Visual Task Specification User Interface for Uncalibrated Visual Servoing

by

Mona Gridseth

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

Master of Science

Department of Computing Science
University of Alberta

Abstract

In today's world robots work well in structured environments, where they complete tasks autonomously and accurately. This is evident from industrial robotics. However, in unstructured and dynamic environments such as for instance homes, hospitals or areas affected by disasters, robots are still not able to be of much assistance. Moreover, robotics research has focused on topics such as mechatronics design, control and autonomy, while fewer works pay attention to human-robot interfacing. This results in an increasing gap between expectations of robotics technology and its real world capabilities.

In this work we present a human robot interface for semi-autonomous human-in-the-loop control, that aims to tackle some of the challenges for robotics in unstructured environments. The interface lets a user specify tasks for a robot to complete using uncalibrated visual servoing. Visual servoing is a technique that uses visual input to control a robot. In particular, uncalibrated visual servoing works well in unstructured environments since we do not rely on calibration or other modelling.

The user can visually specify high-level tasks by combining a set of geometric constraints. Our interface offers a versatile set of tasks that span both coarse and fine manipulation. The main contribution of this thesis is twofold. First of all we have developed an interface for visual task specification. Second we complete experiments to explore the visual task specification technique and find how to best use it in practice. Finally, we complete experiments to asses the performance of the system.

Acknowledgements

I would like to extend my gratitude to my advisor Professor Martin Jägersand. I am grateful that he offered me a place in his lab as well as for his guidance and encouragement. I also give my sincerest thanks to my advisor at the German Aerospace Centre (DLR) Dr. Katharina Hertkorn for her support, insight and encouragement.

Furthermore I would like to thank all my colleagues in the Computer Vision and Robotics lab in the Department of Computing Science. I have only been able to complete my work because of their help, contributions and excellent collaboration. In particular, I would like to thank Camilo Perez Quintero, Oscar Ramirez and Dr. Romeo Tatsambon Fomena for their help in anything from practical matters to discussions of ideas. Many parts of my work are thanks to their suggestions and contributions.

I also thank the University of Alberta Department of Computing Science, my advisor and the German Aerospace Centre (DLR) for support through research and teaching assistantships.

Finally, I would like to express my gratitude to family and friends for their patience and constant support. I am especially grateful to my parents Torhild and Jostein, who always support and encourage me, even when my work and interests take me far away from home.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

HRI	*Human robot interaction*
IBVS	*Image based visual servoing*
PBVS	*Position based visual servoing*
UVS	*Uncalibrated visual servoing*

Chapter 1

Introduction

1.1 Motivation

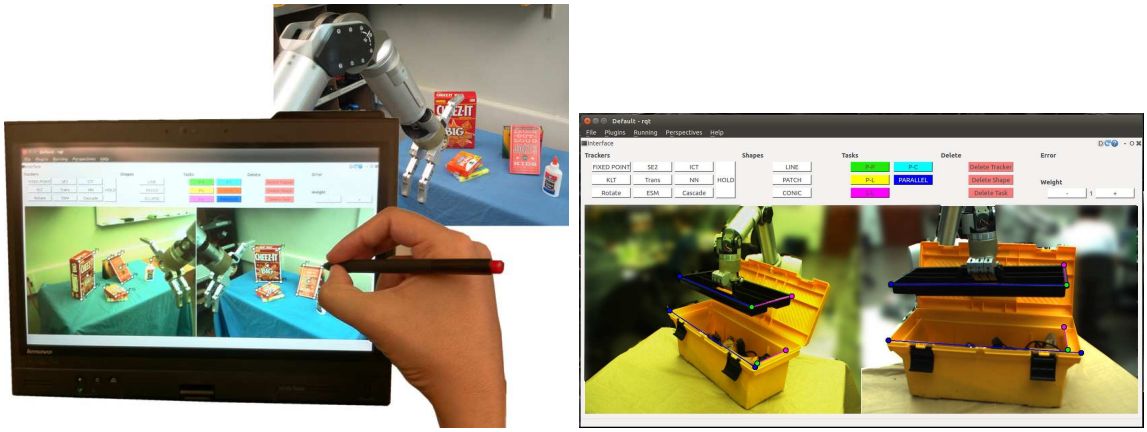
In today's world robots work well in structured environments, where they complete tasks autonomously and accurately. This is evident from industrial robotics. However, in unstructured and dynamic environments such as for instance homes, hospitals or areas affected by disasters, robots are still not able to be of much assistance. Commercially available autonomous robots can vacuume, but cannot handle more complex scenarios like clearing a cluttered table, see Fig. 1.1. Moreover, robotics research has focused on topics such as mechatronics design, control and autonomy, while fewer works pay attention to human-robot interfacing. This results in an increasing gap between expectations of robotics technology and its real world capabilities. In this work we present a human robot interface that aims to tackle some of the challenges that exist for robotics in unstructured environments.

Visual servoing is a technique that allows us to control a robot based on visual input. Despite much published research on visual servoing, little work has resulted in real world applications. We aim to make a real world visual servoing system that can tackle uncontrolled environments. It should be able to accomplish a wide variety of tasks, including both coarse and fine manipulation. Fine manipulation, in particular, is a challenge for many robotic systems that rely on 3D sensors such as the Microsoft Kinect [3]. A part of our strategy is to put the human-in-the-loop. This means that we benefit from both the ability of robots to preform accurate tasks as well as a human user's knowledge, observation and decision making skills.

To this end we develop a human robot interface that allows a user to specify tasks for the robot to complete using visual servoing, see Fig. 1.2. Since the input to visual

(a) (b)

Figure 1.1: (a) Commercially available autonomous robots, like the iRobot Roomba [1], can complete simple tasks (image taken from [2]). (b) A more challenging scenario, where the robot arm should clear a cluttered table.



(a) (b)

Figure 1.2: (a) The idea of having a visual interface to specify task for the robot to complete using visual servoing. (b) More detailed view of the interface we have developed. The task is to insert the black tray into the yellow toolbox.

servoing comes in the form of images, the user will need to specify the tasks visually. Our interface is therefore based on the technique of visual task specification, which is detailed in Chapter 4.

1.2 Research Questions

As mentioned in the previous section, we create a visual servoing interface for human robot interaction in unstructured environments, see Fig. 1.2. The user specifies actions

for the robot by visually defining tasks in images. These tasks are essentially geometric constraints. Examples include point-to-point tasks that move a point on the robot end-effector to a point in the workspace and point-to-line tasks that align the robot end-effector with a line in the workspace. The visual task specification technique is detailed in Chapter 4.

Visual task specification has seen previous publication, but a lot of the work is theoretical or includes only a few examples of tasks completed using visual servoing. Our interface will allow us to examine the practicality of visual task specification more elaborately through experimentation. We want to find out if visual task specification can accomplish high-level robotic tasks in uncontrolled environments. Moreover, we investigate the technique's ability to perform both coarse and fine manipulation tasks.

High-level tasks, such as inserting a marker in a cap or placing an object on a table, cannot be specified using only a single visual task. Hence, we explore how to combine several visual tasks to create high-level actions. In particular, we want to find what the challenges are to this approach and how to best tackle them. Finally, we would also like to comment on the accuracy with which visual servoing can perform actions that are defined by combinations of visual tasks.

1.3 Contributions

The main contribution of this thesis is twofold. We have developed an interface that lets a user specify tasks for the robot and examined how to specify coarse and fine manipulation tasks using visual geometric constraints.

First we developed a visual interface to let the user interact with robot visual servoing, see Fig. 1.2. This work not only involved general interface development and implementation of the geometric constraints for visual task specification, but also required integration with visual tracking and visual servoing.

We used our interface to explore visual task specification. We conducted experiments to asses the ability of visual task specification to complete high-level robotic tasks. Both coarse and fine manipulation were tested, but we especially focussed on fine manipulation tasks as these pose a significant challenge to many robotics systems. The work also includes accuracy and repeatability experiments to asses the performance of our system. We were able to show that our system can reliably and accurately complete a variety of fine manipulation tasks.

High-level robotics tasks cannot be accomplished using single visual geometric

constraints, such as point-to-point tasks, alone. We therefore defined high-level tasks by combining several geometric primitives. For example, we inserted a marker in a cap and picked up a screw by combining point-to-point and parallel lines constraints. The main challenge when combining constraints is the fact that their error signals have varying magnitude. This has the effect of some tasks dominating over others. Through our work we observed the effects of combing constraints and devised strategies for how to best mitigate them.

In summary, this work includes the development of a human robot interface for visual servoing. Moreover, we use this interface and the visual task specification technique to complete a variety of coarse and fine manipulation tasks. Finally, we detail the knowledge we gained on how to successfully combine geometric constraints for visual task specification and some other practical challenges such as camera placement.

1.4 Thesis Outline

Background and related work are presented in Chapters 2 through 4 of the thesis. In particular, Chapter 2 discusses different approaches to human robot interaction in uncontrolled environments. Chapter 3 presents visual servoing, which is the technique we use to control the motion of our robot arm. Chapter 4 presents previous work on visual task specification, but also includes some discussion on how we would like to build on this approach in our work. Chapter 5 describes the user interface that we have created and gives some details on its development. In Chapter 6 we present our experiments and their results. Finally, Chapter 7 contains conclusions and suggestions for future work.

Chapter 2

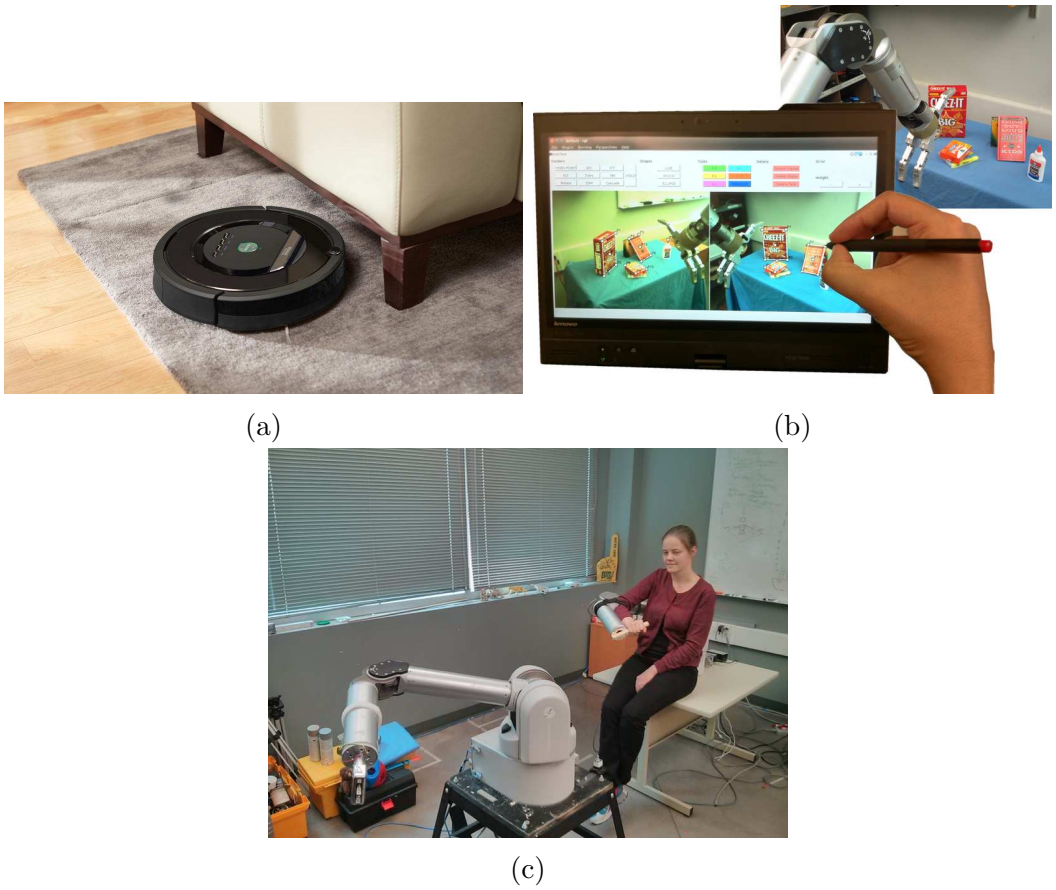
Human Robot Interaction (HRI) in Uncontrolled Environments

Today most robots work in controlled environments such as on factory assembly lines. They can complete tasks accurately and autonomously because they have sufficient knowledge about their surroundings. However, robotic applications have seen less penetration into uncontrolled and dynamic environments. For the most part, robots are still not able to autonomously complete useful tasks in more complex environments such as homes, hospitals or disaster zones. A great deal of research is devoted to finding a solution to this problem [4], as robotic assistance would be of great help to humans in many scenarios. We can imagine robots taking care of household chores, assisting older adults or people with disabilities, helping with rescue efforts after a natural disaster or doing work in environments that are hazardous to humans.

2.1 Human-in-The-Loop Semi-Autonomy

There exists many distinctions between different approaches to robotic development. One such important distinction is related to human robot interaction (HRI) and characterizes the level of human involvement in controlling the robot's actions, see Fig. 2.1. In autonomous robotics the robot acts on its own without human intervention. For example, the iRobot Corporation produces various autonomous robots that do work in and around the house [1]. In particular, the Roomba vacuum robot travels around a room on its own based purely on sensory input.

On the other extreme, we have teleoperation, where a human operator completely controls the robot motion. Teleoperation can be done both with the operator in



(a) (b)

(c)

Figure 2.1: Different levels of human involvement in robot control. (a) The iRobot Roomba [1] vacuum robot that can operate autonomously without human intervention (image taken from [2]). (b) Semi-autonomy, where a user guides a robot arm with the use of an interface to specify tasks. (c) A user completely controls a robot arm through teleoperation.

close proximity to or far away from the slave robot. The DLR teleoperation system lets a human control a humanoid robot equipped with a torso, a head with a set of stereo cameras and two robot arms, each with a robot hand attached. The user interface consist of two robots arms, exactly equal to the ones on the slave robot, and a virtual reality display [5, 6]. The robot will replicate the motions that the user makes with the robot arms and moves the head in line with how the user moves the head set. This robot is imagined to be deployed in space or in disaster zones. It is not a requirement in teleoperation that the input device is equal to the robot being controlled. Different systems employ a diverse set of input devices such as game pads, joy sticks, the Phantom Omni Haptic Device [7] or even the user's own arms.

Figure 2.2: Robotics in an uncontrolled environment. The robot needs to grasp objects from a cluttered table. Grasping any of the objects requires a combination of coarse movements and fine manipulation.

Examples of the latter two modes can be found in [8] and [9]. Another set of examples of teleoperation are the MicroSurge [10] and da Vinci [11] surgical robots. Research into surgical robotics considers both situations, where the surgeon and patient are in the same room and when they are in different locations.

The middle ground between autonomous robotics and teleoperation is semi-autonomy. Semi-autonomy integrates human assistance in the guidance of the robot, but the user does not retain complete control. Semi-autonomy lets us take advantage of the complementary abilities of humans and robots. The human can initialize tasks and monitor their progress, while the robot controls the dynamics for movement and positioning. Essentially, semi-autonomy is said to include the human-in-the-loop. The VIBI system [12] exemplifies human-in-the-loop control over a wheelchair-attached robot arm that assists disabled users with daily tasks. The 7DOF Jaco robot arm [13] is normally controlled using a 2DOF joystick, which leads to switching and complex control. Through a visual interface, the VIBI system allows the user to click on objects for the robot arm to grasp and specify grasping direction. This reduces the work load on the human compared to teleoperation using the joystick. Completing a grasping task autonomously would be too difficult for the robot and so the human involvement is beneficial.

Autonomous robotics proves challenging when dealing with uncontrolled environments. Simple tasks, such as vacuuming, that require little intelligence can be completed autonomously. However, more complex tasks involving advanced reasoning require accurate knowledge and understanding of the surroundings. One such task is grasping and manipulating objects, which is essential to many activities like

cooking, emptying a dishwasher or folding laundry, see Fig. 2.2. Teleoperation is able to tackle unknown and dynamic environments because the user can observe the location through appropriate sensors and make decisions accordingly. The downside to teleoperation is that it requires too much and often taxing work from the human and so results in very complex control, as exemplified by the aforementioned 2DOF joystick control of the 7DOF Jaco robot arm. Moreover, if we use robots to assist older adults or people with disabilities, then some users may not have the cognitive or physical capabilities to completely control the robot. Human-in-the-loop semi-autonomy provides a good compromise. The human involvement makes it possible for the robot to tackle an uncontrolled environment, whereas the robot will handle parts of the task to minimize the workload on the human.

There are many examples of research, where roboticists use semi-autonomy to operate in unstructured environments. In [14], Tsui *et al.* use a wheelchair mounted light-weight robot arm to grasp household objects from a shelf. After the user chooses an object to grasp through a visual interface, the arm retrieves it autonomously. Quintero *et al.* have developed a Selection by Pointing (SEPO) system [15], where the user interacts with the robot using a gesture language and by pointing towards objects he/she wants the robot to grasp. The system uses a Kinect [3] to detect where the user is pointing and what gestures are performed. In [16] the interface is appended with feedback gestures from the robot. [17] presents a flexible system for robot manipulation. Again using the Kinect, a user can position the robot gripper in 3D space and manipulate objects. The system includes a manual mode if the user wants to adjust the position. [18] provides another example of human-in-the-loop robotic grasping. Collaboration between robots and humans also proves useful in surgical robotics. In particular, Bettini *et al.* [19] use computer vision and virtual fixtures to restrict movements for a surgeon in micro surgery. Virtual fixtures showcase another benefit of semi-autonomy. While the user provides direction and observation, the robot offers precise movement. Imagine the robot needs to follow a line for welding. It is difficult for the user to accurately teleoperate line following. Instead the user can specify where the line is and drive the robot forward, while the robot constraints the motion along the line.

Semi-autonomy is also central outside the realm of manipulation. Quigley *et al.* have developed an interface to fly fixed-wing mini-UAVs [20]. Complete teleoperation of such UAVs is technically demanding. The UAVs can be flown autonomously by using waypoints. However, such pre-determined flight plans often need to be adjusted

or are rendered unusable during a mission. Their semi-autonomous interface makes it easier to fly, while at the same time leaving the user in control of the flight. Instead of navigating autonomously, mobile robots can be made to follow humans [21, 22]. This way humans not only show the robot its goal location, but also helps it avoid obstacles along the way. In the first example the robot Big Dog from Boston Dynamics can carry heavy gear, while the user leads the way. Finally, Chen *et al.* have developed a user interface for robots that assist nurses, where the users guide the robot movement through direct physical interfaction (DPI) [23].

In our work we are interested in robotic arm manipulation in uncontrolled environments and semi-autonomy is a promising direction towards this goal. In the next section we will discuss a suitable approach to implement human-in-the-loop robot arm manipulation.

2.2 Visual Servoing in Uncontrolled Environments

The previous examples of semi-autonomous robotics show the advantage of the human-in-the-loop approach to robotic manipulation. Although these techniques facilitate useful HRI, some of them will not be able to tackle manipulation of small objects or other tasks that require high precision. The VIBI [12] and SEPO [15, 16] systems use the Microsoft Kinect sensor [3]. Cheap depth sensors have become useful to HRI because they provide 3D information about the environment surrounding the robot, but they have some weaknesses that limit their usability. One such issue is accuracy. In particular, the respective position accuracies for selecting objects with the SEPO and VIBI systems are 9.6 ± 1.6 cm and 3.0 ± 1.0 cm. This is useful for large movements and coarse object manipulation, but the method will fail for fine manipulation [24]. For example, in [25] the authors use the Kinect sensor for coarse positioning of a robot arm, but need to resort to manual user control with a gesture based interface to grasp objects. Fig. 2.2 illustrates that completing useful tasks requires both coarse movement and fine manipulation. The robot can make large movements towards an object. Additionally, to grasp the red box on top of the Oreo box, the robot must make high precision movements.

In [24] the authors further explain that the limits on sensor range, the robot to sensor calibration and inability to detect small objects as well as an open loop 'look-then-move' approach inhibit fine manipulation using 3D depth sensors. Since depth sensors usually work in about a 0.8m to 5.0m range, the robot can not move close to

the desired objects. Necessary calibration of the sensor to the robot may introduce errors that are large enough to affect high precision manipulation. Moreover, a 3D depth sensor may not always be able to detect small (or shiny and partially transparent) objects, such as a straw in a cup. Lastly, the above mentioned systems employ an open loop approach to grasping. This means that the system first detects an object location and then the robot moves to its assigned goal position in one step. This can prove problematic if the object were to move. Furthermore, the approach does not allow for adjustments if the positioning is imprecise. With inaccurate visual information the robot would be unable perform fine manipulation tasks such as drilling a screw into the wall, inserting an object into a box when space is tight, applying glue along an edge or picking up and manipulating a spoon to name a few examples.

Visual servoing is a technique that allows us to control a robot's movements using visual input from regular 2D cameras [26]. Visual servoing yields much higher position accuracy than 3D depth sensors because we can reach sub-pixel accuracy for the trackers that provide the robot and object positions. Since we do not need camera calibration or camera to robot calibration when using uncalibrated visual servoing, we do not introduce additional errors to these image measurements. This allows us to tackle fine manipulation. Furthermore, the lack of calibration makes visual servoing suitable for operating in uncontrolled environments. Unlike 3D sensors, regular cameras do not suffer from range limits close to objects or the inability to see small objects. Finally, visual servoing is an iterative technique. This means that, at each step of the robot's motion towards a goal position, the system draws on visual input to estimate the current location of the goal. Hence, visual servoing will not fail if the goal position were to change and it continues to improve positioning until a desired precision is achieved.

Due to the advantages detailed in the previous paragraph, we would like to work on versatile robot manipulation in unstructured environments by using uncalibrated visual servoing to control a robot arm based on visual input from regular 2D cameras. In [9] Quintero *et al.* introduce a hybrid system to control a robot arm. The robot arm replicates the user's arm movements, which are detected using a Kinect [3]. The authors show that this works well for large movements. For fine positioning, the system switches to use visual servoing. Similarly [24] suggests a hybrid system that combines the depth sensor based manipulation of the aforementioned SEPO and VIBI systems for large motions with visual servoing for fine manipulation. In [14], the authors switch to 2D visual servoing for retrieving objects from a shelf in order

to archive the necessary precision needed to reliably grasp objects with a robot hand.

We would like to create an HRI interface for robotic arm manipulation that is purely based on visual servoing and handles both low and high precision tasks. Unfortunately, despite much published research, few fully integrated visual servoing systems have been developed that can assist humans in natural environments. This is due to a set of challenges that still needs solving [27, 28, 29]. First of all we need to handle the control of the robot [27]. As mentioned in this chapter, semi-autonomous robot control, where humans and robots work together is beneficial when tackling complex environments. Visual servoing can be implemented using a set of different control algorithms (see Chapter 3 for more detail). Some of these control approaches require calibration of the robot and cameras, which can introduce errors that affect the precision of the positioning. Furthermore, some visual servoing algorithms also rely on 3D information, which require calibration as well as complex modelling of objects and the robot's surroundings. Hence, we will use uncalibrated visual servoing control (UVS) (see Section 3.4), which lets us work in uncontrolled environments without needing any calibration of the system. Furthermore, we will work directly in image space and so eliminate the need for any 3D information or modelling.

A second challenge in developing visual servoing systems that are useful for human needs, is how to specify the actions that the robot should do. We have already established that the user will assist the robot by initializing and monitoring tasks. How can this be done, especially when we only have access to 2D information about our surroundings? One common approach is called 'teach-by-showing' [30]. The robot is manually moved to its goal, where visual image features are captured and stored. Then the robot uses these features as goals when completing the desired task. This works well for repetitive tasks, but will not work in dynamic surroundings, where we would like the robot to be able to accomplish a large set of varied tasks. As mentioned earlier, in [14] the authors use visual servoing to move a robot arm and hand to grasp objects from a shelf. In order to complete a grasp they store SIFT features for each object in a database. Their approach is therefore also not completely general. Instead we will let the user visually specify tasks in an image-editor like interface that displays the camera input. This will be done by combining geometric constraints to construct high-level tasks. Our approach does not require any stored information. In Chapter 4 we explain this approach in more detail and discuss whether we can know if the tasks that are specified in 2D, can in fact be successfully accomplished by the robot.

Fomena *et al.* [27] futher explain that tracking of moving objects is another

challenge that strongly affects robotics. Visual servoing control of robots relies on knowing the location of visual features in 2D image space. In order to accomplish useful tasks, tracking needs to be robust and work real-time. Furthermore, tracking should work in a variety of environments. For example UAVs rely on tracking working well outdoors, whereas some robot arms will be situated indoors. Tracking must be robust and reliable and we need to be able to tackle situations where trackers get lost. Unfortunately, accurate tracking often relies on markers to be successful. The need for markers severely limits the applicability of visual servoing in natural environments. In our system we have chosen to mediate part of this obstacle by providing a set of trackers in out interface. The user can choose which tracker to use that best fits the situation. Moreover, several of these trackers are patch trackers that do not rely on markers.

Finally, a good human-in-the-loop visual servoing system requires a good user interface. The user is supposed to assist the robot in initializing and monitoring tasks. Hence, we need an interface that lets the user easily and intuitively construct tasks and observe how the task completion progresses. We have created an image-editor like interface that displays the image inputs from two cameras in real time. The user will create simple geometric shapes and constraints, much akin to what is done in image editors by setting points and drawing shapes. Shapes and tasks will be displayed on top of the camera images. Moreover, if they are constructed using trackers, they will move according to camera or object motion. For instance, if a user draws a patch on the outline of a tracked cereal box, then this patch will move with the cereal box. That way the user can easily observe the state of the task as the robot moves.

We have developed a semi-autonomous visual servoing system with an intuitive human robot interface. This system aims to perform useful tasks in uncontrolled environments. Through specific implementation choices our system will tackle some of the main obstacles in bringing visual servoing systems into real world environments.

2.3 Summary

Today robots can work autonomously in controlled environments with known surroundings. However, making robots useful in uncontrolled environments is more challenging. We have seen that autonomy, teleoperation and semi-autonomy are possible approaches to robot control. We have argued that including a human-in-the-loop for

semi-autonomous robot control is beneficial when operating in natural environments because we can leverage complementary skills of humans and robots. Humans can initialize and monitor task completion, while robots tackle precise movement and positioning. We discussed using visual servoing together with a visual interface to facilitate human-in-the-loop control. This approach has several benefits such as high precision and lack of calibration or modelling. The interface will let the user specify a diverse set of tasks for the robot to complete. Indeed, it will facilitate HRI in uncontrolled environments.

Chapter 3

Visual Servoing

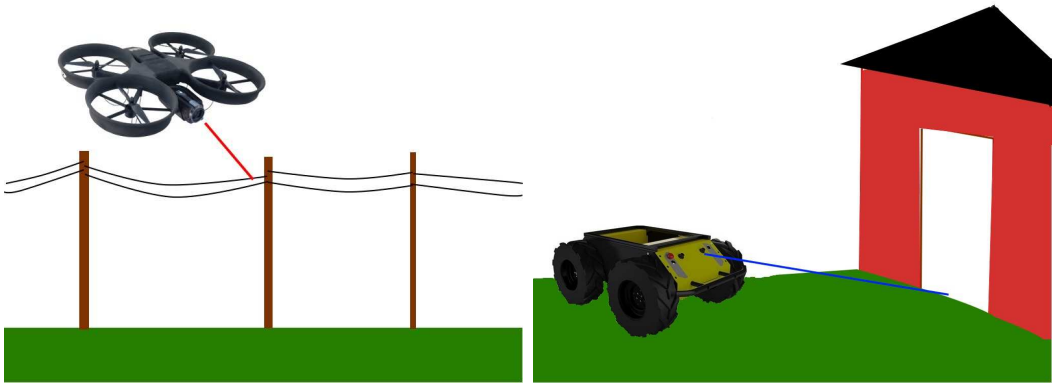
Visual servoing is a technique that allows us to control the motion of a dynamic system, such as a robot, using visual input [26, 31, 30, 32]. This chapter will describe a general formulation of visual servoing and, in three separate sections, detail the position based visual servoing (PBVS), image based visual servoing (IBVS) and uncalibrated visual servoing (UVS) techniques.

3.1 Visual Servoing Overview

Visual servoing is a technique that allows us to use visual input to control the pose of a robot. Although our work is focused on manipulation using robot arms, which uses visual servoing to control the pose of the end-effector in the workspace, visual servoing can also be used in the control of other robots such as mobile robots and unmanned aerial vehicles (UAV), see Fig. 3.1. To illustrate, the authors in [33] use visual servoing to control the position and orientation of a quadrotor and [34] uses visual servoing to let a micro unmanned aerial vehicle (MAV) navigate in indoor environments in cooperation with a ground robot. Furthermore, Mariottini *et al.* [35] devise visual servoing for driving a nonholonomic mobile robot toward a desired configuration

To reiterate, in the case of robot arm manipulation the goal of visual servoing is to use real-time visual input to move a robot end-effector to a desired location in the workspace. That is, we want to control the position and orientation of a robot arm end-effector or an object attached to the end-effector, see Fig. 3.2.

We get our visual information from one or more cameras. They can be placed on the moving robot (eye-in-hand camera configuration) or they can be placed in



(a) (b)

Figure 3.1: Figures of visual servoing with a uav and a mobile robot. (a) We can imagine a UAV using visual servoing to follow a power line. (b) A mobile robot wants to drive through a doorway. The images of the robots are taken from [36, 37].

Figure 3.2: Visual servoing moves the end-effector to a desired location in the workspace to get close to an object in order to grasp it.

(a) (b)

Figure 3.3: (a) Eye-in-hand and (b) eye-to-hand camera configurations.

the workspace to observe the robot motion (eye-to-hand camera configuration), see Fig. 3.3.

Visual servoing performs the positioning task by minimizing an error. This error is created from the camera inputs. That means we parametrize a set of visual features. An example of such visual features are tracked points or edges in the image. We can define the following error:

$$\mathbf{e} = \mathbf{s}(t) - \mathbf{s}^*. \tag{3.1}$$

$\mathbf{s}$ is the vector that holds the current value of the visual features, whereas $\mathbf{s}^*$ represents their desired values, i.e. the state of the visual features when the robot end-effector has reached its goal. Different visual servoing techniques mainly differ in how $\mathbf{s}$, and hence the control error, is defined. On one hand, IBVS or 2D visual servoing constructs $\mathbf{s}$ with 2D features that are directly available from the camera input. On the other hand, in PBVS or 3D visual servoing the image input is used to calculate 3D features that make up $\mathbf{s}$. Finally, there exists a hybrid technique called hybrid visual servoing (HVS) [31], which is normally implemented as 2 1/2 D visual servoing [38] or as a hybrid switching controller [39]. We explain IBVS and PBVS in more detail in Section 3.2 and Section 3.3, respectively.

Now that we have an expression for the error, we need to construct a control scheme that will minimize the error by driving the movement of the robot. Let us assume that we are using an eye-in-hand camera configuration. According to [26], we can create a velocity controller that uses a relation between the change in $\mathbf{s}$ and the spatial camera velocity, $\mathbf{v}_c = (\boldsymbol{v}_c, \boldsymbol{\omega}_c)$. $\boldsymbol{v}_c$ and $\boldsymbol{\omega}_c$ refer to the instantaneous linear and angular velocities of the camera frame. The authors go on to define the relationship

$$\dot{\mathbf{s}} = \mathbf{L}_\mathbf{s}\mathbf{v}_c, \tag{3.2}$$

where $\mathbf{L}_\mathbf{s} \in \mathbb{R}^{k\times 6}$ is the feature Jacobian given that we are controlling the 6 DOF pose of the camera.

Combining the error from (3.1) with (3.2) we get

$$\dot{\mathbf{e}} = \mathbf{L}_\mathbf{e}\mathbf{v}_c, \tag{3.3}$$

Here $\mathbf{L}_\mathbf{e} = \mathbf{L}_\mathbf{s}$. In [26] Chaumette *et al.* alters (3.3) as follows to get an exponential decoupled decrease of the error ($\dot{\mathbf{e}} = -\lambda\mathbf{e}$):

$$\mathbf{v}_c = -\lambda\mathbf{L}_\mathbf{e}{}^{+}\mathbf{e}. \tag{3.4}$$

Figure 3.4: Position based visual servoing moves the robot based on knowledge of the robot end-effector pose and the object pose.

$\mathbf{L_e}^+ \in \mathbb{R}^{6\times k}$ is the Moore-Penrose pseudo-inverse of $\mathbf{L_e}$, so $\mathbf{L_e}^+ = (\mathbf{L_e}^{\mathrm{T}}\mathbf{L_e})^{-1}\mathbf{L_e}^{\mathrm{T}}$. It is not possible to have exact data for $\mathbf{L_e}$ and therefore we must use an approximation that we call $\widehat{\mathbf{L_e}^+}$, resulting in the control law

$$\mathbf{v}_c = -\lambda\widehat{\mathbf{L_e}^+}\mathbf{e}. \tag{3.5}$$

This is a general control law that can be adapted to different types of visual servoing. In particular, in the next sections we will elaborate on how to choose the input $\mathbf{s}$ for different techniques and on how we can estimate the feature Jacobian.

3.2 Position Based Visual Servoing (PBVS)

In PBVS the vector $\mathbf{s}$ of visual features consist of 3D parameters. In the case of an eye-in-hand camera configuration these 3D parameters equal the pose of the camera. 3D information is estimated using image data as well as a model of the system, see Fig. 3.4.

Chaumette *et al.* [26] give an example of a case of PBVS, where they use one eye-in-hand camera. The authors introduce the camera frames $\mathcal{F}_c$, $\mathcal{F}_{c^*}$ and $\mathcal{F}_o$, which represent the current camera frame, the goal camera frame and a reference frame attached to the object the camera is observing. They further define $\mathbf{s} = (\mathbf{t}, \theta\mathbf{u})$, where $\mathbf{t}$ is translation and $\theta\mathbf{u}$ is rotation. $\mathbf{R}$ is the rotation between the current and desired camera frames. Finally, $\mathbf{t}$ is defined relative to the object frame and so we get

$$\mathbf{e} = ({}^c\mathbf{t}_o - {}^{c^*}\mathbf{t}_o, \theta\mathbf{u} - 0) = ({}^c\mathbf{t}_o - {}^{c^*}\mathbf{t}_o, \theta\mathbf{u}). \tag{3.6}$$

The feature Jacobian is defined as follows:

$$\mathbf{L_e} = \begin{pmatrix} \mathbf{R} & 0 \\ 0 & L_{\theta\mathbf{u}} \end{pmatrix}, \tag{3.7}$$

where

$$\mathbf{L}_{\theta\mathbf{u}} = \mathbf{I}_3 - \frac{\theta}{2}[\mathbf{u}]_\times + (1 - \frac{\text{sinc }\theta}{\text{sinc}^2\frac{\theta}{2}})[\mathbf{u}]_x^2. \tag{3.8}$$

Taken together this PBVS algorithm gives the following result for the camera velocities:

$$\begin{aligned} \mathbf{v}_c &= -\lambda \mathbf{R}^{\mathrm{T}c^*}\mathbf{t}_c \\ \boldsymbol{\omega}_c &= -\lambda\theta\mathbf{u}. \end{aligned} \tag{3.9}$$

Different visual servoing techniques have different robustness and accuracy properties. PBVS has the advantage of global asymptotic stability, which means the algorithm theoretically converges for any movement, small or large, as long as the tracked features stay in the camera field of view. However, its success also depends on calibration, pose estimation and tracking of features as well as a model of the robot system. PBVS can do better than a simple position based 'look-then-move' as visual servoing iteratively updates to achieve improved alignment. Hence, if camera frames are close at the final alignment, then the error can be quite low despite a significant robot kinematics error. However, PBVS is not always attractive in a real world scenario as it is not robust to errors and noise.

As mentioned above, global stability also hinges on tracked features always being visible. PBVS can follow optimal trajectories in Cartesian space, but we cannot control the movement of features in image space. Therefore an optimal Cartesian trajectory can lead to features leaving the camera field of view and so the algorithm fails. Since PBVS requires knowledge of camera intrinsic parameters, a model for the observed object and knowledge of the robot calibration, and since it is sensitive to errors PBVS is not a well suited technique for uncontrolled environments.

3.3 Image Based Visual Servoing (IBVS)

In image based visual servoing the vector $\mathbf{s}$ of visual features consist of data taken directly from the images. As an example, $\mathbf{s}$ can be tracked points on the robot end-effector, whereas $\mathbf{s}^*$ can contain tracked points on an object in the workspace that the robot should move towards, see Fig. 3.5. Assume we have knowledge of the camera intrinsic parameters and that $\mathbf{p} = (x, y)$ is a projection of the 3D point $\mathbf{P} = (X, Y, Z)$ from the workspace into 2D image space. If f is the focal length, (u, v) are the pixel coordinates of the point, (c_u, c_v) is the principal point and α is the ratio of pixel

Figure 3.5: Image based visual servoing by minimizing the error between a 2D point feature on the robot, $\mathbf{s}$, and a goal point feature, $\mathbf{s}^*$, on the object.

dimensions, then the projection of P into image space is the following:

$$\begin{aligned} x = X/Z = (u - c_u)/f\alpha \\ y = Y/Z = (v - c_v)/f. \end{aligned} \tag{3.10}$$

Given (3.2), we have

$$\dot{\mathbf{p}} = \mathbf{L}_\mathbf{p}\mathbf{v}_c. \tag{3.11}$$

Chaumette *et al.* [26] show that we can derive an analytical expression for the feature Jacobian matrix, which results in

$$\mathbf{L}_\mathbf{p} = \begin{pmatrix} \frac{-1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & \frac{-1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{pmatrix}. \tag{3.12}$$

This Jacobian only provides 2 DOF control over the robot. However, we can expand to higher DOF control by adding more tracked points and stacking their respective Jacobians into a lager Jacobian matrix. 6 DOF control would require that we stack 3 such matrices. The tracked points can come from more than one image. In particular, we can use a stereo set of cameras. Then, if we track one point in both images, this would give us a size 4 error vector, which is sufficient to control 3 DOF translation of the robot.

Finding the analytical Jacobian (3.12) requires perfect knowledge of the camera intrinsic parameters as well as the depth, Z. This is not feasible in a real world setting, where we can only get approximate calibration. Therefore we rely on using estimates of the Jacobian. Furthermore, in an uncontrolled environment, calibrating the cameras might not be desirable in the first place. In Section 3.4 we discuss the uncalibrated visual servoing (UVS) technique that is similar in some respects to IBVS,

but is able to find an estimate of the Jacobian matrix without having to calibrate the cameras.

Finally, we discuss the robustness and accuracy of IBVS. IBVS suffers from only having local asymptotic stability [26], which means that during large motions IBVS may reach local minima or run into singularities in the feature Jacobian. Furthermore, IBVS may make the robot follow unintuitive or sub-optimal trajectories in Cartesian space. Since the control error is defined in image space, IBVS has the advantage that it is more accurate than PBVS and it does not rely on models of the system to calculate 3D data. The technique is more robust to calibration errors and measurement noise.

3.4 Uncalibrated Visual Servoing (UVS)

In the previous sections we have seen that visual servoing requires a feature Jacobian matrix. In a real world visual servoing scenario the Jacobian needs to be estimated, as the exact version would require perfect calibration of the system. UVS [31, 40, 41, 42] finds a Jacobian and performs visual servoing without the need of any camera intrinsic parameters or models of the robot or observed objects.

Similarly to IBVS, UVS gives us a mapping directly from image space to joint motor space. Furthermore, the algorithm does not estimate depth (which is used in the IBVS analytical Jacobian (3.12)) or any other 3D parameters. As we discussed, having to calibrate cameras is not desirable and so UVS is a technique that will work well in unstructured environments, which are the focus of our work.

Let $\mathbf{q} \in \mathbf{R}^N$ be the configuration of a robot with N joints and let $\mathbf{s} \in \mathbf{R}^M$ represent M visual features. These can for example be point features. Then, we can relate the image space $\mathbf{s}$ and robot joints $\mathbf{q}$ via the visual-motor function $F : \mathbf{R}^N \rightarrow \mathbf{R}^M$ as follows:

$$\mathbf{s} = \mathbf{F}(\mathbf{q}). \tag{3.13}$$

We can take the derivative of this function to get

$$\dot{\mathbf{s}} = \mathbf{J_u}(\mathbf{q})\dot{\mathbf{q}}, \tag{3.14}$$

where $\dot{\mathbf{q}}$ is the control input and $\mathbf{J_u} = \dfrac{\partial \mathbf{F}(\mathbf{q})}{\partial \mathbf{q}} \in \mathbf{R}^{MxN}$ is the visual-motor Jacobian. If we replace $\mathbf{J_u}$ by $\widehat{\mathbf{J}}_\mathbf{u}(\mathbf{q})$ we get the discrete time approximation of (3.14):

$$\Delta\mathbf{s} \simeq \widehat{\mathbf{J}}_u(\mathbf{q})\dot{\mathbf{q}}. \tag{3.15}$$

We can now formulate a proportional control law as

$$\dot{\mathbf{q}} = -\lambda \widehat{\mathbf{J}}_{\mathbf{u}}^{+}(\mathbf{s} - \mathbf{s}^{*}), \tag{3.16}$$

where $\widehat{\mathbf{J}}_{\mathbf{u}}^{+}$ is the Moore-Penrose pseudo-inverse of $\widehat{\mathbf{J}}_{\mathbf{u}}$. $\mathbf{s}$ is the vector of features, while $\mathbf{s}^{*}$ is the vector corresponding to the desired features values.

3.4.1 Jacobian Estimation

Above we see the estimate of the Jacobian, $\widehat{\mathbf{J}}_{\mathbf{u}}(\mathbf{q})$, used in the UVS control law (3.16). An initial estimate of the Jacobian is made using orthogonal exploratory motions [43]. This means that we separately move each robot joint a small δ and record the corresponding changes in the image features. We get N joint movements $\mathbf{q}_1 = [\delta, 0, \cdots, 0]^{\mathrm{T}}, \cdots, \mathbf{q}_N = [0, \cdots, 0, \delta]^{\mathrm{T}}$ corresponding to N feature displacements, $\Delta\mathbf{s}_1, \cdots, \Delta\mathbf{s}_N$, each of size M. Since all the joint movements are of equal size, we solve the following system of equations:

$$\widehat{\mathbf{J}}_{u}(\mathbf{q}) = \frac{1}{\delta}\Delta\mathbf{s}. \tag{3.17}$$

As visual servoing moves the robot towards its target, we need to update the Jacobian matrix. It is not feasible to do so using exploratory motions because this would be slow and we could also collide with objects in the environment. Instead we update the Jacobian using the Broyden method [41, 44] detailed below:

$$\widehat{\mathbf{J}}_{\mathbf{u}}^{(k+1)} = \widehat{\mathbf{J}}_{\mathbf{u}}^{(k)} + \alpha \frac{\left(\Delta\mathbf{s} - \widehat{\mathbf{J}}_{\mathbf{u}}^{(k)}\Delta\mathbf{q}\right)\Delta\mathbf{q}^{\top}}{\Delta\mathbf{q}^{\top}\Delta\mathbf{q}}, \tag{3.18}$$

where α is the learning rate applied to the Broyden update.

3.5 Summary

We have discussed different visual servoing techniques that all have advantages and disadvantages. For our work we will be using UVS because it allows us to work directly with image features to construct the control error. Moreover, UVS does not require calibration of the system or knowledge of any 3D models. UVS lets us approach the challenge of working with HRI in an uncontrolled environment. In the next chapter we will explore how we can combine visual servoing with task specification to facilitate semi-autonomous control by allowing a user to guide the robot to manipulate objects and interact with its environment.

Chapter 4

Visual Tasks

As we saw in the previous chapter, visual servoing is a technique that allows us to control a dynamic system, such as a robot, using visual input. Despite the fact that substantial work has been published on visual servoing over the years, the technique has seen little real world application. As we discussed in Chapter 2, several aspects of visual servoing need to be improved for it to be applicable to real world problems [27, 28, 24]. We wish to make one such improvement by putting the human-in-the-loop for semi-autonomous control, which means that the user needs to be able to guide the robot's actions. In other words we want the user to specify tasks for the robot to accomplish. In this chapter we define more precisely what we mean by such tasks. Furthermore, we show how tasks can be a bridge between visual servoing and HRI that will allow for versatile robot manipulation in uncontrolled environments.

4.1 Visual Task Overview

In [45], Hespanha *et al.* informally characterize a positioning task as the objective of bringing a robot to a target in its workspace. We can describe the pose of the robot and the target using features or combinations of features that we observe through visual input. For example, a task can be to move the robot end-effector towards a bottle that it is supposed to pick up. As mentioned in the previous chapter, visual servoing can also be applied to robotics outside the manipulation realm. We could therefore imagine tasks, such as letting a UAV fly above and along a power line to inspect it, a mobile robot driving though a narrow gate or a robotic boat approaching a dock.

In order to use tasks with visual servoing to accomplish good HRI, we need to

(a) (b)

Figure 4.1: A point-to-point task is specified (a) and completed (b). Each y_i is a 2D point.

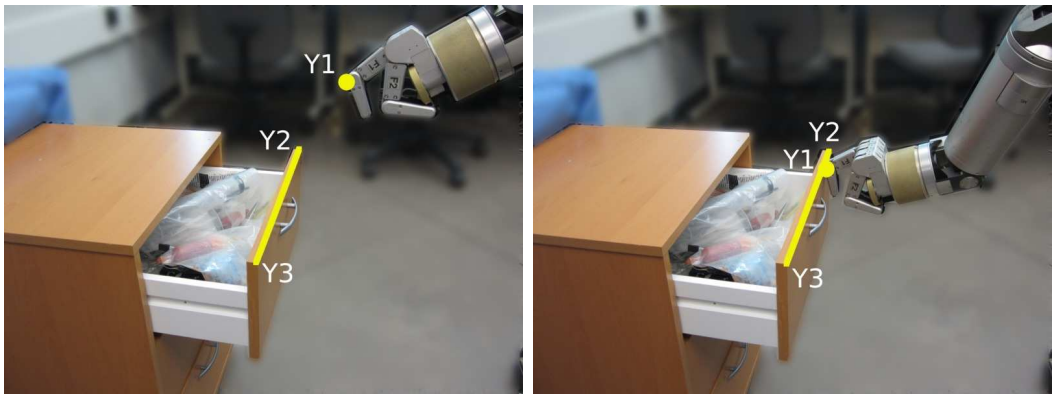
be able to specify the tasks we want the robot to accomplish and we need to know whether these tasks can in fact be completed with the visual information we have. For example, the number of cameras used and their level of calibration can affect the system's ability to accomplish a given task. These questions are tackled by the authors in [45, 46, 47, 48, 49]. Furthermore, [50] describes a visual servoing system that uses tasks for robot manipulation.

[45] provides a formal definition of a task as an equation. Our work is concerned with a robot arm that moves in $\mathcal{W} \in SE(3)$ and is observed by a pair of stereo cameras. Let $\mathbf{f}$ be a list of point features observed through the visual input. The cameras have a joint field of view $\mathcal{V}$ in which the point features lie. [45] describes $\mathcal{V}$ as typically a non-empty subset of $\mathbb{R}^3$ or $\mathbb{P}^3$. The task function maps an ordered set of point features $\mathbf{f} = \{f_1, f_2, \ldots, f_n\} \in \mathcal{F}$ into $\{0, 1\}$. $\mathcal{F}$ is a subset of $\mathcal{V}^n$. The task is defined by the equation:

$$T(\mathbf{f}) = 0. \tag{4.1}$$

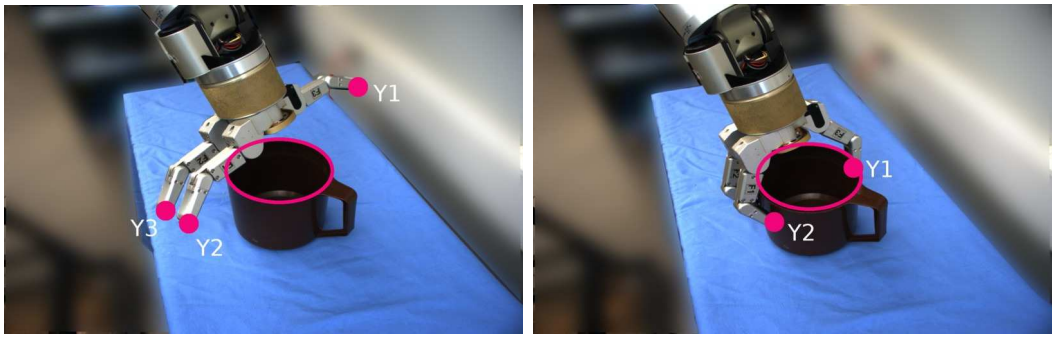
The task is completed when the configuration of the robot is consistent with the task description, i.e. when (4.1) holds.

Below we give examples for each type of task we have implemented in our work. First we describe a *point-to-point* task T_{pp}, which is defined on $F_{\text{pp}} \triangleq \mathcal{V}\text{x}\mathcal{V}$. A point-to-point task aligns two points in the work space, for instance it drives a point on the robot finger to a point on a drawer handle so the robot can close the drawer, see



(a) (b)

Figure 4.2: A point-to-line task is specified (a) and completed (b). Each y_i is a 2D point.



(a) (b)

Figure 4.3: Three point-to-ellipse tasks are specified (a) and completed (b). Each y_i is a 2D point.

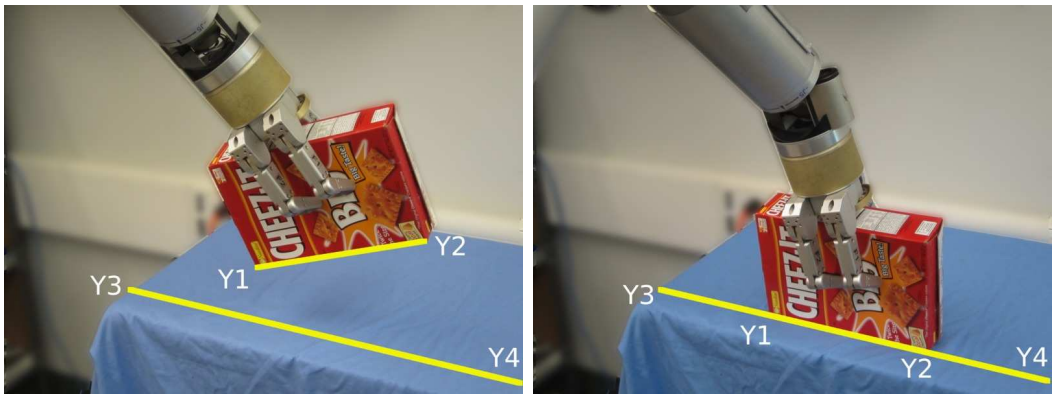
Fig. 4.1. For $T_{\text{pp}}, \mathbf{f} = \{f_1, f_2\}$ is mapped as follows:

$$\mathbf{f} \to \begin{cases} 0 & f_1 \text{ and } f_2 \text{ are the same point in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases}$$

The next task is a *point-to-line* task T_{pl}, which is defined on $F_{\text{pl}} \triangleq \mathcal{V}^3$. A point-to-line task aligns a point with a line defined by two other points. As an example, a point on the robot end-effector can move to a line on the edge of an open drawer in order to close the drawer, see Fig. 4.2. For $T_{\text{pl}}, \mathbf{f} = \{f_1, f_2, f_3\}$ is mapped as follows:

$$\mathbf{f} \to \begin{cases} 0 & f_1 \text{ lies on the same line as } f_2 \text{ and } f_3 \text{ in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases}$$

We continue with the *point-to-ellipse* task T_{pe}, which is defined on $F_{\text{pe}} \triangleq \mathcal{V}^6$. A point-to-ellipse task aligns a point in the work space with an ellipse. For example



(a) (b)

Figure 4.4: A line-to-line task is specified (a) and completed (b). Each y_i is a 2D point.

it can drive a robot finger to the edge of a round pitcher, see Fig. 4.3. For $T_{\text{pe}}, \mathbf{f} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$ is mapped as follows:

$$\mathbf{f} \rightarrow \begin{cases} 0 & f_1 \text{ lies on the ellipse formed by } f_2, f_3, f_4, f_5 \text{ and } f_6 \text{ in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases}$$

Next we add a *line-to-line* task T_{ll}, which is defined on $F_{\text{ll}} \triangleq \mathcal{V}^4$. A line-to-line task aligns two lines in the work space. If the robot is holding a box and we want to put it down on a table, we can define a line-to-line task between one edge of the box and a line on the surface of the table, see Fig. 4.4. For $T_{\text{ll}}, \mathbf{f} = \{f_1, f_2, f_3, f_4\}$ is mapped as follows:

$$\mathbf{f} \rightarrow \begin{cases} 0 & \text{the line formed by } f_1 \text{ and } f_2 \text{ and the line by } f_3 \text{ and } f_4 \text{ are the same in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases}$$

Finally, we present the *parallel* task T_{par}, which is defined on $F_{\text{par}} \triangleq \mathcal{V}^4$. A parallel task makes two lines in the workspace parallel, see Fig. 4.5. For $T_{\text{pp}}, \mathbf{f} = \{f_1, f_2, f_3, f_4\}$ is mapped as follows:

$$\mathbf{f} \rightarrow \begin{cases} 0 & \text{the line formed by } f_1 \text{ and } f_2 \text{ and the line by } f_3 \text{ and } f_4 \text{ are parallel in } \mathbb{P}^3 \\ 1 & \text{otherwise} \end{cases}$$

Above we discussed the role of point features in different tasks. As described in [45], the point features are mapped into the joint image space of the stereo cameras, $\mathcal{Y}$, by the not precisely known camera model $C_{\text{actual}} : \mathcal{V} \rightarrow \mathcal{Y}$. $C_{\text{actual}} \in \mathcal{C}$, which is

(a) (b)

Figure 4.5: A parallel lines task is specified (a) and completed (b). Each y_i is a 2D point.

a set of injective functions. [45] also introduces the extension of C to $\mathcal{F}$, which the authors call $\bar{C}$. For each C in $\mathcal{C}$, $\bar{C}$ is the function that maps $\mathcal{F} \in \mathcal{V}^n$ to $\mathcal{Y}^n$ as follows:

$$\{f_1, f_2, \ldots, f_n\} \rightarrow \{C(f_1), C(f_2), \ldots, C(f_n)\}$$

Taken together we need to use the knowledge of $C, T, \mathcal{V}$ as well as the measured image features

$$\mathbf{y} \triangleq \bar{C}_{\text{actual}}(\mathbf{f}) \tag{4.2}$$

to decide whether or not a task is accomplished. Since C_{actual} is not know precisely, relying on image features alone will not provide sufficient information for every type of task.

In order to address the issue of whether a task can in fact be accomplished with the given visual information, the authors in [45] introduce the concepts of *task decidability* and *task encoding*. Let $E : \mathcal{Y}^n \rightarrow \mathbb{R}$ be the encoded task function. Then the encoded task is represented by the following equation:

$$E(\mathbf{y}) = 0, \tag{4.3}$$

where the task is accomplished if (4.3) holds. Moreover, E is a function that does not depend on knowledge of the calibration of the cameras. According to [45], if there exists an encoding of the task that can verify if the task has been accomplished, then the task is said to be decidable. We therefore know that a task T is accomplished if and only if the encoded task E is accomplished. In particular, a task $T(\mathbf{f}) = 0$ is

verifiable on $\mathcal{C}$ with an encoding $E_T(\mathbf{y}) = 0$ if, $\forall \mathbf{f} \in \mathcal{F}, \forall C \in \mathcal{C}$,

$$T(\mathbf{f}) = 0 \iff E_T|_{\mathbf{y}=\bar{C}(\mathbf{f})} = 0. \tag{4.4}$$

Furthermore, the authors state that a task T is decidable on $\mathcal{C}$ if there exists an encoding $E_T(\mathbf{y}) = 0$ for which (4.4) holds. That means that a decidable task is one where we can determine if the task is accomplished or not based on measured data.

