# Direct Visual Servoing for Grasping Using Depth Maps

by

Abdullah Hojaij

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2013

## Abstract

Visual servoing is extremely helpful for many applications such as tracking objects, controlling the position of end-effectors, grasping and many others. It has been helpful in industrial sites, academic projects and research.

Visual servoing is a very challenging task in robotics and research has been done in order to address and improve the methods used for servoing and the grasping application in particular. Our goal is to use visual servoing to control the end-effector of a robotic arm bringing it to a grasping position for the object of interest.

Gaining knowledge about depth was always a major challenge for visual servoing, yet necessary. Depth knowledge was either assumed to be available from a 3D model or was estimated using stereo vision or other methods. This process is computationally expensive and the results might be inaccurate because of its sensitivity to environmental conditions.

Depth map usage has been recently more commonly used by researchers as it is an easy, fast and cheap way to capture depth information. This solved the problems faced estimating the 3-D information needed but the developed algorithms were only successful starting from small initial errors.

An effective position controller capable of reaching the target location starting from large initial errors is needed.

The thesis presented here uses Kinect depth maps to directly control a robotic arm to reach a determined grasping location specified by a target image. The algorithm consists of a 2-phase controller; the first phase is a feature based approach that provides a coarse alignment with the target image resulting in relatively small errors. The second phase is a depth map error minimization based control. The second-phase controller minimizes the difference in depth maps between the current and target images. This controller allows the system to achieve minimal steady state errors in translation and rotation starting from a relatively small initial error.

To test the system's effectiveness, several experiments were conducted. The experimental setup consists of the Barrett WAM robotic arm with a Microsoft Kinect camera mounted on it in an eye-in-hand configuration. A defined goal scene taken from the grasping position is inputted to the system whose controller drives it to the target position starting from any initial condition.

Our system outperforms previous work which tackled this subject. It functions successfully even with large initial errors. This successful operation is achieved by preceding the main control algorithm with a coarse image alignment achieved via a feature based control.

Automating the system further by automatically detecting the best grasping position and making that location the robot's target would be a logical extension to improve and complete this work.

**Acknowledgements**

I would like to express my sincere gratitude and appreciation to Prof. John Zelek for the time he has dedicated to my thesis and for his constructive feedback.
I extend my thanks to my readers Prof. Alex Wong and Prof. Steven Waslander for their very valued feedback that contributed towards improving my thesis.
Special thanks to Adel Fakih for his substantial help he provided. His advice was very crucial for the completion of this work.
To Dr. Daniel Asmar I owe thanks for the support and advice he generously provided me with.
And to my family, I would like to thank them for the spiritual support they gave me as they always motivated me to overcome the hardships I faced. Without them, my experience would have been much tougher.
I would also like to thank the Systems Design Engineering Department for offering the necessary amenities and facilities.
I owe many thanks to my friends especially Firas Mansour for standing next to my side at all times.

## Dedication

This work is dedicated to my family, especially my beloved parents.

# Table of Contents

# List of Figures

# List of Abbreviations

| | |
|---|---|
| PBVS | Position based visual servoing |
| IBVS | Image based visual servoing |
| RGB-D | Red, green, blue and depth |
| ToF | Time of flight |
| BoW | Bag of words |
| SIFT | Scale invariant feature transform |
| SURF | Speeded up robust features |
| U-SURF | upright speeded up robust features |
| CenSurE | Center surround extrema |
| DoC | Difference of circles |
| HoG | Histogram of oriented gradients |
| FLANN | Fast library for approximate nearest neighbors |
| LoG | Laplacian of Gaussian |
| DoB | Difference of boxes |
| DoO | Difference of octagons |
| NNS | Nearest neighbors search |
| SSD | Sum of squared differences |
| SFM | Structure from motion |
| RANSAC | Random sample consensus |
| ICP | Iterative closest point |
| PCA | Principal component analysis |
| WAM | Whole arm manipulator |
| DoF | Degree of freedom |
| ssh | Secure shell |

# Nomenclature

| | |
|---|---|
| $\alpha$ | The ratio of pixel dimensions |
| $\lambda$ | Constant proportionality gain |
| $\omega_K$ | Rotational velocity of the Kinect |
| $\mathbf{L}_Z$ | Interaction matrix related to $\mathbf{Z}$ |
| $\mathbf{L}_Z^+$ | The pseudo-inverse of $\mathbf{L}_Z$ |
| $\mathbf{v}_K$ | Kinect velocity screw |
| $\mathbf{Z}^*$ | Vector of depth map values of target scene |
| $\mathbf{Z}$ | Vector of depth map values of current scene |
| ${}^xX_y$ | Coordinate transformation from frame y to frame x |
| $c_u$ | x-coordinate of Kinect camera's principal point |
| $c_v$ | y-coordinate of Kinect camera's principal point |
| $e$ | The difference between current and target depth maps |
| $f$ | The Kinect camera focal length |

# Chapter 1

# Introduction

The use of visual data as the source of feedback information for controlling robots and robotic systems is becoming popular and is called visual sevoing as explained in Fig. 1.1. This topic attracted the attention of many researchers [17, 21, 2] and has become an interesting area of study because of its applicability to many of today's systems such as personal robots and industrial machines [11, 19, 34]. Various visual servoing techniques, have been developed and introduced for experimenting and testing [6, 7], such as Image Based Visual Servoing [12], Position Based Visual Servoing [46] and 2.5 D Visual sevoing [8]. Many of these new methods proved to be effective and successful in performing the control task for a given problem [43, 40] which other previous methods, like using only infra-red sensors for example, struggled with because of the existence of several issues, such as light reflection and refraction for instance. Currently, visual servoing faces many limitations including accuracy, expensive computations, robustness and stability [43]. The main drive behind improving upon existing methods, and overcoming the mentioned limitations, is to suit various applications, ranging from position control to autonomous grasping, for which the available techniques cannot perform effectively; the current methods are either slow, too expensive, computationally involved, limited to initial conditions, or not sufficiently robust.

## 1.1 Motivation

The benefits of visual servoing are many including the use of low cost sensors (such as cameras) which are readily available off-the-shelf and the potential for accurate operation by visually closing the control loop. These benefits made the usage of visual servoing in

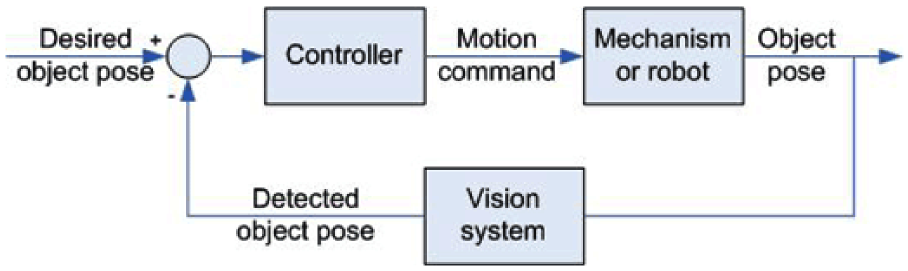


Figure 1.1: Visual Servoing Control Loop

many of the industrial robotic products more common [48]. It has shown to be extremely helpful for numerous applications such as tracking objects [34], controlling the position of end-effectors, grasping [37] and many others. Visual servoing made owning a personal robot that helps in everyday activity, which once was a dream, possible as shown in Fig. 1.2. But, as the ideas of more applications, with more complicated requirements and operating under various conditions, are presented on the table, the need for enhanced visual servoing techniques increases. These needed techniques should be simple enough not to overload the available computational capabilities, yet powerful and accurate in order to effectively control the robotic system resulting in only very small final errors that doesn't affect the success of the required task. To be able to do so, the limitations of current visual servoing methods have to be overcome. This will be discussed further in Section 1.2.

## 1.2 Challenges

Despite the aforementioned benefits of visual servoing, it is a very challenging task in robotics. Research has been done in this area in order to address and improve the methods used and overcome the challenges. Currently, many of the algorithms are slow because of complex matrix computations, particularly the homography matrix used to estimate 3D information about the scene, or 3D model reconstruction. They are also prone to singularities, that are present when computing the inverse of the image Jacobian necessary for the control law [17], and are sensitive to calibration errors and image noise [43]. In addition, most of these require 3D knowledge that is either computed using stereo vision or acquired using some depth sensors such as laser sensors. Therefore, speed, stability, robustness, simplicity and availability of necessary 3D information are all issues to be handled when working with visual servoing.

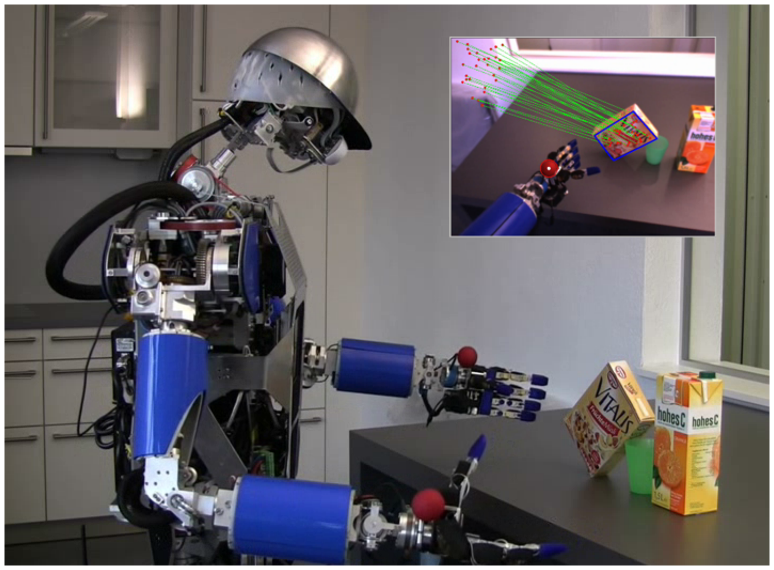


Figure 1.2: A humanoid Detecting Objects to Grasp [42]

Various visual sevoing techniques have been used but were subject to some of the limitations mentioned earlier. Also, the use of depth maps, which have recently become easily, quickly and cheaply computed, has been explored by some papers but were only successful for small movements [41]. Our goal is to combine the usage of depth maps with visual servoing in order to control and position the end-effector of a robotic arm close to an object of interest starting from large initial errors.

The main features of the thesis work are:

- seamless operation (achieved when the algorithm's computations are performed to send smooth and continuous motion commands to the robotic arm),
- simplicity and robustness (avoiding complex matrix computations because of the availability of 3D information extracted directly form the depth maps making the system also less sensitive to image noise),
- low cost setup and operation (a cheap and readily available Microsoft Kinect camera is the only sensor needed),
- independence of camera calibration and

- independence of any sort of training (ability to handle situations when confronted with novel objects).

These features, combined, lead to effective work mainly contributing in the process of successfully controlling the position of a robotic arm towards reaching a specified target location starting from almost any initial position.

## 1.3 Outline

Previous work and a literature review is presented in chapter 2 showing the various basic and improved visual servoing techniques, their advantages and disadvantages and their suitable applications. Chapter 3 explains our proposed method utilizing visual servoing techniques and depthmaps information to handle a wide variety of applications. In chapter 4 we explain the experimental setup and the experiments performed to evaluate the effectiveness of our technique. Experimental results are presented and discussed in chapter 5. Finally, chapter 6 concludes this thesis and opens a window to look at potential future work.

# Chapter 2

# Literature Review

## 2.1 Introduction

Visual servoing is a very challenging task in computer vision, but it has lots of benefits and it is especially useful in many robotics applications. Research has been done in this area in order to address and improve its speed, robustness, accuracy and reliability. The work done in this field ranges from explaining and testing the basic visual servoing techniques, to exploring and developing new and enhanced methods to achieve the at required performance. The following sections present what was discussed in the literature regarding visual servoing techniques and their applicability. For instance, tracking is one obvious application that uses visual servoing [2].

## 2.2 Visual Servoing

Visual servoing is the use of visual data as the source of information used as feedback necessary to effectively control robotic systems as shown in Fig. 1.1. Because of the uses of visual servoing, it attracted the attention of many researchers who developed several techniques to perform it.

### 2.2.1 Summary

The most basic and most famous visual servoing techniques are the IBVS (Image Based Visual Servoing) and PBVS (Position Based Visual Servoing) from which hybrid methods were developed that outperform the basic ones. One of the hybrid visual servoing techniques is 2.5 visual servoing which is a compromise between the IBVS and PBVS benefiting from their advantages and avoiding their disadvantages.

- The PBVS extracts locations in the images and control the robot based on these locations.
- The IBVS uses information in the images to directly control the robot without computing positions, making it less computationally expensive than the PBVS, however, it is more prone to singularities.
- The main advantage of 2.5 visual servoing is that it decouples the rotational and translational control loops, making it more robust and stable against singularities but more sensitive to image noise.

All these techniques require 3D knowledge and thus 3D information techniques need to be used. Stereo vision was adopted by some methods to compute the necessary 3D parameters. Another way is to directly extract the depth information using depth maps.

### 2.2.2 Visual Servo Control for Manipulation

The authors of [17] present an introduction to visual servo control for robotic manipulators. They discuss the two main methods for visual servoing, namely image-based visual servoing (IBVS) and position-based visual servoing (PBVS). Feedback from visual sensors can close the loop for more accuracy in the system. Details regarding IBVS and PBVS are further explained in [6] to which [7] adds more advanced techniques such as hybrid visual servoing.

Visual servoing is the use of machine vision to control the position of the end-effector in a closed loop system. It has been used in numerous applications including robot juggling and grasping.The tutorial presented in [17] touched upon many important definitions and aspects of visual servoing which are briefed below.

- Definitions:
  - Coordinate Transformation: For tracking, rotations alone are sufficient, while for simply picking and placing objects, translations only are enough. ${}^{x}X_{y}$ represents rotation and translation matrices of either a pose or a coordinate transformation.

- Camera Projection Model: Depth information is lost after projecting a 3D scene into the 2D image plane. To retrieve additional information, several alternatives can be used including multiple cameras, one camera with multiple views or having some geometric knowledge. Three major projection models are discussed:
  1. Perspective projection which represents a linear projection from camera frame coordinates to image plane coordinates,
  2. scaled orthographic projection which approximates the perspective projection by a linear scaled orthographic one and
  3. affine projection which is another approximation of the perspective projection.
- Image Features: The image plane coordinates are used as the parameters of the point features. The projective geometry of the camera is used to map the pose of the end-effector to the relevant image feature parameters.
- Camera Configuration: There are two main camera configurations:
  1. Eye-in-hand configuration represents mounting the camera on the robot itself so that the camera moves with the robot. In such a configuration, the relation between the poses of the camera and the end-effector is known and usually constant denoted by ${}^{e}X_c$.
  2. Eye-to-hand configuration represents cases in which the camera is fixed in the workspace or on a mobile platform and the robot is seen by the camera. Camera calibration is a must for such configurations.

• Categorizing Visual Servoing: Visual servoing can be divided into four categories [17]:

1. Dynamic look and move system with a hierarchical control architecture and the vision system providing input to the joint level controller.
2. In direct visual servo, the joint inputs are directly computed with the visual servo controller completely replacing the robot controller.
3. In position based control, the vision is used to extract features to estimate the pose of the target.
4. While in image-based servoing, the extracted features are used to directly compute inputs to the controller. It has lower computational delays than position-based servoing and is not affected by calibration errors, but it is more challenging.

The first category is more commonly used in favour of the second one for various reasons:

1. The vision system has relatively low sampling rates
2. Available robots usually utilizes Cartesian position and velocity inputs
3. It separates the kinematic singularities from the visual controller

Endpoint open-loop (EOL) systems only observe the target while endpoint closed-loop (ECL) systems observe the target and the end-effector resulting in a more accuracy in positioning since it is less sensitive to calibration, but it is more demanding.

