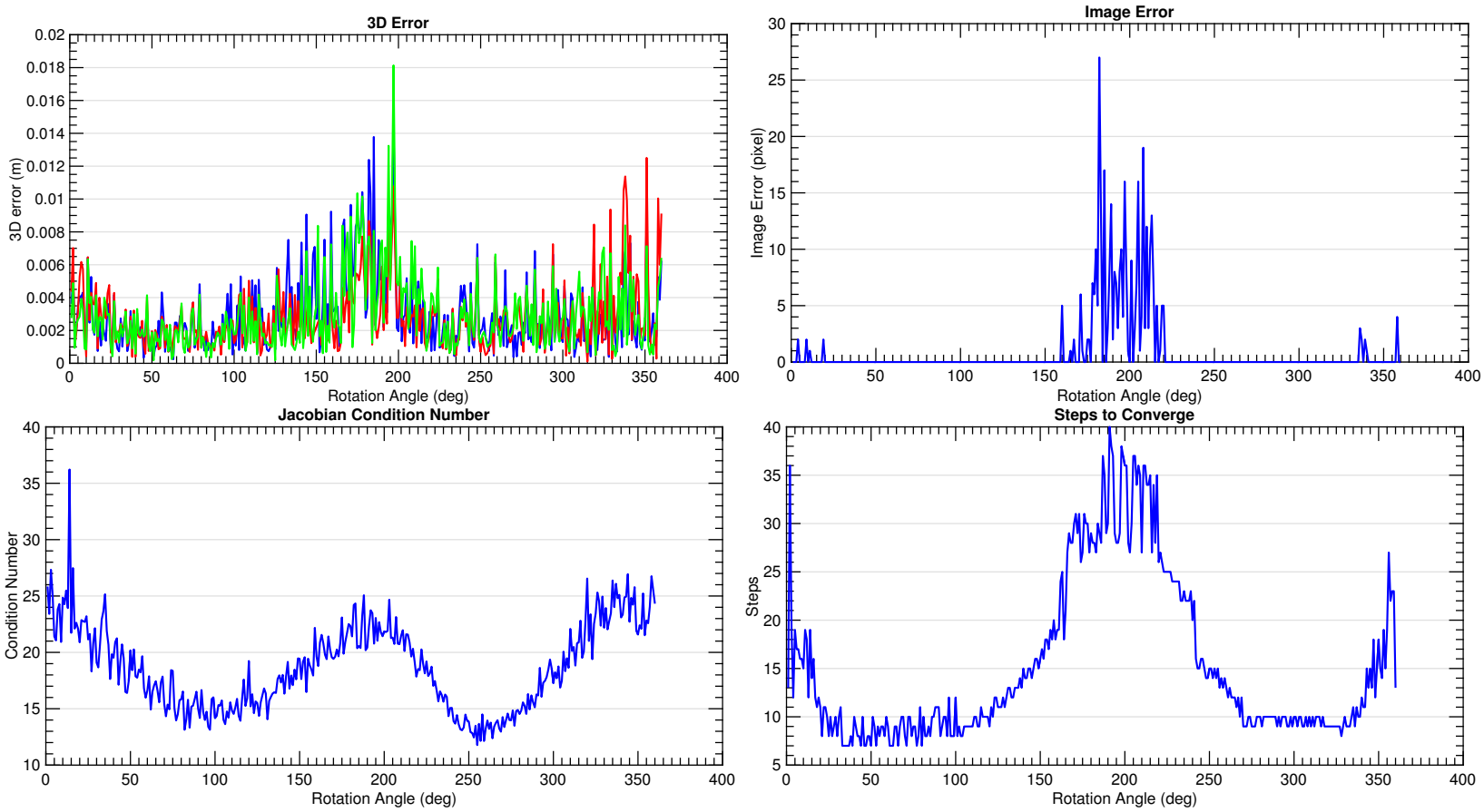


Figure 5.8: Detailed graphs of image and 3D error as well as Jacobian condition number and steps to convergence for a camera rotation simulation experiment where $\Delta = 6$ and $\lambda = 0.5$. Cameras in this simulation reported integer pixel precision by rounding to the nearest pixel.

Control Parameters	**Mean Image Error**	**Mean 3D Error**	**Mean Steps to <10px**	**Iterations Diverged**
Jacobian $\Delta = 2$				
$\lambda = 0.25$	5.264595	0.003251	8.855263	0
$\lambda = 0.50$	0.546380	0.000382	4.738889	0
$\lambda = 0.75$	0.171979	0.000165	3.211111	0
$\lambda = 1.00$	1.042769	0.000614	3.161290	43
Jacobian $\Delta = 4$				
$\lambda = 0.25$	5.456003	0.003420	8.844884	0
$\lambda = 0.50$	0.594838	0.000420	4.752778	0
$\lambda = 0.75$	0.198895	0.000187	3.255556	0
$\lambda = 1.00$	0.585607	0.000438	3.098361	52
Jacobian $\Delta = 6$				
$\lambda = 0.25$	5.644139	0.003586	8.831683	0
$\lambda = 0.50$	0.643409	0.000460	4.766667	0
$\lambda = 0.75$	0.228489	0.000212	3.302778	0
$\lambda = 1.00$	0.818172	0.000629	3.102310	53
Jacobian $\Delta = 8$				
$\lambda = 0.25$	5.831923	0.003751	8.808581	0
$\lambda = 0.50$	0.691667	0.000504	4.777778	0
$\lambda = 0.75$	0.261425	0.000241	3.330556	0
$\lambda = 1.00$	0.625176	0.000468	3.099338	57
Jacobian $\Delta = 10$				
$\lambda = 0.25$	6.021153	0.003916	8.798680	0
$\lambda = 0.50$	0.739442	0.000552	4.772222	0
$\lambda = 0.75$	0.298541	0.000273	3.358333	0
$\lambda = 1.00$	0.913177	0.000645	3.145695	55

Table 5.2: Resulting error convergence properties from rotating the cameras around a point to point task. For these simulations the trackers reported sub-pixel accuracy.

