VISUAL SERVO BASED SPACE ROBOTIC DOCKING FOR ACTIVE SPACE DEBRIS REMOVAL

Sukhjinder Lal

A thesis submitted to the Faculty of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Science

Graduate Program in Mechanical Engineering

York University

Toronto, Ontario

December 2021

Abstract

This thesis developed a 6DOF pose detection algorithm using machine learning capable of providing the orientation and location of an object in various lighting conditions and at different angles, for the purposes of space robotic rendezvous and docking control. The computer vision algorithm was paired with a virtual robotic simulation to test the feasibility of using the proposed algorithm for visual servo. This thesis also developed a method for generating virtual training images and corresponding ground truth data including both location and orientation information. Traditional computer vision techniques struggle to determine the 6DOF pose of an object when certain colors or edges are not found, therefore training a network is an optimal choice. The 6DOF pose detection algorithm was implemented on MATLAB and Python. The robotic simulation was implemented on Simulink and ROS Gazebo. Finally, the generation of training data was done with Python and Blender.

Acknowledgements

I would like to start off this thesis by thanking my supervisor Dr. Zheng Hong Zhu for welcoming me into his lab group in 2019 and being patient with the work I did. It was a tough time over the last two years, with the COVID-19 pandemic. It pushed a lot of my research back, but Dr. Zhu was patient and gave me the time to complete my work to the best of my ability and to a high standard. Dr. Zhu provided a wealth of knowledge that went a long way and inspired confidence in my work. Thank you, Professor.

I would also like to extend a thank you to Dr. Dan Zhang for being on my committee and providing me with help in robotics and advice on my thesis. I would also like to thank Dr. Franz Newland for helping me during my master's with support and feedback.

I would also like to thank my lab group, who I got to know a lot better and worked with closely during this journey, for helping me and pushing to get the best out of me. Aniket Prabhudesai, Rajika Pati Arambage, Lucas Santaguida, Ahmad Alzoubi, Bahador Beigomi, and Ping Li; I would like to thank you all for your help and support.

Finally, if it were not for the support my parents and brother gave me, I would not have come this far. Thank you, I hope I made you proud.

Table of Contents

Abstract ii

Acknowledgements iii

Table of Contents iv

List of Figures vii

List of Tables x

Symbols and Conventions xi

Chapter 1: Introduction and Justification 1

1.1 Space Robotics 1

1.2 Scope of Research 3

1.3 Justification of Research 3

1.3.1 Space Debris and On-Orbit Servicing 3

1.3.2 6DOF Pose Estimation 5

1.3.3 Limitations of Existing Methods 7

1.4 Objectives of Research 8

1.5 Methodology of Approach 8

1.6 Layout of Thesis Document 10

Chapter 2: Literature Review 12

2.1 Computer Vision 12

2.1.1 Object Detection 12

2.1.2 Object Orientation Estimation 15

2.2 Debris Removal and Space Robotics 21

2.3 Conclusion 25

Chapter 3: 6DOF Pose Detection 26

3.1 Introduction 26

3.2 Object Detection 26

3.3 Pose Detection 36

3.4 Pipeline 42

3.5 Image Parameters 44
3.6 Experimentation and Results 45
Chapter 4: Board Deployment 49
4.1 Introduction 49
4.2 Board Hardware 51
4.3 Experimentation and Results 51
Chapter 5: Creating Virtual Training and Testing Data 54
5.1 Introduction 54
5.2 Advantages of Using Virtual Data 55
5.3 Methodology 57
Chapter 6: Robot Control 63
6.1 Introduction 63
6.2 Robot Specifications 65
6.3 Limitations on the Robot 67
6.4 Robot Equations 67
6.5 Control 86
6.6 Simulink Environment Setup 91
6.7 Experimentations and Results 98
Chapter 7: Conclusion and Future Work 103
7.1 General Conclusions 103
7.1.1 Thesis Accomplishments 103
7.1.1.1 Neural Network for Object Detection 103
7.1.1.2 Development of a Custom Neural Network for Pose Detection 103
7.1.1.3 Deployment of 6DOF Pose Detection Network on a Microcomputer 104
7.1.1.4 Development of an Algorithm for Custom Virtual Data Generation 104
7.1.1.5 Validation of a Software-in-the-Loop Simulation for Robotic Rendezvous and Docking 105
7.2 Contributions of Thesis Work 105

7.3 Future Work 106
7.3.1 Hardware Setup and Integration 106
7.3.2 Re-Training the Object and Pose Detection Network 108
7.3.3 Obstacle Avoidance 109
7.3.4 Improving Joint Controllers 109
References 110

List of Figures

Figure 1.1: The multiple phases of a space rendezvous procedure. Not drawn to scale. .. 2

Figure 1.2: An example of the architecture of an artificial neural network. Included are the input layer, multiple hidden layers, and an output layer. 7

Figure 1.3: Methodology of the parallel development of the computer vision algorithm and robot simulation for a virtual implementation. 10

Figure 3.1: An example of anchor boxes tiled across an image. 27

Figure 3.2: The YOLOv2 network architecture. 28

Figure 3.3: A funnel was used as the target due to its similarity to a thruster nozzle...... 29

Figure 3.4: Augmentation of training data to increase the size of the training data set... 31

Figure 3.5: The origin and orientation of a pixel frame relative to an image. 32

Figure 3.6: A depth frame with the funnel shown inside the red ellipse. 33

Figure 3.7: Camera world frame origin and orientation relative to a camera sensor....... 33

Figure 3.8: The initial simulation setup with the Kinect camera and a target, and then the camera docking with the target. 35

Figure 3.9: Examples of object detection from the YOLOv2 network. 36

Figure 3.10: Detecting a physical funnel and retrieving the 3D coordinates of the funnel. 36

Figure 3.11: The custom pose regression network architecture. 37

Figure 3.12: Model accuracy vs. training epoch of the training and validation data. 38

Figure 3.13: Model loss vs. training epoch of the training and validation data. 39

Figure 3.14: The quaternion orientation of the funnel is [-0.541, -0.159, -0.249, 0.787].40

Figure 3.15: On the left is a $Rx30Ry30Rz30$ rotation, on the right is a $Ry30Rx30Rz30$ rotation. 42

Figure 3.16: Example of an output from the pose regression network after a prediction.42

Figure 3.17: Architecture of the custom 6DOF pose detection pipeline. 43

Figure 3.18: The image on the left had the spotlight to the right of the camera, the image in the middle had the spotlight placed above the object, and the image on the right had the spotlight placed above the camera. 45

Figure 3.19: The area of overlap is the blue region; the area of union is the gray and blue regions combined. 45

Figure 3.20: The precision-recall plot of the YOLOv2 network. 47

Figure 4.1: An image of the NVIDIA Jetson Nano board [90]. 50

Figure 4.2: The NVIDIA Jetson Nano with all connections during the experiment. 52

Figure 4.3: A wider view of the experimental setup with the camera. 52

Figure 5.1: An example of real images used for initially training a network without ground truth labels. 56

Figure 5.2: An example of real images used for initially training a network without ground truth labels. 56

Figure 5.3: An example of real images used for initially training a network without ground truth labels. 56

Figure 5.4: Custom STL file of a funnel. 57

Figure 5.5: Scaling of the virtual funnel. 58

Figure 5.6: Environment setup in Blender. The red line is the X-axis, the green line is the Y-axis, and the blue line is the Z-axis. 58

Figure 5.7: Default orientation of the funnel. 59

Figure 5.8: Automated ground truth label around the virtual funnel in a training image. 61

Figure 5.9: An example of Euler outputs from Blender (left) converted to their respective Quaternions (right). 62

Figure 6.1: The proposed setup for the FANUC robots at the Space Engineering Design Lab at York University. 64

Figure 6.2: The FANUC M20iD/25 robot [95]. 65

Figure 6.3: The motion range of the FANUC M20iD/25 robot [97]. 66

Figure 6.4: A planar 2 link kinematic arm. 68

Figure 6.5: Parameters used with the DH method for forward kinematics [98]. 69

Figure 6.6: The joint frames for the DH method. As seen in Figure 6.3, all frames originate in the same plane (x_0-z_0 plane). 71

Figure 6.7: Another image of the derived DH frames for the FANUC M20iD/25 robot. 71

Figure 6.8: The general form of a PI controller. 89

Figure 6.9: The various types of responses from a controller output [103]. 90

Figure 6.10: Method of operations of the developed simulation algorithm. 91

Figure 6.11: The first part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot. 93

Figure 6.12: The second part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot. 94

Figure 6.13: The third part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot. 94

Figure 6.14: The PI Joint Controller block in more detail. 95

Figure 6.15: The Fanuc Robot block expanded in more detail. 95

Figure 6.16: The Configure ROS Network Address function in detail. 97

Figure 6.17: The images taken by the Kinect Camera inside ROS Gazebo. 99

Figure 6.18: Joint controller response for joint 1. 100

Figure 6.19: Joint controller response for joint 2. 101

Figure 6.20: Joint controller response for joint 3. 101

Figure 6.21: Joint controller response for joint 4. 101

Figure 6.22: Joint controller response for joint 5. 102

Figure 6.23: Joint controller response for joint 6. 102

Figure 7.1: The DLR's OOS docking simulation setup, similar to what is being planned at the Space Engineering Design Laboratory at York University [104]. 108

List of Tables

Table 1.1: The various hardware devices used in this thesis work and their specifications. 9

Table 3.1: The errors in the X, Y, and Z axis of the pose regression network.......... 48

Table 5.1: The ranges of orientation of the funnel used in each axis during data generation. 59

Table 5.2: The ranges of the position of the spotlight used in each axis during data generation.......... 60

Table 5.3: Method for finding the different coordinates of a bounding box.......... 61

Table 6.1: Joint ranges and maximum joint speeds of the FANUC M20iD/25 robot...... 65

Table 6.2: Joint limits imposed on the FANUC M20iD/25 robot.......... 67

Table 6.3: The DH table for the FANUC M20iD/25 robot.......... 72

Table 6.4: The coordinates of the target relative to the camera and robot base in the setup seen in Figure 3.8.......... 99

Table 6.5: The joint values obtained from the inverse kinematics block in Simulink from the setup seen in Figure 3.8.......... 100

Symbols and Conventions

All units of measurement used in this thesis are given in SI units, unless otherwise stated.

All vectors are enclosed with the following curly brackets { }, while matrices are enclosed with the following square brackets [ ].

List of Abbreviations

2D	2 Dimensions
3D	3 Dimensions
ADR	Active Debris Removal
AI	Artificial Intelligence
ANN	Artificial Neural Network
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
CSA	Canadian Space Agency
CUDA	Compute Unified Device Architecture
CV	Computer Vision
CVFH	Clustered Viewpoint Feature Histogram
DH	Denavit-Hartenberg
DLR	German Aerospace Center
DOF	Degrees of Freedom
EPOS	European Proximity Operations Simulator
EVA	Extravehicular Activities
FPS	Frames per Second
GB	Gigabyte
GEO	Geostationary Orbit

GPU	Graphics Processing Unit
HIL	Hardware-in-the-Loop
IP	Internet Protocol
IBVS	Image Based Visual Servo
IoU	Intersection over Union
LEO	Low Earth Orbit
LSTM	Long Short-Term Memory
mAP	Mean Average Precision
NASA	National Aeronautics and Space Administration
NIR	Near Infrared
OOS	On-Orbit Servicing
PBVS	Position Based Visual Servo
PI	Proportional-Integral
PnP	Perspective-n-Point
R-CNN	Region Based Convolutional Neural Network
RGB	Red Green Blue
RGB-D	Red Green Blue and Depth
ROS	Robot Operating System
RPN	Region Proposal Network
SIL	Software-in-the-Loop
SSD	Single Shot Detector
STL	Standard Tessellation Language
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
UV	Ultraviolet
YOLO	You Only Look Once

Chapter 1: Introduction and Justification

Summary: This chapter defines the problem, justifies the undertaking of this research, and outlines the various approaches used to complete set objectives. A layout of this thesis is provided as well.