In order to know when a task is satisfied we need an error signal that will be zero once the task is accomplished. We use this measurement as the error for our visual servoing control law (3.16) that we described in the previous chapter. Minimizing the error function using the visual servoing control algorithm will drive the robot pose to complete the task. Hence we can use $E(\mathbf{y})$ as the error vector for visual servoing.

Each of the tasks we described above has a corresponding encoding. The size of the resulting vector defines the number of DOF of the robot that we can control using this particular task. Below we give the encoding for each task as we have implemented them. Let each y_i be a 2D point in $\mathcal{Y}_n$. These encodings correspond to the tasks seen in Figures 4.1, 4.2, 4.3, 4.4 and 4.5.

$$E_{\text{pp}}(\mathbf{y}) = (y_2 - y_1) \tag{4.5}$$

$$E_{\text{pl}}(\mathbf{y}) = (y_1 \cdot (y_2 \times y_3)) \tag{4.6}$$

$$E_{\text{pe}}(\mathbf{y}) = y_1^{\text{T}} C_{\text{ellipse}} y_1,\ C_{\text{ellipse}} = \begin{pmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{pmatrix} \tag{4.7}$$

$$E_{\text{ll}}(\mathbf{y}) = (y_1 \cdot (y_3 \times y_4)) + (y_2 \cdot (y_3 \times y_4)) \tag{4.8}$$

$$E_{\text{par}}(\mathbf{y}) = (y_1 \times y_2) \times (y_3 \times y_4) \tag{4.9}$$

C_{ellipse} is the coefficient matrix that defines the ellipse. These coefficients are estimated from the point features that lie on the ellipse.

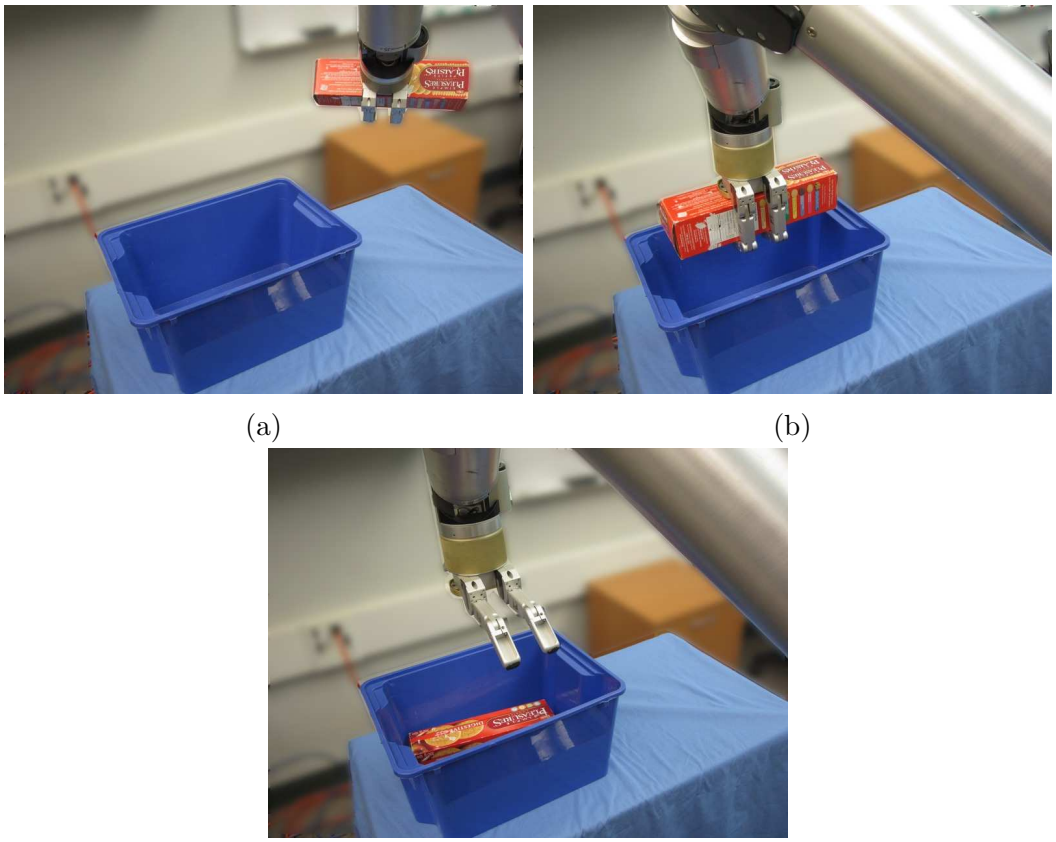
In [45], Hespanha *et al.* not only describes visual task specification, but also prove that given a weakly calibrated stereo pair, a positioning task can be precisely accomplished if and only if the task specification is invariant to projective transformations. This does not hold for UVS. In fact, the authors only proved that point-to-point tasks are decidable for UVS. It may therefore seem counter intuitive that we build a system for task specification using UVS. However, in this regard Hespanha *et al.* point to [49], which completes a set of projectively invariant tasks using UVS. They

state that the approach works well in practice although the tasks are not theoretically decidable. The reason for this is that the tasks are still verifiable in most cases, except for in some special configurations. On example is the failure of the collinearity (point-to-line) task for a stereo camera pair when the feature points and the camera optical centres all lie in the same plane [45]. In our set of constraints, the parallel lines task is not projectively invariant. For that reason it is not decidable for whether weakly nor uncalibrated cameras. However, in light of the results in [49] that show the practicality of visual task specification for UVS, we would like to work with the parallel lines task and through experimentation find if it is useful in practice.

4.2 Combining Tasks

In the previous section we discussed some of the details of specifying tasks for visual servoing. Moreover, we listed examples of such tasks; point-to-point, point-to-line etc. However, it may not be immediately apparent how we can use these tasks for useful and versatile HRI. After all, completing a real world task with a robot involves more than single point-to-point or point-to-line alignments. Imagine that the robot is holding a cookie box and we want to place it in a bin (for example a trash bin), see Fig. 4.6. A simple point-point task could bring the box towards the bin, but the robot would not be able to accurately place the box in the bin because we cannot guarantee its orientation relative to the bin, see Fig. 4.7.

In this section we suggest how we can take the theory of tasks presented in Section 4.1 and use it in a way that allows us to perform versatile actions with the robot. We propose that, if we use tasks correctly, they do have the power and flexibility to complete useful manipulation. The simple tasks described in the previous section can be combined to form more complex high-level tasks. In particular, we can specify tasks in sequence or in parallel. For example, in the case of placing a cookie box in a bin, we can first specify two parallel line tasks to make sure the box has the correct orientation relative to the bin, see Fig. 4.8. We want to make sure that the edges of the box will be parallel to the edges of the bin. These two tasks can be specified in parallel, meaning they will constrain the robot movement simultaneously. Once the orientation is correct, we can bring the box to the bin using a point-to-point task between one corner of the cookie box and a point in the bin, see Fig. 4.8. The two parts of this manipulation would then be specified in sequence, the robot performing one step before continuing with the next.



(a) (b)

(c)

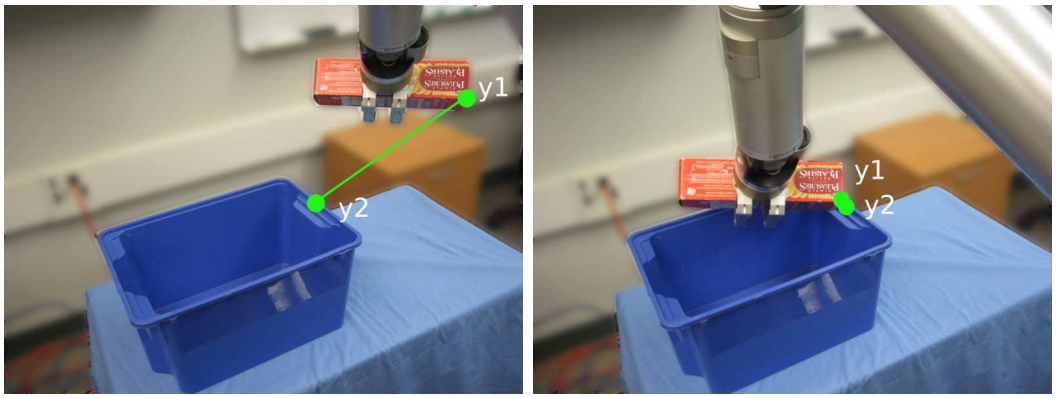
Figure 4.6: (a) The robot holds a cookie box and we want to place it in the bin. (b) The task has been accomplished. (c) The box has been dropped into the bin.

Let us elaborate on the above concept in a more formal manner. In [50] Dodds *et al.* explain that tasks can be specified simultaneously or in parallel by simply stacking their error vectors.

$$E(\mathbf{y}) = \begin{pmatrix} E_1(\mathbf{y}) \\ E_2(\mathbf{y}) \\ \vdots \\ E_k(\mathbf{y}) \end{pmatrix}, \tag{4.10}$$

where $\mathbf{y} \in \mathcal{Y}^n$ is the image measurement. This stacked vector becomes the error vector for visual servoing. If we specify two parallel line tasks in one image, as shown in Fig. 4.8, it would look as follows:

$$E(\mathbf{y}) = \begin{pmatrix} E_{\text{par}}(\mathbf{y}) \\ E_{\text{par}}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} (y_2 \times y_3) \times (y_5 \times y_6) \\ (y_1 \times y_2) \times (y_4 \times y_5) \end{pmatrix}. \tag{4.11}$$



(a) (b)

Figure 4.7: (a) We specify a point-to-point task to place the cookie box in the bin. Each y_i is a 2D point used to construct the geometric primitives. (b) We see that specifying only a 3 DOF point-to-point task allows us to translate the box to the bin, but we have no control over its orientation and hence the task fails.

If we use two cameras, the errors from the second camera will simply be stacked into the same vector. In [50], the authors point out that only specifying tasks in parallel does not solve many manipulation problems. Therefore they also create temporal manipulation sequences that they call task chains. Each such chain consists of task links that represent one task. Each task link can be a parallel composition of geometric constraints such as (4.10). In the example mentioned above, the first link would be the two parallel line tasks to adjust the orientation of the cookie box, whereas the second link would correspond to the point-to-point task that brings the box close to the bin.

One important thing to consider when specifying complete tasks for visual servoing is the number of DOF we are controlling. Each primitive task that is stacked into the final error vector for visual servoing (4.10) contributes to its size. A point-to-point constraint is of size 2 for each camera, whereas the other constraints we have introduced are of size 1. The total number of DOF that we can control with visual servoing equals the size of the stacked error vector minus any redundant specifications. Hence, we need to choose a set of tasks that allows us to control the necessary DOF. For example a point-to-point task specified in two images gives an error vector of size 4. This means we can control 3 DOF translation, but we cannot control 6 DOF

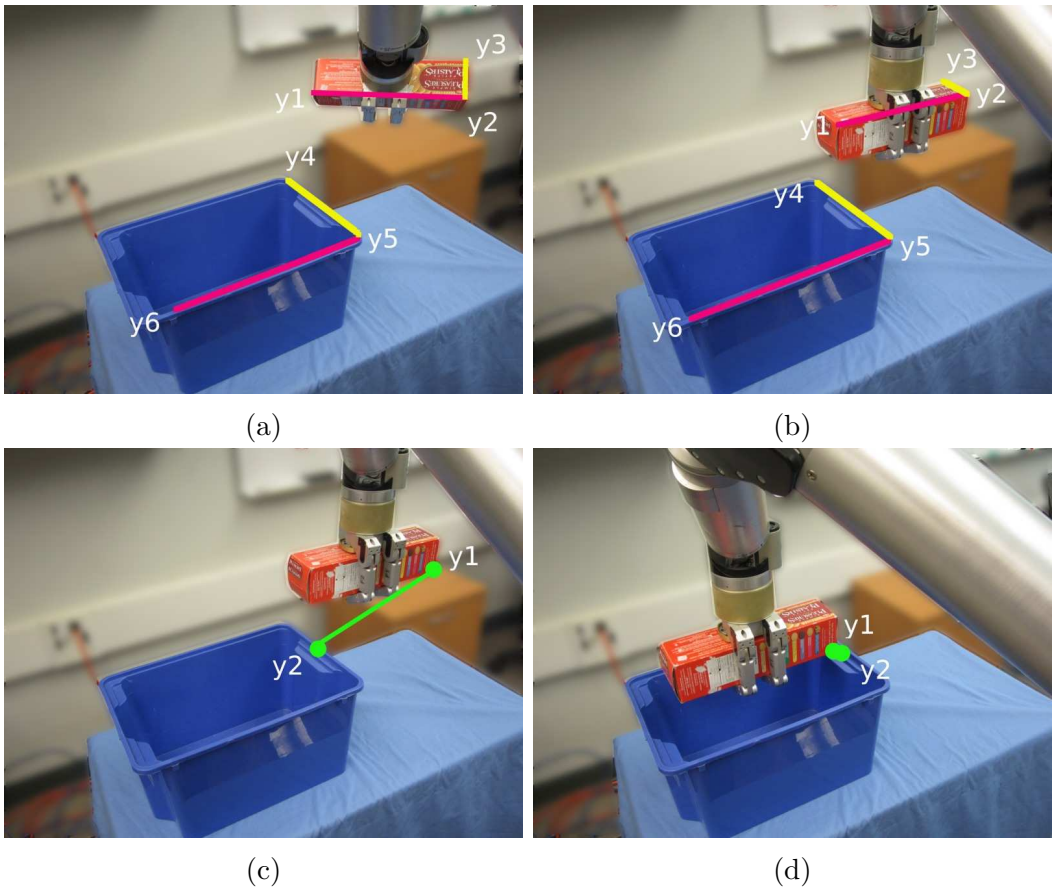


Figure 4.8: (a) We specify two parallel line tasks simultaneously to have the cookie box be oriented correctly with respect to the bin. Each y_i is a 2D point used to construct the geometric primitives. The red line on the box should be parallel to the red line on the shelf and similarly for the yellow lines. (b) The alignment has been completed. (c) We go on to specify a point-to-point task that will bring the cookie box to the bin. (d) The task has been accomplished.

translation and orientation.

4.3 Task Specification for Robot Manipulation

As we learned from the last section, simple tasks can be combined to create high-level tasks. As a part of our work we need to tackle the challenge of how this can be done in practice. The theory tells us that such tasks can be combined, but we need to find out for instance in which order to combine tasks, what tasks can be specified at the same time and whether the tasks will behave as expected in a real system?

Dodds *et al.* [50] give an example of a real system for robot manipulation that integrates the use of task primitives to complete high-level tasks. We will present this as an example of one strategy for combing tasks and explain their approach. In our own work we experiment with different strategies and present our results on how we combine tasks in Chapter 6. [50] uses task chains to create a sequence of actions that allows a robot arm to interact with its environment. The authors discuss how we can break large and complex tasks into task links.

They argue that high-level tasks have some natural delimiters that would help us divide them into smaller manageable parts, namely change in features, dimensionality and precision of constraints. For example, if we are doing a coarse movement to place the robot arm close to an object, we require 3 DOF translation. If we were using the robot to follow a line for welding, the orientation of the end-effector would matter and it would be important that the task is implemented with high precision. For these two tasks the dimensionality is different (3 DOF vs 6 DOF), the level of precision required is not the same and the number of features we need in order to build the task constraints is also different.

As is emphasized in [50], we only need to use the simplest robot movement that is required to complete each part of a task. This means that in a translation task we only specify constraints that will control 3 DOF of the robot. Controlling the full 6 DOF pose is unnecessary. This strategy makes sure that our task specifications are as simple as possible and it leaves us with less features to track.

As a strategy for dividing a complex robot task into a chain of task links, Dodds *et al.* [50] suggest that we divide tasks into three categories; transport, alignment and fine manipulation. Transport refers to large or coarse movements that brings the robot close to an object. The movement consists of a translation, which only requires a 3 DOF constraint. This can be accomplished by specifying a point-to-point task in two images, which provides an error vector of size 4. Transport is robust and does not require high precision.

Fine alignment refers to tasks, where we control precise translation and orientation, i.e. we require 6 DOF control of the robot arm. These tasks are more precise and they require more complex task specification as well as more tracked point features. Moreover, they are not as robust as transport tasks.

Alignment is the bridging step between transport and fine manipulation. As mentioned, fine manipulation needs accurate orientation and positioning, whereas transport only provides a 3 DOF constrained placement of the end-effector. An

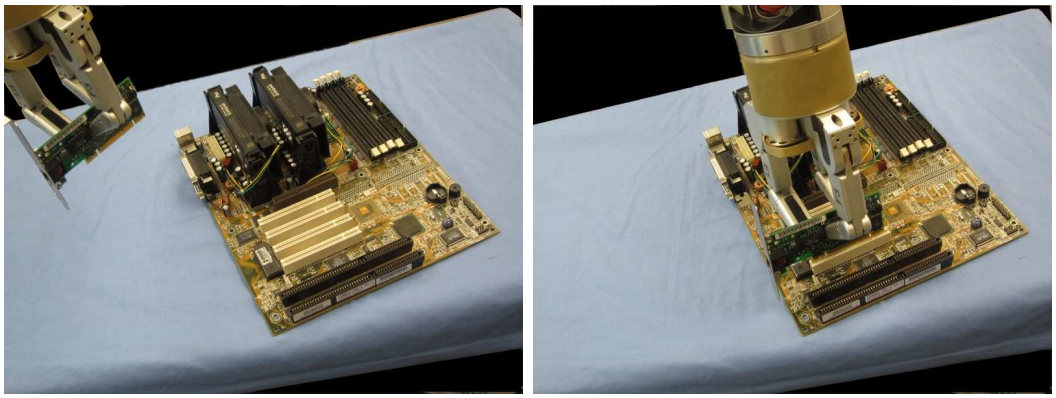
alignment task starts with the coarse 3 DOF position from a transport and adds constraints to adjust the robot to a more precise orientation and position. After the alignment the robot is ready to continue with a fine manipulation task.

The task chain approach creates a practical challenge when we sequentially combine tasks. When constraints change, we need to update the Jacobian for visual servoing as its size will change. For example, assume we have a Jacobian for a positioning task. The alignment step that follows may need a larger Jacobian as more constraints are being added. If we have enough space to move in, we can do a new set of exploratory motions to create a new Jacobian in the same way as is done at the beginning of visual servoing. If exploratory motions are not possible, we need to update the Jacobian and change its size by using information from the previous constraints.

Now we give an example of how a high-level task can be specified by dividing it into transport, alignment and fine manipulation segments and how we combine primitive tasks to construct each segment. Imagine that the robot holds a network card and we want to insert into its slot, see Fig. 4.9. First we want to move close to the slot with a transport task. However, we do not want to move too close as we need space to adjust the card's orientation with respect to the slot. Therefore, we construct a point in the air above the vicinity of the goal (this is done by adding a pixel column). We specify a point-to-point task between a bottom corner on the card and the constructed point, see Fig. 4.10. We proceed with the alignment part of our manipulation, where we specify two parallel line tasks that will align the orientation of the board with that of the slot, see Fig. 4.10. After the alignment is accomplished, we add a point-to-point task to the already specified parallel line tasks. This will specify the fine manipulation that brings the card to the slot, while preserving the correct orientation, see Fig. 4.10.

More than just allowing us to construct versatile high-level tasks, visual servoing in combination with specification of task chains, tackles one of the challenges discussed in Chapter 2. Where many HRI systems based on depth sensors can only handle coarse movement and manipulation, task chains allows a system to complete actions that contain both coarse movement and fine manipulation.

The papers that we discussed in Section 4.1 and 4.2 provide the theoretical framework for visual task specification. Furthermore, [50] presents an implementation of two high-level tasks based on the aforementioned work. In our work we build a system with a user interface that allows us to explore task specification for high-level actions



(a) (b)

Figure 4.9: We want to complete a task to insert the network card in (a) into its slot as seen in (b).

in a wide variety of scenarios.

4.4 Summary

We have discussed how we can use tasks when controlling a robot arm with visual servoing. High-level tasks for the robot can be constructed from simple task primitives. This has promise to allow users to specify versatile coarse and fine manipulation tasks for robot manipulation in natural environments.

In order for the user to be able to specify such actions, we need an interface. Specifying tasks requires interaction with visual information. In particular, we need to be able to track point features in images and we need to be able to construct geometric primitives. We have worked to develop an interface, where the user can do so in an intuitive manner. This interface will also allow us to explore task specification in real world robotics systems. The details of this interface will follow in Chapter 5.

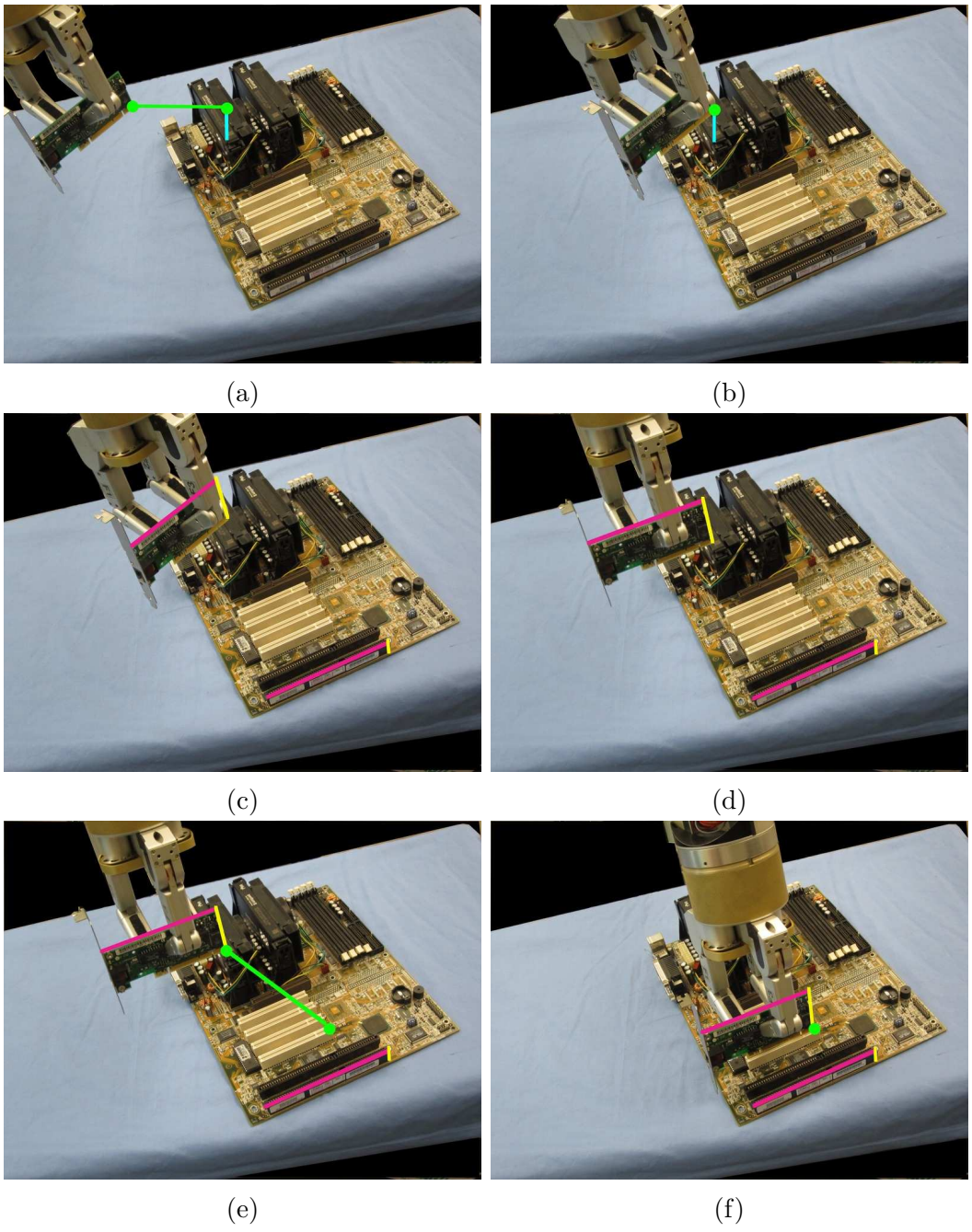


Figure 4.10: This figure shows the specifications needed to insert a network card in its slot. (a) We specify a coarse transport task to take the card close to the board. (b) The task is accomplished. (c) We specify two parallel line tasks to adjust the orientation of the card. (d) The task is accomplished. (e) We specify the fine manipulation to insert the card, by adding a point-to-point task at the same time as preserving the correct orientation using the parallel lines tasks. (f) The task is accomplished.

Chapter 5

User Interface for Visual Task Specification

In the background chapters of this thesis we have discussed how visual servoing combined with visual task specification has the potential to facilitate semi-autonomous robot control in uncontrolled environments. The main contribution of this thesis is twofold. We explore whether we can use combinations of simple geometric primitives for task specification as put forward by previous work detailed in Chapter 4. We also create as a user interface that facilitates the interaction for task specification. The interface lets users visually specify high-level tasks for a robot to complete. Our work involves manipulation with a robot arm. However, this interface is general and can be included in any system that uses visual servoing to control robot movement. In this chapter we will describe the interface and how it works. Furthermore, we will elaborate on some challenges and design decisions that were a part of the development process.

5.1 Visual Task Specification Interface

As discussed in Chapter 2, robotics in uncontrolled environments poses several challenges. To summarize, we elaborated on semi-autonomy or human-in-the-loop control, using uncalibrated visual servoing to avoid calibration and allow for high precision manipulation, visual task specification based on 2D information, the challenges of reliable tracking as well as designing a good interface for the user. Our uncalibrated visual servoing system in combination with the user interface addresses these concerns.

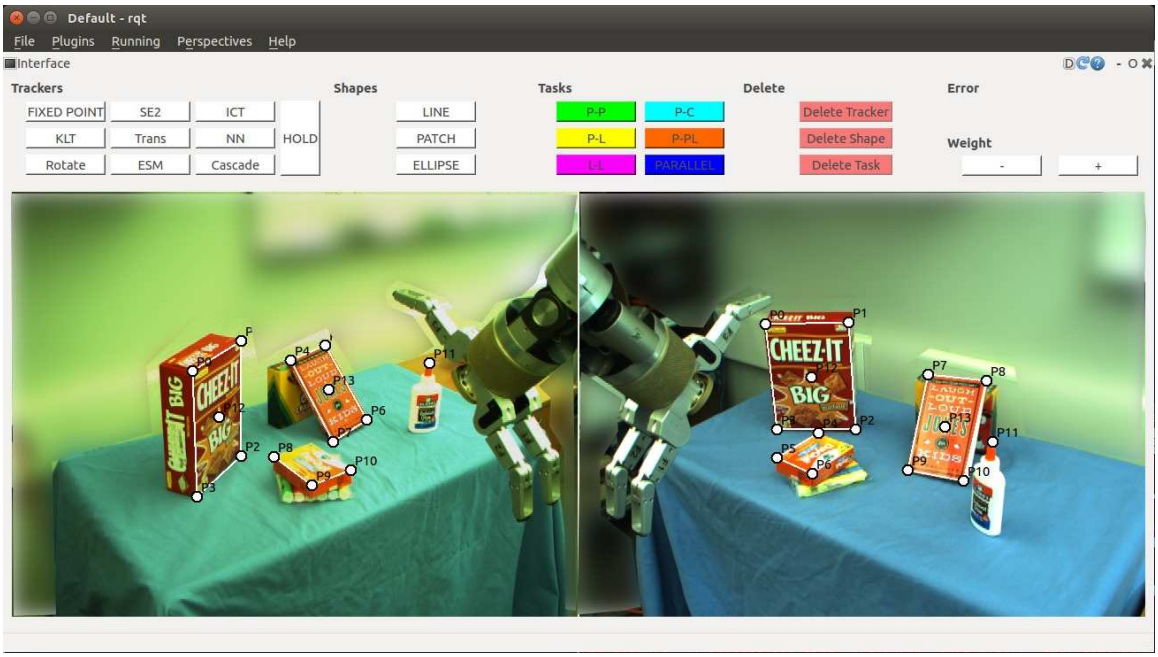


Figure 5.1: A view of the interface, where the user has drawn points, lines and patches.

We would like a user to easily be able to specify high-level tasks for a robot to complete. As discussed in Chapter 4, we can do this by combining simpler geometric primitives and creating task chains. We gave an example, where the user can combine some parallel lines and point-to-point tasks sequentially and in parallel to drop a bx of cookies in a bin. In order to specify a variety of tasks in a general way, we need to develop a simple and intuitive user interface. In this section we will discuss the different features of our user interface and give some justification for their design.

In an image based visual servoing system the user gets his/her information from the robot cameras. Therefore, it is inherently a visual system and in particular, one that is viewed in 2D. Furthermore, the tasks we construct are geometric. Hence, we have strived to make the interface reminiscent of a drawing or image-editor program. Users are familiar with drawing program interfaces and they translate well to our application. We let users create points and geometric shapes. For example, users can draw lines and ellipses on the images. These shapes will in turn be the basis for the tasks. Fig. 5.1 shows the interface with points and shapes drawn by the user.

An essential feature to any visual servoing system is trackers. Depending on the camera configuration (eye-in-hand vs. eye-to-hand), the robot or the objects in the environment or both will be moving in the images. We need to track the location of

Tracker	Implentation
KLT	ViSP
NN	Developed in our lab
ESM	Implemented from [54]
IC	Implemented from [55]
CASCADE	Combination of NN and IC
TRANS	XVision
ROTATE	XVision
SSD	XVision
RT	XVision

Table 5.1: The table lists the different trackers available in the interface and shows where the implamentation comes from.

the robot end-effector as well as target objects. Tracking is one of the most difficult challenges for visual servoing in real world environments. Trackers must work in real time and they need to be robust. It is difficult to find trackers that work well under all conditions. Therefore we introduce the ability to simultaneously use a number of different trackers in the interface. For instance, the user can initialize a KLT [51, 52, 53] point tracker to track a marker on the end-effector or he/she can initialize an ESM tracker [54] to track a textured patch, such as the side of a cereal box. Table 5.1 gives an overview of the trackers we have included in the interface. For robotics in uncontrolled environments, trackers should ideally work robustly without having to put markers on objects, however this is difficult to accomplish with most existing point trackers. The trackers that track larger patches work without markers. If a tracker is lost, the specified task becomes invalid and the visual servoing will stop. The user can then set a new tracker before continuing the servoing.

Tasks are formed from simple geometric primitives. It naturally follows that our interface needs a way to define geometric shapes in the input images. To that end we have implemented points, lines, patches (with four corner points) and ellipses. The user can initialize fixed points or tracked points in the image and then use these to construct the other shapes. Once created, these shapes can be used to create tasks. For example, a line-to-line task can be constructed from two lines.

The main goal of the interface is to enable the user to specify high-level tasks, which we have argued can be done by combining simpler tasks. In the interface we have implemented a set of simple tasks that the user can create from points or shapes.

When the user specifies several such tasks simultaneously, they are automatically combined. The user can easily create and delete tasks. Once a set of tasks has been completed and removed, the user can move to the next part of the task chain and initialize a new set of tasks.

Finally, in order to move the robot, the interface must integrate the task specification with visual servoing. This means that the errors we get from each task must be sent to a visual servoing module that calculates the robot's next step.

In this section we have described in broad strokes what is needed from a visual task specification interface. Section 5.2 will discuss in more detail how we designed the interface with these requirements in mind and how we dealt with some of the accompanying challenges. Section 5.3 will give a thorough description of how the interface works in practice, which is illustrated through an example use case.

5.2 Design and Development Considerations

As mentioned in the previous section, we identified different features that we need in a task specification interface. Here we will elaborate further on the development of some of these features.

Our interface was developed using the Robotics Operating System (ROS) [56, 57]. We specifically used a part of ROS called rqt that lets us develop a graphical user interface (GUI) with the Qt library [58] within the ROS framework. ROS acts as a communication layer. We can create individual nodes for different processes and these nodes can communicate with each other by sending and receiving messages, see Fig. 5.2 for an illustration of the system. This allows for modularity and makes it easy to integrate the interface with other code, such as the visual servoing code. In particular, the interface only needs to send a message containing the stacked error vector to the visual servoing, no other information is required. Furthermore, ROS is open source and widely used in the robotics community. We are therefore complying with standards commonly used by other roboticists and it would be easy for others to use our code. In particular, our interface can be integrated with other visual servoing systems, also those using a different kind of robot.

As mentioned earlier, one of the most difficult challenges in visual servoing is to achieve real time and robust tracking. Introducing a set of different trackers gives the user a chance to choose the tracker that is best fit for the task. Due to the modularity of ROS we have implemented different trackers their own nodes, see Fig. 5.2. This

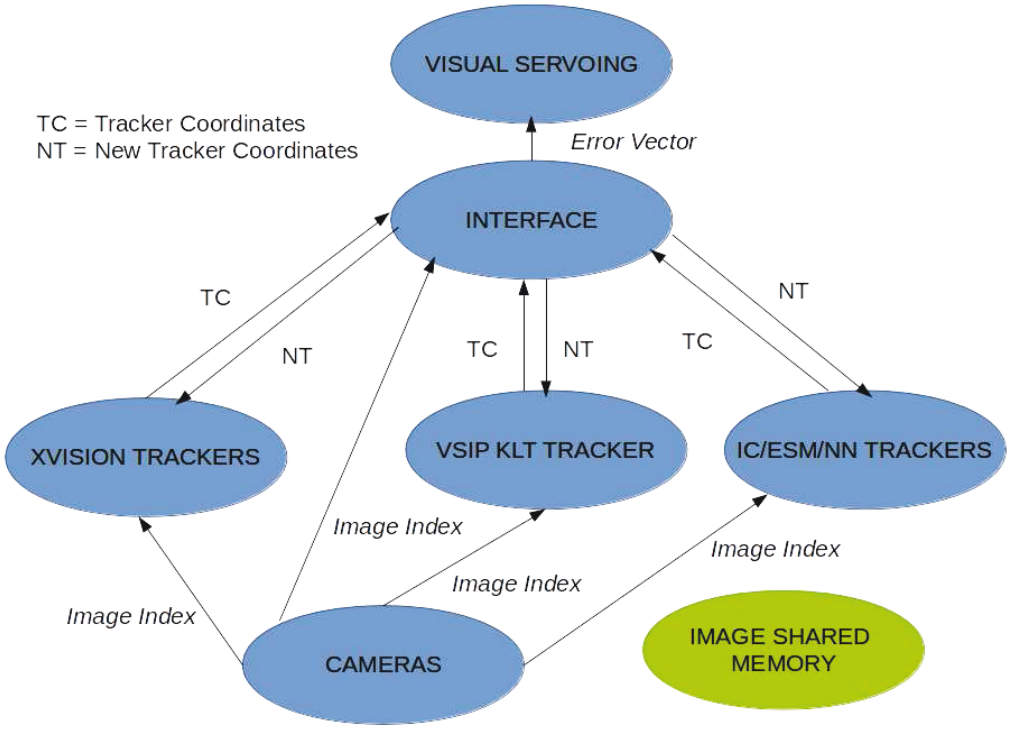


Figure 5.2: The image shows an overview of how our system is put together and the communication between different parts. ROS allows us to put these different processes in separate nodes and send messages between them. Images are stored in a shared memory that can be accessed by the tracker and interface nodes. The camera node publishes the index of the current captured image in the shared memory buffer.

lets us interface several heterogeneous trackers to the system and future users can add new ones to the list. The trackers we have added to the interface can bee seen in Table 5.1. We integrated the KLT point tracker [51, 52, 53] implementation from the Visual Servoing Platform (ViSP) [59] into our code. The IC [55] and ESM [54] patch tracker code had previously been implemented in our lab based on their published papers. The NN tracker [60] was developed and implemented by a student in our lab. The remaining trackers that we incorporated are from the XVison library [61].

When tracking objects in real time it helps the tracking algorithms to get image data fast. Therefore, the pipeline in which images are captured from the cameras and sent to other parts of the code, plays an important role in efficient tracking. One downside of the ROS system is that communication between nodes leads to inter-process copying of images. Originally images were published in messages from the

camera node and was read by the tracker and interface nodes. To avoid copying of images and so make our program more efficient, we put each image into a shared memory buffer. The camera node then only publishes the index of the current image into the buffer instead of publishing the image itself.

When combining primitive tasks to create more complex high-level tasks, we stack the different task errors together and provide these to the visual servoing. Visual servoing then minimizes this error in order to compute the next step for the robot. Once we had implemented and started testing the different tasks, we observed that the errors of different tasks were of different magnitudes. Whereas the point-to-point task error behaves linearly, the remaining task errors are quadratic or higher. For instance, the magnitude of a point-to-line error can be on the order of 10 000, whereas a point-to-point error can never exceed the image size. While working with different task specifications we found that error sizes would impact the visual servoing by some tasks dominating others. We tried a few different approaches to solve this problem. The details of the process is included in the results in Chapter 6, here we only mention the approach we arrived at. To be able to construct high-level tasks by combining geometric constraints, we move the origin to the middle of the image and scale the height to span $(-1, 1)$ and the width to $(width/height, width/height)$, where $width$ and $height$ represent the original image size. The different tasks still have different magnitudes. Hence, in the interface we provide the user with the ability to scale the errors of different tasks to give them different weight.

5.3 Description of the Interface

This section gives a description of the different features of visual task specification and how the user interaction works. While explaining the different features, we will define a possible task specification that serves as an example of how the interface can be used. Moreover, we will present mathematical expressions for the corresponding tasks and show how they can be combined when creating high-level tasks.

The interface lets us specify tasks for a robot to accomplish using visual servoing. The user can choose from the following tasks:

- point-to-point
- point-to-line
- point-to-ellipse

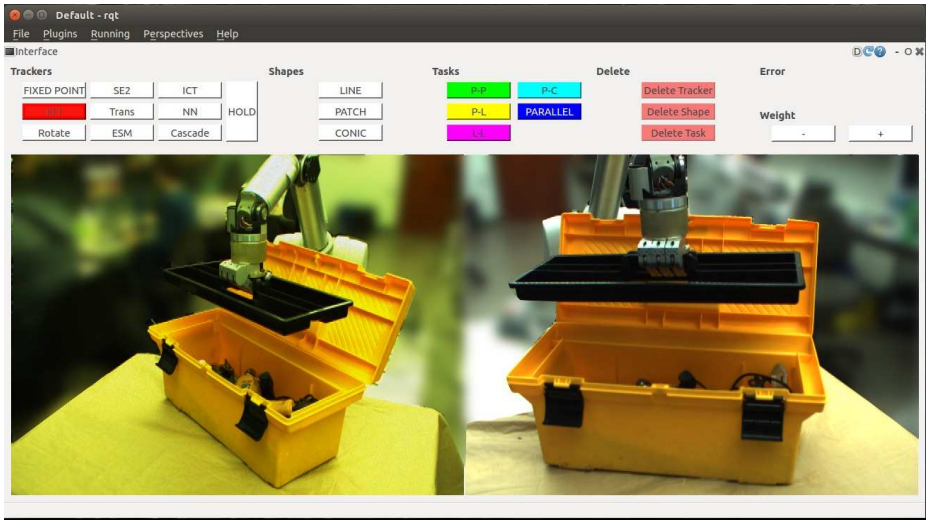


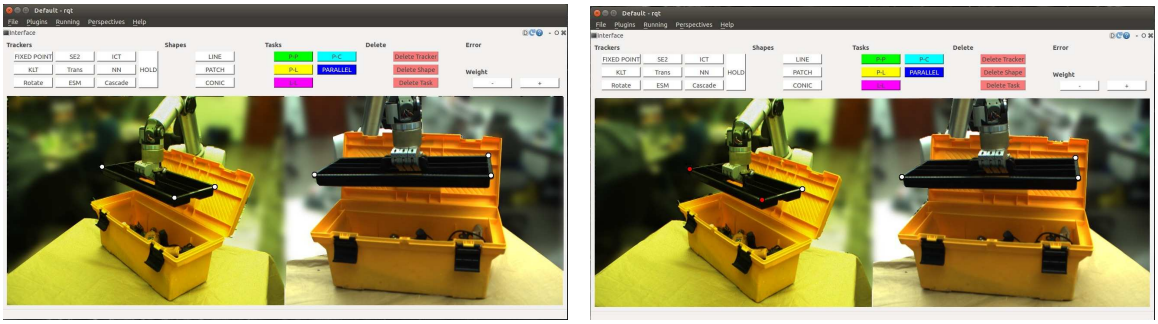
Figure 5.3: The interface layout. The interface displays images from two cameras. The top of the interface contains its various tools.

- line-to-line
- parallel lines

The interface shows the live image stream from two cameras that are observing a robot work space, see Fig. 5.3. Above the images we find the different tools that the interface provides. As the illustration shows, the example task involves moving the black tray that the robot holds into its place in the yellow toolbox. There are many possible combinations of primitive constraints that will create a suitable high-level task. Here we are not interested in the reasoning behind the selection of the specific constraints, but focus instead on how the interface is used to specify the tasks and how the mathematical expressions from Chapter 4 work in our application. We want to create a parallel lines task between the width of the toolbox and the width of the tray. Additionally, we specify a line-to-line task between the short edges of the toolbox and the tray. Finally, we add a point-to-point task between a corner of the toolbox and a corner of the tray.

5.3.1 Points

The basic entity in the interface is the point. Points are marked in the image with white dots with a black border, as seen in Fig. 5.4. They can be fixed points or they can represent trackers. Once points have been initialized they can be used to



(a) (b)

Figure 5.4: (a) Points in the image displayed as black and white circle markers. (b) Selected points, which change colour to red.

(a) (b)

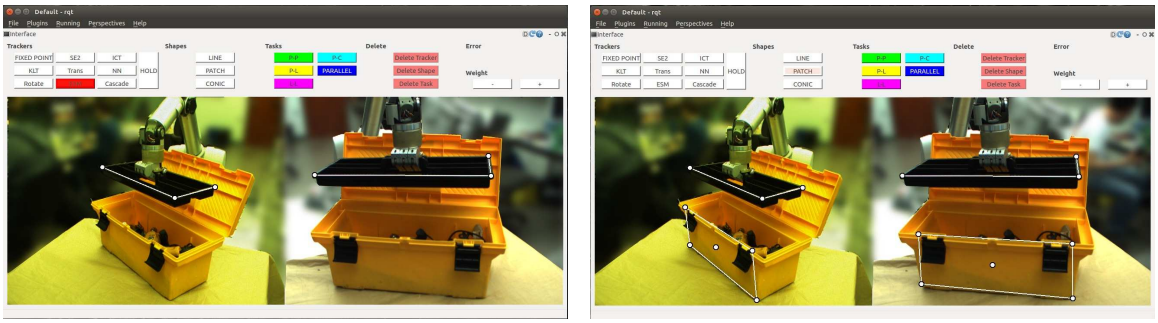
Figure 5.5: The user clicks on the KLT button (a) and then clicks in the image to initialize a new KLT point tracker (b).

construct geometric shapes (lines, patches and ellipses) and tasks (point-to-point, point-to-line etc.)

Points can be selected by clicking on them. When selected, a point changes colour to red, see Fig. 5.4. Several points can be selected simultaneously by holding the CTRL key. If several points are selected, then any action, like deletion, will be applied to all of them.

5.3.2 Trackers

Starting from the left in the interface, the first group of buttons represents the different trackers. To initialize a tracker the user clicks on the button named after the desired tracker and then clicks at the location in either of the two the images where he/she wants to place that tracker. This action creates one or more points, depending on



(a) (b)

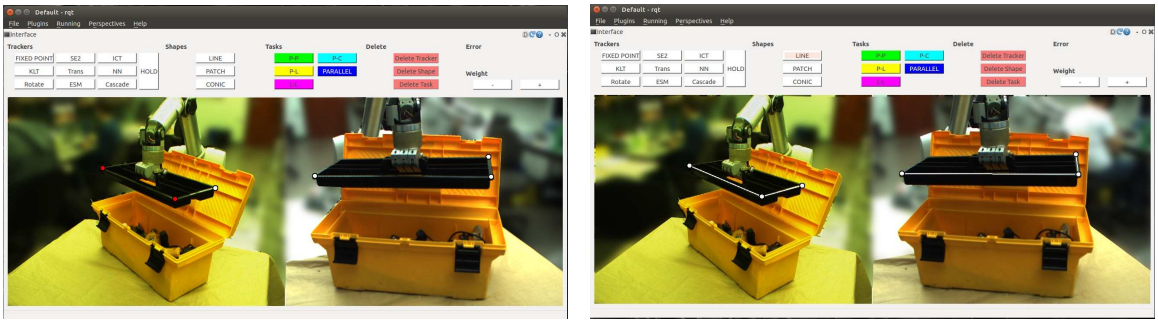
Figure 5.6: The user clicks on the ESM button (a) and then clicks on four points in the image to initialize a new ESM patch tracker (b).

the tracker type. The button FIXED POINT allows the user to set fixed points in the image. In our example task, we use trackers for the moving tray and fixed points for the static toolbox. If the user specifies a patch tracker (ESM, IC, NN etc.), then he/she must click on four points forming a patch for the tracker to be complete.

In the example task specification the user tracks the tray using KLT point trackers. The user clicks on the KLT button and then in the image to choose a location, see Fig. 5.5. A tracker or a fixed point is visualized by points in the image. Though not used in out particular task, if the user wanted to track a patch, then Fig. 5.6 shows the result of initializing and ESM tracker. In the case of a patch tracker, the interface displays four corner points connected by lines as well as the centroid of the patch.

The HOLD button can be pressed if the user wants to initialize several trackers of the same type in a row, so that he/she does not have to repeatedly click on the same tracker button. The hold is released by clicking again on the HOLD button or by clicking on any other button in the interface.

To remove a tracker or a fixed point, the user must click on the point marker and press the DELETE TRACKER button further to the right in the interface. This action will delete any tracker, shape or task associated with this point. That means, if the user clicks on one of the four corner points of a patch tracker and presses DELETE TRACKER, then that patch tracker (including all corner points, lines and the centroid) will be removed. The user can remove several trackers at the same time if he/she has selected several points with CTRL.



(a) (b)

Figure 5.7: (a) The user clicks on two points and then on the LINE button to create a shape. (b) shows the resulting line in addition to other created lines.

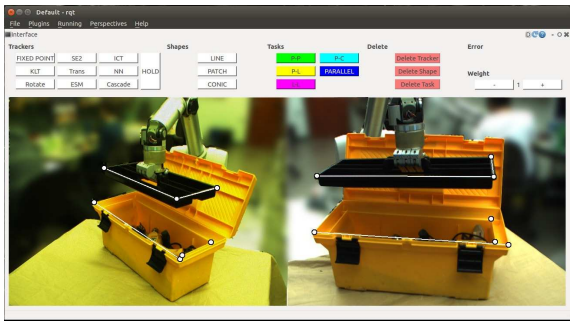


Figure 5.8: All the points ans shapes needed for the task specification have been created.

5.3.3 Shapes

As mentioned in the introduction, the purpose of the interface is to specify tasks for robot manipulation. Examples include point-to-point, point-to-ellipse and line-to-line tasks. These are basically geometric primitives. Hence, it is necessary for the interface to provide means to work with geometric objects. To that end we have implemented shapes. These shapes include lines between points, patches constructed from four points and ellipses defined by five or more points.

To create a shape the user selects the necessary number of points (e.g. 2 points for a line) and clicks on one of the buttons in the second group of buttons from the left. In Fig. 5.7, the user has clicked on two points to define a line on the tray that the robot holds. The figure also displays the resulting line in addition to other lines that were created. Fig. 5.8 shows the complete set of fixed points, trackers and shapes that have been set for our example specification. As a second alternative, the user

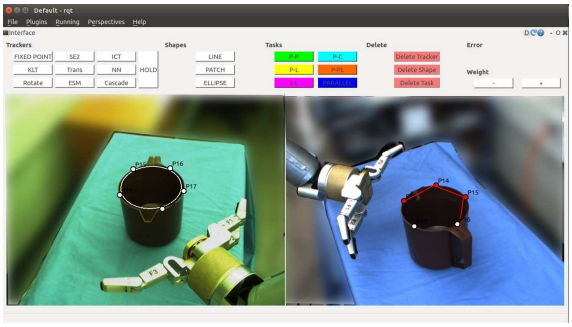
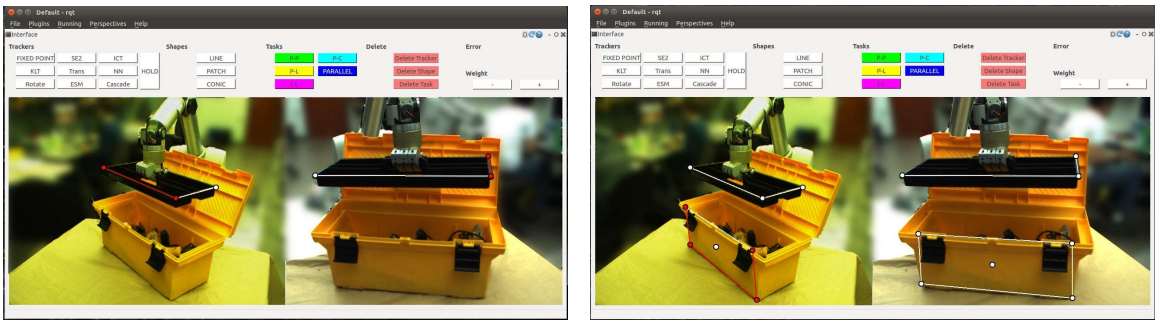


Figure 5.9: The user clicks on the ELLIPSE button without any points being selected. Then he/she can draw an ellipse by dragging the pointer to join points together.



(a) (b)

Figure 5.10: (a) shows the user's selection of a line. The line and connected points are selected and change colour to red. (b) shows that different parts of a patch, in particular individual lines, can be selected.

can click on the shape button when no points are selected. Then the he/she can drag the marker between two or more points to create a shape. In Fig. 5.9, the user creates and ellipse by dragging a marker between five points.

Shapes can be selected by clicking on them. For example, a user can click on a line and it will then be coloured red to indicated that it has been selected. The points connected to a shape will be selected along with it. A patch has four lines and a centroid. Each individual line on the patch can be selected. To make a selection of the whole patch at once, the user must click in the space inside the patch. Examples of shape selections are shown in Fig. 5.10

Similarly to trackers, shapes can be deleted by selecting them and then clicking the DELETE SHAPE button. Several shapes can be deleted simultaneously. If a point that is a associated with a shape is deleted, then that shape is automatically deleted. If one of the lines associated with a patch is deleted, then its other three

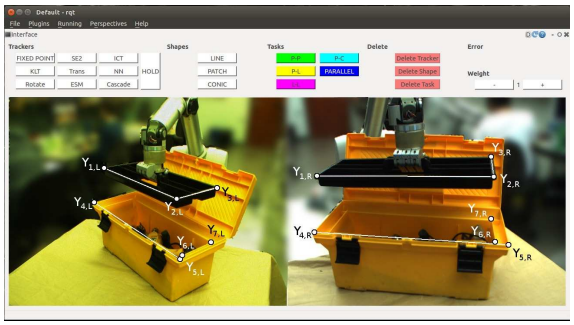
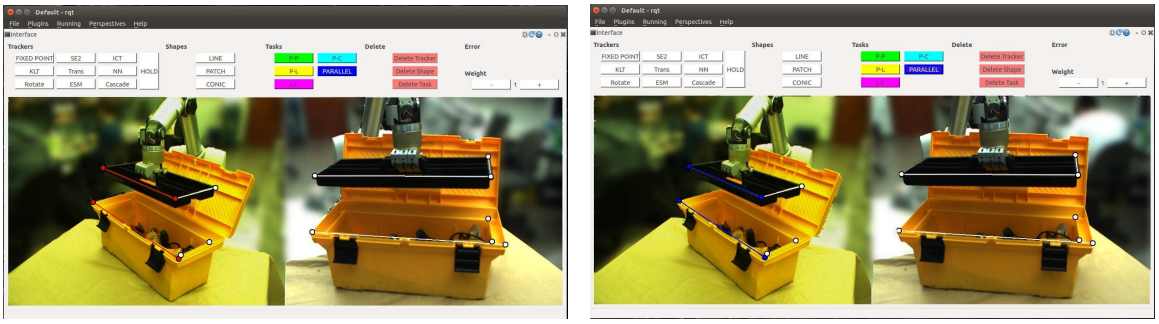


Figure 5.11: The image shows the tracked and fixed points marked with variable names. Each y_i is a 2D image space coordinate. L and R specify the left and right image.



(a) (b)

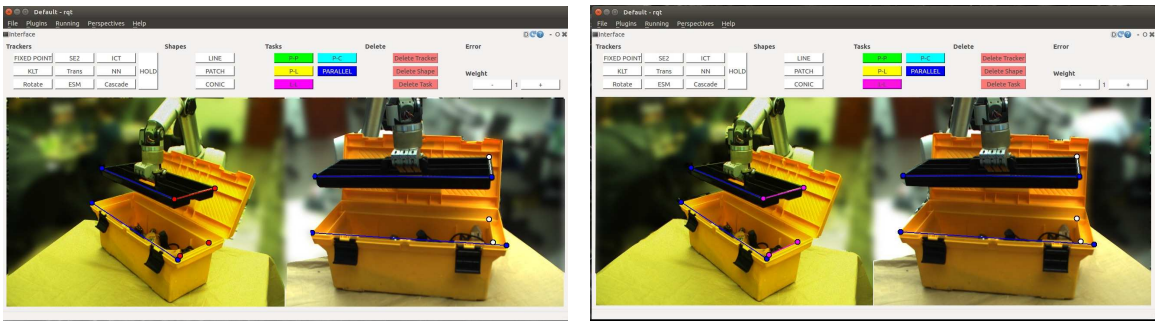
Figure 5.12: (a) The user selects two lines. (b) He/she then clicks on the PARALLEL button to get the resulting parallel lines task.

lines and the centroid are deleted as well. The points connected to a shape are not deleted along with it.

5.3.4 Tasks

The tasks are placed in the third group of buttons from the left. Each task has a colour. When points or shapes are added to a task, they will take on that task's colour. The available tasks are the following:

- point-to-point
- point-to-line
- point-to-ellipse
- line-to-line
- parallel lines



(a) (b)

Figure 5.13: (a) The user selects a line and two points. (b) He/she then clicks on the L-L button to get the resulting line-to-line task. Since only one line was created beforehand, the second line was added automatically when the user specified the task.