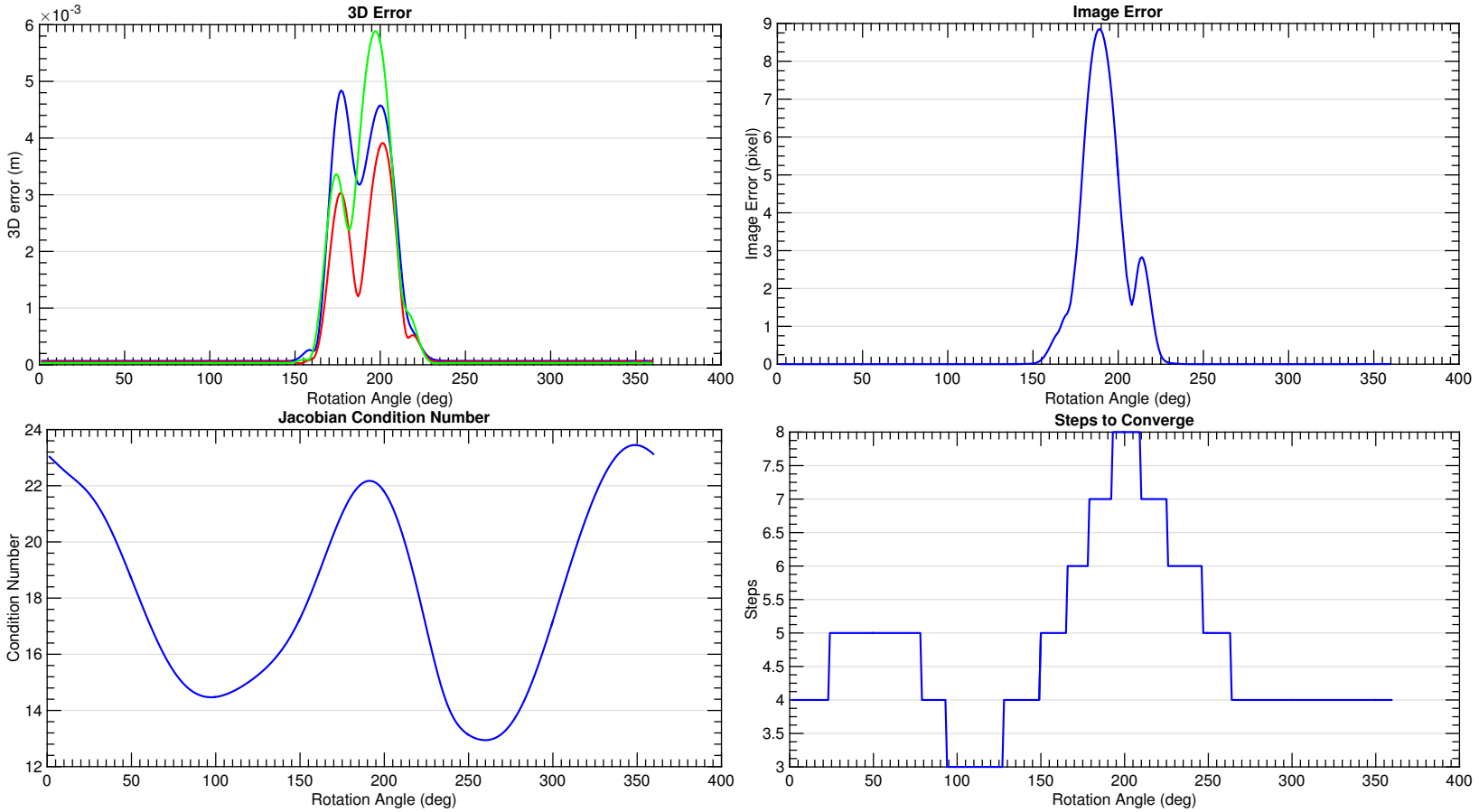


Figure 5.9: Detailed graphs of image and 3D error as well as Jacobian condition number and steps to convergence for a camera rotation simulation experiment where $\Delta = 8$ and $\lambda = 0.5$. For these simulations the trackers reported sub-pixel accuracy.

Control Parameters	Mean Image Error	Mean 3D Error	Mean Steps to <10px	Iterations Diverged
Jacobian $\Delta = 2$				
$\lambda = 0.25$	8.868672	0.006711	8.769231	1
$\lambda = 0.50$	6.266691	0.004368	4.485207	3
$\lambda = 0.75$	6.071691	0.004064	2.996970	12
$\lambda = 1.00$	6.527398	0.003631	2.540268	31
Jacobian $\Delta = 4$				
$\lambda = 0.25$	7.145731	0.005205	8.776667	0
$\lambda = 0.50$	3.809346	0.002252	4.661017	1
$\lambda = 0.75$	4.435710	0.002510	3.135057	2
$\lambda = 1.00$	5.905690	0.003537	2.915033	26
Jacobian $\Delta = 6$				
$\lambda = 0.25$	8.374407	0.005089	8.827815	0
$\lambda = 0.50$	4.006011	0.002318	4.662857	0
$\lambda = 0.75$	4.521807	0.002685	3.197708	1
$\lambda = 1.00$	6.235470	0.003426	2.919861	39
Jacobian $\Delta = 8$				
$\lambda = 0.25$	7.860871	0.004988	8.794020	0
$\lambda = 0.50$	3.998573	0.002287	4.667614	0
$\lambda = 0.75$	4.410463	0.002336	3.289773	0
$\lambda = 1.00$	5.754420	0.003414	3.035211	44
Jacobian $\Delta = 10$				
$\lambda = 0.25$	8.266994	0.005281	8.771523	0
$\lambda = 0.50$	4.132635	0.002533	4.648415	0
$\lambda = 0.75$	4.324643	0.002491	3.284507	0
$\lambda = 1.00$	6.246298	0.003606	2.985455	44

Table 5.3: Resulting error convergence properties from rotating the cameras around a point to point task. For these simulations the trackers introduced a small amount of Gaussian noise with a mean of 0 and a standard deviation of 0.2.

5.2.2 Camera Baseline Rotation Experiment

In the previous experiment the cameras were rotated about the target point while keeping the distance between them constant. In this experiment we rotated one camera only. This changes the baseline distance and angle of view between the cameras with respect to the target. The outcome of this experiment is shown in figure 5.10. In these graphs a negative value is placed wherever the system diverged. In this iteration the experiment was run with a $\Delta = 6$ and $\lambda = 0.50$. The relationship between the Jacobian condition number and the image error remains. As the condition number increases, the longer it takes for the Visual Servoing to converge.

The Jacobian condition number fluctuates based on the rotational baseline between the two cameras. As can be seen when the cameras are placed orthogonally the condition number is at its minimum. When the cameras overlap the Visual Servoing diverges as the system is left with only one point of view of the scene.

In practice this suggests that placing the cameras with a large baseline rotational difference will give the best results. The error is minimized faster and thus after the 50 steps of servoing we get a better final position. Logically this makes sense, when the cameras are rotated separately from each other around the target, the view along the camera center of projection of one camera is better covered by the other. This better coverage of the depth dimension leads to a more stable system.

Although it would be ideal to be able to set-up the cameras in an optimal position this is not always possible. By having a 90 degree offset it becomes hard to view the same surface on both cameras. This makes tracking challenging when trying to follow a planar surface for instance the image will be skewed and as the surface rotates away from the camera the area it covers in the image decreases. As the area decreases the tracking becomes harder and it is more likely to get lost.