1.1 Space Robotics

Space Robotics is increasingly used for autonomous space debris removal and on-orbit servicing (OOS). With OOS, the possible applications for space robotics consist of assembly, maintenance, repair, deployment, releasing, retrieving, extra-vehicular activity (EVA) support, inspection, refueling, and multi-arms cooperation [1]. With the space debris population increasing, studies show active space debris (ADR) missions are needed to halt the growth of low earth orbit (LEO) debris collisions [2]. There are disadvantages to space robotics missions though, such as high development and manufacturing costs [3], but it has been shown that OOS is economically feasible by repairing satellites rather than launching new ones, and de-orbiting them with robot manipulators if necessary [1].

To successfully service or grasp a target, four phases are completed as part of the rendezvous procedure: (1) launch and early orbit, (2) far range rendezvous (5 km to 300 m from the target), (3) close range rendezvous (less than 300 m from the target), and (4) capturing &/or removal [1] [4] [5]. These can be seen in Figure 1.1. The European Proximity Operations Simulator (EPOS) is a facility in Germany that studies phases 3 and 4 [4] [6]. To control a robot in space, four approaches are used: (1) fixed spacecraft attitude and position, (2) fixed spacecraft position, (3) free-floating system, and (4) free-flying systems [7] [8]. A free-floating system has its advantages, such as less fuel consumption leading to longer mission life [8].

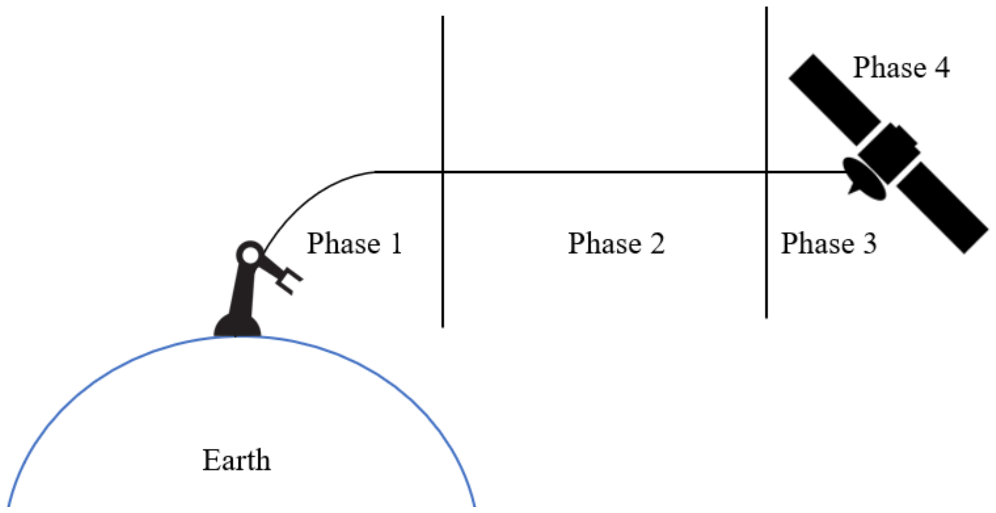


Figure 1.1: The multiple phases of a space rendezvous procedure. Not drawn to scale.

Since current space robotics methods consist of teleoperation and semi-autonomous operations [9], space robotics research is now shifting towards autonomy. Timing and communication are large challenges that autonomy can overcome [10]. Operations in LEO need autonomy due to blackout periods from Earth ground stations and operations in the geostationary orbit (GEO) need autonomy due to time delays [6]. With that said, autonomous systems must be capable of identifying hazards in their surrounding environment and on-board algorithms must be well tested [9]. Pre-verification and simulations, along with highly confident computer vision algorithms are necessary for autonomy to take the next step and to assist in solving the problem of space debris [11].

The research question posed in this study is: How can we efficiently rendezvous with a target for on-orbit servicing or removal using visual servo based autonomous space robotics.

1.2 Scope of Research

While the development of a hardware system capable of rendezvous and docking is necessary to assist with the problem of space debris and on-orbit servicing, a software-in-the-loop (SIL) system provides a way to test capabilities. The aim of this study is to develop a 6DOF pose detection algorithm and simulated visual servo robot control.

The scope of this study is limited to the simulation environment due to the limit on access to lab in the COVID-19 pandemic period. The 6DOF pose detection algorithm will be developed without the use of a physical target object. The visual servo robot control software will be developed using a simulated computer vision algorithm, focused on close range rendezvous and a free-floating robot. Tests will be completed with virtual data and virtual simulations. The only physical hardware used for testing is the NVIDIA Jetson Nano, used for testing the deployment capabilities of the 6DOF pose detection algorithm on a microcomputer.

1.3 Justification of Research

1.3.1 Space Debris and On-Orbit Servicing

Space debris consists of non-functional man-made objects without the capability of resuming their functions taking up space in orbits [12]. These pieces are often non-cooperative and are therefore differ from targets for OOS [13]. Debris is either (1) mission-related, (2) accidental, or (3) intentional [14]. Mission-related debris consists of payload shrouds, launch vehicle stages, aluminum oxide due to solid-rocket motors, leaking thermal-control systems, explosive bolts, instrument covers, adapter rings, and paint or surface materials from ultraviolet (UV) radiation or oxidation [12] [15]. During launch, spacecraft payloads only account for a fraction of the mass brought into space [12]. Accidental debris consists of items lost during EVAs by astronauts or unintentional explosions or collisions [14]. Intentional debris can accumulate from experiments meant to test satellite weapon capabilities, like the tests completed by the United States of America and Soviet Union in the 20th century [14].

As of 2015, there was a tracked orbital debris population of 33, 323; of which 17, 000 were larger than 10 cm. Orbits of 800 km in LEO are found to be the most crowded, whereas orbits near 600 km, 800 km, and 1000 km are considered to have the largest mass of debris. Orbits at inclinations of 82.5° to 83.5° and at altitudes of 900 km to 1050 km are primary targets for debris removal [4]. A 10 cm piece of debris traveling at the average speed of 10 km/s, the average speed of tracked debris, can cause damage equivalent to 7 kg of TNT in a collision [8].

There are hazards to the space debris population increasing over the years since Sputnik-1 was launched. The probability of a collision in 2013 with an object larger than 1 cm was 50% - 67% per year, an increase from 2007 (25% - 33% per year) [14]. As probability increases, an increase in the debris population is a result, even if new launches are halted, which is the theory behind the Kessler Syndrome [4]. Some other hazards of debris include collisions in space and debris falling back to Earth [12] [15]. To halt this, many methods for space debris removal have been studied such as the use of tentacles, net capturing, electric tethers, and harpoons [4] [13].

Along with space debris, failures in space are common and costly. From 1996-2000, 18% of payloads launched into space suffered a failure [16]. Rather than repairing and launching replacement satellites is costly and leads to unnecessary debris. In GEO, non-functional satellites occupy limited orbital slots. Astronauts have been tasked with repairs in the past, but this is a costly method and not feasible in higher orbits. Not advancing the field of OOS earlier has been a costly mistake made by the space sector [16].

On-orbit servicing can allow for assembly, maintenance, repair, deployment, release & retrieval, EVA support, inspections, and refueling of non-functional satellites [1]. This market is increasing and is projected to be a $4.5 billion USD industry by 2028 [17].

Space robots are applicable to space debris removal and on-orbit servicing.

1.3.2 6DOF Pose Estimation

As objects in space move in their orbital directions, non-functional debris tumbles. Tumbling takes place when the orientation of a spacecraft or an object is uncontrolled. Defunct satellites in GEO have been found to be tumbling at maximum rates of 2.60 °/sec, like the Intelsat-604 satellite [18]. To grasp and align with tumbling space debris or tumbling OOS targets, the target's 6DOF pose needs to be determined. The 6DOF pose of an object is both its position and orientation relative to a reference frame, which in robotics can be calculated in a camera frame or in the base frame. Determining the 6DOF pose is important in tracking objects, autonomous driving, augmented reality, and robotics [19]. Effective 6DOF object detection allows for better awareness of autonomous systems and avoiding collisions with obstacles, which can cause costly damage.

A technique for detecting the 6DOF pose of a target object is deep learning (DL). Using artificial intelligence (AI) allows for greater accuracy and eliminates the need for singling out key features, which is needed with traditional computer vision (CV) methods [20]. This is especially difficult in the space environment where lighting conditions can be extreme. Artificial neural networks (ANN) can make predictions with incomplete data as well, which is difficult with traditional CV techniques [21]. An AI based 6DOF pose detection algorithm can be applied to autonomous vehicles, autonomous unmanned aerial vehicles (UAV), and autonomous underwater vehicles.

ANNs are layered, inspired by the human brain. Networks consist of neurons connected to other neurons. During training, information is passed through the layers of neurons and the network is tasked with learning patterns between the input and output [21]. An example of an ANN's architecture can be seen in Figure 1.2.

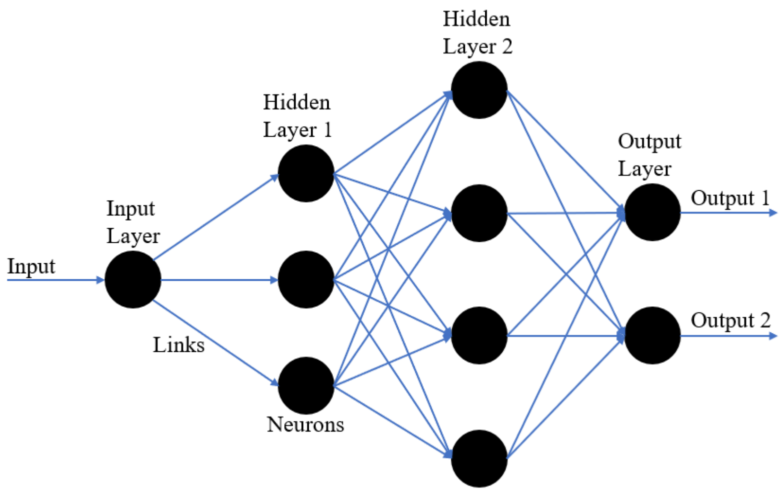


Figure 1.2: An example of the architecture of an artificial neural network. Included are the input layer, multiple hidden layers, and an output layer.

1.3.3 Limitations of Existing Methods

There are numerous methods being researched for active space debris removal such as net capturing, tethers, and harpoons [4]. Using humans is a method that has been used in the past for both space debris removal and on-orbit servicing as well. The Discovery Space Shuttle and its team was used to retrieve two non-functional satellites and an Endeavor Space Shuttle team was used for repairing the Hubble Space Telescope. This method has been proven to be costly compared to robots, and work on non-cooperative debris is difficult to complete with EVAs [22]. Electric tethers take up valuable space on satellites and cannot be used for the debris already in orbit as they must be on-board the satellite as a payload. A 30-50 kg tether payload needs to be on-board with a 5 km tether and a 10 m balloon to de-orbit a 500 kg satellite over multiple weeks [23]. Nets are difficult to control since the angle at which the net is pushed out drives the net capture distance [24]. Harpoons can lead to more debris as small fragments of targets can enter orbits [25]. Lasers are another technique researched, but they can be classified as weapons, making it ethically difficult to approve their use in space [26]. The use of robotics presents a cost-effective method for both space debris removal and on-orbit servicing. Combined with computer vision, robotic manipulators can match the angular rates of objects as well and quickly de-orbit them with a push towards the Earth's atmosphere.

1.4 Objectives of Research

The objectives of this research are to:

(i) Develop a method for detecting the 6DOF Pose of a target object,

(ii) Develop robot equations for kinematic control and visual servo of Fanuc M-20iD/25 Robot, and

(iii) Implement the method for detecting the 6DOF Pose of a target object on a microcomputer.

