School of Engineering, Discipline of Mechanical Engineering and Mechatronics, pp. 1–55

Vision-Based Control of Industrial Robot Arm

Final Year Project Report for MECH4841

July 2020

Muhammad Hasham Khan [1]

[1] *Student of Mechatronics Engineering,*
The University of Newcastle, Callaghan, NSW 2308, AUSTRALIA
Student Number: 3256011
E-mail: Muhammad.H.Khan@uon.edu.au

Dot Point Summary

This project explores:

- Modelling and controlling a robotic manipulator
- Camera modelling and feature point extraction
- Training a machine learning based object detector
- Deriving a visual-servoing controller
- Experimentally validating the previous dot points on hardware
- Integrating these different subsystems together to demonstrate a pick-and-place task using visual-servoing on a robotic manipulator.

Abstract

Vision-based control allows robots to shift from structured to dynamic environments by using the substantial information obtained through visual sensors. This technique has vast applications in robot navigation, localisation and control. This method is explored by implementing a solution for a *pick-and-place* task using a 5DOF robotic manipulator in a eye-in-hand configuration. This problem is extended to include sorting between different target objects.

The task was successfully demonstrated using image based visual servoing and allowed alignment of feature points with an accuracy of *20* pixels. A robust faster region-based convolutional neural network was also trained to reliably detect target objects. The modular structure of this work allows it to form the basis of further research.

Future improvements include incorporating dynamic models, investigating uncalibrated visual servoing, exploring the use of different camera configurations and applying this work on a ABB IRB-120 industrial robot arm.

Acknowledgements

Thank you to:

- My supervisor, Alejandro Donaire, for all his guidance, support and enthusiasm throughout my project.
- Ampcontrol CSM for providing a platform to implement my work in an industrial environment
- Dr Chris Renton for his standout coursework that inspired my understanding and appreciation of robotics
- My family and friends for bearing with my prolonged absences
- All the mechatronics boys for their camaraderie

Contents

List of Figures

List of Tables

1. Introduction

Advances in control theory have been leveraged to perform repetitive and dangerous tasks. In this context, robots have been implemented to drastically increase productivity in manufacturing. Unlike human workers, machines have typically been limited to structured environments, such as well-defined factory floors, due to the current limitations of robotic environmental awareness. Providing robots with situational awareness and intelligent decision-making capabilities allows them to operate in dynamic environments that traditional methods cannot match. The use of visual feedback techniques is a key asset in this objective.

Visual feedback relies on perceiving and responding to optical information. Controlling a robot using visual data follows naturally from how we use our vision system to gather information from, and interact with, the world around us. Artificial neural networks take inspiration from this system, mimicking the learning capabilities of our brain by recognising environmental markers. In this way, visual-servoing controllers can perform hand-eye coordination tasks. By exploring visual control, robots are becoming smarter.

This project explores visual control in the context of robotic manipulators. A common task in industry is picking up objects and placing them at a desired location. This is known as a *pick-and-place* task and is typically achieved by constraining a target object at a set position in space. Then the robot arm moves its grasping tool to a set position and successfully manipulates the object.

This relies on a key assumption; **the object must be exactly where the robot expects it to be** (Corke, 2017). In the real word, this is rarely guaranteed without the use of expensive jigs and conveyers and requires recalibration each time the object or robot is moved from their original position.

Visual feedback control, commonly known as visual servoing, overcomes these limitations and is therefore the focus of this project. This is highlighted in Appendix C, where an example of visual servoing being used in industry is showcased. The remainder of this section defines in detail the problem, a brief literature review on how others approached the task and the method developed to overcome this limitation.

1.1. Objective

The goal of this project is to use a robotic manipulator with an eye-in-hand camera to implement a pick-and-place task. Additionally, the manipulator must be able to sort between different objects and only grasp the desired one.

This task has a broad scope and various assumptions were made for simplification.

1.2. Assumptions

The following assumptions were made during this project:

Assumption A: The robot arm is fixed and stationary.

Assumption B: The robot arm has a in-built joint controller. It can, through an interface, prescribe desired joint angles.

Assumption C: The desired objects to be picked are uniform and their geometry is known. Additionally, they will be differentiable by some visual indicator.

Assumption D: The robot arm and desired objects will be located on a planar workspace free of obstructions.

Thus, the scope of the project could be defined based on these assumptions.

1.3. Scope

A pick-and-place task must be successfully implemented on hardware. This problem was simplified into the following sub-problems:

1. A mathematical model of the robot arms motion is required.
2. A camera model is required to develop a relationship between visual feedback and the camera's kinematics.
3. A method of visually classifying objects to sort them is required.
4. A visual feedback based controller is required.

These sub-problems can then be integrated and implemented to successfully perform the task. Some research was performed to ascertain how this might be achieved.

1.4. Literature Review

The following is a brief look at of how the tasks mentioned in the Scope (1.3) have been accomplished in literature.

The following books were useful as a reference. "Robotics, Vision and Control: Fundamental Algorithms in Matlab" is a book paraphrased throughout this report and forms the basis through which the problem was understood and formulated. "Robotics, Modelling and Control" by Siciliano et al. (2010) also provides a thorough introduction to kinematic modelling techniques. Uncalibrated visual servoing techniques estimates the image jacobian *online* as known camera motions are performed (Spratling and Cipolla, 1996). Nomura and Naito (2000) proposes a method for tracking and grasping objects in real time using hybrid kalman filters. A controller that switches between image based and visual based servoing is demonstrated by Gans and Hutchinson (2007). Another hybrid controller applied to a 5DOF robotic manipulator is applied by Tsai et al. (2014).

This literature review provided insight into how the Scope (1.3) could be achieved. Now we take a look at at what hardware was utilised in this project.

1.5. Hardware

The following is an outline of the hardware used for this project.

The project was initially designed with the availability of an ABB IRB-120 robotic arm. However, due to licensing issues, it could not be utilised and an alternative was required. Fortunately, a 5 DOF desktop robot arm (Figure 1) could be adapted for this problem.

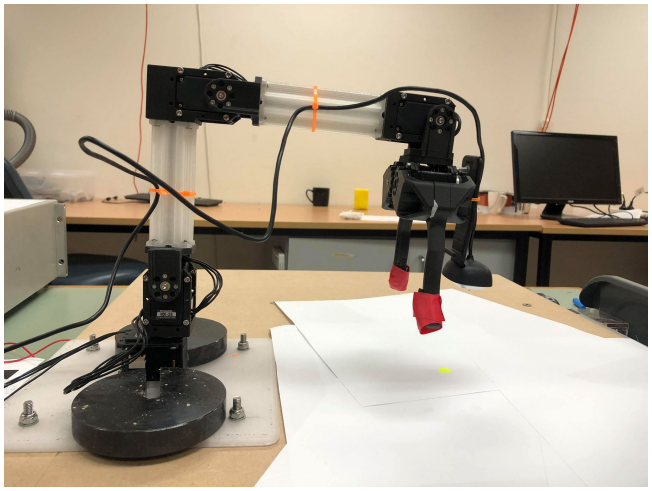


Figure 1: 5DOF desktop robot arm, pictured with MX-28 servos, 3D printed grasper (modified with rubber padding to increase grip) and mounted camera

This arm comprised of 5 Robotis Dynamixel MX-28 servo actuators. Each motor has a serial interface that can be used to command and receive state information. An onboard PID controller is used to control motor-shaft angles and an encoder returns the current position. These motors were assembled using 3D printed parts and arranged as per Figure 1.

The camera used for this project was a *Microsoft HD-3000 LifeCam Webcam* and it's mounting to the robot is shown in Figure 2 . Other cameras were also considered but not implemented on the robot (Appendix H).

Figure 2: The Microsoft camera mounted to the 5DOF robot arm

The desired objects to be interacted with by the robot were a set of wooden blocks with animal figure cut-outs (Figure 3). This common children's toy was chosen for it's uniform shape and the non-trivial classification problem it poses.

Figure 3: Wooden animal blocks used as the target object

Finally, we can take a look at implementation.

1.6. Implementation

Using the assumptions 1.2, 1.3, 1.4 and considering the hardware available, the following implementation was proposed and forms the structure for the body of the report:

- Model and control the robot's motion using kinematics.
- Use a projection model to extract information from the camera.
- Train a neural-network-based object detector.
- Derive a visual servoing controller.
- Validate the prior points. Then integrate them to demonstrate a pick and place task.

2. Robotic Manipulator Modelling

> "The mechanical structure of a robot manipulator consists of a sequence of rigid bodies(links)interconnected by means of articulations(joints);a manipulator is characterized by an arm that ensures mobility,a wrist that confers dexterity, and an end effector that performs the task required of the robot." Siciliano et al. (2010)

Fundamentally, a robot arm is simply a series of actuated links that manipulate a end effector in a desired manner. To control a robotic manipulator, a suitable mathematical model is required that captures it's kinematics. This sections establishes the framework used to define motion and applies it to derive the kinematics of a robot. It then validates this approach in simulation and hardware.

2.1. Homogenous Transformation Matrix

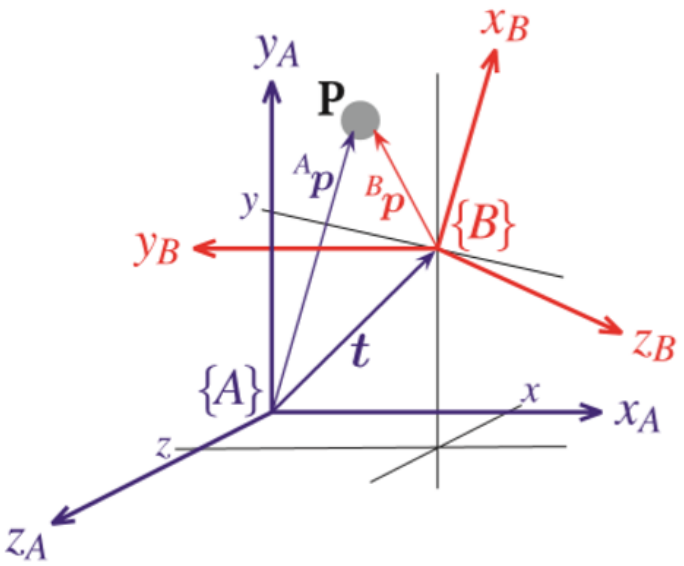


Figure 4: A displacement and rotation from frame A to frame B

Whenever we describe the position or movement of an object, it is useful to have a point of reference from which the object is displaced. This point can be the origin of an orthogonal Cartesian co-ordinate system and any displacements can be described as a deviation from the origin along it's x, y or z axis. This distance must be measured to a chosen point on the object, which will act as the origin of the objects own coordinate system. Thus, the displacement of object B can be referenced as its distance from object A as a *translation vector*.

$$^A\mathbf{t}_B = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Similarly, rotations from one frame to the next can be described by an angular deviation about the principal axis. This can be represented by the *rotation matrix*.

$$ {}^{A}\mathbf{R}_{B} = \begin{bmatrix} R11 & R12 & R13 \\ R21 & R22 & R23 \\ R31 & R32 & R33 \end{bmatrix} $$

Together, these describe the position and orientation of frame A with respect to frame B. This information can be succinctly written in a *homogenous transformation matrix*.

$$ {}^{A}\mathbf{T}_{B} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\mathbf{t}_{B} \\ \mathbf{0}^{3\times1} & 1 \end{bmatrix} $$

where ${}^{A}\mathbf{T}_{B}$ represents the homogeneous transformation matrix that transfroms a coordinate system $\{a\}$ to align with a second space $\{b\}$. The pose $\mathbf{W} = [X, Y, Z, \Omega_1, \Omega_2, \Omega_3]^\top$ represents the location and orientation of point B.

Transformation matrices have some useful properties:

$$ {}^{A}\mathbf{T}_{P} = {}^{A}\mathbf{T}_{B}\,{}^{B}\mathbf{T}_{P} \tag{2.1a} $$

$$ {}^{A}\mathbf{T}_{B}^{-1} = {}^{B}\mathbf{T}_{A} \tag{2.1b} $$

Equation 2.1a states that transformations commute. This allows us to determine the pose of P with respect to A given that we know the pose of B from A (${}^{A}\mathbf{T}_{B}$) and P from B (${}^{A}\mathbf{T}_{B}$).
Equation 2.1b states that transformations are invertible. Knowing the pose of B from A can also be expressed as knowing the pose of A from B.

Thus, we have a powerful framework for describing how objects move using transformation matrixes. This will form the basis of our *Kinematics*

2.2. Robotic Manipulator Kinematics

A robotic manipulator can be seen as a sequence of rigid bodies, known as *links*, interconnected by actuated *joints*. This can be referred to as an *open serial kinematic chain* and an example is shown in Figure 5. The frame $\{0\}$ acts as the *base* of the chain and the goal of kinematics is to express the pose of all subsequent links with respect to the base.

This pose can be expressed through homogenous transformation matrixes (Section 2.1) by assigning a frame to each actuated joint. These joints are typically restricted to a single *Degree of Freedom*: they can either rotate or extrude. These joints can be classified as *revolute* or *prismatic* joints respectively.

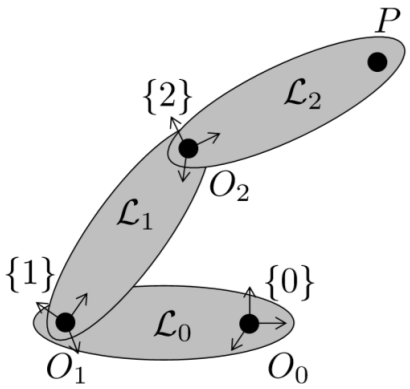
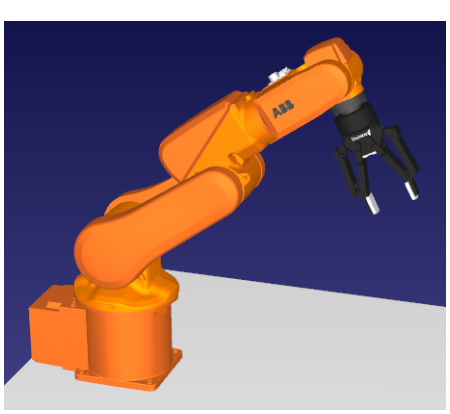
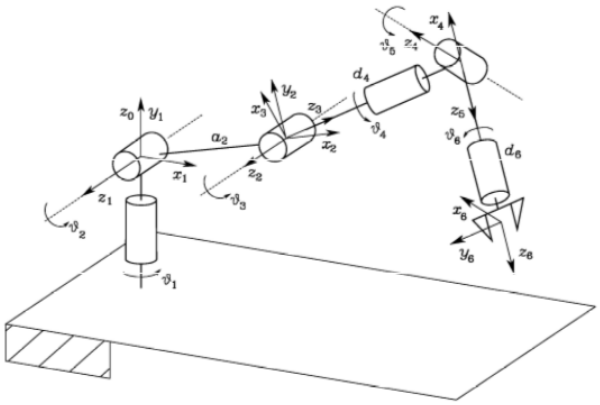


Figure 5: A serial kinematic chain, from (Renton, 2019a)

2.2.1. Robotic Manipulator Forward Kinematics



(a) A 3D model of the arm with a gripper attachment in RoboDK (2020b)



(b) An anthropomorphic arm using revolute joints from Siciliano et al. (2010)

Figure 6: Modelling of the ABB IRB-120

The ABB IRB-120 (Figure 6(a))is a 6DOF robotic manipulator, with 6 actuated joints in a *anthropomorphic* configuration (Figure 6(b)). This arrangement is a form of biomimicry and imitates a biological arm with an *elbow* and *wrist*. The first, second and third joints form the elbow and control the *position* of the subsequent joints while the fourth, fifth and sixth joints form the wrist and control the *orientation* of the final frame. This final frame is known by convention as the *end effector* and is located at the center of the gripper.