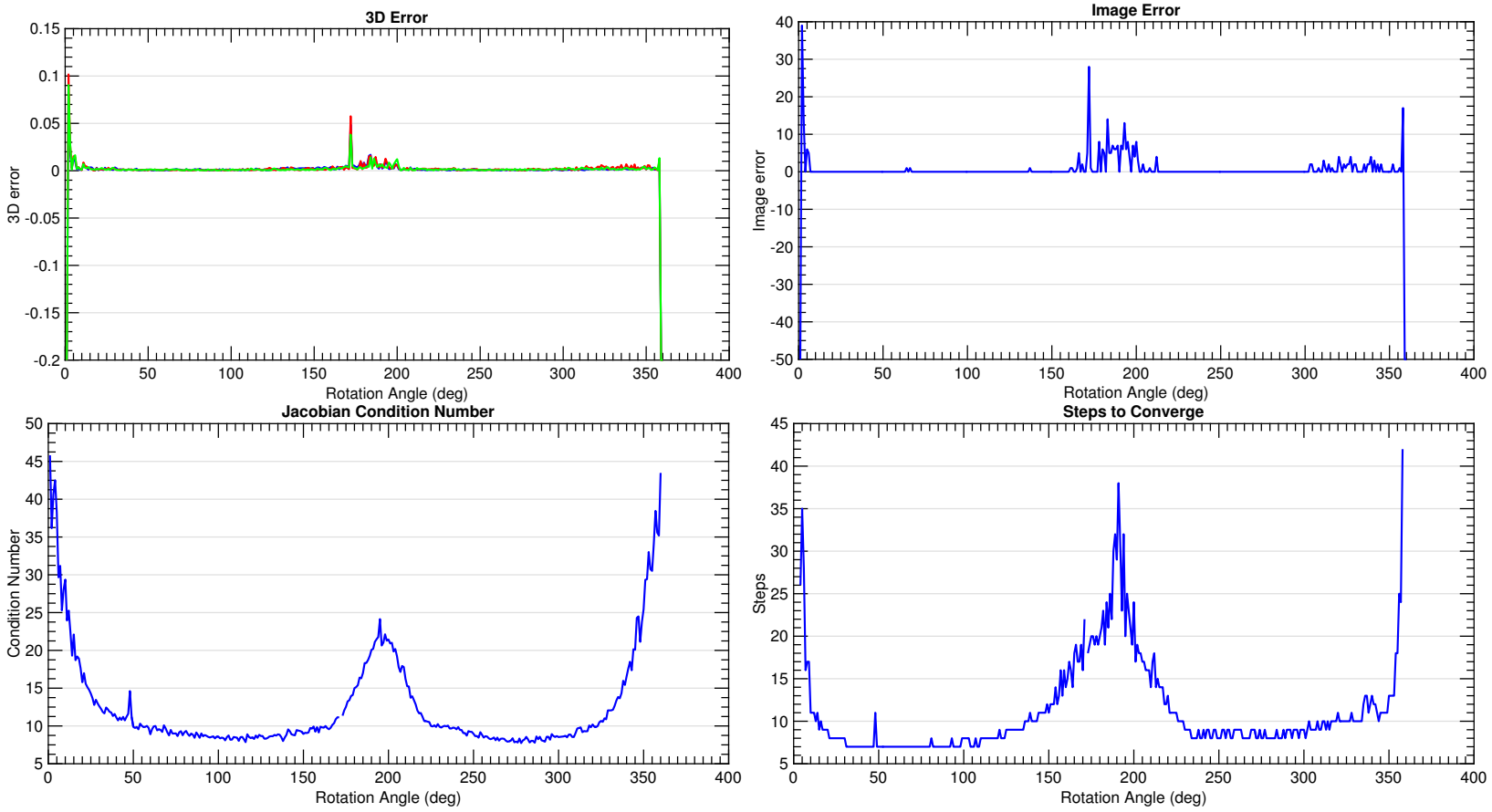


Figure 5.10: Detailed graphs of image and 3D error as well as Jacobian condition number and steps to convergence for a camera rotation simulation experiment where $\Delta = 8$ and $\lambda = 0.5$

5.2.3 Robot Repeatability Experiments

With the lessons learned from the previous experiments, regarding camera placement and Visual Servoing parameters, we set out to measure the repeatability on our WAM robot. The expectation is that the repeatability achieved by the Visual Servoing system should exceed that of the internal robot controllers. The external sensing from the cameras should be able to correct any internal errors within the robot. However in previous attempts to measure repeatability within our group with our WAM robot, Visual Servoing had failed to surpass the internal robot control.

To facilitate this experiment and test the limits of Visual Servoing on our robot we first performed a simple servoing experiment. Using a 0.001 inch dial meter we measured the 3D error when performing a motion of a single joint of the robot. The camera and robot configuration can be seen in figure 5.11.

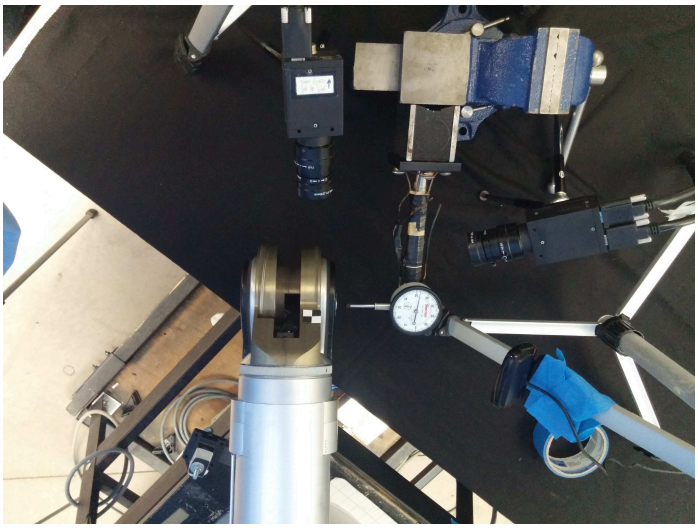


Figure 5.11: Single joint robot repeatability experiment setup

The cameras were placed with close to an orthogonal position with respect to each other. The target was located 20 cm away from the cameras. The first joint was then rotated and a flat surface on the elbow of the arm was used to press on the dial meter. Two different control options were used to compare the robot's internal repeatability to the Uncalibrated Visual Servoing.

First the arm was manually moved to the starting position where the dial meter

	Robot Motion Reported Pose	Robot Motion Commanded Pose	Visual Servoing
Mean Error (mm)	0.163	0.093	0.038
Std Variation	0.055	0.027	0.022
Variance	0.120	0.029	0.018

Table 5.4: Repeatability results for one joint control on the WAM

was placed. Then the position reported by the arm's encoders was saved. The arm was moved away to a position 3 degrees away from the starting position and a uniformly random ± 1 degree noise in the joint position was introduced. This was done to introduce more variation in the motion required by the arm when completing the task. The arm was then commanded back to the starting position which was saved initially.

While controlling the arm this way we noticed that there seemed to be a fairly consistent displacement. We attribute this to two things. First the dial meter exerts a small force on the arm as it is pressed with the arm's motion. This pushes the arm away causing a small increase in the error. Furthermore it appears like there exist small internal errors in the calibration between the arms commanded joint pose and the reported joint pose.

To validate this, we then repeated the experiment by first commanding the robot to a starting position. Once the arm had moved, we placed the dial meter and recorded the initial measurement. Then the arm was again commanded to move away and back to it's initial position. Note the difference here is that the initial position was one that was commanded before, rather than one that was reported by the arm. Performing the measurements this way yielded a slightly reduced error measurement. We attribute this to a reduction on the error introduced by the force required to push the dial in.

Finally Uncalibrated Visual Servoing was performed. Using 2 visual markers and point to point errors Visual Servoing was executed until convergence which we identified by looking at the dial and verifying no more movement was occurring. The trackers were reinitialized on every iteration to prevent errors arising from the tracker drifting from the motion of the arm in between trials. The motions performed to

	Robot Motion Reported Pose	Robot Motion Commanded Pose	Visual Servoing
Mean Error (mm)	0.165	0.132	0.036
Std Variation	0.032	0.031	0.015
Variance	0.041	0.038	0.008

Table 5.5: Repeatability results for 4DOF joint control on WAM

generate the Jacobian matrix were performed by displacing the arm by 3 degrees and adding the same ± 1 degree displacement. While executing the Visual Servoing λ was set to 0.50. These measurements were repeated over 10 trials for all cases.

The resulting error is presented in table 5.4. Using Uncalibrated Visual Servoing we were able to achieve better accuracy than the repeatability of the internal robot controllers. However in order to achieve this we required the cameras to be very close to the regions of interest. Placing the cameras so close to the target is not always practical. In some situations like this eye-in-hand camera placements might be preferred as the robot can make the cameras move to a closer position allowing for a higher resolution view of the scene.