1.5 Methodology of Approach

The methodology to meet the set objectives in Chapter 1.4 can be seen in Figure 1.3. Many mature architectures exist for object localization with the help of AI, all capable of detecting occluded objects, or multiple classes in a single frame. The second version of the You Only Look Once (YOLO) architecture is effective for real time tracking and is used along with a depth sensor, the RealSense d435i depth camera, to localize an object in the pixel frame and to derive the 3D position in the camera space. Using a new CNN developed based off the framework presented in [27], the 3D orientation of the target is predicted. To maximize computational efficiency and to mitigate gimbal locking, quaternions are used for orientation estimation. Concurrently, the robot equations and virtual simulations are developed for the Fanuc M-20iD/25 industrial robot. Kinematic controllers are developed for efficient joint control to meet range and velocity limits. Both the computer vision and robot control algorithms are implemented together for the virtual simulation of robotic docking. Due to Covid-19, all methods were completed virtually as lab access was restricted. In Table 1.1, the hardware used is listed.

Table 1.1: The various hardware devices used in this thesis work and their specifications.

DEVICE	SPECIFICATIONS
Computer: used for training CNN's and the robot simulation	• Intel Core i5-9400F CPU • 32 GB DDR4 memory • NVIDIA GeForce GTX 1660 (1408 CUDA cores, 6 GB) • Windows 10 and Ubuntu 18.04
Camera: used for depth sensor and intrinsic matrix	• RealSense D435
Microcomputer: used for deploying CV algorithm	• NVIDIA Jetson Nano • NVIDIA Maxwell GPU • Quad-Core ARM Cortex-A57 processor • 4 GB memory

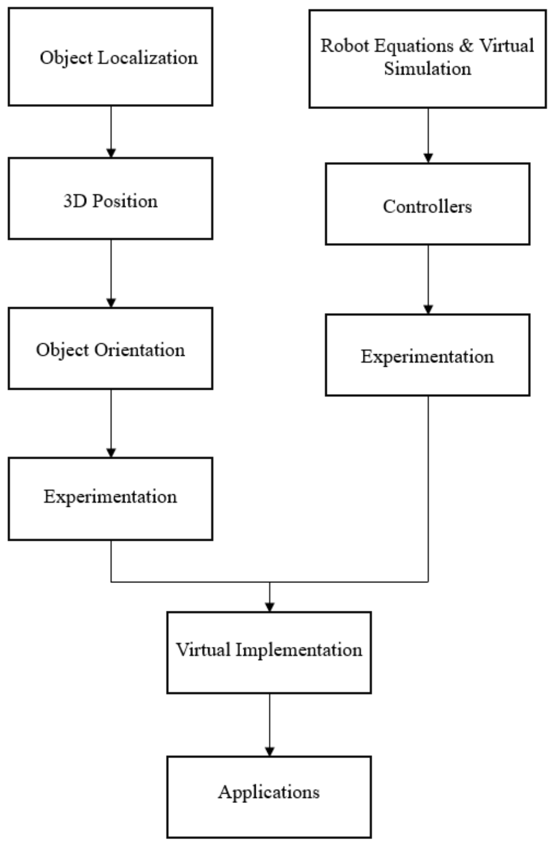


Figure 1.3: Methodology of the parallel development of the computer vision algorithm and robot simulation for a virtual implementation.

1.6 Layout of Thesis Document

This document consists of seven chapters. Following Chapter 1's introduction, Chapter 2 provides a comprehensive literature review of relevant work in the following four areas: (1) existing computer vision approaches, (2) object localization, (3) object pose estimation, and (4) visual servo control. Chapter 3 gives a detailed description of the development of a 6DOF pose detection algorithm. Chapter 4 showcases the deployment of the 6DOF pose detection algorithm on a space grade board. In Chapter 5, an outline of creating virtual training data for training the 6DOF pose detection network and other networks is presented.

Chapter 6 is devoted to the derivation of robot equations of the Fanuc M-20iD/25 industrial robot and virtually simulating its control. Finally, Chapter 7 concludes the work, and outlines methods for future work.

Chapter 2: Literature Review

Summary: This literature review consists of two main topics. The first topic covers computer vision in practice for both object localization and object pose estimation. The second topic covers space debris mitigation techniques and space robotics.

2.1 Computer Vision

The work here is motivated by the need for an accurate computer vision algorithm for 6DOF pose detection, accurate path planning, and to effectively drive the end-effector of a robotic manipulator towards an object for docking. This problem is divided into two parts: object detection and object orientation estimation.

2.1.1 Object Detection

Object detection is a subsection of computer vision, which, since its origination in 1960 at MIT, has aimed to assist in developing autonomous systems to complete tasks like human vision [28]. Object detection can be broken down further into multiple subsections such as traditional CV and deep learning. Traditional CV techniques are the oldest techniques and are considered feature descriptors. Some examples of feature descriptors for feature extraction in images are the SIFT, SURF, BRIEF, and FAST methods [29] [30].

The SIFT method learns representations of images that are scale invariant, meaning features do not change if the size of the image or its features change. This standardizes all images in the process [29] [31]. The SURF method is faster than SIFT as it uses box blurring to extract features from an image. Box blurring extracts the average of all image values in a rectangular patch [29] [32]. The BRIEF method is faster than typical feature description methods as it selects a patch of pixels and computes a pixel intensity metric to represent the image [29]. The FAST method is a high-speed corner detector using sixteen pixels around a potential corner in an image. If the pixels around the candidate are brighter than the candidate, it is labeled as a corner [30].

Traditional CV has advantages over deep learning techniques. Feature extractors are efficient and can be programmed in less code, making it easier to deploy algorithms on low-cost microcontrollers. Programmers have full transparency of the mechanics of traditional CV methods, unlike deep learning techniques which are black boxes; this allows for parameters to be tweaked to improve performance [20].

Deep learning methods have emerged with the growth of deep learning in general. Neural networks allow for greater accuracy and more flexibility with training as the programmer can highlight features needed for detection in training sets [20]. Mature network architectures are readily available such as R-CNN, Fast R-CNN, Faster R-CNN, SSD, YOLO, and RefineNet; and have been tested using large, standardized datasets such as VOC 2007 and VOC 2010.

Girshick et al introduced R-CNN for accurate object detection. R-CNN uses region proposals in which an object’s predicted location is proposed, a large CNN for feature extraction, and class specific support vector machines for each region proposal. This network achieved 53.7 mean average precision (mAP) on the VOC 2010 test dataset. This network is not optimal for tracking though as it is slow [33]. Girshick et al also introduced Fast R-CNN, a network which builds on R-CNN. This network allows for 9 times faster training than R-CNN and 213 times faster detection, as calculated during testing. This network achieved 68.8 mAP on the VOC 2010 test dataset [34]. To build on Fast R-CNN further, Ren et al developed a Region Proposal Network (RPN) to use with Fast R-CNN. This allows for convolutional features to be shared between an RPN and the Fast R-CNN network. The RPN takes an image as the input and outputs object scores for the region proposal. This network allows for up to 5 frames per second (fps) of detection [35].

Zhang et al developed the RefineNet network which overcomes the inefficiencies of Faster R-CNN and provides high efficiency. This network processes images at 40 fps for images of size 320 x 320 and scored 80.0 mAP on both the VOC 2007 and VOC 2012 datasets [36]. SSD is a network introduced by Liu et al which outperforms Faster R-CNN and scored 74.3 mAP on the VOC 2007 dataset [37]. The YOLO network developed by Redmon et al can detect up to 45 fps with the regular network and up to 155 fps on a smaller network called Fast YOLO. This network learns general representations of objects and only looks at an input image once. YOLO makes half the number of background errors compared to Fast R-CNN and outperforms R-CNN. YOLO also scores more than twice the mAP of other real-time tracking networks [38]. There have been multiple iterations of the YOLO network, with each improving on the previous [39] [40].

Other popular networks include SqueezeNet, used for autonomous driving; and MobileNet, used for mobile applications [41].

Xiang et al developed PoseCNN, a network that uses convolutional layers for feature extraction and embedding to detect and label an object [42]. ConvPoseCNN, developed by Capellan et al, builds on PoseCNN, but also does semantic detection of objects [43]. Both these networks use custom layers for object detection and are not readily available networks.

2.1.2 Object Orientation Estimation

To accurately understand where a target object is and its trajectory, its position and attitude are needed relative to a camera, or in the case of OOS or debris capture, the servicing satellite. Targets are often uncooperative, especially if the target is a piece of debris, therefore orientation estimation needs to be done on-board without humans-in-the-loop. In these tasks, monocular cameras are preferred due to their simplicity and low power requirements, especially for smaller satellites [44]. Many methods exist for pose estimation, both traditional and deep learning based, each with their advantages. Traditional methods require less code and can be efficiently deployed on low-cost microcontrollers. Deep learning methods provide greater accuracy but require large datasets for training [20] [44].

Chen et al developed a method for pose estimation using deep learning and geometric methods. This method uses a single image to estimate landmark coordinates and relating them back to 3D points on a known 3D object model. To solve for the object's pose, non-linear optimization is used. Training images are used to create a 3D model of the target satellite to train a deep network which predicts the landmarks [45]. As mentioned, deep learning methods have disadvantages and traditional methods can be more efficient. Liu et al introduced a method in [46] which uses lasers to determine the pose of a non-cooperative target. Three lasers project lines onto a satellite creating various points along the adapter ring. Using these points, a solution for real-time pose estimation can be developed. This method is accurate near the target. First the breakpoints are calculated using edge detection to determine the 3D coordinates of each point. The six points are then used to determine the normal vector of the surface by using the least squares method. In [47], the development of point clouds of an object using triangulation of 2D image features is used to identify and track targets. The generated point cloud is compared to a reference model point cloud, similar to using a 3D model of a target. Matching different feature points, the pose can be found. For tracking of the object and to reduce noise, the estimated pose values are used in an unscented Kalman Filter over time.

There are also several different techniques that can be adopted for space-robotics to estimate spacecraft pose, both traditional and deep learning based. In [48], 2D images are used with CAD files to estimate the 3D coordinates of an object. There is an offline system that stores a text file of poses and Z-coordinates of the object model in a data base. An online system acquires images using a 2D camera in MATLAB. The online system compares the image with the poses and Z-coordinates in the database to find a match. This method has been adapted for robotic handling. Casado et al developed a detection system using 2D pattern recognition to determine the pose of an object. This approach uses two cameras, one RGB and the other NIR, both used in different applications. During detection, the orientation angle is estimated of the object using the scale and position of the image. Linear equations can then be used to determine the pose in the X and Y directions. Unfortunately, this method has high computational costs, even with low resolution images [49]. In [50], Barrois and Wohler introduce a method for spatial-temporal pose estimation of objects. For this method, stereo images are needed to use 3D depth points. A 3D model is created from the stereo images and converted into a 3D point cloud to infer the translation and rotation of the object relative to the stereo images. In [51], it is shown that a Clustered Viewpoint Feature Histogram (CVFH) with a Kinect camera's roll histogram can effectively determine the pose of objects in real environments by relating images back to CAD models of targets. A CVFH represents point clusters for object recognition.

3D CAD models are also used in [52] where CAD models and real images aligned together for different poses can allow a model to be used to estimate the pose in real images. The model introduced here is the FPM, which is a fine pose parts-based model. This model uses geometric information of 3D parts and appearance information relating to the objectness of the object for accurate pose determination.