- Position Based Visual Servoing [17]:
  - An error representing the difference between a desired pose and the current pose of the target with respect to the camera, which is estimated from the extracted features, is defined. A positioning task is fulfilled by driving a kinematic error function to zero. This is done via a regulator that sends end-effector velocity screw inputs to the robot controller at each time instant.
  - Point feature-based motions. Point to point positioning tasks drive a point on the robot's end-effector to a stationing point in the workspace. Another example is moving a point on the robot's end-effector in-line with two fixed points in the workspace.
  - Pose-based motions. Choose a desired stationing pose and define the error function to be the difference between that pose and the current end-effector's pose.
  - Pose-based methods are commonly used. However, they are computationally expensive and depend on knowing a model of the target. When a target model is not available, feature-based approaches are more appropriate.
- Image Based Visual Servoing [17]:
  - The Jacobian matrices for image point coordinates are function of depth which is to be computed.
  - It is computationally less expensive than position based control, but singularities are present when using this approach.
- Feature Extraction and Tracking [17]:

  Window-based tracking techniques are efficient and are done by first acquiring windows and then processing them to locate image features. Edge-based or area-based methods can be used. The latter is used when a specific feature pattern is desired but is not desired when occlusions or background changes exist. Window-based methods only work for small displacements unless lower level resolutions are used. Another disadvantage of such methods is that it relies on exact matches.

### 2.2.3 2.5D Visual Servoing

2.5D visual servoing [25] is a hybrid servoing method that is between image-based and position-based visual servoing avoiding the disadvantages of both. It doesn't need geometric 3-D model and ensures the control law convergence. The main advantage of the 2.5D method is that it allows the decoupling of the translational and rotational control loops. Estimating the rotation and translation of the camera, from the original to the required configuration, is the base of this method. A survey on visual servoing that focuses on image-based, position-based and 2.5D visual servoing can be found in [21].

A thesis work about 2.5D visual servoing is presented in [43]. In this work, a 2.5D visual servoing technique was developed for an eye-to-hand system. Accuracy, real-time operation and the robot kinematics are three main constraints for pose estimation algorithms. For the estimation of the 3D parameters without the use of a 3D model, homography was used. When compared to IBVS and PBVS, experiments showed that the 2.5D is more stable, robust, and less sensitive to calibration errors. It also helps overcome singularity problems. On the other hand, it is more sensitive to image noise due to the homography matrix calculation. Using the depth map of the Kinect as the source for the 3D parameters would be the next step for improving on this work.

## 2.3 Estimating 3D Information

As explained in Section 2.2, the methods explored in the literature require some sort of 3D knowledge. The latter could be either estimated, computed or sensed.

### 2.3.1 Summary

Some work assumed the availability of a 3D model [45], while some computed the 3D information off-line [47]. Other work estimated or sensed the 3D information on-line. Researchers used stereo vision to estimate the 3D pose [14] and others recently relied on low cost off the shelf depth RGB-D sensors to obtain the depth map of the scene [41].

### 2.3.2 Stereo Visual Odometry

The work in [14] presents a visual odometry algorithm for autonomous ground vehicles. The algorithm consists of a feature detection step, followed by descriptor formation. Then comes the inlier detection step after which the frame-frame motion is computed by minimizing the re-projection error of the features. Finding the inliers rather than rejecting the outliers is one of this algorithm's biggest advantages by making it insensitive to incorrect matches. It is done by forcing the 3D locations of the features to obey a rigidity constraint. This also makes the algorithm capable of handling scenes that include motions. Another advantage of this algorithm is that it does not require an initial motion estimate and it can handle large translations, however, the it cannot handle rotations or scaling. The algorithm presented in [14] showed high degrees of reliability and accuracy. It is intended to compliment the use of other sensors for improved performance.

### 2.3.3 Direct Visual Servoing Using Depth Maps

The work in [41] presents a 3D visual servoing technique based on depth maps extracted from RGB-D sensor data without estimating 3D pose or matching image features. The visual error in IBVS techniques is usually the difference between certain geometric features in current image and the desired one. In [41], differences in depth maps are being used for the servoing. Depth maps can be constructed using Time-of-flight (TOF) range sensors or projecting light patterns and sensing the projections to extract the depth information. The latter method is what the Microsoft Kinect uses and is what is being used in this thesis [41].
The error that has to be minimized by this IBVS technique is the difference between the depth map of the current image and a desired depth map as shown in (2.1).

$$\mathbf{e} = \mathbf{Z} - \mathbf{Z}^* \tag{2.1}$$

Where $\mathbf{Z}$ is the current depth map and $\mathbf{Z}^*$ is the desired depth map. Based on the control law discussed in [6], the interaction matrix $\mathbf{L}_Z$ is related to the velocity screw $\mathbf{v}$ and the rate of change of the depth $\frac{\partial \mathbf{Z}}{\partial t}$ as in (2.2):

$$\frac{\partial \mathbf{Z}}{\partial t} = \mathbf{L}_Z \mathbf{v} \tag{2.2}$$

As an example, for an exponential decrease in the error ($\dot{e} = -\lambda e$), a simple proportional control law can be defined as (2.3):

$$\mathbf{v} = -\lambda \mathbf{L}_Z^+ \mathbf{e} \tag{2.3}$$

where $\mathbf{L}_Z^+$ is the pseudo-inverse of $\mathbf{L}_Z$ and is defined as in (2.4):

$$\mathbf{L}_Z^+ = (\mathbf{L}_Z^T \mathbf{L}_Z)^{-1} \mathbf{L}_Z^T \tag{2.4}$$

To improve the robustness of the algorithm, few measures were taken. A $3 \times 3$ Gaussian filter is applied to the depth image to reduce the noise. To reduce the effects of occlusion and scene modifications, M-estimation was implemented by weighing the error according to a robust function.

To test the effectiveness of the algorithm, several experiments were performed. After acquiring the desired depth map, the robot was used to some initial position from which the controller would try the move it to the desired position at which the robot would sense the desired depth map. A fixed gain, $\lambda = 2.5$ was used in all the experiments. The results showed that the algorithm is robust and insensitive to changes in illuminations. It had some limitations when it comes to noise and missing data, but the filtering and the use of M-estimation resolved these limitations.

## 2.4 Image Features

Features are image pixels containing meaningful information. Features could be of various sizes ranging from few pixels to a large part of the image. These features can be described in various ways to be later detected using certain feature detectors. Various feature descriptors and detectors are studied in the following sections.

### 2.4.1 Summary

Images are generally represented, after subsequent processing, by either:

- the bag of words (BoW) [9] also called bag of visual features or
- an unstructured list of features [32].

The BoW focuses on details found in the image and tries to group the features into related patches while the other approach captures general information about the image without finding relations or grouping.

The features present in the image are detected by one of the studied feature detectors. The most common feature detectors are:

- scale invariant feature transform (SIFT) [24],
- speeded up robust features (SURF) [5],
- center surround extrema (CenSurE) [1] and
- The difference of circles (DoC) [13].

SIFT was shown to be very robust and effective but slower to compute while SURF was less invariant to rotations but much faster and simpler to compute.

These features are then described using what is called a feature descriptor. Many feature descriptors were explored in the literature from which we chose to discuss:

- scale invariant feature transform (SIFT) [24],
- speeded-up robust features (SURF) [5] and
- histogram of oriented gradients (HoG) [10].

Each one of the above mentioned descriptors has different advantages and disadvantages when it comes to speed, robustness and invariance to scale, rotation and illumination.

For applications requiring comparison or recognition, the detected features need to be matched. For that we discussed two main feature matchers which are:

- the brute-force matcher and
- the fast library for approximate nearest neighbors (FLANN) matcher [30].

All of the feature descriptors, detectors and matchers are further discussed in the sections to follow.

### 2.4.2 Image Representation

Before deciding on the feature descriptor and detector to be used, we need to choose the suitable criterion for understanding the image.

The two main categories for image representation are:

- the bag of words (BoW) and
- finding an unstructured list of features.

The BoW finds the occurrence of features in the image [9, 23] focusing on small details while finding an unstructured list approach is a simpler approach [32] that gathers general information found in the image by detecting the features that exist [33] and is used to compare the overall context of images as opposed to detailed features. We are more interested in the unstructured approach as it suits our application more since we are interested in finding matches between features in the current and target images and we are not looking for anything specific.

### 2.4.3 Feature Detector

Images are usually rich in features which represent the useful information in them and to be able to utilize this information, these features have to be detected. Feature detection was thoroughly studied in the literature and the most common ones are:

- scale invariant feature transform (SIFT),
- speeded up robust features (SURF),
- center surround extrema (CenSurE) and
- The difference of circles (DoC) was also studied in the literature.

Before discussing the various feature detectors listed above, one should note that feature detection has been an essential building block in many widely used image processing applications [26]. In order to effectively achieve the applications' goals, the detected features have to satisfy certain characteristics including illumination, scale and rotation invariances.

These features' properties are especially necessary in applications such as tracking and object recognition where rotations and scale changes happen quite often. SIFT [24], one of the most popular feature detectors is scale and rotation invariant. The idea behind it is to use extrema in the Laplacian of Gaussian (LoG) as a cue of saliency. However, the LoG is very expensive to compute since it involves computing the square of the derivative at each pixel; this has to be done at each scale. Approximating the LoG is very common in the literature where several different approximations exist. For that purpose, Lowe [24] approximates the LoG by the difference of Gaussian (DoG) which is a faithful approximation of the LoG, however, it involves convolving the image with a Gaussian filter at each scale. The latter filter becomes very large at higher scales and the cost of the convolution becomes very huge. To reduce the computational cost at higher scales, Lowe [24] suggested down-sampling the image by a factor of four each time the scale was doubled to compensate for the computational increase. Sub-sampling the image leads to poorer localization of the features at higher scales.

As a faster detector, the Speeded-up robust features (SURF) was introduced [4]. Specifically, the upright SURF (U-SURF) uses the difference of boxes (DoB) as an approximation of the LoG, however, the features become no more rotation-invariant unless the rotation angles are very small and not exceeding $15^o$ [5]. One of the main advantages of using the DoB is that they can be computed very efficiently using integral images [44]. Other than resulting in a faster detection, the DoB based method doesn't require sub-sampling and hence doesn't have localization troubles with high scale features.

A solution that is midway between the previous approaches and that could outperform the previously mentioned detectors, was introduced in [1]. The authors suggested using the center surround extrema (CenSurE) to detect the image features. The CenSurE are scale and rotation invariant and yet fast to compute because they take advantage of the integral images. To maintain some level of rotation invariance, they used the difference of octagons (DoO) as the LoG approximation, an approximation that is better than DoB. They avoided using the difference of circles (DoC) although it is more faithful to the LoG and would also lead to much better rotation invariance [1] because it can not be computed directly using integral images. The star detector developed at 'Willow Garage' is very similar to the CenSurE and uses the same concepts and ideas, but utilizes the difference of stars (DoS) instead of octagons where a star is created by having two concentric boxes on top of each other with the box on top rotated by $45^o$ with respect to the lower one [20]. The DoC was explored for the purpose of feature detection in [13] which introduces a new kind of circular integral images to speed up the computation of the DoC which is considered a very faithful approximation to the Laplacian of Gaussian (LoG) used for feature detection.

### 2.4.4 Feature Descriptor

Features in the images are represented according to the descriptor used to describe them. Many feature descriptors were studied in details and researchers have done many experiments to compare between them. The most common descriptors are:

- scale invariant feature transform (SIFT),
- speeded-up robust features (SURF) and
- histogram of oriented gradients (HoG).

The histogram of oriented gradients (HoG) evaluates the local normalized histograms of the gradient of orientations of an image and represents them in a grid. The method is based on the observation that the intensity gradients' directions represent local features [10]. The image is first divided into a grid of small group of cells for which histograms of the directions are accumulated. The histograms of all the cells are combined and normalized to form the histogram of oriented gradient (HoG).

Scale invariant feature transform (SIFT) preceded the HoG and was discussed in [24] which uses extrema in the Laplacian of Gaussian (LoG) as a cue of saliency. Computing the LoG is very expensive because it involves computing the square of the derivative at each pixel and at each scale. To speed up the process, Lowe [24] approximates the LoG by the difference of Gaussian (DoG) which is a faithful approximation to the LoG, however, it involves convolving the image with a Gaussian filter at each scale. The latter filter becomes very large at higher scales and the cost of the convolution becomes very huge. To compute the SIFT descriptor, using a cascade of filters, the keypoint locations are found in the image around which the image gradient magnitude and orientations are sampled and filtered using the Gaussain blur [24]. To reduce the computational cost of the Gaussian filter at higher scales, Lowe [24] suggested down-sampling the image by a factor of four each time the scale was doubled to compensate for the computational increase as depicted in Fig. 2.1. The cost of sub-sampling is poorer localization of the features at higher scales but it saves on the computational time required.

The SIFT descriptor showed to be a good representation of saliency and the experiments conducted demonstrated its effectiveness in detecting distinctive features which enabled correct object matching. SIFT proved to be rotation and scale invariant and robust against image distortions [24].

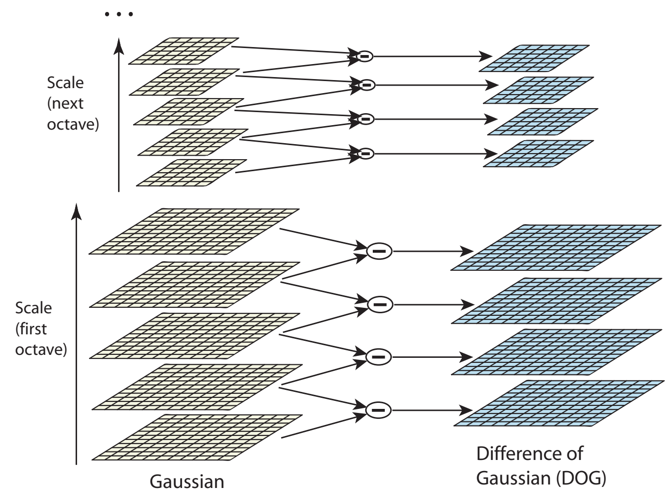


Figure 2.1: The Difference of Gaussian showing the down-sampling at each scale [24]

Despite SIFT's great performance, it still requires complex computations making it a relatively slower detector when compared to few other available ones. As a faster descriptor, the Speeded-up robust features (SURF) was introduced [4]. The SURF descriptor is extracted from square regions which are constructed to be aligned with a fixed orientation chosen based on information around keypoints [5]. Specifically, the upright SURF (U-SURF) uses the difference of boxes as an approximation of the LoG as opposed to the DoG used by SIFT, however, the features become no more rotation-invariant unless the rotation angles are very small and not exceeding $15^o$ [5]. One of the main advantages of using the DoB is that they can be computed very efficiently using integral images [44]. Other than resulting in a faster detection, the DoB based method doesn't require sub-sampling and hence doesn't have localization troubles with high scale features. Therefore, SURF is faster than SIFT but isn't as rotation invariant.

### 2.4.5 Feature Matcher

After describing the detected features in an image, these features has to be compared and matched with features detected in the original image. Several feature matchers are available in the literature from which we discuss two which are:

- brute-force matcher and
- fast library for approximate nearest neighbors (FLANN) matcher.

The brute-force matcher finds the distances between a feature in the current image and all the features in the target image. The features with the closest distance represents a match.

The fast library for approximate nearest neighbors (FLANN) requires training that allows to quickly find the feature closest to the processed features. This library performs a fast nearest neighbors search (NNS) and is written in C++ language but can be used in other programming languages as well [30].

## 2.5 Grasping

Visual servoing is a tool for robotic manipulation. Grasping is an application that would use that tool to benefit the world in real life situations. We are particularly interested in the ability to grasp any object including those seen for the first time. The grasping component of this research is beyond the scope of this thesis, but it is explored to study its feasibility for future completion.