Knowing that we were able to outperform the robot's controls, we then set out to test the system when using 4DOF. The experiment setup can be seen in figures 5.12 and 5.13. Cameras were placed with a baseline distance of 30cm. The target position was located 50cm away from the cameras. Three points from each camera view point were tracked on playing cards attached to the robot. These were used as they have a rich texture that made it easy to accurately track. The results for this experiment are summarized in table 5.5. Again we were able to successfully outperform the internal robot's control.

When we began performing these experiments the 3D position error was not being reduced fully as expected. This was a result of the control limitations we describe in section 5.1.3. A simple heuristic that allowed us to decrease the final positioning error with our robot was to drive the robot to convergence until the control limitations came into place. Once there, a couple of visual servoing steps were performed with a doubled λ value. This increased the commanded motion above the minimum motion threshold we experience in our robot and allowed the system to reach a better final

Figure 5.12: Robot repeatability for 4 DOF experiment setup. Red lines outline the region tracked with the centroid of the quadrilateral representing a point of interest. Point to point constraints are defined between these centroids and the green points.

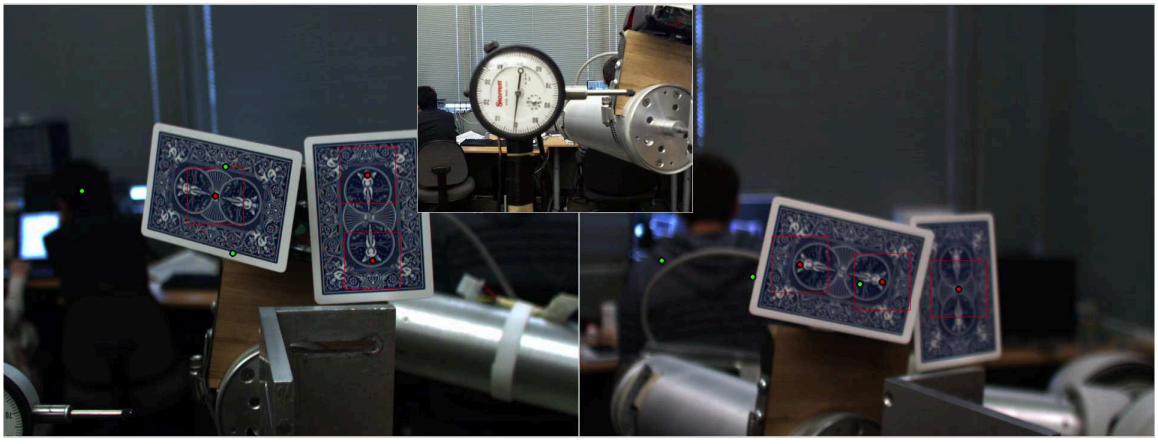


Figure 5.13: Robot repeatability for 4 DOF camera views

position. This final movement improved the position of the robot by about 0.1-0.2mm.

Figure 5.14: Robot repeatability for 4 DOF robot with eye-in-hand camera configuration

As a comparison point we also performed the repeatability experiment with an eye-in-hand configuration. The cameras were rigidly mounted on the robot with a 40cm offset. The target was placed 35cm away from the final robot pose. The robot configuration can be seen in figure 5.14. The resulting measurements are shown in table 5.6. This experiment was performed in two ways. First using both cameras in a stereo configuration, and then using only the left camera.

This configuration yielded interesting results as the mean error between the stereo and monocular vision systems was almost the same. The main difference however can be seen on the variance. In the case of the monocular camera configuration the variance is almost double that of the stereo case.

	Robot Motion Reported Pose	Robot Motion Commanded Pose	Visual Servoing Stereo Vision	Visual Servoing Mono Vision
Mean Error (mm)	0.155	0.099	0.036	0.038
Std Variation	0.039	0.037	0.027	0.036
Variance	0.059	0.053	0.029	0.052

Table 5.6: Repeatability results for 4 DOF joint control on WAM with eye-in-hand camera configuration

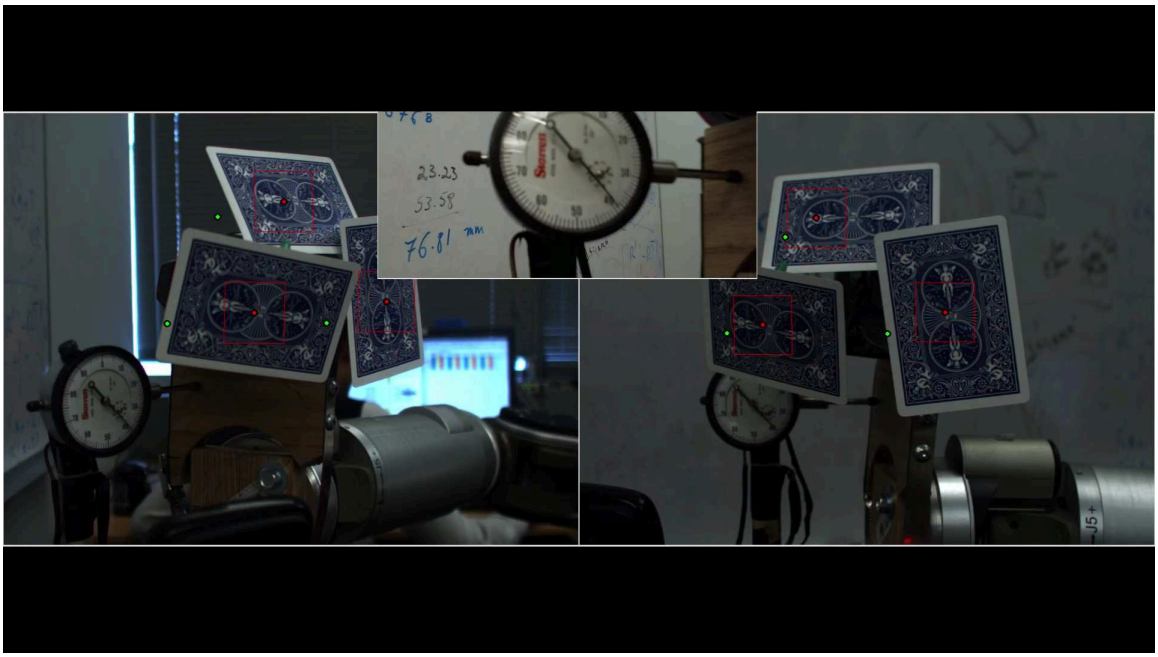


Figure 5.15: Robot repeatability for 6 DOF camera views

	Robot Motion Reported Pose	Robot Motion Commanded Pose	Visual Servoing
Mean Error (mm)	0.472	0.140	0.093
Std Variation	0.088	0.119	0.0951
Variance	0.306	0.555	0.3556

Table 5.7: Repeatability results for 7DOF joint control on WAM

Finally we performed a similar configuration with our 7DOF WAM arm. Here the achieved accuracy was not as high as with the 4DOF arm. The results of our experiments with this arm are shown in table 5.7.

There are several sources of error when performing the Uncalibrated Visual Servoing for these experiments. One such problem is that the robot is not completely rigid, thus any non rigid motion in the joints will cause the robot configuration to shift. This causes a discrepancy in two places. First it introduces error in the measurement of the final pose at the dial as the robot configuration changes. And in the case of the visual servoing, the specified targets are also shifted causing the target pose of the visual servoing to be different from the expected one. Another source of error comes from measuring the final 3D position in one direction at a time. Given the robot motion it is not easy to measure the full robot pose externally.