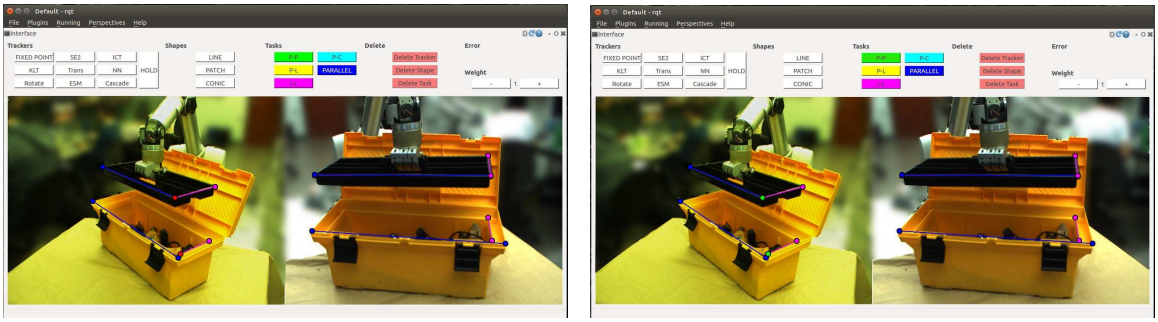
Now that we have created points and shapes, we can use these to form tasks. In Fig. 5.11 we have named the points that are included in the mathematical formulation of the tasks. A task needs a specific number of points. For example, a parallel lines task requires four points, all of which are the end-points of lines. Tasks can be created in two ways. In the first option the user can select points and shapes that make up a task and then click on the corresponding task button. For example, the user can click on two lines in the image and then push the PARALLEL button, which indicates a parallel lines task, see Fig. 5.12. Given the variables in Fig. 5.11, the parallel lines task can be formulated as follows:

$$E_{\text{par}}(\mathbf{y}) = \begin{pmatrix} E_{\text{par,L}}(\mathbf{y}) \\ E_{\text{par,R}}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} (y_{1,L} \times y_{2,L}) \times (y_{4,L} \times y_{5,L}) \\ (y_{1,R} \times y_{2,R}) \times (y_{4,R} \times y_{5,R}) \end{pmatrix}. \tag{5.1}$$

It is important that the points and shapes are selected in the order given by the task, which means that in a point-to-line task, for instance, the point is selected before the line. Once the task is created the points and shapes take on the colour of the task.

A second way to create a task is by clicking on a point or shape and then add it to a task by clicking on the corresponding task button. This way the user can build the task gradually. For example, the user can click on a point, then on the P-L (point-to-line) button. The point is added to the task and changes colour. However, the task is still not complete. The user can then proceed to click on a line and on the P-L button to add the line to the task. Now the task is complete.

As mentioned above, a task needs a certain number of points to be completely specified. For tasks that involve lines, these lines do not need to be pre-defined. If



(a) (b)

Figure 5.14: (a) The user selects two points. (b) He/she then clicks on the P-P button to get the resulting point-to-point task.

the user clicks on four points and adds them to a line-to-line task, then lines will automatically be drawn, if they do not already exist. In Fig. 5.13 we can see the second line being created only once the line-to-line task is initialized. The line-to-line task is located on a ledge inside the toolbox since this is where the tray needs to be aligned. Given the variables from Fig. 5.11 the error expression corresponding to the line-to-line task is:

$$E_{\text{ll}}(\mathbf{y}) = \begin{pmatrix} E_{\text{ll,L}}(\mathbf{y}) \\ E_{\text{ll,R}}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} (y_{2,L} \cdot (y_{6,L} \times y_{7,L})) + (y_{3,L} \cdot (y_{6,L} \times y_{7,L})) \\ (y_{2,R} \cdot (y_{6,R} \times y_{7,R})) + (y_{3,R} \cdot (y_{6,R} \times y_{7,R})) \end{pmatrix}. \tag{5.2}$$

The final task needed for the example specification is a point-to-point task between a corner on the tray and the corresponding corner on the toolbox, see Fig. 5.14. The point-to-point error is formulated as follows:

$$E_{\text{pp}}(\mathbf{y}) = \begin{pmatrix} E_{\text{pp,L}}(\mathbf{y}) \\ E_{\text{pp,R}}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} y_{5,L} - y_{2,L} \\ y_{5,R} - y_{2,R} \end{pmatrix}. \tag{5.3}$$

The complete specification combining all the primitive tasks can be seen in Fig. 5.15. Each individual task error from both images ((5.1), (5.2), (5.3)) is then stacked together in an error vector that is passed to the visual servoing.

$$E(\mathbf{y}) = \begin{pmatrix} E_{\text{par}}(\mathbf{y}) \\ E_{\text{ll}}(\mathbf{y}) \\ E_{\text{pp}}(\mathbf{y}) \end{pmatrix} = \begin{pmatrix} E_{\text{par,L}}(\mathbf{y}) \\ E_{\text{par,R}}(\mathbf{y}) \\ E_{\text{ll,L}}(\mathbf{y}) \\ E_{\text{ll,R}}(\mathbf{y}) \\ E_{\text{pp,L}}(\mathbf{y}) \\ E_{\text{pp,R}}(\mathbf{y}) \end{pmatrix}. \tag{5.4}$$

In addition to tasks involving points and lines, the interface also has a point-to-ellipse task. To illustrate this task we have added an example of the robot grasping a pitcher, see Fig. 5.16.

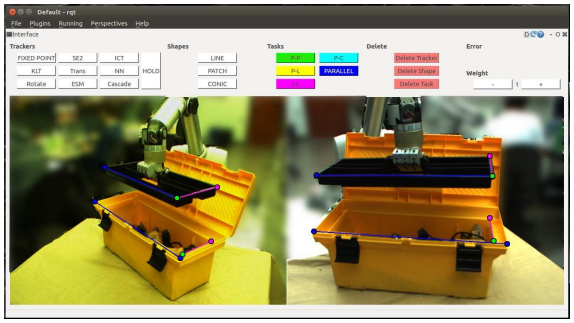


Figure 5.15: All the tasks have been specified and are viewed simultaneously. Since some points or shapes can be part of several tasks, the tasks may be overlapping and we cannot view all the individual tasks at once.

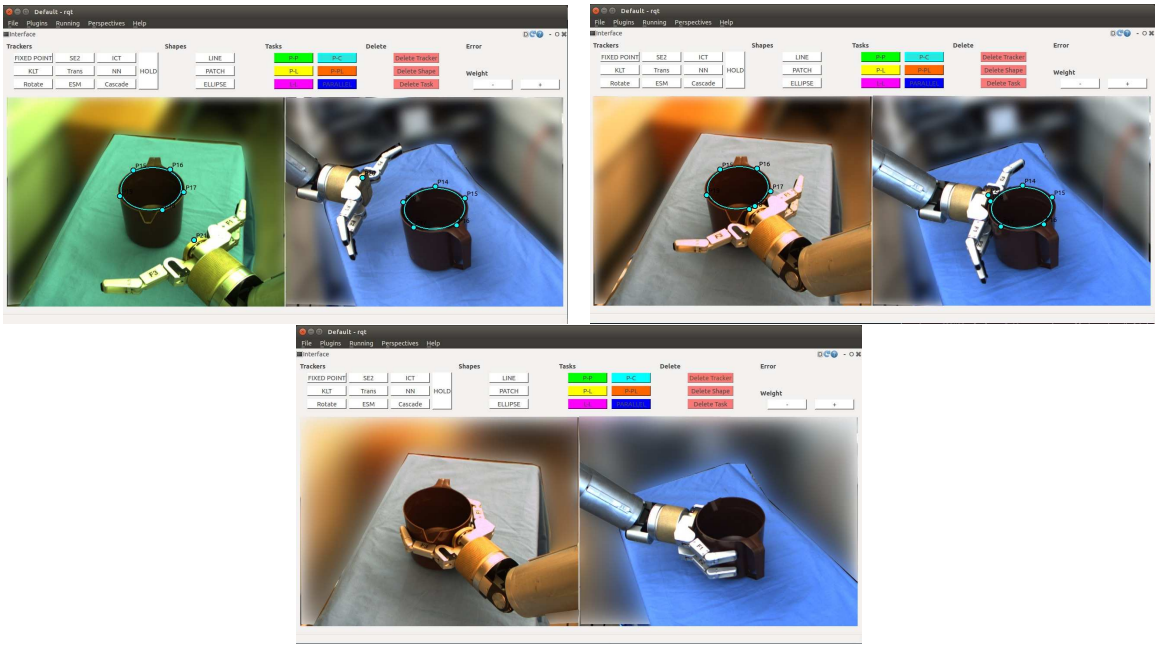


Figure 5.16: The user specifies a point-to-ellipse task to move the robot close to the pitcher in order to grasp it.

(a) (b)

Figure 5.17: By pressing the L key the user can enter layer view in one of the camera views. It allows the user to shuffle through and see each task individually. Any other points, shapes or tasks are invisible. (a) The user sees only the parallel lines task in the left image view. (b) The user sees only the line-to-line task in the left image view.

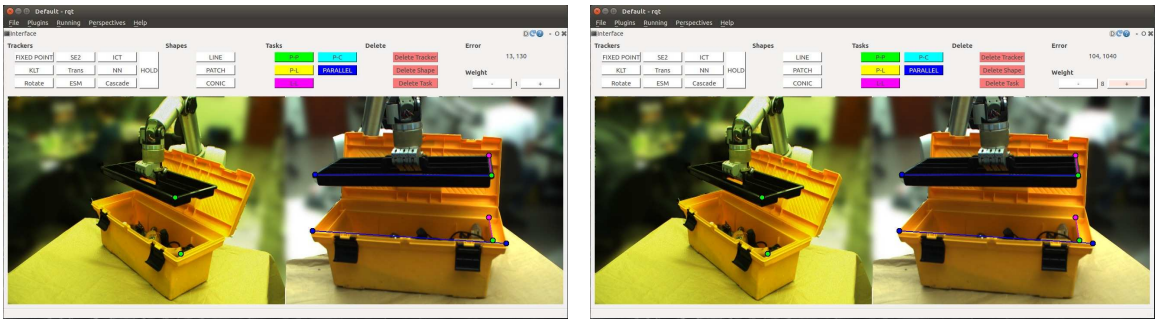
Similarly to points and shapes, tasks can be deleted by clicking on one (or several) of the points or shapes that are a part of a task and then push the DELETE TASK button. Sometimes one point or shape can be a part of several tasks. In this case all the associated tasks will be deleted. When deleting a task the underlying points and shapes remain.

5.3.5 Error Weights

As mentioned in the previous section we allow the user to give weights to different errors to control their behaviour when combined. For example, the user can choose to give more significance to a point-to-line task, than a point-to-point task or he/she can choose to make them similar. In the rightmost part of the interface we have an area that displays the task errors. Below the error display, there are two buttons that allow us to increase or decrease the scale added to a particular task error.

Once the full specification of a high-level task is finished, we have many tasks displayed simultaneously, see Fig. 5.15. In particular, since one point can be a part of more than one task, the tasks overlap. Hence, in order to be able to see all the different tasks as well as see and scale their errors, we have implemented a layer view.

The layer view is initiated by pressing the L key. The layer view will only appear in one camera view at the time, in particular in the view that was last clicked on. It will not appear in both images at the same time. The layer view allows the user to see one task displayed at the time (all other tasks, shapes and points are invisible),



(a) (b)

Figure 5.18: When in layer view the user can adjust the weight of the task being viewed. (a) The task error is displayed to the right in the interface. (b) By pressing the '+' button the user changes the scale to 8.

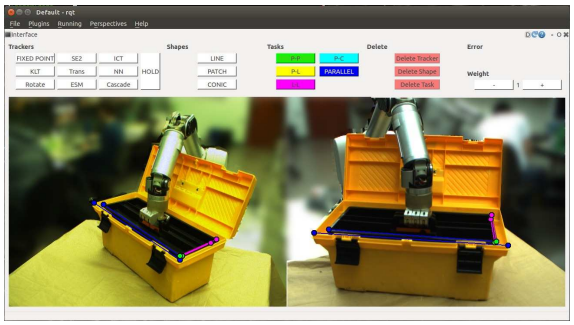


Figure 5.19: Complete view of the accomplished task.

shuffling through the tasks using the 'up' and 'down' arrow keys, see Fig. 5.17. When a particular task is displayed, its error will be visible in the menu and the user can adjust its scale with integer steps by pressing the '+' and '-' buttons. In Fig. 5.18 the point-to-point task has been scaled up by a factor of 8. In order to go back to normal view the user must press F for 'full view'. Pressing L will always take the user to the first task in the cycle. If the user is looking at the layer view in the left image, but then changes to layer view in the right image, the left image will automatically go back to normal view.

Finally, after visual servoing minimizes the task errors, the tray will be placed inside the toolbox as seen in Fig. 5.19. We can see that all the primitive tasks have been completed.

5.3.6 Keys

Finally, this is a list of the actions corresponding to different key strokes.

- CTRL: hold to select several points or shapes
- L: switch to layer view in the active image (the one last clicked in). Use up and down arrow keys to shuffle through the tasks. L always takes us back to the first task.
- F: Return to regular full view if all tasks.

5.4 Summary

In this chapter we have presented a user interface for visual task specification. The interface facilitates HRI in real world unstructured environments by putting the human-in-the-loop to specify high-level tasks for the robot. The interface allows a user to visually set trackers, create geometric shapes and construct complex tasks in a simple and flexible manner. In Chapter 6 we present our experiments with visual task specification and discuss their results.

Chapter 6

Evaluation

In this thesis we have created an interface for uncalibrated visual servoing using visual specification of tasks. Our work not only aims to create an interface for visual servoing, but also to explore what tasks can and cannot be done within this framework. Moreover, we examine the ease of user interaction, which contributes to the feasibility of the task specification approach and interface. This chapter details experiments we have completed to that end. Furthermore, we discuss common challenges that have arisen while working on the system and how we addressed them.

6.1 Experimental Setup

We have completed six fine manipulation experiments. As discussed in earlier chapters, we have created an interface that can specify a versatile set of tasks that span both coarse and fine manipulation. We also explained how visual servoing has the advantage over many other systems because of its potential to accomplish fine manipulation tasks that are out of reach for 3D sensors like the Kinect. Therefore we choose to focus our experiments on fine manipulation. Moreover, if we are able to do a certain task in fine manipulation, we would also be able to do it on a larger scale. We have included three examples of coarse manipulation tasks to show examples of such use of the interface, but do not go into depth with these tasks.

During the experiments we are able to perform some of the tasks, while others fail. We explain how we arrived at the different task specifications. For the tasks that fail we try to reason about why that is the case. These experiments are meant to investigate examples of useful tasks. We want to gain knowledge about how visual task specification works and what tasks it can and cannot solve.

For experimentation we can naturally only choose a finite set of tasks to explore. We want tasks that span a range of useful actions. Furthermore, we want to make sure that we test all types of geometric constraints provided in our interface. The tasks we have chosen are: inserting a marker in its cap, inserting a cube into a shape sorter, following a line, grasping a circular lid, grasping a small screw and cutting along a line. These actions translate to a variety of useful scenarios. For example, we can imagine line following being useful for applying glue to an edge or for welding. Aligning a tool for cutting can be used to cut a cake. The chosen tasks represent a variety of challenges such as orientation, accurate positioning, tracking, combining constraints and camera placement. In addition to the fine manipulation tasks, we provide three examples of coarse manipulation tasks, namely, grasping a box, grasping a cylinder and placing box on the table. Finally, we also complete two experiments that measure the performance of our system. We first test the accuracy of specifying and completing a marker in cap task. Second we complete a repeatability experiment that will give us the accuracy of our visual servoing system. The latter experiment will lend some context to the results of the first.

For each task that we complete we collect a set of data. We collect the coordinates of each tracker in the image, the task error, the robot joint states, the end effector pose, the Jacobian and the condition number of the Jacobian. We use this data to reason about a task's behaviour and present the data in detail where it is relevant. Finally, we record the number of clicks in the interface needed to specify a given task, as well as the number of geometric constraints that must be specified. This can be used to reason about the feasibility of the interface in terms of user interaction.

6.2 Results

6.2.1 Coarse Manipulation

Although our experiments focus on fine manipulation, we have added a few examples of coarse manipulation tasks to display the versatility of the interface and task specification technique. Below we give an overview of our results, but more detailed information is available in Section A of the Appendix.

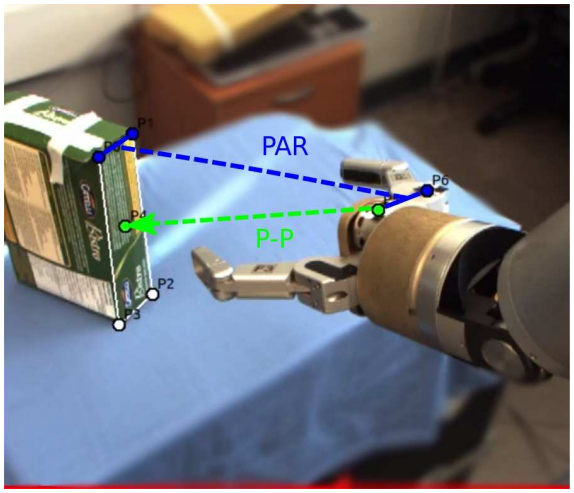
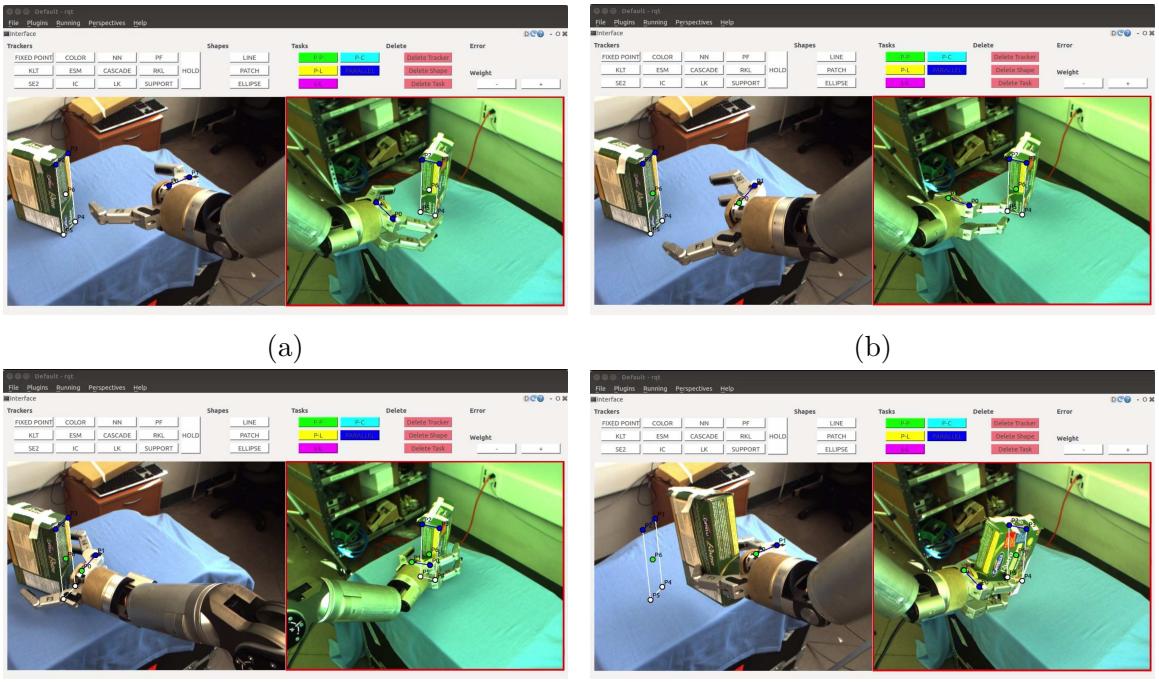


Figure 6.1: Task to grasp a box with the specifications illustrated.



(a) (b)

(c) (d)

Figure 6.2: (a) The task that we want to accomplish and the parallel lines (blue) geometric constraints we have specified. (b) The orientation has been modified and we add a point-to-point task (green) for translation. (c) Resulting gripper position after completing the task in sequence. (d) Grasp competed.

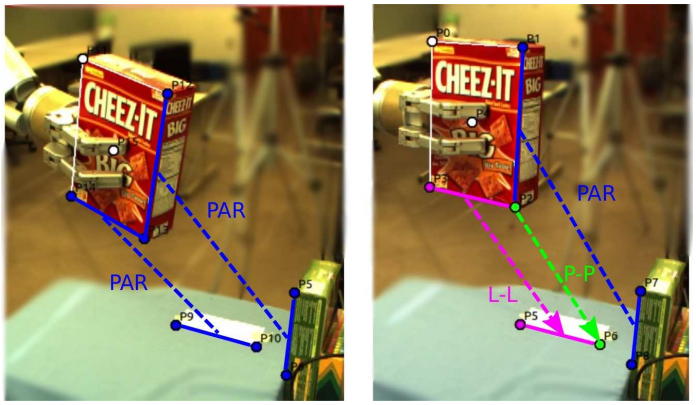


Figure 6.3: Task to place a box on the table with the specifications illustrated.

6.2.1.1 Grasp Box

In this task the robot grasps a box that sits on a table. The box is 21.0 cm wide, 13.9 cm tall and 5.2 cm deep. The distance between the robot fingers is 15 cm, so we have a little less than 5 cm between the box and each finger if the box is centred in the middle. The orientation of the gripper must be adjusted for the grasp to succeed. We specify a point-to-point task between the gripper and the box to translate the gripper to the box and we use a parallel lines task between a line on the gripper and a line on the box to adjust the orientation, see Fig. 6.1. Parallel lines tasks and point-to-point tasks have different error magnitudes. Combining the two tasks with no adjustments makes to the gripper move to the box, but the gripper orientation is barely affected. Next we tried to scale up the parallel lines error. This improved the result enough to grasp the box, but the gripper orientation could still be improved. Finally, we split the task into a sequence, where we adjusted the orientation with the parallel lines task first and then added the point-to-point task after. This allowed the parallel lines task to properly adjust the gripper orientation before approaching the box, see Fig. 6.2. More information, including plots, can be found in Section A.1.

6.2.1.2 Place Box

The second task we include is placing a box on a table, see Fig. 6.3. The box is 15.9 cm wide, 22.9 cm tall and 6.0 cm deep. The white rectangle on the table is the same size as the box and gives us an indication of how well the task is performed. We first specify two parallel lines tasks to improve the orientation of the box before moving it to the table, see Fig. 6.4. Afterwards we try three different approaches to place the

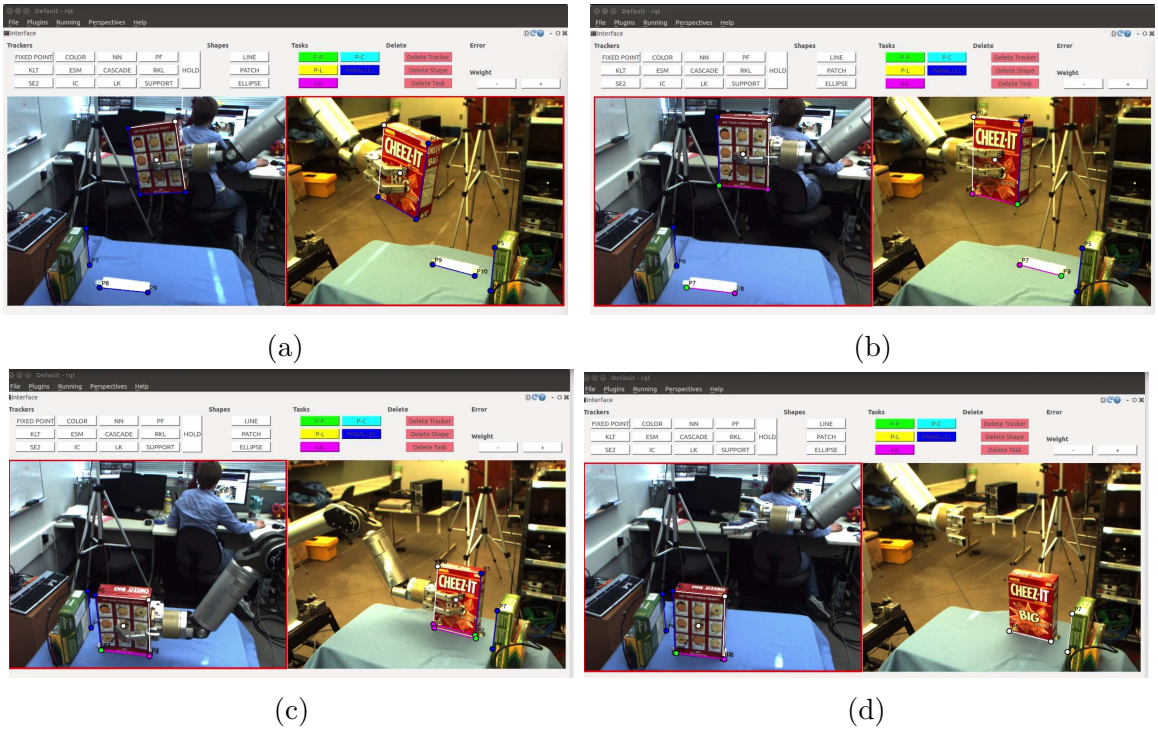


Figure 6.4: (a) We have specified parallel lines tasks to adjust orientation. (b) Orientation has been adjusted and we have specified the final task. (c), (d) Placing the box on the table.

box on the table, varying the tasks and number of DOF. We specify a parallel lines task between the vertical edge of the box and a box already on the table, a line-to-line task between the horizontal bottom edge of the box and a line on the table as well as a point-to-point task from the corner of the box to the desired goal on the table. We first complete the task using joints 1, 2, 4 and 5. We can place the box, but the final orientation of the box is not aligned in pitch. Therefore we add joint 6, which enables us to accurately place the box. The orientation is maintained and the box is placed on top of the white rectangle. The error in each direction is less than 3 mm.

Part of the yaw orientation that finishes the alignment happens just as the box touches down on the table. Although the line-to-line error has a similar magnitude to the point-to-point errors, it only has two elements in the error vector, while the point-to-point tasks have four. Therefore the line-to-line tasks adjusts part of the yaw orientation right at the end of the task once the point-to-point errors are small. More information, including plots, can be found in Section A.2.

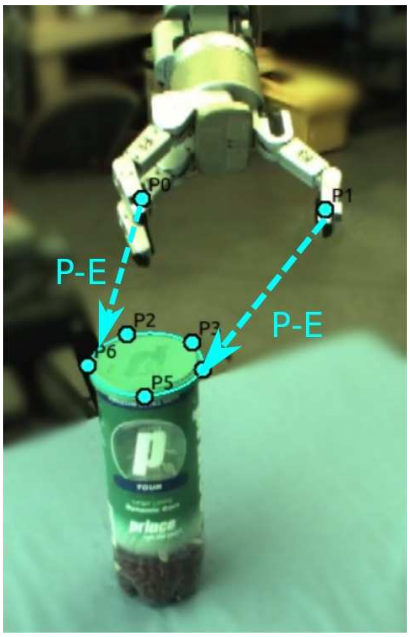


Figure 6.5: Task to grasp a cylinder with the specifications illustrated.

6.2.1.3 Grasp Cylinder

In this task we want the robot to grasp a cylinder from the side, see Fig. 6.5. The cylinder is 7.7 cm in diameter and 20.8 cm tall, whereas the gap between the robot fingers is 15 cm. Therefore, if the cylinder is centred between the fingers, we have about 3.7 cm space between each finger and the cylinder. We specify a point-to-ellipse task form each of the two robot fingers to the ellipse. Since the gripper is open when it moves to the cylinder, neither of the two point-to-ellipse task errors will converge completely to zero. By specifying two such tasks, the visual servoing will centre the gripper around the cylinder trying to minimize both tasks simultaneously, see Fig. 6.6. The starting position of the gripper is not very far from the cylinder and so we do not need to worry about the visual servoing reaching another local minimum (one finger converging to its goal point and leaving the gripper at the side of the cylinder). Reaching a good starting position can easily be accomplished by coarse translation of the gripper towards the cylinder using a point-to-point task. More information, including plots, can be found in Section A.3.

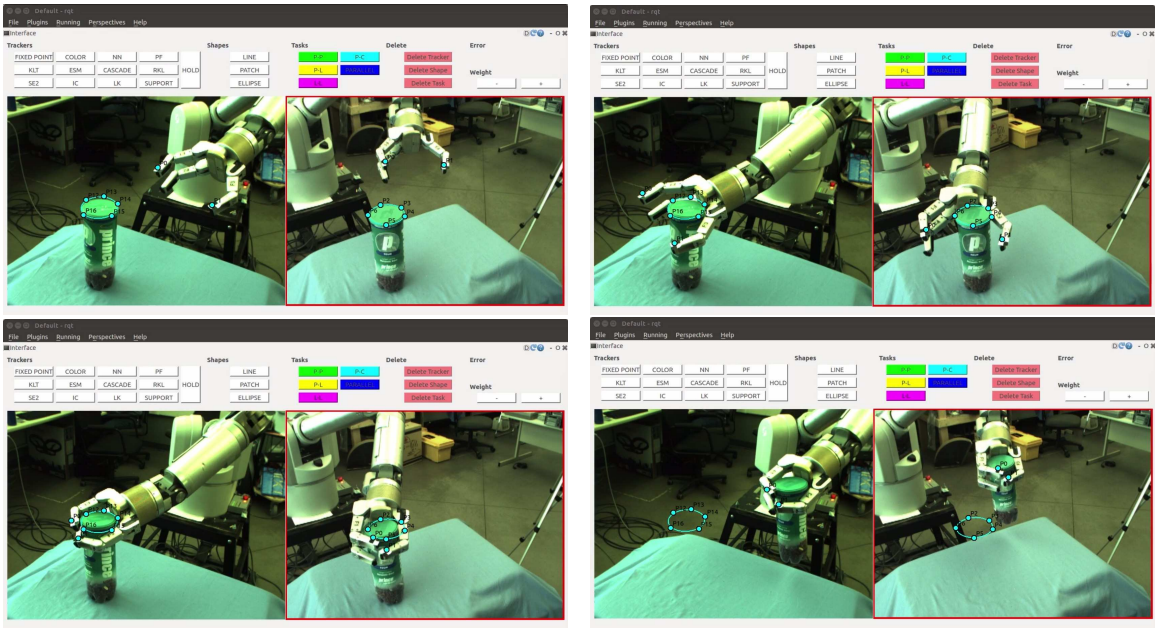


Figure 6.6: This figure shows the execution of grasping a cylinder. We specify two point-to-ellipse tasks in each image.

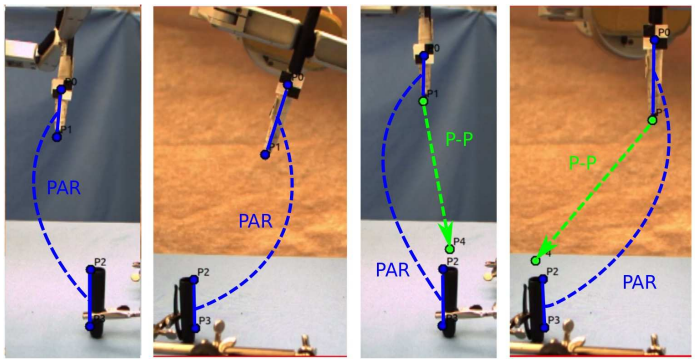


Figure 6.7: Task to a marker in its cap with the specifications illustrated.

6.2.2 Fine Manipulation

In the following sections we discuss the results of six fine manipulation experiments. The purpose of the experiments is to investigate how we can best use task specification to complete fine manipulation tasks. More detailed information on each experiment is provided in Section B of the Appendix.

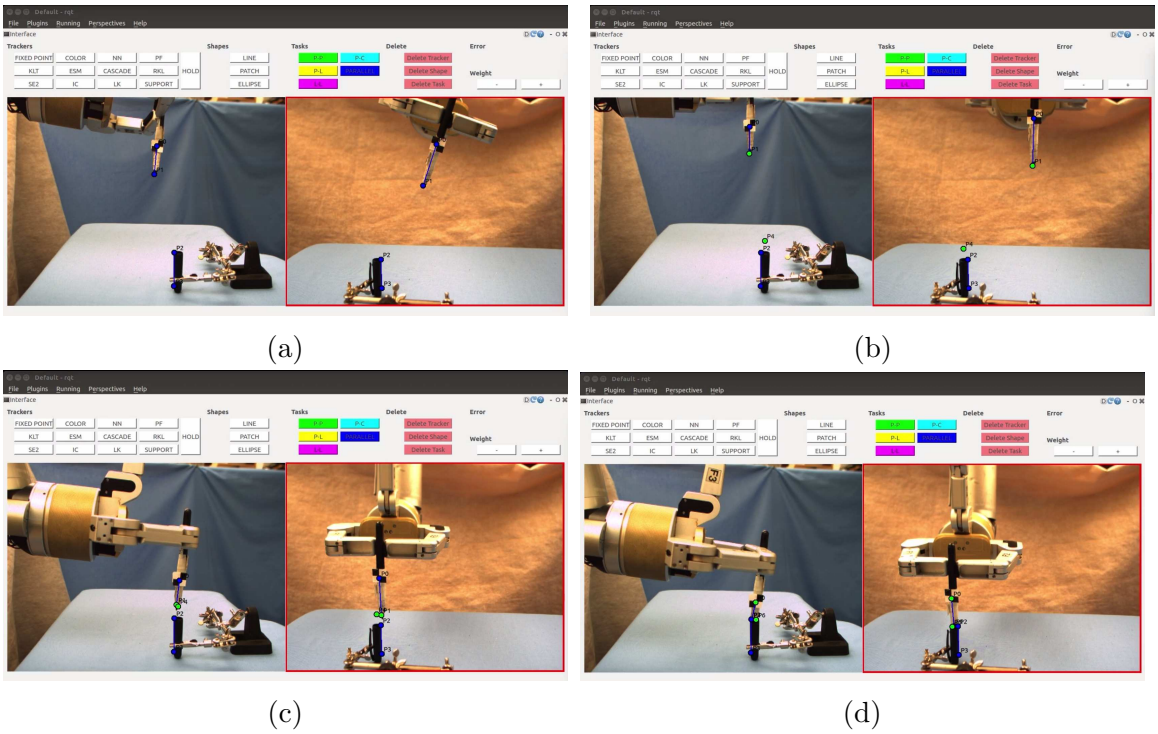


Figure 6.8: (a) Specify parallel lines task to adjust orientation. (b) Orientation is adjusted and we add a point-to-point task. (c) Marker has moved to the cap. (d) The marker's position has been adjusted and the marker has been inserted.

6.2.2.1 Marker in Cap

The first experiment inserts a marker in its cap. The cap has a diameter of 9.9 mm, while the marker tip's diameter is 2.5 mm and the diameter of the marker (including the added tape) is about 9.5 mm. This is similar to the experiments, where glue in a syringe is applied to a point [50] and putting a screwdriver on a screw [49]. The cap is placed vertically on a table and the marker is held in a pinch grasp between two robot fingers, see Fig. 6.8. In order to make the task as general as possible, the orientation of the marker is perturbed in two directions, i.e. it is not parallel to the cap. The robot must align the marker with the cap, centre the marker above the cap and insert it. The task from [50] only deals with orientation offset of the syringe in one direction, while the experiment in [49] seems not to take orientation into consideration, only displacement.

We complete the task in two different ways. First we divide the task into two segments and specify two parallel lines constraints to adjust the orientation. Then we add a point-to-point constraint and use 3 DOF translation to move the cap, see

Fig. 6.7. Second we specify the parallel lines and point-to-point constraints simultaneously and control 5 DOF of the robot.

When translating the marker to the cap we cannot set the goal point of the task to be directly at the centre of the cap opening. Because we are not able to track the exact tip of the marker, such a goal point would make the marker collide with the cap before it has been properly positioned in the table plane. The point-to-point goal point is specified as a free space point hovering slightly above the cap. Although the goal points are specified in free space and cannot be guaranteed to represent the exact same 3D point, they work well. Since they are specified in two different images the visual servoing will mediate whatever discrepancy there may be between the two point locations. During our later accuracy experiment in Section 6.2.3.1 we devise a strategy for setting a more accurate and consistent goal point.

The marker in cap task requires precise positioning before insertion. It is difficult to attain a sufficient level of precision in one large motion. Hence, when the translation to the cap has been accomplished, we need to adjust the location of the marker before we can finally insert it into the cap. Because we have a human-in-the-loop we can rely on the user to judge whether higher accuracy is needed. We can adjust the marker's placement by moving the the location of the goal point in the point-to-point task. This does not change the composition of tasks and so we do not need to change the Jacobian. Finally, we insert the marker into the cap by setting the goal point of the point-to-point task to create a downwards movement. Using adjustments we were able to position the marker with less than 3.0 mm error to the centre of the cap, which is sufficient for completing insertion.

As mentioned before, we complete the marker in cap experiment with two different strategies. Both succeed, but they have different advantages. Combining all the task at once saves us the extra step of adjusting the orientation before completing translation. Specifying the tasks in sequence has the advantage that we are using less DOF for each step of the task. This results in a smaller Jacobian conditioning number and so a better conditioned numerical problem. Moreover, finely adjusting the marker's position is easier in this case because the resulting movements are more intuitive to the user. When the user sets new goal points and we use 5 DOF, not only the position of the marker, but also its orientation will be affected. In the sequenced task, however, only the position changes. More information, including plots and more detailed images of the task execution can be found in Section B.1.

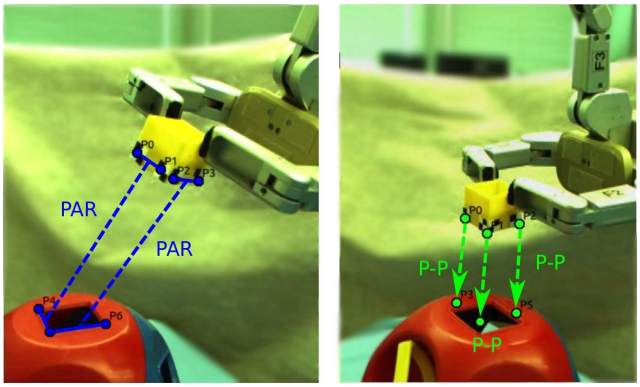


Figure 6.9: Task to insert a cube in a shape sorter with the specifications illustrated.

6.2.2.2 Cube Insertion

In this task we insert a cube into a narrow square opening in a shape sorter, see Fig. 6.9. The cube's edges measure 32.5 mm, while the length of the opening is 34.6 mm. This leaves only 2.1 mm leeway in each direction and so the task requires high precision. The cube's orientation is displaced in yaw, pitch and roll. Similarly to the marker in cap task, we complete the cube insertion in three stages. We adjust the cube's orientation, move it close to the square opening and finally insert it.

We align the cube's orientation with the square opening by specifying parallel lines tasks and using the last three DOF of the robot, see Fig. 6.10. This is where we run into the first challenge with the cube insertion task. As we adjust the orientation, the trackers are about to go out of view just as the task converges. The cube is especially challenging to track because we need to see two faces of the cube in each camera. To get around this we manually change the camera view, before setting the new task to translate the cube to the square opening.

We translate the cube to the shape sorter using 3 DOF. We specify three point-to-point tasks with goal points hovering just above the opening to avoid collisions. Point-to-point task are useful in this case because they are simple to specify and they work intuitively with translation. Moreover, they all have the same magnitude error. Once the cube is close to the goal we do the final fine positioning by adjusting the goal points. This adjustment is slightly more challenging than for the marker in cap task. Finally, we insert it by moving the goal points to create motion straight downwards.

In earlier iterations we tried various task combinations to insert the cube. We found that the cube needed to be properly oriented before we could translate it to the shape sorter. Specifying point-to-point tasks together with parallel lines or line-

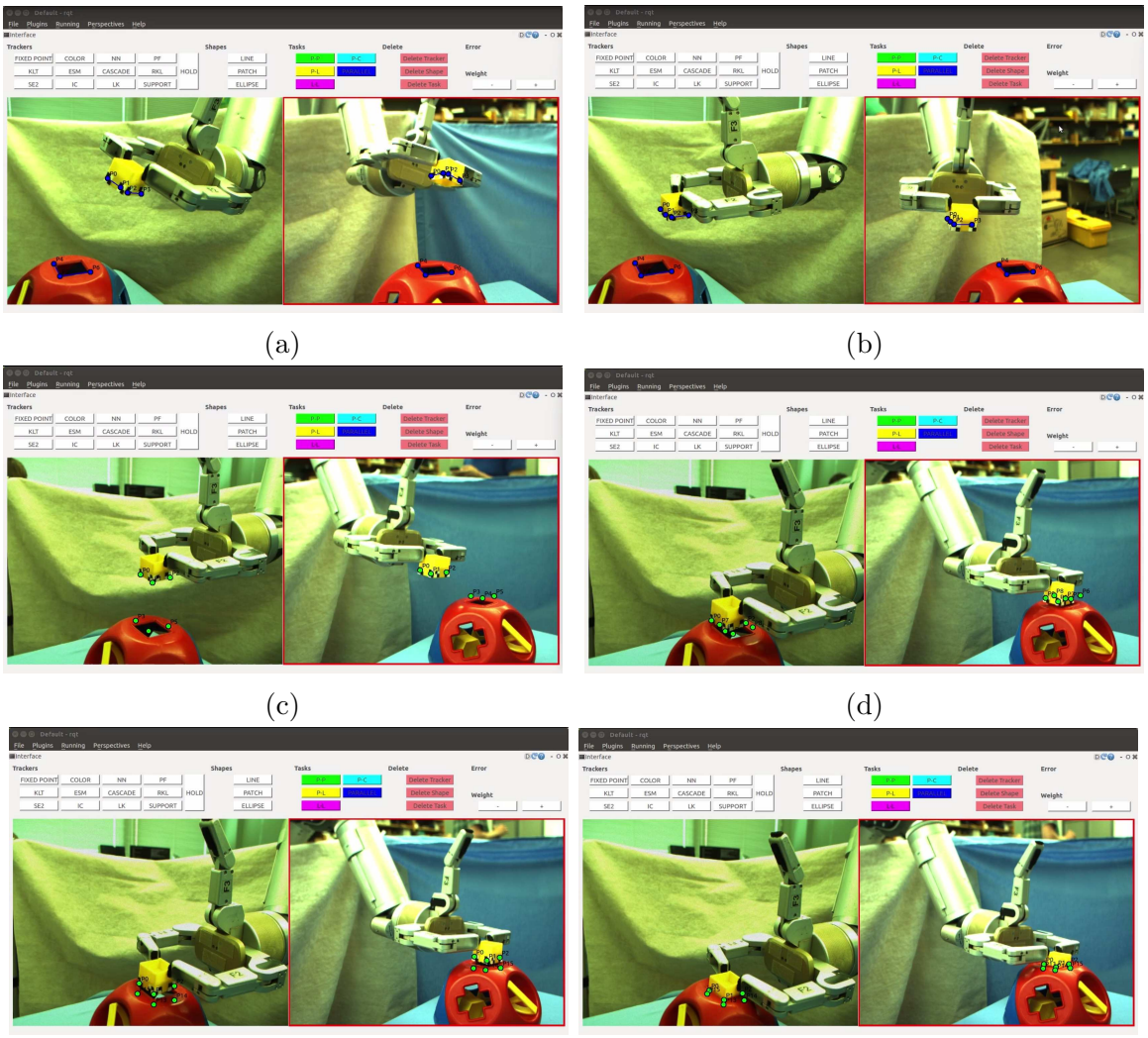


Figure 6.10: (a) and (b) show the orientation alignment. (c) The camera has moved and we specify the translation to the shape sorter. (d) Making fine adjustments to the cube's position. (e) and (f) show inserting the cube.

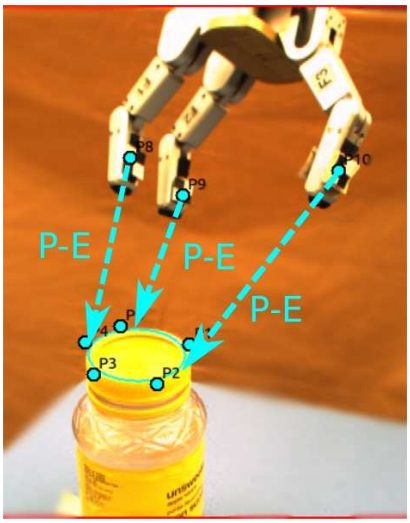


Figure 6.11: Task to grasp a circular lid with the specifications illustrated.

to-line tasks did not work. We could get close to the shape sorter, but the following position and orientation adjustments became too difficult when more DOF and a heterogeneous set of tasks were involved. More detailed explanation of the process to find the correct task specification as well as more pictures and plots can be found in Section B.2.

6.2.2.3 Grasp Ellipse

We want to grasp a circular shape from above, see Fig. 6.11. This can be useful for grasping a cylindrical container like a jar from above or to grasp the lid of a jar. We do our task with a jar of apple sauce. The lid has a diameter of 6.5 cm and 1.5 cm height. The jar is 14.7 cm tall. The distance between the opposing robot fingers is 15 cm, which leaves a gap of about 4.3 cm between each finger and the lid. We complete two different versions of the task. First we try and grasp the lid in the simplified case when the orientation of the gripper is already aligned with the plane of the lid. Second we attempt the grasp with an arbitrary orientation.

In the case where the orientation of the gripper is already aligned with the lid, we use 3 DOF of the robot. We specify point-to-ellipse tasks from the fingers of the robot to the ellipse projected into the image from the lid, see Fig. 6.12. Since the gripper is open the tasks will not converge completely to zero, but they will make the gripper centre on the ellipse. The gripper's initial position is fairly close to the ellipse and so we avoid any local minima where the gripper could converge to the

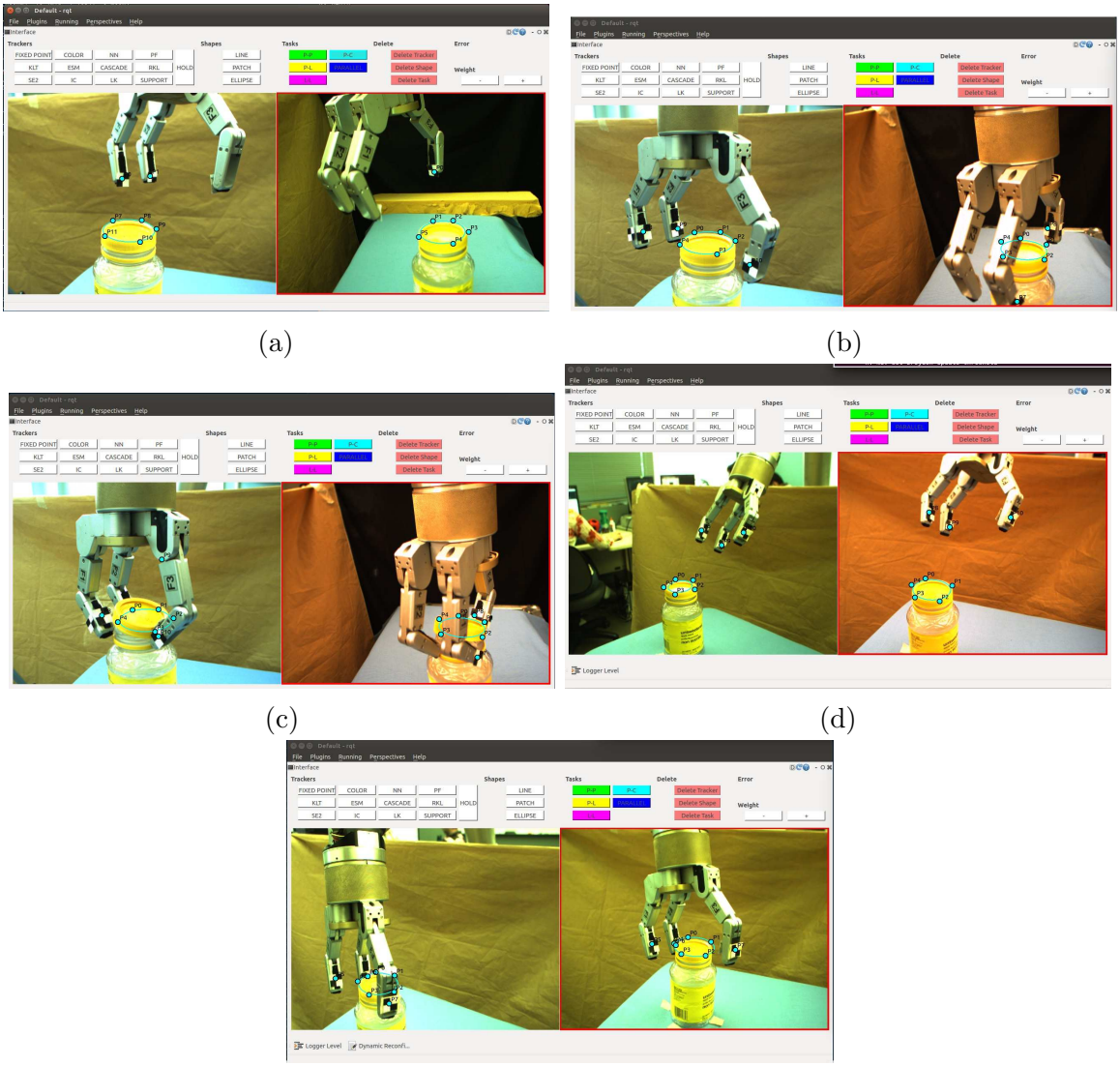


Figure 6.12: (a) The orientation of the gripper is already aligned. Set five point-to-ellipse tasks in two images. (b) The gripper moves to the lid. (c) Lid slips when grasped. (d) The gripper orientation is arbitrary. (e) The visual tasks converge in the image, but the camera views are ambiguous and so the robot cannot grasp the lid.

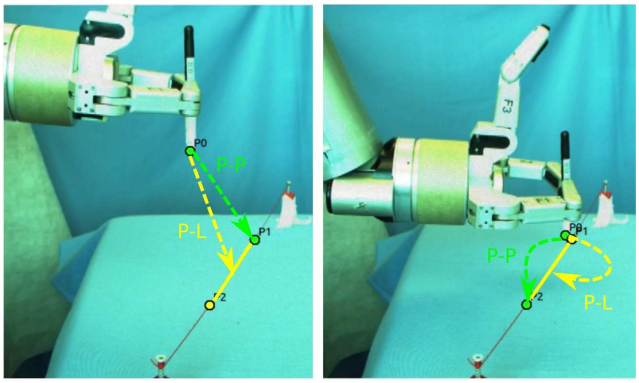


Figure 6.13: Task to follow a line with the specifications illustrated.

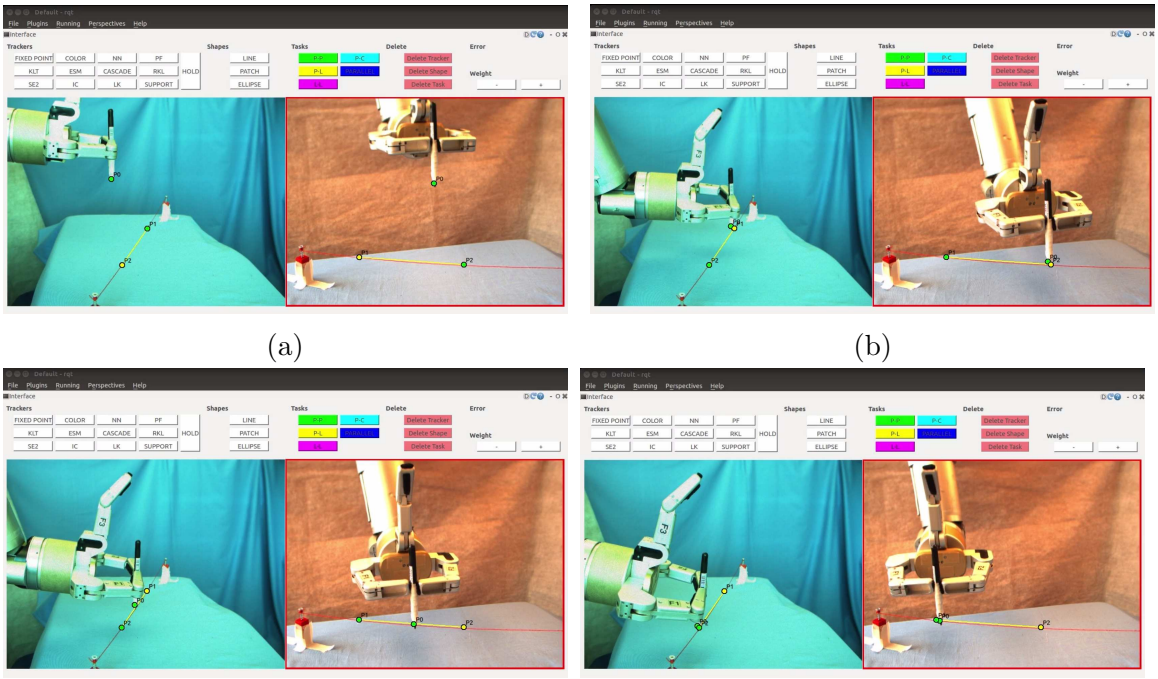
side of the lid. Getting the robot to a good initial position can easily be done with a point-to-point task coarse motion.

We set five point-to-ellipse tasks. The gripper moves to the ellipse. However, since the ellipse is specified at the top of the lid, the fingers are situated too high up to successfully grasp the lid. We adjust the marker locations to be higher up on the robot fingers to bring the fingers lower down when grasping. It is difficult to get an accurate enough placement to grasp the lid as it slips easily, but we can easily grasp the jar, see Fig. 6.12.

Next we attempt to give the robot gripper an arbitrary orientation. We find that we need to specify three point-to-ellipse tasks in each image in order to be able to align the gripper with the plane defined by the lid. Tracking all three fingers in two images is challenging and we need to place the cameras with fairly similar views to ensure trackers do not get lost. In the cases where we manage to execute the task without losing trackers, the result is not correct. Although the visual tasks converge in the image, the final position of the gripper is not ideal because the similar camera views give an ambiguous specification. See Section B.3 for more details on different approaches we tried.

6.2.2.4 Line Following

We want to make the robot follow a line constraint. One can imagine this being useful in cases such as applying glue to an edge or for welding. We let the robot hold a marker that will follow the line we have set up, see Fig. 6.13. We use a marker because it is possible to track its tip. Moreover, since the tip is round, we can track it equally from any direction. The tip has a 2.5 mm diameter. The line is made from



(a) (b)

(c) (d)

Figure 6.14: (a) We have specified a point-to-line task and a point-to-point task to the starting point. (b) The marker has reached the line and we set the new point-to-point task goal point. (c) and (d) show the movement along the line.

sewing thread and the marker follows it for about 16 cm.

We first bring the robot close to the line and then we make it travel along the line to a goal point. This means that we specify a point-to-line task from the marker tip to the line and a point-to-point task from the marker tip to the starting point on the line. See, Fig. 6.14. Once the marker has reached the line, we change the point-to-point goal to a point further down the line. We use 3 DOF control.

The robot moves accurately to the first point. When the robot moves along the line, it pushes forward a bit on the line as if it were making a slightly curved motion between the two points on the line. We suspected that this motion was caused by the point-to-point task and therefore implemented the line following only using two point-to-point tasks as well. Indeed, we found that the movement is the very similar in both cases. The point-to-point error is larger and therefore dominates. When the marker touches the starting point on the line, the point-to-line error has already been minimized. When we move to initialize the new point-to-point task, its error will naturally dominate.

We try different approaches to mediate the challenge of combining the point-to-line task with the point-to-point task. First we simply try to add more weight to the point-to-line task. Since the advantage of the extra weight diminishes when the point-to-line error approaches zero, this method does not help when we initialize the new point-to-point task. Next we try to modify the error function itself. We take the cube root of the point-to-line error to give it more similar behaviour to the point-to-line task. This works when the errors are large and we are approaching the first point on the line. However, once the point-to-line error approaches zero, it becomes very unstable. Even if the error were not to become unstable, the approach would still not solve our problem. Even if augmented, the point-to-line error would be close to zero and the point-to-point error would grow as soon as we set the new goal point. Finally, we tried to divide up the line in smaller segments. This makes the point-to-point error smaller, but the previous behaviour remains. The marker's path between each pair of intermediate points is still curved.