Deep learning methods make use of CAD models as well. In [53], analysis-by-synthesis is used with RGB-D images. The input image is compared with a rendered image of the target at a specific pose. Due to occlusions, a traditional comparison like the one in [48] can be inaccurate, therefore a CNN is introduced that completes the comparison. The CNN takes a combination of inputs that include the rendered objects coordinates, rendered depth from a 3D model, the observed object coordinates, and observed object depth. The output is an energy function which is minimized. In [54], RGB images are matched to CAD models using a Quadruplet CNN. One of the four branches takes the real image as its input, another takes the same view but of a rendered version of the image. The last two branches take different rendered views of the object. This allows for different feature sets to be extracted and combined for accurate estimation. Since multiple views and branches are needed, this method is computationally expensive. To mitigate some computational cost, this method does not require expensive 3D pose annotations for training.

Dwibedi et al use 2D bounding boxes to create 3D bounding boxes. Initially, a 2D image is inputted to localize the object of interest with a 2D bounding box. From there, the corners of the object are localized, producing a 3D interpretation of the object. An RPN is used to localize the object. The localized object is inputted back into the network and fully connected layers are used to predict the location of the corners [55]. This method is well-defined for cuboid shaped targets. The method introduced in [56] is similar to the one in [55], as segmentation is used to detect the object of interest in 2D. A CNN is used to predict the 2D projections of the corners of a projected 3D bounding box around the target. This method works for objects of any size but is limited to a specific range of poses used in training to be effective for symmetric objects.

Crivellaro et al have developed a method in [57] which predicts the 3D pose using grayscale images. The objective is to detect control points on the object, similar to landmarking. The control points, such as corners of an object, are then projected in 2D to create a 3D representation of the object. This method is useful for occluded objects as only specific parts of an object are needed for detection, not the whole object. This method is computationally expensive as a CNN is used for control point detection, needing one CNN for each 32 x 32 patch of the input image. The detected parts are then inputted into a second CNN which takes 64 x 64 patches of the image centered around the control points to predict 2D projections. The approach introduced by Grabner et al in [58] uses an approach similar to control points. An RGB image is used to detect an object and a CNN is used to predict the location of the 2D projections of a 3D bounding box around the object. A PnP algorithm is used after finding the projections to determine the pose of the object.

Do et al proposed a method that extends the Mask R-CNN object detector to include a pose regression branch. By decoupling translational detection from rotational detection using Lie Algebra, quick pose detection can be attained. The network is trained with 480 x 640 sized images and 6DOF labels. Using the Lie Algebra output, Rodrigues mapping is used to create a rotation matrix. To obtain the translation of the object, the predicted Z component is used with 3D to 2D projection to find the X and Y location of the object [59].

Guo et al developed a method using RGB-D images to estimate the camera pose relative to a target. A CNN called PoseNet is used to determine a camera's pose parameters, while an LSTM block is used for adding temporal information for improved accuracy [60]. Melekhov et al developed a method for determining the pose between two images of the same object using CNNs. RGB images are used for training the network and is similar to using CAD models, but rather than comparing an image to a CAD model, the network compares it to a pre-defined image with known pose of the target [61].

Liu et al introduced a technique in [62] which uses a cropped image within the 2D bounding box detection as an input into Q-net, a network which regresses a unit quaternion rotation of the object relative to the camera. The quaternion is used along with the translation of the object to create a 3D bounding box around the object for visualization. Langlois et al introduce a network which uses 2D images to predict the quaternion of an object relative to the camera as well. Their network is trained by using virtually rendered images of an object with transparent backgrounds. During training, the transparent background is replaced with different environments on the fly. The network itself is small with only eight layers including the input and output. This network is trained on 64 x 64 grayscale images. This network is simple, yet effective [27]. The PoseCNN network uses quaternion regression as well for predicting object rotation. The orientation branch of the network uses bounding boxes encompassing the object in an input image after the image goes through the feature extraction network. Pooling layers are used to extract quaternions for each object of interest in the image [42]. The ConvPoseCNN network builds on the PoseCNN network by using semantic labels to filter out objects, rather than bounding boxes. The orientation prediction branch uses convolutional layers on the semantic segmentations to predict the orientation quaternion of each object of interest in the image [43]. Mahendran et al developed a network that regresses 3D pose with a small CNN, but rather than quaternions as the output, an axis-angle representation is given. The network has ten layers including the input and output. To train this network, data augmentation is carried out by changing the camera tilt and azimuth of images to create larger training sets. Real RGB images are used for training [63].

Pose regression networks provide quick predictions of quaternion rotations with small networks which are computationally efficient, which is an asset when deploying software on a satellite's computer.

2.2 Debris Removal and Space Robotics

Space debris is a growing issue. As more satellites are launched, the probability of collisions increases. There are methods being researched for space debris removal which include using tentacles, net capturing, tethers, harpoons, and robotic arms [13].

One application of a tentacle gripper is the TAKO-Flyer, a concept developed by Kazuya et al. This technology is a robotic device made up of numerous fingers working like octopus tentacles. Using fewer actuators than traditional robotic joints would, it in theory is able to grasp arbitrary shapes with tumbling motion by wrapping itself around the target. The mission concept consists of rendezvous and fly around, separation and approach, grasping and stabilization, and robotic capture or deorbiting [64]. The OctArm Continuum Manipulator is another example of a tentacle used for grasping targets. The goal of using a continuum manipulator is to replace the serial chain of rigid links found on robotic arms with continuous and flexible links. The OctArm was able to grasp and manipulate objects of varying sizes and shapes in testing. The downside of this concept was its use of air to actuate the joints in the fingers, which is not feasible in the space environment [65].

Nets have been studied for deorbiting space debris or to park non-functional spacecraft into graveyard orbits. The ROGER Spacecraft created by ASTRIUM is a proposed net to transport targets into graveyard orbits. The mission concept uses 20 nets to capture a target by closing them around the target, then pulling the target into a graveyard orbit and severing ties to the net to leave the target satellite. Each net is proposed at 9 kg, but this concept leaves nets in the graveyard orbit with the target, creating more debris [66]. Lavagna et al proposed the D-CoNe project in [67], where a cone shaped net is designed for medium sized debris capture. The net is shot from a satellite, grabbing debris, and moving it to a disposal position. This project has proved the feasibility of using nets for object capture, but further testing in a microgravity environment is needed. The Junk Hunter mission concept consisted of using a deployable, inflatable boom with a mesh netting capable of grabbing debris, targeting large debris in the 800 – 900 km altitude range with inclinations of 82 – 83˚, where a large population of debris already exists. The project was proven to be feasible for operation in orbit [68].

York University launched its own mission to test the capabilities of an electrodynamic tether for deorbiting, called DESCENT. Electric current runs through the electrodynamic tether, creating a Lorentz force when the current interacts with the Earth's magnetic field. The Lorentz force can be increased or decreased to increase the altitude or to decrease the altitude of the spacecraft. The DESCENT spacecraft was launched into space in October 2020 and entered its orbit the following month [69].

Harpoons have been tested at ASTRIUM Stevenage for penetrating ability, resistive strength, and how much extra debris is created when latching on to a satellite. Tests have shown this method is feasible [70].

Robotic arms provide the most promise due to their ability to adapt to OOS missions along with space debris capture and due to their maturity in terrestrial applications. One proposed mission for robotic arm de-orbiting is the DEOS mission from the DLR. This mission consists of a 7DOF robot on a servicing satellite tasked with capturing a tumbling non-cooperative satellite or object, servicing it, and de-orbiting it from LEO. Since it is in LEO, it is possible for a direct radio link with the servicing satellite during operation, therefore the system is controlled semi-autonomously and from Earth [71].

There have been space robotics missions operating in orbit to test robotic capabilities. The ROTEX mission from DLR tested space automation by verifying joint control in microgravity, evaluating 6-DOF hand controllers, and verifying a man-machine interface. The robotic arm had 6 joints and was teleoperated with 5 – 7 seconds of lag. Image processing was done on ground to ensure less computation was taking place on-board the spacecraft [72].

ETS-VII was tested next by the DLR to verify autonomous rendezvous, docking, and robotics technology in space. A servicing satellite and a target satellite was launched with a 6-DOF arm on the servicer. This was also teleoperated from ground controllers with a 7 second delay [72].

The ROKVISS program demonstrated the feasibility of high performance of intelligent robotic systems in space for OOS by the DLR. This was also teleoperated but with less communication lag than the ROTES and ETS-VII satellites [72].

Canada has also developed robotic arms for use in space. Canadarm was developed by Canadian companies and installed on-board the Space Shuttles. This arm was controlled both manually and could be programmed to complete tasks autonomously. Canadarm completed OOS tasks such as sending satellites into orbit, capturing satellites for repairs, and assembling the International Space Station. Canadarm was also used for assisting astronauts in EVAs [73].

Canadarm2 is a robotic manipulator system on-board the International Space Station teleoperated by ground control teams at the CSA or NASA. With multiple arms having 7 joints each, this manipulator is used for installing or replacing small equipment on the International Space Station, replacing defective components, and testing new tools or robotic techniques [74].

An issue with space robotic missions when servicing in further orbits is increased communication lag and the need for autonomy. Space is an expensive environment to test in, therefore testing for autonomous systems needs to be completed on the ground in dynamically simulated systems. DLR's EPOS facility tests high risk rendezvous and docking with a hardware-in-the-loop system. Two 6-DOF industrial robots are used, one equipped with a mock satellite on a 25-meter rail system simulating close range approach. The second robot is equipped with a vision based gripping system. The objective of this system is test autonomy for orbital life extensions for geostationary satellites [75].

The WMS LEMUR facility is dedicated to the testing of capturing tumbling satellites autonomously. A 7-DOF robotic arm is used for testing short range rendezvous, formation flying, and active space debris removal missions. Contact forces are measured between the gripper and mock target [76]. Testbeds can also be used for testing the autonomy of potential space manipulators by using air bearing tables to simulate microgravity on a manipulator [77].

2.3 Conclusion

In conclusion, although traditional CV methods are mature and efficient, they do not provide effectiveness in detecting objects moving through a changing environment like space. Neural network-based object detectors are well researched, and some architectures have been developed with the capability of real-time object tracking, which is needed when objects are moving at orbital velocities in space. Traditional CV methods are not quite as effective at 3D orientation and therefore a neural network is needed to accurately predict an object’s 3D orientation. Combining two neural networks provides the ability to quickly predict the 6DOF pose of an object at various orientations and in changing lighting conditions with occlusions and noise.

Neural networks have been paired already with industrial robots for grasping and placing objects in factory settings, and the maturation of this field will allow for adaptation for the space environment, rather than having to advance technologies such as nets, harpoons, or tentacles. These technologies are being researched but provide their own challenges. Whereas robotics is well advanced on Earth and provide a simpler transfer to the space environment. Combining 6DOF pose detection with robotics will allow for visual servo control to rendezvous and dock with a target for active space debris removal or OOS.

Chapter 3: 6DOF Pose Detection

Summary: In this chapter, a detailed description and derivation is given of a deep learning based 6DOF pose detection network. First, by using an object detection network, the 3D position of an object is determined by using a depth camera. Once localized, the image of the object is inputted into a second convolutional neural network to determine its quaternion orientation relative to the camera. Although quaternions are predicted, the Euler angle rotation can be calculated. The two networks are linked together to create a pipeline for 6DOF pose detection.

3.1 Introduction

The current thesis is motivated by the need for an effective computer vision algorithm capable of detecting the 6DOF pose of an object of interest in varying environments. The algorithm needs to be fast for real-time tracking, especially in the space environment where objects are traveling at orbital velocities. To address the limitations of current computer vision algorithms used in space applications, a 6DOF pose detection algorithm using deep learning is developed. An object detection network is trained using the YOLOv2 structure for real time object tracking applications and a pose regression network is trained for fast quaternion pose regression of an object.

3.2 Object Detection

For object detection, the YOLOv2 network architecture was used. This network was chosen due to its fast object detection capabilities of up to 45 fps, which is optimal for object tracking [38]. An improvement of YOLOv2 over the original YOLO network is the use of anchor boxes. Anchor boxes are filters of predetermined size tiled across an input image when the network is making predictions as seen in Figure 3.1. Rather than predicting the location of objects by using a sliding window across the input image, anchor boxes allow the network to predict across the whole image at once [78].