The WAM specifications datasheet [56] reports repeatability measures of 0.1mm and 0.2mm for the 4 and 7DOF configurations respectively when the encoder option is installed as is the case with the robots used in these experiments. In [57] Barrett reports repeatability experiments performed by them where the precision reached averaged between 0.042mm and 0.078mm for large and small movements respectively on a 4DOF arm. Although we performed the robot calibration before performing our own tests, we were not able to achieve such high precision using the robot's control for the 4DOF arm. However our experiments using Uncalibrated Visual Servoing were still able to outperform their reported performance in our tests using the 4DOF configuration.

5.2.4 Tracker Noise Effect Experiment

As a final test we investigate the effects of tracker noise. To further validate our system we perform a comparison between our 4DOF WAM and the simulation system created. For this purpose we created a model of the 4DOF WAM using the Robot Toolbox [42] as described by the Denavit-Hartenberg parameters [58] provided by Barrett at [59]. We extend the end effector position of the arm by 3.5 cm to account for the tool attachment equipped on our robot. Figure 5.16 illustrates our experimental

setup with the physical robot and it's simulation counterpart.

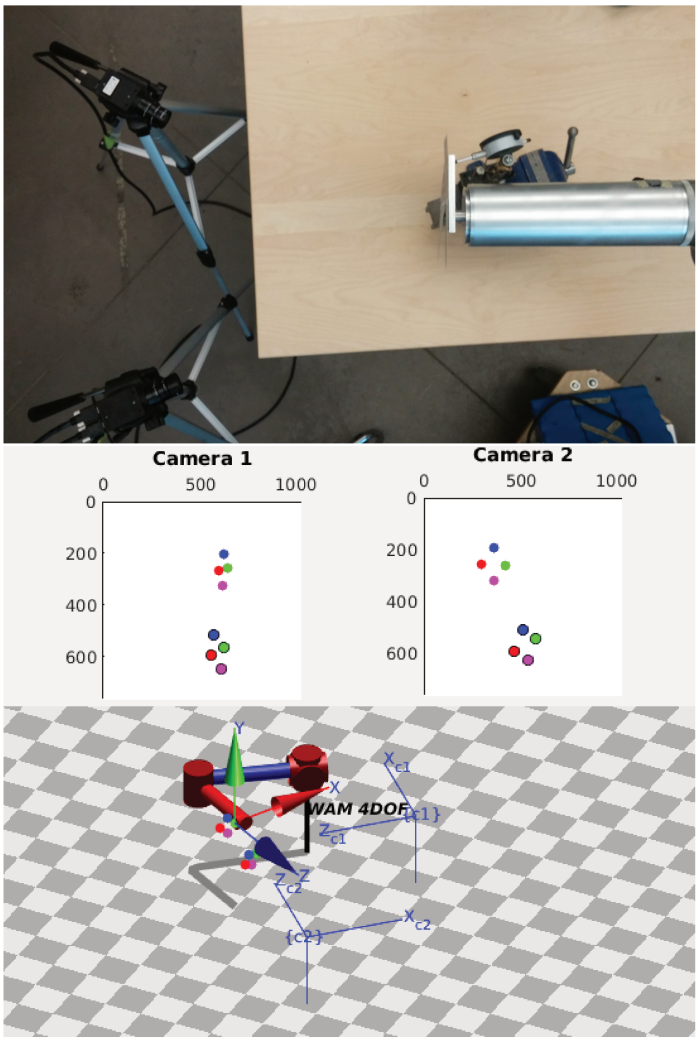
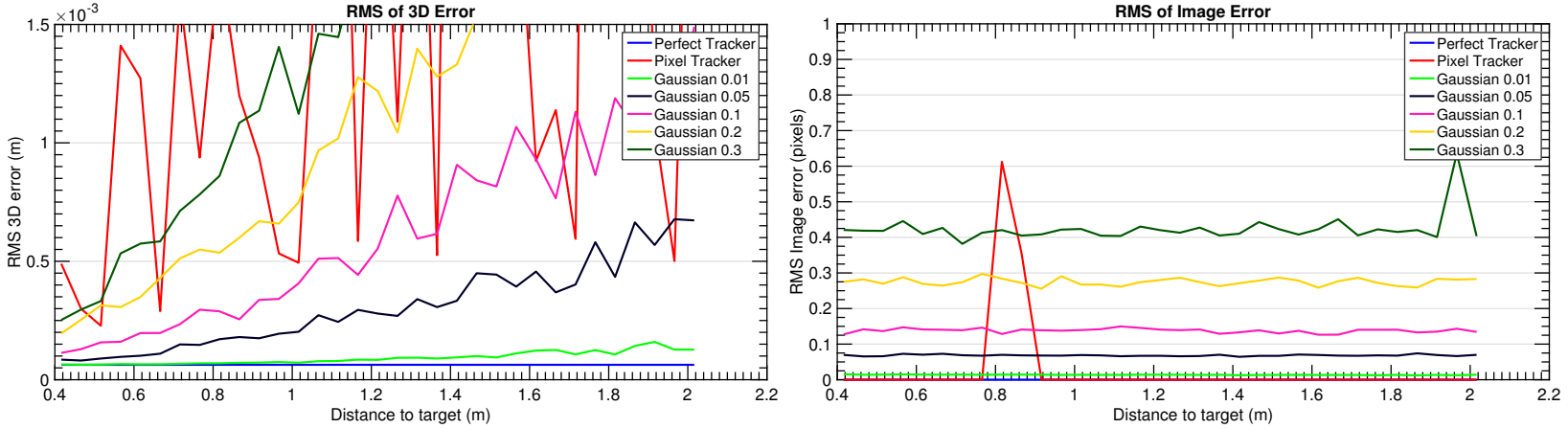


Figure 5.16: Robot configuration for calibrated simulation system experiments. The simulated robot and world are configured to mirror exactly the real world robot

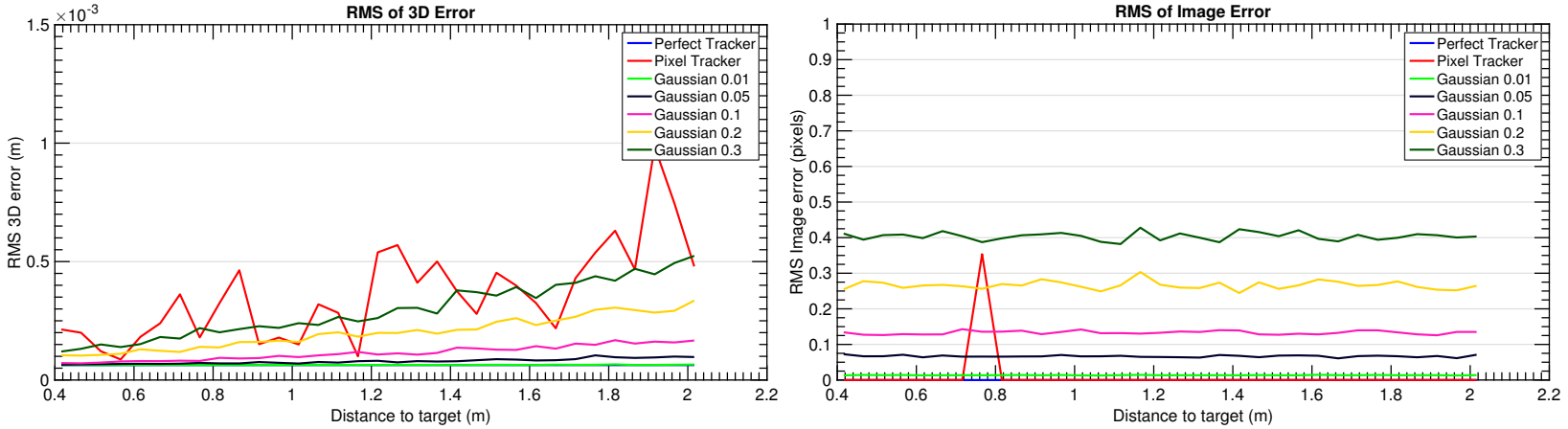
Having a proper simulated robot that mirrors our own we then calibrated our camera system. Using the Camera Calibration package [60] from ROS we found the intrinsic parameters for our cameras. This system reports the projection matrix as implemented by OpenCV [61]. The calibration is based on the methods described by Zhang et al. in [62]. Once calibrated, both the robot motions, and the resulting image feature changes are mirrored between the physical robot and the simulated world.