The goal of Forward Kinematics is to obtain the pose of the last frame x_e, known as the end effector, with respect to the base frame. This can be expressed as:

$$^{0}\mathbf{T}_{e} = k(q) \tag{2.2}$$

To do this we need to find the transformation matrices that relate the motion from each joint frame to

the next as a function of joint angles q. We can then use the commutative property of transformation matrices 2.1a to express this as:

$$ {}^{0}\mathbf{T}_{e} = {}^{0}\mathbf{T}_{1}\,{}^{1}\mathbf{T}_{2}\,{}^{2}\mathbf{T}_{3}\,{}^{3}\mathbf{T}_{4}\,{}^{4}\mathbf{T}_{5}\,{}^{5}\mathbf{T}_{6}\,{}^{6}\mathbf{T}_{e} \tag{2.3} $$

Deriving these transformations from one joint to the next is not a straightforward task. Fortunately, the Denavit Hartenberg convention can be used. This convention is a methodology used to place and relate coordinate frames between joints and is used to generate the desired transformations (Hartenberg and Danavit, 1964; Hartenberg and Denavit, 1955).

As shown in Figure 7 , it relates the coordinate frame between joint j and joint $j+1$ using four parameters. The definition of each parameter is summarised in Table 1/.

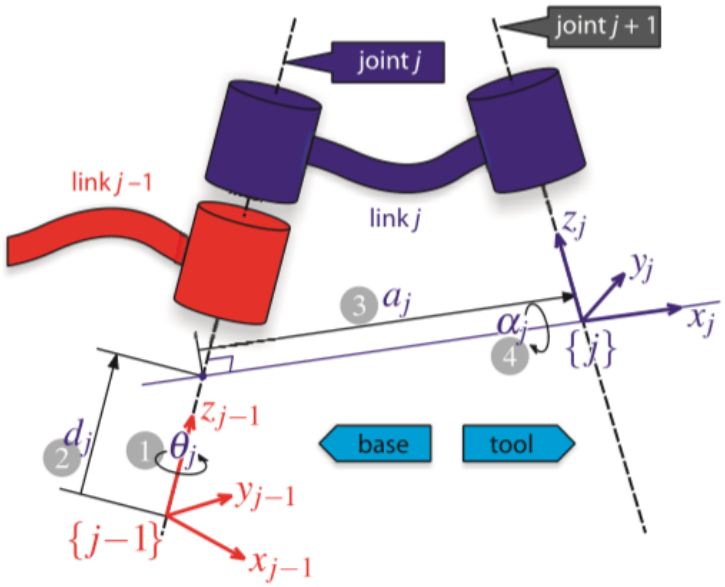


Figure 7: Transformation between two joints using the Denavit Hartenberg Convention, from Corke

Table 1: Denavit-Hartenberg parameters: their physical meaning, symbol and formal definition based on Corke (2017)

Joint angle	θ	The angle between the x_{j-1} and x_j axes about the z_{j-1} axis	Variable for revolute joints; Constant for Prismatic Joints
Link offset	d	The distance from the origin of frame $j-1$ to the xj axis	Variable for Prismatic joints; Constant for Revolute Joints
Link length	a	The distance between the z_{j-1} and z_j axes along the x_j axis; for intersecting axis parallel to $\hat{z}_{j-1} \times \hat{z}_j$	Constant for rigid links
Link twist	α	The angle from the z_{j-1} axis to the z_j axis about the x_j axis	Constant for rigid links

Using this convention, transformations between joints can be expressed using the aforementioned parameters as 2 consecutive *screw displacements* about the z_j and a_j axis. This can be expanded and rewritten as:

$$ {}^{j-1}\mathbf{T}_j = \begin{bmatrix} \cos\theta_n & -\sin\theta_n \cos\alpha_j & \sin\theta_j \cos\alpha_j & r_j \cos\theta_j \\ \sin\theta_n & \cos\theta_n \cos\alpha_j & -\cos\theta_j \sin\alpha_j & r_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4} $$

which *generates* transformation matrices as a function of 4 parameters.

The Denavit hartenberg paramaters for the robot were found using RoboDK (2020a) and shown in Table 2

Table 2: Denavit Hartenberg parameters for ABB IRB 120

	q	**d**	**a**	α
Joint 1	0	0	290	-pi/2
Joint 2	-pi/2	270	0	0
Joint 3	0	70	0	-pi/2
Joint 4	0	0	302	pi/2
Joint 5	pi/2	0	0	-pi/2
Joint 6	pi	0	72	0

Using the parameters from Table 1, the desired transformation matrices could be generated, with a fixed translation to get from the spherical wrist to the end effector.

Thus, we can now solve Equation 2.3, which can be written in homogenous form:

$$ {}^{0}\mathbf{T}_{\mathbf{ee}} = \begin{bmatrix} {}^{0}\mathbf{R}_{\mathbf{ee}} & {}^{0}\mathbf{t}_{\mathbf{ee}} \\ \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{2.5} $$

and the pose can be more intuitively expressed using euler angles (see Appendix G) as:

$$ \mathbf{x}_e = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.6} $$

and the pose of the end effector is expressed as a function of joint angle q:

$$ \mathbf{x}_e = k(q) \tag{2.7} $$

2.2.2. Robotic Manipulator Inverse Kinematics

Forward kinematics allows us map the joint angles q in *joint space* to the end effector pose xe in *Cartesian space*. As tasks are commonly prescribed in terms of position and orientation, we require a method to map a pose in Cartesian space to joint angles in *joint space*. This mapping is known as a jacobian. This section borrows heavily from Renton (2019b).

One method of solving the inverse kinematic problem numerically requires use of the *geometric* and *analytical* jacobians.

The *geometric Jacobian* relates the linear and angular velocity of the end effector in Cartesian space to the time derivative of joint angles in joint space and can be expressed as:

$$\nu_e = \mathbf{J}_G(\mathbf{q})\dot{\mathbf{q}}$$

where:

$$\nu_e = \begin{bmatrix} \mathbf{v}_e^0 \\ \omega_e^0 \end{bmatrix} \tag{2.8}$$

The geometric jacobian can be generated for a series of revolute joints, using the notation given by the Denavit-Hartenberg convention, as:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_v \\ \mathbf{J}_\omega \end{bmatrix} = \begin{bmatrix} R_{j-1}^0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} (t_n^0 - t_{j-1}^0) \\ R_{j-1}^0 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{bmatrix} \tag{2.9}$$

Which is given for a 6DOF arm by:

$$\mathbf{J} = \begin{bmatrix} J_{v_1} & J_{v_2} & J_{v_3} & J_{v_4} & J_{v_5} & J_{v_6} \\ J_{\omega_1} & J_{\omega_2} & J_{\omega_3} & J_{\omega_4} & J_{\omega_5} & J_{\omega_6} \end{bmatrix}$$

Once again, we wish to prescribe the joint rates using a rate of change of euler angles. This can be done using the *analytical jacobian*, which can be computed using:

$$\mathbf{J_A} = \mathbf{T_A}(\mathbf{R_n^0})\mathbf{J_G}$$

where:

$$\mathbf{T_A}(\mathbf{R_n^0}) = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{E^{-1}}(\mathbf{R_n^0}) \end{bmatrix} \tag{2.10}$$

and:

$$\mathbf{E}(\mathbf{R}_{\mathbf{n}}^{\mathbf{0}}) = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & cos\phi & -\sin\phi cos\theta \\ 0 & -sin\phi & cos\phi cos\theta \end{bmatrix} \tag{2.11}$$

Thus, we have a direct relationship between the time derivative of the end effector pose $\dot{\mathbf{x}}_{\mathrm{e}}$ and joint angles $\dot{\mathbf{q}}$ using the analytical jacobian:

$$\dot{\mathbf{x}}_{\mathrm{e}} = \mathbf{J}_{\mathbf{A}}\dot{\mathbf{q}} \tag{2.12}$$

Given a desired pose:

$$\mathbf{x}_d = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.13}$$

We can define an error:

$$\mathbf{e} = \dot{\mathbf{x}}_{\mathrm{d}} - \dot{\mathbf{x}}_{\mathrm{e}}$$

and take its time derivative as:

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_{\mathrm{d}} - \dot{\mathbf{x}}_{\mathrm{e}} \tag{2.14}$$

into which we can substitute Equation 2.12 to give:

$$\dot{\mathbf{e}} = \dot{\mathbf{x}}_{\mathrm{d}} - \mathbf{J}_{\mathbf{A}}(\mathbf{q})\dot{\mathbf{q}} \tag{2.15}$$

If we consider the homogenous ordinary differential equation:

$$\dot{\mathbf{e}} = \mathbf{K}\mathbf{e} \tag{2.16}$$

we can rewrite Equation 2.15 as:

$$\dot{\mathbf{q}} = \mathbf{J}_{\mathbf{A}}(\mathbf{q})^{-1}(\dot{\mathbf{x}}_{\mathrm{d}} - \mathbf{K}\mathbf{e}) \tag{2.17}$$

where K is a diagonal matrix whose eigenvalues act as the *gain* of a proportional controller. This is expressed as a block diagram in Figure 8.

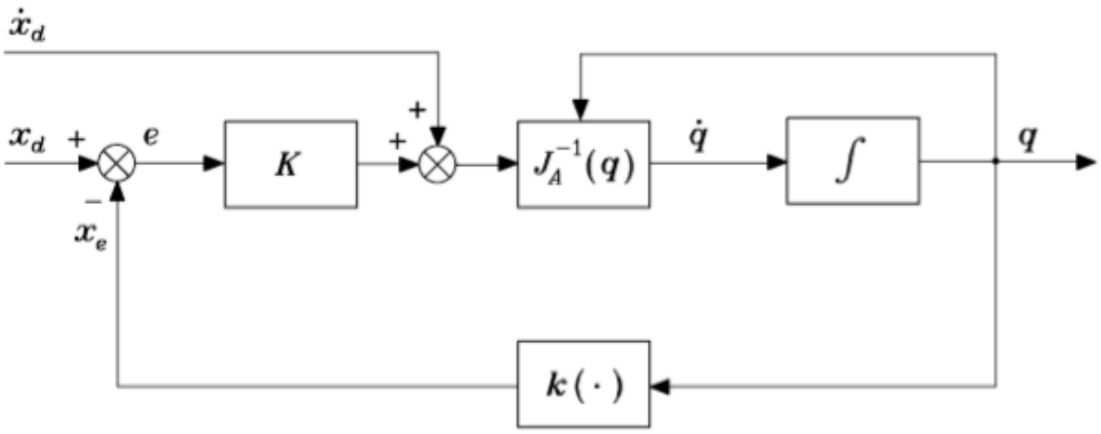


Figure 8: Inverse kinematic model Control Scheme

Provided that the analytical jacobian is square and of full rank and assuming a fixed reference ($\dot{\mathbf{x}}_\mathrm{d}$), we can simulate equation 2.17 to obtain a solution to our inverse kinematic problem.

Conversely, an optimisation based solution to the inverse kinematic problem can be implemented by posing problem as a nonlinear program:

$$\begin{aligned}(\mathbf{q}^\star, \mathbf{x}^\star) = \arg\min_{\mathbf{q},\mathbf{x}} \mathbf{q}^\top \mathbf{W}\mathbf{q} + (\mathbf{x} - \mathbf{x}_e)^\top \mathbf{K} (\mathbf{x} - \mathbf{x}_e) \\ \text{s.t. } \mathbf{x} - \mathbf{k}(\mathbf{q}) = \mathbf{0}\end{aligned} \tag{2.18}$$

This can be solved using `fmincon`, a nonlinear programming solver in Matlab. This technique has the added benefit of allowing us to impose minimum and maximum joint angles as a lower bound(`lb`) and upper bound(`b`) respectively.

2.3. Simulation and Hardware

The forward kinematic model for the ABB robot is shown as a "ball and stick" plot in Figure 9.

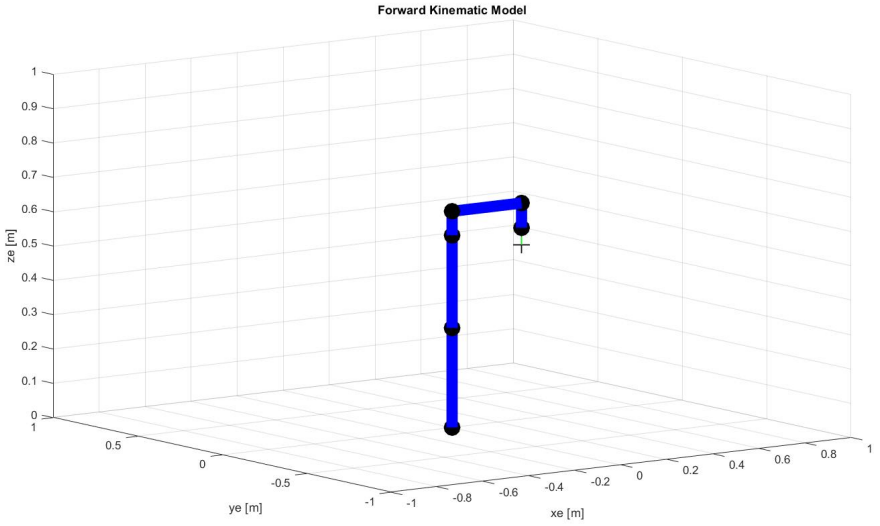


Figure 9: Forward kinematic model simulation

While it's inverse kinematics was used to follow a simple *path* shown in Figure 10.