### 2.5.1 Summary

For a robotic arm to grasp, it has to deicide on a grasping point, drive the gripper to the relevant location and perform the grasp. A robot can learn to grasp using a trial and error training approach to assess future grasps starting from a full 3D model. Other strategies handles occlusions and/or unavailability of complete 3D information. Contact and proximity sensors may be added to enhance on the grasping process. Some grasps were preceded by visual tracking to make the grasps more realistic and applicable to many tasks including picking objects from a moving conveyor belt or playing chess. More complicated grasps could be virtually assessed using a grasping simulator, 'GraspIt'. Limitations still existed hindering the applicability of grasping, so, to overcome some of these limitations, a jamming gripper was introduced, a new gripper design, allowing compliant grasps using less information.

### 2.5.2 Learn To Grasp

Finding the grasping location is a crucial step when attempting a grasp. [19] helps a robotic arm learn to grasp using visual information based on saving the locations of grasping trials after grading them. This information is later used to choose grasping points on new targets. Another teaching scheme is presented in [39] which is designed to function even when only an incomplete 3D model of the object to grasp is available. The latter algorithm is able to utilize the partial information visible in order to position a robotic arm in a suitable grasping location.

### 2.5.3 Improving Grasps

When attempting an actual grasp, the object to grasp is not always fully known or sensed. Thus, a robust grasp algorithm should be able to deal with uncertainties in the environment and the object's location and shape. To select a proper grasp, [15] uses a contact-reactive algorithm that performs compliant grasping based on information sensed by sensors installed on the fingertips of the robotic arm allowing for robust and reliable grasp even when information is missing.

Another way to improve on grasp algorithms is to use optical proximity sensors installed on the fingertips of the hand of a robotic arm. [16] uses such proximity sensors to perform online grasp adjustments to improve on initially found grasp points before the hand gets in contact with the object to grasp. The presented algorithm compliments previously presented grasp algorithms.

### 2.5.4 Control with Visual Feedback

A great way to perform effective grasps is to use computer vision to provide feedback and close the control loop. A vision sensor could be used to perform visual tracking, for instance, as a step preceding a grasp attempt in a realistic robotic problem. Such technique was developed in [34] which used the sum-of-squared differences (SSD) optical flow followed by a Kalman filter to estimate the motion of object being tracked. Similar technique was used in [48] in order to track and pick objects from a fast moving conveyor belt.

### 2.5.5 Grasp Simulator

Performing grasps in real life environments requires the robotic arm to have extra capabilities, mainly flexibility. Designing robots with more degrees of freedom needed for more flexibility comes at the cost of increasing the complexity of the robotic system. Evaluating and testing these new robots is very essential for the assessment of the validity and effectiveness of these designs. Thus, [29] came up with a tool that simulates and evaluates, visually and quantitatively, the feasibility and effectiveness of grasps performed by the modelled robotic arms. The simulator is called 'GraspIt' and very helpful when tackling a grasping task.

### 2.5.6 Applications

Grasping methods are developed to eventually suit real life tasks. Few of these applications were tested and explored in the following papers each addressing a different aspect showing the vast applicability of grasping.

- Gambit: An Autonomous Chess-Playing Robotic System [28].
- Grasping From the Air: Hovering Capture and Load Stability [35].
- Robotic Ball Catcher [11].

### 2.5.7 Universal Gripper

Despite the enhancement of the grasping methods and incorporating various sensors to increase the availability of useful information, limitations still exist, some of which are imposed by the design of the gripper being the end-effector in contact with the object being grasped. To eliminate some of these limitations, [18] designed a new innovative gripper called the 'universal jamming gripper'. The latter consists of bag with from a flexible membrane containing granulated matter that conforms to the shape of the contact area between the gripper and the target object. The bag is then vacuum hardened by adjusting the air pressure inside it allowing for compliant grasps as well. This gripper is very versatile because it doesn't require the 3D model knowledge of the target, nor does it require accurate localization.

## 2.6 Grasping Novel Objects

Most grasping algorithms require knowledge about the object to be grasped and thus, they struggle when confronted with novel objects seen for the first time by the robot. To address this issue, several methods were discussed in the literature.

### 2.6.1 Summary

To be able to grasp novel objects especially those seen for the first time by the robot, grasping points were determined from the 2D images. Only those points were then triangulated to find the corresponding 3D grasping position. Such algorithms were enhanced to even deal with cluttering and the to avoid obstacles existing in the working environment. Other work was done to generate 3D models using laser scanners, or off-line using structure from motion.

### 2.6.2 Determining the Grasping Point

[37] presents a machine learning algorithm that finds grasping points on objects seen for the first time. This is done after the system is trained on a synthetic set of objects. The proposed algorithm does not use a 3D-model of the object, nor attempts to build one as opposed to previous work which assumed that the object being grasped is known or a 3D-model of it is given.

The algorithm works by first detecting features in the image and then calculating the probability of being a valid grasping point. For each grasping point identified, the 3D position of that grasping point is computed using triangulation from 2 or more images of the grasping point. Triangulation is performed only for the valid points rather than triangulating the whole image as is done when acquiring the 3D-models saving on computational time. The details of [37] are explained in the sections to follow.

#### Triangulation

For the triangulation process, the 3D-space around the determined grasping point is discretized into a grid. Rays are then built from the grasping points using the knowledge of the camera position and pose with a Gaussian conical error around it. The grid cell with the maximum response from the rays is chosen as the grasping point. Because grasping points can be vague, the spacial structure of the object can be used to improve the choice of grasping points.

#### Control

Grasping objects with the robotic arm was done using a two-step control strategy. First, the end-effector approaches the object, and then move on a straight line towards the grasping point.

The algorithm was tested using a 5-DOF robotic arm mounted on a mobile platform. Experiments showed high success rates and accuracies in grasping novel objects.

### 2.6.3 Grasping Points Based on Features

In [36] and [38], the authors described the approach used to detect features representing edges, texture and color from which grasping points are selected. To include global properties of the object, features are extracted at 3 spatial scales. Then, features in a 5×5 window of 25 patches are computed thus including information from 24 neighbours.

#### Probabilistic Model

To grasp, two or more images are taken from different camera positions. Camera positioning errors are taken into account and assumed to be Gaussian. Correspondence issues are challenging. To deal with them, the 3D-workspace is discretized and rays are built from each different image to each grid element. Intersecting rays represent good matches. A match represents a valid grasping point if its 2D position on the image is a grasping point.

### 2.6.4 Grasping in Cluttered Environments

The authors of [37] and [36] continue their research to address the problem of grasping in cluttered environments [40]. Their proposed algorithm does four tasks:

1. Decide on a grasp
2. Locate obstacles
3. Plan a trajectory to reach the object without hitting the obstacles

4. Finally, actually grasping the object

As mentioned before, most of the work done in the literature assumes the knowledge of a 3D-model of the object. However, even if stereo vision or laser scanners are used, the output is not good enough for real world grasping applications; the former produces noisy point clouds and the points acquired by the latter are sparse.

**Algorithm**

The system is taught some visual features that represent good grasps and that are consistent among objects of particular types. By this, the robot's kinematics and workspace are not considered, so, the best grasp (grasping point and orientation) is to be chosen from the set of possible grasps determined. Knowing the environment and perceiving the obstacles in it requires the 3D-model of the workspace. But, because of the inaccuracy problem accompanied with acquiring the 3D-model (discussed earlier), several fixed objects were stored into the system. Probabilistic Road Map (PRM) was used as the path planner to decide on how to navigate the arm while avoiding the obstacles. As for the hand, minimal opening of the fingers was attempted. Experiments were carried out on the 7-Dof Barrett WAM and a grasping success rate of 80% was achieved.

### 2.6.5 Grasping Based on 3D-Model

Others addressed grasping by reconstructing a 3D-model of the object being grasped [45]. The authors used a laser scanner to construct the 3D-model of the object in interest. For the grasping process, the paper focusses on four issues:

- features found in the object to be grasped,
- the configuration of the hand needed for a proper grasp,
- directions and magnitudes of the contact forces between the hand and the object and
- required forces to be exerted by the fingers on the grasped objects.

Virtual simulations of grasps are done and evaluated using software in order to select the best grasp that is executed by the actual robot while avoiding collision. The success rate of the grasps performed was 94% where failures were mainly caused either by the hand moving the object from the grasp location or by the object slipping from the robot's hand.

### 2.6.6 Grasp Planning based on Off-line 3D Modelling

In [47], the authors try to generate a 3D-model of an object and manipulate it autonomously. 3D-models were generated using SFM(Structure From Motion) without any prior object knowledge. The object modelling achieved by this paper had 5mm error for 15cm big objects. 2 major problems faced the authors:

1. Large amount of redundant data was available. The authors dealt with this issue by using voxels, small cubic bins.
2. Shape errors were present. The authors tackled this by averaging the shape error.

The criteria used to select a grasp were:

- Enough contact area,
- stability against gravity,
- availability of enough space to manipulate the object after grasping it and
- little robot's motion while executing the grasp.

To meet the criteria, the authors tried to minimize an objective function that encompasses the requirements. The experiments performed proved their approach's effectiveness.

## 2.7 3D-Modelling

The availability of a 3D model of the target is very essential for many grasping algorithms. Work has been done to generate a 3D-models of the objects to be grasped. Various techniques and approaches were used.

### 2.7.1 Summary

A 3D model of the object of interest could be generated using RANSAC and ICP to fuse images into a coherent 3D model. It was even possible to generate 3D models from a single scene view using a probabilistic model fitting.

### 2.7.2 Modelling Objects in-Hand

Most of the available literature (including ones discussed earlier) base their modelling techniques on either training or prior knowledge and cannot handle totally new objects. [22] addresses the issue of modelling new objects without depending on accurate sensors or delicate robots. Based on RANSAC (Random Sample Consensus) and ICP (Iterative Closest Point), the authors use a Kalman filter tracking approach to fuse the images of the object into a coherent 3D-model. In attempt to autonomously grasp and model new objects confronting the robot for the first time, the authors use a very simple approach that performs a PCA (principle component analysis) of the object scene obtained after subtracting the background from the image. They claimed that this approach worked well enough for performing initial grasps.

### 2.7.3 Modelling from a Single View

Based on a single image snapshot, the authors in [27] tried to generate a 3D-model of the object of interest using a probabilistic model fitting that assumes the object to be a combination of simple basic shapes. Since the only available data is taken from one image, then only one view of the object is available; therefore, their approach guesses the shape of the back of the object assuming symmetries common in everyday's objects and that these objects are physically stable. The sensors used are stereo cameras, 3D sensors or a combination of both. The authors claim that their generated models were precise enough for the grasping application.

## 2.8 Conclusion

Much research has been done to improve on visual servoing. Hybrids between the most common visual servoing techniques (IBVS and PBVS) were developed to help suit a larger number of real life applications. The use of depth maps was later introduced to directly perform visual servoing. Conventional visual servoing techniques struggled with stability and robustness issues in addition to their computationally expensive costs. Despite the benefits of the state of the art direct visual servoing that include elevating the need for training or camera calibration, the main limitation of such an approach is that it is only applicable to small errors. The work in this thesis addresses the direct visual servoing task to make it suitable for a large range of applicability.

# Chapter 3

# Theory

As discussed earlier, visual servoing is extremely helpful for numerous applications such as, tracking objects, controlling the position of end-effecters, grasping and many others. It is a very challenging task in robotics and research has been done in order to address and improve the methods used for servoing and the grasping application in particular. Our goal is to utilize visual servoing in order to control the end-effecter of a robotic arm bringing it to a grasping position for the object of interest.

The research uses off-the-shelf 3D and image sensor, the Kinect, to provide correlated depth maps and camera images. They are used to control the robotic arm to get to a determined grasping location specified by a target image. It consists of a 2-phase control; the first phase is a feature based approach that provides a coarse alignment with the target image resulting in small errors. The second phase is a depth map error minimization based control. The latter uses the depth maps directly to perform the control task.

A defined goal scene taken from the grasping position is inputted to the system whose controller drives it to the target position. This is done via the two-phase controller. The first controller functions by matching features between the target and current images until a significant overlap between them is achieved. Only coarse alignment between the images is required for this phase. The second controller does the fine tuning. It functions by aligning the current and target depth maps in an attempt to minimize the difference in depth maps between them. The latter controller is expected to achieve minimal steady state errors in translation and rotation starting from a relatively small initial error. As a complete system, it is anticipated to function successfully even with large initial errors making it capable of handling cases where the robot is on the other side of the working space. This is achieved by preceding the main control algorithm with a coarse image alignment achieved via a feature based control which functions properly despite the initial errors.

To apply the described algorithm to a practical set up, a Microsoft Kinect camera was installed on a Barrett WAM (Whole Arm Manipulator) in an eye-in-hand configuration as depicted in Fig. 3.1.

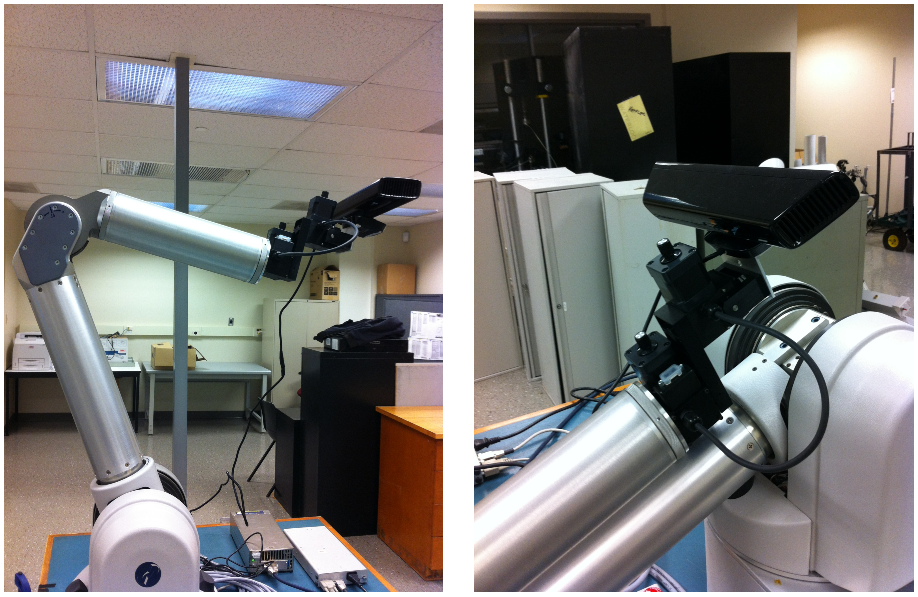


Figure 3.1: The Barrett WAM with the Kinect camera installed

## The Two-Phase Controller

In Section 2.2, we discussed the various visual servoing techniques available in the literature. Each of these visual servoing methods has its own strengths and weaknesses as discussed in Section 2.2. None of them is capable of satisfying all our needs. We need a visual servoing technique that is capable of controlling a robotic arm to reach a specified target location. The task should be accomplished with minimum complexity and using low cost sensors. It should also function properly without being restricted to particular initial conditions.

To solve the aforementioned problem, we devised the following technique. It is, essentially, a two-phase controller: The first phase is a feature-based approach that provides coarse alignment with the target image to reduce initial errors. When significant overlap is achieved, the second phase controller starts. The second phase is a depth map error minimization based control in which we adopt a strategy similar to the direct servoing approach. It uses error in depth maps between current and target images as input to the controller to arrive at the target position.

## 3.1 Phase One: Coarse Alignment

To tackle the image matching component of the first controller, this area had to be further explored and a detailed analysis of the background is described in Section 2.4.

### 3.1.1 Image Representation

As discussed in Section 2.4.2, two main categories for image understanding are:

- the bag of words (BoW) [9] and
- an unstructured list of features [32].

The BoW finds the occurrence of features in the image while the other approach that uses unstructured list of features captures general information about the image without grouping or finding relations between them. The unstructured list approach suits our application more and was the method of choice.