In this configuration we performed a simulation where we compare three camera models as the cameras are moved away from the specified target. We compare cameras with trackers reporting perfect accuracy, accuracy with added Gaussian noise with a standard deviation of 0.2 and a mean of 0, and a tracker reporting integer pixel precision. The cameras were moved away from the target starting at a position 40cm away and ending at a position 2m away. This was performed by moving the camera backwards at 0.05m increments. We repeated the experiment with three camera configurations. First with the cameras aligned and facing the same direction while having a 40cm baseline offset. In the second configuration the cameras were placed with a 90 degree offset while viewing the task target. Finally the experiment was repeated with a single camera.

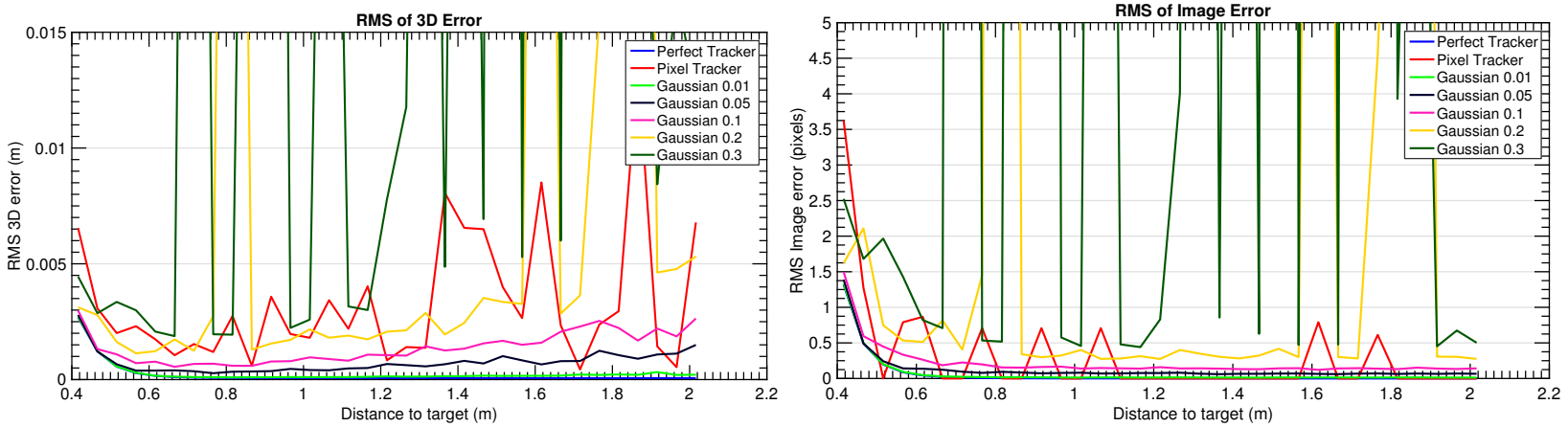
The servoing was performed for 500 time steps with a step size of $\lambda = 0.05$. The trials were performed 50 times and the outcomes were averaged. As discussed previously this creates a system that is equivalent to having a fast camera frame rate. The 3D and image error are shown in 5.17. From this simulation we found that the final image error was fairly consistent between the two camera configurations. The final 3D error however increased as the cameras were placed further away from the target. Similarly as the Gaussian noise was increased the speed of convergence for the system decreased. In the case of a pixel tracker the servoing tended to be inconsistent. When performing the experiment with larger values of λ we saw cases where the system would diverge when the camera was only about 1 meter away. These divergent cases became more frequent as the tracker noise increased. Using a single camera the effects of noise in the visual trackers were more pronounced. When introducing a noise with a standard deviation of 0.3 divergence occurred when the cameras were placed at 0.6m from the target. Also worth noting is that this camera configuration caused the servoing to converge slower. When the camera was close to the target (up to abuot 0.6m) the error is significantly larger. This is reflected in the image error where convergence was not achieved after 500 simulation steps.



(a) Cameras with 40cm offset



(b) Cameras with 90 degree offset



(c) Single Camera

Figure 5.17: 3D error and image error is shown when performing UVS with visual trackers with different noise properties.

	Robot Motion Reported Pose	Robot Motion Commanded Pose	Visual Servoing 40cm from target	Visual Servoing 55cm from target
Mean Error (mm)	0.1575	0.1245	0.0737	0.1321
Std Variation	0.0262	0.0222	0.0736	0.1509
Variance	1.0666	0.7666	0.2134	0.8963

Table 5.8: Repeatability results for 4 DOF WAM in calibrated simulation configuration

Having done the simulation we performed the experiment with our 4 DOF WAM arm. The trials were performed with 2 cameras placed with close to a 90 degree baseline rotation. The cameras were placed at a distance of 40cm and 55cm. Moving the cameras further away resulted in poor tracker performance. Only 1 iteration out of 5 finalized the exploratory motions when estimating the Jacobian without having trackers fail. They were lost shortly after the servoing began. The results for these trials are shown in table 5.8.

The results from performing the same experiment on the real robot allow us to make some conclusions. The error was definitely larger when the cameras were placed further apart. However the increase was much larger than expected. This can be explained by a fast increase in tracking error as the cameras were moved away from the target. As mentioned previously, moving the cameras further back from the 55cm mark made the tracking almost impossible. From this we can learn that although the error from performing UVS will tend to grow as we move further away from the target, in reality this growth will be much more apparent given the propensity for trackers to begin failing when the target is further away.

5.3 Conclusions and Lessons Learned

As we tackled the three challenges outlined in this chapter we have learned several things. The placement of the cameras with respect to the target is not as important as the placement of the cameras with respect to each other. From the camera positioning experiments we saw that as the cameras rotated around the target together the 3D error increased by about 0.006m. On the other hand when changing the camera rotational position with respect to each other the error increased by about a factor of

10 to 0.06m. Given this information we can then prioritize the positioning such that the cameras have a good view of the robot and the target of the servoing. Once placed we should try to make them stand as close to an orthogonal position as possible.

The repeatability we were able to achieve with the real robot when optimizing for accuracy surpassed the figures reported by Barrett. In our case we were not able to match their reported values when using the internal robot controllers. Using UVS however we performed better. This improvement attests to the precision that can be reached by using visual sensing to drive the robot. As an added bonus UVS was able to compensate for the external force that was exerted by the measuring dial that we used. UVS in general seems to be robust to internal robot calibration issues and unaccounted external forces acting on the robot. We did encounter difficulty when performing small motions with the robot as directed by UVS. We were able to compensate for this by modifying the gain λ when the position of the robot was close to the target. This could be automatically identified by looking at the residual error and using that as a guide to tune the control gain.