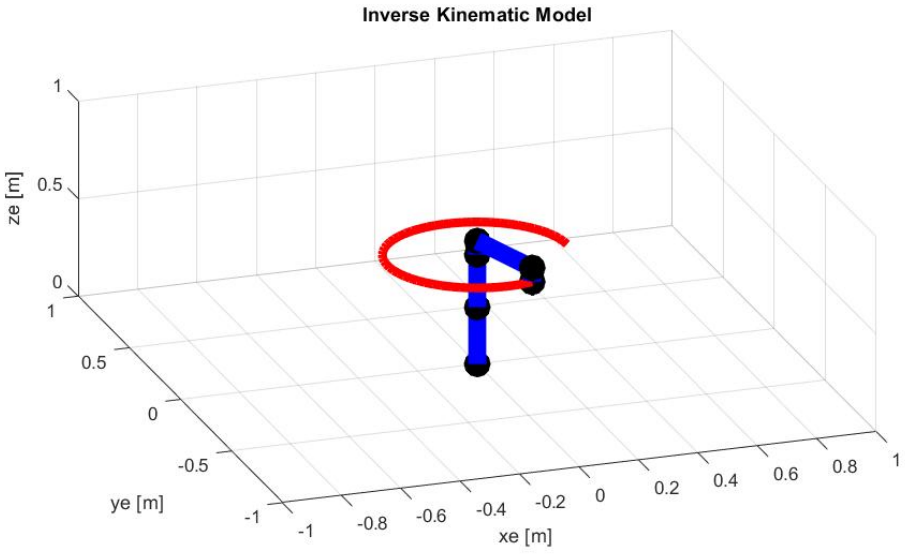


Figure 10: Inverse kinematic model Simulation

Due to issues accessing hardware highlighted in Section 1.5, the forward and inverse kinematics were adapted for implementation with the 5DOF arm. Fortunately, this was relatively straightforward as the structure was largely the same. The actuators responsible for *roll* and *yaw* had their associated

transformations set to $\boldsymbol{I}$ and new values for the Denavit Hartenberg parameters were found. The new kinematic model is visualised in Figure **??**.

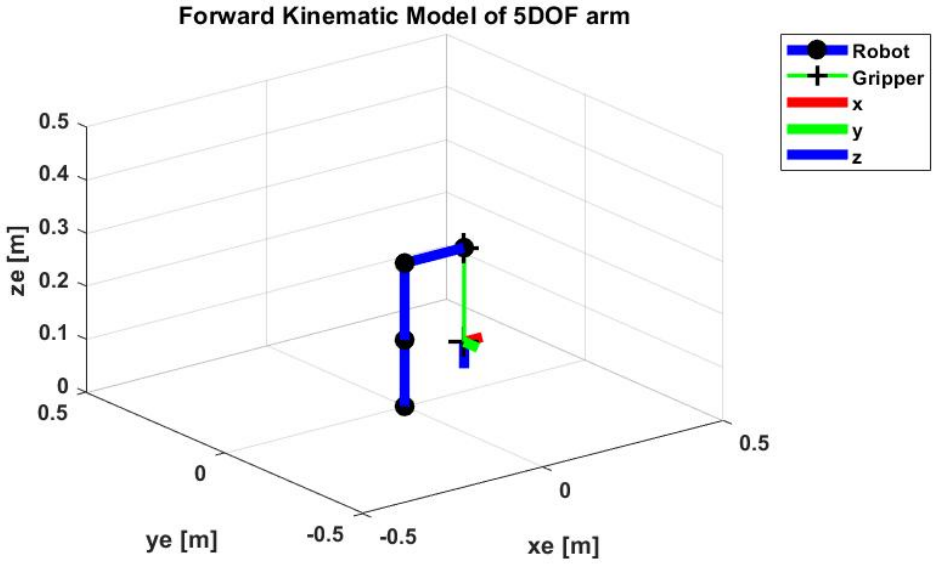


Figure 11: 5DOF arm Forward kinematic model Simulation

It is important to note that only the first 4 joint angles $\mathbf{q}$ affect the pose of the robot while the final joint control the gripper. Additionally, the robot is no longer *fully actuated* in its *task space*. It can not independently achieve a prescribed roll or yaw angle and a desired pose can only be prescribed as:

$$\mathbf{x}_d = \begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} \tag{2.19}$$

3. Camera Modelling

Visual servoing relies on the relationship between real world points and how they are represented by cameras. This section defines the relationship using the camera projection model and subsequently introduces camera calibration, camera pose estimation and the Image Jacobian.

3.1. Camera Projection Model

Using a standard pinhole model, a camera projects a 3D world onto a 2D image plane as shown by Figure 12.

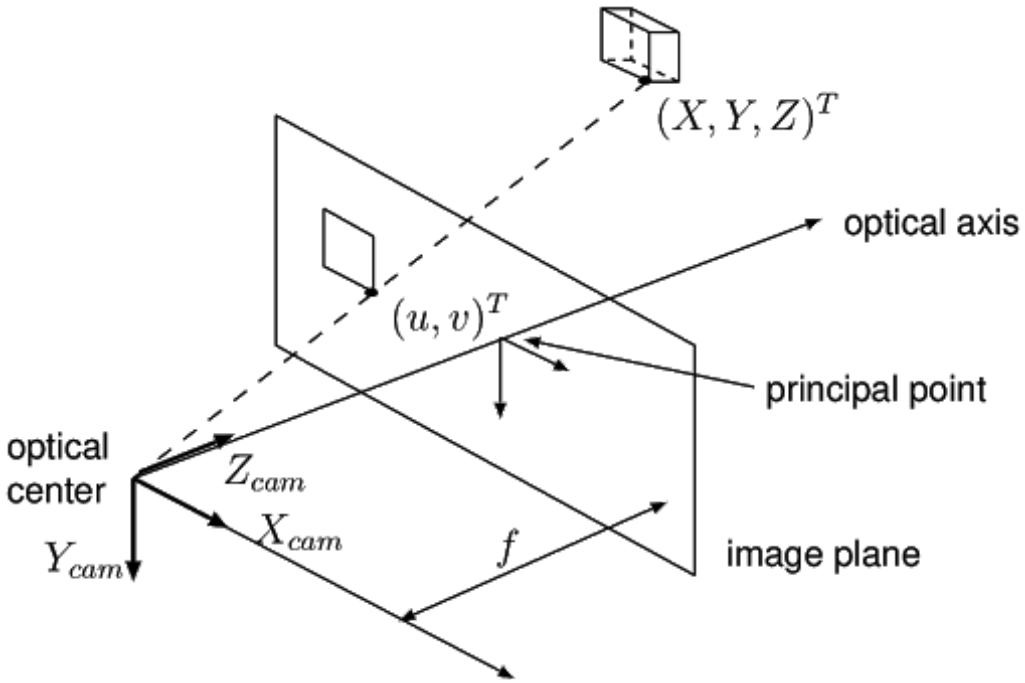


Figure 12: The Camera Projection Model

Zhang (2000) describes this as the *projection* of world point $P = [XYZ]^T$ to the image point $p = [uv]^T$ through the function:

$$p = \mathcal{P}(P, \mathbf{I}, T_{cam}) \tag{3.1}$$

where $\mathbf{I}$ and T_{cam} are the intrinsic and extrinsic parameters of the camera respectively. The extrinsic parameter is simply the pose of the camera, while its intrinsic parameters comprise of features such as the camera focal length (f), principal point (u_0, v_0) and camera skew (γ). This information is compiled into the *Intrinsic Matrix*:

$$\mathbf{I} = \begin{bmatrix} f & \gamma & u_0 \\ 0 & f & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

The projection model is based on an ideal projection through a pinhole whereas a real camera contains a non-ideal lens that distorts the image. Imperfect curvature of the lens manifests itself as radial distortion (see Figure 13).

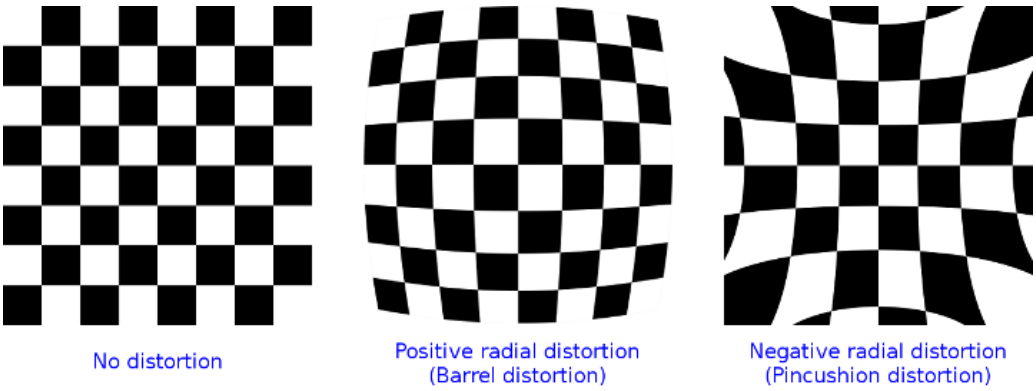


Figure 13: The effects of Radial Distortion

whereas misalignment between the lens and image sensor causes tangential distortion (see Figure 14).

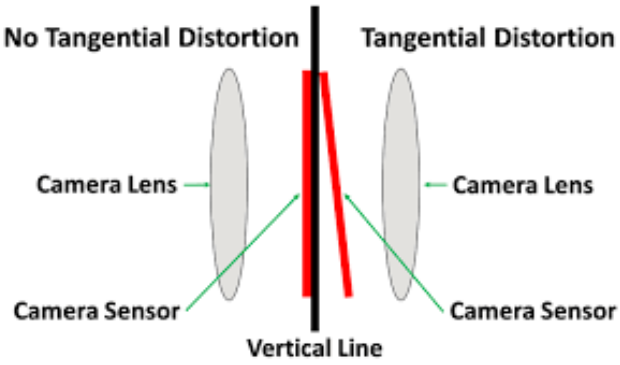


Figure 14: The effects of Radial Distortion

These intrinsic parameters and distortions are unique to each camera and necessitate calibration.

3.2. Camera Calibration

Camera calibration is technique that estimates the intrinsic parameters and distortions of a camera Zhang (2000). This estimation is done by by exploiting the projection equation (3.1).

If the precise geometry of the world points and their projection at a variety of camera orientations is know, the intrinsic parameters of the camera can be estimated using a closed form solution and refined using maximum likelihood estimation. This refinement method can also be used to estimate the radial and tangential distortion of the camera.

The calibration was performed using the Matlab Camera Calibrator toolbox (Bouguet, 2004). This method involved collecting images of a planar checkerboard pattern (Figure 15) using the desired

camera.

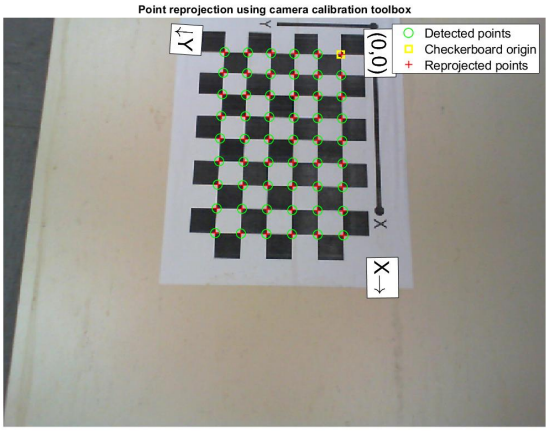
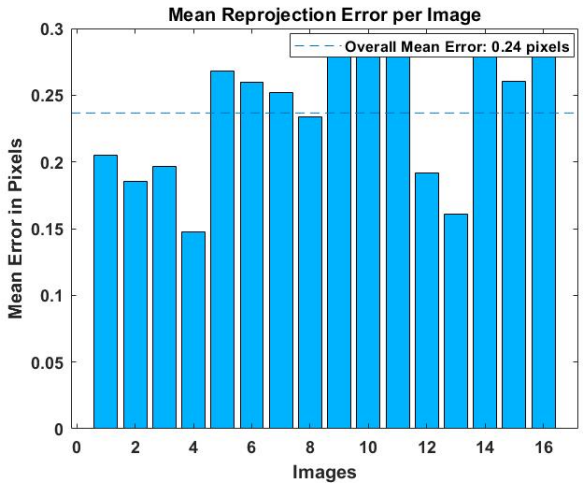
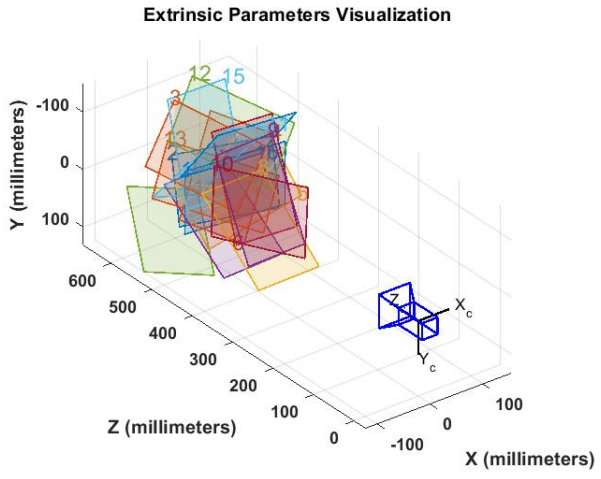


Figure 15: Checkerboard pattern with detected and re-projected feature points

The calibrator extracts the corners of the checkerboard as feature points (see Section 3.3) and then projects them to the image plane. By providing the toolbox with a precise measurement of the checkerboard squares, which are all identical, a precise geometric model can be formed.



(a) Mean reprojection error in the calibration images



(b) Estimated relative pose between the camera and planar checkerboard

Figure 16: Camera calibration results

The calibration procedure was then performed by the toolbox and the calculated parameters are shown in Table 3.

Table 3: Camera Parameters for Microsoft HD-3000 LifeCam Webcam

Parameter	Symbol	Estimated Value
Focal length (mm)	f	683.23
Principal point (pixels)	u_0, v_0	322.1226 , 222.6395
Radial Distortion*	k_1, k_2	0.2024 , -1.2111
Tangential Distortion*	$p_1 p_2$	0.0037 , 0.0019
Skew*	s	-1.4685

* Unitless Parameter

3.3. Feature Point Extraction

Our goal was to extract feature points from an image of the target objects shown in Figure 17

Figure 17: The target object with added tape

Objects are typically described by their characteristics such as size, shape and colour. In images, measurements of these properties are known as *features*. Techniques using corner and edge detection have been developed. However, it was difficult to reliably obtain corners due to shadows. Therefore, red tape was added to the corners to act as fiducials and colour thresholding was used.