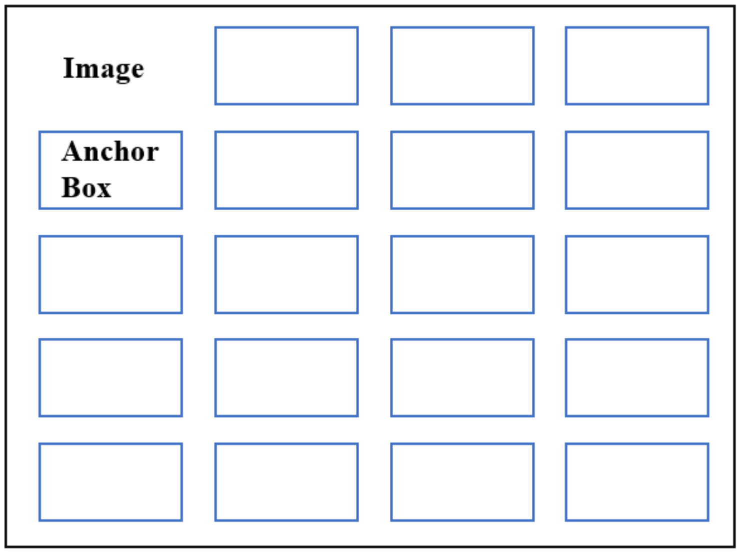


Figure 3.1: An example of anchor boxes tiled across an image.

Another advantage of using the YOLOv2 network is the implementation of batch normalization on all convolutional layers [38]. Batch normalization regularizes a network by equalizing the data batches which are being inputted into the layers of the network. Seen in Figure 3.2 is the network architecture of the YOLOv2 network.

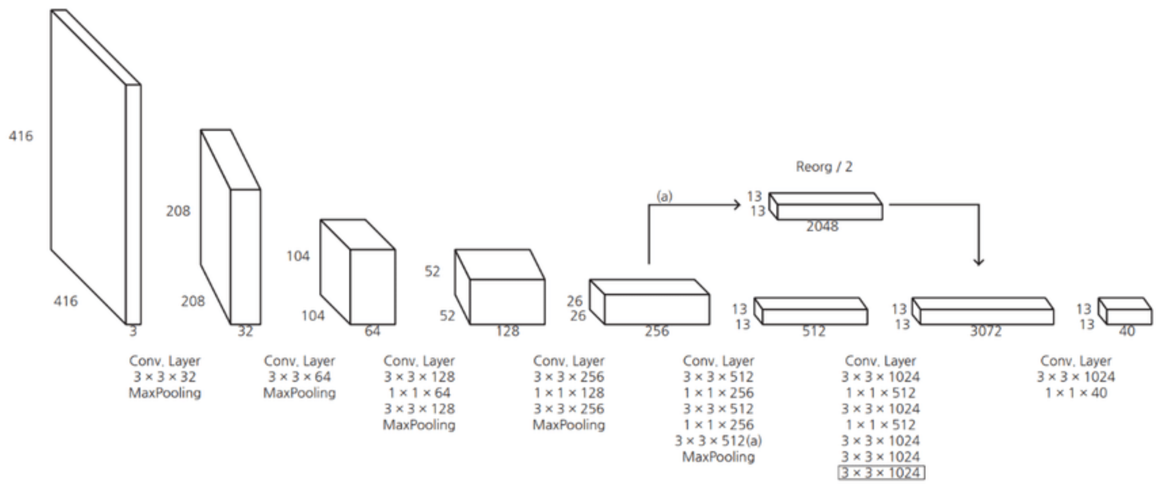


Figure 3.2: The YOLOv2 network architecture.

A general neural network is designed to develop a relationship between the input data and the expected output by using features found in the input data. As seen in Figure 3.2, the YOLOv2 network consists of convolutional and pooling layers. Convolutional layers are used to create convolutional kernels capable of scanning the input data for features. Pooling layers down sample the feature maps outputted from convolutional layers to summarize the features. The YOLOv2 network is closed off with fully connected layers which are present for the purpose of connecting all neurons from one layer to the next.

To create the object detection model, MATLAB was used. The network size is 188 layers from input to output with a LeakyRelu52 feature layer and darknet53 feature extraction layer. The Leaky ReLu function is an activation function used in neural networks, helping networks create relationships between the input and output in complex data. A leaky ReLu function builds on the ReLu function, but rather than having a slope of 0 for negative x-values, it has a scaled slope. The equation for a leaky ReLu function is seen in Equation

3.1. The darknet53 network was chosen since it is a CNN commonly used as a backbone for object detection networks like YOLO.

$$f(x) = \begin{cases} x, x \geq 0 \\ scale * x, x < 0 \end{cases} \tag{3.1}$$

The object being detected is a funnel due to it being similar to a satellite thruster nozzle. The comparison can be seen in Figure 3.3. The network was trained on virtually rendered images with ground truth labels consisting of the x and y position of the top left corner of a bounding box around the target, and the width and height of the bounding box.

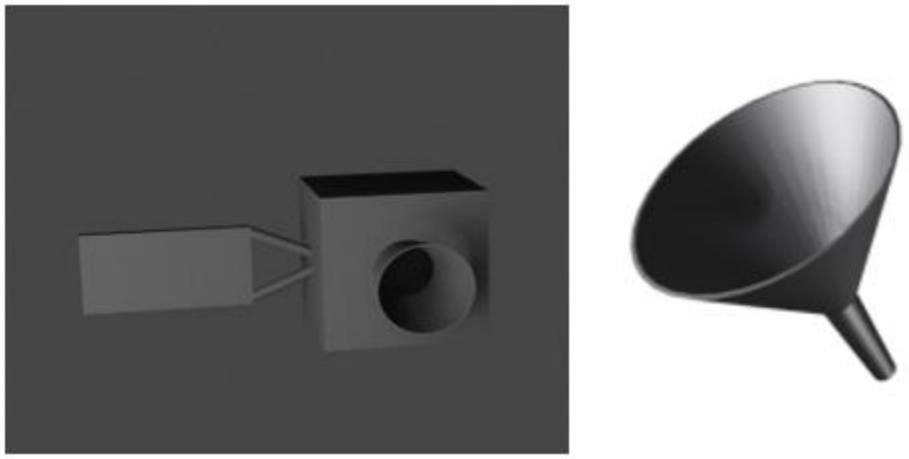


Figure 3.3: A funnel was used as the target due to its similarity to a thruster nozzle.

The object detection network was trained on an NVIDIA GTX 1660 6 GB GPU for images of size 224 x 224. The network was trained through 100 epochs with a mini-batch size of 4. The initial learning size was set to 0.001 and the sgdm solver was used. The number of epochs in machine learning is the number of times the training data will be passed through during the training process. The mini-batch size is the size of the batch of data to be used for each training set. After the amount of training data specified in the mini-batch size is passed through by the network, the network updates its weight. The initial learning rate is a tuning parameter used to calculate the step size while training to minimize the loss function. The sgdm solver is a stochastic gradient descent solver.

The network was trained on 648 virtually generated images of a funnel which were augmented to generate more training data. Augmenting data generates modified versions of pre-existing training data to lower overfitting during the training process, generating more training data in the process [79]. Overfitting occurs when a model creates a relationship with noise present in training data, making it difficult to perform well with new inputs [80]. Figure 3.4 shows augmented versions of an example training image. The top left image in Figure 3.4 is the original training image, the remaining three are augmented images.

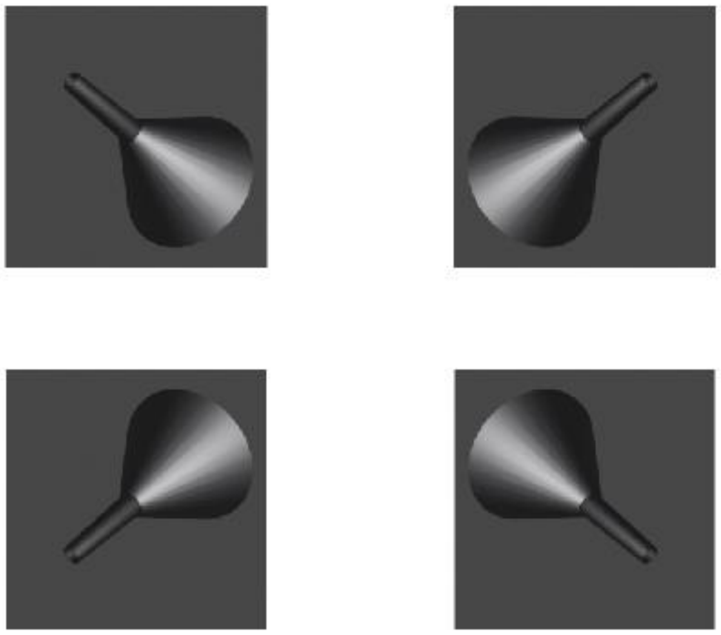


Figure 3.4: Augmentation of training data to increase the size of the training data set.

When an object is detected, the predicted output is the x and y pixel coordinates of the bounding box's top left corner, the width and height of the bounding box, and the confidence score of detection. The object class can also be outputted which is beneficial when detecting multiple classes in one frame or for image classification. The x and y pixel coordinates are in the pixel frame, a 2D frame which can be visualized in Figure 3.5. The x_{pixel} coordinates increase from left to right in an image, the y_{pixel} coordinates increase from the top to the bottom of an image. The confidence score is the probability the object of interest is inside the bounding box as determined by the neural network.

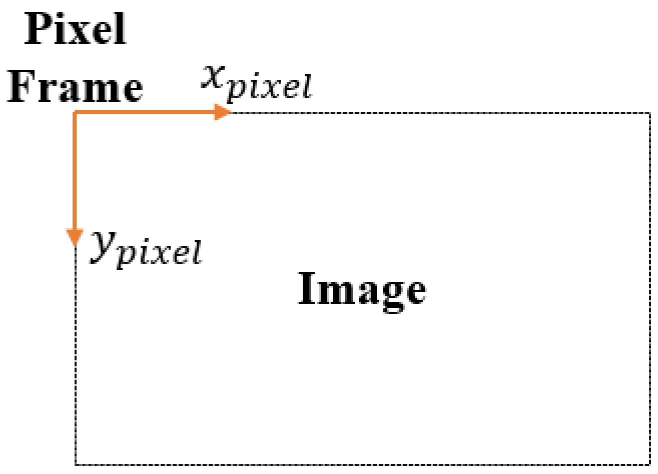


Figure 3.5: The origin and orientation of a pixel frame relative to an image.