Using a calibrated simulation environment allowed us to validate the modeling of the cameras and the robot motion with respect to the real system. We also learned that the simulation of the tracker noise is not consistent as the noise changes in the real world with respect to the distance to the cameras. In our current simulation system we are using a constant amount of noise which is not entirely realistic. This however a good way to find a baseline of best case scenarios given constrained levels of noise in the visual trackers. With this knowledge we can see that the graphs portrayed in figure 5.17 are very conservative. In reality the increase in the expected error is much more steep with respect to the tracker noise. However as shown by the experiments on the 4DOF WAM robot with an eye-in-hand configuration, we can see the increase in variation on the resulting pose when using only one camera.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In our work we presented ROS-UVS. Our minimalistic framework for Uncalibrated Visual Servoing. Our hope is that through this library we can aid in the adoption of UVS for practical applications. ROS-UVS includes a proven implementation that allows users to perform Uncalibrated Visual Servoing. It provides routines that estimate the full non-parametric Jacobian of the robot system without requiring a-priori calibration or object models. We provide a general interface that is easy to adapt across different robots and environment configurations. By creating this base framework users can focus on their system specific requirements and can easily adapt the sample controllers we provide to connect our system to their robot. Developed with ROS integration in mind, ROS-UVS is easy to integrate and run within the ROS infrastructure. We also understand that ROS is not for everyone and our system can also be run as a stand alone library.

Our system is free and open-source and adheres to design principles outlined in other software libraries such as ROS. The result of our design is a standalone library that has no external dependencies. This allows for code extraction and reuse beyond its original intent.

In using our system we have been able to perform UVS across several different platforms. Through simulation we were also able to explore challenges with respect to camera placement and tracker noise. We also used the simulation environment

alongside the physical robot and validated the robot motions and virtual camera implementation. Through our system we managed to facilitate the use of UVS and improve our understanding and intuitions of this robot control approach.

6.2 Future Work

Identifying and reacting to failing trackers would an interesting direction to expand this work in. In general it would require the system to reconfigure the Jacobian and error vectors on the fly for the lost trackers. Users would also have to define over specified tasks so that the servoing can continue and perhaps stop once the task is not specified fully for all the DOF being controlled or if the task is completed.

In [63] Samson defines redundancy in a robotic system when a defined task can be achieved in an infinite number of ways and the solution set forms a dense set in the joint space. Using this concept we can then define primary and secondary tasks where the secondary task takes place within the solution space of the primary task. Formalizing this idea in the context of visual servoing and integrating it with the robot control provided is another area we would wish to pursue further.

During development of the library we shortly tried visually defining virtual fixtures similarly to how it was done in [64]. In their work they use the XVision [65] system to create visual trackers that sense tangent directions to defined paths. Several problems make this a hard task, however developing a stable extension for our library that supports such features would be great. For instance, in their system they sample images at 30 and 60hz, however in studies of haptic interfaces it has been shown that humans can sense up to about 300hz [66]. This suggests that natural haptic interfaces require a high update rate. Having visual trackers perform at such a high frequency is not easy. Further more quick snappy motions of the arm can cause trackers to be lost. In our lab we have tested doing these actions at 120hz with some success. However more work is required in this area.

Another great addition to the framework would be path planning integration. Kazemi et al. [67] present a survey of planning algorithms for visual servoing. They

classify planning approaches into four different categories: (1) image-space planning, (2) optimization-based planning, (3) potential field-based planning, and (4) global path-planning. Integrating these within the current framework would allow for a more stable and robust system that could be trusted to maintain the regions of interest within the cameras field of view.

Bibliography

[1] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software* 3.3.2 (2009), p. 5.

[2] C.C. Kemp, A Edsinger, and E. Torres-Jara. "Challenges for robot manipulation in human environments [Grand Challenges of Robotics]". In: *Robotics Automation Magazine, IEEE* 14.1 (2007), pp. 20–29.

[3] *Amazon Picking Challenge*. 2015. URL: `http://amazonpickingchallenge.org/` (visited on 10/2015).

[4] Nikolaus Correll et al. "Lessons from the Amazon Picking Challenge". In: *arXiv preprint arXiv:1601.05484* (2016).

[5] Adam Eric Leeper et al. "Strategies for human-in-the-loop robotic grasping". In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*. ACM. 2012, pp. 1–8.

[6] Matei Ciocarlie et al. "Mobile manipulation through an assistive home robot". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 5313–5320.

[7] Tiffany L Chen et al. "Robots for Humanity: Using Assistive Robots to Empower People with Disabilities". In: (2013).

[8] Sebastian Muszynski, J Stuckler, and Sven Behnke. "Adjustable autonomy for mobile teleoperation of personal service robots". In: *RO-MAN, 2012 IEEE*. IEEE. 2012, pp. 933–940.

[9] Kinova Robotics. *Jaco*. URL: `http://kinovarobotics.com/products/jaco-robotics/` (visited on 02/16/2016).

[10] Camilo Perez, Oscar Ramirez, and Martin Jagersand. "VIBI: Assistive Vision-Based Interface for Robot Manipulation". In: (2015).

[11] Hairong Jiang, J.P. Wachs, and B.S. Duerstock. "Integrated vision-based robotic arm interface for operators with upper limb mobility impairments". In: *Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on*. 2013, pp. 1–6. DOI: `10.1109/ICORR.2013.6650447`.

[12] Camilo Perez et al. “Small Object Manipulation in 3D Perception Robotic Systems Using Visual Servoing”. In: (2014).

[13] Mona Gridseth et al. “ViTa: Visual Task Specification Interface for Manipulation with Uncalibrated Visual Servoing”. In: *IEEE International Conference on Robotics and Automation*. IEEE. 2016.

[14] Mona Gridseth. “Visual Task Specification User Interface for Uncalibrated Visual Servoing”. MA thesis. University of Alberta, 2015.

[15] Peter Corke et al. “The Machine Vision Toolbox: a MATLAB toolbox for vision and vision-based control”. In: *Robotics & Automation Magazine, IEEE* 12.4 (2005), pp. 16–25.

[16] Eric Marchand, Fabien Spindler, and François Chaumette. “ViSP for visual servoing: a generic software platform with a wide class of robot control skills”. In: *Robotics & Automation Magazine, IEEE* 12.4 (2005), pp. 40–52.

[17] Barrett Technology Inc. *WAM ARM*. URL: `http://www.barrett.com/products-arm.htm` (visited on 10/02/2015).

[18] Seth Hutchinson, Gregory D Hager, Peter Corke, et al. “A tutorial on visual servo control”. In: *Robotics and Automation, IEEE Transactions on* 12.5 (1996), pp. 651–670.

[19] François Chaumette and Seth Hutchinson. “Visual servo control. I. Basic approaches”. In: *Robotics & Automation Magazine, IEEE* 13.4 (2006), pp. 82–90.

[20] Gerald J Agin. *Real time control of a robot with a mobile camera*. SRI International, 1979.

[21] Yoshiaki Shirai and Hirochika Inoue. “Guiding a robot by visual feedback in assembling tasks”. In: *Pattern recognition* 5.2 (1973), pp. 99–108.

[22] Won Jang et al. “Concepts of augmented image space and transformed feature space for efficient visual servoing of an’eye-in-hand robot’”. In: *Robotica* 9.2 (1991), pp. 203–212.