Images are digitally stored as an intensity in 3 RGB channels representing the red green and blue spectrum. We can simply extract the red channel and filter for a colour intensity corresponding to the tape that also minimises *noise*. This results in the image shown in figure 18, which consist of a set of interconnected pixels known as *blobs*.

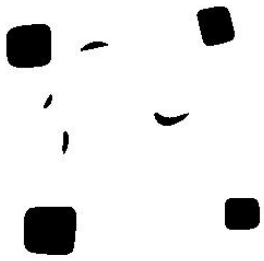


Figure 18: Figure 17 after colour thresholding

We'd like to filter for the desired blobs corresponding to the tape and obtain the location of their centroid. This can be done using *Region feature extraction*. An implementation of this method is available in the Machine vision toolbox by Corke (2017) as a Matlab function called `iblobs`.

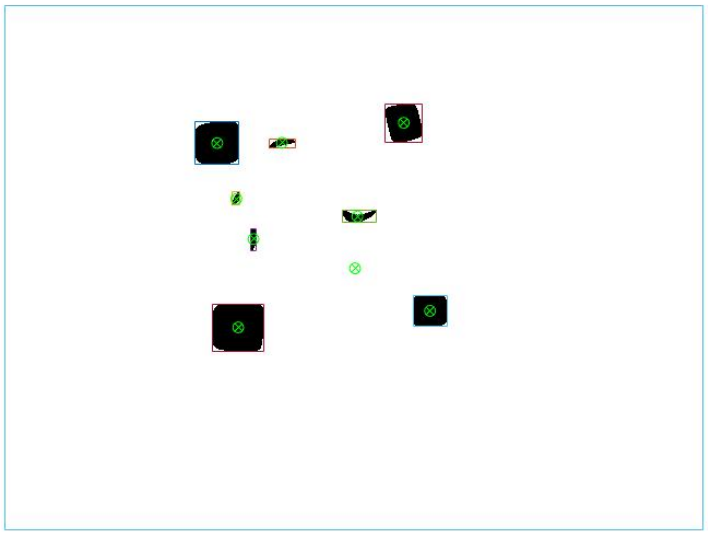


Figure 19: The image after region feature extraction using `iblobs`

Filtering for the desired blobs was done based on total *pixel area* and *aspect ratio*. Pixel area is

simply a sum of a blob's pixels and values were chosen based on tuning. The aspect ratio is based on transforming a blob into an equivalent ellipse (Figure 20) with a major and minor axis length a and b respectively. This ratio is then defined as $\frac{b}{a}$ and can be used to indicate the *circularity* of a shape.

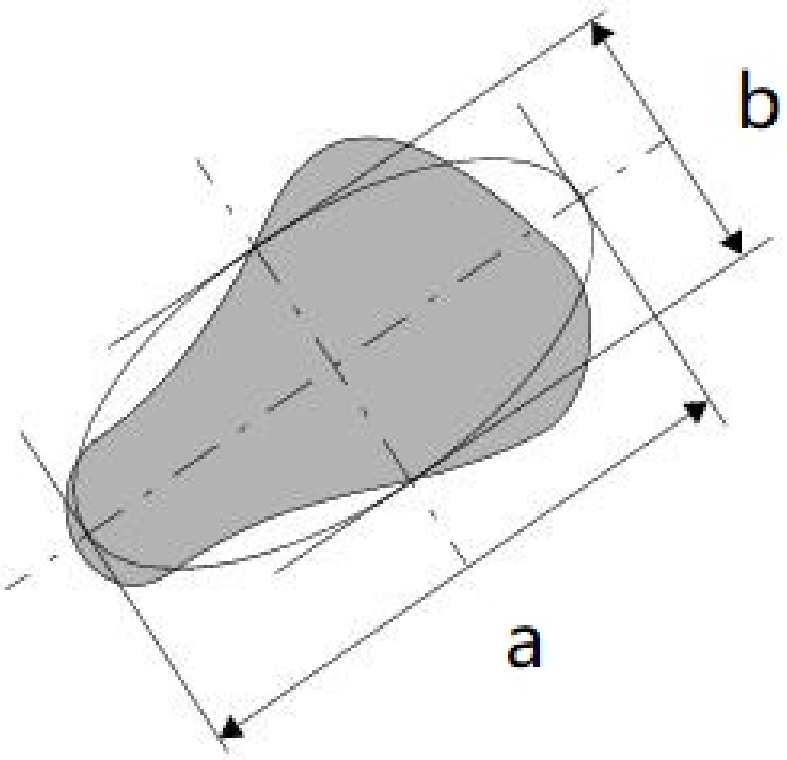


Figure 20: A blob and it's equivalent ellipse , retreived and modified from (Microscan, 2020)

The tape pieces were cut to a square shape to *maximise* this circularity and allow robust filtering. Their *centroid*, which is the average pixel coordinate of each pixel in a blob, was then extracted and used as our feature points.

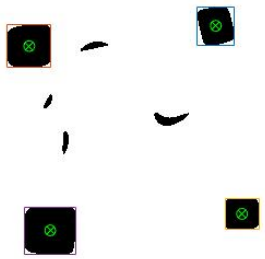


Figure 21: The desired feature points after filtering based on circularity

The *order* of these feature points was not guaranteed as they were sorted based on the raster order (smallest v coordinate) of each blob. This order changes as the object or camera reorients itself and the feature points subsequently change coordinates. Therefore, the points were filtered by comparing their current location to their previous and minimising the error.

3.4. Pose estimation

Pose estimation was done using an implementation of Ke and Roumeliotis (2017). This functionality is implemented in the *Machine Vision Toolbox* by Corke (2017) as the *Central camera class* method `estpose`. An example of a pose estimated by the camera using extracted feature points is shown in Figure 22.

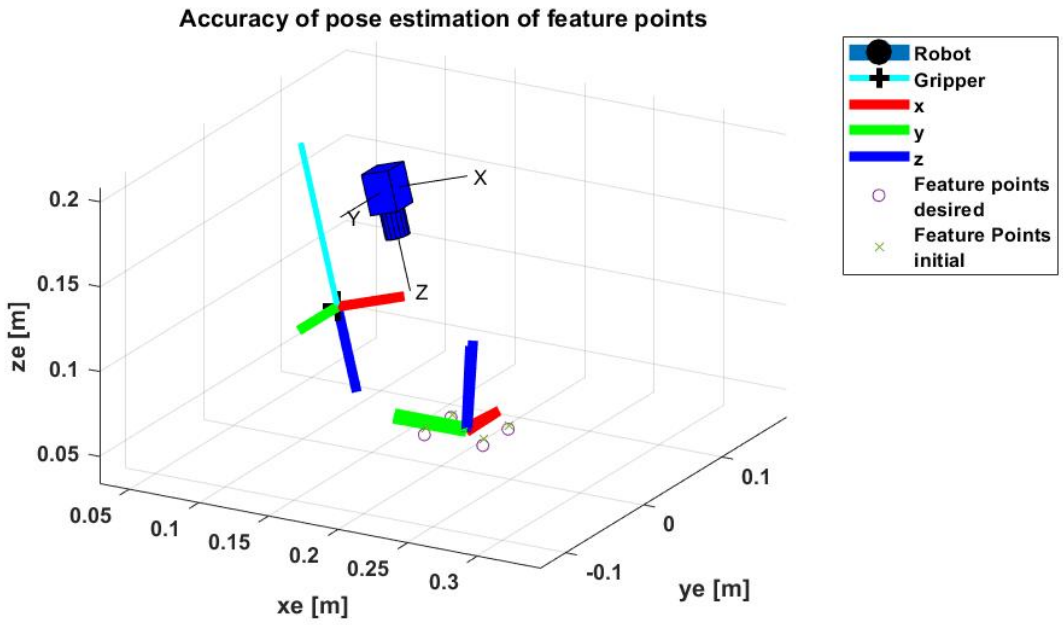


Figure 22: The estimated pose of the feature points at different camera orientations

3.5. Image Jacobian

Another useful relationship of the camera projection model is the *Image Jacobian*, which relates the motion of world points to image points. This can be expressed as a derivative with respect to time of the projection equation 3.1:

$$\dot{p} = \mathbf{J}_P(P, \mathbf{I}, T_{cam})\nu \tag{3.3}$$

where $\dot{p}$ is the feature point velocity, ν the camera velocity and $\mathbf{J}_P$ is the Image Jacobian. If we consider point $\mathbf{P} = [X, Y, Z]^\top$ and camera with velocity $\nu = [\mathbf{v}, \omega]^\top$, the velocity of the points with respect to the camera frame is:

$$\dot{\mathbf{P}} = -\omega \times \mathbf{P} - \mathbf{v}$$

which can be expressed in scalar form as:

$$\begin{aligned} \dot{X} &= Y\omega_z - Z\omega_y - v_x \\ \dot{Y} &= Z\omega_x - X\omega_y - v_y \\ \dot{Z} &= X\omega_y - Y\omega_y - v_z \end{aligned} \tag{3.4}$$

The perspective projection of for normalized image plane coordinates is:

$$x = \frac{X}{Z}, \quad y = \frac{Y}{Z} \tag{3.5}$$

Whose derivative with respect to time is, using the quotient rule:

$$\dot{x} = \frac{\dot{X}Z - X\dot{Z}}{Z^2}, \dot{y} = \frac{\dot{Y}Z - Y\dot{Z}}{Z^2} \tag{3.6}$$

Substituting in equation 3.4 and using $X = Zx, Y = Zy$ gives:

$$\begin{aligned} \dot{x} &= -\frac{1}{Z}v_x + \frac{X}{Z}v_z + xy\omega_x - (1+x^2)\omega_y + y\omega_z \\ \dot{y} &= -\frac{1}{Z}v_y + \frac{y}{Z}v_z + +(1+x^2)\omega_x - xy\omega_y - x\omega_z \end{aligned} \tag{3.7}$$

which can be written out in matrix form as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{X}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & (1=y^2) & -xy & -x \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.8}$$

Thus, we have a relationship between camera spatial velocity and normalised image coordinate velocity. We now need to transform this *image velocity* to *pixel velocity*. The relationship between normalised image velocity and pixel coordinates is given by:

$$u = \frac{f}{\rho_u}x + u_0, u = \frac{f}{\rho_v}y + v_0 \tag{3.9}$$

which can be rearranged to give:

$$x = \frac{\rho_u}{f}\bar{u}, y = \frac{\rho_v}{f}\bar{v} \tag{3.10}$$

where $\bar{u} = u - u_0, \bar{v} = v - v_0$ are the pixel coordinates relative to the principal point. The time derivative is:

$$\dot{x} = \frac{\rho_u}{f}\dot{\bar{u}}, \dot{y} = \frac{\rho_v}{f}\dot{\bar{v}} \tag{3.11}$$

This can be substituted into equation 3.8, giving:

$$\begin{bmatrix} \dot{\bar{u}} \\ \dot{\bar{v}} \end{bmatrix} = \begin{bmatrix} -\frac{f}{\rho_u Z} & 0 & \frac{\bar{u}}{Z} & \frac{\rho_v \bar{u}\bar{v}}{f} & -\frac{f^2+\rho_u^2\bar{u}^2}{\rho_u f} & \frac{\rho_v \bar{v}}{\rho_u} \\ 0 & -\frac{f}{\rho_v Z} & \frac{\bar{v}}{Z} & \frac{f^2+\rho_v^2\bar{v}^2}{\rho_v f} & -\frac{\rho_u \bar{u}\bar{v}}{f} & -\frac{\rho_u \bar{u}}{\rho_v} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.12}$$

As most image sensors use square pixels, we can assume $\rho_u = \rho_v = \rho$. This allows the focal length to be expressed in pixels as $f' = \dfrac{f}{\rho}$

$$\begin{bmatrix} \dot{\bar{u}} \\ \dot{\bar{v}} \end{bmatrix} = \begin{bmatrix} -\frac{f'}{Z} & 0 & \frac{\bar{u}}{Z} & \frac{\bar{u}\bar{v}}{f'} & -\frac{f'^2+\bar{u}^2}{f'} & \bar{v} \\ 0 & -\frac{f'}{Z} & \frac{\bar{v}}{Z} & \frac{f'^2+\bar{v}^2}{f'} & -\frac{\bar{u}\bar{v}}{f'} & -\bar{u} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{3.13}$$

Thus, we have our desired relationship between pixel motion with respect to the principal points and camera velocty. This can written as:

$$\dot{p} = \mathbf{J}_p(\mathbf{pZ})\nu \tag{3.14}$$

Where J_p is the image jacobian expressed as a function of point feature pixel coordinates p and feature depth Z.
This equation can be vectorised for n feature points:

$$\begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{bmatrix} = \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ \vdots \\ J_p(p_n, Z_n) \end{bmatrix} \nu \tag{3.15}$$

To ensure the Jacobian is full rank and nonsingular, a minimum of 3 feature points are required.

4. Object Detection

Machine vision techniques such as *nearest neighbor* or Support Vector Machines (SVM) can be used to *classify* images (Li, 2009). However, this technique requires image feature extraction if we wish to detect multiple objects in a single image.

A superior technique that has become increasingly popular since the success off "Alexnet" (Krizhevsky et al., 2012) is to use *Convolutional Neural Networks* to perform classification. These have then been adapted to perform *object detection*, the task of *localising* and *classifying* objects in an image. The following discusses the *intuition* behind convolutional neural networks and how one was trained.

4.0.1. Background

Hubel and Wiesel (1959) performed experiments to investigate a cats visual system (Figure 23). By analysing the electrical signal produced by neurons in a cat's brain they were able to demonstrate the presence of "specialised neurons that respond only to certain sensory information".