Though we did not have time to implement it, we leave it as a suggestion for future work to divide the line following task into a primary and a secondary task [62], which is a known approach in visual servoing. Task specification, as we have defined it so far, does visual servoing by minimizing a vector of stacked errors. The servoing therefore averages out the errors in the task. Hence, the weight of a task matters. Primary and secondary tasks allows us to assign importance to one task over another regardless of error size. In our case we could set the point-to-line task to be primary. The secondary point-to-point task will converge towards its goal, but only as long as it conforms with the constraints set by the primary task. This way the point-to-line task will control the line following motion despite its low weight. See Section B.4 for more details on the line following experiment .

6.2.2.5 Grasp screw

In this task we use the robot to grasp a screw, see Fig. 6.15. The screw is 8.05 mm in diameter and 19.12 mm tall (above the wooden base). This task requires high precision as the screw is small. Furthermore, we need to make sure that the orientation of the gripper is correct for the grasp to succeed. Otherwise the screw could slip as we try to grasp it. In our set-up the possible camera views have some limitations. The cameras cannot be placed too far from each other as they both must see the front of the robot fingers.

We first specified a parallel lines task to change the orientation of the gripper only

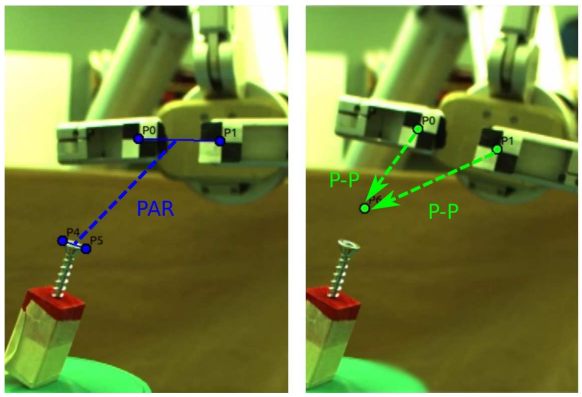
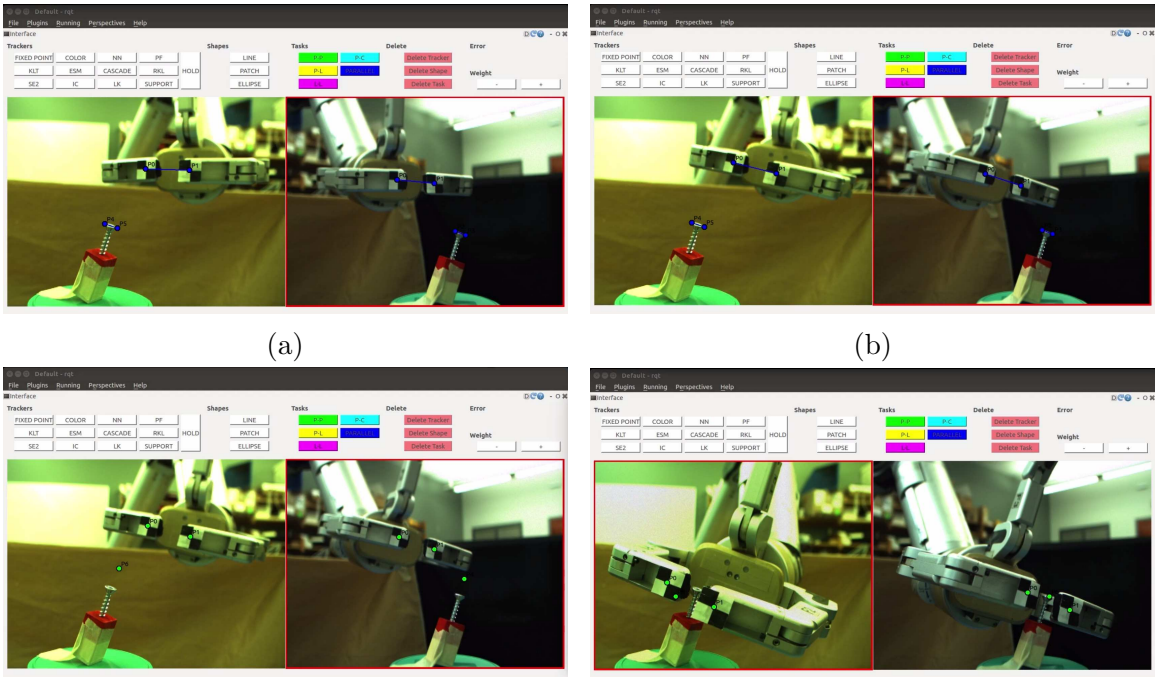


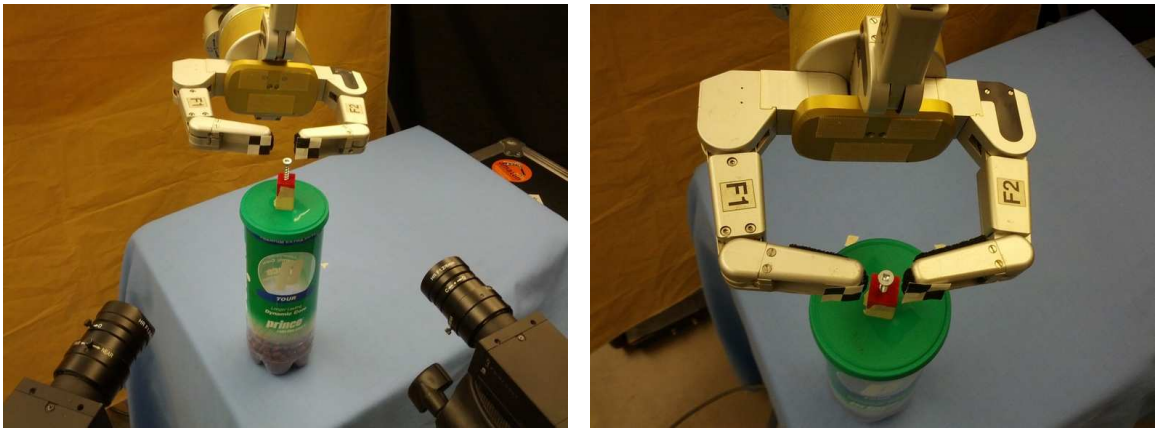
Figure 6.15: Task to grasp a screw lid with the specifications illustrated.



(a) (b)

(c) (d)

Figure 6.16: (a) We define a parallel lines task to adjust the orientation of the gripper. (b) The task has converged. (c) We specify point-to-point tasks to translate the gripper to just above the screw. (d) We move the gripper downwards to the screw and adjust the positioning to properly place the gripper for grasping.



(a) (b)

Figure 6.17: (a) Shows where the gripper converges to with the point-to-point translation to the screw. Because of the camera view from the side, this does not correspond to the desired position for grasping. (b) By making adjustments to the point-to-point task goal points we are able to properly centre the gripper.

using joint 7, see Fig. 6.16. When the orientation of the gripper and the screw is aligned, we specify two point-to-point tasks and use joint 1, 2, 4 and 5 of the robot. The point-to-point tasks translate the gripper to a free space point above the screw. Then we move the gripper downwards to the screw. Since we set a point on each robot finger, the robot centres its position around the screw.

The robot gripper converges to its goal and centres on the screw, but because both of the cameras are located in front of the robot gripper, the fingers do not move far enough forward to actually grasp the screw. Therefore we adjust the point-to-point goal positions to create more forward movement. In Fig. B.28, we can see the difference between the initial and adjusted gripper position.

Finally, we tried to adjust one of the cameras to get a top view. We wanted to find out whether forward positioning of the gripper would be more accurate and adjustments in position more intuitive for the user. One downside to having a top view was that the gripper's orientation would change while we approached the screw. Hence we need to add a parallel lines task to the specification even tough the screw's orientation was aligned with the gripper. The gripper did get some more forward movement as a result of the new camera view, but adjusting the final position before grasping did not become easier. See Section B.5 for more details.

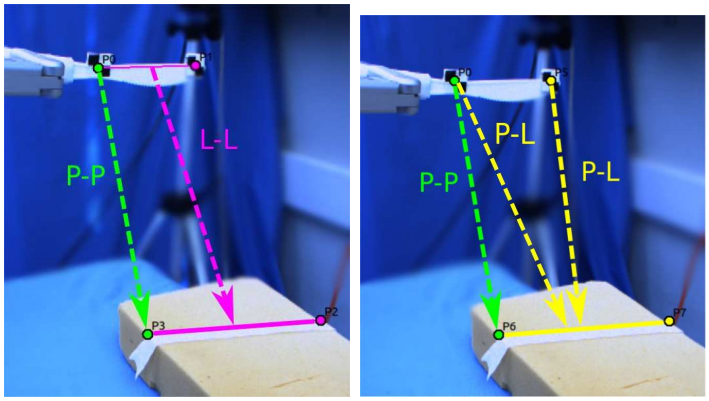


Figure 6.18: Task to cut along a line with two specifications illustrated.

6.2.2.6 Cut Along Line

The final task replicates the action of cutting along a line, like for instance cutting a piece of cake. The robot uses a tool, in this case a knife, that it holds in its grip. For systems that rely on 3D vision using tools can be challenging. We cannot calculate the pose of a tool from forward kinematics alone. A model of the tool would be needed. In our case the robot can grasp any tool or object and we do not have to make any modifications to our procedure. We only need to make sure that we can track the tool.

In our experiment the robot holds a plastic knife. It is 18.5 cm long and measures 1.8 cm at the widest part of the blade. We have put markers on the knife to be able to track it. The knife is supposed to move down and position itself along a line on a piece of foam, see Fig. 6.18. For cutting the knife needs to be correctly oriented and it must also move to the correct point along the line. There are several options for how to accomplish this task. We have tested three different ways of specifying the task, see Fig. 6.19. The first combines a line-to-line task for controlling the orientation of the knife with a point-to-point task for driving the robot to the correct position along the line. The second specification combines two point-to-line tasks with a point-to-point task. The final specification consist of a parallel lines task and a point-to-point task.

A line-to-line task can be implemented in several ways. It can be created by stacking the errors of two point-to-line task or by stacking the errors of a parallel lines task and a point-to-point task. In our system we have chosen to sum the errors of two point-to-line tasks. Hence, the cutting task will give us a chance to compare the behaviour of our line-to-line implementation and the combination of two point-to-line

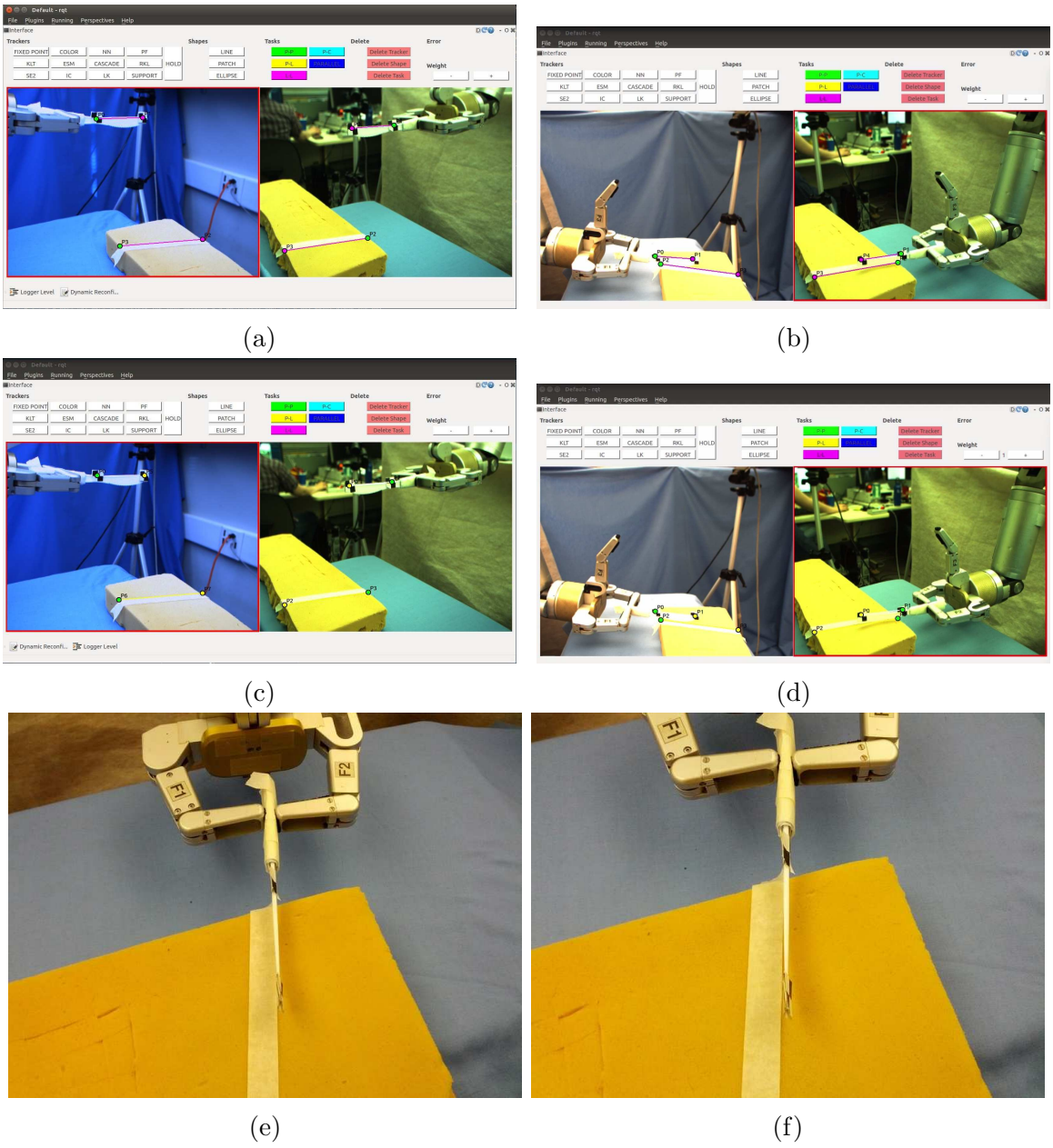


Figure 6.19: (a) and (b) We complete the cutting task by using a scaled line-to-line task together with a point-to-point task. (c) and (d) We complete the task with two scaled point-to-line tasks combined with a point-to-point task. (e) and (f) The results from a different view angle.

tasks.

We started our experiments by exploring different combinations of robot DOF and task specifications. Depending on the camera view, combining point-to-line and line-to-line tasks with a point-to-point task does not always succeed as the translation completes before the knife is properly aligned. Therefore we tried to add a higher weight to the point-to-line and line-to-line tasks. This approach succeeded. We also tried using an intermediate point, where we set a goal point for the translation half way between the knife's initial position and the final goal. Hence the translation is divided into two segments and the point-to-point error decreases. This also worked. Finally, combining a parallel lines task with a point-to-point task succeeded without need for any modifications. See Section B.6 for more details.

6.2.3 Accuracy

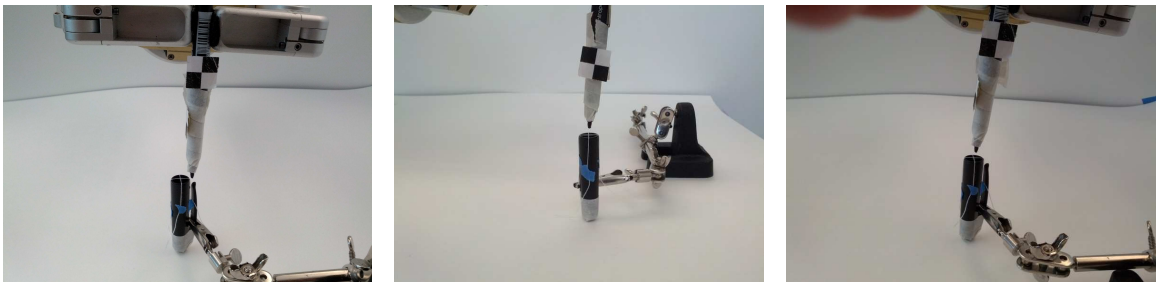
Though most of the experiments in this thesis aim to explore how to use visual specification to complete tasks, we also include two experiments to measure performance. First we complete an experiment to measure the accuracy of specifying a task using visual specification with our interface. Second we complete a repeatability experiment for basic visual servoing with our robot arm. The second experiment will provide some context to the results of the first.

6.2.3.1 Task Specification Accuracy

We want to conduct experiments to find how accurately we can accomplish a task that requires us to combine more than one visual task and that also requires user input. We complete the positioning for the marker in cap task ten times to see how often we can succeed and how accurately we can position the marker in the table plane above the cap before insertion. The cameras are placed at 31 cm and 37 cm from the cap. The cap has a diameter of 9.9 mm, while the marker tip's diameter is 2.5 mm and the diameter of the marker (including the added tape) is about 9.5 mm. The set-up and the task can be seen in Fig. 6.20.

The user specifies a parallel lines task to align the orientation of the marker with that of the cap. She also adds a point-to-point task to translate the marker to the cap. Since we are not able to track the exact tip of the marker, we put the goal point above the cap to avoid collisions. We use 5 DOF of the robot. During our initial work to complete the marker in cap task, we chose goal points by setting free space points.

Figure 6.20: The image shows the task and the set-up.



(a) (b) (c)

Figure 6.21: (a) Front view of one of the poorer positioning results. (c) and (d) Front and side view of a better result. We can see that one direction as more error than the other.

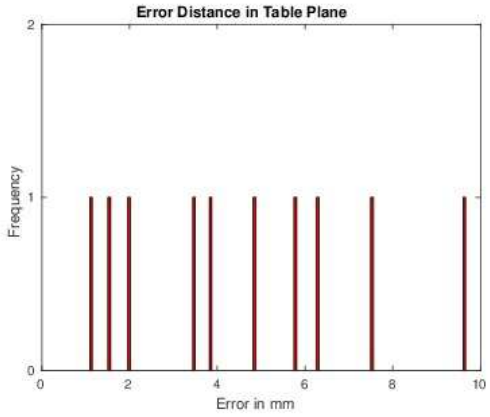


Figure 6.22: A plot of the error from the centre of the cap to the marker tip in the table plane. The x axis show the range of the errors in mm. A bar is displayed at each recorded error.

This works as we can make adjustments to the marker's location. However, for this experiment, we want a more consistent and precise goal point. We accomplish this by an affine approximation. We mark two lines, each at one side of the cap, which are two occluding contours. We then extend these to lines by a third point, placed at distance 0.3 times the line length away from the top end-point. Then we find the midpoint between these two extended points.

We measure the marker's distance from the centre of the cap in the table plane. We mark the centre by crossing two pieces of thin sowing thread and measure the distance with a calliper. Since we measure by hand this naturally affects the resulting accuracy, but we find that the collected errors still provide useful data for a real world task.

Our experiment succeeded in all ten cases, meaning that the task did not diverge and we did not have to stop short because a tracker got lost. The average error was 4.6 mm with variance equal to 7.5 mm and standard deviation 2.7 mm. A plot of the resulting distance for each experiment can be seen in Fig. 6.22. In all the experiments the marker's location was well positioned in one direction of the table plane, whereas it had a larger error in the other, see Fig. 6.21. When examining the image errors we found that the point-to-point task's x component in the left-most image was larger than the other errors. This agrees with the observation of the marker's error being less accurate in this direction.

When inserting the the marker into the cap the marker does not need to be perfectly centred on the cap. Given the size of the marker, it must be 3.0 mm or closer to the centre. In 3 out of out 10 experiments the cap could have been inserted without any further adjustments. The user clicked an average of 58.6 times in the interface to set up the task.

We have 4.6 mm of error in the table plane when positioning the marker above the cap. What causes this error? As we mentioned earlier, the goal point that we calculate for the point-to-point task is an affine approximation and hence not exactly correct. We rely on the user to click on points in the image. In particular the user marks fixed points on the cap. These point locations will have errors. To track the points higher on the marker we use tracking markers, whose placement will also have some error. Whenever the user starts a point tracker, the interface will not track exactly the point that the user chose, but instead find the best point to track within a small window. We want to track the tip of the marker. This is not possible and instead the tracker is above the tip. Moreover, the marker tip has width of about 2.5

(a) (b)

Figure 6.23: (a) Overview of the set-up with two cameras, a dial meter and the robot. (b) The box pushes on the dial meter in the direction of motion to collect measurements.

mm and the tracker might not centre exactly in the middle. The trackers also drift throughout the visual servoing. See Section C.1 for more details.

6.2.3.2 Visual Servoing Repeatability

In the second experiment we test the repeatability of 6 DOF positioning using visual servoing with our robot arm. We complete an experiment that is similar to those presented in [41] and [29]. [41] explains that repeatability tests the robot's ability to go back to a pose it has previously attained, while accuracy tests the robot's ability to achieve any specified pose. In the accuracy experiment and when otherwise using our interface, we find that the system's performance is affected by a user's ability to set correct visual goal points as well as tracking. In the repeatability experiment we test the performance of visual servoing itself without being affected by user input. We also try to improve the tracking. This will provide a baseline for how well the visual servoing works and therefore serves as a good comparison with the accuracy results of the previous experiment. In particular, we will gain insight into the performance of combining different visual tasks and having the user click on goal points versus completing basic visual servoing.

We complete two experiments. One experiment measures the accuracy in the depth direction of the cameras and the other measures the accuracy in the sideways direction. We measure the accuracy by servoing the end-effector towards a 0.001" dial meter that is set up in the direction we want to measure.

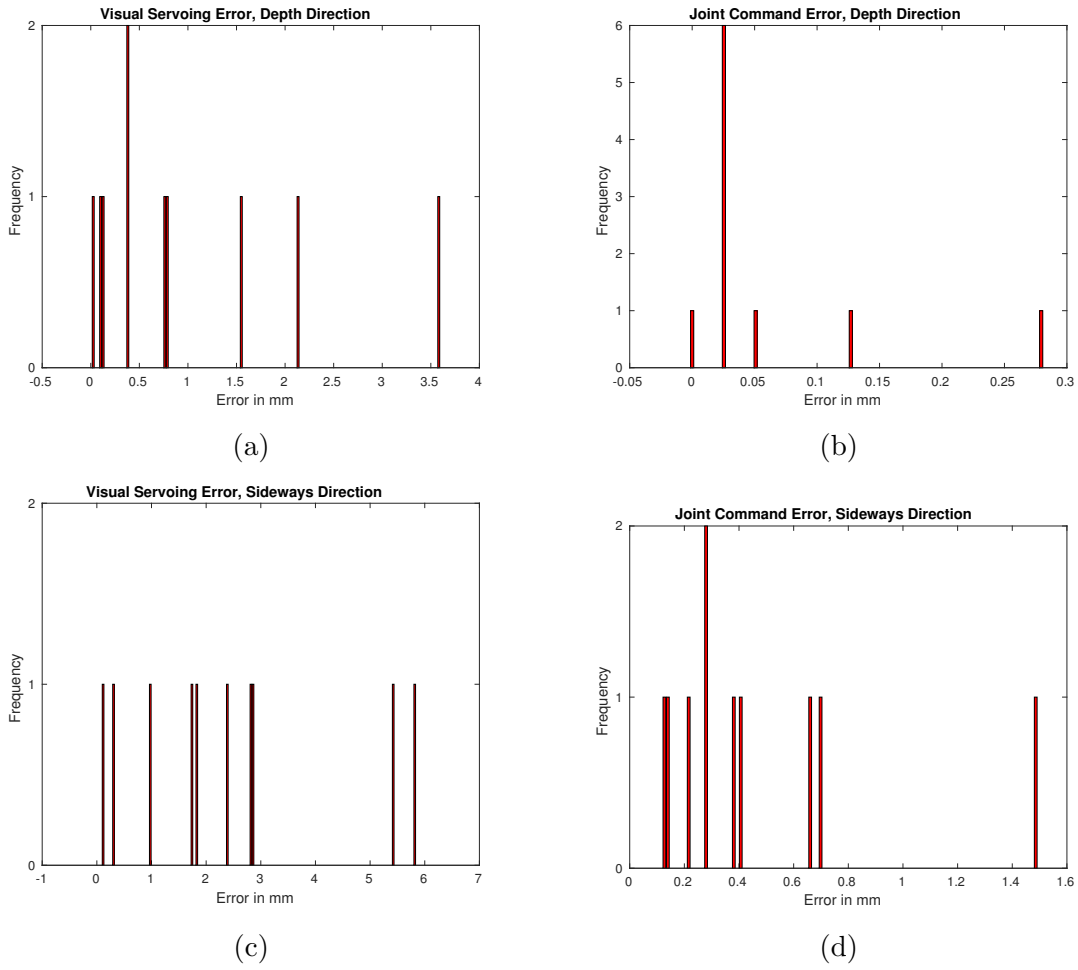


Figure 6.24: A plot of the repeatability experiment error in the depth and sideways directions. The x axis shows the range of the errors in mm. The y axis displays the number of experiment iterations with the given error on the x axis. (a) Values for visual servoing in the depth direction. (b) Values for using joint commands in the depth direction. (c) Values for visual servoing in the sideways direction. (d) Values for using joint commands in the sideways direction.

	Depth Direction		Sideways Direction	
	Joint Command	Visual Servo	Joint Command	Visual Servo
Mean Error	0.0610 mm	0.983 mm	0.467 mm	2.43 mm
Standard Dev.	0.0840 mm	1.14 mm	0.408 mm	1.93 mm
Variance	0.00705 mm	1.29 mm	0.166 mm	3.73 mm

Table 6.1: The table lists the results of ten iteration of each repeatability experiment.

In this experiment the robot gripper holds a box, see Fig. 6.23. We place a total of seven LEDs on the box. Each camera can see five of those LEDs. We place the LEDs on different planes of the box to avoid ambiguous task specification. The cameras are placed about 90 degrees and 103 cm apart, distanced 57 cm and 65 cm from the dial meter, respectively. The base of the robot is 90 cm from the dial meter. The base of the robot is 90 cm from the dial meter. One pixel in the camera views correspond to approximately 0.94 mm robot movement. The measurement is taken between the initial position and goal (closer to the goal).

We visual servo the box to the same goal from ten different initial position and record the accuracy. The visual servoing runs in a loop and we do not initialize new trackers each time we get back to the start position. We also do not choose the goal points by clicking. Instead we move the robot to the goal point and read the values of the trackers. This way the experiment does not incur errors due to user input.

The average positioning errors for the experiments can be seen in Table C.1. The plot in Fig. C.8 shows a bar plot of the error for each run of the experiment for the visual servoing and joint command cases. We can see that the mean error for visual servoing is 0.983 mm and 2.43 mm, respectively, which is better than the 4.6 mm average error from out marker in cap experiment. The visual servoing, however, has a higher mean error than the robot joint command with 0.0606 mm and 0.467 mm error respectively. We also completed one experiment in the depth direction with the same initial and goal positions, where we let the user set the goal points by clicking. This experiment resulted in 1.86 mm error (1.14 mm standard deviation and 0.00130 mm variance), which is higher than for the pure repeatability.

The fact that the experiments in the sideways direction have a higher error than in the depth direction, is consistent with our observation from the accuracy experiment, where one direction had a higher error than the other. We think that the difference between the sideways and forward motion is due to the calibration of the robot. The

joints do not compensate for the weight of the robot and the box as well in the sideways as in the forward direction. This is also evident from the robots behaviour when stopped at the goal. For the forward movements the robot sits still at the goal, whereas for sideways motion the robot makes oscillations from side to side. Recall that in the accuracy experiment we would be unable to insert the marker into the cap directly without position adjustments in 7 out of 10 iterations. If it were not for the higher error in the sideways direction, we would likely be able to complete the insertion directly in most cases.

The measurements we achieve in this experiment are not as good as those reported in [41] and [29], where they accomplished results at least as good as robot joint command positioning. In [41] the authors get 0.13 mm and 0.059 mm error with two different robots and the authors of [29] get 0.1020 mm error. In the latter case they only complete a 3 DOF positioning task, which is not exactly comparable to 6 DOF positioning.

The higher visual servoing errors can come from different sources. Our first explanation is that camera placement will play a role. The closer a camera is placed to a task, the higher resolution is given to the visual servoing. In the 3 DOF robot finger visual servoing in [29], the cameras are placed about 30 cm from the task, which is about twice as close as the cameras in our experiment. [41] does not explicitly state the distance between the cameras and the task, but says that one pixel approximately corresponds to 0.25 mm of robot movement near the goal. In our case one pixel corresponds to about 0.94 mm movement not exactly at, but close to the goal. These numbers cannot be exactly compared due to the different measurement location, but suggest that we may be able to improve the results by moving the cameras closer to the task.

Although the image errors converge, they do not go exactly to zero. First and foremost tracking will not be perfect. Especially when the experiment is run for ten iterations, the trackers drift. We found that the repeatability errors did not get consistently worse for each iteration, which we might expect if the tracking drifted more for each iteration. In fact, when the robot moved from the goal and back to a new initial position, the tracking would sometimes get better. Even so, the visual error at the goal is too high and we think that better tracking will help improve it.

More than just tracking, there is also another cause that can affect the final visual error to not completely converge. The contact with the dial meter is a possible source of error. We chose a box that has a smooth surface to allow the tip of the meter to

slide on the box. Some resistance will still be there and can affect the robot's ability to make small adjustments in position close to the goal. When we use the robot joint commands, the end-effector moves to the goal fast, which gives it force to push on the meter. When close to the goal, image errors are small, which leads the visual servoing to command smaller steps that makes it harder for the robot to push against the meter with sufficient force.

Our repeatability result can be affected by more than just the visual error. The LEDs in [41] are placed further a part and are not restricted to the same planes, whereas our LEDs have less optimal placement. We think that because the LEDs are close together, several are in the same plane and four are only seen by one camera, some ambiguity may occur. Small alterations in the robot pose in some directions may lead to smaller changes in the visual error than if the LEDs had better placement. Hence, we may reach some poses where the robot is slightly perturbed from the goal pose, but the visual error is still small.

In this experiment we presented results from a repeatability experiment for visual servoing. They show how accurately we were able to complete visual servoing without the involvement of the user clicking on goal points or combining tasks of different types. This gives some context to the errors we found in the accuracy experiment. Although we showed that the positioning error improved compared to the previous accuracy experiment, we think that the repeatability experiment can be altered to get improved performance closer to that of using robot joint commands. For more detailed explanation and more figures, see Section C.2.

6.3 Discussion

This section includes some of the lessons we have learned from working with our interface. They were gained from the process of developing the interface, trying to use it to specify tasks and conducting experiments. All these results will provide useful knowledge for how to best use our system, or visual task specification in general, to complete tasks with the robot.

6.3.1 Error Normalization

As mentioned in Chapter 5, the task errors have different magnitudes, which can lead to challenges when they are combined to create high-level tasks. Because visual

servoing minimizes all the errors at the same time, effectively averaging them, one error can dominate another. While point-to-point task errors have two components and are linear, the other task errors have one component and are at least quadratic. We tried different approaches to tackle this problem and will present our findings here.

We first tried to normalize the task errors. The 1D and 2D errors had to be normalized separately. We translated the centroid of the errors to the origin and then scaled such that the average distance from the origin was $sqrt(2)$. This approach unfortunately did nor work as it changed the behaviour of the errors and the tasks did not converge.

Since the errors are linear and quadratic or higher we wanted to try to square the point-to-point error or take the root of the other errors. In the case of squaring the point-to-point error, the point-to-point task at first moves as expected towards its goal. However, the robot stops short of the goal point and does not converge. Moreover, adjusting the point-to-point error would be impractical since the other errors are of different orders. For example, the point-to-line task is cubic, while the parallel lines task is quartic. Next we tried to take the root of the other error functions. Recall that we took the cubic root of the point-to-line task in the line following experiment, see Section 6.2.2.4. This approach fails when the error is small because it becomes unstable.

If we cannot reliably change the errors to all be of the same order, we can still try and work with the scale. Therefore we let the user adjust the weight of each error in the user interface. This does not affect the task's ability to converge. Instead it adjust how dominant one task will be compared to another. It gives the user flexibility when combining tasks for manipulation, but it has the obvious downside of not being general. The error weights would have to be adapted for each manipulation. Moreover, this scale naturally diminishes when the error gets small, as we saw with the point-to-line task in the line following experiment, Section 6.2.2.4.

Another idea we had was to normalize the image coordinates. This would not improve on the variation in order between linear and higher order errors, but it could make the magnitude difference easier to handle. In particular, it gets easier to scale errors because we can get away with lower factors. When working with 640x480 size images, some errors could get to the order of 100 000, whereas the point-to-point error never exceeds the size of the image. We moved the origin to the centre of the image and made the height of the image equal 1 and the width of the image $width/height$.

The normalization did not affect the behaviours of the errors and could therefore be added to our system.

To overcome the practical challenges of combining different tasks, we decided to combine normalizing the image with allowing the user to add weights to the different errors. The ability to weigh errors is useful for some tasks and normalizing the image made weighing the errors easier for the user. Because the of the change in magnitudes due to normalization, less weight has to be added to the different errors.

6.3.2 Cameras

As mentioned in the results of our experiments, camera placement matters. If camera views are too similar, it can affect the ability of a set of tasks to appropriately position the end-effector or object. Not all tasks will be equally sensitive to camera placement. The marker in cap task from Section 6.2.2.1 is an example of a task where camera placement does matter. If camera views are too similar, the positioning of the marker above the cap will be inaccurate in the direction looking away from the cameras. Moreover, we needed two separate views to be able to observe both directions of perturbation in orientation. In Section 6.2.2.3 we failed to grasp the ellipse because the camera views lead to poor placement of the gripper and in Section 6.2.2.5 we needed to add forward movement to accurately place the gripper relative to the screw. Hence, the camera views that we choose can affect the accuracy of a task in corresponding directions.

The proximity of the cameras to the task will naturally also affect the performance of visual servoing. When the cameras are close by, the task has a higher resolution and we are able to position more accurately. Therefore we try and place the cameras as close as possibly to the task without any part of the task leaving the field of view during execution.

6.3.3 Task Specification and Execution

The most challenging part of using task specification and visual servoing to complete tasks, is to learn how different tasks behave and slowly understand what the best strategies are for accomplishing actions. During our work, we have learned that just specifying the necessary constraints to control the robot is not enough for successful task completion. Issues such as choosing the best DOF of the robot to use, how to combine different task errors and how to divide up the task must be tackled. There

is no one way to do task specification so that it will work in every case. Instead the human-in-the-loop must asses the situation and decide what approach works best. In this section we elaborate on the experience we have gained in how to best use the interface and task specification to have the robot complete certain tasks.

As mentioned above the user must decide whether to specify the visual constraints in sequence, in parallel or a combination of both. The choice depends on the task at hand and we suggest the following possible strategies:

- separate the task into a sequence
- combine the tasks in parallel
- use intermediate points
- weigh tasks
- use primary and secondary tasks (future work)

The two first items represent the two extremes. Recall the marker in cap experiment from Section 6.2.2.1. We used parallel lines tasks as to adjust the orientation of the marker and a point-to-point task to move to the cap. In the first instance we chose to separate the task by completing the orientation before the translation. In the second instance we completed both in parallel. Completing the tasks in parallel requires less work from the user. However, as mentioned in Section B.1, sequencing the tasks and using a lower number of DOF, not only results in a better conditioned problem, but also makes the fine adjustments easier for the user.

Completing tasks in parallel is most convenient when they all are of the same type. Visual servoing will average the tasks and so it is easiest to predict the resulting behaviour when the task errors are similar. In fact we can take advantage of this behaviour in many cases. Recall the experiments when we grasped a cylinder in Section 6.2.1.3 and inserted a cube in Section 6.2.2.2. We were able to centre the gripper on the cylinder, because the visual servoing averaged the point-to-ellipse tasks from each robot finger. In the cube insertion we define several point-to-point tasks to do one translation. The averaging effect made it easier to produce precise motion since the errors we made while placing the goal points would even out.

Sometimes we are neither able to separate the tasks completely nor combine them without any modification. This was the case with the cutting task in Section 6.2.2.6.

We combined a line-to-line task and a point-to-point task to get a knife to cut along a line. Running the line-to-line task by itself at first to adjust orientation does not work because we cannot successfully use a reduced number of DOF of the robot that will map well to this task. Furthermore, because the point-to-point translation was large and dominated the task, combining the two constraints together was also not an option. Therefore we tested the two strategies of weighing the tasks and setting an intermediate point. The weighing succeed and helped mitigate the dominance of the point-to-point task. Setting an intermediate point worked in the case when we used four joints of the robot.

Setting an intermediate point has three main advantages. First of all it can help us avoid obstacles by augmenting the visual servoing path. Second it can give us some more control over tasks that require high precision. For the marker in cap task in Section 6.2.2.1 as well as the cube insertion task in Section 6.2.2.2, we were not able to track the exact edges of the cube and the tip of the marker. If we place the goal point of the point-to-point task directly at the physical goal, in this case on the cap or the container, the marker or cube will collide before the position is sufficiently adjusted above the goal. Setting an intermediate point above the final goal solves this problem. The last advantage of intermediate points is that they can lower the error magnitude of the point-to-point task and so reduce its dominance.

Another possible strategy to task specification is one that we did not have time to implement, namely, primary and secondary tasks. We discussed its potential usefulness in Section B.4, where the robot follows a line. Since visual servoing will minimize all task errors at the same time effectively averaging them, it is sometimes difficult to control and predict the behaviour of tasks. In the line following example the point-to-line task lost significance once it had converged to the line. When we added a point-to-point task to drive the robot forward along the line, the point-to-line task had no effect. By assigning the point-to-line as a primary task and the point-to-point task as secondary task we assure that the point-to-line task has priority. Any movement that the point-to-point induces can only happen as long as it is not in conflict with the primary point-to-line task.

The primary and secondary task method has potential benefit for a second scenario. In several tasks such as grasping a box (Section 6.2.1.1), placing a box on a table (Section 6.2.1.2) and putting a marker in a cap (Section 6.2.2.1) we specified the tasks in sequence by first adjusting orientation before translating. Then we added more tasks to complete the motion. In this second step we would keep the parallel

lines task that adjusted orientation and reduce some number of DOF. When inserting the marker in cap we could complete the translation with 3 DOF and the orientation of the marker remained the same. In the case of placing the box on the table we needed to use 5 DOF control, otherwise the orientation that we had aligned would get worse. When completing orientation and translation in sequence the size of the error of the tasks related to orientation will have become small before translation starts. Even though we keep these tasks around, they will have very little impact. Therefore we end up solving this problem with ad hoc solutions like reducing the number of DOF or inserting a task with higher error (as with the line-to-line task when placing the box). Instead, we think it would be a more general solution to either specify these tasks as primary and secondary from the start or do so once the first orientation step has been completed to make sure that the orientation constraints remain significant to the overall task.

Apart from choosing how to combine tasks, other methods can also aid in successful task completion. We found the technique of setting free space points to be particularly useful. Free space points are convenient both for making coarse movements towards a goal, as when we moved the marker towards the cap or the cube towards the square opening, as well as for making fine adjustments in positioning, which we did to get accurate alignment of the marker and cube. Though free space points will not be exact, we have found them to work well. Because the points are defined in two images and because the visual servoing will converge somewhere between the points, they are accurate enough for coarse positioning. They also work for fine position adjustment because in those cases we do not choose points necessarily to correspond to a 3D location, but more to pull an object in a certain direction. The pull of the different points will average out and move the object in the desired direction. Being able to set points above a goal helps us create intermediate points as mentioned in the previous paragraph. Later on during our accuracy experiment we also implemented the ability to extend lines to create free space points. This technique requires the user to think about how to construct the correct points, but the result will be more accurate.

A second useful technique is keeping the composition of geometric constraints constant throughout several steps of a task. This technique will not work for all tasks, but is useful for many. In particular it is useful for fine manipulation. As explained in the the marker in cap and cube insertion experiments, having a constant set of tasks after the orientation has been aligned allows us to make adjustments

without having to acquire a new Jacobian. We used this to adjust the positioning of the marker and the cube and to complete the insertions. This technique seems most natural to use with point-to-point constraints because these are intuitive for the user to adjust. Making point adjustments can be tedious in some cases, as we saw with the cube insertion, because several actions are needed from the user. In our experiments we have showed that such adjustments work and that they can be useful. In future work we can explore more efficient interaction for the user by for instance dragging goal points to adjust them or to create movements in a given direction.

6.3.4 Task Decidability

Recall our discussion of task decidability in Section 4.1. Even though projectively invariant tasks are not decidable for uncalibrated cameras, Hager showed through his work in [49] that visual task specification for UVS still works well in practice. [45] point to the fact that projectively invariant tasks only lose verifiability in some special cases. We noted that the parallel lines task is not even projectively invariant and could therefore not predict whether this task could be successfully used in our system. Through experimentation we have shown that the parallel lines task has indeed been a very useful addition to our set of geometric constraints. We used it in many tasks to adjust gripper or object orientation. It may be, similarly to the fact that projectively invariant tasks only fail in special configurations, that the parallel lines task is also verifiable except for in a few cases.

We also made a second observation. When working with manipulation tasks the cameras are placed fairly close to the task and parallel lines in the workspace look parallel in the image. We can compare this to images that look at a road or a set of rail lines going into the horizon. Is these views it is much more clear that parallel lines do not remain parallel when subjected to projective transformations. Both our observations are not proven, but it provides motivation to further investigate the parallel lines task with more rigour in future work.

6.3.5 User Interaction

More than just the ability to accomplish tasks, we want to gain knowledge about the usability of the task specification approach and the way it is currently implemented in our interface. To that end we stated the number of clicks necessary to complete each of the tasks we have listed in our experiments. We also counted how many

geometric constrains were created. In the ten iterations of the accuracy experiment we counted the number of clicks made by the user each time. The numbers are for each experiment can be found in the Appendix.

The number of clicks needed vary greatly between experiments, depending on how complicated the tasks are to construct. The lowest number of clicks was for the simplest versions of the line following and cutting tasks with 29 clicks. The highest number was 185 for the cube insertion. Similarly the number of corresponding geometric constraints were two for the line following and 28 for the cube insertion. We noticed that the number of clicks increased increased quite a lot when we needed to use scaling or intermediate points or when we needed to make fine adjustments to the end-effector positioning.

Our results show that the user interaction can become tedious, especially when we need to adjust positioning at the end if the task or if we need to add scale or intermediate points when combining asks. Another challenge for the users of the interface is to decide exactly what geometric constraints are needed to unambiguously specify a task and decide the corresponding robot DOF to use. This suggests that future work should look into ways of making the interaction less taxing on the user. To make the choice of tasks more intuitive and reduce the number of clicks we can pre-define some often used combinations of geometric constraints. This would give the user some easy to understand higher-level tasks to choose from. Moreover, we can let the user more easily adjust positioning by defining a direction for movement by dragging goal points in the image to move them.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

With our work we want to contribute to making robotics useful in uncontrolled environments. We focus on human-in-the-loop control by developing an interface that lets a user specify actions for a robot to complete. A robot arm is controlled using uncalibrated visual servoing and the user interaction is based on the technique of visual task specification. The user describes high-level actions by visually specifying geometric constraints in the interface.

The goal of our work was not only to build an interface for versatile human robot interaction, but also to explore the feasibility and performance of the visual task specification approach. In particular, we wanted to study the effects of combining a set of visual tasks to form higher-level actions.

In this thesis we have presented our visual interface and given details on its development. We have conducted experiments that show that high-level actions can indeed be specified by combining visual constraints. The experiments show that the system is capable of executing a range of tasks spanning both coarse and fine manipulation. Being able to complete fine manipulation tasks gives our system an advantage over many others that use 3D sensors like the Kinect. Moreover, our system is general in that it can specify any tasks as long as we can describe it with geometric constraints. Additionally, we provide results from accuracy and repeatability experiments that demonstrate the performance of our system.

The experiments have given us the opportunity to investigate the behaviour of visual tasks and so gain knowledge on how to best specify tasks for successful manipulation. The fact that different types of task errors have different error magnitudes

provided the greatest challenge for our approach. Through trial and error we were able to devise strategies for task specification that best mediate this problem, some of which include normalizing the image space, weighing task errors, combining tasks in sequence or parallel and using intermediate points. Since we have a human-in-the-loop system, the user can asses the tasks and tailor the approach on a task-by-task basis.

The visual task specification system also has some downsides. Fully specifying tasks requires many actions by the user. Moreover, choosing the correct geometric constraints and DOF of the robot is not always straight forward. Although the current state of the system is not a perfectly feasible system for any user, we imagine it to be a step on the way for good human robot interaction.

7.2 Future Work

As we mentioned in the conclusions, our system can be used to specify a wide variety of tasks, but the user interaction can be tedious. In this section we discuss two interesting paths for improving the user interaction and end with some smaller and more specific improvements.

Although visual constraints give a lot of freedom in what tasks to specify, the required user interaction is sometimes tedious. Not only does the interaction require a lot of clicking, but it is also challenging for the user in many cases to choose the best geometric constraints to specify a complete task. It would be desirable to look for ways to simplify. When we spend more time working with the system, we will start to see combinations of tasks that are used repeatedly. With this knowledge we can add pre-defined task combinations. Then the user would have a set of easy to understand higher-level tasks ready to use. Imagine, for instance, if the user wants to insert one box into another, similarly to the example of inserting a tray into a toolbox from Chapter 5. Inserting a box shape into another will require the same set of specifications most times. If the user marks the corners on the two box shapes, we could provide a function that automatically sets the necessary point and line specifications. Furthermore, we can imagine including image processing that detects shapes in the image. In that scenario the system could detect square shapes and the user would only have to select them. Our suggestion would be similar to a strategy used in CAD software. There they have some especially useful and common actions already pre-defined.

Another possible path for future work is to combine visual task specification with teleoperation. Instead of defining an entire action using visual tasks, we can constrain part of the task visually and let the user have control over other DOF via teleoperation. This way we benefit from the precision of visual specification as well as the observation and decision making of the human. Say, for instance, that we want the robot to follow a line. It is difficult for a human user to accurately follow a line with unassisted teleoperation. Instead we can create a virtual fixture [19] by adding a visual constraint for the line. The robot motion will be constrained along the line using visual servoing, but the user controls the movement along the line, i.e. the forward motion. One practical example of such a task is the satellite servicing experiment [63], where the user teleoperates a robot to cut a square piece of satellite insulation. Adding visual servoing to a teleoperation system can have many benefits for teleoperation such as avoiding obstacles, completing tasks faster and aid fine manipulation. Essentially we would be using visual constraints to make teleoperation less taxing on a user.

Other than high level ideas for improving the task specification, there are also several smaller adjustments that should be made for the interface itself. First of all we can implement more tasks such as for example point-to-plane or curve following [64] tasks. Improvements in tracking and adding of more trackers will always benefit a visual servoing system and make it more robust. Another addition that would greatly benefit tracking, is support points. Points that are difficult to track, but rigidly attached to areas that we can track, can be related to the latter trough a homography. Say we want to grasp the edge of a dinner plate. The rim of the plate is white and not possible to track. However, there is a textured pattern in the middle of the plate. We can use a patch tracker to track the pattern. If we know the homography transformation between the patch's previous and current position, then we apply this same homography to the support point to update its position. Another enhancement to tracking comes from projective invariants. We already use projective invariants to find the centroid of patch shapes. Calculating such invariants will allow us to define more points that can be useful for task specification.

There is also room for improvement in the user interaction with points and shapes. As seen in several of the experiments, we adjust point-to-point tasks to make positioning for fine manipulation more precise. We have shown that adjusting goal points in this manner works well. However, the interaction required is tedious. Instead the user can make adjustments by dragging the mouse marker from the tracked point on

the robot (or tool) in the desired direction or by dragging the goal point instead of having to delete and create new points. This would generate a path of intermediate goal points for servoing, though these need not be displayed to the user. The interface could also benefit from making other adjustments simple. If the user has created a shape, as for instance an ellipse, then he/she should be able to dynamically change its size or placement, just as one would do in a drawing program. Similarly, the user should be able to move points by dragging them. Among other things, this would be useful for resetting trackers that have drifted.

As discussed in Section 6.3.4 we showed that the parallel lines task performed well despite that fact that the task is not projectively invariant. We only made observations as to why this may be the case. Our result warrants more rigorous exploration of why the parallel lines task works and in which cases it may fail. Another task that will also benefit from more testing is the point-to-ellipse task. To our knowledge this task has not seen as much application as the more well known point-to-point or point-to-line tasks. Although we were able to successfully use the point-to-ellipse task to accomplish higher-level actions, we think it will be advantageous to further explore the task in order to learn about its behaviour in a variety of situations.

For visual servoing the Jacobian maps the results of our image space specifications to movements in joint space. Before we start a task we create the Jacobian using initial motions and throughout the execution we update the Jacobian using the Broyden method, as detailed in Section 3.4.1. In our experiments we were able to set new goal points for some tasks without re-acquiring the Jacobian, because we kept the composition of tasks constant. However, it would be beneficial to be able to change the composition of tasks during servoing without having to re-initialize the Jacobian. If we are close to an object we cannot do the required exploratory motions. We want to implement the ability to update the Jacobian when we either add or remove tasks.

From the experiments we have seen how combining tasks of different magnitudes have caused challenges. We have also suggested strategies for how to mitigate this problem in different scenarios. While completing the line following experiment, we found that our current techniques fell short. Moreover, we think that some of task that are solved in sequence can be solved in a more general and consistent way. Therefore we suggest the technique of using primary and secondary tasks for future work. In visual servoing one can differentiate between the importance of different tasks by assigning them to be primary or secondary. In our line following example we would set the point-to-line task as primary and the point-to-point task as secondary. For

this to work there must be free DOF left after the primary task has been determined. Visual servoing will use both task constraints in its minimization, but the secondary task will only converge as long as the motion it generates does not conflict with the primary task. This technique will allow us to give priority to tasks regardless of their error magnitude.

Towards the end of this thesis we presented an accuracy and a repeatability experiment for the task specification interface and the visual servoing system. More work can be done to bet better results from the repeatability experiments by improving LED placement, camera placement and tracking. Furthermore, it would be of interest to expand the experiment to include more iterations, visual servoing directions and robot poses. The accuracy experiments can be expanded to explore the effect of different combinations of tasks and user interaction.

Since our visual task specification system has been developed from scratch, a lot of time has gone into basic implementation. For future work we should take time to evaluate the user interaction and use the results to improve the interactive aspects of the interface. Furthermore, the interface should be ported to and tested on other devices, such as a tablet or even a phone. We can also try using the interface with visual servoing for different types of robots, such as a UAV or a mobile robot.

The work in this thesis is the first iteration in our lab of using a visual task specification interface for visual servoing. We have explored its use and gained knowledge on how to best employ the interface to complete both coarse and fine manipulation tasks. As the work continues we can continue to explore task visual task specification, make improvements on the system and integrate it with other techniques.

Bibliography

[1] iRobot Corporation, "irobot for the home," 2015. [Online]. Available: http://www.irobot.com/For-the-Home.aspx

[2] [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/Robots%282f%29Husky.html

[3] Microsoft, "Kinect for windows," 2015. [Online]. Available: https://www.microsoft.com/en-us/kinectforwindows/default.aspx

[4] C. Kemp, A. Edsinger, and E. Torres-Jara, "Challenges for robot manipulation in human environments [grand challenges of robotics]," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 20–29, March 2007.

[5] T. Hulin, K. Hertkorn, P. Kremer, S. Schatzle, J. Artigas, M. Sagardia, F. Zacharias, and C. Preusche, "The dlr bimanual haptic device with optimized workspace," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 3441–3442.

[6] C. Borst, C. Ott, T. Wimbock, B. Brunner, F. Zacharias, B. Bauml, U. Hillenbrand, S. Haddadin, A. Albu-Schaffer, and G. Hirzinger, "A humanoid upper body system for two-handed manipulation," in *Robotics and Automation, 2007 IEEE International Conference on*, April 2007, pp. 2766–2767.

[7] Sensable, "Phantom omni haptic device," 2015. [Online]. Available: http://www.dentsable.com/haptic-phantom-omni.htm

[8] A. Herdocia, A. Shademan, and M. Jagersand, "Unimodal asymmetric interface for teleoperation of mobile manipulators: A user study," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, Oct 2012, pp. 5214–5219.

[9] C. Perez Quintero, R. Fomena, A. Shademan, O. Ramirez, and M. Jagersand, "Interactive teleoperation interface for semi-autonomous control of robot arms," in *Computer and Robot Vision (CRV), 2014 Canadian Conference on*, May 2014, pp. 357–363.

[10] U. Hagn, R. Konietschke, A. Tobergte, M. Nickl, S. Jrg, B. Kbler, G. Passig, M. Grger, F. Frhlich, U. Seibold, L. Le-Tien, A. Albu-Schffer, A. Nothhelfer, F. Hacker, M. Grebenstein, and G. Hirzinger, "Dlr mirosurge: a versatile system for research in endoscopic telesurgery," *International Journal of Computer Assisted Radiology and Surgery*, vol. 5, no. 2, pp. 183–193, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11548-009-0372-4

[11] Intuitive Surgical, Inc., "The da vinci surgical system," 2015. [Online]. Available: http://www.intuitivesurgical.com/products/davinci_surgical_system

[12] C. Perez Quintero, O. Ramirez, and M. Jagersand, "Vibi: Assistive vision-based interface for robot manipulation," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 000–000.

[13] Kinova Robotics, "Jaco^2 rehabilitation," 2015. [Online]. Available: http://kinovarobotics.com/products/jaco-rehabilitation

[14] K. M. Tsui, D.-J. Kim, A. Behal, D. Kontak, and H. A. Yanco, "I want that: Human-in-the-loop control of a wheelchair-mounted robotic arm," *Appl. Bionics Biomechanics*, vol. 8, no. 1, pp. 127–147, Jan. 2011. [Online]. Available: http://dx.doi.org/10.3233/ABB-2011-0004

[15] C. Perez Quintero, R. Fomena, A. Shademan, N. Wolleb, T. Dick, and M. Jagersand, "Sepo: Selecting by pointing as an intuitive human-robot command interface," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 1166–1171.

[16] C. A. Perez Quintero, R. Tatsambon Fomena, M. Gridseth, and M. Jagersand, "Visual pointing gestures for bi-directional human robot interaction in a pick-and-place task," in *Robot and Human Interactive Communication (RO-MAN), 2015 IEEE International Symposium on*, August 2015, pp. 000–000.

[17] T. Chen, M. Ciocarlie, S. Cousins, P. M. Grice, K. Hawkins, K. Hsiao, C. Kemp, C.-H. King, D. Lazewatsky, A. E. Leeper, H. Nguyen, A. Paepcke,

C. Pantofaru, W. Smart, and L. Takayama, "Robots for humanity: A case study in assistive mobile manipulation," *IEEE Robotics & Automation Magazine, Special issue on Assistive Robotics*, vol. 20, 2013. [Online]. Available: http://lifesciences.ieee.org/images/pdf/06476704.pdf

[18] A. Leeper, K. Hsiao, M. Ciocarlie, L. Takayama, and D. Gossow, "Strategies for human-in-the-loop robotic grasping," in *Human-Robot Interaction (HRI), 2012 7th ACM/IEEE International Conference on*, March 2012, pp. 1–8.