### 3.1.2 Feature Detector

Deciding on a feature detector is necessary because it is required to detect the features in an image to extract this information. Feature detection was discussed in details in Section 2.4.3. Researchers developed many feature detectors resulting in a large variety of detectors possessing different advantages and disadvantages. These detectors include:

- scale invariant feature transform (SIFT) [24],
- speeded up robust features (SURF) [5],
- center surround extrema (CenSurE) [1] and

- the difference of circles (DoC) [13].

For the sake of our system, accuracy is not the biggest concern since the desired outcome of this phase is only a coarse alignment between images. Choosing a fast, simple and easy to compute detector is the best choice for this task. Scale and rotation invariance leading to detecting persistent features is important too. DoC, a new method developed by computing the DoC feature detector, is fully discussed in [13]. Albeit the latter feature detector outperforms many of the common famous detectors, we decided to use the SURF algorithm because our algorithm is not yet fully optimized for speed. However, using DoC is a subject of study for future implementation.

### 3.1.3 Feature Descriptor

The next step is to decide on the suitable descriptor to be used. Many feature descriptors were studied in details and researchers have done many experiments to compare between them. The most common descriptors are:

- scale invariant feature transform (SIFT),
- speeded-up robust features (SURF) and
- histogram of oriented gradients (HoG).

For the reasons mentioned in Section 3.1.2, choosing a fast and simple descriptor is the best choice for this task. The choice fell on SURF because it's one of the most popular, easy to implement and very fast. Studying the various descriptor available and their effect on our system and thus selecting the best descriptor is a subject for future research.

### 3.1.4 Matching Features

Firstly, the SURF algorithm is applied to the target image to which the robot is required to reach and the features are stored along with their corresponding locations in the target image. After that, the SURF algorithm is applied to the images taken from the robotic arm at every later time instant resulting in finding numerous features for each of these images along with their locations in the current image.

As discussed in Section 2.4.5, two famous feature matchers are:

- brute-force matcher and
- fast library for approximate nearest neighbors (FLANN) matcher [30].

We decided to match the features in the current image with the features in the target image using the fast library for approximate nearest neighbors (FLANN) matcher because it is a much faster matcher and speed is a major concern for our work.

#### Deciding on Salient Features

To ensure the robustness of the matching, we picked the strongest matches and ignored the weaker ones which include the false positives. Choosing the most salient features was done by assigning a simple threshold to which the match strength was compared. This threshold is empirically obtained from experiments discussed in Section 5.1. Only those matches which were stronger than the specified threshold were accepted. This lead to consistent results which are necessary for deciding on the control input to the the robotic arm.

Two general cases are possible to occur and are either:

- a match existed and the matches were stored along with their locations in the image or
- there was no significant match between the two images and the vector of matches was empty.

### 3.1.5 Controlling the Robotic Arm

The locations of the features corresponding to the salient matches described in Section 3.1.4 were compared and the relative position of the current image with respect to the target one was deduced. The relative position is a direct representation of the velocities required to overlap the current image with the goal image.

Considering the cases mentioned in Section 3.1.4, the controller was faced with one of two decisions. The two modes that the controller is in are:

- ‘targeted motion mode’ in which the robot moves towards the target image using the velocities found when salient matches exist depicted in Fig. 3.2 or

- ‘search mode’ in which the robot searches its working space taking images at each time instant and searching for feature matches as shown in Fig. 3.3. Once a match is found, the robot shifts back to ‘targeted motion mode’.

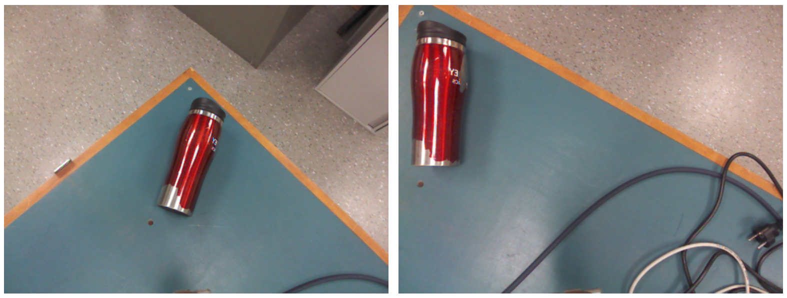


Figure 3.2: Case requiring ‘targeted motion mode’

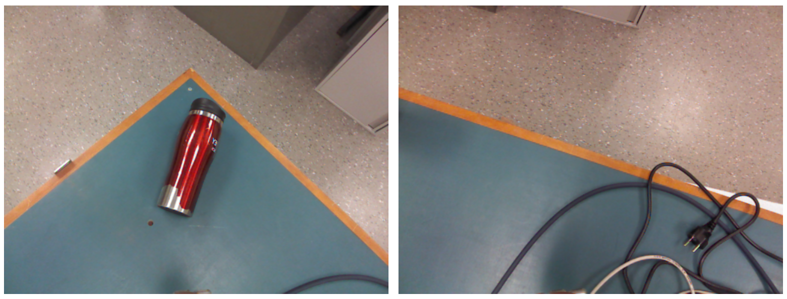


Figure 3.3: Case requiring ‘search mode’

### Stopping Criterion

The robot will stay in the coarse alignment-phase in an attempt to make the current scene viewed by the robot’s camera close to the target scene. In this phase of the controller, accuracy is not a concern. This phase’s role is only to bring the robot into the proximity of the target location. Thus, simply setting up a threshold is sufficient. The threshold specified detects when significant overlap between the two images is achieved. As discussed in Section 5.1, the threshold has to be large enough to allow the feature matching phase to function without the need for accuracy, yet sufficiently small for the second phase to be able to function properly because this threshold will directly dictate the initial conditions of the latter phase. A good value for this threshold was chosen empirically. Once the relative locations of the features in the current scene are within the specified threshold, then this is a sign for the phase-one controller to be terminated and for phase-two controller to take over.

## 3.2 Phase Two: Error Minimization

Phase-two is concerned with doing the finer job of accurately positioning the robotic arm in the specified target location. This phase is a depth map-based controller which utilizes the depth maps to directly control the robotic arm. This phase commences only when the errors between the target and the current images are small enough for a successful operation. Small errors are ensured by either starting the control task close to the target image or by implementing phase-one controller to drive the robot close to the target image. The following sections describe the details of this phase while mathematically deriving the control law which is a similar implementation of what was done in [41].

### 3.2.1 Direct Servoing Using Dense Depth Maps

For achieving the task mentioned above, dense depth maps, extracted directly from the Microsoft Kinect, are used. One advantage of using dense depth maps is its independence of any sort of matching because the whole depth map is compared with the target depth map and the difference is directly used as the input to the controller.

### 3.2.2 The Error to Minimize

To find the difference between the current and the target depth maps, a simple matrix subtraction is used where the error $e$ is defined as in (3.1):

$$\mathbf{e} = \mathbf{Z} - \mathbf{Z}^* \tag{3.1}$$

where $\mathbf{Z}$ and $\mathbf{Z}^*$ are vectors of the depth values of the current and target images respectively. We are basically interested in controlling the velocity screw $\mathbf{v}_K = (v_K, \omega_K)$ of the Kinect camera, where $v_K$ and $\omega_K$ are the instantaneous linear and rotational velocities of the

Kinect. It is obvious that $\mathbf{v}_K$ is directly related to the temporal variation of the depth $\dot{\mathbf{Z}}$. The latter relation is given by (3.2) which would serve as our control law:

$$\dot{\mathbf{Z}} = \mathbf{L}_Z \mathbf{v}_K \tag{3.2}$$

where $\mathbf{L}_Z$ is the interaction matrix related to $\mathbf{Z}$ [6] and computed in (3.7) and (3.13). The goal is to minimize $\mathbf{e}$; in order to reduce $\mathbf{e}$ exponentially (i.e. $\dot{\mathbf{e}} = -\lambda \mathbf{e}$), where $\lambda$ is just a constant, (3.2) becomes (3.3):

$$\mathbf{v}_K = -\lambda \mathbf{L}_Z^{+} \mathbf{e} \tag{3.3}$$

where $\mathbf{L}_Z^{+}$ is the pseudo-inverse of $\mathbf{L}_Z$ and is computed by (3.4):

$$\mathbf{L}_Z^{+} = (\mathbf{L}_Z^T \mathbf{L}_Z)^{-1} \mathbf{L}_Z^T \tag{3.4}$$

### 3.2.3 The Interaction Matrix

As a fundamental part of our control law, the interaction matrix has to be computed. The interaction matrix related to the depth Z is the interaction matrix of interest because depth is the measurement obtained by the Kinect depth sensor. Thus, $\mathbf{L}_Z$ is being computed in this section. Based on (3.2), we need to find $\dot{\mathbf{Z}}$. Since, $Z = Z(x, y, t)$, then the total derivative of $Z$ with respect of time is calculated in (3.5):

$$\frac{dZ}{dt} = \frac{\partial Z}{\partial x}\dot{x} + \frac{\partial Z}{\partial y}\dot{y} + \frac{\partial Z}{\partial t} \tag{3.5}$$

where $\dot{x}$ and $\dot{y}$ are the respective rate of change of $x$ and $y$, the 2D velocity of image points. Rearranging (3.5) leads to (3.6):

$$\frac{\partial Z}{\partial t} = \dot{Z} - \frac{\partial Z}{\partial x}\dot{x} + \frac{\partial Z}{\partial y}\dot{y} \tag{3.6}$$

with the depth's velocity in the camera's frame denoted by $\dot{Z} = \frac{\partial Z}{\partial t}$. Thus, at each depth point, the interaction matrix is (3.7):

$$\mathbf{L}_Z = \mathbf{L}_{P_Z} - \frac{\partial Z}{\partial x}\mathbf{L}_x + \frac{\partial Z}{\partial y}\mathbf{L}_y \tag{3.7}$$

where the interaction matrices $\mathbf{L}_x$, $\mathbf{L}_y$ and $\mathbf{L}_{P_Z}$ are such that $\dot{x} = \mathbf{L}_x \mathbf{v}_K$, $\dot{y} = \mathbf{L}_y \mathbf{v}_K$ and $\dot{Z} = \mathbf{L}_{P_Z} \mathbf{v}_K$. To find these interaction matrices, we perform the well-known 3D velocity analysis of a point $\mathbf{X}$ = (X,Y,Z) in the Kinect's frame which is described in (3.8) [6]:

$$\dot{\mathbf{X}} = -\mathbf{v}_K - \omega_K \times \mathbf{X} \Leftrightarrow \left\{ \begin{array}{lcl} \dot{X} & = & -v_x - \omega_y Z - \omega_z Y \\ \dot{Y} & = & -v_y - \omega_z X - \omega_x Z \\ \dot{Z} & = & -v_z - \omega_x Y - \omega_y X \end{array} \right. \tag{3.8}$$

where (x,y) are the depth-normalized coordinates in the image frame computed from the pixel coordinates (u,v) and the camera's intrinsic parameters as in (3.9)[6]:

$$\left\{ \begin{array}{lclcl} x & = & \frac{X}{Z} & = & \frac{(u-c_u)}{f\alpha} \\ y & = & \frac{Y}{Z} & = & \frac{(v-c_v)}{f} \end{array} \right. \tag{3.9}$$

Differentiating (3.9) with respect to time results in (3.10):

$$\left\{ \begin{array}{lclcl} \dot{x} & = & \frac{\dot{X}Z - X\dot{Z}}{Z^2} & = & \frac{\dot{X} - x\dot{Z}}{Z} \\ \dot{y} & = & \frac{\dot{Y}Z - Y\dot{Z}}{Z^2} & = & \frac{\dot{Y} - y\dot{Z}}{Z} \end{array} \right. \tag{3.10}$$

Substituting (3.8) into (3.10) results in (3.11):

$$\left\{ \begin{array}{lcl} \dot{x} & = & -\frac{v_x}{Z} + \frac{xv_z}{Z} + xy\omega_x - (1+x^2)\omega_y + y\omega_z \\ \dot{y} & = & -\frac{v_y}{Z} + \frac{yv_z}{Z} + (1+y^2)\omega_x - xy\omega_y - x\omega_z \end{array} \right. \tag{3.11}$$

Therefore, the interaction matrices in (3.7) are:

$$\begin{array}{lcl} L_x & = & [ \begin{array}{cccccc} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \end{array} ] \\ L_y & = & [ \begin{array}{cccccc} 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{array} ] \\ L_{P_z} & = & [ \begin{array}{cccccc} 0 & 0 & -1 & -yZ & xZ & 0 \end{array} ] \end{array} \tag{3.12}$$

Using the matrices computed in (3.12), the depth interaction matrix $\mathbf{L}_{Z_i}$, at any point $i$, can be computed using (3.7). To find the interaction matrix for the whole depth map of size $N$, we stack all the individual interaction matrices which results in a $N \times 6$ depth interaction matrix as shown in (3.13):

$$\mathbf{L}_Z = \begin{bmatrix} \mathbf{L}_{Z_1} \\ \mathbf{L}_{Z_i} \\ \vdots \\ \mathbf{L}_{Z_N} \end{bmatrix} \tag{3.13}$$

### 3.2.4 Noise and Incompleteness

Noise has always been an important issue to deal with in most, if not all computer vision applications. To handle noise, a simple $3 \times 3$ Gaussian filter was applied to the depth map in order to reduce the effect of noise. For additional more accurate noise removal, the empirically derived noise model found by [31] can be used. However, for this work, the $3 \times 3$ Gaussian filter sufficed for the accuracy needed. While, in order to deal with the fact that some of the measurements were incomplete or missing, we used a simple solution; we only used meaningful measurements. In other words, any measurement that was drastically skewed or invalid was just ignored. We were able to tolerate ignoring all these measurements because the entire depth map was used which had a lot of redundancy in it.

### 3.2.5 Outcome

The second phase controller continues its operation until the current depth map is almost an exact match with the target depth map resulting in a minimal steady state error.

## 3.3 Conclusion

A novel two phase controller was introduced that drives a robotic system it to a target position. The first phase-controller functions by matching features between the target and current images until a coarse alignment between them is achieved. The second-phase controller aligns the current and target depth maps by minimizing the difference between them. As a complete system, it is anticipated to handle even the cases with large initial errors. Testing the algorithm would be the best way to evaluate the performance of the algorithm as shown in Chapter 4.

# Chapter 4

# Methodology of Experiments

To put our algorithm to the test, we arranged the following setup:

## 4.1 Experimental Setup

We are using the Barrett WAM (Whole Arm Manipulator) with a Microsoft Kinect camera mounted on one of its links as shown in Fig. 3.1. This configuration is called eye-in-hand configuration. The Barrett WAM available is a 4 DoF (Degree of Freedom) arm which cannot be positioned at all valid locations in the 3D space. To gain more freedom and to avoid the limited movement of such a robot we mounted a 2 DoF PanTilt unit as depicted in Fig. 3.1. The movement of the now 6 DoF robotic arm is illustrated in Fig. 4.1. More details about the robotic arm configuration and degrees of freedom can be found in Appendix B. As explained in Chapter 3, our control algorithm solves for the velocity of the Kinect camera using (3.3). However, since a closed form solution that solves for the Kinect's DoFs could not be found, MapleSim software was used. More details about the closed form solution and the usage of MapleSim can also be found in Appendix B.

The setup is simple, no camera calibration, training or any knowledge is needed except for the target image. The Microsoft Kinect camera captures RGB-D images. Two types of images are outputted by the Kinect, color images used for the first-phase controller and depth maps used for the second-phase controller demonstrated in Fig. 4.2 (a) and (b) respectively.