To get the 3D position of the object of interest, the object's depth information, and a transformation from the pixel coordinate frame to the camera world frame is needed. To determine the depth of the object, a depth camera was used. Both RGB and depth images were used in unison to complete the transformation. A depth image is unique, as each pixel value contains the distance between the depth sensor to a surface, rather than pixel intensities. Figure 3.6 shows an example of a depth image. Equations 3.2 and 3.3 use the bounding box parameters after detecting the object of interest's location to find the coordinates at which to detect the object's distance to the camera sensor. The top left corner of the bounding box typically does not contain the object; therefore, the center location of the bounding box is needed. This is deemed to be the geometric center of the object.

$$x_{depth} = x_{pixel} + \frac{width}{2} \tag{3.2}$$

$$y_{depth} = y_{pixel} + \frac{height}{2} \tag{3.3}$$

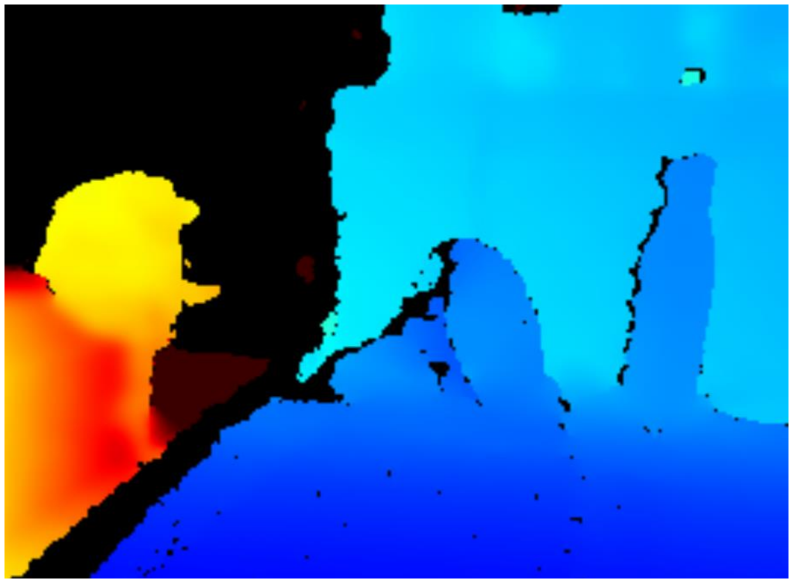


Figure 3.6: A depth frame with the funnel shown inside the red ellipse.

The transformation from the pixel frame to the camera world frame can be seen in Equation 3.4 [46]. Figure 3.7 shows the camera world frame relative to the camera sensor.

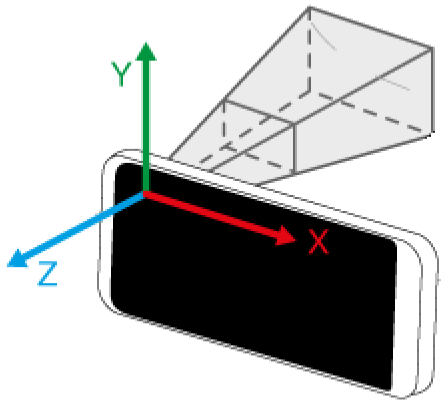


Figure 3.7: Camera world frame origin and orientation relative to a camera sensor.

$$\begin{Bmatrix} x_{pixel} \\ y_{pixel} \\ 1 \end{Bmatrix} = \frac{1}{Z}\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} \tag{3.4}$$

After re-arranging the transformation above, Equations 3.5, 3.6, 3.7 are used to calculate the X and Y position of the object in the camera world frame.

$$X = \frac{(x_{pixel} - c_x)}{f_x} Z \tag{3.5}$$

$$Y = \frac{(y_{pixel} - c_y)}{f_y} Z \tag{3.6}$$

$$Z = Z \tag{3.7}$$

The values f_x and f_y are the focal length of the camera in the x and y direction, measured in pixels. The values c_x and c_y are the principal points of the projection, measured in pixels. This is the intersecting point of the optical axis and the image plane, where the optical axis is normal to the camera sensor towards the image plane. To obtain the intrinsic parameters of the RealSense d435i, calibration and measurements were not required. Intel provides a software wrapper for controlling RealSense cameras in Matlab, allowing intrinsic parameters to be extracted. The following are the intrinsic values for a 224 x 224 image.

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 160.5988 & 0 & 113.9144 \\ 0 & 285.4730 & 119.4302 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

To test the feasibility of Equations 3.5 to 3.7, ROS Gazebo Simulator was used. In the simulation, a virtual Kinect camera was used to determine the $(X, Y, Z)_{\text{world}}$ coordinates of a target using Equation 3.9 by detecting the object in front of it. The camera was moved to the calculated $(X, Y, Z)_{\text{world}}$ coordinates to prove Equations 3.5 to 3.7 are accurate for a transformation between the image frame to the simulated camera world frame. The following is the intrinsic matrix of the virtual Kinect camera used in the simulation. Figure 3.8 shows the initial simulation setup and then the camera meeting the target.

$$\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}_{Kinect} = \begin{bmatrix} 554.2547 & 0 & 320.5 \\ 0 & 554.2547 & 240.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.9}$$

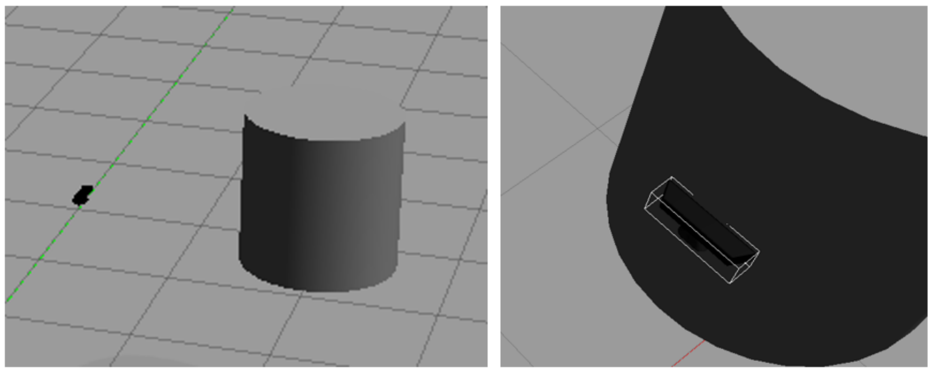


Figure 3.8: The initial simulation setup with the Kinect camera and a target, and then the camera docking with the target.

Figure 3.9 shows examples of object detection with the bounding boxes around the target and respective confidence scores. Figure 3.10 shows the detection of a physical funnel and the calculated $(X, Y, Z)_{\text{world}}$ coordinates.

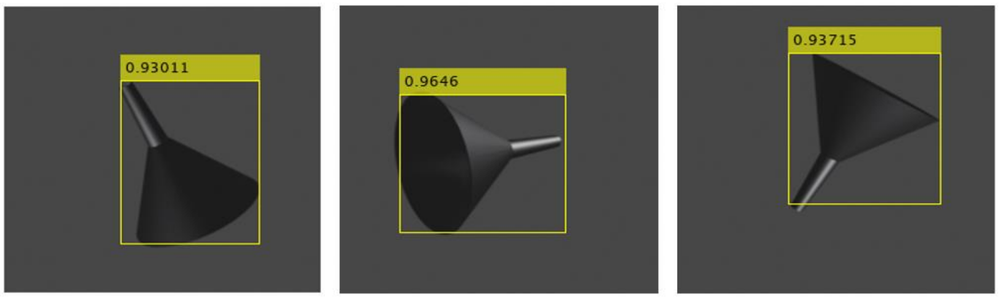
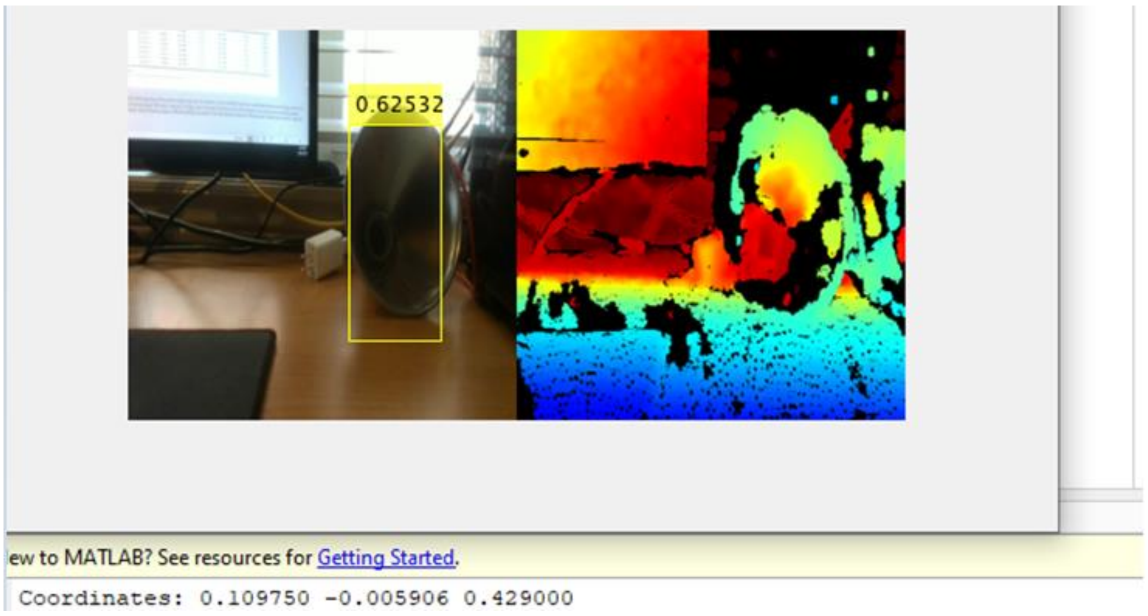


Figure 3.9: Examples of object detection from the YOLOv2 network.



```
Coordinates: 0.109750 -0.005906 0.429000
```

Figure 3.10: Detecting a physical funnel and retrieving the 3D coordinates of the funnel.

3.3 Pose Detection

For regressing the 3D pose of an object, the regression network seen in Figure 3.11 was developed based on the network used by Langlois et al [27].

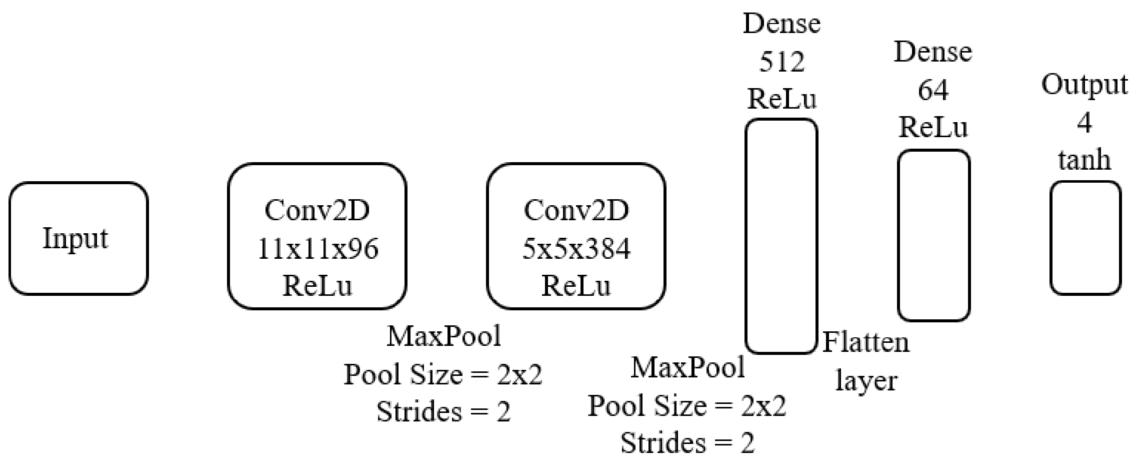


Figure 3.11: The custom pose regression network architecture.

The input layer of this network takes an input shape of 224 x 224 x 3 leading in to the first 2D convolutional layer. The first convolutional layer uses a ReLu activation function and has 96 kernels of size 11 x 11 to look for features throughout the data. The features from the kernels are pooled with a max pooling layer of size 2 x 2. The second convolutional layer uses a ReLu activation layer as well, but with 384 kernels of size 5 x 5. An identical max pooling layer is used to pool all data. The first two fully connected layers use ReLu activations function with 512 and 64 neurons, respectively. A flattening layer is used to convert data from the first fully connected layer into a 1D array. The output layer is a fully connected layer of size 4 and uses a tanh activation function. The four outputs each represent the four parameters found in a quaternion. A tanh activation function, the equation of which can be seen in Equation 3.10, constrains all four output values from -1 to 1, forcing the output quaternion to be a unit quaternion. The network has a relatively small architecture meant for quick pose regression.