[23] Zachary Dodds et al. “Task specification and monitoring for uncalibrated hand/eye coordination”. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1607–1613.

[24] BJF Driessen et al. “Collaborative control of the manus manipulator”. In: *Universal Access in the Information Society* 4.2 (2005), pp. 165–173.

[25] Katherine Tsui et al. “Development and evaluation of a flexible interface for a wheelchair mounted robotic arm”. In: *Proceedings of the 3rd ACM/IEEE international conference on Human robot interaction*. ACM. 2008, pp. 105–112.

[26] Koh Hosoda and Minoru Asada. "Versatile visual servoing without knowledge of true jacobian". In: *Intelligent Robots and Systems' 94.'Advanced Robotic Systems and the Real World', IROS'94. Proceedings of the IEEE/RSJ/GI International Conference on*. Vol. 1. IEEE. 1994, pp. 186–193.

[27] Martin Jägersand, Olac Fuentes, and Randal Nelson. "Experimental Evaluation of Uncalibrated Visual Servoing for Precision manipulation". In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE. 1997, pp. 2874–2880.

[28] Herry Sutanto, Rajeev Sharma, and Venugopal Varma. "The role of exploratory movement in visual servoing without calibration". In: *Robotics and Autonomous Systems* 23.3 (1998), pp. 153–169.

[29] Gary R Bradski. "Computer vision face tracking for use in a perceptual user interface". In: (1998).

[30] Ankush Roy et al. "Tracking Benchmark and Evaluation for Manipulation Tasks". In: (2015).

[31] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. "Online object tracking: A benchmark". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2013, pp. 2411–2418.

[32] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. "Tracking-learning-detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.7 (2012), pp. 1409–1422.

[33] Carlo Tomasi and Takeo Kanade. *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[34] Gregory D Hager and Kentaro Toyama. "X vision: Combining image warping and geometric constraints for fast visual tracking". In: *Computer VisionECCV'96*. Springer, 1996, pp. 507–517.

[35] Jan Smisek, Michal Jancosek, and Tomas Pajdla. "3D with Kinect". In: *Consumer Depth Cameras for Computer Vision*. Springer, 2013, pp. 3–25.

[36] Francois Chaumette, Patrick Rives, and Bernard Espiau. "Classification and realization of the different vision-based tasks". In: *Visual Servoing* 7 (1993), pp. 199–228.

[37] João Pedro Hespanha et al. "What tasks can be performed with an uncalibrated stereo vision system?" In: *International Journal of Computer Vision* 35.1 (1999), pp. 65–85.

[38] M Gridseth et al. "Bringing visual servoing into real world applications". In: *Human Robot Collaboration Workshop, Robotics Science and Systems RSS*. Vol. 13.

[39] Camilo Perez Quintero et al. “Interactive Teleoperation Interface for Semi-autonomous Control of Robot Arms”. In: *Computer and Robot Vision (CRV), 2014 Canadian Conference on*. IEEE. 2014, pp. 357–363.

[40] Zachary Dodds et al. “A hierarchical vision architecture for robotic manipulation tasks”. In: *Computer Vision Systems*. Springer, 1999, pp. 312–330.

[41] Romeo Tatsambon Fomena et al. “Towards practical visual servoing in robotics”. In: *Computer and Robot Vision (CRV), 2013 International Conference on*. IEEE. 2013, pp. 303–310.

[42] Peter Corke. “A robotics toolbox for MATLAB”. In: *Robotics & Automation Magazine, IEEE* 3.1 (1996), pp. 24–32.

[43] James Kramer and Matthias Scheutz. “Development environments for autonomous mobile robots: A survey”. In: *Autonomous Robots* 22.2 (2007), pp. 101–132.

[44] Anders Orebäck and Henrik I Christensen. “Evaluation of architectures for mobile robotics”. In: *Autonomous robots* 14.1 (2003), pp. 33–49.

[45] Eigen. *Eigen Benchmark*. 2011. URL: `http://eigen.tuxfamily.org/index.php?title=Benchmark` (visited on 02/2016).

[46] P. Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer Tracts in Advanced Robotics. Springer, 2011. ISBN: 9783642201431.

[47] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge books online. Cambridge University Press, 2003. ISBN: 9780521540513.

[48] Herbert Bay et al. “Speeded-up robust features (SURF)”. In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[49] Microsoft. *Kinect for Windows*. 2015. URL: `https://www.microsoft.com/en-us/kinectforwindows/default.aspx`.

[50] G Chesi et al. “Keeping features in the cameras field of view: a visual servoing strategy”. In: *Proceedings of the 15th International Symposium on Mathematical Theory of Networks and Systems*. 2002.

[51] Youcef Mezouar and François Chaumette. “Path planning for robust image-based control”. In: *Robotics and Automation, IEEE Transactions On* 18.4 (2002), pp. 534–549.

[52] Travis Dick et al. “Realtime Registration-Based Tracking via Approximate Nearest Neighbour Search.” In: *Robotics: Science and Systems*. Citeseer. 2013.

[53] Selim Benhimane and Ezio Malis. "Real-time image-based tracking of planes using efficient second-order minimization". In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 1. IEEE, pp. 943–948.

[54] Simon Baker and Iain Matthews. "Equivalence and efficiency of image alignment algorithms". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001, pp. I–1090.

[55] Peter I Corke and Brian Armstrong-Helouvry. "A search for consensus among model parameters reported for the PUMA 560 robot". In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE. 1994, pp. 1608–1613.

[56] Barret Technology Inc. *WAM Arm Datasheet*. 2016. URL: `https://www.cs.iastate.edu/~cs577/handouts/quaternion.pdf` (visited on 02/2016).

[57] Barrett Technology Inc. *WAM Accuracy and Repeatability Study*. 2011. URL: `http://support.barrett.com/wiki/WAM/AccuracyRepeatability` (visited on 02/2016).

[58] J. Denavit and R. S. Hartenberg. "A kinematic notation for lower-pair mechanisms based on matrices". In: *J. Appl. Mech.* (1955), pp. 215–221.

[59] Barrett Technology Inc. *WAM Denavit-Hartenberg Parameters*. URL: `http://support.barrett.com/wiki/WAM/KinematicsJointRangesConversionFactors` (visited on 10/02/2015).

[60] Vincent Rabaud. *ROS Camera Calibration Package*. 2014. URL: `http://wiki.ros.org/camera_calibration` (visited on 02/2016).

[61] G. Bradski. In: *Dr. Dobb's Journal of Software Tools* (2000).

[62] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 22.11 (2000), pp. 1330–1334.

[63] Claude Samson, Bernard Espiau, and Michel Le Borgne. *Robot control: the task function approach*. Oxford University Press, 1991.

[64] Alessandro Bettini et al. "Vision-assisted control for manipulation using virtual fixtures". In: *Robotics, IEEE Transactions on* 20.6 (2004), pp. 953–966.

[65] Gregory D Hager. "The X-vision system: A general purpose substrate for real-time vision-based robotics". In: *Proceedings of the Workshop on Vision for Robots*. 1995, pp. 56–63.

[66] RE Ellis, OM Ismaeil, and MG Lipsett. "Design and evaluation of a high-performance haptic interface". In: *Robotica* 14.03 (1996), pp. 321–327.

[67] Graziano Chesi and Koichi Hashimoto. *Visual servoing via advanced numerical methods*. Vol. 401. Springer, 2010.