The experimental setup consists of the Barrett WAM with a Microsoft Kinect camera mounted on it in an eye-in-hand configuration. A defined goal scene taken from the grasping

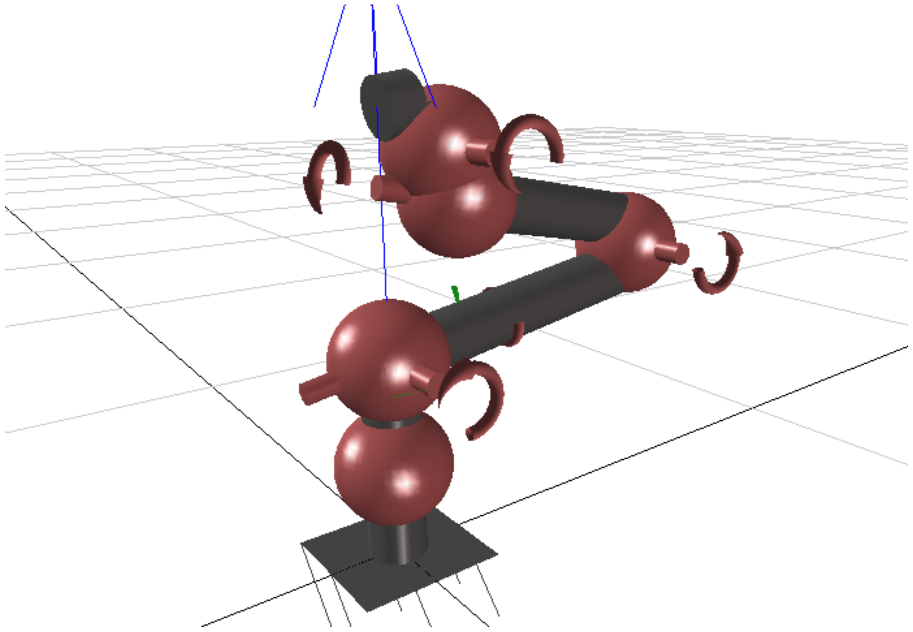
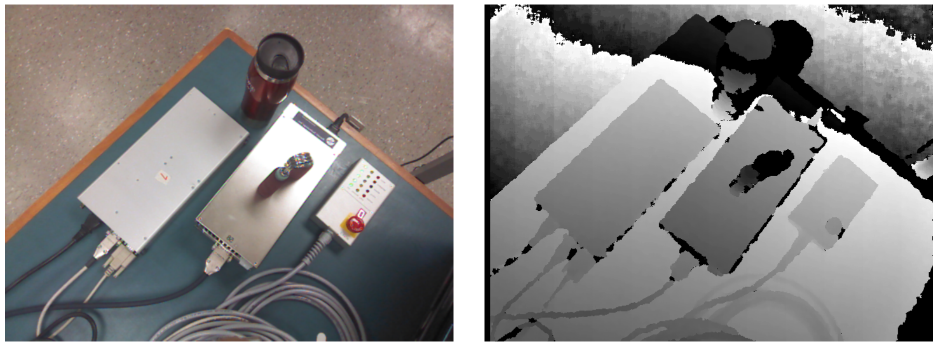


Figure 4.1: The six degrees of freedom illustrated

position is inputted to the system whose controller drives it to the target position. This is done via the two-phase controller. The first controller functions by matching features between the target and current images until a significant overlap between them is achieved. Only coarse alignment between the images is required for this phase. The second controller does the fine tuning. It functions by aligning the current and target depth maps in an attempt to minimize the difference in depth maps between them. The latter controller is expected to achieve minimal steady state errors in translation and rotation starting from a relatively small initial error. As a complete system, it is anticipated to function successfully even with large initial errors. This is achieved by preceding the main (depth map-based) control algorithm with a coarse image alignment achieved via feature based control which functions properly despite the initial errors.

## 4.2 Coding and Communicating with Barrett WAM

The theory behind the algorithm was detailed in chapter 3. To evaluate the theory, it had to be implemented. For that reason, we wrote the required code that would perform



(a) (b)

Figure 4.2: Kinect output examples: (a) Color image and (b) Depth map image

the control algorithm while receiving feedback from the sensors. The feedback used was received from images and depth maps captured by the Microsoft Kinect camera mounted on the Barrett WAM.

The coding language used was C++ because of its efficiency, speed and compatibility. Several libraries had to be included and used. The main libraries used were the ‘OpenCv’ and the ‘Microsoft SDKs’ libraries. The latter was necessary for the Microsoft Kinect to capture RGB images and depth maps which, combined, represent the feedback received. The depth maps were used by the phase-two controller while the RGB images were used by the phase-one controller. For the RGB images to be meaningful, further processing was required and for this task we used the ‘OpenCv’ library that helped implement the feature detection and matching tasks.

Before attempting the algorithm testing, we had to establish the communication environment necessary to send control commands, that were decided on by the control algorithm, to the Barrett WAM. The communication used was the ssh (Secure SHell) communication protocol and for that we included and used the ‘ssh’ library which opens a ssh session between the main computer and the Barrett WAM’s PC, creates a channel and then starts an interactive shell in it that allows information to be sent and received between the main computer and the Barrett WAM’s PC. The main computer then sends commands to the Barrett WAM’s PC through this shell in order to run a different code written in C language and that is saved on the WAM’s PC. The latter code is interactive and is responsible for controlling the various links of the WAM. After that, control commands are sent to the Barrett WAM’s PC giving it instructions on how to move. These instructions are executed via the C code run by the Barrett WAM’s PC.

Few sample pieces of code picked from the complete C++ code were included in Appendix A.

After having the prerequisites for executing the control algorithm, the algorithm was ready to be tested.

## 4.3 Experiments

As described before, our control algorithm is a two-phase controller. Therefore, we need to design the experiments in a way that tests both controllers. Thus, we devised the following experiments to evaluate the control algorithm. For all the experiments that were performed the experimental setup described above was used.

### 4.3.1 Effectiveness Against Initial Positions and Initial Errors

After getting the setup ready, the goal is to test the effectiveness of the algorithm when confronted with various situations. We came up with several cases that would cover the possible scenarios that could occur. Since, for the sake of this test particularly, we are only interested in handling initial errors, images with white backgrounds are used. Three different sets of images were used to test the ability of the robot to deal with various initial errors:

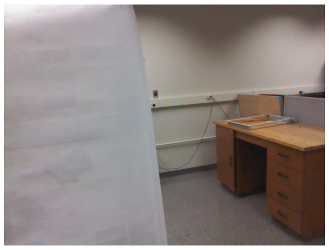
1. images close to target image, yet far enough for the depth map controller to handle,
2. images that are very far from target image with minimal or no similarity and
3. images that are close to the target image with significant overlap.

Examples of images used for the experiments described in Section 4.3.1 are shown in Fig. 4.3.

### 4.3.2 Robustness Against Environment Noise

Another common limitation that many algorithms face is environmental noise, whether it is cluttered environments or environments with more than one dominant object. It is important for good control algorithms not to fall into local minima. The goal of the

(a) Experiment 1 Example


(b) Experiment 2 Example

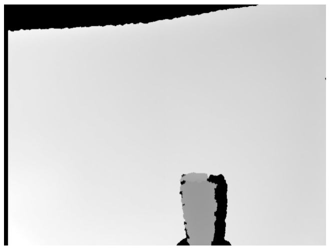

(c) Experiment 3 Example

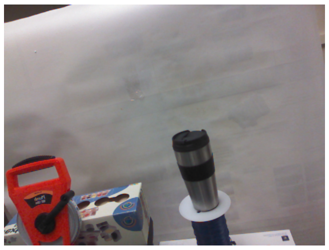
Figure 4.3: (a), (b) and (c) show examples of images used for the three experiments explained in Section 4.3.1

following experiments is to check whether our algorithm will get stuck in local minima and whether it could handle noisy environments. For this reason, another three sets of images were studied:

1. images with noisy backgrounds,
2. images with white backgrounds but containing two dominant objects and
3. images with noisy backgrounds and containing two dominant objects.

Examples of images used for the experiments described in Section 4.3.2 are shown in Fig. 4.4.

(a) Experiment 1 Example


(b) Experiment 2 Example

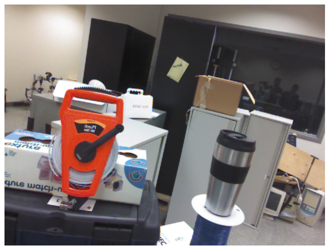

(c) Experiment 3 Example

Figure 4.4: (a), (b) and (c) show examples of images used for the three experiments explained in Section 4.3.2

# Chapter 5

# Experimental Results and Discussion

The results of the experiments described in chapter 4 are presented, analysed and discussed in this chapter.

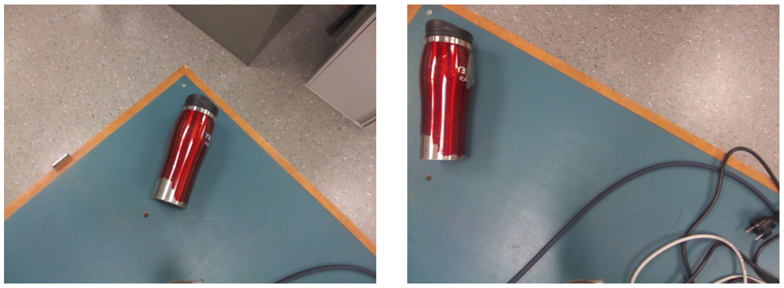
## 5.1 About Feature Detection

Before starting the actual experiments which aim at evaluating the effectiveness of our control algorithm, a simple experiment was performed to test that the controller is functioning properly especially the feature detection part of it. This experiment considered two cases:

1. a case where the some overlap between the initial and the target images exist as shown in Fig. 5.1 and

2. a case where minimal overlap exist between between the two images of interest as shown in Fig. 5.2.

In the first experiment, the current image (shown on the right (b)) is taken by the camera at some position distant, but not too far from the target image (shown on the left (a)). Features from both images were detected and then matched. As seen in Fig. 5.1, the strongest features detected were the ones detected on the object. The algorithm successfully detected the match and commanded the robotic arm to move accordingly towards the target image.



(a) Target Image (b) Current Camera Scene

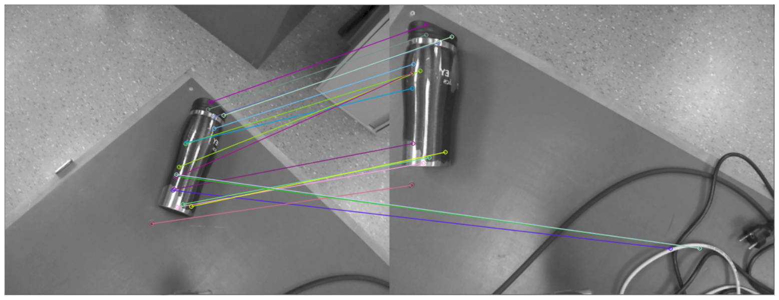
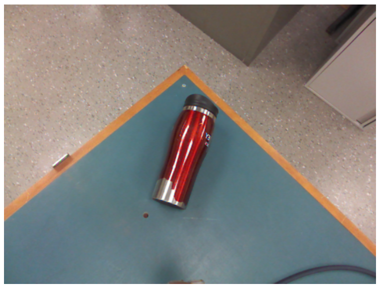


Figure 5.1: Example of initial image close to target image showing good feature matches

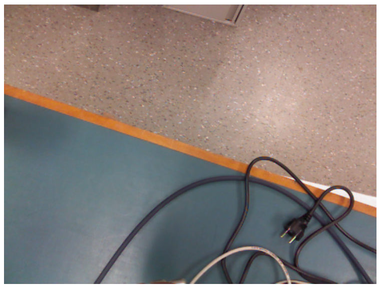
As for the second experiment, the current image (shown on the right (b)) is taken by the camera at a different position that is also distant, but, this time, far from the target image (shown on the left (a)). Again Features from both images were detected and then matched. It is shown in Fig. 5.2 that even the strongest features matched were still very weak to represent any true overlap between the two images. The feature matches were assessed to be weak based on the empirically obtained threshold used in section3.1.4. Setting this threshold is further discussed in Section 5.4.2. The algorithm was able to detect the lack of good matches and commanded the robot to go into search mode described in Section 3.1.5. This is only a general test for the phase-one controller described to show how it works and to visualize the feature detection and matching steps.

## 5.2 Initial Positions and Initial Errors

The main contribution of this work is its capability to control the robotic arm without being limited to scenarios with small initial errors since we argue that, for such a control algorithm to be practical and applicable to many industrial and real life applications, it has



(a) Target Image



(b) Current Camera Scene

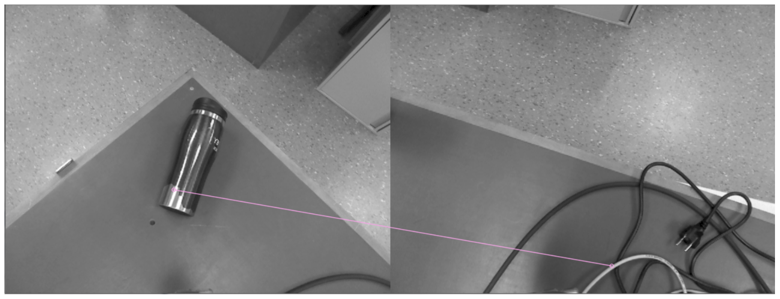


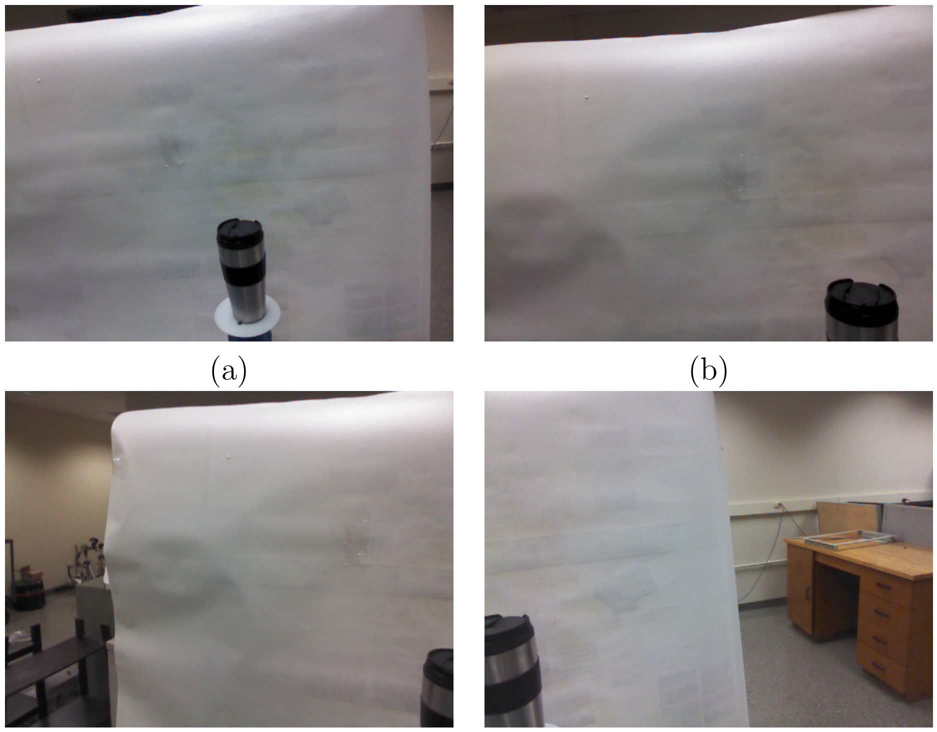
Figure 5.2: Example of initial image far from target image where good feature matches are absent

to have a large range of functionality and not just limited to small error minimizations. As explained previously in chapter 3, and because our algorithm is composed of a two-phases controller, it reacts differently to different situations as it has to decide on which controller-phase to engage and to smoothly and seamlessly shift between the two phases. This allows our algorithm to handle the large range of initial conditions and the variety of situations. Having this in mind helped put together the experiments described in Section 4.3.1. All the experiments in this section used white background with only one dominant object because the goal of these experiments is to test the effectiveness of our algorithm as the initial position of the robot changes and we are not interested in the effect of environmental changes on the robustness of the algorithm which will be addressed in Section 5.3.