$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \qquad (3.10)$$

To train the network, Python was used along with the TensorFlow and Keras libraries. Images used to train the object detection network were used to train the pose network, but with quaternion ground truth labels and transparent backgrounds. Initially, each training image was rendered with Euler angles. The Euler Angles were then transformed to quaternions using an XYZ transformation. The network was trained over 100 epochs with a learning rate of 0.001 with the Adam optimizer. The Adam optimizer combines the best traits of other optimizers that use root mean square methods to improve their ability to work with noisy data. The default parameters of the Adam optimizer work well enough to train networks effectively [81]. Figures 3.12 and 3.13 highlight the improvements in model accuracy and model loss of the network throughout training.

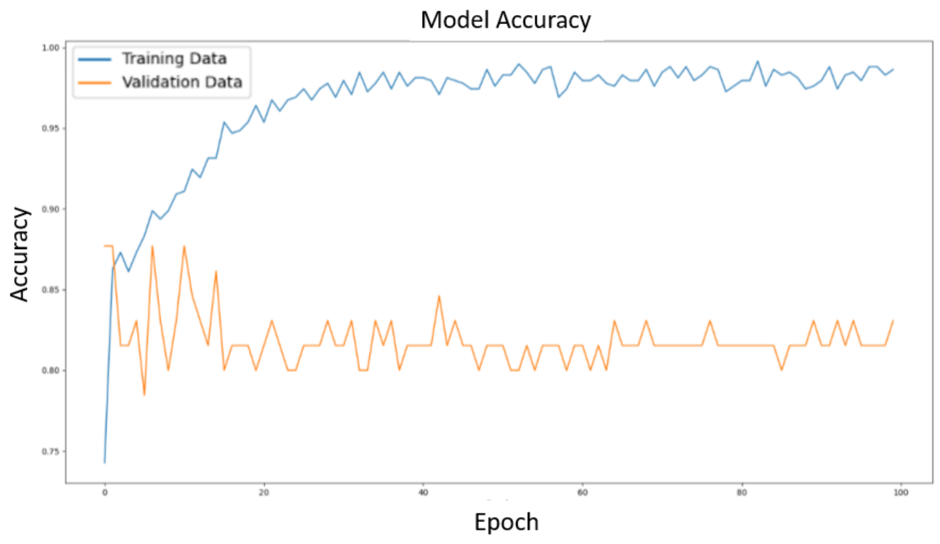


Figure 3.12: Model accuracy vs. training epoch of the training and validation data.

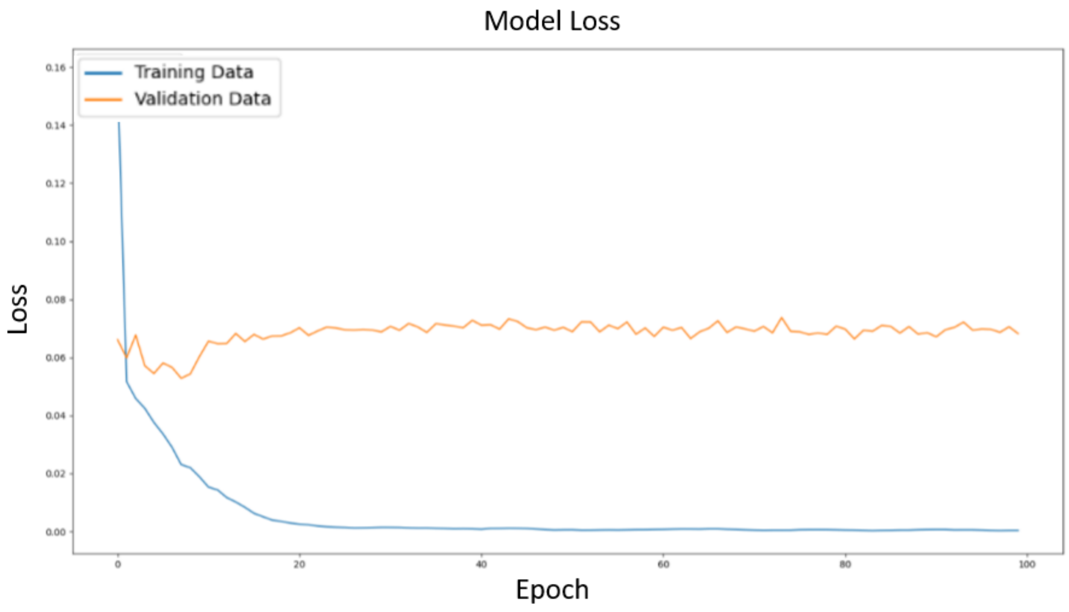


Figure 3.13: Model loss vs. training epoch of the training and validation data.

The same computer was used to train the pose network as the object detection network. The funnel was once again the object of interest due to its similarity to satellite thrusters. All training images used had transparent backgrounds to eliminate unnecessary noise from the image data. Figure 3.14 is an example of a training image with its quaternion ground truth listed. The format for quaternions used was [w, x, y, z], where w is the scalar number representing the rotation about the axis described by $x\hat{\imath} + y\hat{\jmath} + z\hat{k}$.

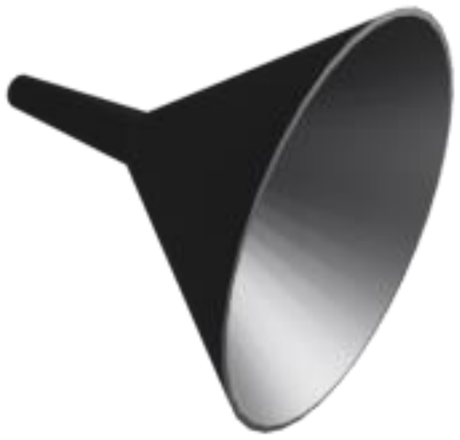


Figure 3.14: The quaternion orientation of the funnel is [-0.541, -0.159, -0.249, 0.787].

Quaternions can be used to represent orientation in a 3D space and are often used in the control of aircrafts or rockets since they can represent orientation in one rotation, rather than three with Euler angles. This saves time and storage [82]. They are also used in 3D graphics and recently, computer vision applications [83].

Orientation can be represented using rotation matrices, Euler angles, and quaternions. Rotation matrices are disadvantageous due to computational inefficiency since 9 parameters need to be regressed. For pure rotations, back-propagation is needed, and orthogonality needs to be enforced on the matrix outputs [84]. Back-propagation is a technique used in training neural networks where the previous epoch's loss value is fed back into the model being trained to adjust model weights [85]. Rotation matrices are also unintuitive for picturing rotations, and thus were not chosen for orientation representation.

Euler angles provide orientation with three distinct rotation matrices about the axis of rotation, as seen in Equations 3.11 to 3.13. The disadvantages of using Euler angles come from rotation order. A $\mathrm{R_x(30)R_y(30)R_z(30)}$ rotation differs from a $\mathrm{R_y(30)R_x(30)R_z(30)}$ rotation, as seen in Figure 3.15. There are six orders of multiplication with Euler angles, all with up to six different final orientations, making it difficult to regress rotation order. Euler angles also face the problem of gimbal locking, a phenomenon in which the second rotation angle goes to 90°, locking the other two rotations [84]. Euler angles were not chosen for orientation representation either.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{3.11}$$

$$R_y(\beta) = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{3.12}$$

$$R_z(\gamma) = \begin{bmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

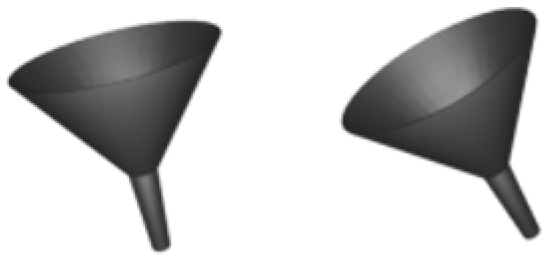
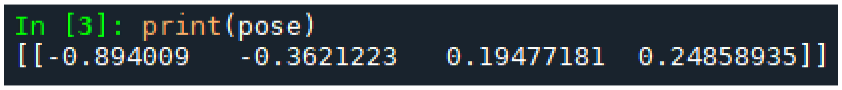


Figure 3.15: On the left is a $R_x(30)R_y(30)R_z(30)$ rotation, on the right is a $R_y(30)R_x(30)R_z(30)$ rotation.

Quaternions are compact as they have four parameters needed for regression. They are computationally efficient, rotation order does not matter, and they do not experience gimbal locking [84].

From the trained network, outputs were in the form of unit quaternions in Python and an example is shown in Figure 3.16. When feeding an input into the pose regression network, the image was read by Python using the *cv2.imread* function, then scaled down by dividing each pixel value by 255. The updated image array was then reshaped to have a size of (-1, 224, 224, 3). To get an output, the *predict* function was used.



```
In [3]: print(pose)
[[-0.894009   -0.3621223   0.19477181  0.24858935]]
```

Figure 3.16: Example of an output from the pose regression network after a prediction.

3.4 Pipeline

The combined network pipeline can be seen in Figure 3.17. An input image of size 224 x 224 x 3 is inputted and goes through the YOLOv2 object detection network discussed in

Chapter 3.2. The output from the YOLOv2 network is the bounding box information consisting of the x and y pixel locations of the top left corner of the box, and the confidence score. Using Equations 3.5 to 3.7 and the intrinsic properties of the RealSense depth camera, along with the depth measurement of the object; the camera world coordinates were found.

The image is cropped within the pipeline using the determined bounding box and expanded to 224 x 224 to input the cropped image into the pose regression network. The output from the pose network is the quaternion orientation of the object as discussed in Chapter 3.3.

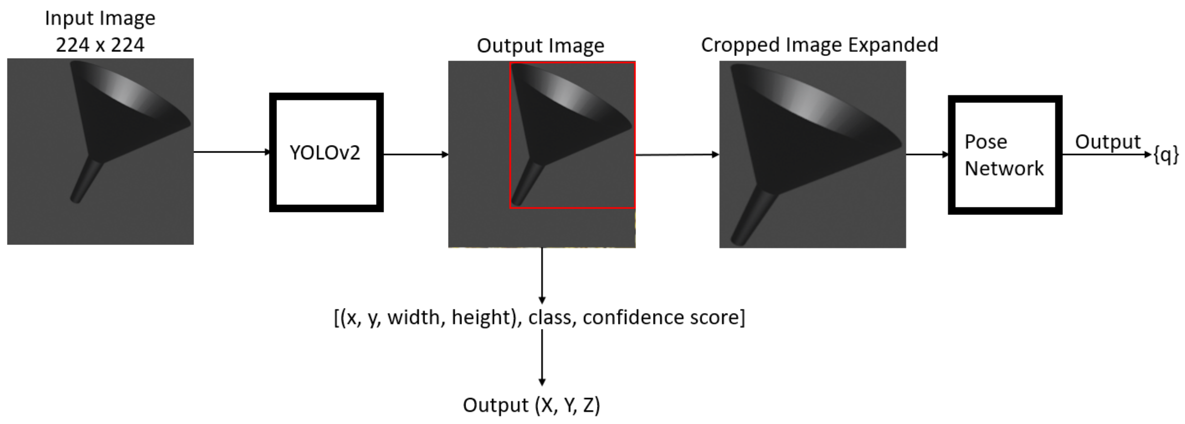


Figure 3.17: Architecture of the custom 6DOF pose detection pipeline.

To get the 6DOF pose output of the object of interest, the quaternion output is converted to the XYZ Euler angles, the same rotation order used when converting training data from Euler angles to ground truth quaternions. The formula for the conversion is seen in Equation 3.14 [86].

$$\begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix} = \begin{Bmatrix} a\tan 2[2(w*x+y*z),1-2(x^2+y^2)] \\ a\sin[2(w*y-x*z)] \\ a\tan 2[2(w*z+x*y),1-2(y^2+z^2)] \end{Bmatrix} \tag{3.14}$$

In total, the combined network consists of 197 layers. There are no parallel branches, all layers are in series. Depth images are not inputted into the network, they are used in parallel to determine the world coordinates of the object.