[19] A. Bettini, P. Marayong, S. Lang, A. Okamura, and G. Hager, "Vision-assisted control for manipulation using virtual fixtures," *Robotics, IEEE Transactions on*, vol. 20, no. 6, pp. 953–966, Dec 2004.

[20] M. Quigley, M. Goodrich, and R. Beard, "Semi-autonomous human-uav interfaces for fixed-wing mini-uavs," in *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3, Sept 2004, pp. 2457–2462 vol.3.

[21] M. Raibert, K. Blankespoor, G. Nelson, and R. Playter, "Bigdog, the rough-terrain quaduped robot."

[22] Y. Hiroi, S. Matsunaka, and A. Ito, "Mobile robot system with semi-autonomous navigation using simple and robust person following behavior," *J. Man, Mach. Technol*, vol. 1, pp. 44–62, 2012.

[23] T. L. Chen and C. C. Kemp, "A direct physical interface for navigation and positioning of a robotic nursing assistant," *Advanced Robotics*, vol. 25, no. 5, pp. 605–627, 2011.

[24] C. P. Quintero, O. Ramirez, M. Gridseth, and M. Jägersand, "Small object manipulation in 3d perception robotic systems using visual servoing," in *todo*, 2014.

[25] H. Jiang, J. Wachs, and B. Duerstock, "Integrated vision-based robotic arm interface for operators with upper limb mobility impairments," in *Rehabilitation Robotics (ICORR), 2013 IEEE International Conference on*, June 2013, pp. 1–6.

[26] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *Robotics Automation Magazine, IEEE*, vol. 13, no. 4, pp. 82–90, Dec 2006.

[27] R. Fomena, C. Perez Quintero, M. Gridseth, and M. Jagersand, "Towards practical visual servoing in robotics," in *Computer and Robot Vision (CRV), 2013 International Conference on*, May 2013, pp. 303–310.

[28] M. Gridseth, C. Perez Quintero, R. Tatsambon Fomena, O. Ramirez, and M. Jägersand, "Bringing visual servoing into real world applications," in *Human Robot Collaboration Workshop at RSS*, 2013.

[29] M. Gridseth, K. Hertkorn, , and M. Jagersand, "On visual servoing to improve performance of robotic grasping," in *Computer and Robot Vision (CRV), 2015 Canadian Conference on*, June 2015, pp. 000–000.

[30] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 5, pp. 651–670, Oct 1996.

[31] F. Chaumette and S. Hutchinson, "Visual servo control. ii. advanced approaches [tutorial]," *Robotics Automation Magazine, IEEE*, vol. 14, no. 1, pp. 109–118, March 2007.

[32] D. Kragic and H. I. Christensen, "Survey on visual servoing for manipulation," Computational Vision and Active Perception Laboratory, Tech. Rep., 2002.

[33] R. Ozawa and F. Chaumette, "Dynamic visual servoing with image moments for a quadrotor using a virtual spring approach," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 5670–5676.

[34] P. Rudol, M. Wzorek, G. Conte, and P. Doherty, "Micro unmanned aerial vehicle visual servoing for cooperative indoor exploration," in *Aerospace Conference, 2008 IEEE*, March 2008, pp. 1–10.

[35] G. Mariottini, G. Oriolo, and D. Prattichizzo, "Image-based visual servoing for nonholonomic mobile robots using epipolar geometry," *Robotics, IEEE Transactions on*, vol. 23, no. 1, pp. 87–100, Feb 2007.

[36] [Online]. Available: http://www.cybertechuav.com.au/

[37] [Online]. Available: http://library.isr.ist.utl.pt/docs/roswiki/Robots%282f%29Husky.html

[38] E. Malis, F. Chaumette, and S. Boudet, "2 frac12;d visual servoing," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 2, pp. 238–250, Apr 1999.

[39] L. Deng, F. Janabi-Sharifi, and W. J. Wilson, "Hybrid motion control and planning strategies for visual servoing," *Industrial Electronics, IEEE Transactions on*, vol. 52, no. 4, pp. 1024–1040, Aug 2005.

[40] A. Farahmand, A. Shademan, and M. Jagersand, "Global visual-motor estimation for uncalibrated visual servoing," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, Oct 2007, pp. 1969–1974.

[41] M. Jagersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," in *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, vol. 4, Apr 1997, pp. 2874–2880 vol.4.

[42] J. Piepmeier, G. McMurray, and H. Lipkin, "Uncalibrated dynamic visual servoing," *Robotics and Automation, IEEE Transactions on*, vol. 20, no. 1, pp. 143–147, Feb 2004.

[43] H. Sutanto, R. Sharma, and V. Varma, "The role of exploratory movement in visual servoing without calibration," *Robotics and Autonomous Systems*, vol. 23, no. 3, pp. 153 – 169, 1998.

[44] K. Hosoda and M. 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, Sep 1994, pp. 186–193 vol.1.

[45] J. P. Hespanha, Z. Dodds, G. D. Hager, and A. S. Morse, "What tasks can be performed with an uncalibrated stereo vision system?" *Int. J. Comput. Vision*, vol. 35, no. 1, pp. 65–85, Nov. 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1008111128520

[46] Z. Dodds, G. Hager, A. Morse, and J. Hespanha, "Task specification and monitoring for uncalibrated hand/eye coordination," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 2, 1999, pp. 1607–1613 vol.2.

[47] G. Hager and Z. Dodds, "On specifying and performing visual tasks with qualitative object models," in *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, vol. 1, 2000, pp. 636–643 vol.1.

[48] J. Hespanha, Z. Dodds, G. Hager, and A. Morse, "Decidability of robot positioning tasks using stereo vision systems," in *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, vol. 4, Dec 1998, pp. 3736–3741 vol.4.

[49] G. Hager, "A modular system for robust positioning using feedback from stereo vision," *Robotics and Automation, IEEE Transactions on*, vol. 13, no. 4, pp. 582–595, Aug 1997.

[50] Z. Dodds, M. Jagersand, G. Hager, and K. Toyama, "A hierarchical vision architecture for robotic manipulation tasks," in *Computer Vision Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1999, vol. 1542, pp. 312–330. [Online]. Available: http://dx.doi.org/10.1007/3-540-49256-9_19

[51] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679. [Online]. Available: http://dl.acm.org/citation.cfm?id=1623264.1623280

[52] C. Tomasi and T. Kanade, "Detection and tracking of point features," International Journal of Computer Vision, Tech. Rep., 1991.

[53] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, Jun 1994, pp. 593–600.

[54] S. Benhimane and E. 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, Sept 2004, pp. 943–948 vol.1.

[55] S. Baker and I. 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, 2001, pp. I–1090–I–1097 vol.1.

[56] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[57] Open Source Robotics Foundation, "Robot operating system (ros) website," 2015. [Online]. Available: http://www.ros.org

[58] The Qt Company, "Qt website," 2015. [Online]. Available: http://www.qt.io

[59] Lagadic Research, "Visual servoing platform (visp) website," 2015. [Online]. Available: http://www.irisa.fr/lagadic/visp/visp.html

[60] T. Dick, C. P. Quintero, M. Jägersand, and A. Shademan, "Realtime registration-based tracking via approximate nearest neighbour search," in *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013*, 2013.

[61] G. D. Hager and K. Toyama, "X vision: A portable substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, pp. 23–37, 1996.

[62] E. Marchand, F. Spindler, and F. Chaumette, "Visp for visual servoing: a generic software platform with a wide class of robot control skills," *Robotics Automation Magazine, IEEE*, vol. 12, no. 4, pp. 40–52, Dec 2005.

[63] I. Kandaswamy, T. Xia, and P. Kazanzides, "Strategies and models for cutting satellite insulation in telerobotic servicing missions," in *Haptics Symposium (HAPTICS), 2014 IEEE*, Feb 2014, pp. 467–472.

[64] E. Marchand, "Visp: a software environment for eye-in-hand visual servoing," in *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, vol. 4, 1999, pp. 3224–3229 vol.4.

Appendix A

Coarse Manipulation Experiments

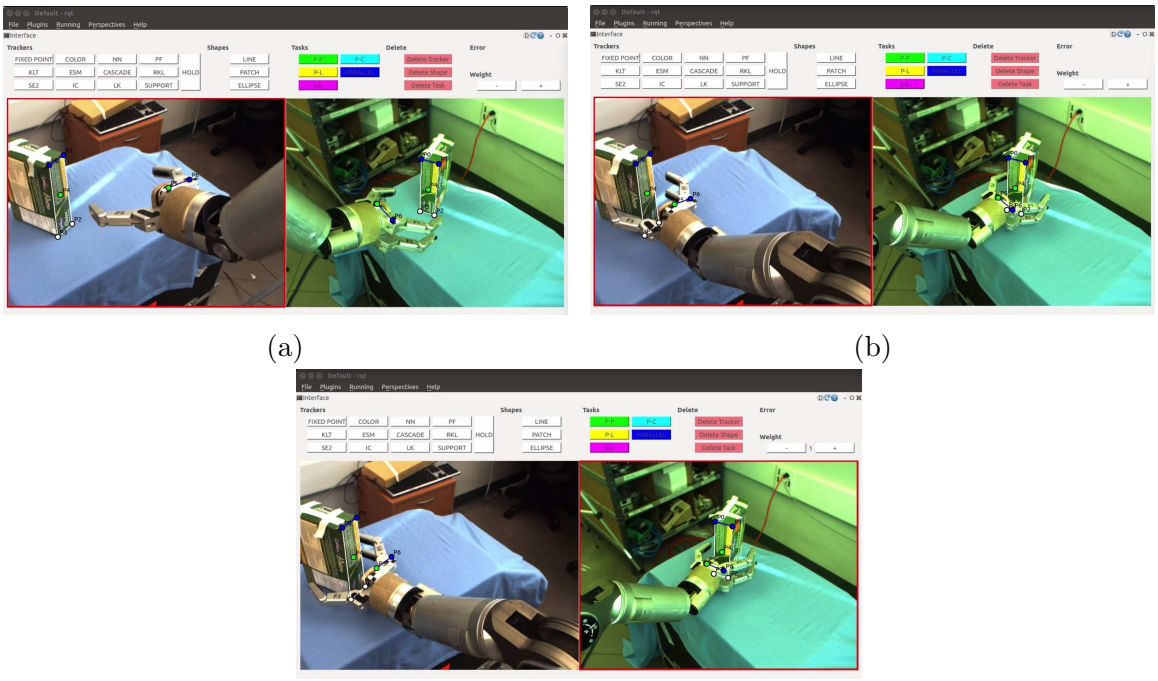
A.1 Grasp Box

The first task that we present is grasping a box that sits on a table. The box is 21.0 cm wide, 13.9 cm tall and 5.2 cm deep. The robot must move the gripper towards the box to grasp it and also take care to properly align its orientation, see Fig. A.1. We use four DOF of the robot, three to translate towards the box as well as yaw to orient the gripper. A parallel lines task between a line on the gripper and a line on the box adjusts the orientation. We add a point-to-point task to move to the box.

As can be seen in Fig. A.1, we execute the task, but it fails. The gripper moves forward, but the orientation has not changed enough. This happens because the point-to-point error has a larger magnitude than the parallel lines task error and so ends up dominating, see Fig. A.3.

We try two other approaches to solve this problem. First we scale the errors by adding extra weight to the parallel lines tasks. In Fig. A.1 we can see that this approach performs better and we can grasp the box. However, the grippers orientation could still be improved upon. In our final attempt we separate the task into two segments. We first run the parallel lines task only using one wrist joint. Next we add the point-to-point task to move the gripper to its goal, see Fig. A.2. The final method gives the best result. Plots of the task errors for all approaches can be found in Fig. A.3. Completing tasks separately takes a bit more work than combining them and in many cases scaling will be sufficient. Since we have a human-in-the-loop the user can decide in each situation what will work best.

The user needs to click 33 times in the interface and create four geometric constraints to specify the task. For the case where we weigh the errors the number of



(a) (b)

(c)

Figure A.1: (a) shows how we have specified a parallel lines task to adjust the gripper's orientation and a point-to-point task to adjust its position. (b) We have executed the task, but the gripper's orientation is not well aligned. (c) We have completed the task by adding extra weight to the parallel lines task. The gripper orientation is better, but still has room for improvement.

clicks increases to 73.

A.2 Place Box

The second task we include is placing a box on a table, see Fig. A.4. The box is 15.9 cm wide, 22.9 cm tall and 6.0 cm deep. The white rectangle on the table has the same size as the box and it will provide a way to observe the accuracy of our task. We first specify two parallel lines tasks to improve the orientation of the box before moving to the table, see Fig. A.4. Next we try three different approaches to place the box on the table. First we use joint 1, 2 and 4 of the robot and keep the parallel lines task, but add two point-to-point tasks, see Fig. A.5. We are able to place the box, but we do so with low accuracy. Since we only use 3 DOF of the robot, the previously aligned orientation is worsened. Furthermore, the point-to-point tasks dominate over

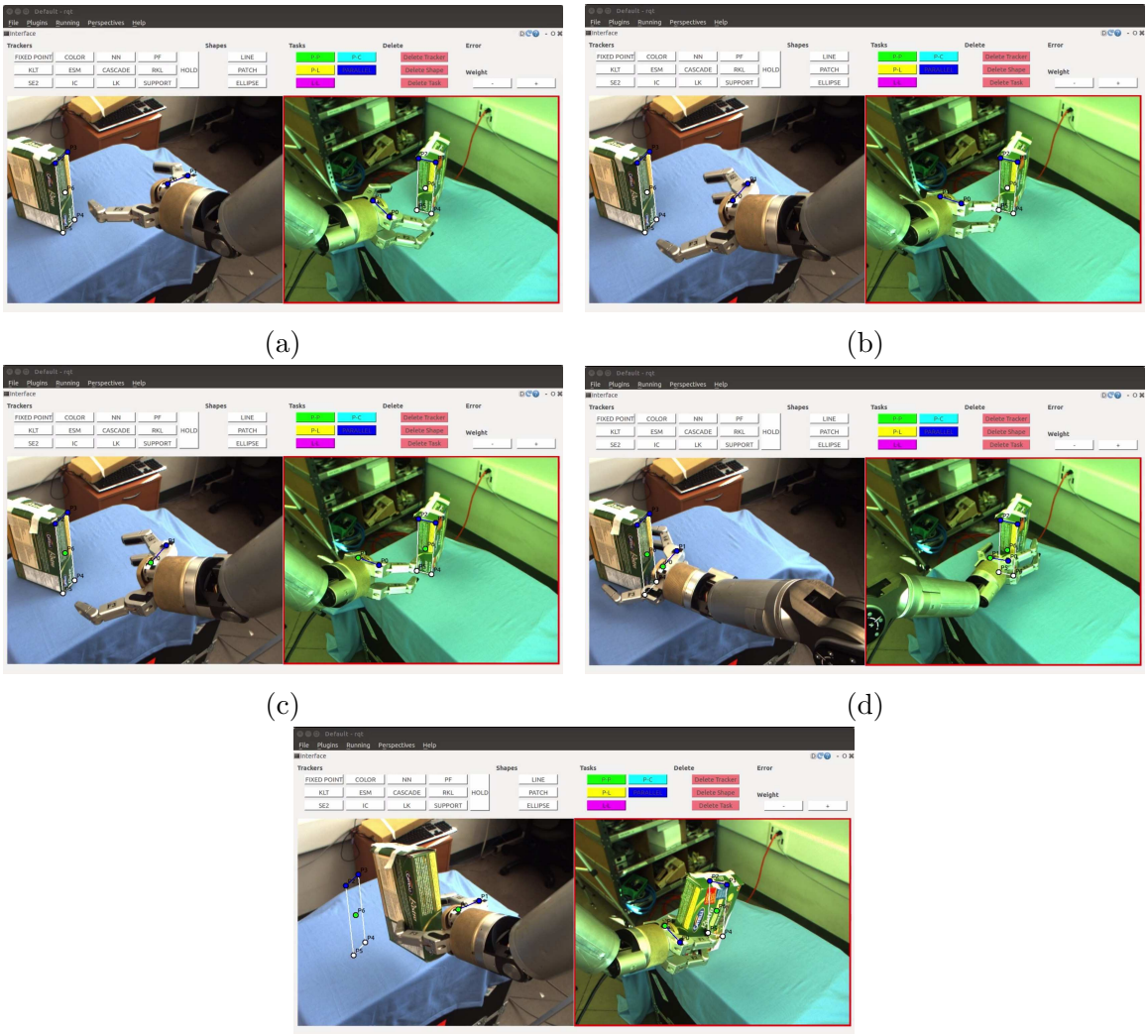


Figure A.2: We change the task to be specified sequentially. (a) We have specified a parallel lines task to adjust the orientation of the gripper. (b) The final alignment. (c) We add a point-to-point task to drive the gripper to the box. (d) The completed task. (e) The robot has successfully grasped the box.

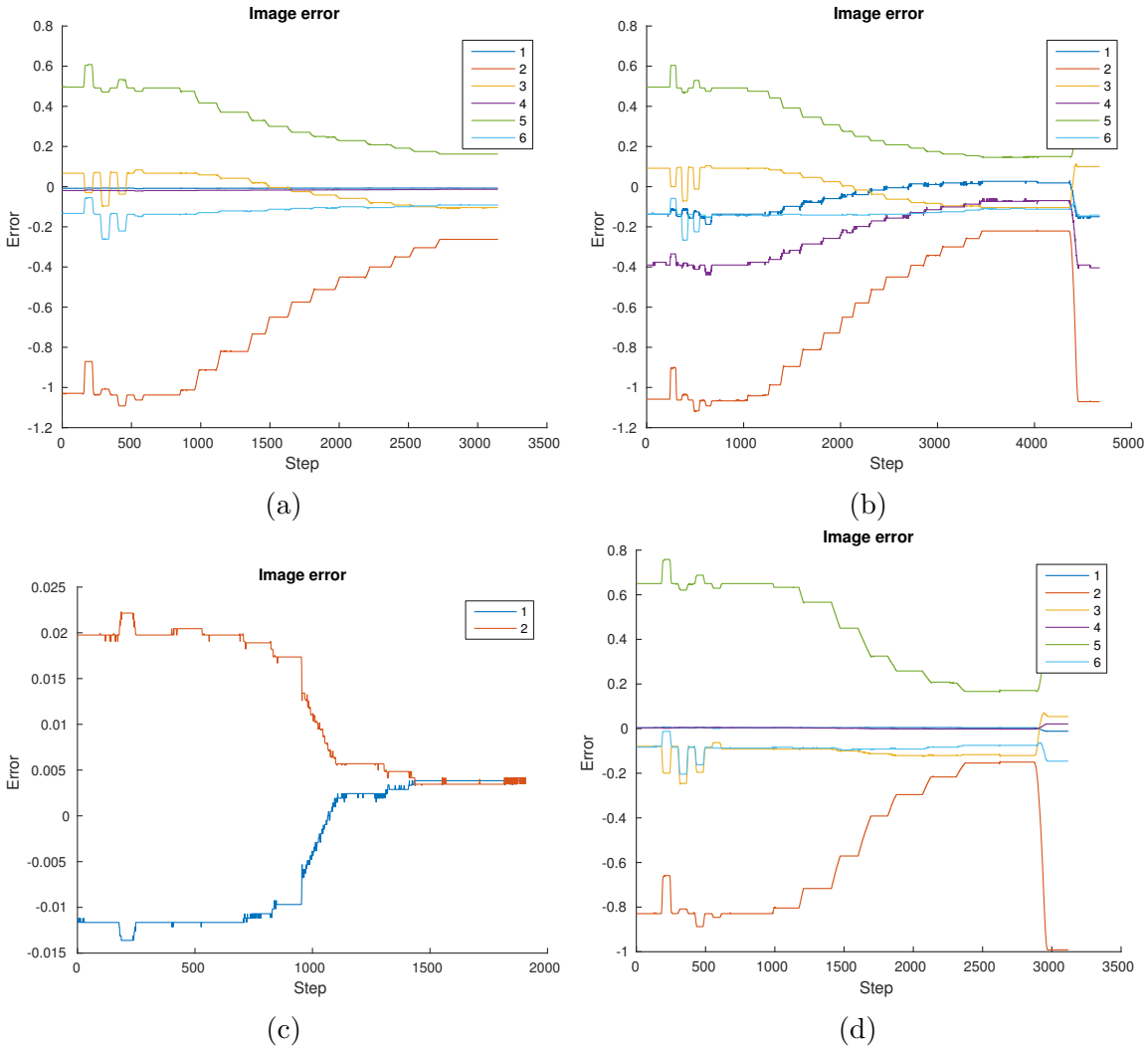
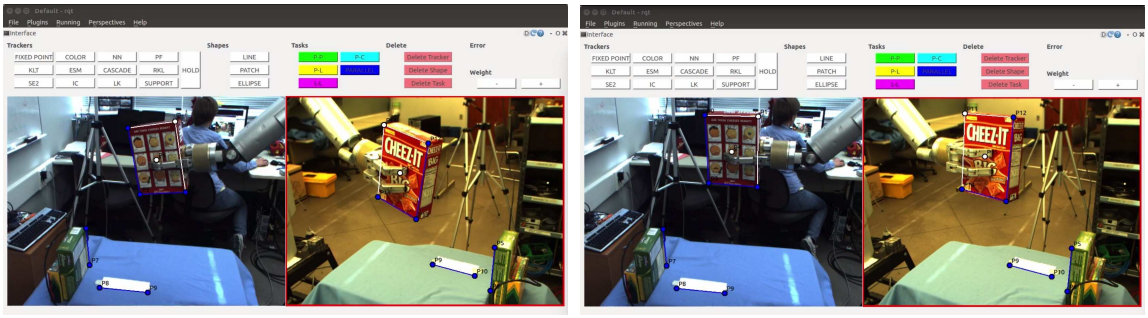


Figure A.3: The plots show the task errors for grasping the box. 1 and 4 are the parallel lines task errors in the first and second image, respectively. 2, 3, 5 and 6 are the point-to-point errors in the first and second images. The spike in the error at the end of some plots come from grasping the box once the task has been completed. (a) Task specification with no adjustments. The parallel lines task errors are very small compared to the others. (b) Weight added to parallel lines tasks. (c) Parallel lines segment of divided task. 1 and 2 stand for the parallel lines task error in the first and second image. (d) Second segment of the sequential task.



(a) (b)

Figure A.4: (a) The parallel lines task we have added to adjust orientation. The white piece of paper on the table is the same size as the box. (b) shows the improved orientation.

the parallel lines tasks. In Fig. A.6, we can see that the parallel lines task errors increase while the robot moves to the table.

Next we try a different task specification by combining one parallel lines task with a line-to-line task and a point-to-point task, see Fig. A.5. We also add one more robot joint to control the pitch. The results improve and we place the box with higher precision. However, the yaw orientation still lost its previous alignment. We can see in Fig. A.6 that with the new specification, the parallel lines task error remains small, which makes sense now that the pitch orientation is correct. Finally, we use the same task specification but add the yaw joint and so use joints 1, 2, 4, 5, and 6. The result can be seen in Fig. A.5 and the corresponding plot in Fig. A.6. Although the Jacobian conditioning number approximately triples when we add the extra degree of freedom, it is needed for the task to succeed. The last orientation is maintained and the box is placed accurately on top of the white rectangle, both directions had less than 3 mm error. Part of the yaw orientation that finishes the alignment, happens just as the box touches down on the table. Although the line-to-line error has a similar magnitude to the point-to-point error, it only has two elements in the error vector, while the point-to-point task has four. Therefore the line-to-line tasks adjusts some of the yaw orientation right at the end of the task once the point-to-point errors are small.

We are able to accurately place a box on a table by selecting a good set of tasks and choosing the right DOF of the robot to use. The user needs to click 53 times in the interface and create eight geometric constraints to specify the task.

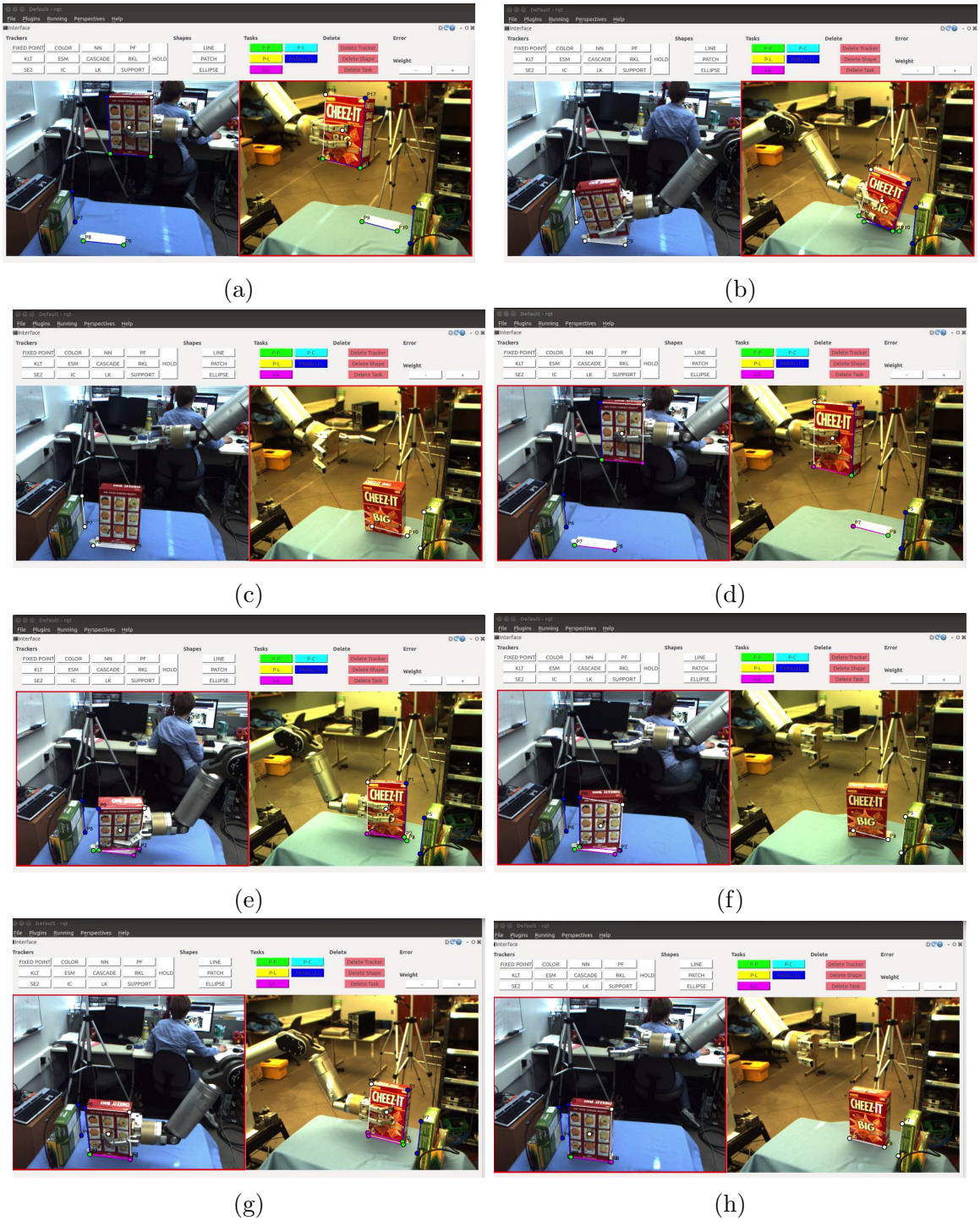


Figure A.5: Orientation has been adjusted and we add new tasks to translate the box to the table. (a) shows our first specification. We use joints 1, 2 and 4. (b) and (c) The result. We can place the box, but not accurately. (d) A new task composition with joints 1, 2, 4 and 6. (e) and (f) A Better result, but the orientation is still off in one direction. (g) and (h) Results of last attempt with the same tasks but one more joint (1, 2, 4, 5, and 6). We succeed and place the box on the white rectangle.

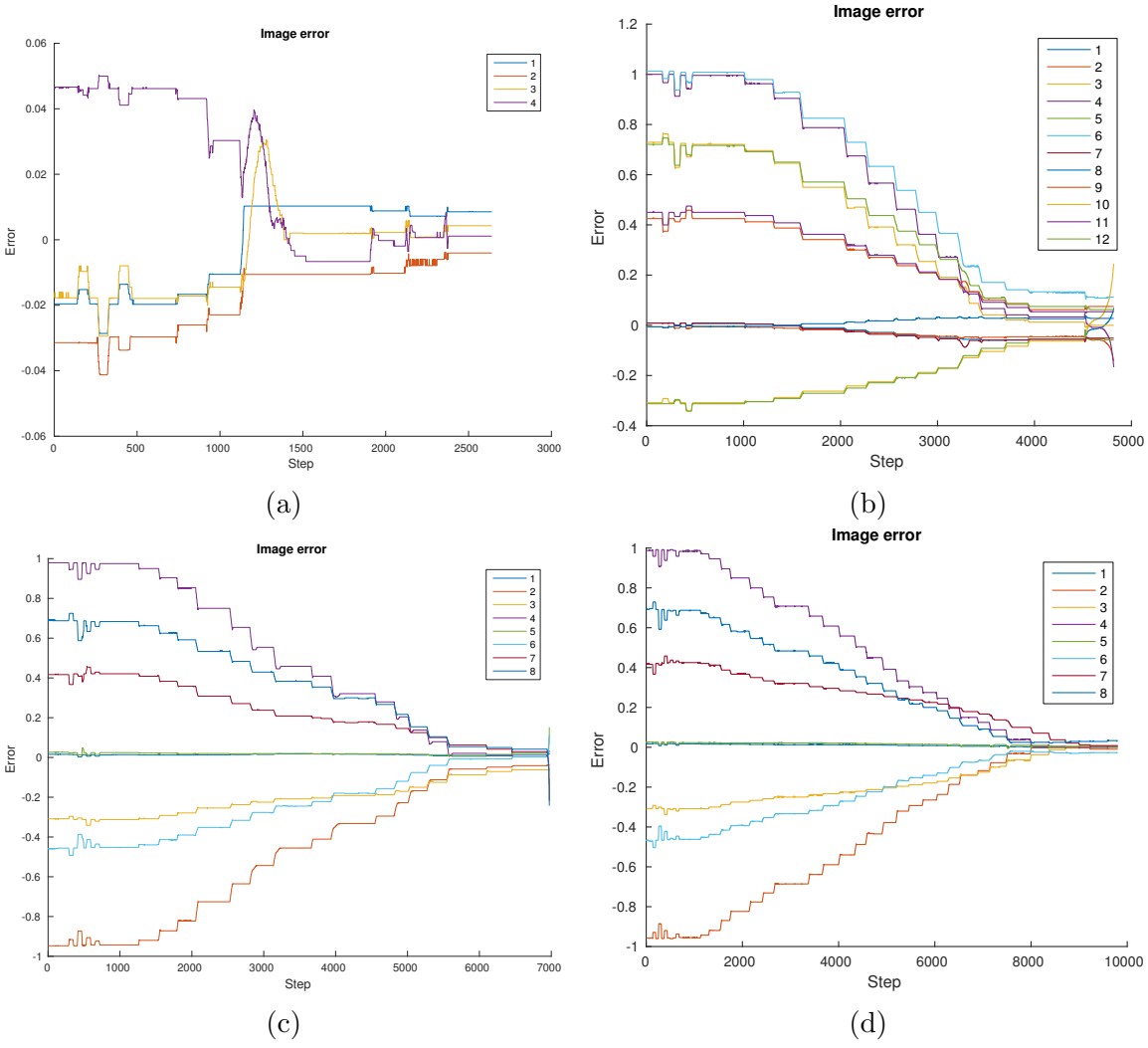


Figure A.6: (a) shows the parallel lines task error for orienting the gripper. (b) corresponds to the first failed attempt. Errors 1, 2, 7 and 8 represent parallel lines tasks, while the other are point-to-point errors. We can see that the parallel lines errors start out small but increase throughout. (c) and (d) show the errors from the last two attempts, respectively. 1 and 5 are parallel lines errors, 2 and 6 are line-to-line errors and the rest are point-to-point errors. In these cases the parallel lines errors remain small throughout.

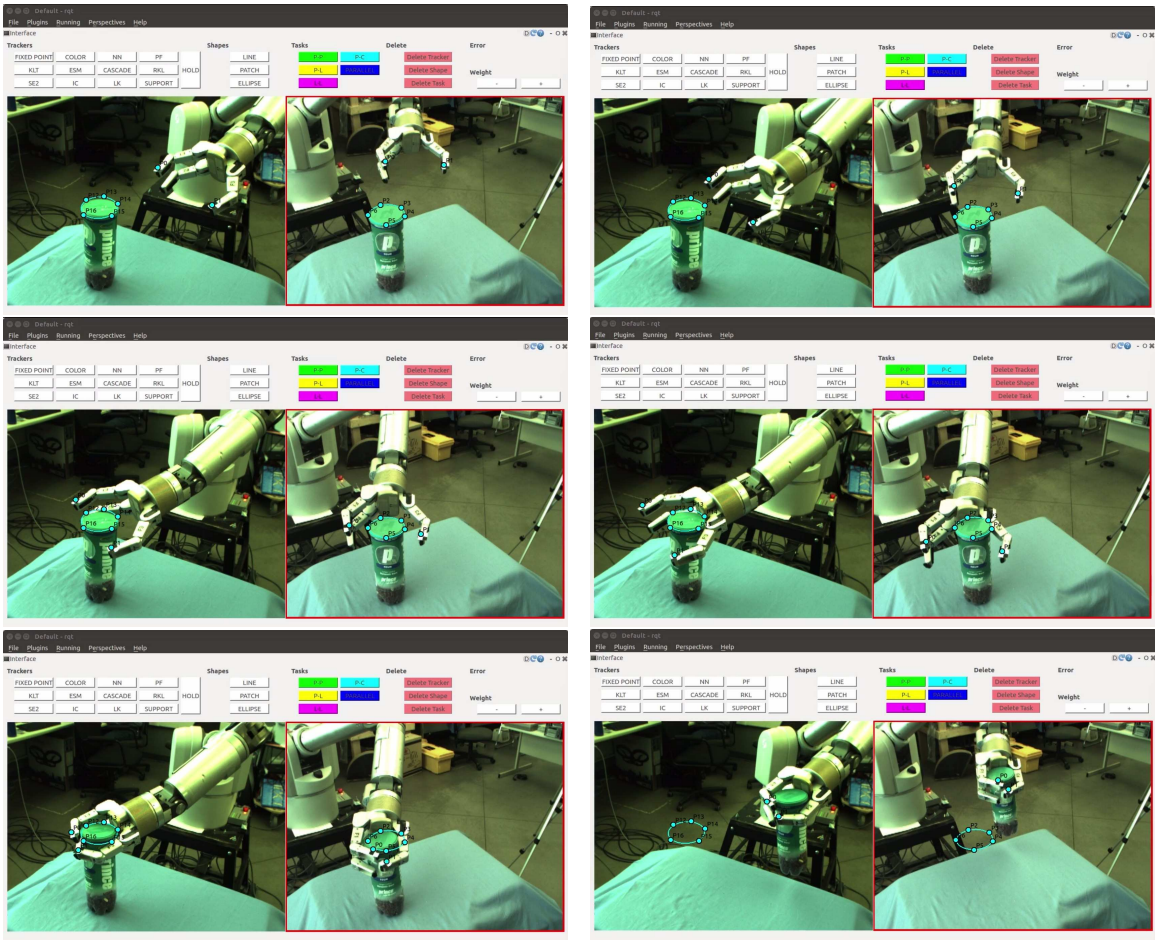


Figure A.7: This figure shows the execution of grasping a cylinder. We specify two point-to-ellipse tasks in each image. The combination of the two tasks allows the gripper to centre on the cylinder and approach it.

A.3 Grasp Cylinder

In this task we want the robot to grasp a cylinder from the side, see Fig. A.7. The cylinder is 7.7 cm in diameter and 20.8 cm tall. We specify a point-to-ellipse task form each of the two robot fingers to the ellipse. The combined error vector is of size four and we control joints 1, 2 and 4 of the robot. Since the gripper is open when it moves to the cylinder, neither of the two point-to-ellipse task errors will converge completely to zero, see Fig. A.8. By specifying two such tasks, the visual servoing will centre the gripper around the cylinder trying to minimize both tasks simultaneously. The starting position of the gripper is not very far from the cylinder and so we do not need to worry about the visual servoing reaching another local minimum (one

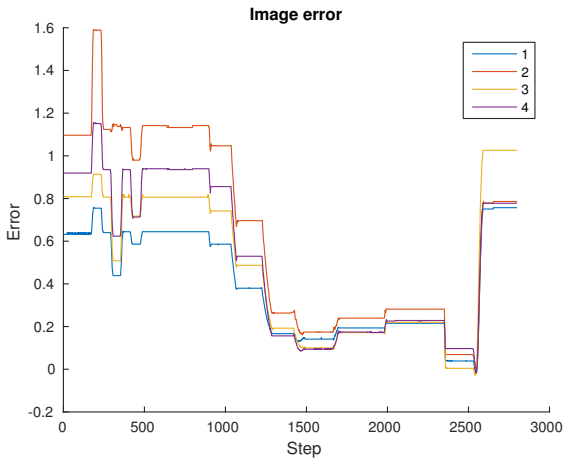


Figure A.8: The plot shows the error of each point-to-ellipse task. 1 and 2 represent the errors in the first image and 3 and 4 the errors in the second image. We can see that the errors behave very similarly. The spike in the error at the very end is due to the robot grasping the cylinder.

finger converging to its goal point and leaving the gripper at the side of the cylinder). Reaching a good starting position can easily be accomplished by coarse translation of the gripper towards the cylinder using a point-to-point task. The user needs to click 33 times in the interface and create four geometric constraints to specify the grasp.

Grasping the cylinder is a coarse manipulation task and the cylinder is fairly tall so we do not need the precision to be very accurate. Therefore two point-to-ellipse tasks are sufficient for centring the gripper around the cylinder. If we instead needed more accurate gripper orientation we would have to specify three point-to-ellipse tasks.

Appendix B

Fine Manipulation Experiments

B.1 Marker in Cap

The first experiment inserts a marker in its cap. The cap has a diameter of 9.9 mm, while the marker tip's diameter is 2.5 mm and the diameter of the marker (including the added tape) is about 9.5 mm. This is similar to the experiments, where glue in a syringe is applied to a point [50] and putting a screwdriver on a screw [49]. The cap is placed vertically on a table and the marker is held in a pinch grasp between two robot fingers, see Fig. B.1. In order to make the task as general as possible, the orientation of the marker is perturbed in two directions, i.e. it is not parallel to the cap. The robot must align the marker with the cap, centre the marker above the cap and insert it. The task from [50] only deals with orientation offset of the syringe in one direction, while the experiment in [49] seems not to take orientation into consideration, only displacement.

To complete the marker in cap task, we divide it into three segments. First we adjust the orientation of the marker so it is parallel with the cap. Second we translate the marker to align its tip above the cap. Finally, we insert the marker in the cap. For the first part we specify two parallel lines tasks, while controlling two joints of the robot wrist (joint 6 and 7 for pitch and roll). These tasks correct the orientation of the marker so it becomes parallel with the cap, see Fig. B.1.

Next we want to translate the marker to the opening of the cap. We keep the parallel tasks, but add a point-to-point task from the marker tip to a point above the opening of the cap, see Fig. B.1. Since orientation has already been solved, we use joints 1, 2 and 4 to translate the marker to the cap. The point-to-point goal point is specified as a free space point hovering slightly above the cap. This ensures that

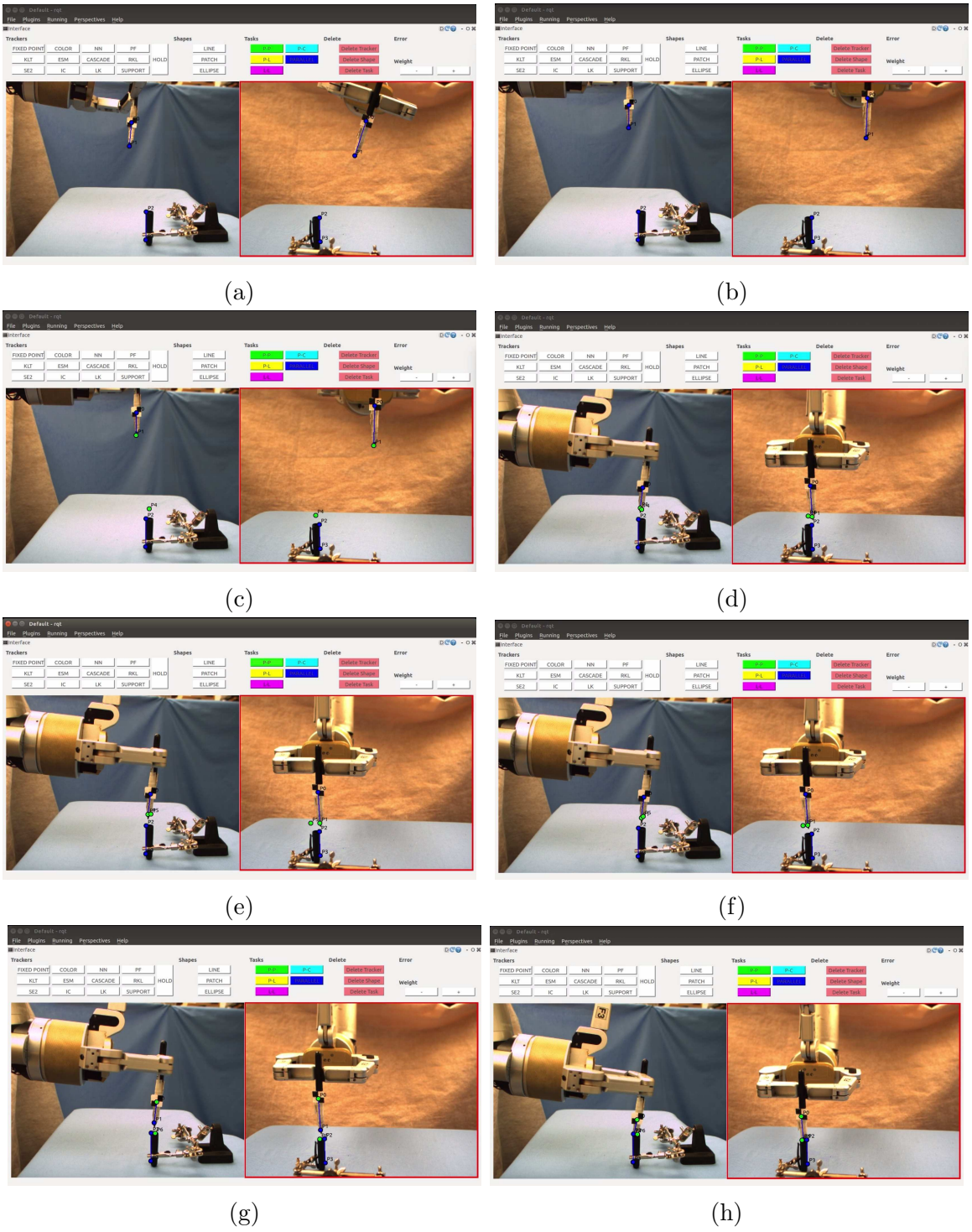


Figure B.1: This figure shows the sequence of the marker in cap task. (a) Specify the parallel lines task to adjust orientation. (b) The orientation has converged. (c) We specify a point-to-point task to move the marker to the cap. (d) The translation is completed. (e) We move the point-to-point goal point to more finely adjust the marker position. (f) The adjustment is done. (g) We specify the constraints to insert the marker. (h) The task is done.

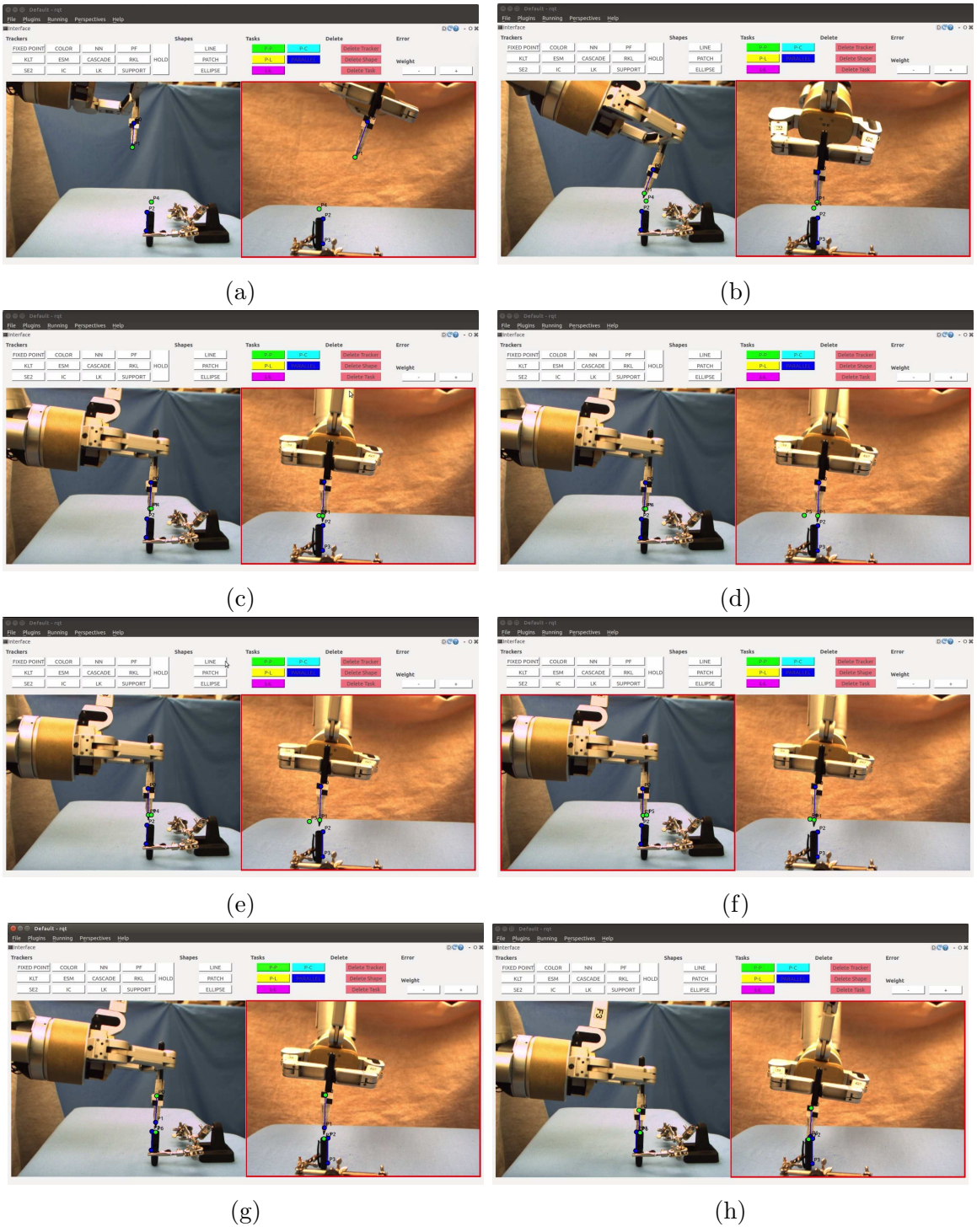
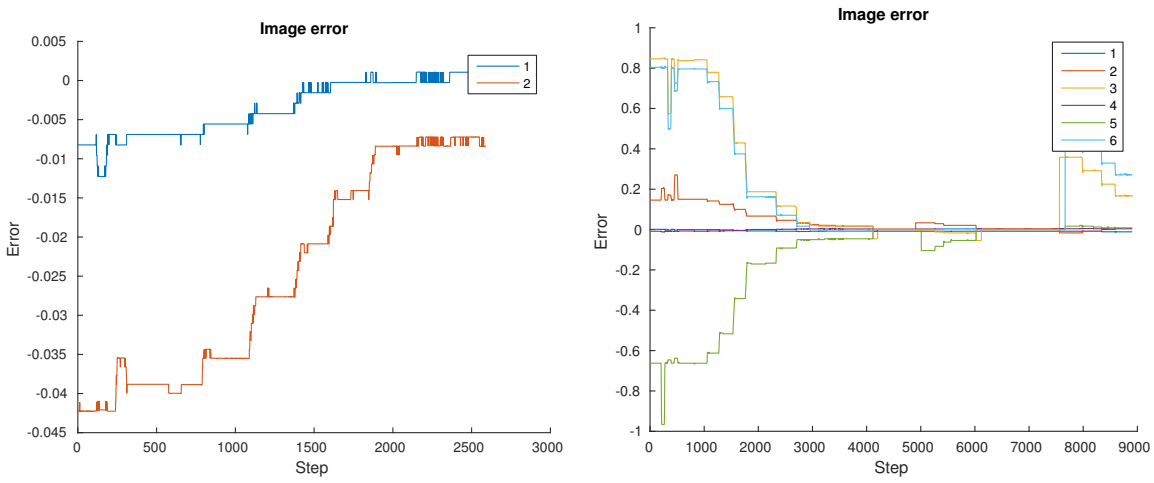


Figure B.2: A second way of specifying the task. (a) Parallel lines task to adjust orientation and a point-to-point task to move the marker to the cap. (b) Pitch orientation is overshooting the goal before it converges in (c). (d) We make an adjustment to the marker position, which reaches its goal in (e). (f) We specify another adjustment. (g) We define the task to insert the marker in the cap. (h) The task is done.



(a) (b)

Figure B.3: The plots show the change in the task errors over time. (a) shows the error of the parallel lines tasks in the first part of the task. The first error corresponds to the first image. (b) shows the errors form the second part of the task. 1, 2 and 3 correspond to the parallel lines error, the x and y coordinates of the point-to-point task, respectively. Error 4, 5 and 6 follow the same sequence for the second image.

we do not run into the cap. A collision can happen because we are not tracking the exact tip of the marker, but instead slightly above it. Therefore the marker moves too close to the cap before the position has converged in the table plane. Setting the goal point above the cap will also give us the chance to make fine adjustments to the marker's position before inserting it. Although the goal points are specified in free space and cannot be guaranteed to represent the exact same 3D point, they work well. Since they are specified in two different images the visual servoing will mediate whatever discrepancy there may be between the two point locations.

The marker in cap task requires good precision. It is difficult to attain a sufficient level of precision in one large motion. Hence, when the translation to the cap has been accomplished, we need to adjust the location of the marker before we can finally insert it into the cap. Because we have a human-in-the-loop we can rely on the user to judge whether higher accuracy is needed. We can adjust the marker's placement by moving the the location of the goal point in the point-to-point task, see Fig. B.1. This does not change the composition of tasks and so we do not need to change the Jacobian. We made one adjustments before the marker was correctly centred above the cap.

The final part of the task is the insertion of the marker in the cap. Similarly to

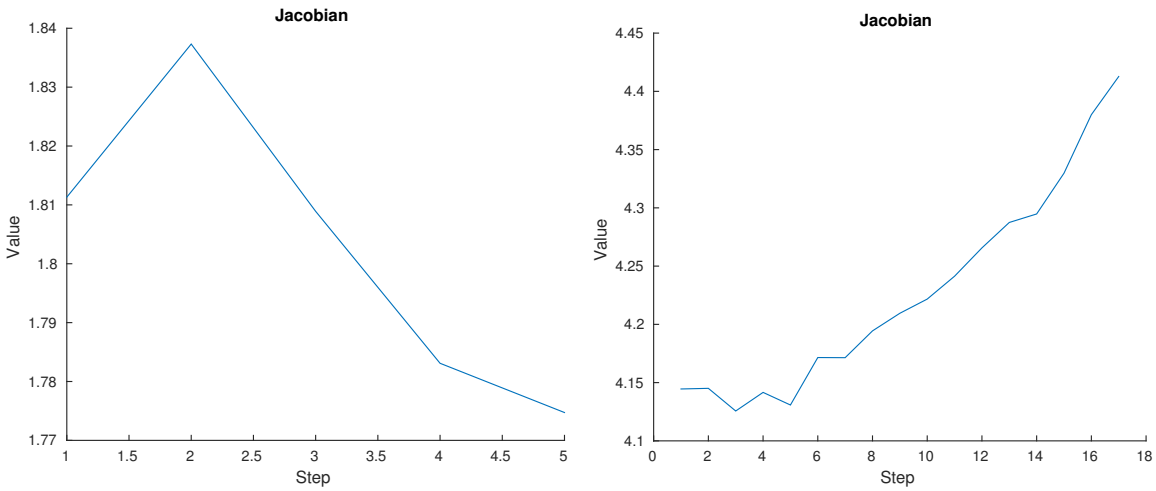


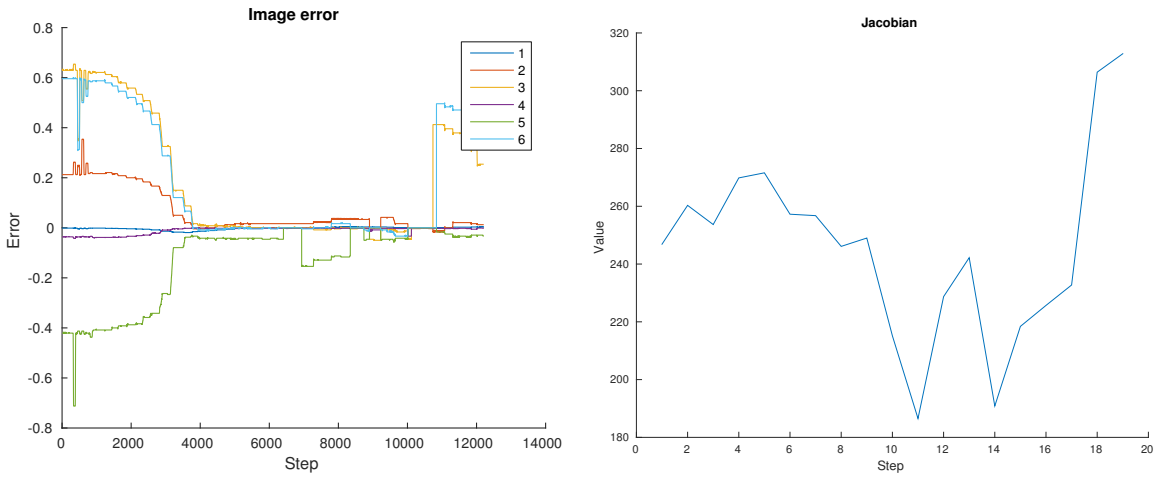
Figure B.4: The plots show the change in the Jacobian condition number throughout the task. (a) Orientation. (b) Approach to cap.

when we are adjusting the location of the marker above the cap, we can now complete the insertion simply by changing the point-to-point tasks to create a downward movement, see Fig. B.1.

Plots of the task errors can be seen in Fig. B.3. As expected the task errors decrease. The first bump in the error in the second plot corresponds to when we move the point-to-point task to adjust the position of the marker. The final bump happens when we want to insert the marker. After this the error does not go completely to zero because the moving point is specified fairly high up on the marker. The evolution of the Jacobian condition number can be seen in Fig. B.4. The Jacobian changes when a tracker has moved ten pixels and is updated using the Broyden method from Section 3.4.1.

An alternative approach to solving the marker in cap task, which may also seem more natural, is to do the orientation and translation step simultaneously by specifying the parallel lines and point-to-point constraints together from the beginning. We were able to complete the task in this way, see Fig. B.2. The resulting error and Jacobian plots can be seen in Fig. B.5.