### 5.2.1 Experiment 1:

The first experiment deals with images that are not too far from the target image yet not too close. The overlap between the images at the current camera location and the target image was not sufficient for the depth map controller to handle. This decision is based on a threshold checking step. First, the distance between the two images is roughly estimated after quick feature detection and matching steps. Then, this distance is compared against a specified threshold. If the distance is lower than the threshold distance then the depth map error minimization (phase-two) controller is called. However, if the distance is higher than the threshold, then the feature-based (phase-one) controller starts attempting to minimize the error below the threshold after which the phase-two would take over and continue moving the robotic arm towards the target location. The threshold is chosen in a way that is big enough for our coarse alignment (phase-one) to operate successfully, yet small enough to be within the range of operation of our phase-two controller. Thus, a threshold of 10 cm was chosen.

As expected, since the initial camera position was sufficiently far from the target position, the phase-one controller commenced its operation and commanded the Barrett WAM in the direction towards reducing the relatively large initial errors. Once the error reached the chosen threshold, the phase-two controller took over and the latter error was minimized via the direct depth map error minimization algorithm until the steady state target location was achieved. The details of the last step and more quantitative results are further discussed in experiment 3 in Section 5.2.3. Few examples of some of the experimental cases that were tested are shown in Fig. 5.3.

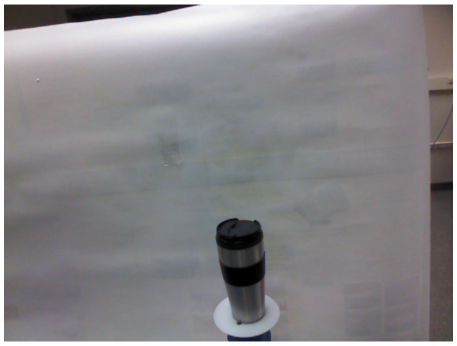


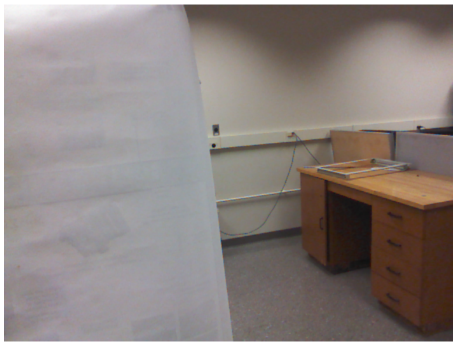
(a) (b) (c) (d)

Figure 5.3: Example of the images captured by the Kinect for several cases in experiment 1 showing the initial state at which the experiment started. (a) shows the target image while (b), (c) and (d) show different images representing several initial positions with relatively large initial errors

### 5.2.2 Experiment 2:

The second set of experiments were performed starting from initial positions far from the target location. The images taken from these locations were very far from the target image showing minimal or no similarity. The algorithm discovered this lack of similarity and asked the robot to enter 'search mode' which requires the robot to move around in the its workspace until some sufficient match between the current image and the target image is sensed after which the feature matching controller takes over. The phase-one controller continued and the remainder of this experiment was continued in a fashion similar to experiment 1. Few examples of some of the experimental cases that resulted in the Barrett WAM behaving as described above are shown in Fig. 5.4.


(a)


(b)

(c)

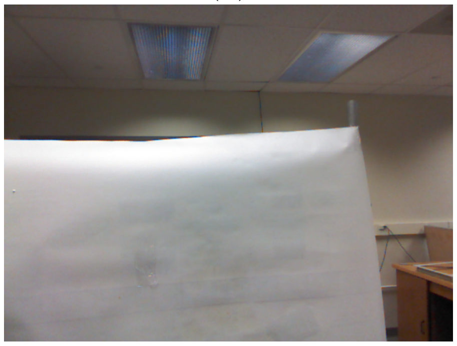

(d)

Figure 5.4: Example of the images captured by the Kinect for several cases in experiment 2 showing the initial state at which the experiment started. (a) shows the target image while (b), (c) and (d) show different images representing several initial positions with very large initial errors

### 5.2.3 Experiment 3:

The main goal of the third experiment is to make sure that the algorithm is able to handle the various situations including the special case that requires the phase-two controller to start and finish without needing the phase-one controller. Even though this particular case was tested in experiments 1 and 2 of this section as part of the whole controller, we left the discussion of the details of the results to this section. Fig. 5.5 shows few depth map images taken by the Kinect at various time instants during the execution of the commands sent to the robotic arm in one of the performed experiments. The image sequence in Fig. 5.5 shows that the algorithm functioned successfully minimizing the initially not too large error. The algorithm was able to function successfully for the set of experiments performed in which the initial error did not exceed the specified threshold of 10 cm discussed in Section 5.2.1. In these experiments, the steady state error we achieved was less than 5 mm in translation and 1 deg in rotation. The graphs in Fig. 5.6 show the translational and rotational errors being reduced from the initial errors until the steady state is achieved.

### 5.2.4 Conclusion

The experiments presented in Section 5.2 showed that our proposed algorithm handles a large range of initial positions and was capable of using the two phase controller to drive the Barrett WAM to the target location.

## 5.3 Environmental Noise

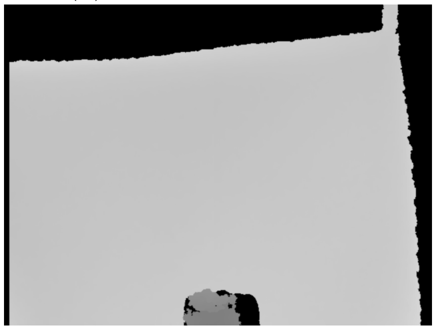
In our algorithm, which was detailed in chapter 3, we performed two simple but effective measures to deal with image noise and environment clutter. The first measure was applying a $3 \times 3$ Gaussian filter to the depth maps before processing them. The second measure was ignoring all the outliers and considering only the depth values of the pixels that had depth measurements in both the current and target depth maps. To test how robust our algorithm is against environmental irregularities after incorporating these measures, we performed the experiments described in Section 4.3.2.

### 5.3.1 Experiment 1:

This set of experiments considers the case when the background is not controlled resembling a more realistic scenario that the robot might face. To simulate that, we tested our

(a) Target Depth map

(b) Depth map at some time


(c) Depth map at a later time

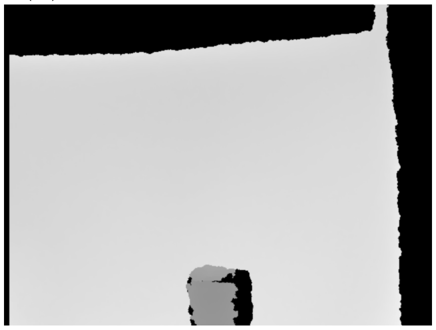
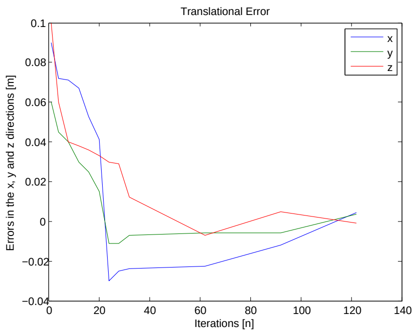
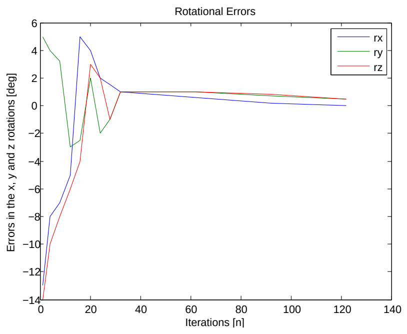

(d) Depth map at steady state

Figure 5.5: Example of the depth maps captured by the Kinect during experiment 3 showing the progression of the algorithm until it reaches steady state



(a) Translational Errors



(b) Rotational Errors

Figure 5.6: (a) and (b) show the progression of the translational and rotational errors plotted against the iteration number starting from the initial position until the steady state was reached

algorithm by using our lab's environment as the background for the taken images while placing a dominant object in the foreground. A couple of examples of the tested cases are shown in Fig. 5.7.

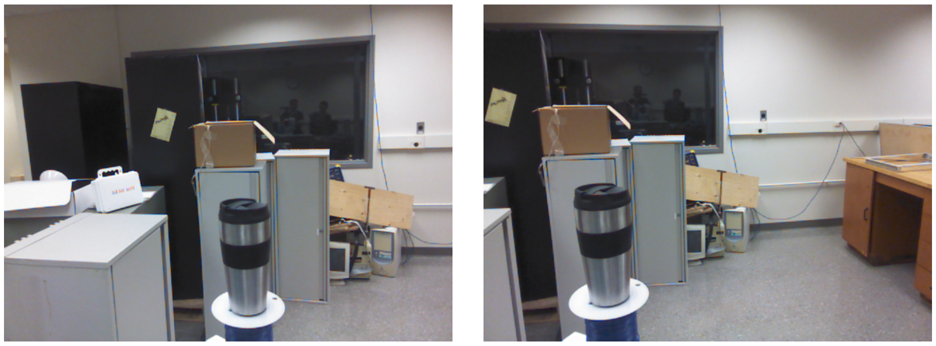

(a) (b)

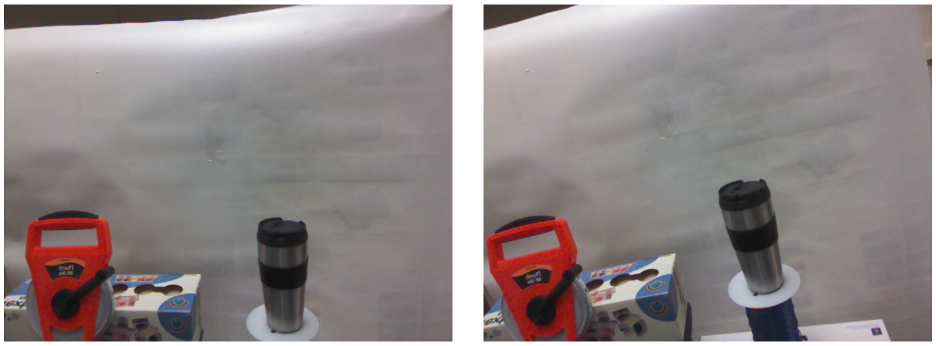
Figure 5.7: Example of the images captured by the Kinect for cases in experiment 1 showing the noisy background. (a) and (b) show two different examples of images representing cases with noisy backgrounds and containing one dominant object

The results obtained from these images were consistent with the results of the experiments performed in Section 5.2 and the robot was able to successfully move itself to reach the target location. The reasons behind the robustness against noise are two, the Gaussian filtering performed and the presence of a dominant object that ensures the existence of strong feature matches.

### 5.3.2 Experiment 2:

The second set of experiments performed tackled images that contained two dominant objects to test if our algorithm is capable of avoiding getting trapped in local minima. In this experiment, the backgrounds were chosen to be white in order to evaluate the local minima issue separately. Fig. 5.8 shows two examples of images taken by the robot representing this experiment's conditions.

The results for this experiment were also successful, and the algorithm didn't fall in local minima because it was able to properly match the features between the objects. The only extra limitation faced was that we had to adjust the threshold mentioned earlier in Section 5.1.

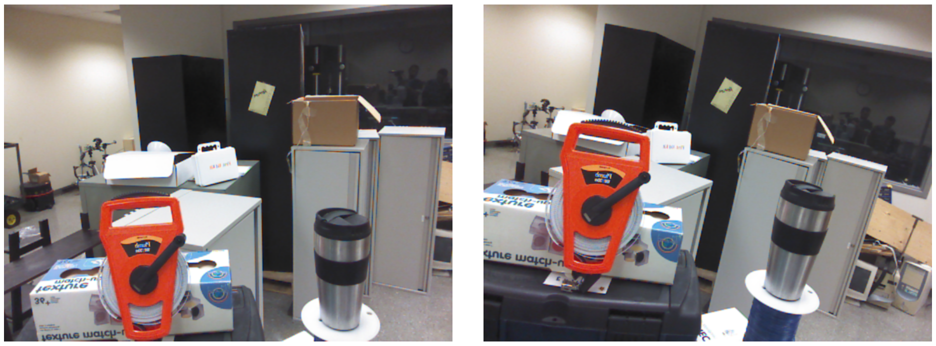


(a) (b)

Figure 5.8: Example of the images captured by the Kinect for cases in experiment 2. (a) and (b) show two different examples of images representing cases with white backgrounds but containing two dominant objects

### 5.3.3 Experiment 3:

The final experiment performed tested the combined effects of the first two experiments together. For this experiment we used images with noisy backgrounds and containing two dominant objects. Again, two examples of images taken by the robot representing this experiment's conditions are shown in Fig. 5.9.



(a) (b)

Figure 5.9: Example of the images captured by the Kinect for cases in experiment 2 showing the noisy background. (a) and (b) show two different examples of images representing cases with white backgrounds but containing two dominant objects

The results for this experiment were similar to those of experiment 2 explained in Section 5.3.2. Adjusting the threshold for the feature matching strength led to the system being successfully controlled without falling into local minima.

### 5.3.4 Conclusion

The experiments conducted in this section showed the robustness of our algorithm against environment noise, thanks to the 3×3 Gaussian filter and to the outlier rejection process.

## 5.4 Limitations

Despite the promising results produced by the proposed algorithm, we encountered some limitations which open a window of opportunity to improve on the algorithm. We mention here the two main limitations that we faced.

### 5.4.1 Rotational Limitations for Phase-One Controller

In the feature-based phase-one controller, the algorithm only deals with translational error reductions and does not attempt at reducing rotational errors. This could be a problem if the initial rotational error is too big for the phase-two controller to handle. The latter controller proved to function effectively with rotational errors up to 20 deg. Therefore, to be able to control scenarios with initial rotational errors larger than 20 deg, the phase-one controller has to be modified to be capable of reducing rotational errors. Again, only rough rotational alignment to an intermediate error below 20 deg is sufficient because phase-two controller is capable of handling such relatively small errors.

### 5.4.2 Manual Thresholding

Another limitation, that hiders this work from being completely applicable as a standalone platform for any environment, is that it requires specifying a threshold (mentioned in Section 5.1) for the phase-one controller that is used to decide on which mode to put the robotic arm in, namely 'targeted motion mode' or 'search mode'. This threshold was empirically chosen after performing several experiments (not explained in this work) and looking at the relevant data.

A good way to elevate this limitation is to come up with a different criterion to help the algorithm make the distinction, or make the algorithm smart enough in order to be capable of estimating this threshold autonomously.

On the other hand, the threshold specified for transitioning from phase-one to phase-two (explained in Section 5.2.1) is not a problem and can be extended to any situation because the phase-two controller proved to be successful for the cases with errors below that threshold.

# Chapter 6

# Conclusion and Future Recommendations

## 6.1 Conclusion

This thesis presented a novel two-phase control algorithm that was used to successfully position the end effector of the Barrett WAM at a target location in the vicinity of an object of interest. Visual sevoing methods were implemented to execute this task. Particularly, a coarse feature matching technique helped approach the object of interest. A second, more fine, control algorithm was used to match the current scene, captured by the Kinect camera mounted on the robotic arm, with a target one. Depth maps from the scenes of interest were matched in order to perform the control motion.

Lots of research has been done to improve on the visual servoing processes. In all the work that we came across, 3D information was estimated using homography, stereo vision or was just assumed to be known. Our work managed to overcome this necessity and used the Kinect's depth maps to get the 3D information which made our system more robust, less sensitive to calculation errors and more accurate.