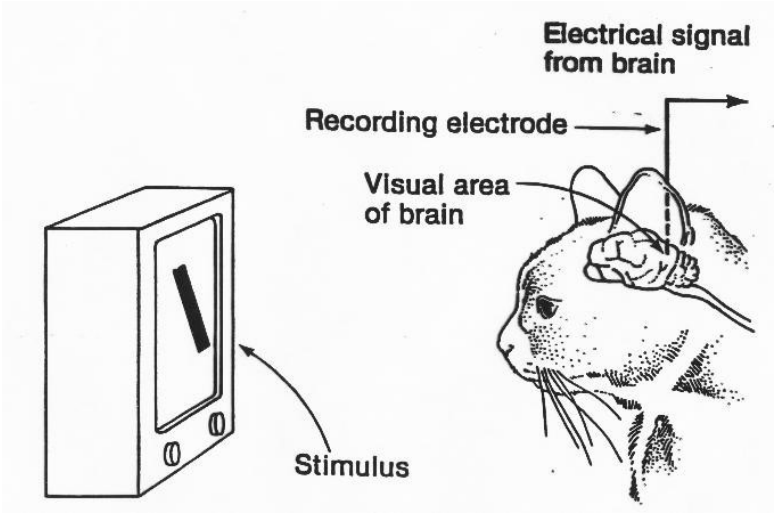


Figure 23: Experimental investigation of a cats visual system

Image features such as lines, curves and motion caused activity in different areas of neurons in the brain. This gave insight as to how a complex visual scene could be composed of simpler patterns that are detected by different groups of neurons.

Artificial neurons have been modelled to mimic the functionality of biological neurons (Figure 24.

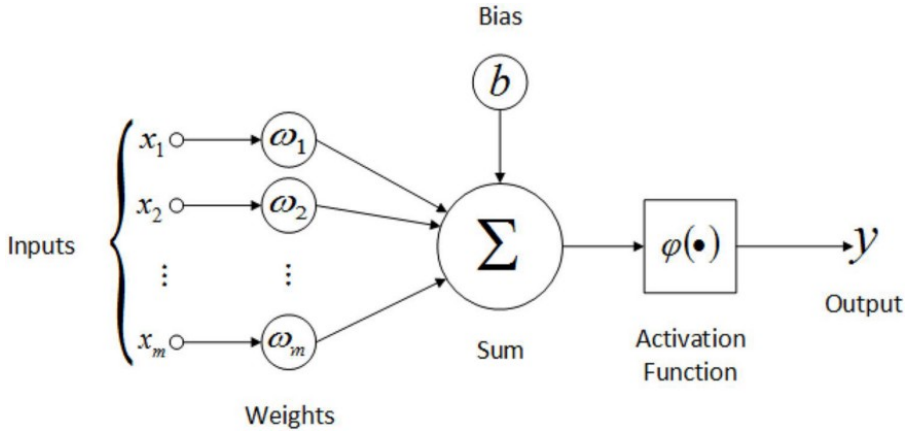


Figure 24: A model of an artificial neuron

which could be expressed by the equation:

$$y = \psi\left(\sum_{j=0}^{m} \omega \mathbf{x}\right) \tag{4.1}$$

The cross product of the *weights* and the *inputs* results in a scalar value which is then outputed by a *activation function* . If this activation function is chosen to be nonlinear, the output y is nonlinear. These individual nodes can then be stacked into a structure known as a *neural network* (Figure 25) . Bernardini and De Fina (1991) showed that various non-linear continious functions can be approximated using a neural networks with hidden layers.

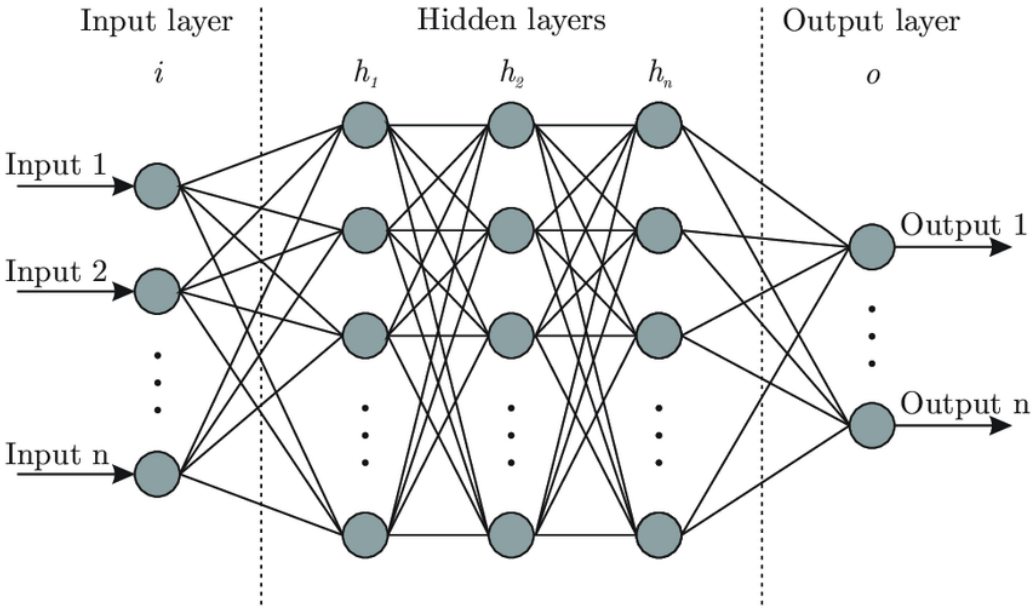


Figure 25: Basic architecture of a neural network, retrieved from (Shukla, 2020)

This approximation is done by *training* the network using *backpropagation*. This technique uses a set of inputs that produce *known outputs* y^* for the function that needs to be approximated. Those same

inputs are then computed through a suitable neural network producing a unique output y. If the weights of the neural networks are *adjusted* such that $y^* - y$ is *minimised*, the network tends towards an approximation of the desired function. These networks have been adapted to take in images as *inputs* and approximate the nonlinear function of object classification and location.

4.0.2. Training and Validation

For the case where multiple objects might be in an image, we'd like to not only classify the object but also know where in the image it is. This can be accomplished using a Faster Region-based convolutional neural network (Faster RCNN). A Faster RCNN was selected based on the performance criteria available at the tensorflow github repository Tensorflow (2020). This was because it was very accurate at the cost of being slower than other architectures such as YOLO or SSD. Fortunately, a Faster RCNN model does not need to be designed from scratch as models are already available on Tensorflow, which provides a framework for designing and training neural networks. These models are pretrained on large datasets that allows them to detect common everyday objects. Using a pretrained model and adapting it for a new classifcation task is called *transfer learning*. This technique *hotstarts* the training process and greatly reduces the computational time required to perform reliable object detection. Training to detect the target objects mention in Section 1.5 required generating a custom dataset. Research suggested a minimum of 100 images of each *class* be used for successful detection. It was later found after training that extending this dataset to 250 images of each class provided a much more robust detector. Examples of the images used for training are shown in Figure 26. Dataset generation also required labelling each image. This was done using an open-source labelling tool called *Labelmg*.

(a) Example of varying lighting conditions

(b) Example of an occluded image

(c) Example of captured image

Figure 26: Example of noisy backgrounds

Large variations of the dataset prevents *overfitting*, where variation or sample size is too low and the model starts learning to predict the dataset rather than the generalising. Variation included varying light saturation and intensity, noisy backgrounds and partial occlusions of the target object.

Additionally, before training additional *preprocessing* was performed to inflate the dataset. This included randomly converting images to black and white, random rotations of the image and random saturation changes.

This model was trained and Figure 27 shows the *loss* over training steps.

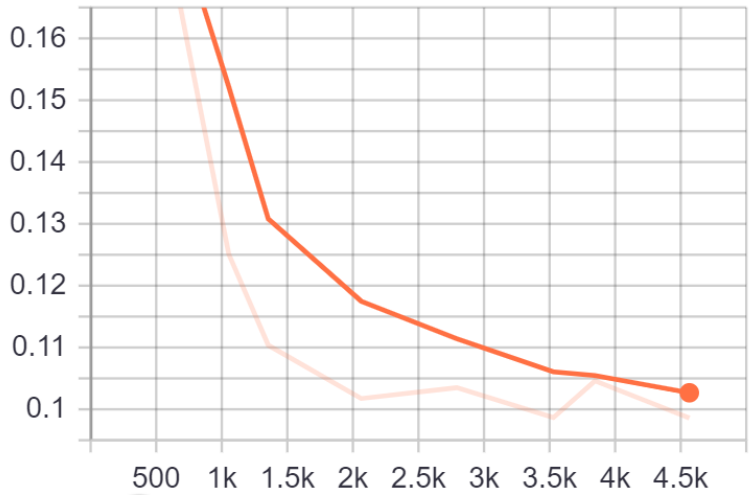


Figure 27: The loss over time as the model trained

The resultant object detector can be seen successfully detecting the desired objects with high accuracy in Figure 28. It was proven to be robust and performed well under natural lighting conditions and partial occlusions.

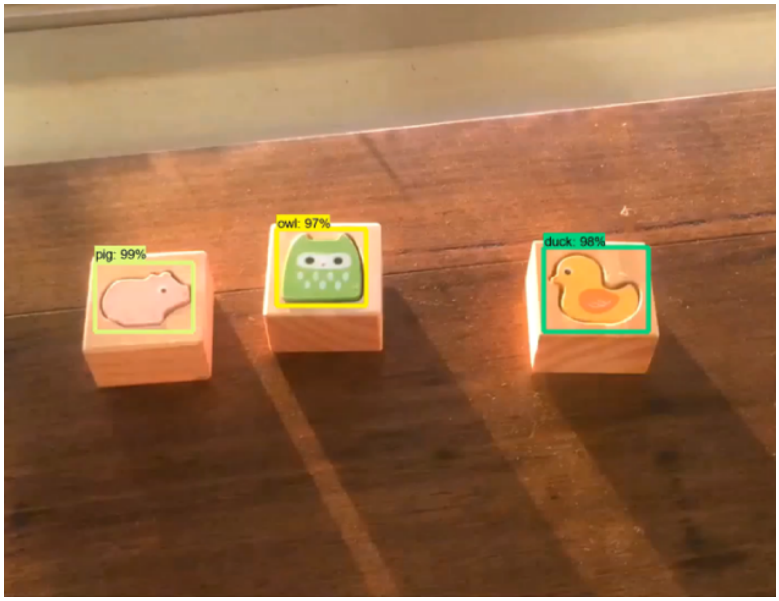


Figure 28: Successful detection of target objects

5. Visual Servoing

Visual servoing aims to mimic the natural skill of hand-eye coordination. Instead of a robot operating in a fixed environment where the location of all objects is known, the robot is now able to operate in dynamic environments. This is facilitated by a visual sensor, which allows a vision-based feedback loop. Now a robot can observe the target object and interact with it without any prior indication of its pose. This approach requires continuous measurement of the targets and controls the robot accordingly, which allows for any errors in the system to be corrected over time.

This can be done using either a camera rigidly attached to the end effector of the robot, as shown in Figure 29 in an *eye-in-hand* configuration. It could also be done using a fixed camera which overlooks the workspace in an *eye-to-hand* configuration shown in the same figure. There are also hybrid approaches, which combine information from multiple cameras in either or both configurations (Tsai et al., 2014).

As specified by the Objective 1.1, the eye-in-hand configuration must be used and will subsequently be explored in this section.

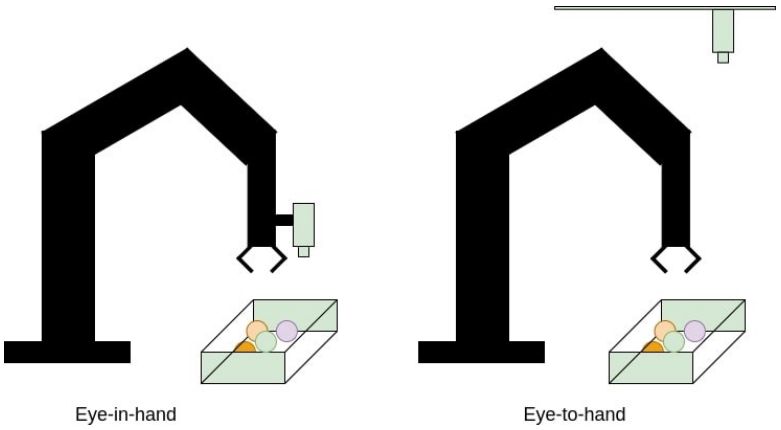


Figure 29: The 2 classes of visual servoing

Additionally, There are various choices of camera that can be used, such as stereo cameras, fish-eye lens cameras and LIDAR units. These can all influence the type and amount of data that is collected. For this project, only a *monocular* camera with a standard lens was considered based on the hardware available.

There are two fundamental approaches to visual servoing. There is position based visual servoing, which extracts feature points from an image, estimates the pose of the target object using those feature points and then performs a control action using the estimated pose. This task takes information from the image space and projects it to the cartesian space to perform control. This differs fundamentally from image based visual servoing, which extracts features from the image and then performs a control action entirely in the image space. There are also hybrid approaches which combine the two methods.

The following subsections will discuss these various approaches.

5.1. Position Based Visual Servoing

Position Based Visual Servoing(PBVS) relies on estimating the pose of an object relative to the camera, then using this information to move to a desired pose relative to the object. This approach requires knowledge of the cameras intrinsic parameters , pose estimation , a joint controller for the robot, feature extraction and a model of the target objects geometry. The control scheme is shown in Figure 30.