Specifying all the tasks at once eliminates one step in the process. However, we see that completing the task in parallel with more joints leads to a higher Jacobian conditioning number compared with the sequential approach. Moreover, since the robot now uses five joints (1,2,4,6 and 7) during the whole task, the position adjustment at the end becomes a little more challenging. When we adjust the point-to-point goal



(a) (b)

Figure B.5: (a) shows the change in the task errors over time. 1, 2 and 3 correspond to the parallel lines error, the x and y coordinates of the point-to-point task respectively in the first image. Error 4, 5 and 6 follow the same sequence for the second image. In the plot we can see two bumps in the error where we adjust the position of the marker and a final bump for the insertion. (b) shows the Jacobian condition number.

point at then end, it is easy for the user to judge the resulting 3 DOF translation. While controlling five joints, the motion is more unintuitive because orientation is also affected, see Fig. B.6. We end up adjusting the position twice. As seen in the second image in Fig. B.2, invoking all the joints also leads to a more unintuitive approach. The orientation in the first camera view (pitch) moves too far in one direction before getting adjusted right at the end of the task. This is visible for joint six in Fig. B.6 as the joint angle increases, before it turns and decreases. The feasibility of this alternative approach depends on the specific task at hand. In a task with larger translation distance, the difference in magnitude between the point-to-point and parallel lines tasks will be greater and the point-to-point task will likely dominate more. This was seen in Section A.1, where the robot grasps a box. The approach of dividing up the task will work well, whereas combining the tasks will work in some cases and fail in others.

While working with this task we realized that camera placement is essential. We placed the cameras so that they view the tasks from angles that are approximately 90 degrees apart. If the views are not sufficiently different, it is difficult to centre the marker above the cap. For example, if the cameras were both viewing the task from the front, then the placement of the marker would be precise sideways, but we

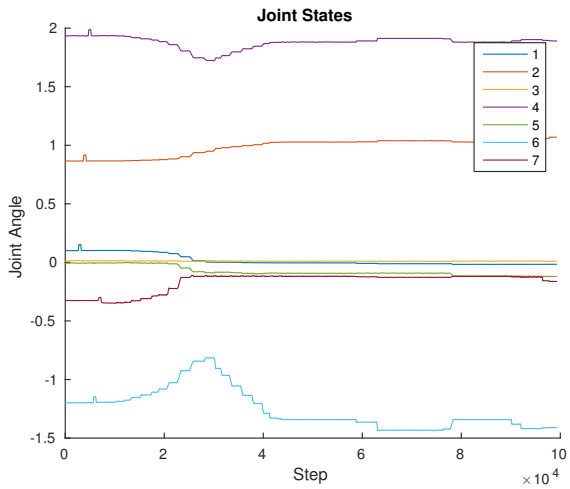


Figure B.6: This plot shows the joint angle of each joint throughout the task. A little after 4 on the x axis the marker has been moved to the cap. Right after 6 and right before 8 on the x axis we make the fine adjustment to the marker position and at about 9 we start the insertion. We can see that for both the adjustment and the insertion especially joint 6, but also joint 7 moves. These are the joints that take care of the marker orientation. Joint 6 has less than ideal approach to the goal because the angle increases past the goal and it decreases again before converging.

would loose precision in the direction pointing away from the cameras. Similarly, if the camera views are too close to each other, then only one of the two orientation displacements can be properly resolved. We can note that the parallel tasks used for this experiment is specified with different points on the marker in each image. The parallel lines task in one camera controls one direction of the orientation, whereas the task in the second camera controls the other orientation. Finally, we take care to place the cameras close to the task objects (about 25 cm) to achieve good resolution, while ensuring that they stay within the field of view.

For the marker in cap task the user needs to click 60 times in the interface when we include the one fine adjustment (46 clicks with no adjustments) and ten task are specified (four tasks with no adjustments).

B.2 Cube Insertion

In this task we would like to insert a cube into an opening only a bit bigger than the cube itself, see Fig. B.7. The cube's edges measure 32.5 mm, while the length of the opening is 34.6 mm. This leaves only 2.1 mm leeway in each direction and

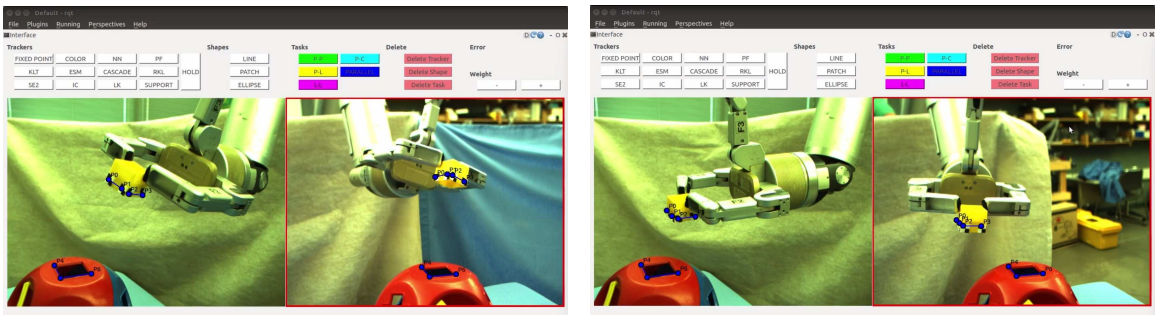
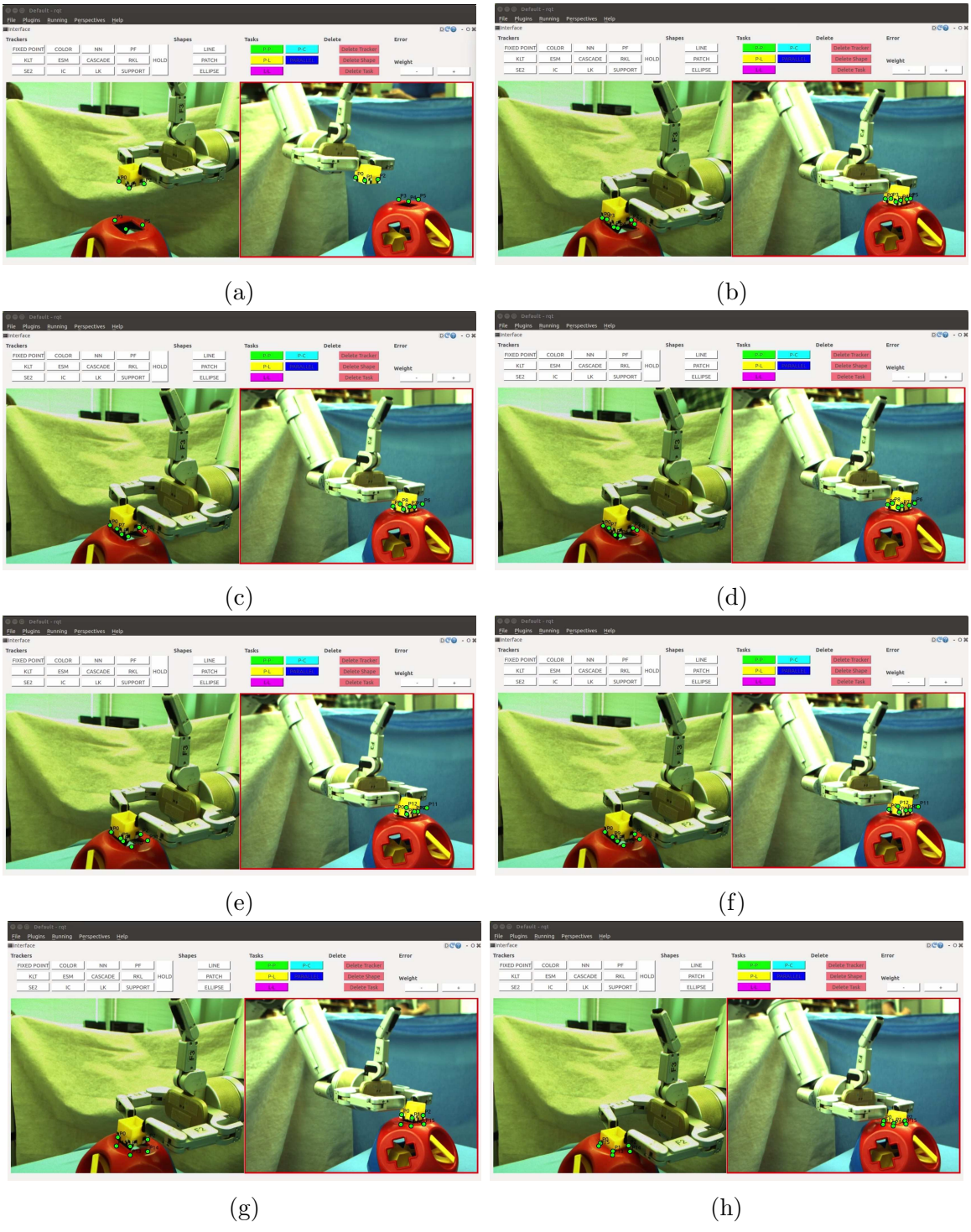


Figure B.7: We align the orientation of the cube with the square opening by specifying two parallel lines tasks. At the end of the alignment some of the trackers are about to get lost and we need to change the camera view.

so the task requires high precision. The cube's orientation is displaced in yaw, pitch and roll. Similarly to the marker in cap task, we complete the cube insertion in three stages. We adjust the cube's orientation, move it close to the square opening and finally insert it.

At the first stage of the task we try to align the cube's orientation with the square opening. We do this by specifying two parallel lines tasks, see Fig. B.7, and using the last three DOF of the robot (joint 5, 6 and 7 for yaw, pitch and roll). This is where we run into the first challenge with the cube insertion task. As we adjust the orientation, the trackers are about to go out of view just as the task converges. The cube is especially challenging to track because we need to see two faces of the cube. They easily get out of view when the orientation changes. To get around this we manually change the camera view, before setting the new task to translate the cube to the square opening. Since we use uncalibrated visual servoing we can move the cameras without problem. In our system we have not integrated moving cameras. Instead we suggest it as a useful path for future work. In particular, one can imagine putting one of the cameras on a second robot arm that the user can control.

Once the cube is accurately oriented we translate it to the square opening. We specify three point-to-point tasks with goal points hovering just above the opening, see Fig. B.10. Point-to-point tasks are useful in this case because they are simple to specify and they work intuitively with translation. Moreover, they all have the same magnitude error and we need some point-to-point tasks if we want to adjust the position of the cube towards the end of the translation. We use joints 1, 2 and 4. Once the cube is close to the goal we do the final fine positioning by adjusting the goal points. This adjustment is slightly more challenging than for the marker in



(a) (b) (c) (d) (e) (f) (g) (h)

Figure B.8: This figure shows the sequence of task execution for the cube insertion once the orientation has been aligned. (a) and (b) We specify three point-to-point tasks to translate the cube to the square opening. (c), (d), (e) and (f) We move the goal points to do two fine adjustments. (g) and (h) We set the goal points to get downwards movement and insert the cube.

cap task. We do two adjustments to accurately position the cube. Finally, we insert it by moving the goal points to create motion straight downwards. Because the task composition remains the same, we are able to move the goal points without needing to acquire a new Jacobian. Fig. B.9 shows the error plots and Jacobian condition number for the two parts of the task.

In one of the earlier iterations of the cube task we were not able to finish aligning the cube's orientation before the trackers went out of view. In particular, joint 7, which controls roll, was aligned, whereas the two other directions were not. At this step we tried two different approaches to moving the cube before arriving at a solution. Since the orientation was fairly close to correct, we decided to try and complete the translation to the square opening and the rest of the orientation adjustment at the same time. This strategy failed and we instead had to finish the orientation using two parallel lines tasks before translating to the goal. In our failed approach we tried a few different options. We combined two parallel lines task with one or two point-to-point tasks whose goal points were just above the square opening. We also tried specifying three and four point-to-point tasks, respectively, where the goal points were situated just above the square opening. Finally we also tried using two line-to-line tasks. See Fig. B.10 for an example. We used joint 1, 2, 4, 5 and 6 of the robot. It soon became apparent that fitting the cube into the square opening was very difficult if the orientation was not already aligned. With the above mentioned tasks we could get close to the goal and then we would move the goal points of the point-to-point tasks to try and adjust the final alignment. However, it was too difficult to adjust the points when five robot joints were moving. If the robot is moving with only three DOF translation, adjusting goal points is more intuitive as the user can more easily predict the resulting motion.

When working to find a solution to how to best translate the cube and do the final insertion we also tried a few different approaches. We tried to combine two parallel lines task with one or two point-to-point tasks. The idea was that the parallel lines tasks would keep the orientation correct while the point-to-point tasks would move the cube forward. First of all the translation was less accurate than when we specify three point-to-point tasks because we are working with free space points. Although it may seem more difficult, using three points works well because they average out some of the error when we specify the goals. One or two point-to-point specifications do not do as well on their own. This also held true for the fine adjustments. It was more difficult to get natural movement of the cube when using a parallel lines

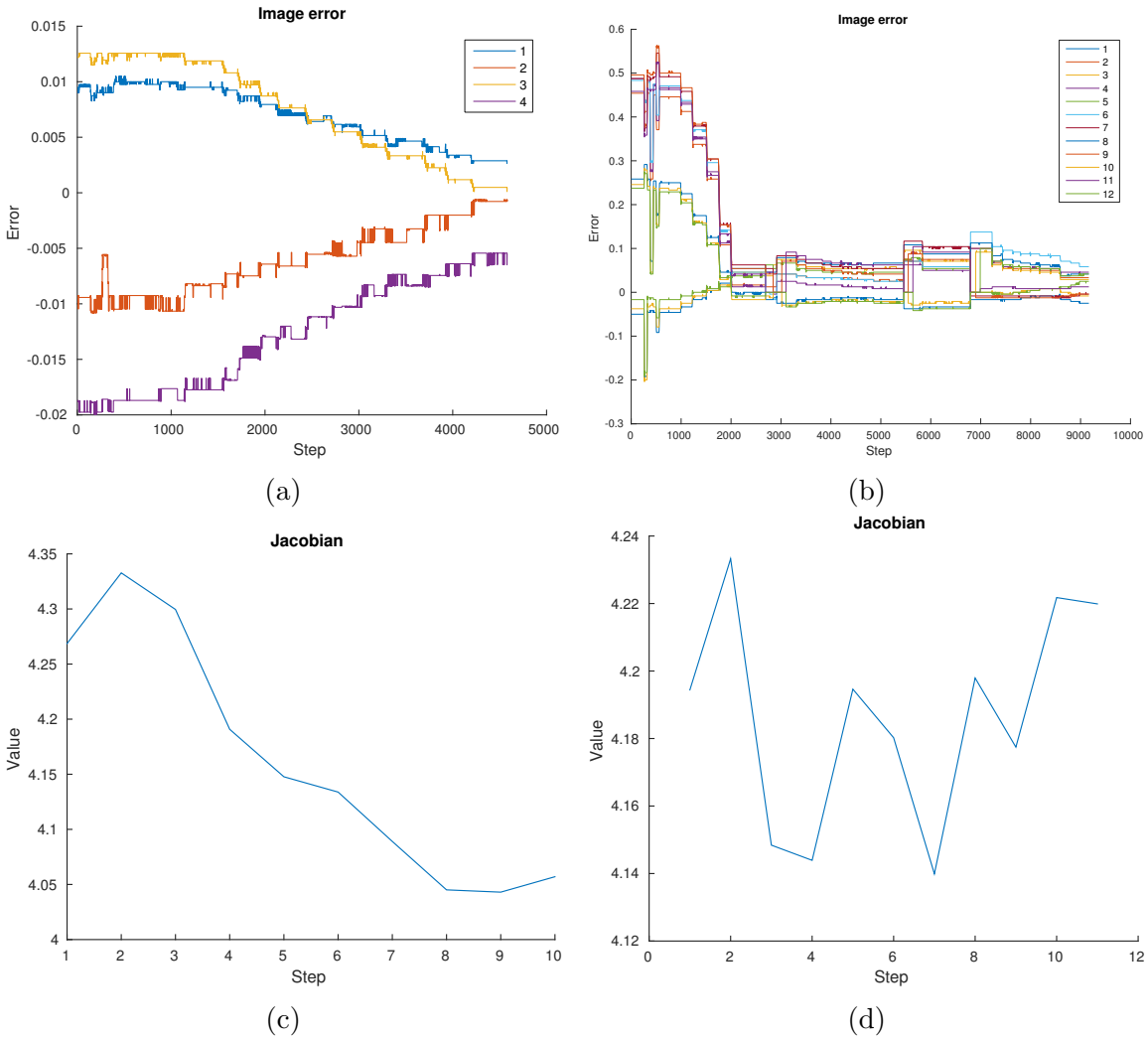


Figure B.9: (a) and (b) show the task error for the orientation and translation tasks respectively. In (a) 1 and 2 are the parallel lines errors in the first image and 3 and 4 in the second image. Similarly in (b) the first half of the lines belong to the point-to-point errors in the first image and the second half to the errors in the second image. In (b) the three bumps in the errors are due to two position adjustments and the insertion. (c) and (d) show the Jacobian conditioning number for the orientation and translation parts of the task.

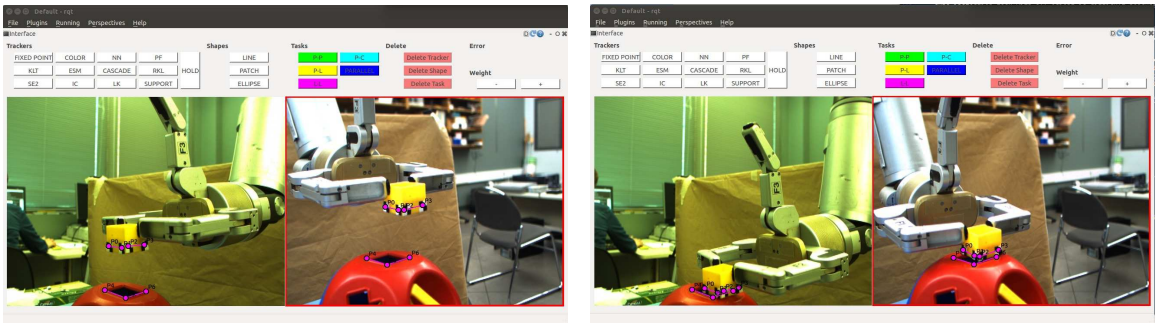


Figure B.10: An example of a failed attempt at translating the cube to the square opening while simultaneously correcting for orientation. This specification uses two line-to-line tasks.

task combined with one or two point-to-point tasks. As mentioned before, using only point-to-point tasks have the advantage of similar error magnitudes. Finally, we also tried to specify the translation task by setting four point-to-point tasks in each image with each goal point situated in the corner of the square opening. We thought the points would be more accurate as we could put them directly in the corners instead of in free space and that the cube would average out and servo to the middle. However, since we are not tracking the exact bottom edges of the cube, we ended up colliding with the container because the cube got too close before it was properly aligned with the opening.

To summarize we accomplished the cube insertion task in three steps. We started by adjusting the orientation until the trackers went out of view. In one case we were able to finish the orientation before the trackers went out of view, in another case were not. In the latter case we finished the orientation alignment after moving the cameras. Next we translated the cube to the square opening and finally inserted it. There were three main challenges to the cube insertion task. First of all it was difficult to keep the trackers in view when the orientation of the cube was perturbed. Second, we needed to make sure that the cube was perfectly aligned before we could translate the cube to its goal. Finally, the fine adjustments at the end of the task are more time consuming as adjusting a cube is more difficult than in the case of the marker tip. The user needs to click 185 times in the interface when we include the two fine adjustments (115 clicks with no adjustments) and 28 task are specified (16 tasks with no adjustment).

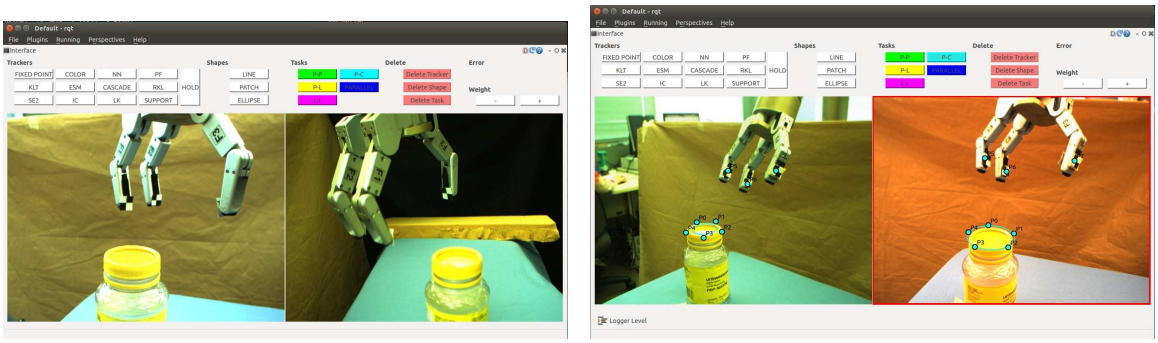
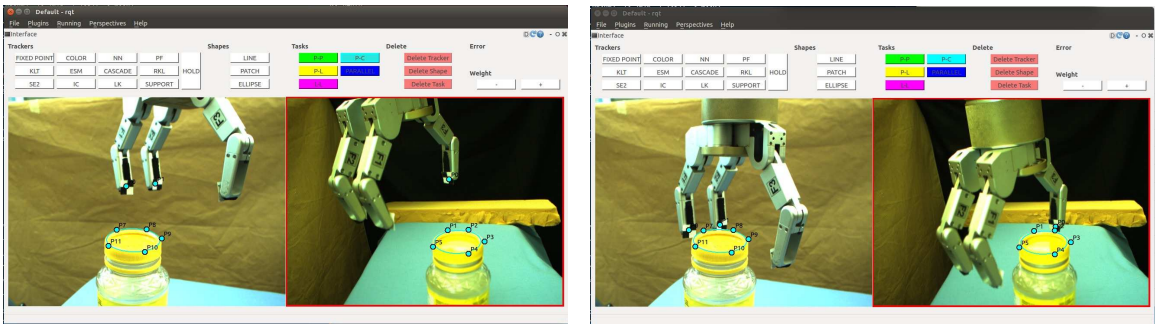


Figure B.11: The task is to grasp a cylinder from above. It could be a jar or the lid of a jar. We attempt two versions of the task. One where the orientation of the gripper is already aligned with the jar and one where it is not.



(a) (b)

Figure B.12: (a) We specify three tasks to control three DOF translation of the robot. (b) It looks like the task has converged. However, the specification is ambiguous and in the workspace the robot stops short in front of the jar.

B.3 Grasp Ellipse

We want to grasp a circular or ellipse shape from above, see Fig. B.11. This can be useful for grasping a cylindrical container like a jar from above or to grasp the lid of a jar. We complete our task with a jar of apple sauce. The lid has a diameter of 6.5 cm and is 1.5 cm tall. The jar is 14.7 cm tall. We complete two different versions of the task. First we try and grasp the lid in the simplified case, where the orientation of the gripper is already aligned with the plane of the lid. Second we attempt the grasp with an arbitrary orientation.

In the case where the orientation of the gripper is already aligned with the lid, we use a lower number of DOF of the robot. In fact, we only need to control three DOF translation. This means fewer tasks need to be specified compared to when

the gripper orientation is arbitrary. We set up two cameras that view the task from different angles. We specify point-to-ellipse tasks from the fingers of the robot to the ellipse projected into the image from the lid. Since the gripper is open the tasks will not converge completely to zero, but they will make the gripper centre on the ellipse. The gripper's initial position is fairly close to the ellipse and so we avoid any local minima, where the gripper could converge to the side of the lid. Getting the robot to a good initial position can easily be done with a point-to-point task coarse motion. We are only using joints 1, 2 and 4 to control translation.

In the first attempt we specify one point-to-ellipse tasks in one image and two point-to-ellipse tasks in the other, see Fig. B.12. Due to the view of the cameras it is difficult to track three points on the robot gripper in both images. However, we can make do with fewer points since we are only controlling translation. Though the task specification gives a vector of size three, the task does not converge to the goal. Together with the camera views, our specification gave an ambiguous definition of the task. Although it looks like the task has converged in the camera views, it does not reach the correct configuration in the workspace. Instead it stops short in front of the jar.

We proceed to set an unambiguous specification using two point-to-ellipse tasks in one image and three point-to-ellipse tasks in the other, see the first picture in Fig. B.13. This instance works and the gripper centres on the ellipse instead of stopping in front of it. Now we run into a second challenge, which is the elevation of the gripper compared to the lid. The ellipse is calculated on the top of the lid. Therefore the robot fingers try to align with the top of the lid. When closing the fingers, they are too high up and the grasp fails, see Fig. B.13. The same figure shows the results of the two next attempts at this task. We put the tracking markers higher up on the robot fingers to have the finger tips move lower when approaching the jar. This results in the fingers moving too far down and we end up grasping the jar instead of the lid. We make a final attempt at adjusting the markers. This brings the fingers close to the lid, but the final grasp fails because the lid slips. These experiments show that we can centre the gripper on the ellipse using point-to-ellipse task, but adjusting the height is challenging. Fig. B.14 shows a plot of the task errors as the gripper converges to the goal in the last execution. The errors behave similarly, though the error corresponding to the finger closest to the camera in the right image decreases before it increases again. This is consistent with its motion in the image. The visual servoing minimizes all the errors together and so these effects are expected.

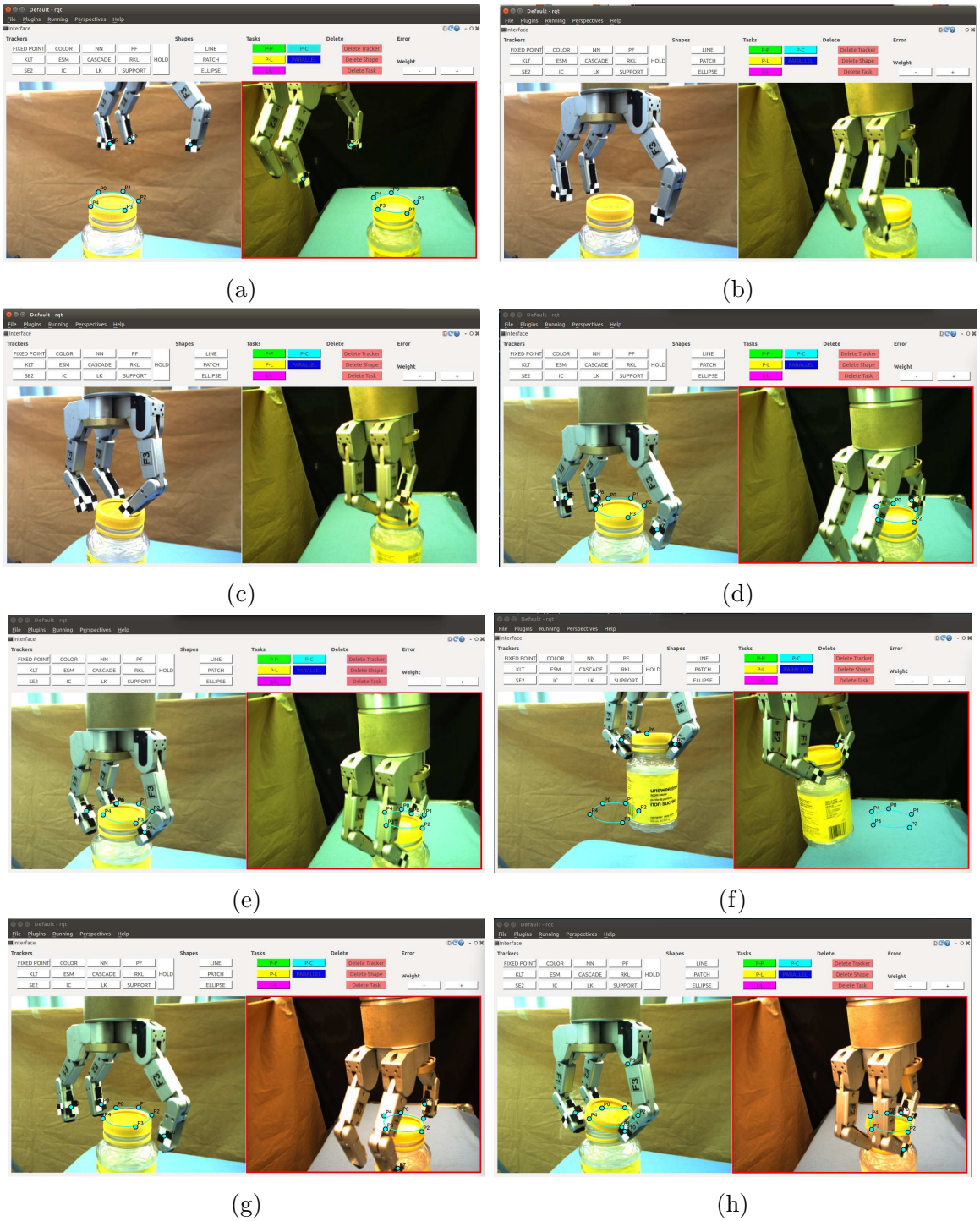


Figure B.13: Initial task specification and Results of a few runs of grasping. (a) Three point-to-ellipse tasks in one image and two in the other. (b) and (c) Result of the first attempt. Gripper centres on the lid, but is too high to grasp. (d), (e) and (f) Second run, with modified marker placement. The gripper is too low and grasps the jar instead of the lid. (g) and (h) Last attempt, with another marker adjustment. The lid slips.

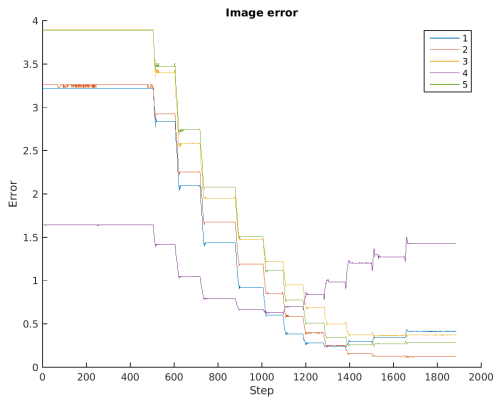


Figure B.14: The plot shows the image error of the task where we move the gripper to the ellipse. The numbers of the errors follow the tasks from left to right in the two images. The tasks errors decrease and behave similarly. One error decreases before it increase again. This behaviour can happen because visual servoing is minimizing all the errors together and consequently averaging their behaviour. The error do not converge to zero as the gripper is open.

In order to make the task more general we change the initial position of the gripper to be arbitrary. In our first attempt we specify three point-to-ellipse tasks in one image and two in the other. The examples in Fig. B.15 with corresponding plots in Fig. B.16 show that the task does not succeed. We also did one repetition of this task where we had changed the camera views to be more different, but the result was the same. The task errors converge, but the task does not because we have not specified a sufficient number of constraints. In order to properly align the orientation of the gripper with the plane defined by the lid, we need to specify three point-to-ellipse tasks in each image.

Fig. B.17 shows the results of specifying three point-to-ellipse task in each image. We run into two main challenges. When defining three point-to-ellipse tasks in both images, we need to be able to track three points on the gripper in both images. This is difficult when the orientation of the gripper changes. I particular, the last example in Fig. B.17 fails because a tracker gets lost. In order to be able to track three points on the gripper, we need similar camera views. In the three tasks corresponding to the four first images from Fig. B.17, the errors converge, as can be seen in Fig. B.18. However, the gripper is not able to grasp the lid because the lone robot finger is placed much higher than the other two. This happens because it looks like the task has converged in both cameras, but these views are not sufficiently different to get

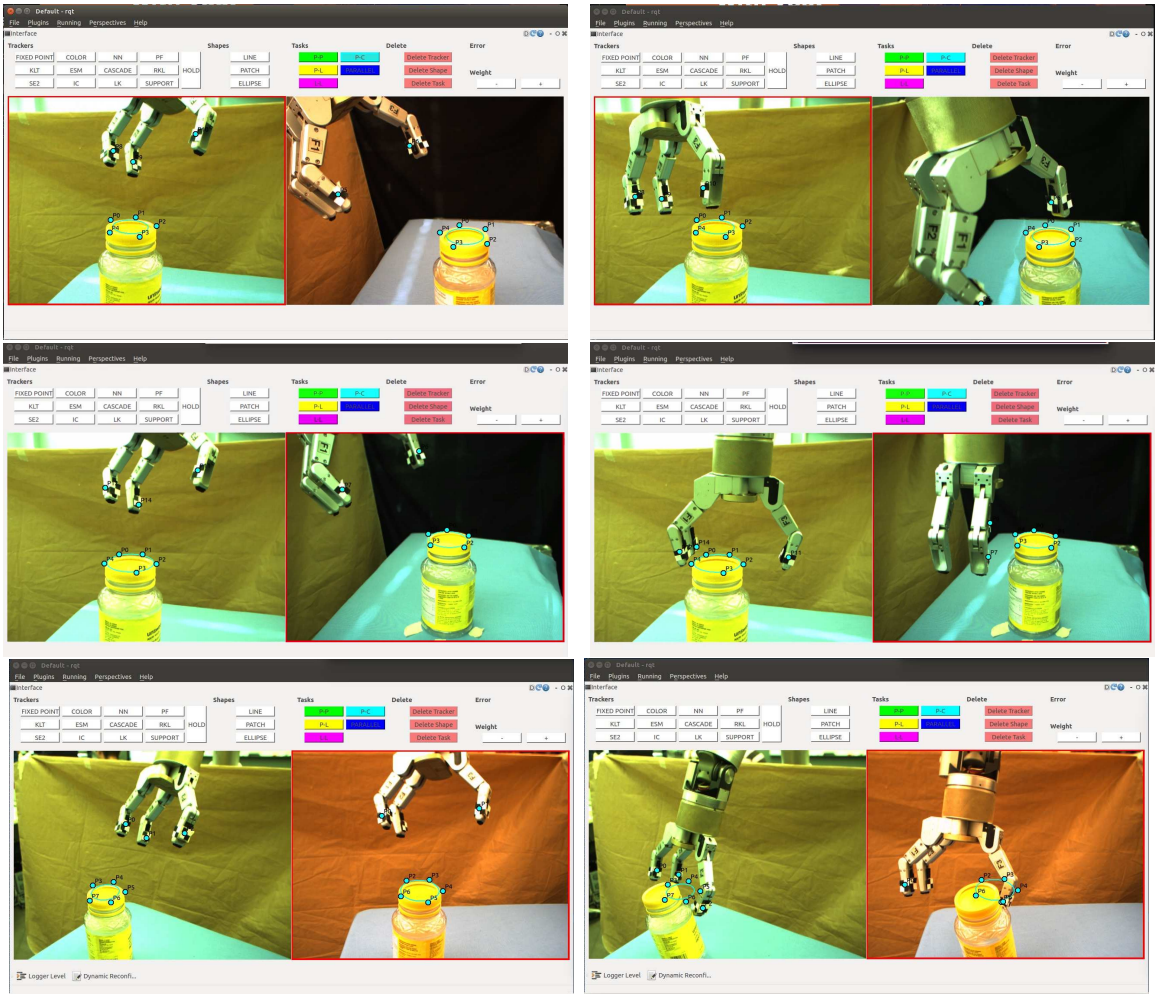


Figure B.15: This figure shows three attempts at grasping the lid when the orientation is not aligned. The specification with three point-to-ellipse tasks in one image and two in the other can be seen in the images to the left. The resulting approaches can bes seen to the right. The specification is not sufficient and the approach to the jar fails. In the last example we have moved the cameras to get more different views, but the result is the same. In the second example we also see that the trackers are about to go out of view at the end of the task.

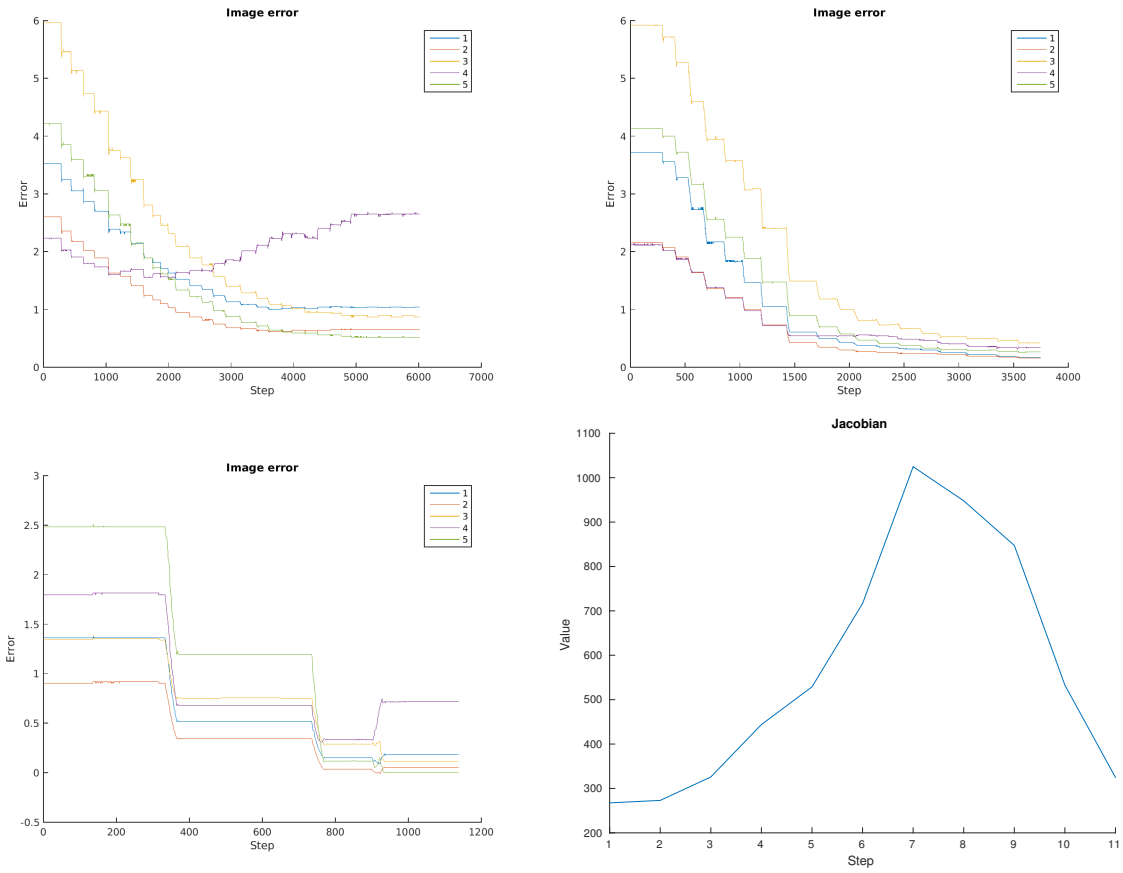
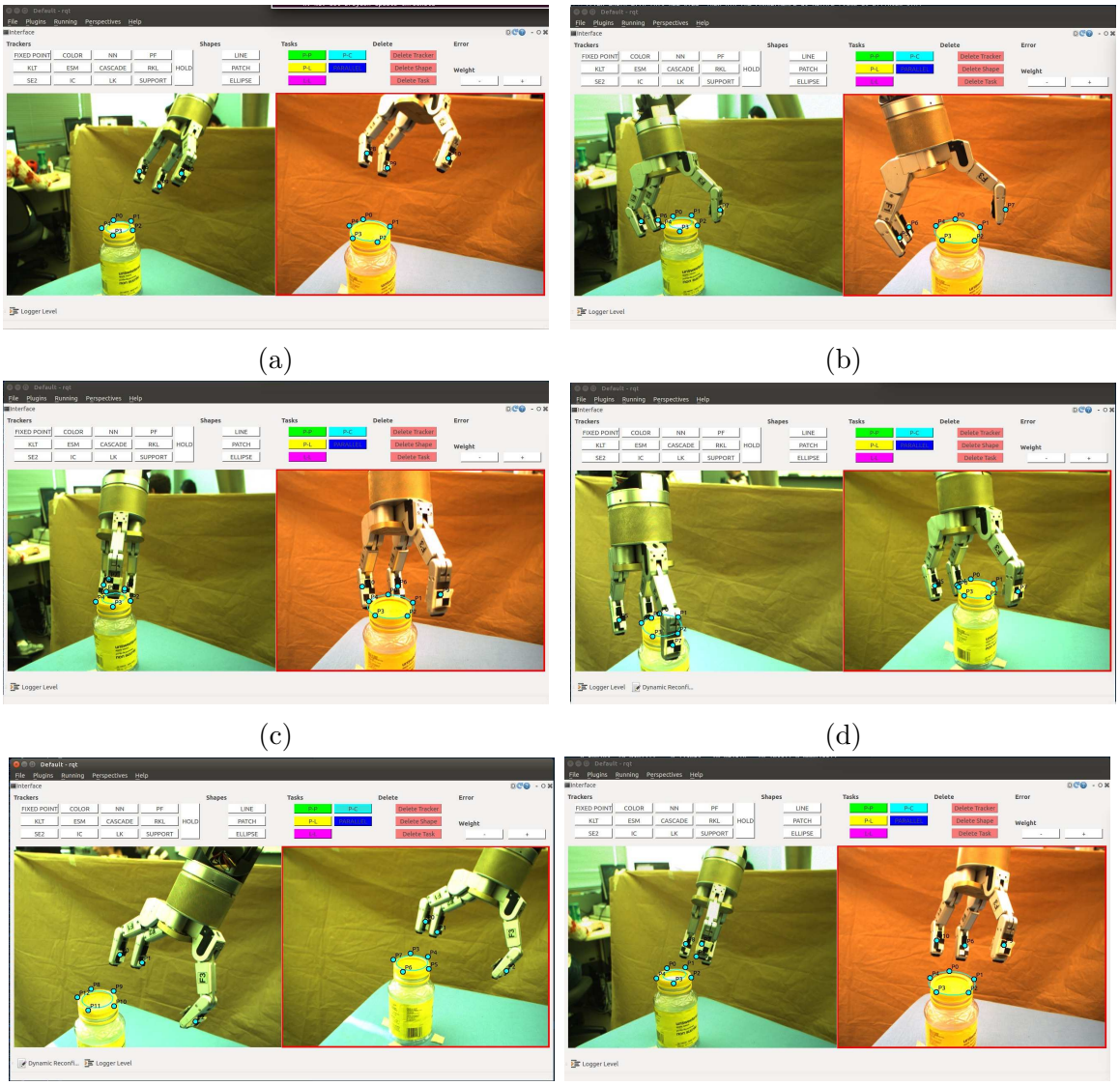


Figure B.16: This figure shows the plots corresponding to the tasks where we specify five point-to-ellipse tasks in total to control the motion of the gripper when its orientation is perturbed. The plots show the image error for each of the tasks as well as the Jacobian condition number for the last task. The error numbers correspond to the task listed from left to right in the two camera views.

a good and unambiguous view of the task. Finally, in the second last result from Fig. B.17 the robot moves close to the lid in a similar way to the other examples, but then turns around and diverges. The plots in Fig. B.18 show the results for two of the convergent tasks and the one that diverges.

We are able to able grasp a jar from above using point-to-ellipse tasks if the orientation of the gripper is already aligned. However, controlling how far down the gripper moves is not straight forward. With perturbed orientation the task becomes difficult because we must track three points on the gripper in two images. This forces us to set up the cameras with similar views, which affect the accuracy of the task when the errors converge. The user clicks 41 times in the interface to specify the task.



(a) (b) (c) (d) (e) (f)

Figure B.17: This figure shows five different attempts at grasping the lid when we start from the initial position in (a). In (b), (c) and (d) the gripper converges towards the ellipse, but the task will not succeed. Because the image views are similar, even though the task converges, the positioning of the gripper is not accurate enough. In (e) we have an example where the gripper moved towards the ellipse, but then turned around and diverged. (f) shows an example where a tracker gets lost. This illustrates how difficult it is to track all three fingers.

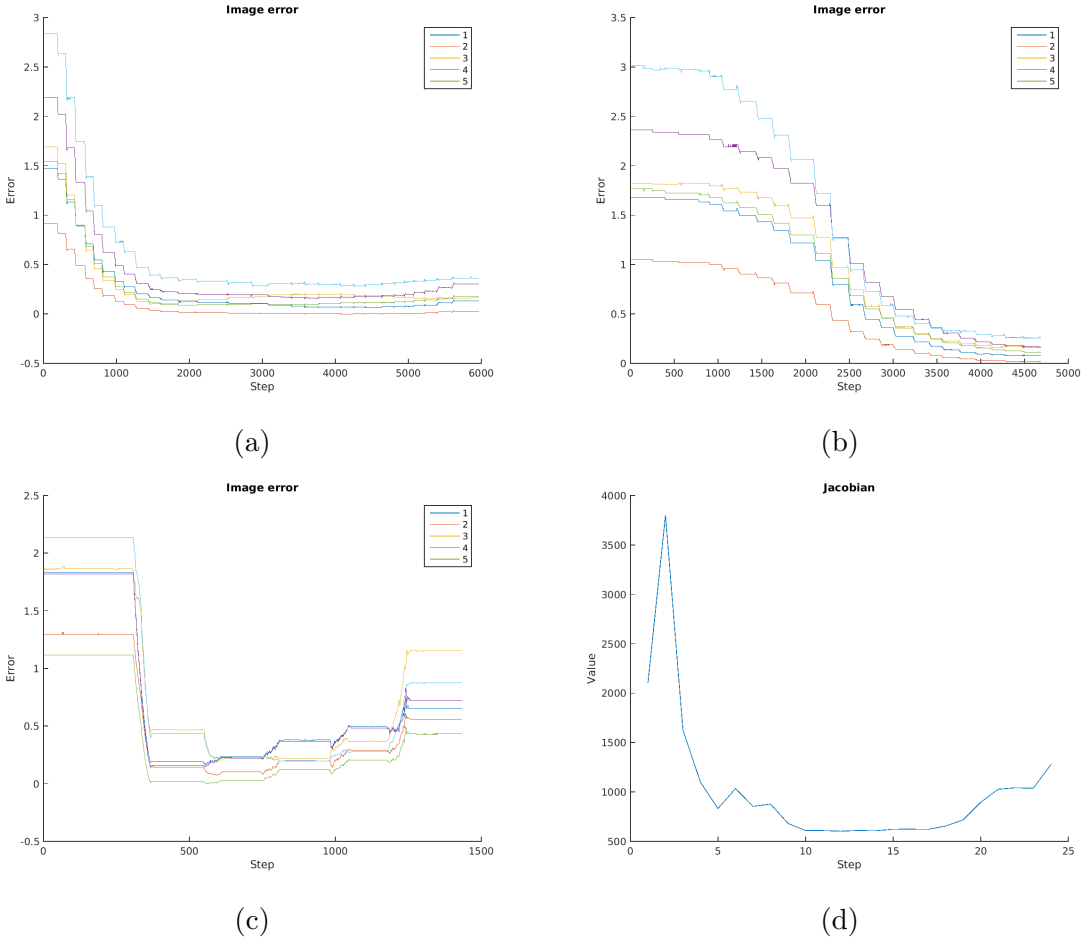


Figure B.18: We show error plots from three of the tasks done with perturbed orientation. The corresponding pictures of the tasks are shown in in Fig. B.17. (a) corresponds to the second image in Fig. B.17. We can see that the errors converge. Similarly (b) corresponds to the third image in Fig. B.17. Here the errors also converge. (c) shows the errors for the task that diverged. The errors first decrease as the gripper moves towards the jar, but then start diverging. (d) shows the Jacobian condition number for this task.

B.4 Line Following

We want to make the robot follow a line constraint. One can imagine this being useful in cases such as applying glue to an edge or for welding. We let the robot hold a marker that will follow the line we have set up. We use a marker because it is possible to track its tip. The tip has a 2.5 mm diameter. The line is made from sewing thread and the marker follows it for about 16 cm. Moreover, since the tip is round, we can track it equally from any direction. Using the robot finger to follow the line would result in lower accuracy since the finger is large compared to a marker tip. It would also be difficult to put markers on the square finger so that it could be tracked well from different directions.

We first bring the robot close to a line and then we make it travel along the line to a goal point. This means that we specify a point-to-line task from the marker tip to the line and a point-to-point task from the marker tip to the starting point on the line. See, Fig. B.19. Once the marker has reached the line, we change the point-to-point goal to a point further down the line. We use joints 1, 2 and 4 for three DOF control.

In our experiment we have the cameras viewing the task from different directions, but from roughly the same height. The robot moves accurately to the first point, see Fig. B.19. When the robot moves along the line, it pushes forward a bit on the line as if it were making a slightly curved motion between the two points on the line. We suspect that this motion is caused by the point-to-point task and therefore implement the line following only using two point-to-point tasks, see Fig. B.20. Indeed, we find that the movement is the very similar in both cases, see Fig. B.21. Fig. B.22 shows the corresponding point tracker trajectories in image space. Looking at the error plots in Fig. B.19 and Fig. B.20 we can see that the point-to-point error is larger and therefore dominates. When the marker touches the starting point on the line, the point-to-line error has already been minimized. When we move to initialize the new point-to-point task, its error will naturally dominate.

We try four different approaches to mediate the challenge of combining the point-to-line task with a point-to-point task. First we simply try to add more weight to the point-to-line task. Since the advantage of the extra weight diminishes when the point-to-line error approaches zero, this method does not help when we initialize the new point-to-point task, see Fig. B.23. In fact, the resulting path was worse with more curving than before, see Fig. B.21. Fig. B.22 shows the corresponding point

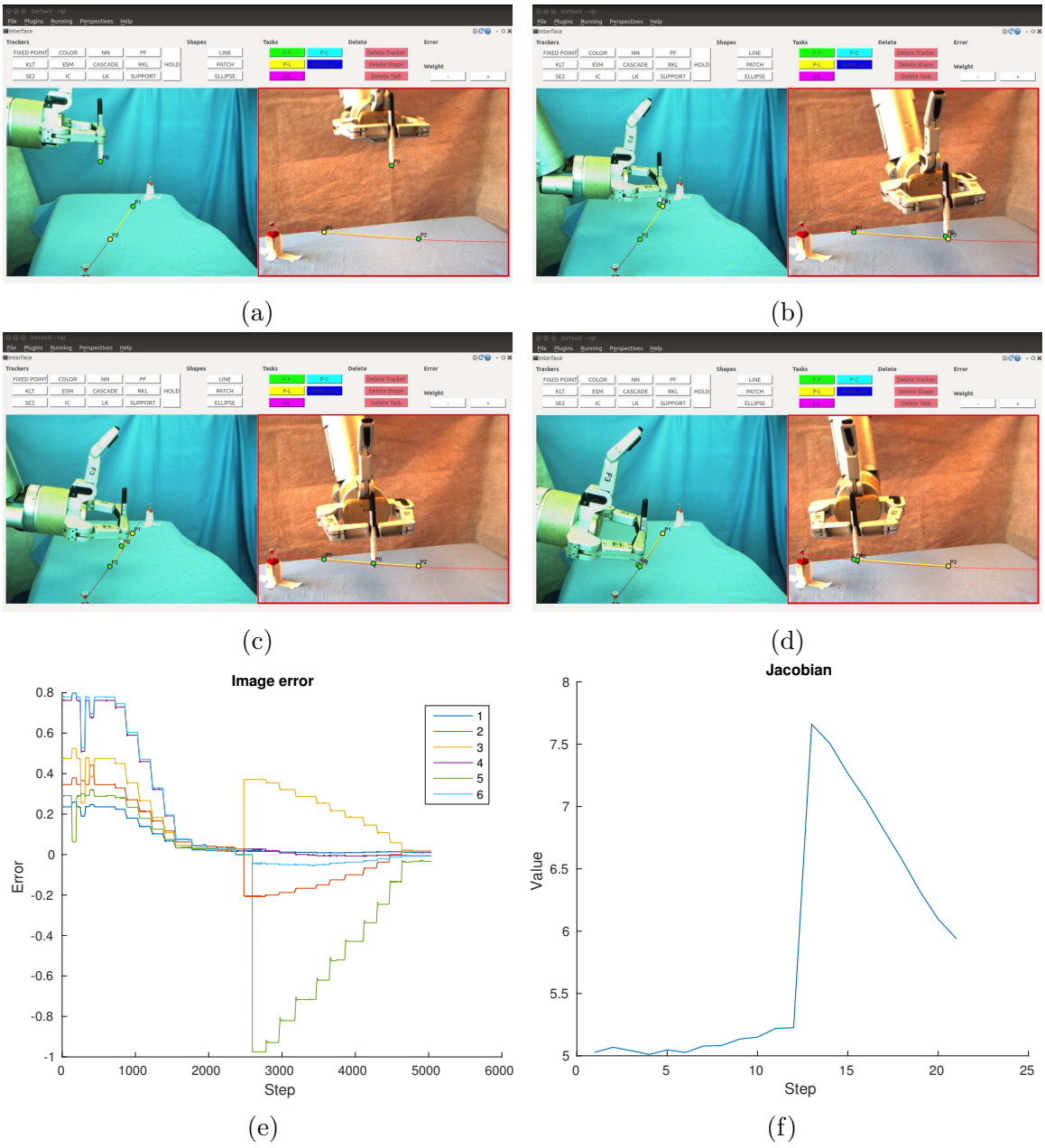


Figure B.19: (a) We complete a line following task by specifying a point-to-line task and a point-to-point task. (b) The robot has converged to the line and we change the goal point to the point-to-point task to move the robot along the line. (c) shows the robot following the line and (d) shows convergence to the goal point. (e) shows the image error. 1 and 4 represent the point-to-line errors in the first and second image, respectively. 3 and 4 belong to the point-to-point task error in the first image and 5 and 6 to the second image. (f) Jacobian conditioning number plot.