To evaluate the effectiveness of our technique, several experiments were performed in an attempt to position a WAM robotic arm in the target position close to an object of interest. Eye-in-hand configuration was the choice for mounting the camera and the sensor used was the Microsoft Kinect which outputs both the image and the depth map of the scene being visualized.

The results of the experiments performed showed that the algorithm worked successfully and was able to achieve accurate control with minimal steady state errors despite large initial errors or environmental noise. However, very large rotational errors were not dealt with; for that cause, the algorithm should be further improved to handle such errors.

## 6.2 Future Recommendations

The very first and direct improvement on this thesis work is to extend the algorithm in such a way that would make it capable of handling very large initial rotational errors. This could be done in a way similar to the way our algorithm handled the large translational errors. Using rotationally sensitive features in the coarse image alignment phase is a possible solution for this necessity. This extension would allow the applicability of the algorithm to an even broader range of real life and industrial applications.

Another recommendation for future enhancement would be to better remove the noise from the Kinect depth maps by replacing the $3 \times 3$ Gaussian filter with a more accurate filter that accounts for the Kinect's built-in noise. Better noise removal will naturally lead to improved accuracies in the results obtained.

The next step would be to add the grasping process and complete the mission of the application of interest. Automatically determining the target location is the first improvement to implement. Later, grasping can also be improved by taking the objects' compliance into consideration. These are natural extensions for the current work, but they are very involved and demanding tasks that require new research and experimentation.

# References

[1] M. Agrawal, K. Konolige, and M. Blas. Censure: Center surround extremas for realtime feature detection and matching. *Computer Vision–ECCV 2008*, pages 102–115, 2008.

[2] P.K. Allen, B. Yoshimi, and A. Timcenko. Real-time visual servoing. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 851–856. IEEE, 1991.

[3] Barrett. Four Degrees of Freedom Barrett Whole Arm Manipulator. *4DoFWAM*, 2005. http://web.barrett.com/supportFiles/FramesAndJoints/B2731_RevAA-00.pdf.

[4] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008.

[5] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[6] F. Chaumette and S. Hutchinson. Visual servo control, part i. basic approaches. *Robotics & Automation Magazine, IEEE*, 13(4):82–90, 2006.

[7] F. Chaumette and S. Hutchinson. Visual servo control, part ii. advanced approaches [tutorial]. *Robotics & Automation Magazine, IEEE*, 14(1):109–118, 2007.

[8] F. Chaumette and E. Malis. 2 1/2 d visual servoing: a possible solution to improve image-based and position-based visual servoings. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, volume 1, pages 630–635. IEEE, 2000.

[9] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision (ECCV)*, volume 1, page 22, 2004.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

[11] U. Frese, B. Bauml, S. Haidacher, G. Schreiber, I. Schaefer, M. Hahnle, and G. Hirzinger. Off-the-shelf vision for a robotic ball catcher. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1623–1629. IEEE, 2001.

[12] T. Hamel and R. Mahony. Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Transactions on Robotics and Automation*, 18(2):187–198, 2002.

[13] A. Hojaij, A. Fakih, A.and Wong, and J. Zelek. Difference of circles feature detector. In *Ninth Conference on Computer and Robot Vision (CRV)*, pages 63–69. IEEE, 2012.

[14] A. Howard. Real-time stereo visual odometry for autonomous ground vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3946–3952. IEEE, 2008.

[15] K. Hsiao, S. Chitta, M. Ciocarlie, and E.G. Jones. Contact-reactive grasping of objects with partial shape information. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 10, page 2010, 2010.

[16] K. Hsiao, P. Nangeroni, M. Huber, A. Saxena, and A.Y. Ng. Reactive grasping using optical proximity sensors. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2098–2105. IEEE, 2009.

[17] S. Hutchinson, G.D. Hager, and P.I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[18] Y. Jiang, JR Amend, H. Lipson, and A. Saxena. Learning hardware agnostic grasps for a universal jamming gripper. In *International Conference on Robotics and Automation (ICRA)*, pages 2385–2391. IEEE, 2012.

[19] I. Kamon, T. Flash, and S. Edelman. Learning to grasp using visual information. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pages 2470–2476. IEEE, 1996.

[20] K. Konolige. Star Detector. *Opencv*, 2011. http://pr.willowgarage.com/wiki/Star_Detector.

[21] D. Kragic, H.I. Christensen, et al. Survey on visual servoing for manipulation. *Computational Vision and Active Perception Laboratory, Fiskartorpsv*, 15, 2002.

[22] M. Krainin, P. Henry, X. Ren, and D. Fox. Manipulator and object tracking for in-hand 3d object modeling. *The International Journal of Robotics Research*, 30(11):1311–1327, 2011.

[23] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2169–2178. IEEE, 2006.

[24] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[25] E. Malis, F. Chaumette, and S. Boudet. 21/2 d visual servoing. *IEEE Transactions on Robotics and Automation*, 15(2):238–250, 1999.

[26] BS Manjunath, C. Shekhar, and R. Chellappa. A new approach to image feature detection with applications. *Pattern Recognition*, 29(4):627–640, 1996.

[27] Z.C. Marton, D. Pangercic, N. Blodow, J. Kleinehellefort, and M. Beetz. General 3d modelling of novel objects from a single view. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3700–3705. IEEE, 2010.

[28] C. Matuszek, B. Mayton, R. Aimi, M.P. Deisenroth, L. Bo, R. Chu, M. Kung, L. LeGrand, J.R. Smith, and D. Fox. Gambit: An autonomous chess-playing robotic system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4291–4297. IEEE, 2011.

[29] A.T. Miller and P.K. Allen. Graspit! a versatile simulator for robotic grasping. *Robotics & Automation Magazine, IEEE*, 11(4):110–122, 2004.

[30] M. Muja and D. Lowe. Flann-fast library for approximate nearest neighbors user manual, 2009.

[31] C. V. Nguyen, S. Izadi, and D. Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *Second International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT)*, pages 524–530. IEEE, 2012.

[32] A. Oliva. Gist of the scene. *Neurobiology of attention*, 696:64, 2005.

[33] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006.

[34] N.P. Papanikolopoulos, P.K. Khosla, and T. Kanade. Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision. *IEEE Transactions on Robotics and Automation*, 9(1):14–35, 1993.

[35] P.E.I. Pounds, D.R. Bersak, and A.M. Dollar. Grasping from the air: Hovering capture and load stability. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2491–2498. IEEE, 2011.

[36] A. Saxena, J. Driemeyer, J. Kearns, and A.Y. Ng. Robotic grasping of novel objects. In *Neural Information Processing Systems*. Citeseer, 2006.

[37] A. Saxena, J. Driemeyer, J. Kearns, C. Osondu, and A.Y. Ng. Learning to grasp novel objects using vision. In *Experimental Robotics*, pages 33–42. Springer, 2008.

[38] A. Saxena, J. Driemeyer, and A.Y. Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[39] A. Saxena, L. Wong, and A.Y. Ng. Learning grasp strategies with partial shape information. In *AAAI*, volume 8, pages 1491–1494, 2008.

[40] A. Saxena, L. Wong, M. Quigley, and A.Y. Ng. A vision-based system for grasping novel objects in cluttered environments. *Robotics Research*, pages 337–348, 2011.

[41] C. Teuliere and E. Marchand. Direct 3d servoing using dense depth maps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1741–1746. IEEE, 2012.

[42] N. Vahrenkamp. Humanoid. *High Performance Humanoid Technologies*, 2010. http://his.anthropomatik.kit.edu/img/planningVS_left.png.

[43] Francisco Vina. 2.5d visual servoing of a 7 dof manipulator. *KTH Computer Science and Communication*, 2011.

[44] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–511. IEEE, 2001.

[45] B. Wang, L. Jiang, JW Li, and HG Cai. Grasping unknown objects based on 3d model reconstruction. In *Proceedings of IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 461–466. IEEE, 2005.

[46] W. J. Wilson, C. Williams H., and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5):684–696, 1996.

[47] K. Yamazaki, M. Tomono, T. Tsubouchi, and S. Yuta. A grasp planning for picking up an unknown object for a mobile manipulator. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pages 2143–2149. IEEE, 2006.

[48] DB Zhang, L. Van Gool, and A. Oosterlinck. Stochastic predictive control of robot tracking systems with dynamic visual feedback. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 610–615. IEEE, 1990.

# APPENDICES

# Appendix A

# WAM Communication Code

Below is the 'main.cpp' C++ file that runs the control algorithm:

```
#include <stdio.h>
#include <iostream>
#include <fstream>

#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
#include "opencv2/nonfree/features2d.hpp"
#include "opencv2/imgproc/imgproc.hpp"

#include "ssh Folder/ssh_Send.h"
#include "Kinect Folder/KinectWrapper.h"

#define feature_matching 1
#define depthmap_matching 2


float* SIFT_Kinect(KinectWrapper, cv::SurfFeatureDetector, cv::Mat, {USHORT *,} cv::Mat,
USHORT*, std::vector<cv::KeyPoint>, cv::Mat, int type);

/** @function main */
```

```
int main()
{
/////////////////////////
// Grabbing OBJECT image:
/////////////////////////
KinectWrapper myKinect;

/* 480x640 Image*/
int rows = 480;
int cols = 640;

unsigned int* color;
color = myKinect.GetImage_color();

cv::Mat img_object;
cv::Mat img_object_Gray;
cv::Mat img_scene_temp;
img_scene_temp = cv::Mat(rows, cols, CV_8UC4, (uchar*)color);
cv::cvtColor( img_scene_temp, img_object_Gray, CV_RGB2GRAY );
img_object = img_object_Gray;
USHORT* object_depth = myKinect.GetImage_depth();

//**********************************************
// Saving Depth Image
//**********************************************
cv::Mat scene_depth2;
float factor;
factor = 255.0/3000.0;
USHORT * scene_depth1;
float * scene_depth_temp = new float[480*640];
scene_depth1 = myKinect.GetImage_depth();

for (int i = 0; i<480*640; i++)
{
scene_depth_temp[i] = scene_depth1[i];//*factor;
}

img_scene_temp = cv::Mat(rows, cols, CV_32FC1, scene_depth_temp);
```

```
scene_depth2 = img_scene_temp;

std::ofstream output_depth("output_depth_new.txt");
for (int i = 0; i<480; i++)
{
float* Mi = scene_depth2.ptr<float>(i);
for (int j = 0; j<640; j++)
{
output_depth<<Mi[j]<<std::endl;
}
}

//– Step 1: Detect the keypoints using SURF Detector
int minHessian = 400;

cv::SurfFeatureDetector detector( minHessian );

std::vector<cv::KeyPoint> keypoints_object;

detector.detect( img_object, keypoints_object );

//– Step 2: Calculate descriptors (feature vectors)
cv::SurfDescriptorExtractor extractor;

cv::Mat descriptors_object;

extractor.compute( img_object, keypoints_object, descriptors_object );


/************************************
Initializing ssh session:
************************************/
std::string Host;
std::cout<<"Who is your Host: ";
std::getline(std::cin, Host);

sshStruct sshStruct_session;
sshStruct_session = open_session (Host);
```

```
sshStruct_session.rc = ssh_channel_open_session(sshStruct_session.channel);
if (sshStruct_session.rc < 0)
{
close_session (sshStruct_session);
return 1;
}

sshStruct_session.rc = open_interactive_shell_session(sshStruct_session.channel);
if (sshStruct_session.rc < 0)
{
printf("error : %s\n",ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
/************************************
Calling the interactive shell:
************************************/
if (sshStruct_session.rc < 0)
{
printf("error : %s\n",ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}

/************************************
Initializing Control Parameters:
************************************/
int Pixel_x = 600/2;//center of image in x
int Pixel_y = 363/2;//center of image in y
int tolerance = 20;
double vx = 0, vy = 0;//endpoint velocities
float px_scene = 0, py_scene = 0, px_obj = 0, py_obj = 0;
float* px_py_scene = new float[2];
float* vx_vy_scene;// = new float[6];
cv::Mat img_scene_Gray;
USHORT * scene_depth;
scene_depth = object_depth;
```

```
int type = 0;

std::string cmd;
int z = 0;

sshStruct_session.rc = interactive_shell_session_send(sshStruct_session.channel,cmd);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
sshStruct_session.rc = interactive_shell_session_receive(sshStruct_session.channel);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}

type = depthmap_matching; // ”feature_matching” for feature matching and ”depthmap_matching”
for depthmap matching

/*****************************************
Running the C code written on the WAM’s PC:
*****************************************/
cmd =”cd btclient/src/btdiag”;
cmd+=”˚”;
sshStruct_session.rc = interactive_shell_session_send(sshStruct_session.channel,cmd);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
cmd =”./btdiag”;
cmd+=”˚”;
sshStruct_session.rc = interactive_shell_session_send(sshStruct_session.channel,cmd);
```

```
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}

while(z<3)
{
z++;
//–Start Velocity Controller:

/************************************
Getting Object Loacation:
************************************/

if (type == feature_matching)
{
color = myKinect.GetImage_color();

std::cout<<”done reading image”;
img_scene_temp = cv::Mat(rows, cols, CV_8UC4, (uchar*)color);
cv::cvtColor( img_scene_temp, img_scene_Gray, CV_RGB2GRAY );

px_py_scene = SIFT_Kinect(myKinect, detector, img_scene_Gray, scene_depth, img_object,
object_depth, keypoints_object, descriptors_object, type);
cv::waitKey(0);
px_scene = px_py_scene[0];
py_scene = px_py_scene[1];
px_obj = px_py_scene[2];
py_obj = px_py_scene[3];

/************************************
Executing Simple Controller:
************************************/
printf(”– Min Index x : %f \n”, px_scene);
printf(”– Min Index y : %f \n”, py_scene);
```

```
if (px_scene == 0 —— py_scene == 0)
{
std::cout<<"No match detected; search mode: \n";
vx = 0.5;
vy = 0.5;
}
else
{
std::cout<<"Centering object: \n";
// Image xy-coordinate system (0,0) starts at top left corner
if (px_scene <=px_obj - tolerance)
{
vx = 0.2;
}
else if (px_scene >=px_obj + tolerance)
{
vx = -0.2;
}
else
{
vx = 0;
}
if (py_scene <=py_obj - tolerance)
{
vy = 0.2;
}
else if (py_scene >=py_obj + tolerance)
{
vy = -0.2;
}
else
{
vy = 0;
}
}
}
else if (type == depthmap_matching)
{
```

```
cv::Mat scene_depth2;
float factor;
factor = 255.0/3000.0;
scene_depth = myKinect.GetImage_depth();
uchar * scene_depth_temp = new uchar[480*640];

for (int i = 0; i<480*640; i++)
{
scene_depth_temp[i] = scene_depth[i];
}

img_scene_temp = cv::Mat(rows, cols, CV_8UC1, scene_depth_temp);
scene_depth2 = img_scene_temp;

cv::imshow(”depth”,scene_depth2);
cv::waitKey(0);

vx_vy_scene = SIFT_Kinect(myKinect, detector, img_scene_Gray, scene_depth, img_object,
object_depth, keypoints_object, descriptors_object, type);
vx = vx_vy_scene[0];
vy = vx_vy_scene[1];
}
if (vx == 0 && vy == 0)
{
type == depthmap_matching;
}

printf(”– Velocity in the x : printf(”– Velocity in the y :
/**************************************************
Calling the interactive shell For Executing Controller:
**************************************************/
cmd =”M”; //Asking the WAM to be ready for the move command
cmd+=”˚”;
sshStruct_session.rc = interactive_shell_session_send(sshStruct_session.channel,cmd);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
cmd =”vx,vy”; //Sending the values for the move command
cmd+=”˚”;
sshStruct_session.rc = interactive_shell_session_send(sshStruct_session.channel,cmd);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
if (z%5==0)
{
sshStruct_session.rc = interactive_shell_session_receive(sshStruct_session.channel);
if (sshStruct_session.rc < 0)
{
printf(”error : %s\n”,ssh_get_error(sshStruct_session.session));
close_session (sshStruct_session);
return 1;
}
}

}
/***********************************
Closing ssh session:
***********************************/
close_session (sshStruct_session);

cv::waitKey(0);

return 0;
}
```