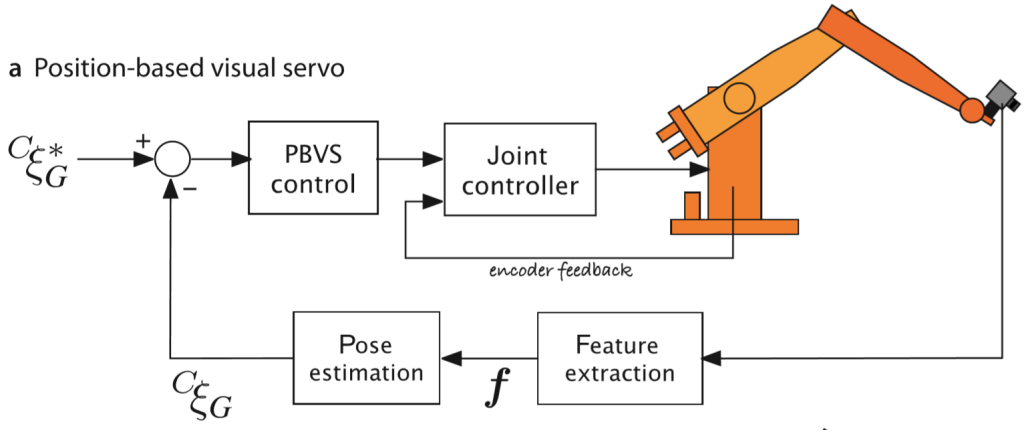


Figure 30: Position Based Visual Servoing control scheme

Using our feature extractor we obtain a point correspondences which relates world points P to image plane points p:

$$\mathbf{P}_n = \begin{bmatrix} X_1 & Y_1 & Z_1 \\ X_2 & Y_2 & Z_2 \\ \vdots & \vdots & \vdots \\ X_n & Y_n & Z_n \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \\ v_n & v_2 & \dots & v_n \end{bmatrix}$$

This correspondence can then be substituted into a pose estimator, which gives the pose of the camera with respect to the world points ${}^{CAM}\mathbf{T}_P$. Given a desired camera pose with respect to the object ${}^{CAM^*}\mathbf{T}_P$, the desired camera transformation can be defined as:

$$\begin{aligned} {}^{CAM}\mathbf{T}_{CAM^*} &= {}^{CAM}\mathbf{T}_P \, {}^{P}\mathbf{T}_{CAM^*} \\ &= {}^{CAM}\mathbf{T}_P \, {}^{CAM^*}\mathbf{T}_P^{-1} \end{aligned} \tag{5.1}$$

This can be used to compute a desired camera velocity $\delta\mathbf{T}_{CAM}$ which is scaled by a control gain λ. This allows for incremental motions:

$$\mathbf{T}_{CAM(k+1)} = \mathbf{T}_{CAM(k)} + \delta\mathbf{T}_{CAM(k)}\lambda \tag{5.2}$$

This can also be expressed as a desired velocity of the camera $\mathbf{v}_{CAM}$ To use this desired velocity in our joint controller, it needs to be transformed from a velocity with respect to the cameras coordinate

frame to a velocity with respect to the origin. This can be done using an adjoint transformation (Appendix D):

$$ {}^{0}\mathbf{ADJ}_{CAM} = \begin{bmatrix} {}^{0}\mathbf{R}_{CAM} & {}^{0}\hat{\mathbf{t}}_{CAM}\,{}^{0}\mathbf{R}_{CAM} \\ \mathbf{0} & {}^{0}\mathbf{R}_{CAM} \end{bmatrix} \tag{5.3} $$

which can be used to compute:

$$ {}_{\{0\}}\mathbf{V}_{CAM} = {}^{0}\mathbf{ADJ}_{CAM}\,{}_{\{CAM\}}\mathbf{V}_{CAM} \tag{5.4} $$

thus, the desired velocity can be provided to the joint controller as:

$$ \mathbf{x}_d = \mathbf{x}_e + \mathbf{V}_{CAM} \tag{5.5} $$

5.2. Image Based Visual Servoing

Image Based Visual Servoing(IBVS) contrasts PBVS by computing its control signal entirely in the image space, as shown by its control scheme in Figure 31.

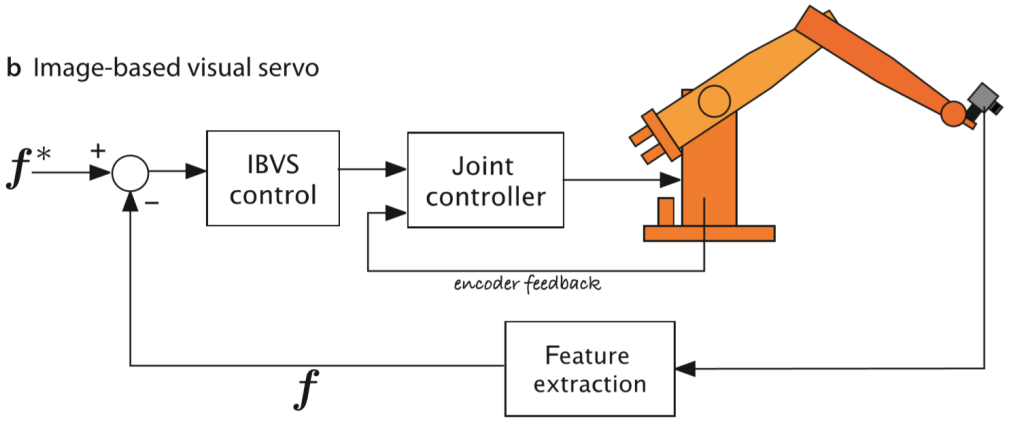


Figure 31: Image Based Visual Servoing control scheme

As before, world points P project to image points p given a camera pose T_{CAM}. Similarly, a desired camera pose T_{CAM^*} will project world points P project to image points p^*. Using this relationship, IBVS aims to move the pixel points p to their corresponding goals p^*, thus implicitly moving the robot to a desired pose T_{CAM^*}.

To perform this control, a method to relate motion in cartesian space to motion in image space is required. This relationship is called the Interaction matrix or, as already defined in this project, the Image Jacobian 3.5. The equation 3.15 can be rearranged to give, for n feature points:

$$\nu = \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ \vdots \\ J_p(p_n, Z_n) \end{bmatrix}^{-1} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ u_n \\ v_n \end{bmatrix} \tag{5.6}$$

This gives us the camera velocity required to achieve a desired motion of the image feature. The simplest way to prescribe this desired motion is to use a simple linear controller

$$\dot{\mathbf{p}}^* = \lambda(\mathbf{p}^* - \mathbf{p}) \tag{5.7}$$

And substitute it into equation 5.6 to give:

$$\nu = \lambda \begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ \vdots \\ J_p(p_n, Z_n) \end{bmatrix}^{-1} (\mathbf{p}^* - \mathbf{p}) \tag{5.8}$$

As the matrix is not square when using the 4 feature points extracted in Section (3.3), the inverse must be computed using the pseudo inverse (Appendix E) which is denoted by $^+$. This results in a velocity that *minimises the norm of the feature velocity error*.

The image jacobian is a first order approximation of the relationship between camera motion and feature points motion. Using a second order approximation will result in faster convergence, and it can be found by taking the *mean* of the pseudo inverse at the current and desired location:

$$\nu = \lambda/2 \left[\begin{bmatrix} J_p(p_1, Z_1) \\ J_p(p_2, Z_2) \\ \vdots \\ J_p(p_n, Z_n) \end{bmatrix}^{+} + \begin{bmatrix} J_p(p_1^*, Z_1^*) \\ J_p(p_2^*, Z_2^*) \\ \vdots \\ J_p(p_n^*, Z_n^*) \end{bmatrix}^{+} \right] (\mathbf{p}^* - \mathbf{p}) \tag{5.9}$$

This velocity is then transformed using an adjoint transformation, as shown in Section 5.1, to give a velocity in the origin frame. This can once again be inputted to a suitable joint controller.

5.3. Simulation

Animated simulation's can be found in the project repository.

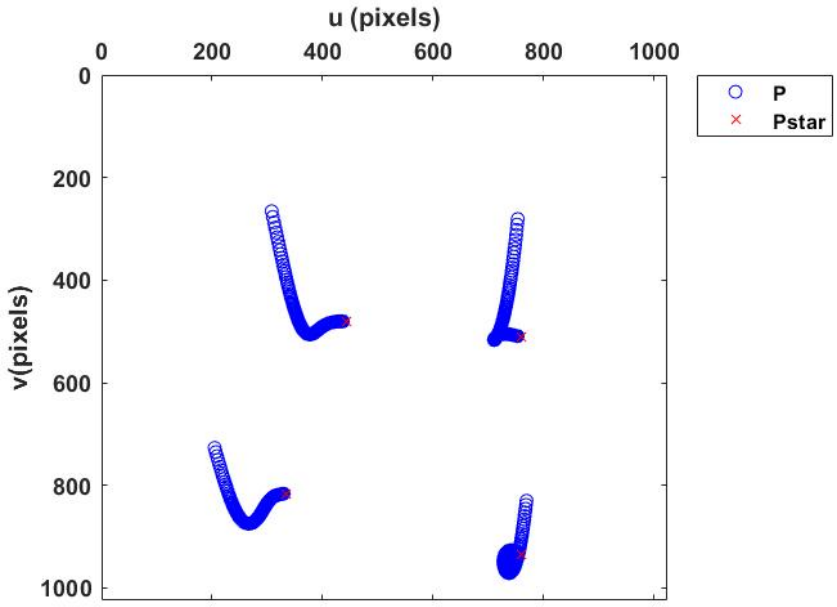


Figure 32: The motion of points in the image space using IBVS

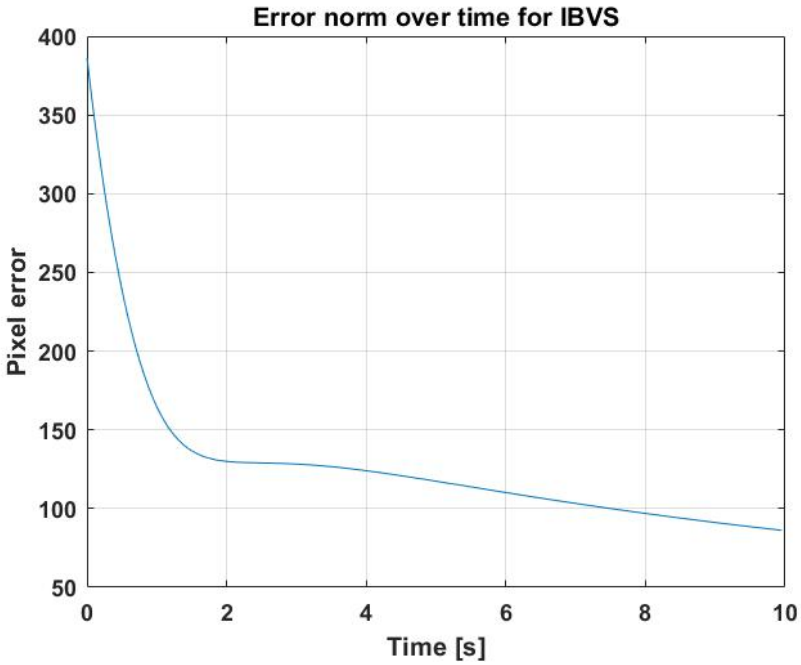


Figure 33: IBVS norm of *pixel error* over time

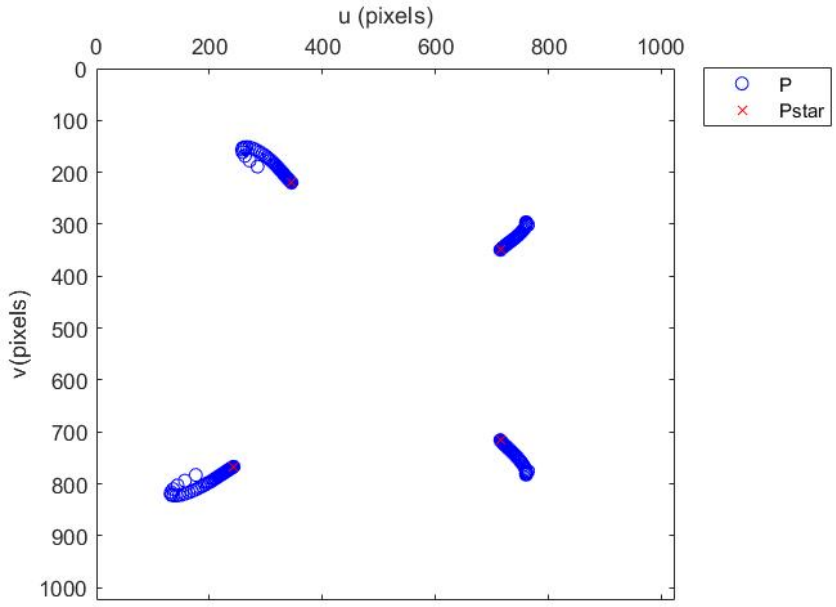


Figure 34: The motion of points in the image space using PBVS

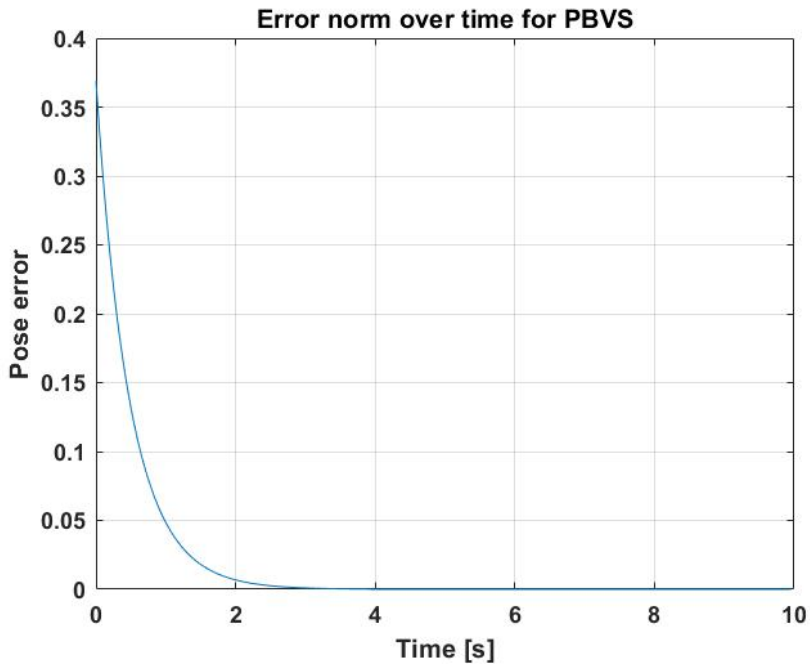


Figure 35: PBVS norm of *pose error* over time

Performing this task on hardware requires *system integration* and is outlined in the following section.

6. Implementation

This section outlines how the theory was applied on hardware. It provides a workflow that *integrates* the subsystems discussed in previous sections,results regarding the pick and place task and suggests future recommendations.

6.1. Workflow

An overview of the workflow used to integrate the systems is providing, relying heavily on the architecture of PBVS (Figure **??**) and IBVS (Figure 31). Common to both formulations is obtaining a desired pose relative to the target object, as shown in Figure 36.

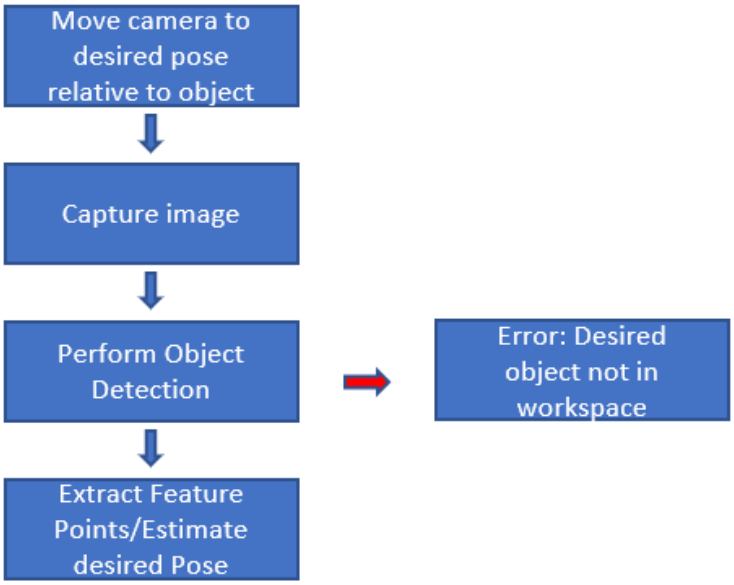


Figure 36: Workflow for defining a desired relative pose.