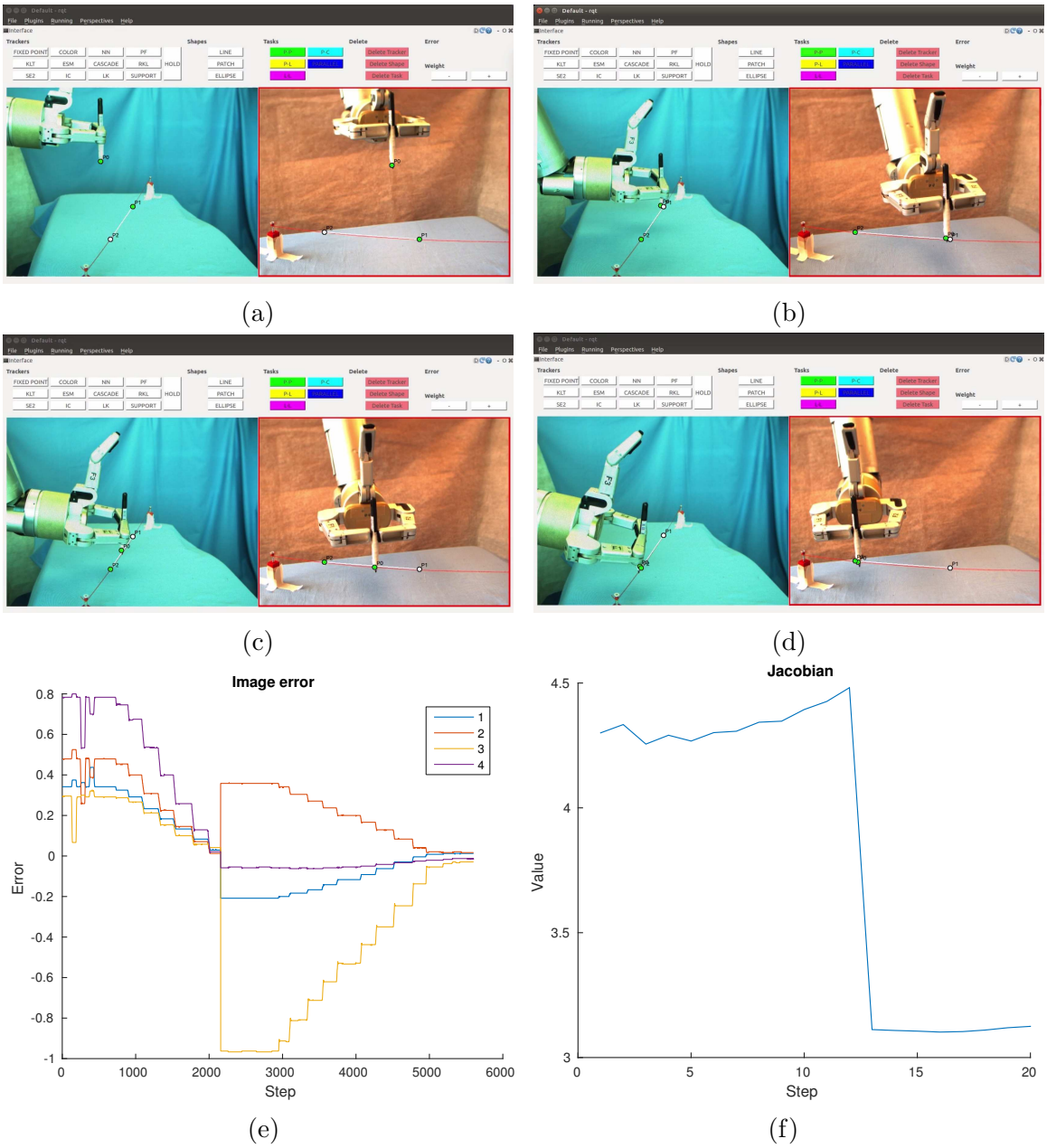


Figure B.20: (a) We complete a line following task by specifying a point-to-point task to a point on the line. In (b) the robot has converged to the line and we change the goal point of the point-to-point task to move the robot along the line. (c) shows the robot following the line and the fourth image shows convergence to the goal point. (e) shows the image error. 1 and 2 belong to the point-to-point task error in the first image and 3 and 4 to the second image. (f) Jacobian conditioning number plot.

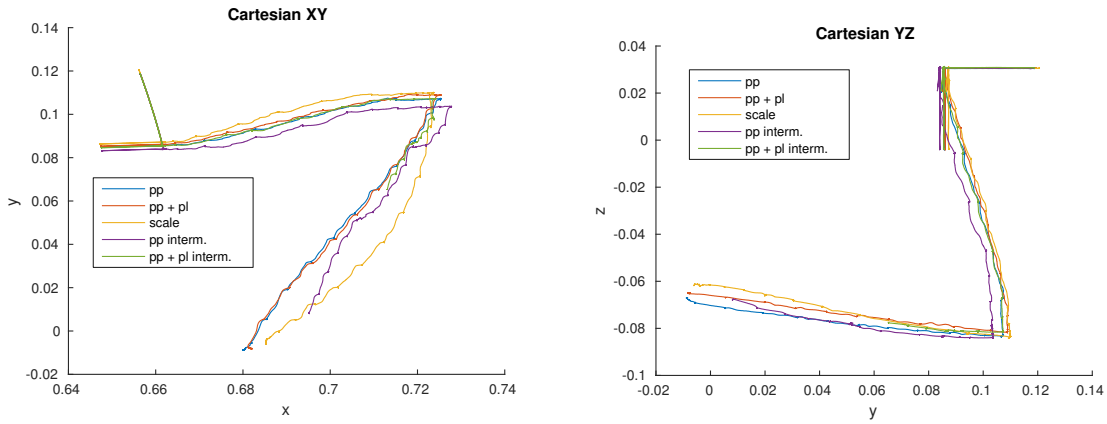


Figure B.21: The two plots show the Cartesian position of the end-effector during the execution of the different line following experiments. The first figure plots x and y, which corresponds to viewing the task from above, which means we see how much the marker moves back and forth in the table plane. The second figure plots y and z, which means we see the task from the side and therefore how much the marker moves up and down.

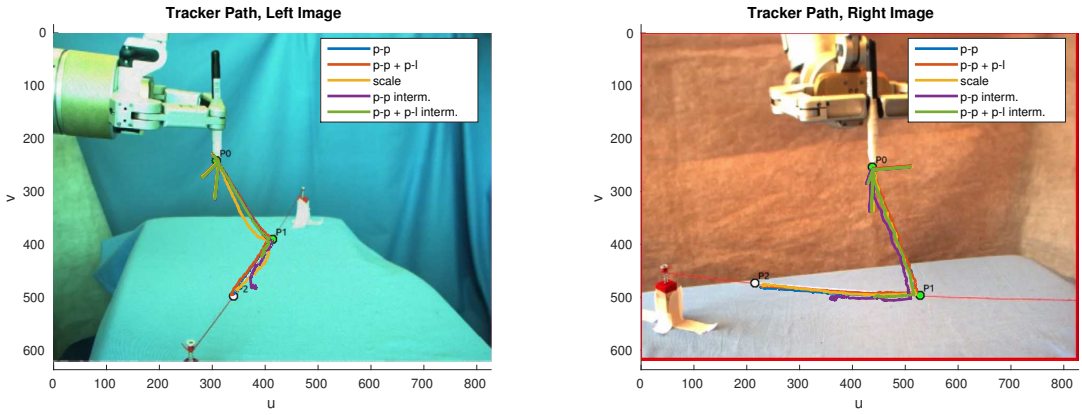


Figure B.22: The two plots show the image space trajectories of the point tracker during the execution of the different line following experiments.

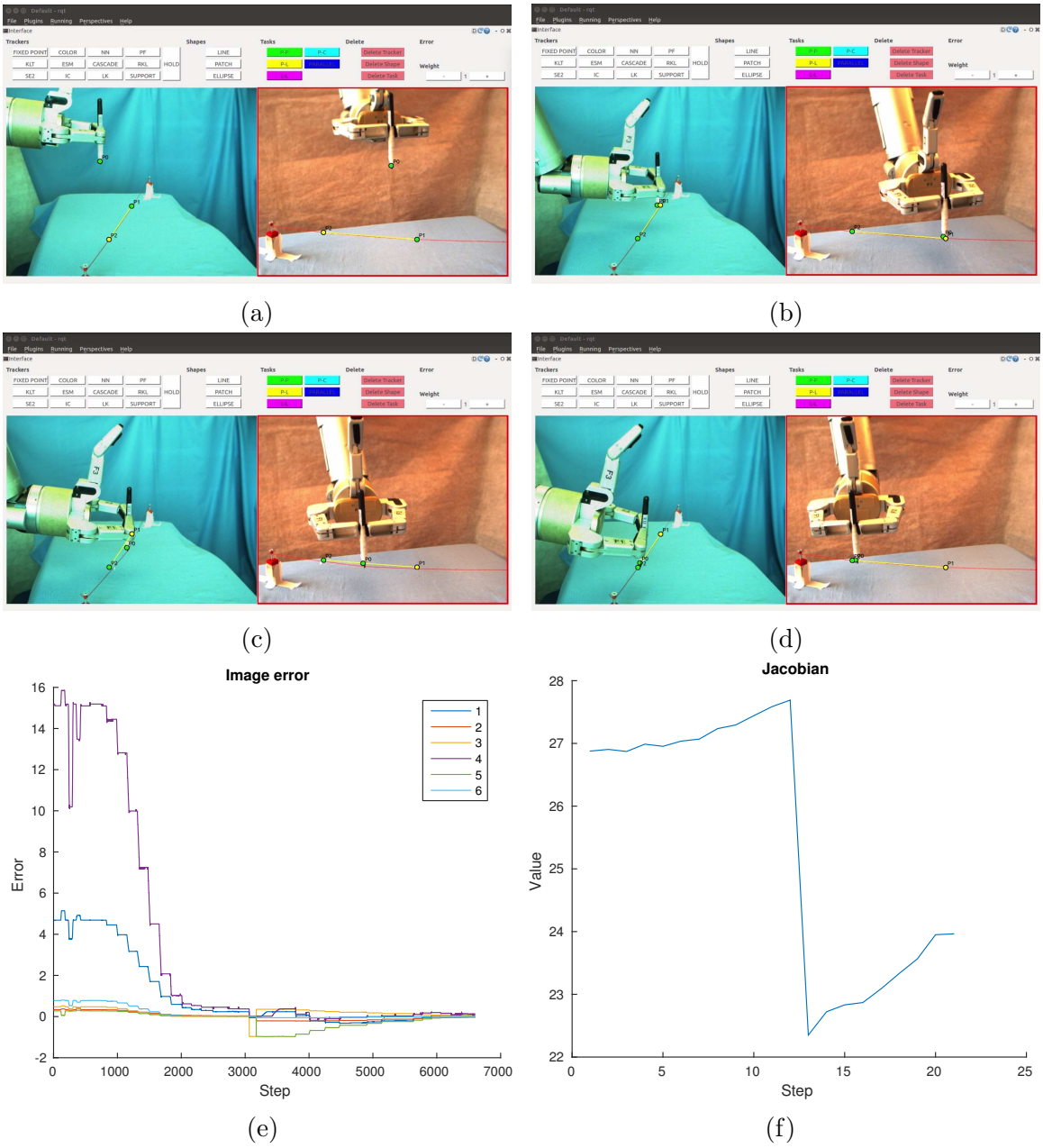


Figure B.23: (a) - (d) We complete a line following task by adding more weight to the point-to-line task. The task robot behaviour is the same as before. (e) The plot shows the image error. 1 and 4 represent the point-to-line errors in the first and second image, respectively. 3 and 4 belong to the point-to-point task error in the first image and 5 and 6 to the second image. We can see that the point-to-line error has increased weight, but this is diminished when the robot has converged to the line. (f) Jacobian conditioning number. We can note that the Jacobian conditioning number is 3-5 times higher in this implementation than for the others.

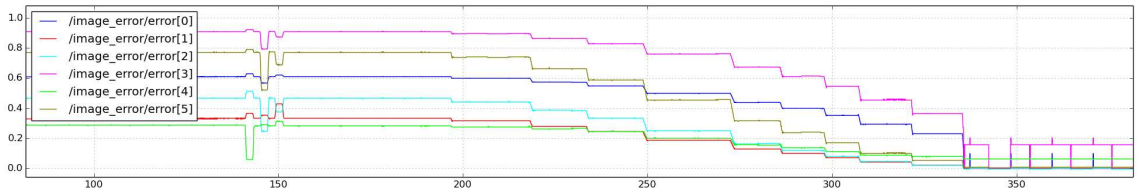


Figure B.24: We tried to change the point-to-line task by taking its cube root. The point-to-line errors correspond to /image/error[0] and /image/error[3]. We can see that the errors behave well until they get close to zero. Then they become unstable.

tracker trajectories in image space.

Next we try to modify the error function itself. We take the cube root of the point-to-line error to give it more similar behaviour to the point-to-line task. This works when the errors are large and we are approaching the first point on the line. However, as can be seen in Fig. B.24, once the point-to-line error approaches zero, it becomes very unstable. Even if the error were not to become unstable, the approach would still not solve our problem because the point-to-line error would be close to zero and the point-to-point error would grow as soon as we set the new goal point.

We need a solution that brings the point-to-point and point-to-line errors closer once the marker has approached the line. In our next approach we try to divide up the line in smaller segments of about 4 cm each, see Fig. B.25. This makes the point-to-point error smaller, but the previous behaviour remains. The marker's path between each pair of intermediate points is still curved, see Fig. B.21. We can imagine dividing up a long line when we only use point-to-point tasks for line following to make the movement somewhat more accurate, but it still does not solve our original problem of giving the point-to-line task significance. Indeed, when implementing intermediate points with only point-to-point tasks we get the same behaviour as when adding the point-to-line task, see Fig. B.26 and Fig. B.21. Moreover, in our example, using intermediate points turned out to be rather impractical since the points were fairly close together. The tasks did not finish to the end of the line due to loss of trackers. We added white markers on the line to define intermediate points, which ended up confusing the trackers.

Though we did not have time to implement it, we leave it as a suggestion for future work to divide the line following task into a primary and a secondary task, which is a known approach in visual servoing. Task specification, as we have defined

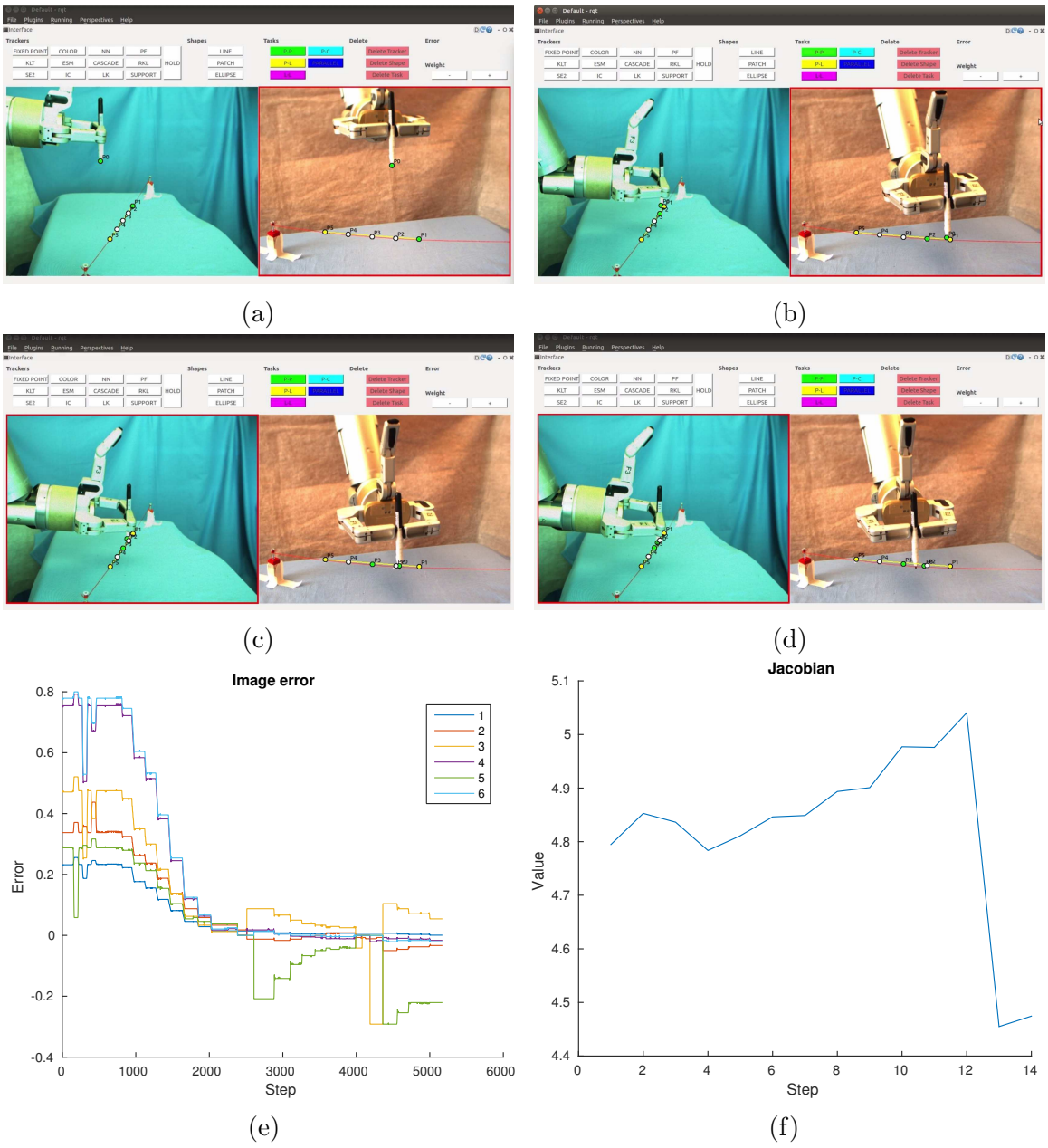


Figure B.25: (a) - (d) We complete a line following task by combining point-to-line and point-to-point tasks, but this time using intermediate points. The task is not completely finished because a tracker gets lost as the robot moves along the line. (d) shows the image error. 1 and 4 represent the point-to-line errors in the first and second image, respectively. 3 and 4 belong to the point-to-point task error in the first image and 5 and 6 to the second image. (f) Jacobian conditioning number.

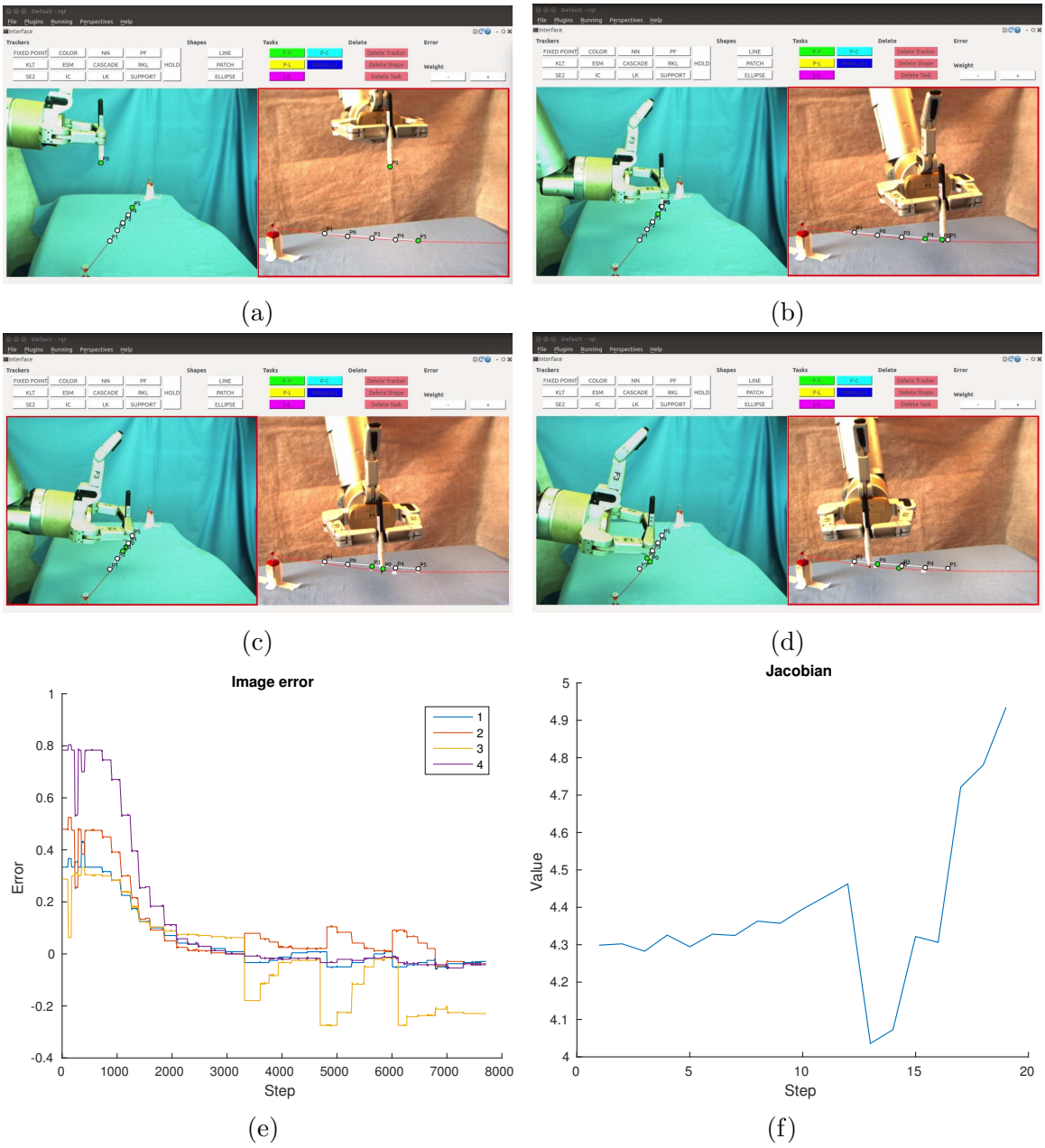


Figure B.26: (a) - (d) We complete a line following task using only point-to-point task and by setting intermediate points. The task is not completely finished as a tracker gets lost towards the end. (e) shows the image error. 1 and 2 belong to the point-to-point task error in the first image and 3 and 4 to the second image. (f) Jacobian conditioning number.

it so far, does visual servoing by minimizing a vector of stacked errors. The servoing therefore averages out the errors in the task. Hence, the weight of a task matters. The point-to-point tasks usually have the largest magnitude errors. They have two entries in the error vector, which again doubles their significance. Primary and secondary tasks allows us to assign importance to one task over another irrespective of error magnitude. In our case we could set the point-to-line task to be primary. The secondary point-to-point task will converge towards its goal, but only as long as it conforms with the constraints set by the primary task. This way the point-to-line task will control the line following motion despite its low weight.

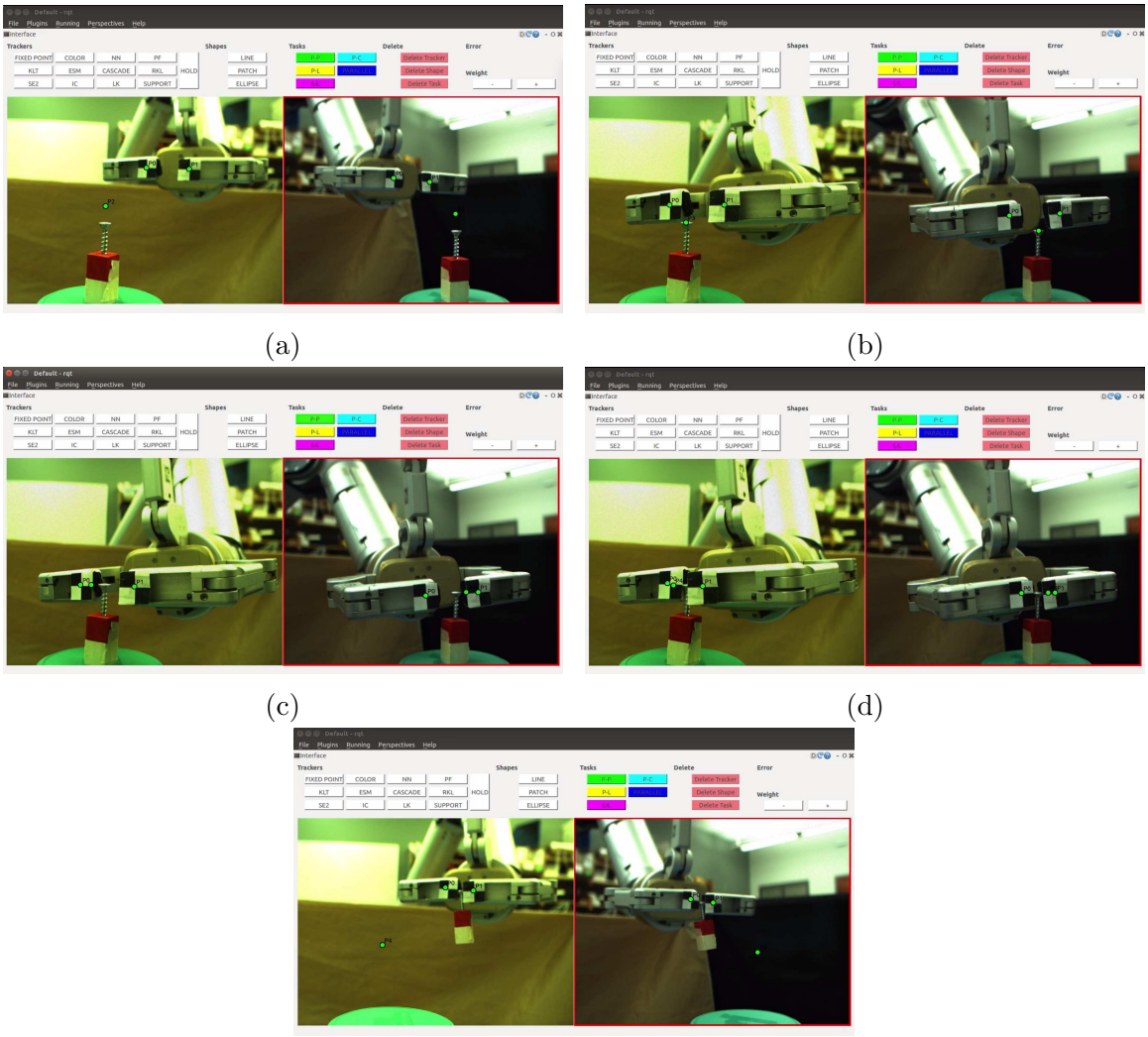
When specifying the line following with only point-to-point tasks, the user clicks 29 times in the interface and creates four tasks. Using point-to-point and point-to-line task brings the number to 41 clicks and eight tasks. If we include scaling the point-to-line task then the number of clicks become 81. For the intermediate point startegy it would depend on the number of intermediate points.

B.5 Grasp Screw

In this task we use the robot to grasp a screw, see Fig. B.27. The screw is 8.05 mm in diameter and 19.12 mm tall (above the wooden base). This task requires high precision as the screw is small. Furthermore, we need to make sure that the orientation of the gripper is correct for the grasp to succeed. Otherwise the screw could slip as we try to grasp it. In our set-up the possible camera views have some limitations. The cameras cannot be placed too far from each other as they both must see the front of the robot fingers. This task can also be done using eye-in-hand cameras, but we used eye-to-hand cameras in our experiment.

We present different approaches that we tested. First we completed the simpler version of this task, where the gripper and the screw were already aligned. Later we tilted the screw to change its orientation. Since in previous tasks we perturbed the orientation of the robot gripper, we decided to change the orientation of the object this time.

When the orientation of the gripper and the screw is aligned, we specify two point-to-point tasks and use joint 1, 2, 4 and 5 of the robot, see Fig. B.27. The point-to-point task translates the gripper to a free space point above the screw. Then we move the gripper downwards to the screw. Since we set a point on each robot finger, the robot centres its position around the screw. The robot starts the task fairly close

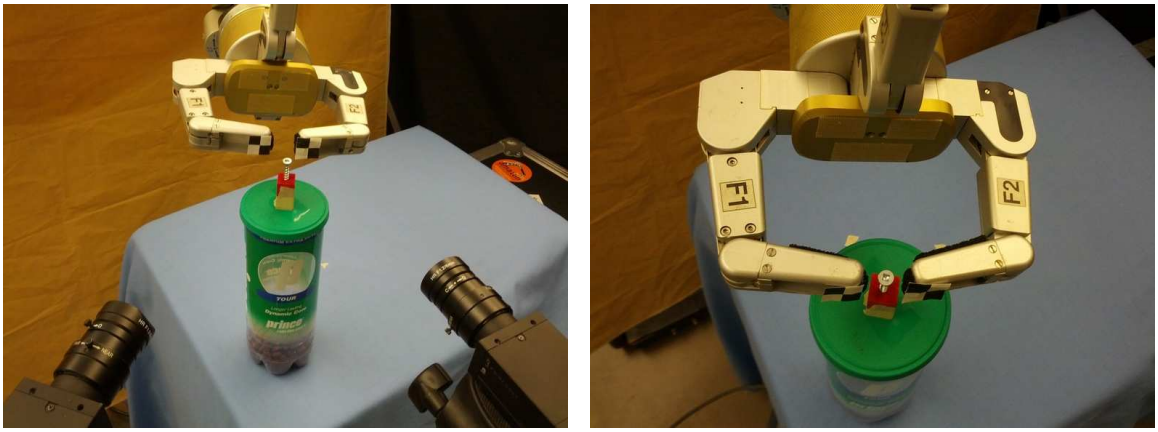


(a) (b)

(c) (d)

(e)

Figure B.27: We want to grasp a screw. The gripper's orientation is already aligned with the screw. (a) We specify two point-to-to-point tasks to center the gripper above the screw. (b) We move the goal point to move the gripper downwards towards the screw. (c) - (e) We do fine adjustments to the position of the gripper and finally we grasp the screw.



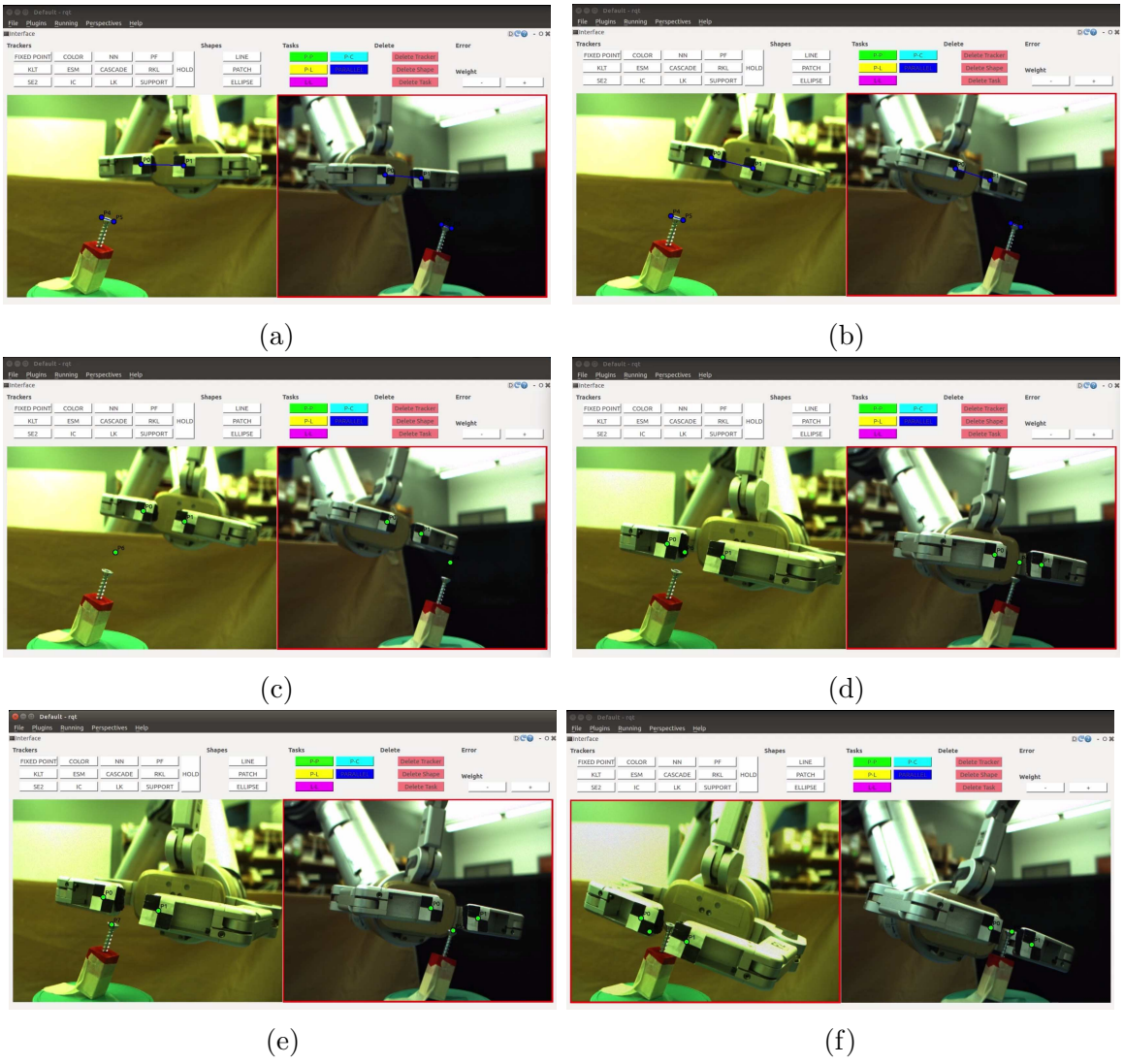
(a) (b)

Figure B.28: (a) shows where the gripper converges to with the point-to-point translation to the screw. Because of the camera view from the side, this does not correspond to the desired position for grasping. By making adjustments to the point-to-point task goal points we are able to properly centre the gripper as seen in (b).

to the screw and so we avoid any problems with other local minima. One example could be if one of the two point-to-point tasks converges to zero, while the gripper is at the side of the screw. Moving the gripper's initial position close to the screw can easily be done by using a point-to-point task for coarse positioning.

The robot gripper converges to its goal and centres on the screw, but because both of the cameras are located in front of the robot gripper, the fingers do not move far enough forward to actually grasp the screw. Therefore we move the point-to-point goal positions to create more forward movement, see Fig. B.28. Since the robot fingers close with a backward movement, predicting the best goal positioning is more difficult than for other tasks. The final placement in Fig. B.28, though it may seem intuitive for grasping, is still not entirely sufficient. The image shows that we can achieve desired accuracy for grasping in the general case. In our first experiment we were able to complete the grasp, see Fig. B.27. However, for the remaining experiments we focused on getting a good alignment, but not on trying to close the gripper as this adds movements that would be unnecessary if the gripper closed in a more intuitive way.

We move on to make the task more challenging by perturbing the orientation of the screw. In this case we first specified a parallel lines task to change the orientation of the gripper only using joint 7 before we changed to point-to-point tasks for translation. The translation was done in the same way as for our previous experiment. We were



(a) (b)

(c) (d)

(e) (f)

Figure B.29: We grasp the screw in the case where the orientation of the screw has been perturbed. (a), (b) We start by defining a parallel lines task to adjust the orientation of the gripper. (c), (d) We specify point-to-point tasks to translate the gripper to just above the screw. (e), (f) We move the gripper downwards to the screw and finally adjust the positioning to properly position the gripper for grasping.

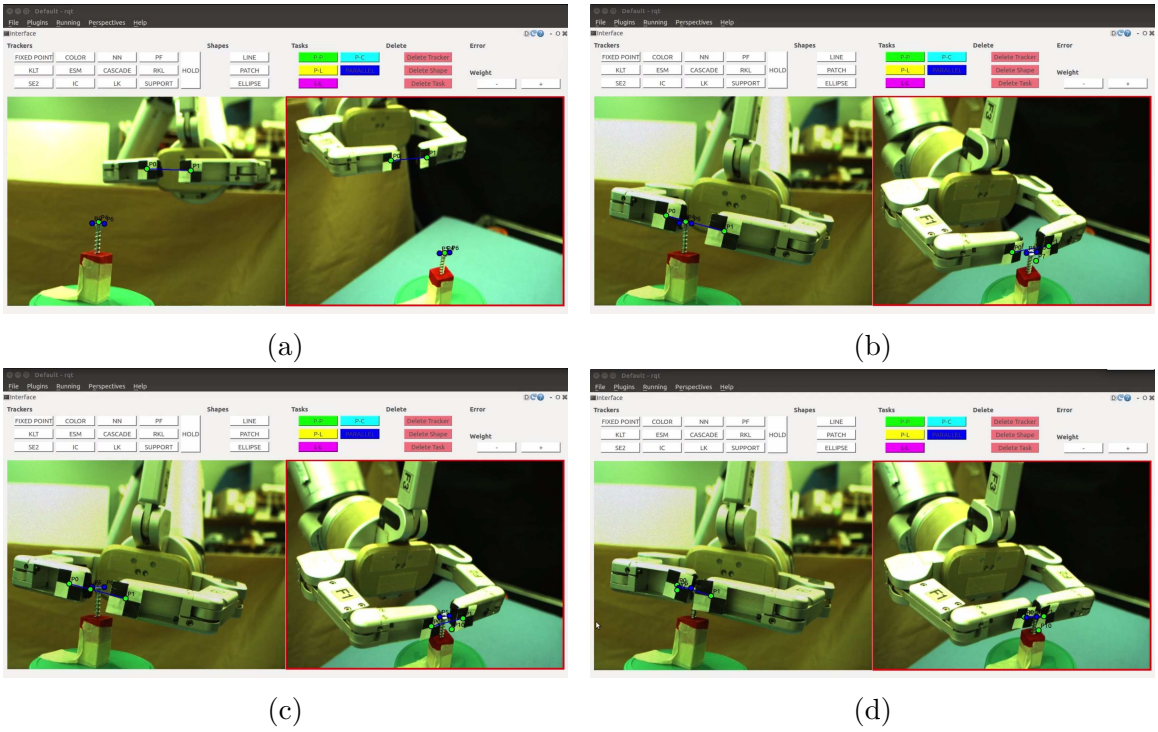


Figure B.30: We move one of the cameras to get a top down view. (a), (b) We translate the gripper as before towards the screw. Since we have a better view, we translate directly to the top of the screw instead of to a free space point above it. In order to maintain proper orientation we added a parallel lines task. (c), (d) Once the gripper has been translated to the screw we adjust its positioning to allow for grasping.

able to properly centre the gripper on the screw, see Fig. B.29.

Finally, we tried to adjust one of the cameras to get a top view. We wanted to find out whether forward positioning of the gripper would be more accurate and adjustments in position more intuitive for the user. One downside to having a top view was that the gripper's orientation would change while we approached the screw. Hence, we had to add a parallel lines task to the specification even tough the screw's orientation started out aligned with the gripper, see Fig. B.30. The gripper did get some more forward movement as a result of the new camera view, but adjusting the final position before grasping did not become easier.

The image error and Jacobian condition number plots for grasping the screw are displayed in Fig. B.31 and Fig. B.32, respectively. Notice that the image errors do not converge to zero once the gripper has centred on the screw. This is to be expected since we are not servoing the fingers to exactly touch the screw. We also see that

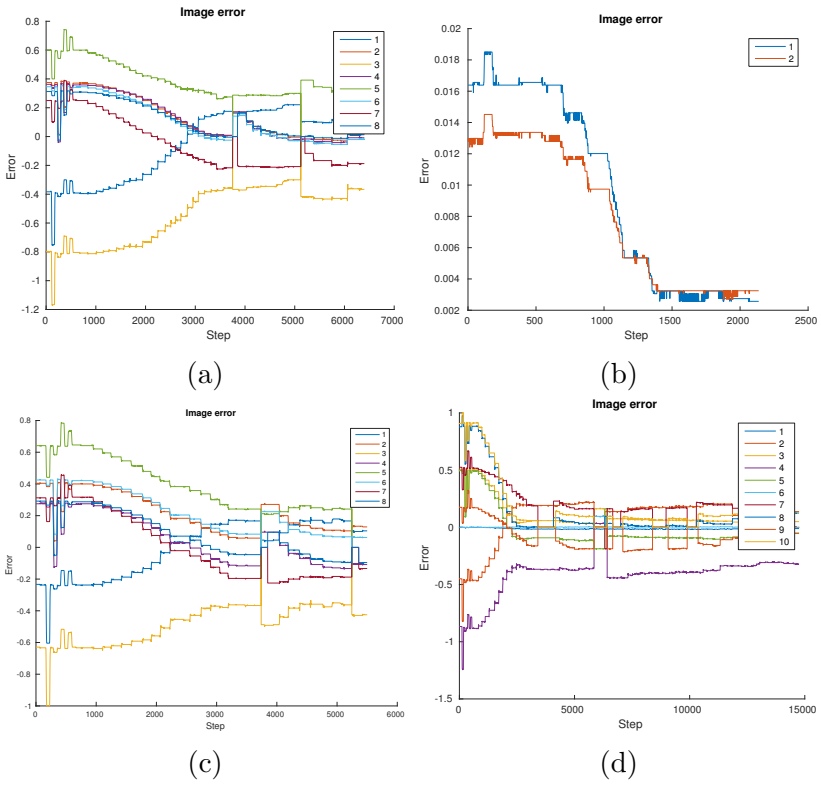


Figure B.31: (a) Image error for aligned gripper and screw orientation. The first and second halves of the numbered lines are point-to-point errors in the first and second images, respectively. (b), (c) The screw's orientation has been perturbed. (b) Errors of the parallel lines tasks to orient the gripper. (c) Point-to-point task errors. Again, each half of the errors belong to one image. (d) Image error for view change. 1 and 6 correspond to the parallel lines task, while the rest are point-to-point errors.

some error functions increase and decrease as we move the goal points to adjust the gripper's position by moving it forward.

In conclusion we are able to achieve high enough precision to position a robot gripper to grasp a screw. We are also able to do so when the orientations of the screw and the gripper are not initially the same. The user needs to click 69 times in the interface to specify the task with another 21 clicks added for each fine adjustment made. Similarly we create 10 task with another four added for each adjustment.

B.6 Cut Along Line

The final task replicates the action of cutting along a line, like for instance cutting a piece of cake or a slice of bread. The robot uses a tool, in this case a knife, that it

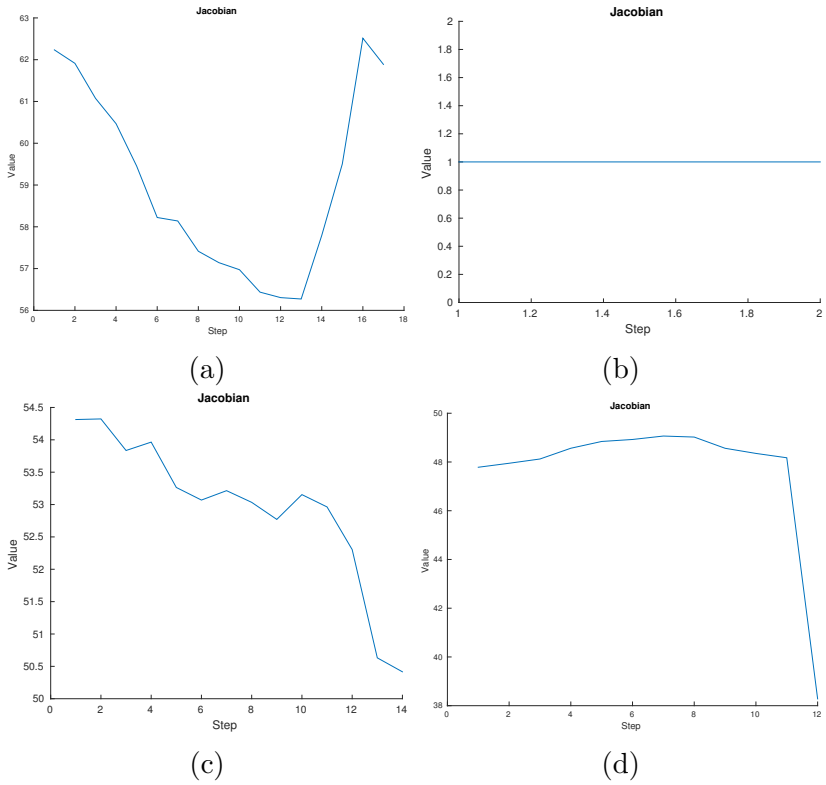


Figure B.32: (a) Jacobian conditioning number for aligned gripper and screw orientation. (b), (c) The screw's orientation has been perturbed. (b) Conditioning for the parallel lines specification to orient the gripper. (c) Point-to-point task translation. (d) Result for changed view.

holds in its grip. For systems that rely on 3D vision, using tools can be challenging. We cannot calculate the pose of a tool from forward kinematics alone. A model of the tool would be needed. In our case the robot can grasp any tool or object and we do not have to make any modifications to our procedure. We only need to make sure that we can track the tool.

In our experiment the robot holds a plastic knife. It is 18.5 cm long and measures 1.8 cm at the widest part of the blade. We have put markers on the knife to be able to track it. The knife is supposed to move down and position itself along a line on a piece of foam, see Fig. B.33. For cutting, the knife needs to be correctly oriented and it must also move to the correct point along that line. There are several options for how to accomplish this task. We have tested three different ways of specifying the task, see Fig. B.33. The first combines a line-to-line task for controlling the orientation of the knife with a point-to-point task for driving the robot to the correct position along the

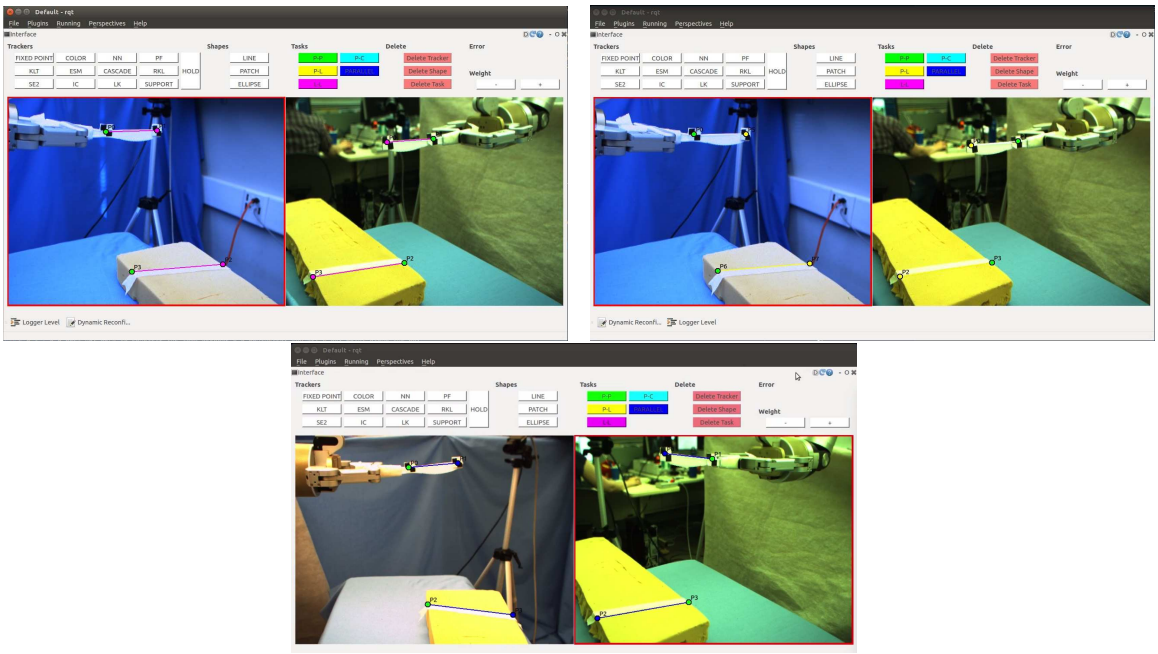


Figure B.33: The figure shows the task we want to accomplish, namely cutting along a line, with three possible task specifications. They combine a line-to-line, two point-to-line and a parallel lines task with a point-to-point task, respectively.

line. The second specification combines two point-to-line tasks with a point-to-point task. The final specification consist of a parallel lines task and a point-to-point task. Since we have used parallel lines and point-to-point tasks in other experiments, we decide to focus on the point-to-line and line-to-line tasks in this experiment. We have included data for one example of the parallel lines and point-to-point combination at the end.

A line-to-line task can be implemented in several ways. It can be created by stacking the errors of two point-to-line task or by stacking the errors of a parallel lines task and a point-to-point task. In our system we have chosen to sum the errors of two point-to-line tasks. Hence, the cutting task will give us a chance to compare the behaviour of our line-to-line implementation and the combination of two point-to-line tasks.

We started our experiments by exploring different combinations of robot DOF and task specifications. The cameras were at opposite sides of the task, the views about 180 degrees apart. We first tried completing the line-to-line and point-to-point task specification as well as the point-to-line and point-to-point specification using joints 1, 2, 4 and 5 of the robot and joints 1, 2, 4, 5 and 6. In both cases the task failed

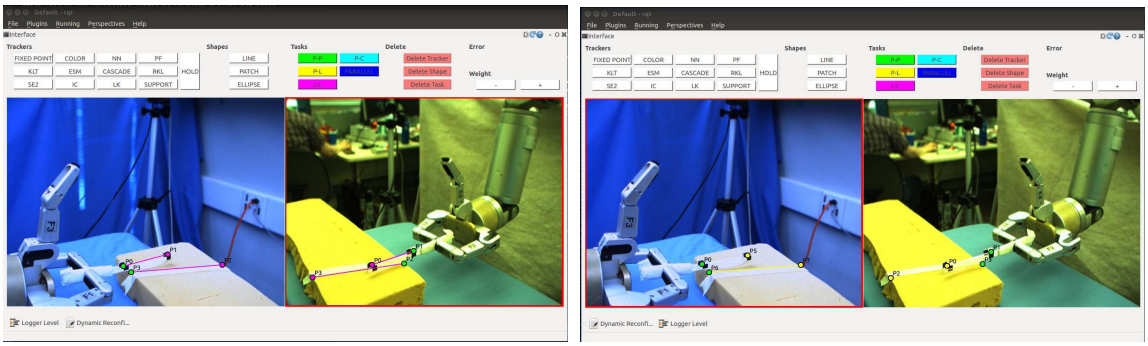


Figure B.34: The pictures show the result of completing the cutting task by combining a line-to-task and two point-to-line tasks with a point-to-point task, respectively. The robot uses joint 1, 2, 4, 5 and 6. The task fails as the orientation is not sufficiently adjusted. The task fails similarly for the case where we use joints 1, 2, 4 and 5.

as the point-to-point task dominated and orientation was not sufficiently aligned, see Fig. B.34.

We then thought of ways to mitigate the challenge of combining different errors and decided to try scaling the errors and to use an intermediate point. We first tested both methods (line-to-line and point-to-line) with an intermediate point using joints 1, 2, 4 and 5. As can be seen in Fig. B.35, we set a fixed point half way between the knife and the goal point. Then we specify the tasks as before, but set the intermediate point to be the goal point for the translation. We hope that the decreased error in the point-to-point task will make it less dominant. Moreover, if the task is still dominant, the robot might adjust its orientation once it gets close to the intermediate point. This would hold for the real goal point as well, but it is often too late to adjust orientation once the robot is close to the goal.

Setting an intermediate point works when we use joints 1, 2, 4 and 5 and we can accomplish the task. The orientation of the knife gets properly aligned before we reach the goal. This technique failed when working with joint 1, 2, 4, 5 and 6, see Fig. B.35. The tip of the knife starts moving downwards. This seems natural as we are using point-to-line and line-to-line tasks. They are not only adjusting orientation, like a parallel lines task, but also trying to move to the goal line. Hence using an intermediate point or splitting a task into sequences, as we did with the marker in cap task, will not always work in the case of line-to-line and point-to-line tasks.

Next we try to accomplish the cutting task by weighing the task errors. We multiply the line-to-line and point-to-line errors by a factor of five. We accomplish the task successfully both in the case of using joints 1, 2, 4 and 5 and when using

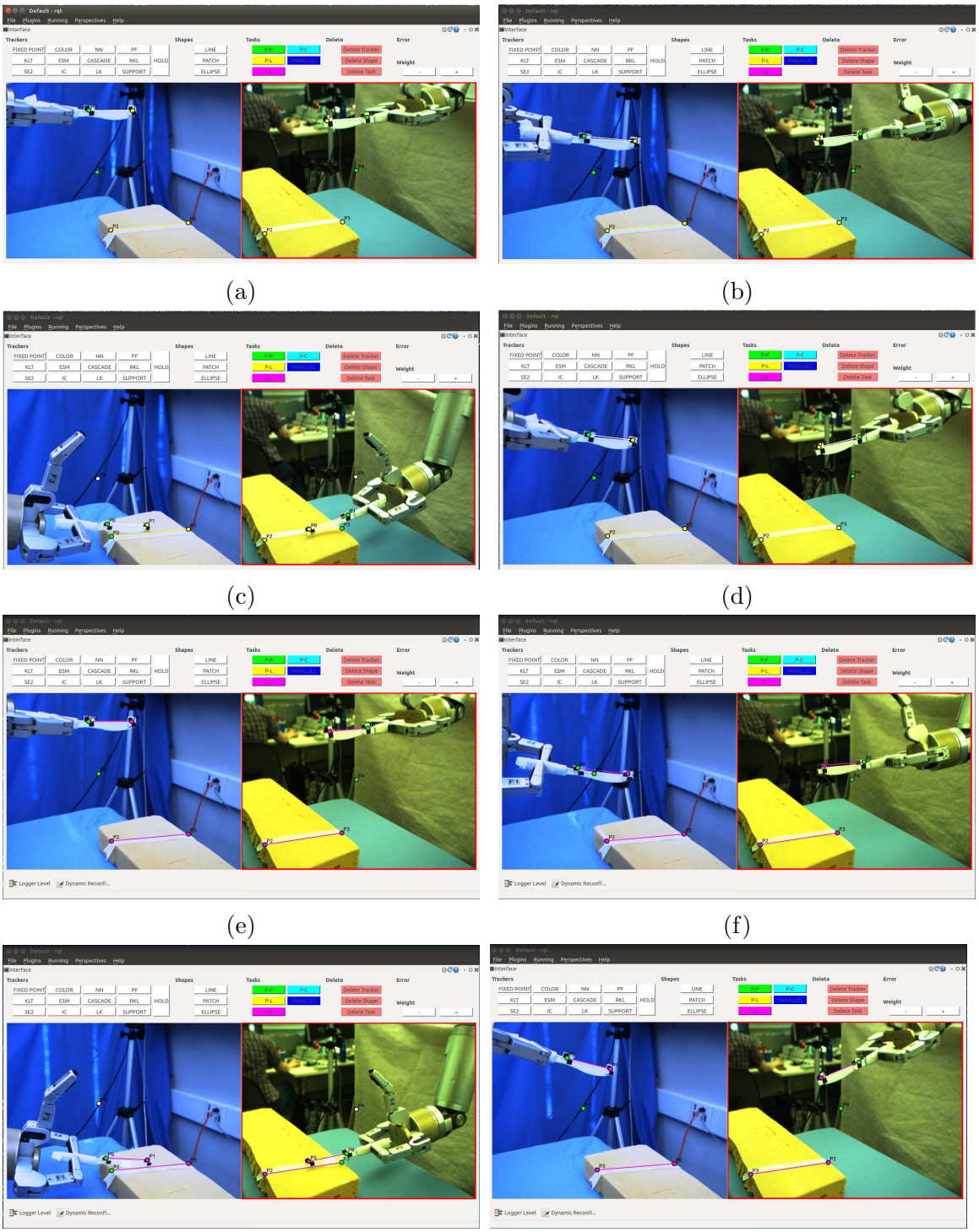


Figure B.35: Experiments using an intermediate point. (a) Specification in the point-to-line case. (b), (c) Results using joints 1, 2, 4 and 5. (d) Result including joint 6. The knife starts pointing downwards. The corresponding results for the line-to-line case are displayed in the same sequence in (e)-(h).