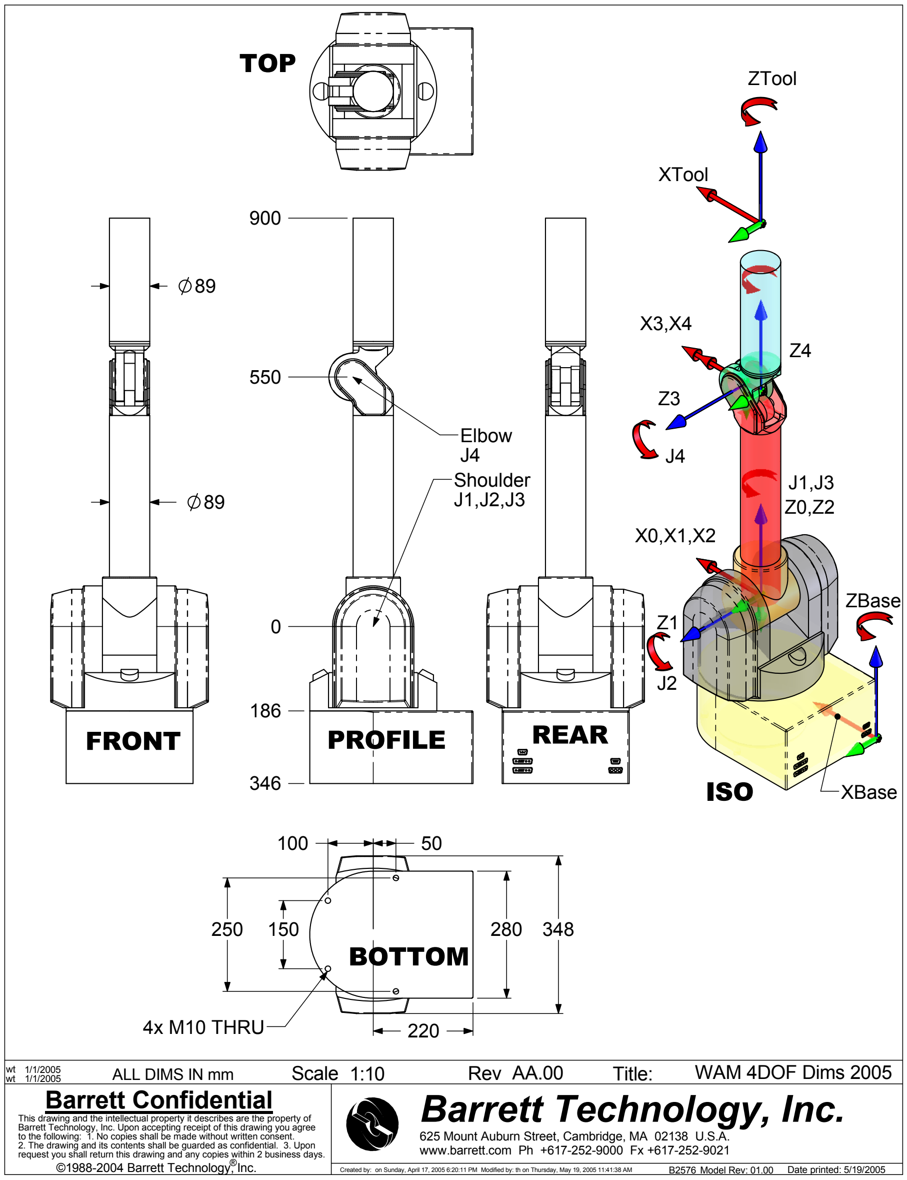
# Appendix B

# Using MapleSim

As mentioned earlier in Section 4.1, the robotic arm used is the 4 DoF Barrett WAM (shown in Fig. B.1) with a 2 DoF PanTilt unit mounted, as an extension, on its last link to form a 6 DoF robotic arm. The described robotic arm is shown in Fig. B.2.

A closed form solution for the whole system could not be found because of redundancies available in the configuration of the robotic arm. To overcome this issue, a MapleSim code was generated which solves for the degrees of freedom to be input to the robotic arm.

The C++ code included in Appendix A is executed first and, at each iteration, the next required position for the robotic arm is computed. The latter position is used as an input to MapleSim which solves for the six degrees of freedom of the robotic arm. These values are fed to the robotic system to control its six joint motors in order to position it at the required position. The MapleSim code is depicted in Fig. B.3, Fig. B.4, and Fig. B.5.



wt 1/1/2005
wt 1/1/2005
ALL DIMS IN mm Scale 1:10 Rev AA.00 Title: WAM 4DOF Dims 2005

**Barrett Confidential**

This drawing and the intellectual property it describes are the property of Barrett Technology, Inc. Upon accepting receipt of this drawing you agree to the following: 1. No copies shall be made without written consent. 2. The drawing and its contents shall be guarded as confidential. 3. Upon request you shall return this drawing and any copies within 2 business days.

©1988-2004 Barrett Technology® Inc.

**Barrett Technology, Inc.**
625 Mount Auburn Street, Cambridge, MA 02138 U.S.A.
www.barrett.com Ph +617-252-9000 Fx +617-252-9021

Created by: on Sunday, April 17, 2005 6:20:11 PM Modified by: th on Thursday, May 19, 2005 11:41:38 AM B2576 Model Rev: 01.00 Date printed: 5/19/2005

Figure B.1: Four DoF Barrett WAM [3]

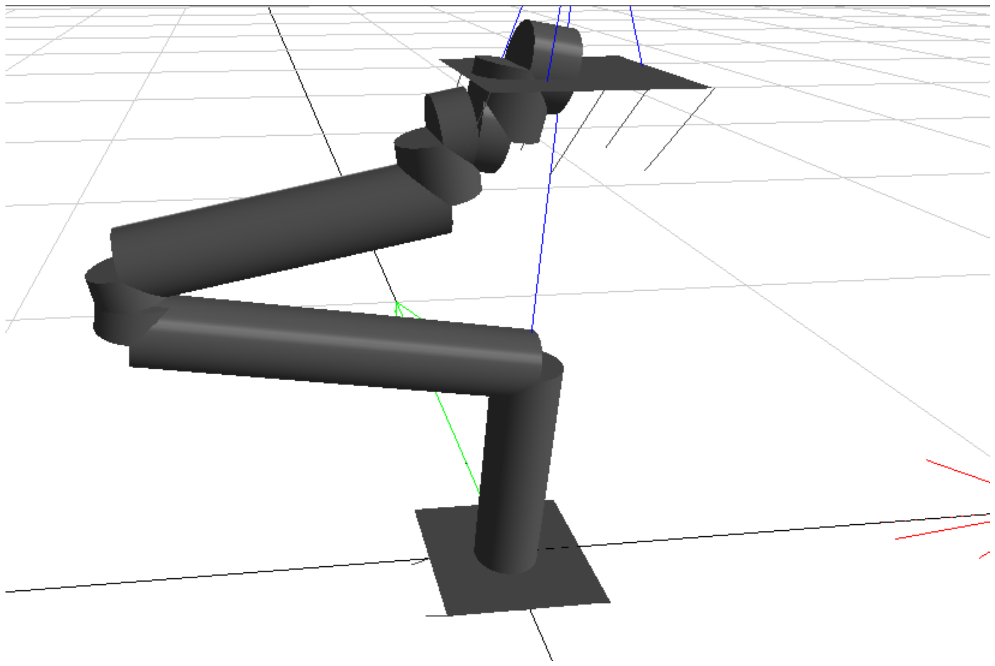


Figure B.2: The six degrees of freedom robotic arm showing the 4 DoF Barrett WAM with the 2 DoF PanTilt mounted

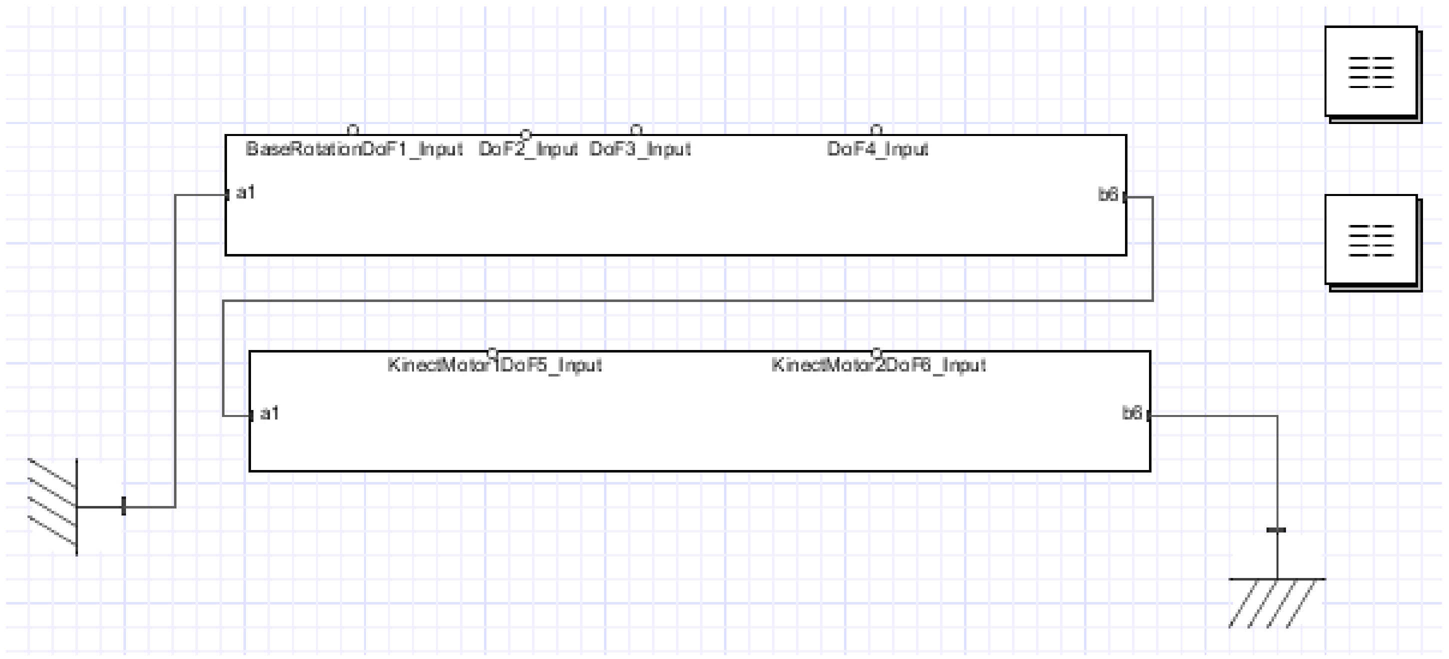


Figure B.3: The MapleSim main overall code containing the code subsystems

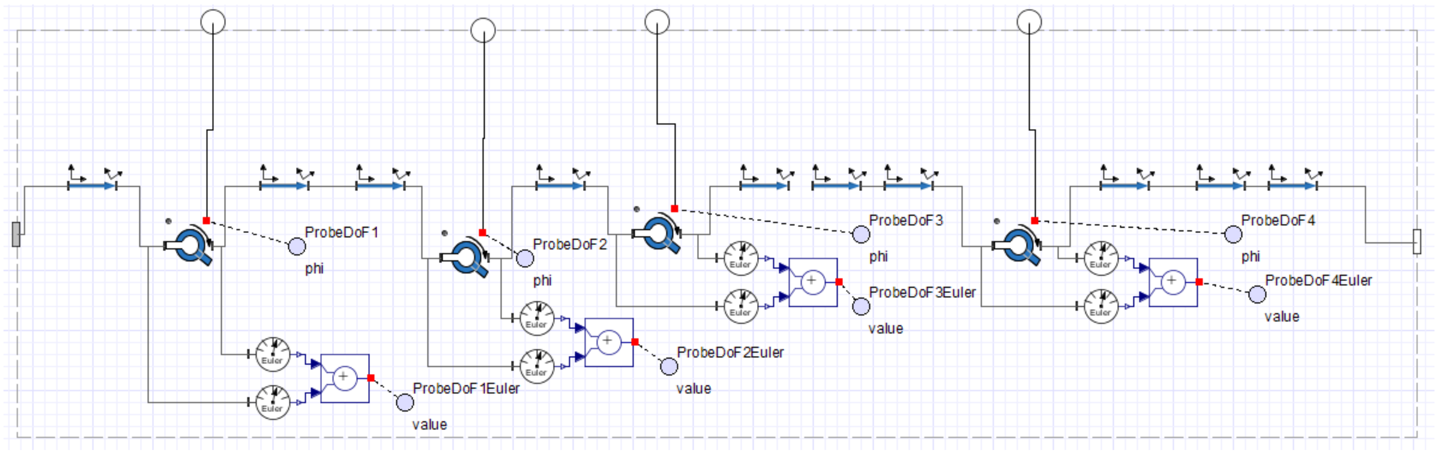


Figure B.4: The MapleSim subsystem code representing the 4 DoF Barrett WAM

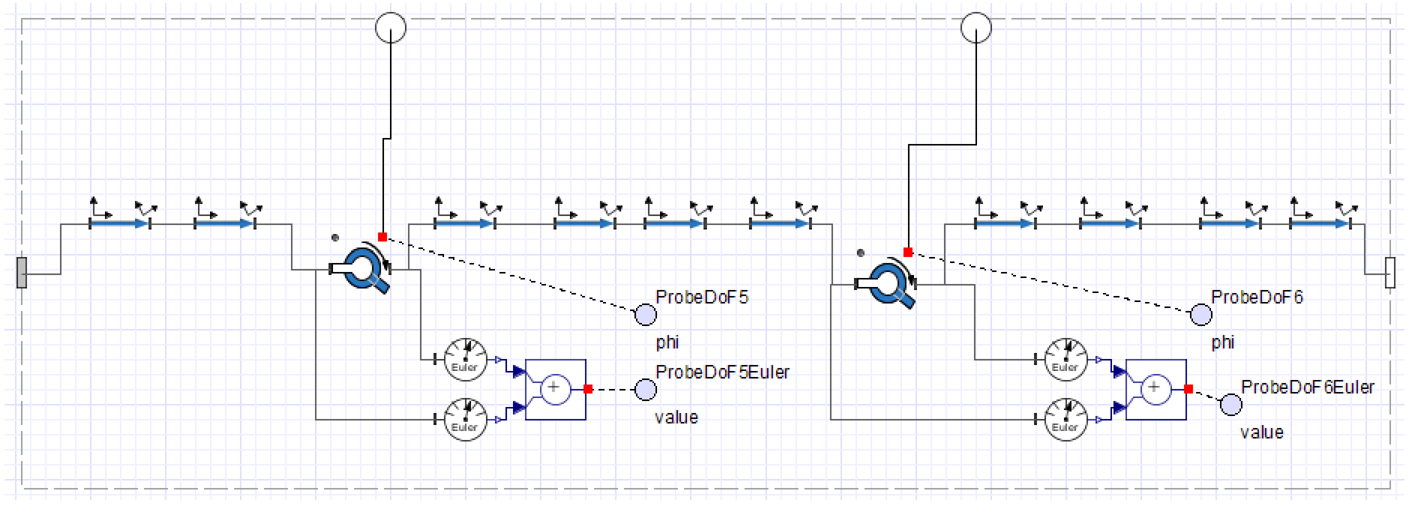


Figure B.5: The MapleSim subsystem modelling the 2 DoF PanTilt unit