If the desired object is not detected in the image using our *Faster-RCNN*, an error is reported.

Then either position-Based visual servoing (Figure 37) or image based visual servoing (Figure 38) is performed.

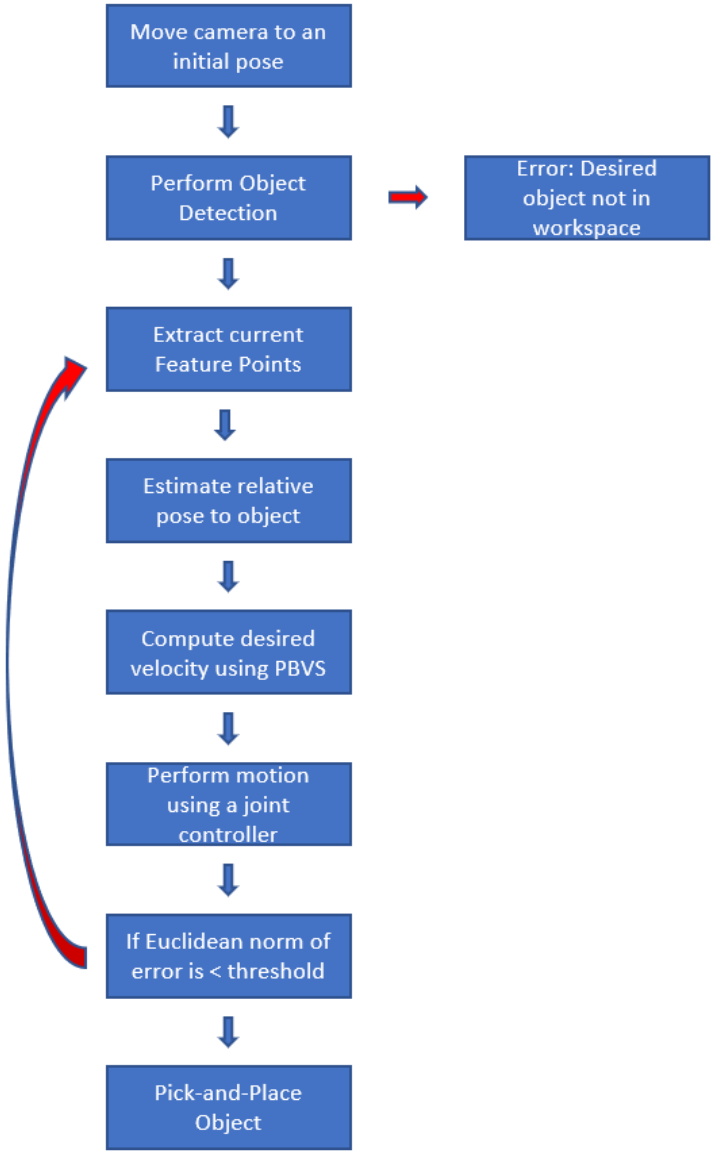


Figure 37: Workflow for PBVS.

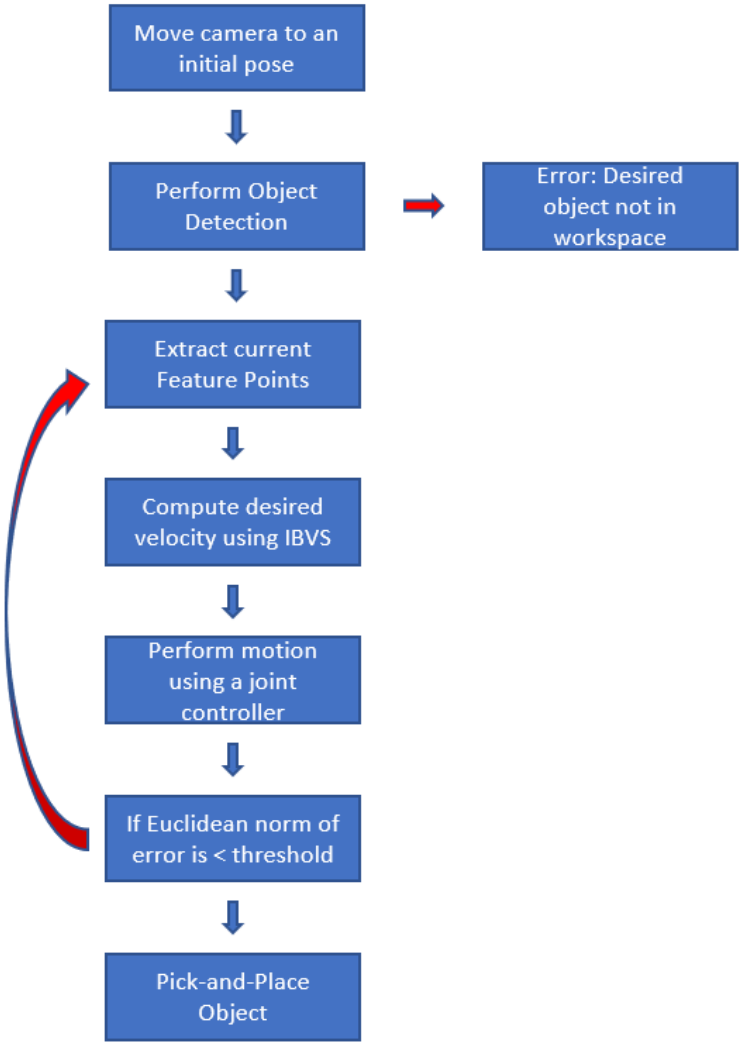


Figure 38: Workflow for IBVS.

where the key difference is whether the task is prescribed in the *image* or *Cartesian* space. Once the error between the current and desired pose falls below a certain threshold, a series of fixed joint movements *relative to the current joint angles* are performed to lower down, grasp, lift and then place the object and then the robot is shut-down. This workflow allows thus integrates our *subsystems* to perform a pick-and place task. Using this workflow achieved the following results.

6.2. Results

A successful demonstration of the pick-and-place task using IBVS control can be visualised in the project repository.

Figure 39 *highlights* a reduction in *pixel error* corresponding to a reducing *pose error*.

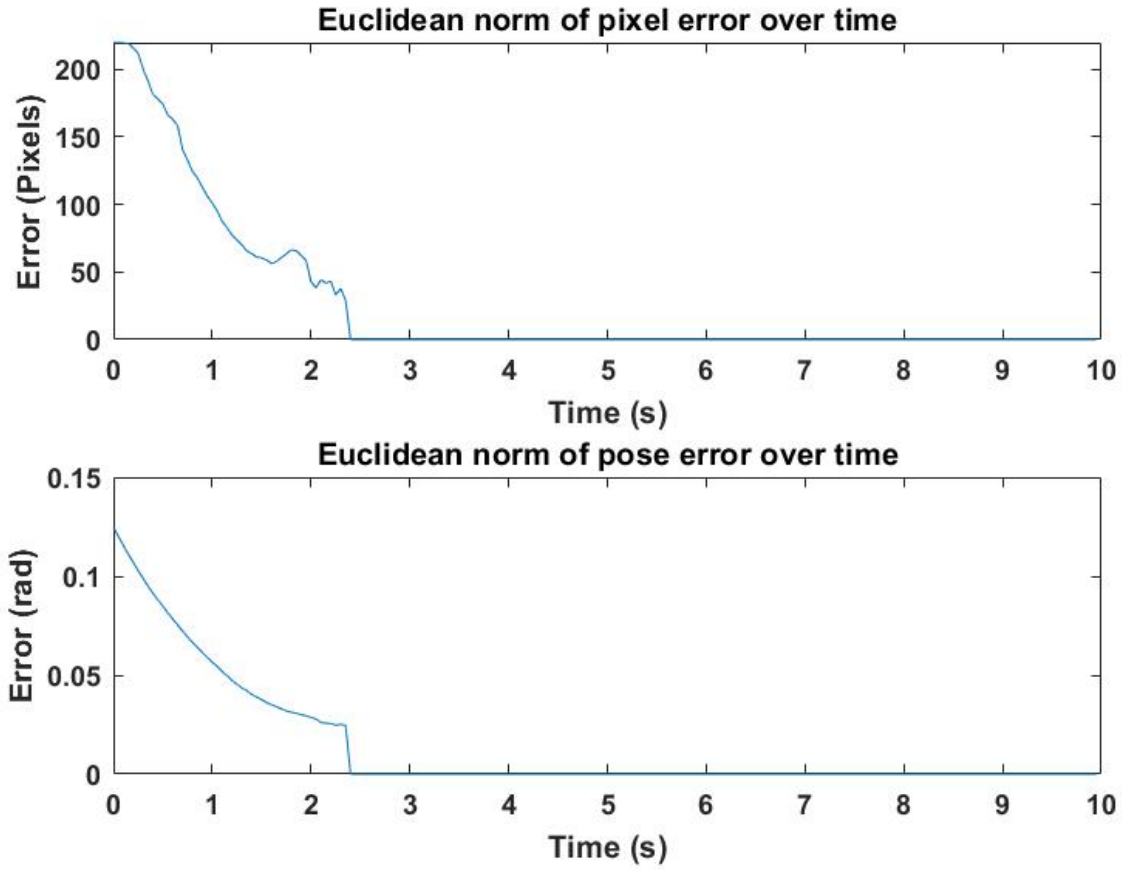


Figure 39: Results for IBVS implemented on hardware

These results will now be discussed in detail and future recommendations made.

6.3. Discussion and Future Recommendations

This project successfully leveraged IBVS and a Faster-RCNN to achieve it's objective. This section provides context for how this was done and suggests future recommendations to improve the workflow.

First and foremost, the 5DOF robot arm was not suitable for the task as it did not have enough *torque* or manipulability to *effectively* implement the workflow designed with a 6DOF manipulator in mind. It is recommended this workflow should either be altered to find better control strategies specific to the 5DOF robot arm or be implemented on a suitable 6DOF arm.

Additionally, the 5DOf arm's limited manipulability range combined with the available webcams substandard *field of view (fov)* restricted the *effective* workspace since both the *current* and *desired* image feature points must be *concurrently* present in the image to initialise the IBVS workflow. This limitation could be overcome by using a larger robot or a camera with a fish-eye lens providing a large *fov*.

As elaborated in Appendix H, other cameras were proposed for this project (including a model with

a suitable fish-eye lens) but were not implemented due to various limitations.

The object detector is currently implemented in Python Tensorflow. A future recommendation is to retrain the classifier using Matlab to simplify integration and implementation.

7. Conclusion

As outlined by the Objective (1.1), the goal of this project was to pick-and-place task a desired object using a robotic manipulator with an eye-in-hand camera. This task was successfully implemented using a image based visual servoing controller (6.2). This was achieved by simplifying this task into various subsystems.

First, a mathematical model was required to understand and control the kinematics of a robotic manipulator. This was done by relating the motion of robotic *joints* to the subsequent movement of it's *end-effector*, leading to the development of a inverse kinematic controller.

Then, a camera model was used to relate 3-dimensional objects to their *projection* in a 2-dimensional image. The *Intrinsic parameters* of a camera were found through *camera calibration* and this allowed us to perform *pose estimation*. Additionally, a technique for extracting *feature points* was established and the mapping of their motion from image to Cartesian space was described using an *image jacobian*.

Subsequently, artificial neural networks were introduced in the context of machine vision. This was used to provide insight into training a *Faster Region-based Convolutional Neural Network* by applying *transfer learning* on a pre-trained model. This resulted in a robust model able to perform in challenging conditions.

Then, pose estimation was used to formulate a *Position Based Visual Servoing* controller while the image jacobian formed the bedrock of a Image Based Visual Servoing controller. The applicability of these techniques was discussed in the context of this project.

Finally, the prior subsystems were integrated on hardware and the project's primary objective was achieved. The various techniques utilised in this project and the hardware limitations faced were discussed and future recommendations were suggested to potentially improve this implementation.

Ultimately, this project explored a few aspects of a challenging control problem and can conceivably form the foundation for further work by future students at the University of Newcastle.

References

Banavar, R., 2008. Robotics, Geometry and Control - Rigid body motion and geometry, Lecture slides.

Bernardini, A., De Fina, S., 1991. A neural network to approximate nonlinear functions. In: [1991] Proceedings of the 34th Midwest Symposium on Circuits and Systems. pp. 545–548 vol.1.

Bouguet, J.-Y., 2004. Camera calibration toolbox for matlab. http://www. vision. caltech. edu/bouguetj/calib_doc/index. html.

Choi, J.-w., Curry, R., Elkaim, G., 2008. Path planning based on bézier curve for autonomous ground vehicles. In: Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008. IEEE, pp. 158–166.

Corke, P., 2017. Robotics, vision and control: fundamental algorithms in MATLAB® second, completely revised. Vol. 118. Springer.

FlirSystems, 2020. Cmln-13s2c-cs usb2 camera, photograph. https://www.flir.com.au/products/usb2-cameras/, viewed: 18/07/2020.

Gans, N. R., Hutchinson, S. A., 2007. Stable visual servoing through hybrid switched-system control. IEEE Transactions on Robotics 23 (3), 530–540.

Hartenberg, R., Danavit, J., 1964. Kinematic synthesis of linkages. New York: McGraw-Hill.

Hartenberg, R. S., Denavit, J., 1955. A kinematic notation for lower pair mechanisms based on matrices. Journal of applied mechanics 77 (2), 215–221.

Hubel, D. H., Wiesel, T. N., 1959. Receptive fields of single neurones in the cat's striate cortex. The Journal of physiology 148 (3), 574.

Ke, T., Roumeliotis, S. I., 2017. An efficient algebraic solution to the perspective-three-point problem. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 7225–7233.

Krizhevsky, A., Sutskever, I., Hinton, G. E., 2012. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105.

Li, S. Z., 2009. Encyclopedia of Biometrics: I-Z. Vol. 2. Springer Science & Business Media.

Microscan, 2020. The features of blobs in images. https://files.microscan.com/helpfiles/visionscapetools_help_file/visionscape_tools_reference-09-021.html, viewed: 18/07/2020.