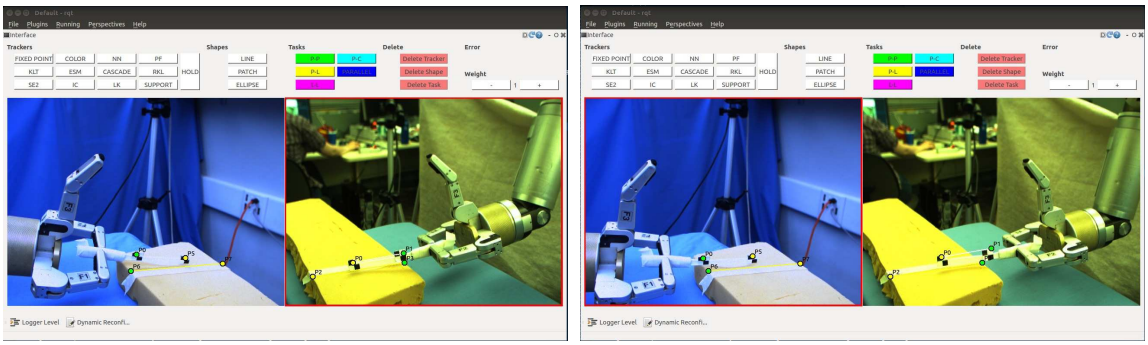


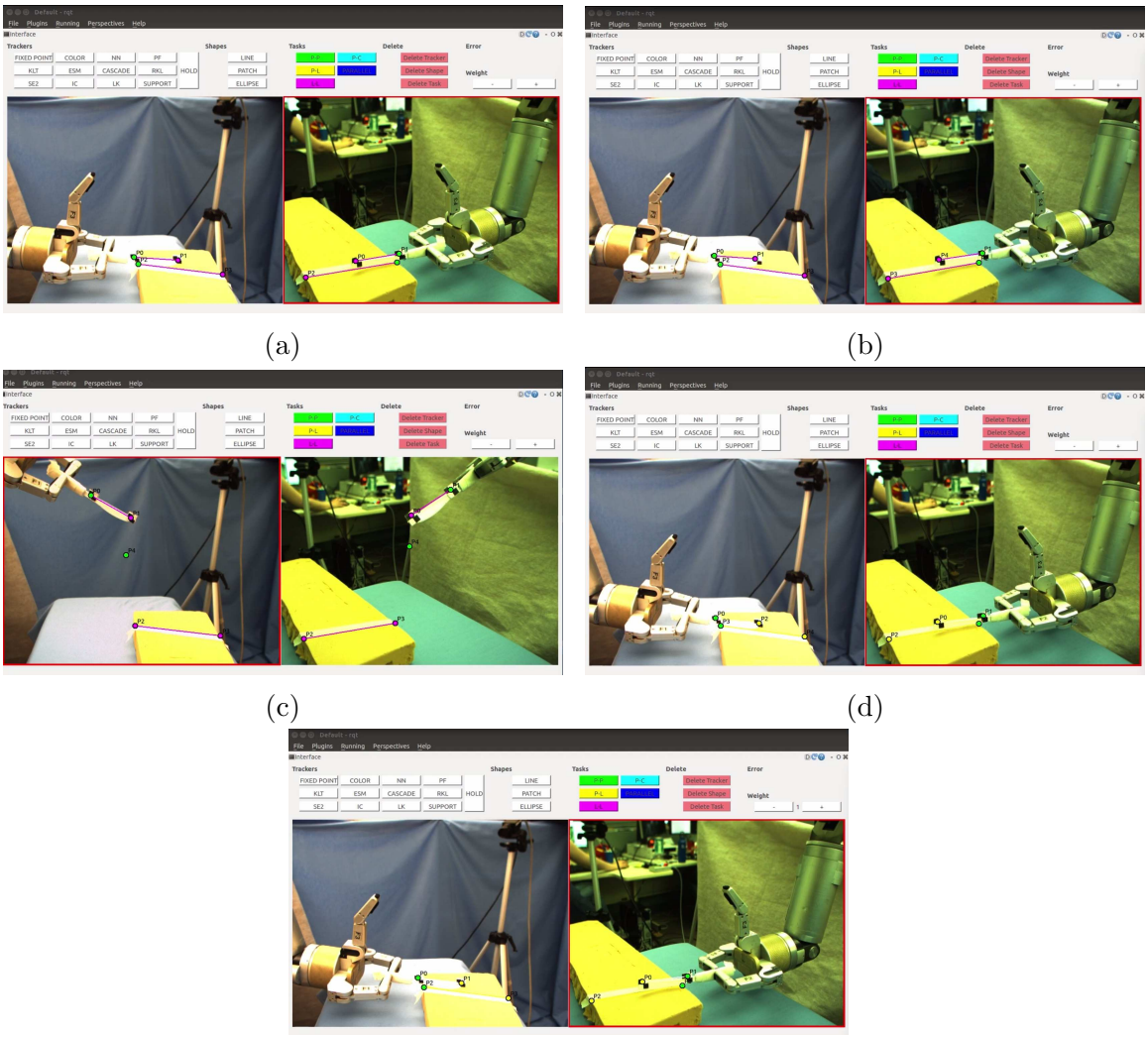
Figure B.36: The pictures show the results of executing the task with scaling for the point-to-line and point-to-point task combination using joints 1, 2, 4 and 5 as well as 1, 2, 4, 5 and 6, respectively. Scaling with the line-to-line method also gives similar performance.

joints 1, 2, 4, 5 and 6, see Fig. B.36.

Through our initial investigation we found that scaling the errors is a good alternative when we cannot split the task into separate segments for orientation and translation. Setting an intermediate point worked when we excluded joint 6. This is however not a useful situation when we want to be able to control the pitch of the knife. We proceeded to change the camera view before collecting data from some experiments. The camera view is important to task completion and we wanted to see if we could get different results with a different view. Now the camera views were placed about 90 degrees apart.

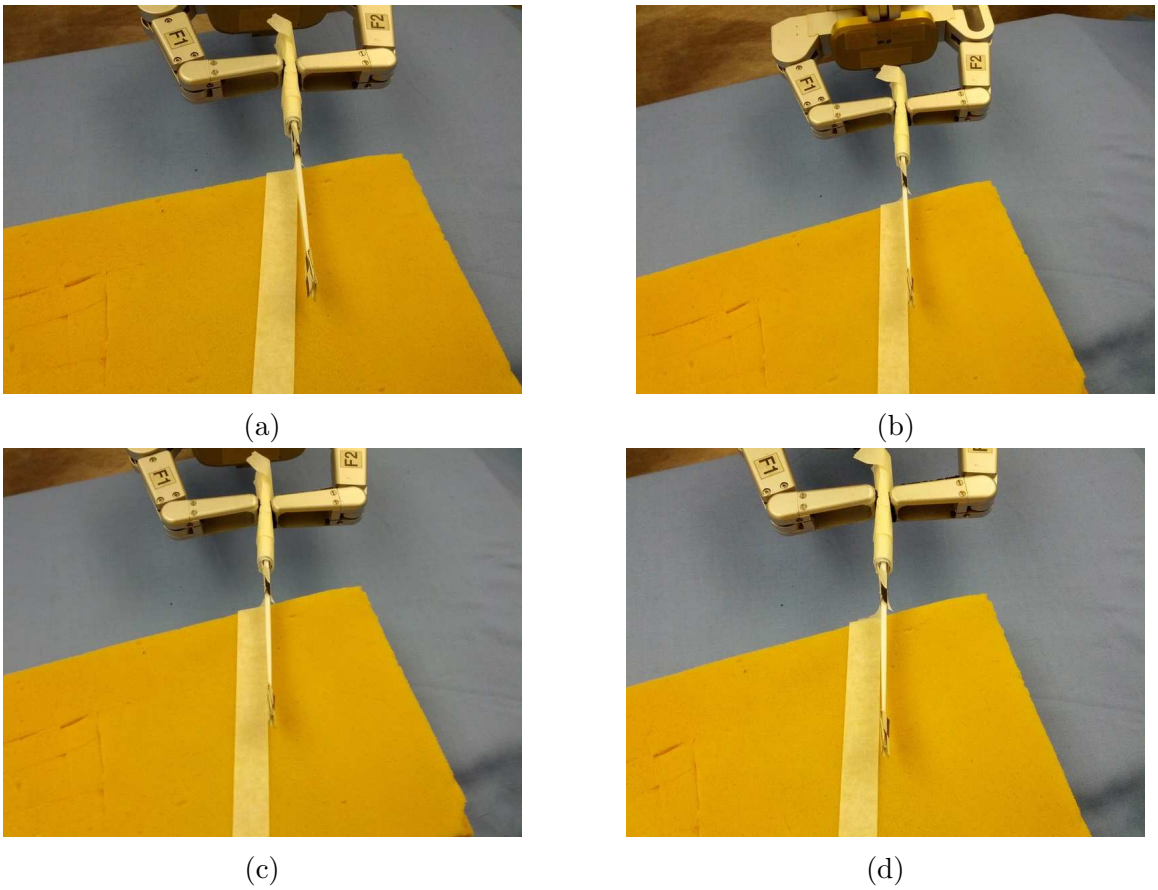
We collected data from a line-to-line and point-to-point task specification as well as a point-to-line and point-to-point task specification without any scaling or intermediate points. The line-to-line specification failed as before but the point-to-line specification succeeded. The change in view was positive. The resulting interface view can be seen in Fig. B.37 and for a better view to judge the task performance see Fig. B.38. The corresponding image error and Jacobian condition number can be found in Fig. B.39 and Fig. B.40. We also tested using an intermediate point for the line-to-line task with five robot joints. We can see it gives divergent image errors and a growing condition number. We continued by running both methods using scaling. As before, both line-to-line and point-to-line responded well to scaling.

Finally, we also collected data form one experiment using a parallel lines task combined with a point-to-point task, see Fig. B.41. For the cutting task this technique succeed without need for weighing, intermediate points or dividing the task into separate parts.



(a) (b) (c) (d) (e)

Figure B.37: The pictures show the results of running the line-to-line method with no modification (a), with scaling (b) and with a intermediate point (c), all using five joints. The first case does not orient the knife well enough and the intermediate point technique diverges. The scaling works well. The last pictures show the point-to-line case without (d) and with scaling (e). This succeeds in both cases.

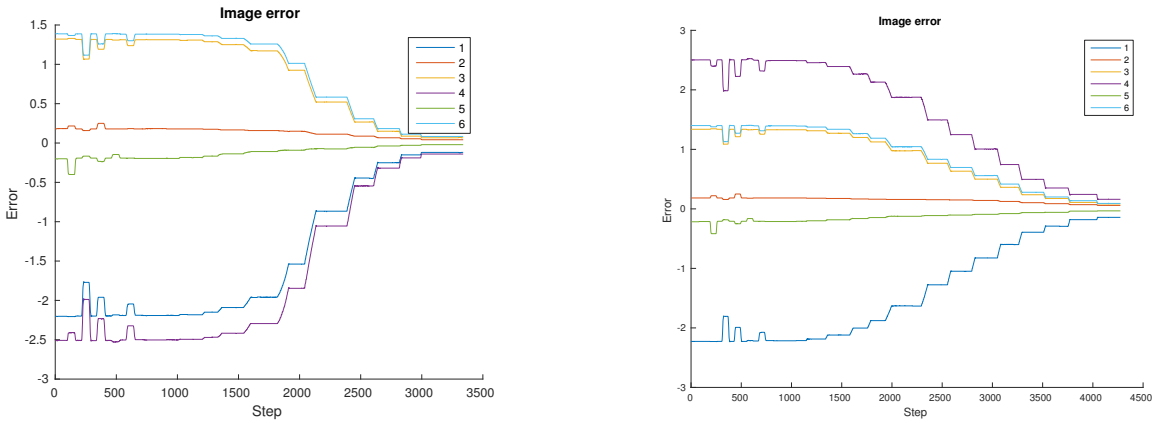


(a) (b)

(c) (d)

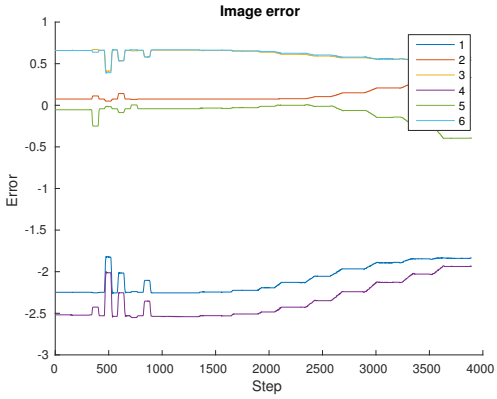
Figure B.38: These pictures show the results of the experiments from a different camera view so we can more easily interpret the difference in performance. We see the line-to-line method without (a) and with (b) scaling. We can see that scaling allows for better orientation. The last two images show the point-to-line method without (c) and with (d) scaling.

In conclusion we were able to solve the cutting task by using a combination of a line-to-line task with a point-to-point task and a combination of two point-to-line tasks and a point-to-point task and scaling the errors. We were able to complete the task with the second method without scaling, whereas the line-to-line approach needed scaling to properly adjust the knife's orientation. Because we are tracking the top of the knife instead of the bottom, the knife hit the foam before the point-to-line, line-to-line and point-to-point tasks fully converged. It might be the case, that with a good camera view and better tracking, that both methods would converge without the need for any scaling. However, using scaling is a more general solution as we know it gives a good result in more cases. We also see that the point-to-line

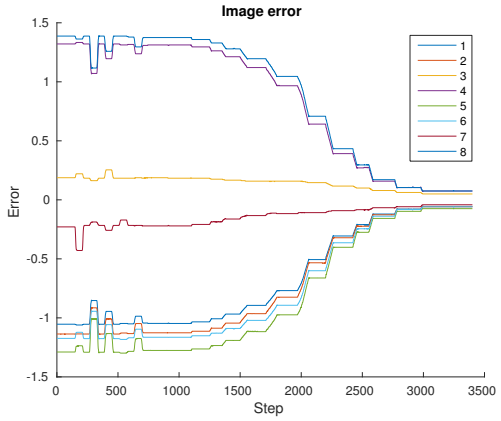


(a)

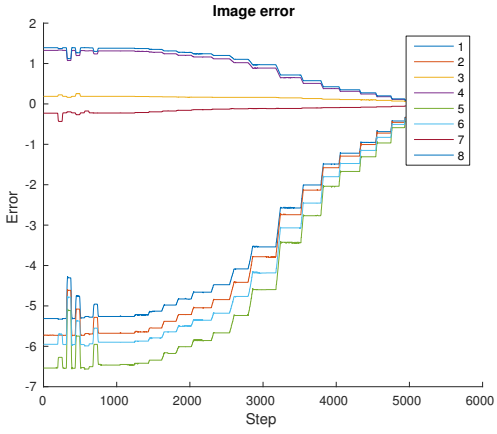
(b)



(c)



(d)



(e)

Figure B.39: (a) - (c) Line-to-line method with no scaling, with scaling and using an intermediate point. 1 and 4 are the line-to-line errors in the first and second image. 2 and 3 are the point-to-point errors in the first image and 5 and 6 in the second image. (d), (e) Point-to-line method without and with scaling. 1 and 2 are the point-to-line error in the first image and 5 and 6 in the second image. 4 and 5 are the point-to-point errors in the first image and 7 and 8 in the second image. The task diverges for the intermediate point cases.

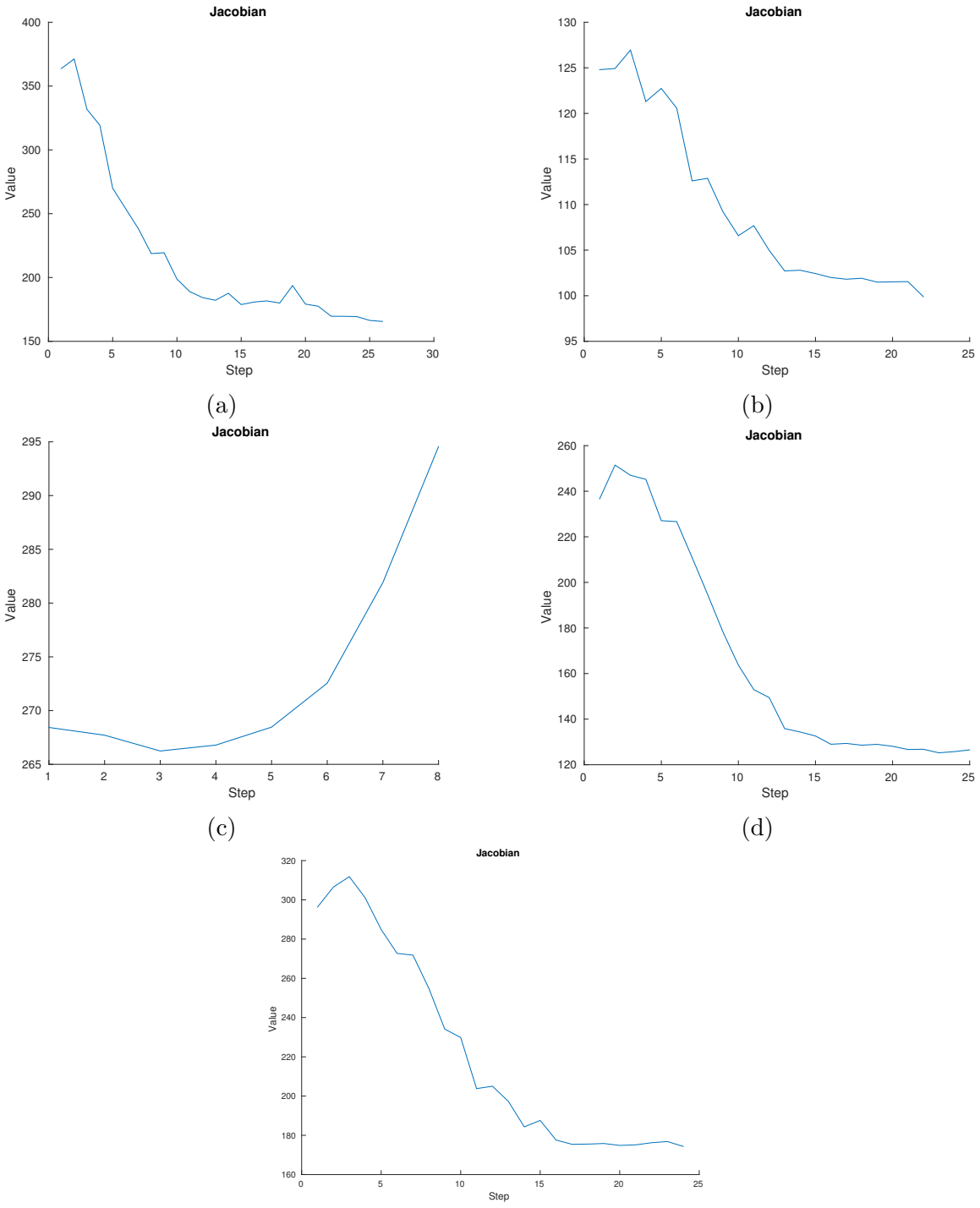


Figure B.40: (a) - (c) show the Jacobian condition number for the line-to-line method with no scaling, with scaling and using an intermediate point. (d), (e) show the point-to-line method without and with scaling. The task that uses an intermediate point diverges and we can see that the condition number increases.

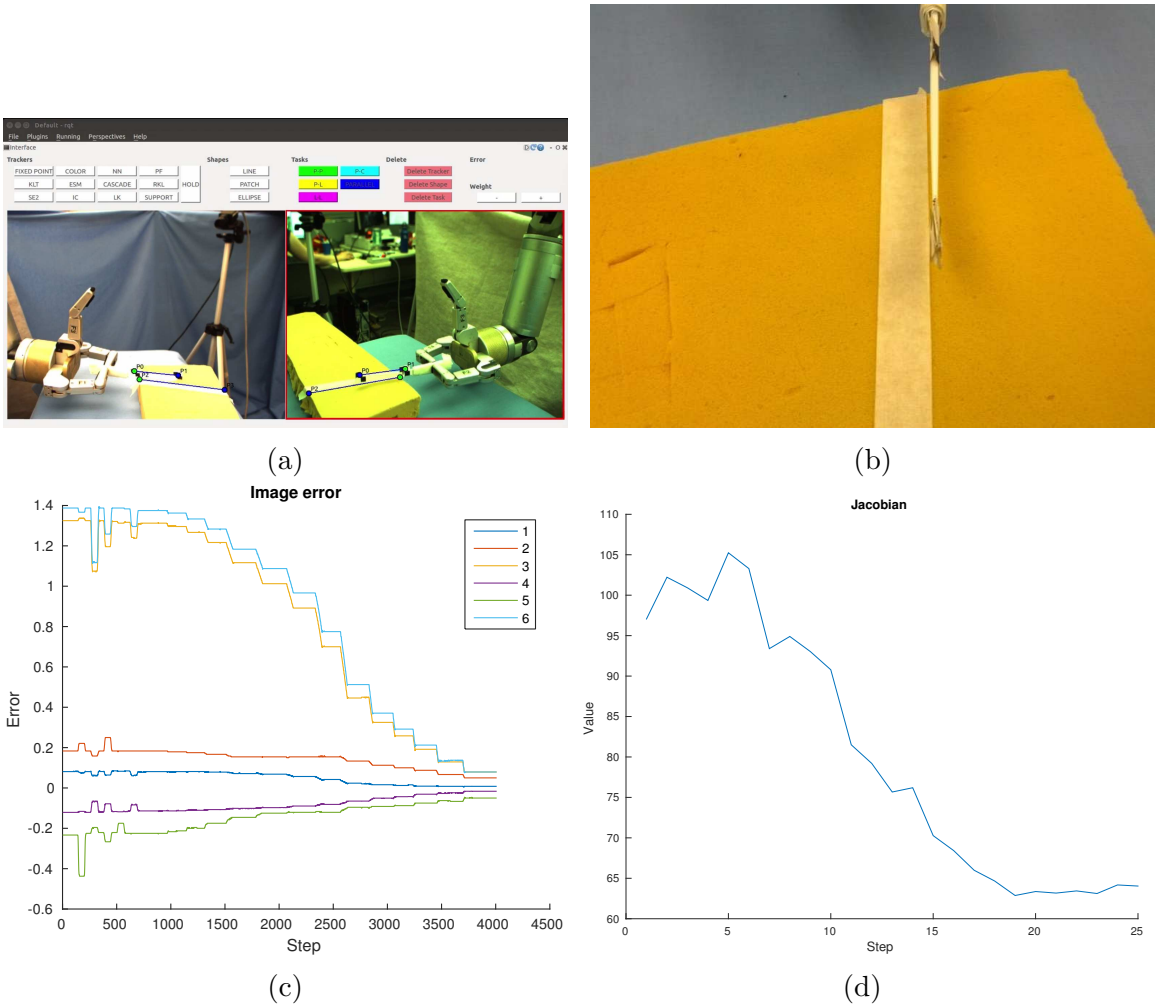


Figure B.41: (a), (b) show the results of using a parallel lines task together with a point-to-point task to execute the cutting task. In the image error plot (c) 1 and 4 represent the parallel lines task errors in the first and second image respectively. 2 and 3 are the point-to-point task error in the first image and 5 and 6 are in the second image. (d) Jacobian conditioning number.

and line-to-line tasks have a similar behaviour. This is natural because the visual servoing will minimize the whole error vector, which means it will average between all the vector entries. Therefore there is not much difference between summing or stacking point-to-line errors to achieve line-to-line specification.

For the basic line-to-line task the user needs to click 29 times in the interface to specify the task, which creates four geometric constraints. For the basic point-to-line approach the numbers are 32 clicks and six tasks. When we add scaling or intermedi-

ate points the number of clicks for the line-to-line and point-to-line approaches are 39 and 42 respectively. When we add the intermediate point the number of task become six and eight. Finally, for the parallel lines approach the number of clicks is 29 and the number of tasks is four.

Appendix C

Accuracy Experiments

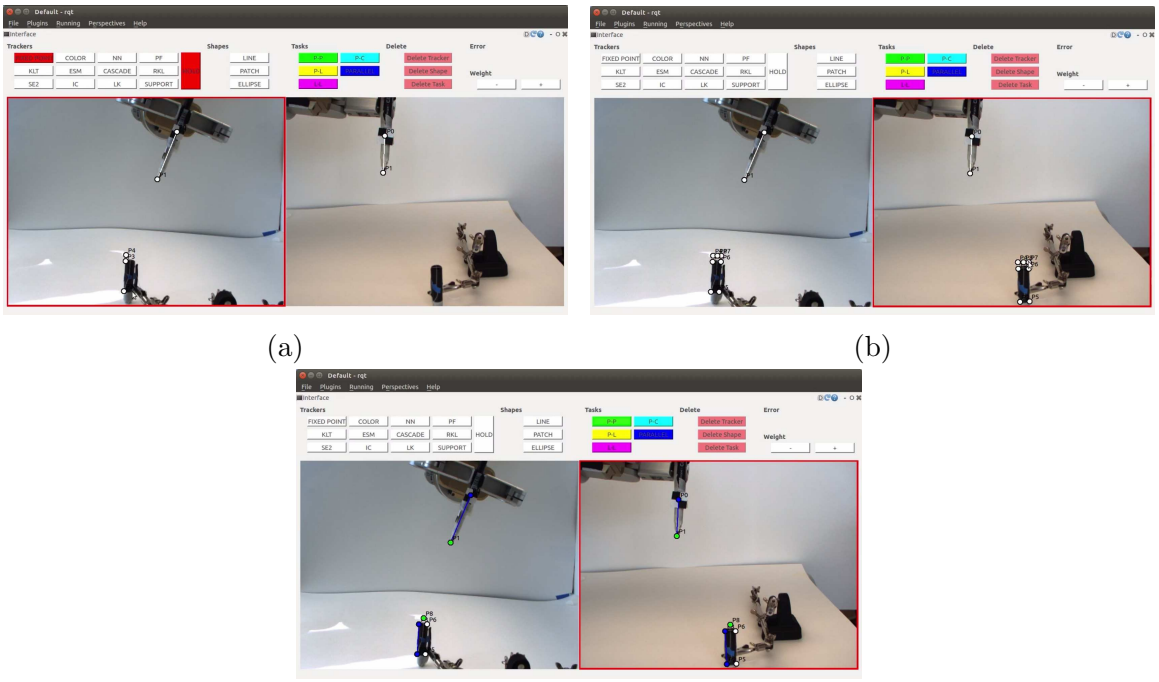
Though most of the experiments in this thesis aim to explore how to use visual specification to complete tasks, we also include two experiments to measure performance. First we complete an experiment to measure the accuracy of specifying a task using visual specification with our interface. Second we complete a repeatability experiment for basic visual servoing with our robot arm. The second experiment will provide some context to the results of the first.

C.1 Task Specification Accuracy

This experiment shows how accurately we can accomplish an example task that requires us to combine more than one type of visual task and that also requires user input. We complete the positioning for the marker in cap task ten times to see how often we can succeed and how accurately we can position the marker in the table plane above the cap before insertion. The cameras are placed 31 cm and 37 cm from the cap. The cap has a diameter of 9.9 mm, while the marker tip's diameter is 2.5 mm and the diameter of the marker (including the added tape) is about 9.5 mm. The set-up and the task can be seen in Fig. C.1.

The user specifies a parallel lines task to align the orientation of the marker with that of the cap. She also adds a point-to-point task to translate the pen to the cap, see Fig. C.2. Since we are not able to track the exact tip of the marker, we cannot put the goal point on the cap itself as the marker will then collide with the cap. Instead we want a goal point slightly above the cap. During our initial work to complete the marker in cap task, we chose goal points by setting free space points. This works because we can make fine adjustments to the marker's location. However, for this

Figure C.1: The image shows the task and the set-up.



(a) (b)

(c)

Figure C.2: (a) The user has set two points on the cap and used these to extend a third point above the cap. (b) The user has created an extended point on both markers and then calculated the midpoint between them. (c) We see the task specification (the extended points have been deleted to improve the view).

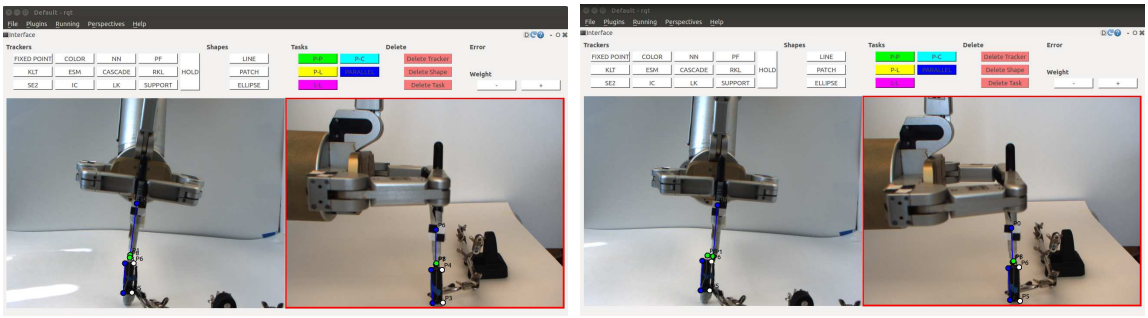


Figure C.3: The images show the finished task for the experiments with the best (1.12 mm error) and worst (9.63 mm error) result respectively.

Figure C.4: The images shows the marker after servoing. (a) Front view of on of the poorer result. (b) Front view of a better result. (c) Side view of the better result. We can see that one side as more error than the other.

experiment we want a more consistent and precise goal point. We accomplish this by an affine approximation. We mark two lines, each at the side of marker, which are two occluding contours. We then extend these to lines by a third point, placed at distance 0.3 times the line length away from the top end-point. Then we find the midpoint between these two extended points. We can calculate this mid point simply by finding the midpoint between the image coordinates (usually one would need to use projective invariants).

Once the specification is done, we run the visual servoing for 40 seconds. That leaves enough time for the robot to converge. We measure the marker's distance from the centre of the cap in the table plane. We mark the centre by crossing two pieces of thin sewing thread and measure the distance with a calliper. Since we measure by hand this naturally affects the resulting accuracy, but we find that the collected errors still provide useful data for a real world task. The trackers and tasks are re-set on each iteration.

Our experiment succeeded in all ten cases, meaning that the task did not diverge

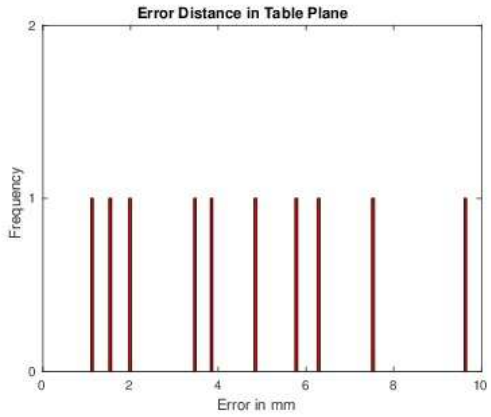
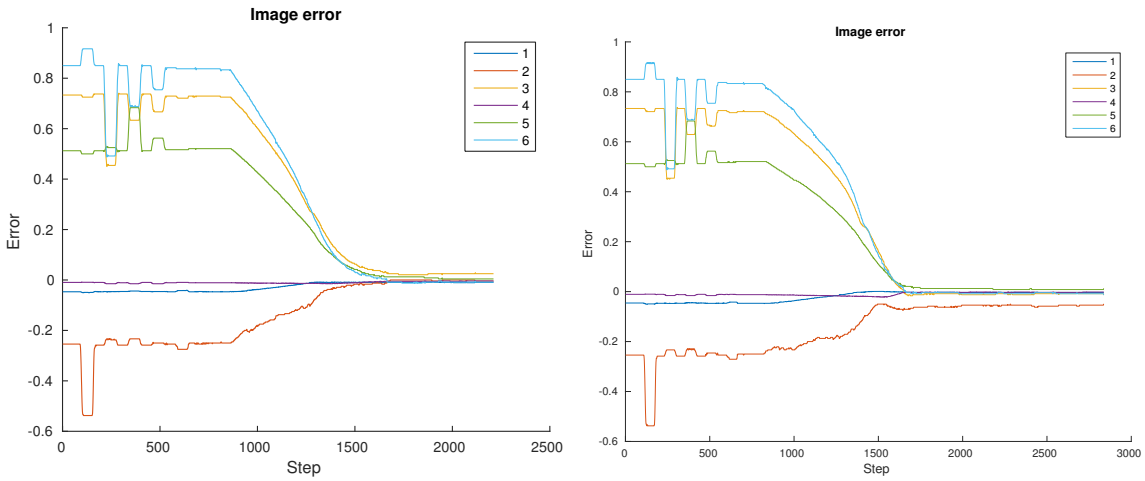


Figure C.5: A plot of the error from the centre of the cap to the marker tip in the table plane. The x axis show the range of the errors in mm. A bar is displayed at each recorded error.

and we did not have to stop short because a tracker got lost. The average error was 4.6 mm with standard deviation equal to 2.7 mm and 7.5 mm variance. A plot of the resulting distance for each experiment can be seen in Fig. C.5 and pictures of two experiments can be seen in Fig. C.3. In all the experiments the marker's location was well positioned in one direction of the table plane, whereas the error was larger in the other, see Fig. C.4. Hence, most of the error was in one direction. In the plot of the image errors of the best and worst experiments in Fig. C.6 we see that the biggest difference between these two tasks lie in the point-to-point task's x direction in the left-most image. For the good experiment this gets close to zero, whereas for the bad experiment it remains a bit bigger. This corresponds to the observation of the marker's error being less accurate in the direction that coincides with the sideways movement in the first image.

When inserting the the marker into the cap, the marker does not need to be perfectly centred on the cap. Given the size of the marker, it must be 3.0 mm or closer to the centre. In 3 out of out 10 experiments the cap could have been inserted without any further adjustments. The user clicked an average of 58.6 times in the interface to set up the task.

When positioning the marker above the cap we have 4.6 mm of error in the table plane. What causes this error? As we mentioned earlier, the goal point that we calculate for the point-to-point task is an affine approximation and hence not exactly correct. We rely on the user to click on points in the image. In particular, the user



(a) (b)

Figure C.6: In the image error plots in (a) and (b) 1 and 3 correspond to the parallel lines and point-to-point errors in the first image. 2, 3, and 4 are the same for the second image. (a) shows the data for the best experiment and (b) for the worst. The biggest difference lies in error 2, which is the image x coordinate for the first (left-most) image in the interface. This corresponds to our results where the marker is mostly misplaced in this direction.

marks fixed points on the cap. These point locations will have errors. To track the points higher on the marker we use tracking markers, whose placement will also have some error. Whenever the user starts a point tracker, the interface will not track exactly the point that the user chose, but instead find the best point to track within a small window. We want to track the tip of the marker. This is not possible and instead the tracker sits above the tip. Moreover, the marker tip has width of about 2.5 mm and the tracker might not centre exactly in the middle. As we can see in Fig. C.3, the trackers also drift throughout the visual servoing.

C.2 Visual Servoing Repeatability

In the second experiment we test the repeatability of 6 DOF positioning using visual servoing with our robot arm. We complete an experiment that is similar to those presented in [41] and [29]. [41] explains that repeatability tests the robot's ability to go back to a pose it has previously attained, while accuracy tests the robot's ability to achieve any specified pose. In the accuracy experiment and when otherwise using our interface, we find that the system's performance is affected by a user's ability to

(a) (b)

Figure C.7: (a) Overview of the set-up with two cameras, a dial meter and the robot. (b) The box pushes on the dial meter in the direction of motion to collect measurements.

set correct visual goal points as well as tracking. In the repeatability experiment we test the performance of visual servoing itself without being affected by user input. We also try to improve the tracking. This will provide a baseline for how well the visual servoing works and therefore serves as a good comparison with the accuracy results of the previous experiment. In particular, we will gain insight into the performance of combining different visual tasks and having the user click on goal points versus completing basic visual servoing.

We complete two experiments. One experiment measures the accuracy in the depth direction of the cameras and the other measures the accuracy in the sideways direction. Since the cameras are not parallel, we choose the depth and sideways directions to be the middle ground between each camera view's depth and sideways direction. We measure the accuracy by servoing the end-effector towards a 0.001" dial meter that is set up in the direction we want to measure.

In this experiment the robot gripper holds a box, see Fig. C.7. The box gives us sufficient surface to place LEDs for tracking. Furthermore, the surface of the box will provide a plane that can push on the dial meter and make the measuring easier. We place a total of seven LEDs on the box. Each camera can see five of those LEDs. We place the LEDs on different planes of the box to avoid ambiguous task specification. The cameras are placed about 90 degrees and 103 cm apart, distanced 57 cm and 65 cm from the dial meter, respectively. The base of the robot is 90 cm from the dial meter. One pixel in the camera views correspond to approximately 0.94 mm robot

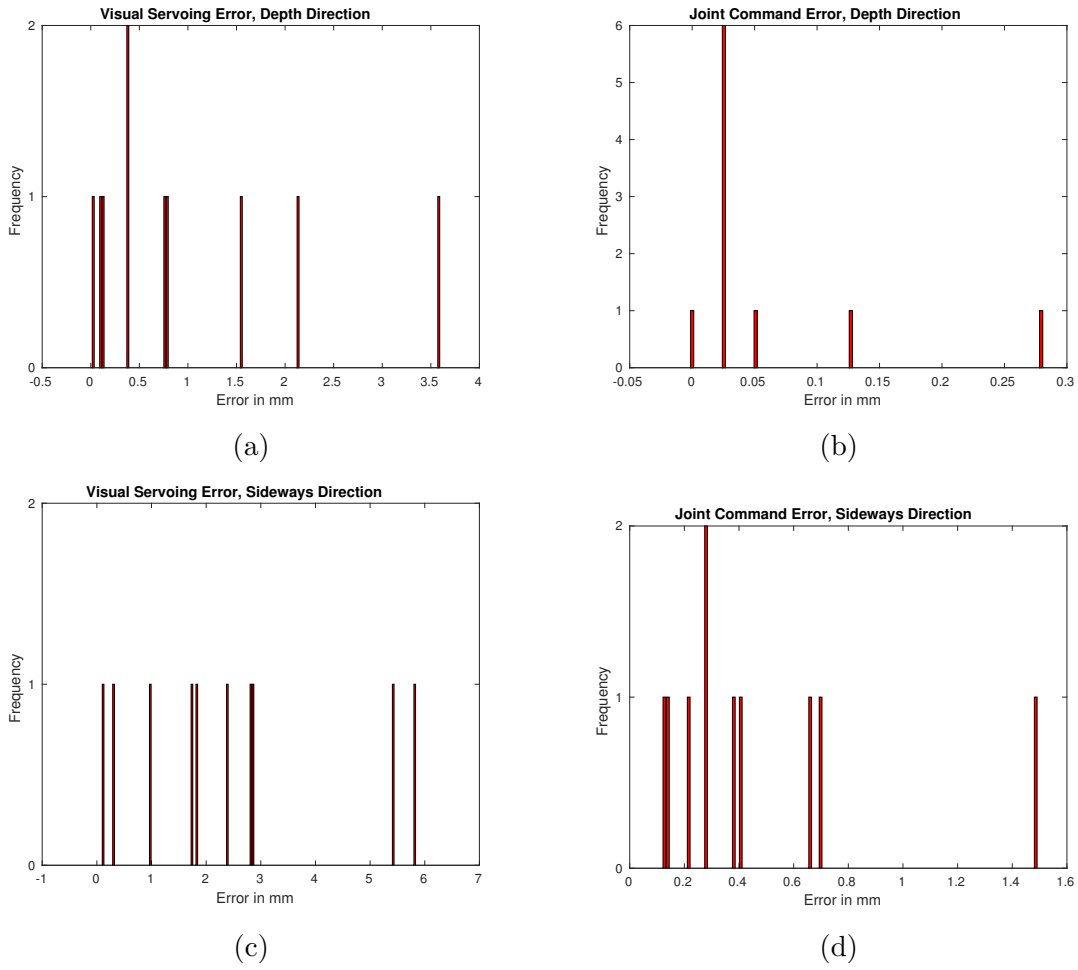


Figure C.8: A plot of the repeatability experiment error in the depth and sideways directions. The x axis shows the range of the errors in mm. The y axis displays the number of experiment iterations with the given error on the x axis. (a) Values for visual servoing in the depth direction. (b) Values for using joint commands in the depth direction. (c) Values for visual servoing in the sideways direction. (d) Values for using joint commands in the sideways direction.

	Depth Direction		Sideways Direction	
	Joint Command	Visual Servo	Joint Command	Visual Servo
Mean Error	0.0610 mm	0.983 mm	0.467 mm	2.43 mm
Standard Dev.	0.0840 mm	1.14 mm	0.408 mm	1.93 mm
Variance	0.00705 mm	1.29 mm	0.166 mm	3.73 mm

Table C.1: The table lists the results of ten iteration of each repeatability experiment.

movement. The measurement is taken between the initial position and goal (closer to the goal).

We move the robot to the desired goal position. There we record the joint angles and read the image location of the LEDs from the trackers as well as the value on the dial meter. Since we read the value of the trackers instead of clicking on goal points, we avoid having errors from user input affect our experiment. After moving the robot back to the initial position, we servo to the goal. The visual servoing error is made up of point-to-point tasks between the tracking markers and the recorded features at the goal. Both experiments are run for ten iterations, each starting at a different initial position. The initial positions were chosen manually. The visual servoing runs in a loop and we do not initialize new trackers each time we get back to the start position.

The average positioning errors for the experiments can be seen in Table C.1. The plot in Fig. C.8 shows a bar plot of the error for each run of the experiment for the visual servoing and joint command cases. We can see that the mean error for visual servoing is 0.983 mm and 2.43 mm, respectively, which is better than the 4.6 mm average error from out marker in cap experiment. The visual servoing, however, has a higher mean error than the robot joint command with 0.0606 mm and 0.467 mm error respectively. We also completed one experiment in the depth direction with the same initial and goal positions, where we let the user set the goal points by clicking. This experiment resulted in 1.86 mm error (1.14 mm standard deviation and 0.00130 mm variance), which is higher than for the pure repeatability.

The fact that the experiments in the sideways direction have a higher error than in the depth direction, is consistent with our observation from the accuracy experiment, where one direction had a higher error than the other. We think that the difference between the sideways and forward motion is due to the calibration of the robot. The joints do not compensate for the weight of the robot and the box as well in the

sideways as in the forward direction. This is also evident from the robots behaviour when stopped at the goal. For the forward movements the robot sits still at the goal, whereas for sideways motion the robot makes oscillations from side to side. Recall that in the accuracy experiment we would be unable to insert the marker into the cap directly without position adjustments in 7 out of 10 iterations. If it were not for the higher error in the sideways direction, we would likely be able to complete the insertion directly in most cases.

The measurements we achieve in this experiment are not as good as those reported in [41] and [29], where they accomplished results at least as good as robot joint command positioning. In [41] the authors get 0.13 mm and 0.059 mm error with two different robots and the authors of [29] get 0.1020 mm error. In the latter case they only complete a 3 DOF positioning task, which is not exactly comparable to 6 DOF positioning.

The higher visual servoing errors can come from different sources. Our first explanation is that camera placement will play a role. The closer a camera is placed to a task, the higher resolution is given to the visual servoing. In the 3 DOF robot finger visual servoing in [29], the cameras are placed about 30 cm from the task, which is about twice as close as the cameras in our experiment. [41] does not explicitly state the distance between the cameras and the task, but say that one pixel approximately corresponds to 0.25 mm of robot movement near the goal. In our case one pixel corresponds to about 0.94 mm movement not exactly at, but close to the goal. These numbers cannot be exactly compared due to the different measurement location, but suggest that we may be able to improve the results by moving the cameras closer to the task.

In Fig. C.9 we show the image error plots from the best and worst iteration of each experiment and Fig. C.10 shows the mean pixel error from the visual tasks at the end of each iteration. Although the image errors converge, we can see that they do not go exactly to zero. First and foremost tracking will not be perfect. Especially when the experiment is run for ten iterations, the trackers drift, see Fig. C.12. If we look at the repeatability errors across time, see Fig. C.11, then they do not get consistently worse for each iteration, which we might expect if the tracking drifted more for each iteration. In fact, when the robot moved from the goal and back to a new initial position, the tracking would sometimes get better. Even so, our mean visual error displayed in Fig. C.10 is too high. [41] report convergence to subpixel accuracy and zero visual error in 22 out of 50 experiments. We think that better

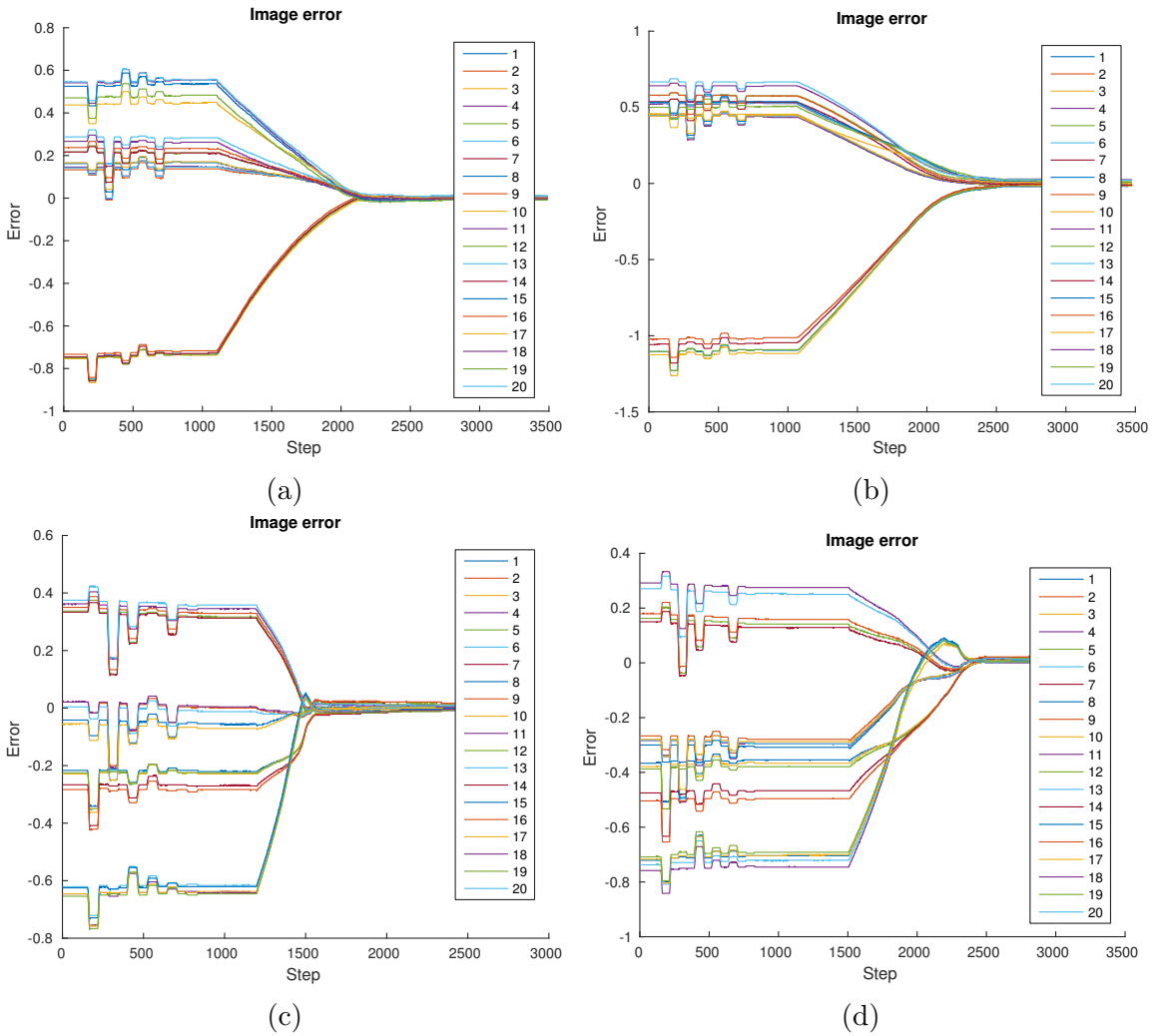
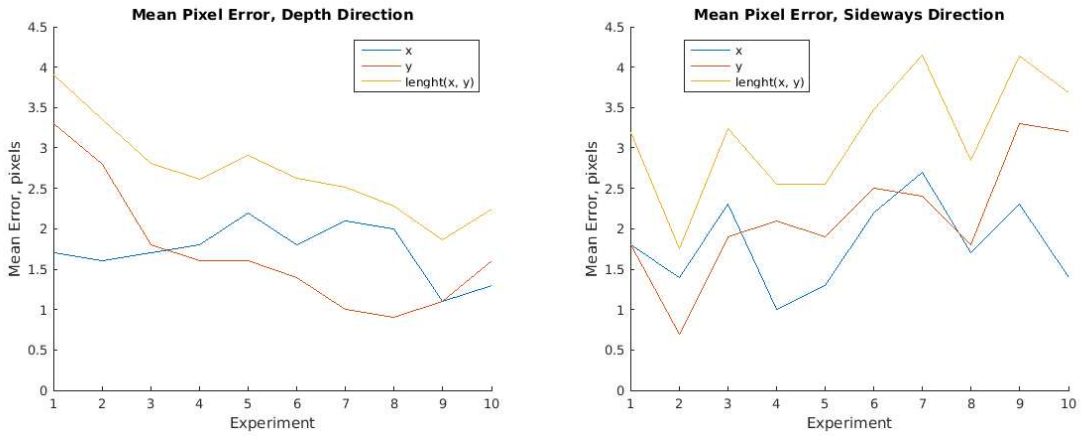


Figure C.9: The plots show the image error for the iterations with the best ((a), (c)) and worst ((b), (d)) positioning using visual servoing. (a) and (b) correspond to the depth direction and (c) and (d) to the sideways direction. The first and second halves of the labels belong to the image errors of the left and right image, respectively.

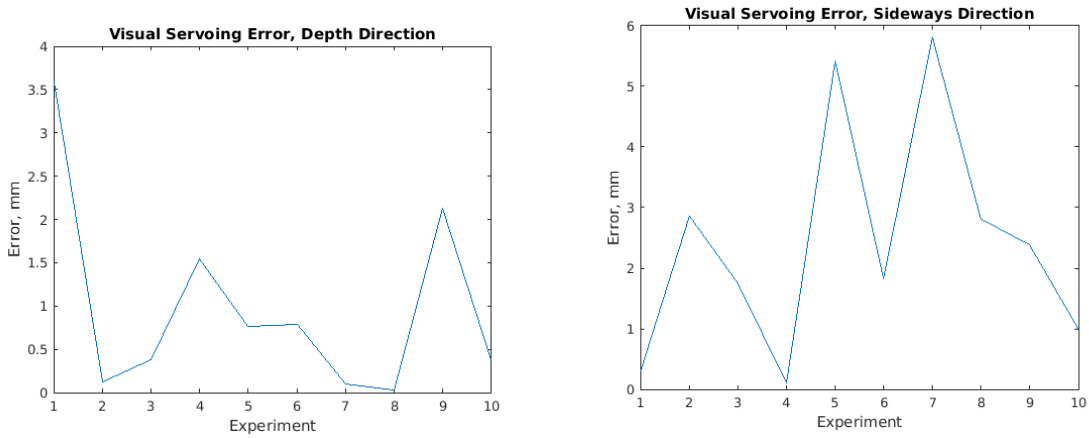
tracking will help improve the final visual error.

More than just tracking, there is also another cause that can affect the final visual error to not completely converge. The contact with the dial meter is a possible source of error. We chose a box that has a smooth surface to allow the tip of the meter to slide on the box. Some resistance will still be there and can affect the robot's ability to make small adjustments in position close to the goal. When we use the robot joint commands, the end-effector moves to the goal fast and in a fairly straight path, which gives it force to push on the meter. When close to the goal, image errors are small,



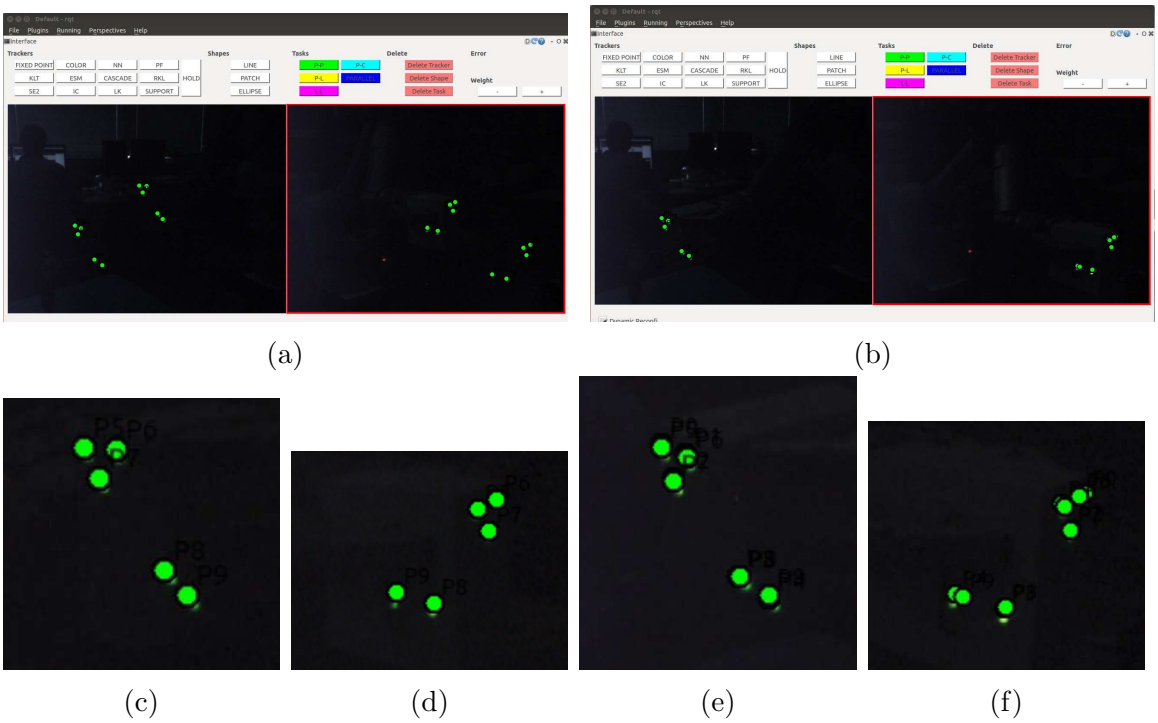
(a) (b)

Figure C.10: Mean pixel error across the point-to-point tasks at the end of each of the ten iterations. (a) Depth direction. (b) Sideways direction.



(a) (b)

Figure C.11: The plots show the error of each of the ten iterations. (a) Depth direction. (b) Sideways direction.



(a) (b)

(c) (d) (e) (f)

Figure C.12: The images show the robot at an initial position (a) and at the goal (b). Some of the trackers had drifted in the previous iteration. We can see close up pictures of the trackers in the left and right images for the initial ((c), (d)) and goal ((e), (f)) positions.

which leads the visual servoing to command smaller steps that makes it harder for the robot to push against the meter with sufficient force. In order to be able to track the LEDs for ten iterations we had to slow down the speed of the robot for visual servoing, which also affected the robots ability to push on the meter.

If we compare the plot of the mean visual error in Fig. C.10 with the repeatability error in Fig. C.11, we can see that the two do not exactly correspond. This means that our repeatability result can also be affected by more than just the visual error. [41] tracks the robot by putting LEDs on strings around the dial meter (the robot holds the dial meter in this experiment). We put LEDs on different planes of the box. We also need to put them close together to free surface space for contact with the dial meter. Furthermore, the LEDs on the sides of the box are only visible in one camera each. Hence the LEDs in [41] are placed further a part and are not restricted to the same planes, whereas out LEDs have less optimal placement. We think that because the LEDs are close together, several are in the same plane and four are only

seen by one camera, some ambiguity may occur. Small alterations in the robot pose in some directions may lead to smaller changes in the visual error than if the LEDs had better placement. Hence, we may reach some poses where the robot is slightly perturbed from the goal pose, but the visual error is still small.

In this experiment we presented results from a repeatability experiment for visual servoing. They show how accurately we were able to complete visual servoing without the involvement of the user clicking on goal points or combining tasks of different types. This gives some context to the errors we found in the accuracy experiment. Although we showed that the positioning error improved compared to the previous accuracy experiment, we think that the repeatability experiment can be altered to get improved performance closer to that of using robot joint commands.