Moore, E. H., 1920. On the reciprocal of the general algebraic matrix. Bull. Am. Math. Soc. 26, 394–395.

Nomura, H., Naito, T., 2000. Integrated visual servoing system to grasp industrial parts moving on conveyer by controlling 6dof arm. In: Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0. Vol. 3. IEEE, pp. 1768–1775.

Penrose, R., 1955. A generalized inverse for matrices. In: Mathematical proceedings of the Cambridge philosophical society. Vol. 51. Cambridge University Press, pp. 406–413.

Renton, C., 2019a. Robotic Manipulator Kinematics I, Mechatronic Design II (MCHA 3900) Lecture slides.

Renton, C., 2019b. Robotic Manipulator Differential Kinematics, Mechatronic Design II (MCHA 3900) Lecture slides.

RoboDK, 2020a. Python api examples. https://robodk.com/Python-API-examples, viewed: 18/07/2020.

RoboDK, 2020b. Robot modelling software. https://robodk.com/, viewed: 18/07/2020.

Shukla, L., 2020. Designing your neural networks. https://www.kdnuggets.com/2019/11/designing-neural-networks.html, viewed: 18/07/2020.

Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G., 2010. Robotics: modelling, planning and control. Springer Science & Business Media.

Spratling, M. W., Cipolla, R., 1996. Uncalibrated visual servoing.

Tensorflow, 2020. Tensorflow github repository. https://github.com/tensorflow/models/tree/master/research/object_detection, viewed: 18/07/2020.

Tsai, C.-Y., Wong, C.-C., Yu, C.-J., Liu, C.-C., Liu, T.-Y., 2014. A hybrid switched reactive-based visual servo control of 5-dof robot manipulators for pick-and-place tasks. IEEE Systems Journal 9 (1), 119–130.

Zhang, Z., 12 2000. A flexible new technique for camera calibration. Pattern Analysis and Machine Intelligence, IEEE Transactions on 22, 1330 – 1334.

A. Time Log

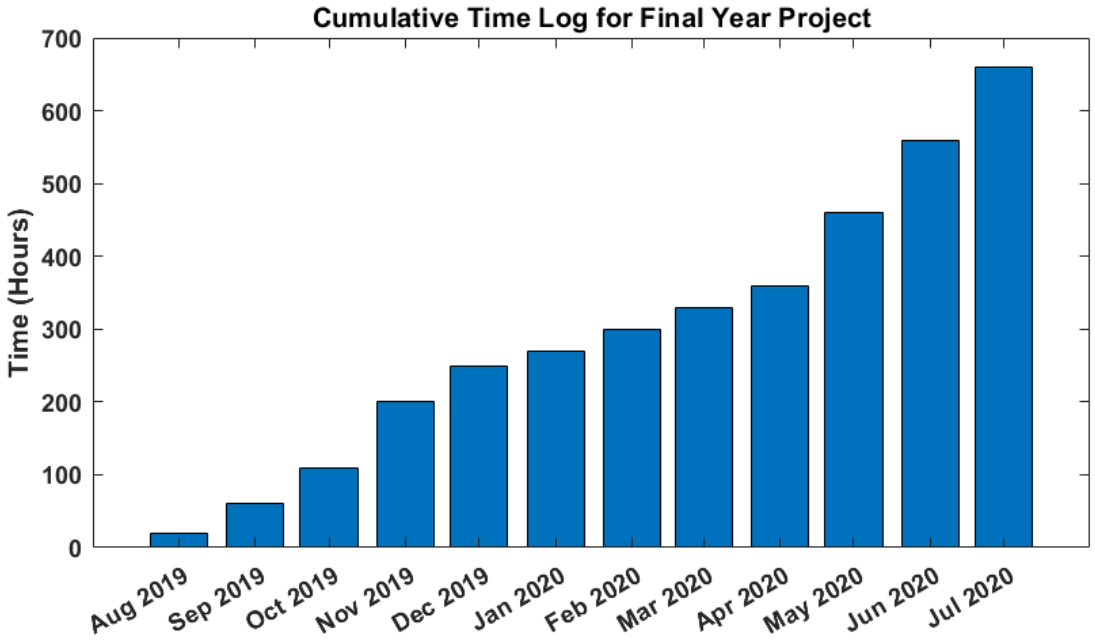


Figure 40: Cumulative Time Log

B. Engineers Australia Competencies

Undertaking this project allowed me to develop competency in the following areas, as required by the "Stage 1 Competency Standard for Professional Engineer" by Engineers Australia.

B.1. Knowledge and Skill Base

This project has developed my understanding of mathematical modelling of physical systems and control theory. It also increased my skills in programming and ability to solve real-world problems.

B.2. Engineering Application Ability

I utilised the engineering method to solve a complex problem by dividing it into simpler sub-problems. I then tackled each sub-problem by researching suitable techniques, synthesising viable solutions and iteratively testing and improving them.

B.3. Professional and Personal Attributes

This project allowed me to improve my project management skills, progressing multiple codependent tasks and finding feasible solutions to a plethora of issues. I also honed my ability to learn new skills through independent research and developed an appreciation for effective communication of complex ideas.

C. Industrial Application

The following provides an example of an application of Visual-servoing used in industry.

Ampcontrol CSM is a site that manufactures and assembles various electrical systems. Many of these systems contain printed circuit boards, which require a layer of protective coating to protect them from harsh environments that may contain high humidity, varying temperatures and airborne particles that may disrupt the circuits functionality.

The current approach to conformal coating on-site is to perform a bulk spray of several printed circuit boards (PCBs) using an aerosol can of polyurethane coating. This process involves covering parts, such as pins, connectors and potentiometers that must be coating-free to remain functional, with tape and vinyl coating. These are then placed in a well-ventilated room, bulk sprayed by production staff and then left to cure overnight. The tape and vinyl are then removed, and the board is inspected.

This process is labour intensive and exposes workers to potentially hazardous fumes. Therefore, an alternative method was proposed and developed which aimed to automate the process.

This method uses a pneumatic nozzle attached to a *Techman Robot TM5-900* collaborative robot arm to autonomously spray the desired PCBs while avoiding parts that must remain coating free.

The developed workflow allows the designation of a desired board are that needs to be coated as a mechanical layer in Altium (industrial PCB design software). This is then converted to a *tool-path* and simulated through RoboDK (Robotics modelling software) for verification. The specified tool-path can then be used to guide the robot arm and operate it's pneumatic dispenser to coat a desired PCB.

The pneumatic dispenser's accuracy, and thus the success of the entire task, is dependent on precise knowledge of the board's position and orientation. This could not practically be achieved through the use of jigs as there were hundred's of different board shapes and sizes. Therefore, visual-servoing was used to *align* the dispenser to the board. This was achieved by using Techman Robot's proprietary *Tmflow* software which has a visual-servoing implementation.

The robot arm successfully demonstrated the task as a proof of concept, highlighting the potential of vision-based control in industry.

D. Adjoint transformation

The adjoint transformation was defined by Banavar (2008) and is used to transform a velocity ${}^{B}\nu = [{}^{B}\mathbf{v}, {}^{B}\omega]^{\top}$ from the basis frame $\{B\}$ to the basis frame $\{A\}$. This is achieved through the relationship:

$$\begin{bmatrix} {}^{A}\mathbf{v} \\ {}^{A}\omega \end{bmatrix} = \begin{bmatrix} {}^{A}\mathbf{R}_{B} & {}^{A}\hat{\mathbf{t}}_{B}\,{}^{A}R_{B} \\ \mathbf{0}^{3\times 3} & {}^{A}\mathbf{R}_{B} \end{bmatrix} \begin{bmatrix} {}^{B}\mathbf{v} \\ {}^{B}\omega \end{bmatrix} \tag{D.1}$$
$${}^{A}\nu = \mathbf{Ad}({}^{A}\mathbf{T}_{B})\,{}^{B}\nu$$

where ${}^{A}\hat{\mathbf{t}}_{B}$ indicates the *skew* of the translation vector ${}^{A}\mathbf{t}_{B} = [{}^{A}X_{B}, {}^{A}Y_{B}, {}^{A}Z_{B}]^{\top}$ as:

$$ {}^{A}\hat{\mathbf{t}}_{B} = \begin{bmatrix} 0 & {}^{A}Z_{B} & {}^{A}Y_{B} \\ {}^{A}Z_{B} & 0 & -{}^{A}X_{B} \\ -{}^{A}Y_{B} & -{}^{A}X_{B} & 0 \end{bmatrix} \tag{D.2} $$

and $\mathbf{Ad}({}^{A}\mathbf{T}_{B})$ is *adjoint transform* expressed as a function of the desired basis transformation.

E. Moore-Penrose Inverse

The Moore-Penrose inverse is defined by Moore (1920) and Penrose (1955). They state, given a matrix $\mathbf{A}$ whose rows are linearly independent:

$$ \mathbf{A}^{+} = \mathbf{A}^{\top}(\mathbf{A}\mathbf{A}^{\top})^{-1} \tag{E.1} $$

Where $\mathbf{A}^{+}$ is the *right inverse* as $\mathbf{A}\mathbf{A}^{+} = \mathbf{I}$.
This is known as a *pseudoinverse* and is denoted by $^{+}$. In cases where a matrix is not invertible (non-square), the pseudoinverse provides a least squares approximation of its inverse and is implemented by the `pinv` function in Matlab.

F. Optimisation of sorting problem

When performing a pick and place problem for multiple objects, we can represent the task as having items $D = [A, B, C]$ at initial positions $\{\mathbf{d}_0\}$ and having desired locations $\mathbf{d}^{*}$. If the order in which these items are picked and place matters, then there is no further work required. However, if the *order does not matter*, then it is purported there must be an optimal method for sorting these objects that minimises the *total displacement* of the end effector.

Fortunately, this task can be formulated as a subset of the *travelling salesman problem* and a solution can be found using a linear optimiser.

This was implemented in Matlab using `intlinprog` as shown by Figure 42.

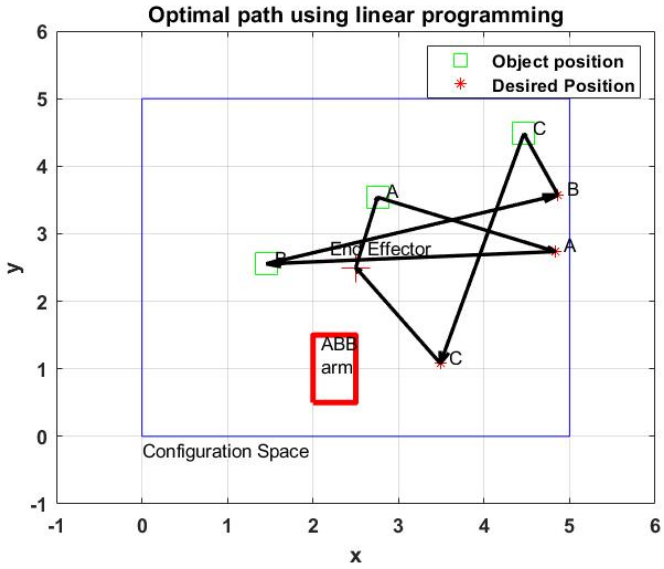


Figure 41: The optimal path to sort objects $[A, B, C]$ to desired locations $[A*, B*, C*]$

Figure 42: A minimal displacement path using linear programming

It was intended for this work to be extended to 3 dimensions and form part of a trajectory generator using bezier curves (Choi et al., 2008).

G. Euler Angles

Rotation matrices are non-minimal representations of orientation and have redundant parameters. For the special orthogonal group (SO3), 3 parameters can be used to provide a minimal representation Siciliano et al. (2010).

Following the Roll-Pitch-Yaw (RPY) convention used in aeronautics, rotations are considered to occur, **in order**, about the z , y and x axis. Rotations can be denoted as roll (ϕ), pitch (θ) and yaw (ψ). The rotation matrix 2.1 can now be expressed as, for θ in the range $(\frac{-\pi}{2}, \frac{\pi}{2})$:

$$\mathbf{\Theta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(-R_{31}, \sqrt{R_{33}^2 + R_{33}^2}) \\ atan2(R_{21}, R_{11}) \end{bmatrix} \tag{G.1}$$

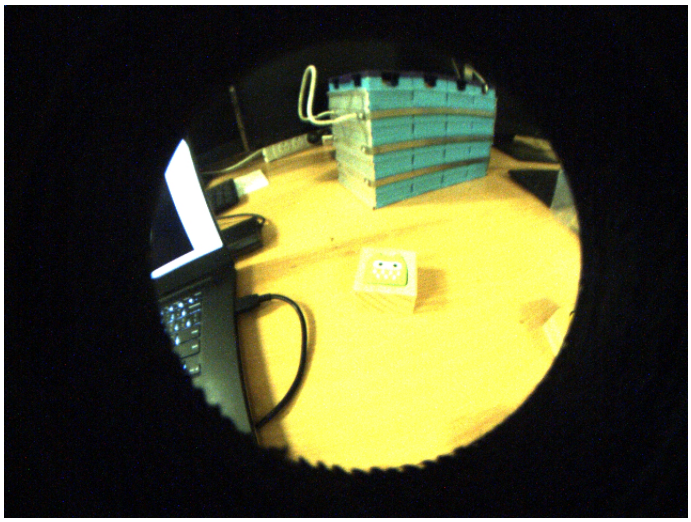
and for θ in the range $(\frac{\pi}{2}, \frac{3\pi}{2})$:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2(R_{32}, R_{33}) \\ atan2(-R_{31}, \sqrt{R_{33}^2 + R_{33}^2}) \\ atan2(R_{21}, R_{11}) \end{bmatrix} \tag{G.2}$$

H. Alternative Cameras

The first camera considered for use during this project was a *Point Gray chameleon cmln-13s2c* camera with a fish-eye lens. Due to it's protective casing limiting its field of view, low frame-rate and poor picture quality when interfaced with Matlab (Figure 43), this camera was deemed unfit for the task.

(a) The camera's standard case (protective casing not shown), retreived from FlirSystems (2020)



(b) Example of captured image

Figure 43: The Point Gray chameleon cmln-13s2c

Therefore, a *Raspberry Pi Zero W* intefaced with a *Raspberry Pi Camera Board v2* was acquired. This setup provided a high resolution (1280x720) image at 60fps and could be directly interfaced with Matlab. This camera was used to generate the training and validation dataset for object detection and was also calibrated using camera calibration.

Unfortunately, a suitable mount to integrate it with the 5DOF robot arm was not manufactured due to time constraints and weight concerns due to actuator torque limits.

Thus, the Microsoft webcam was used as it could be easily mounted to the robot and was light enough to minimise joint current overloading.