3.5 Image Parameters

All training images were color images of size 224 x 224 x 3. The training data used for the pose network was normalized by dividing all pixel values by 255 since RGB values have 256 possible values ranging from 0 (black) and 255 (white). Depth maps have distance values encoded into each pixel of the depth image. Their size was 224 x 224 x 1.

The object of interest in the training images was always set to 0.4 m away from the camera, with a spotlight being placed near the object in random locations for different lighting in each frame, as seen in Figure 3.18. Different lighting allows the network to better fit to different lighting environments when used in practice.

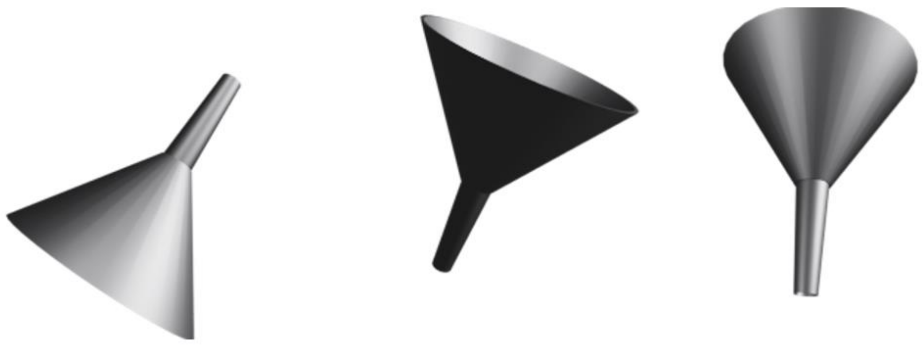


Figure 3.18: The image on the left had the spotlight to the right of the camera, the image in the middle had the spotlight placed above the object, and the image on the right had the spotlight placed above the camera.

3.6 Experimentation and Results

When evaluating object detection networks; Intersection Over Union (IOU), precision, recall, average precision, and mean average precision are used. IoU measures the accuracy of a network and is the ratio of the area of overlap between a predicted bounding box & the ground truth bounding box and the area of the image covered by both bounding boxes. The formula can be seen in Equation 3.15. Figure 3.19 shows how this is calculated.

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} \tag{3.15}$$

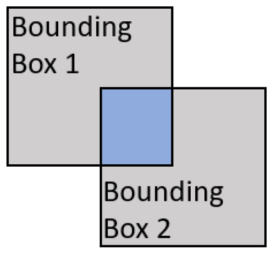


Figure 3.19: The area of overlap is the blue region; the area of union is the gray and blue

regions combined.

Precision is the ratio of true predictions made by the network and the total number of predictions as shown in Equation 3.16. A prediction is a false positive based on an IoU threshold set by the developer, typically less than 0.5.

$$\Pr ecision = \frac{True \Pr edictions}{True \Pr edictions + False \Pr edictions} \tag{3.16}$$

Precision does not consider the total number of objects of interest in an image as it does not account for objects missed by the network, therefore recall is used as well. Recall is the ratio of true predictions and all ground truth positives in the data as seen in Equation 3.17.

$$\mathrm{Re}\, call = \frac{True \Pr edictions}{True \Pr edictions + False\ Negatives} \tag{3.17}$$

Precision and recall are typically plotted against each other in a precision-recall plot, which is a model evaluator. The area under the precision-recall curve is the single performance measure used with precision-recall plots and is known as average precision. The general equation for average precision is seen in Equation 3.18. The average precision is bounded from 0 to 1. When more than one object is being detected by a model in data, mean average precision is used. Each object’s average precision is calculated, then the mean of the average precisions is calculated to find the mean average precision.

$$Average \Pr ecision = \int_0^1 \Pr ecision(R)dR\ where\ R\ is\ \mathrm{Re}\, call \tag{3.18}$$

To test the performance of the YOLOv2 object detection model, test images were rendered virtually and inputted into the network. From the outputted results, the precision-recall plot was generated, and the average precision of the model was found, seen in Figure 3.20. The mean average precision was not generated since only one object class was detected. The average precision was 1.00 on the hardware used to train the network. This is an improvement over the studies conducted in [38] but should take into consideration the fact only one object was being detected in a test set made up of virtual images with little to no noise.

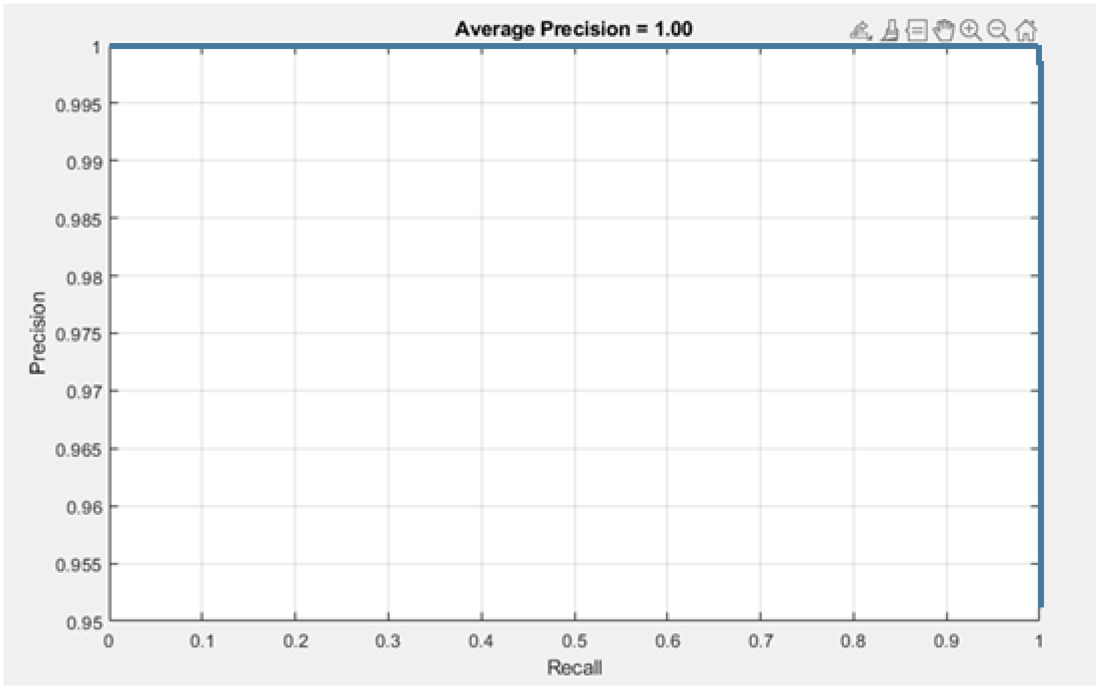


Figure 3.20: The precision-recall plot of the YOLOv2 network.

To test the pose regression network, test images were rendered with the ground truth quaternions. On each test image, predictions were made and were written to a file. The quaternion predictions were converted to Euler angles using an XYZ transformation, the same used to convert ground truth Euler angles to ground truth quaternions. For each prediction, the difference between the ground truth and the prediction was calculated in each of the three axes. The mean error in each axis can be found in Table 3.1.

Table 3.1: The errors in the X, Y, and Z axis of the pose regression network.

X ERROR	Y ERROR	Z ERROR
3.2712°	4.3342°	3.2111°

This is an improvement over the results for the X and Z axes from the study conducted in [27].

In conclusion, a 6DOF pose detection algorithm was developed to localize an object in a 3D space, allowing for a second convolutional network to predict the orientation of the target. The method presented is able to perform at a standard higher than studies done in the field. This algorithm is to be paired with a robotics simulation to successfully rendezvous and dock with a non-cooperative target.

Chapter 4: Board Deployment

Summary: In this chapter, a detailed description of the methodology for deploying the computer vision algorithm discussed in Chapter 3 on a microcomputer is given. Since the objective of this thesis is to develop a simulation of space robotics rendezvous and docking for active space debris removal, it is of interest to deploy the computer vision algorithm on a computer with computational abilities similar to spacecraft computers. An NVIDIA Jetson Nano is used for testing the feasibility of the developed computer vision program.

4.1 Introduction

To further test the 6DOF pose detection network discussed in Chapter 3 for space applications, it is to be tested on a microcomputer. The microcomputer being used is an NVIDIA Jetson Nano, which can be seen in Figure 4.1.

Onboard computers on satellites include microprocessors, memory banks, and interfacing chips to communicate with different subsystems. They are used for attitude determination and control using sensors and actuators, telemetry management, communications, housekeeping, on-board time synchronizing, and fault detection [87].

Of the three heat transfer methods, radiation is most used in space. Convection requires the transfer of heat energy through a fluid, conduction requires contact between two substances to move heat, whereas radiation transfers heat energy through space by electromagnetic radiation [88]. Therefore, on-board computers do not have fans, as no air is present to dissipate heat.

An effective on-board computer has good processing power with the ability to process desired operations, such as the NVIDIA Jetson Nano's dedicated GPU capable of handling computer vision algorithms. They also have reliable software, use low power, and have a small physical footprint [87].

The NVIDIA Jetson Nano was chosen for testing the computer vision algorithms due to its cooling system. It does not use a fan and allows for modular heat exchange systems to be developed by the end user for their needs. The operating temperature of the board is -25℃ to 97℃. The Jetson Nano was also tested for mechanical shock loads of 140 G (multiples of gravitational acceleration) and 50 G, sine vibrations of 3 G, and random vibrations of 2 G and 1 G; all tests completed on space grade computers to withstand random shocks to the system during operational and launch loads [89]. The operating system on the board is Linux based. The board is 100 mm x 80 mm x 29 mm and uses 5 to 10 W of power depending on usage.

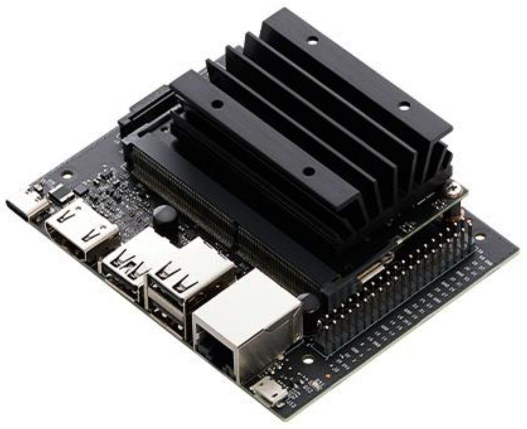


Figure 4.1: An image of the NVIDIA Jetson Nano board [90].

4.2 Board Hardware

Onboard the NVIDIA Jetson Nano is 4 GB of memory, a quad-core ARM Cortex-A57 processor, and an NVIDIA Maxwell GPU as mentioned in Table 1.1. Maxwell GPUs are used for computational applications such as machine learning using built in CUDA cores. The board has three (3) USB ports for peripherals such as a depth camera and MIPI CSI-2 connectors to directly connect a compatible camera sensor to the board. The advantage of this board is it can decode video inputs at high resolutions, optimal for object tracking.

4.3 Experimentation and Results

Initially, the NVIDIA Jetson Nano's operating system was flashed on to a micro-SD card to be able to use the board. Figures 4.2 and 4.3 below show the experiment setup. Connected to the board during the experiment were a keyboard, mouse, the RealSense d435i camera, ethernet, and an HDMI cable for display. To use the RealSense d435i camera on the NVIDIA Jetson Nano board, the RealSense wrapper developed by Intel needed to be installed as it did on MATLAB. This was a simple installation and allowed for the use of the depth sensor and for the extraction of the intrinsic matrix, although the intrinsic matrix was already known. After installing the RealSense library, it could be called in Python code [91].

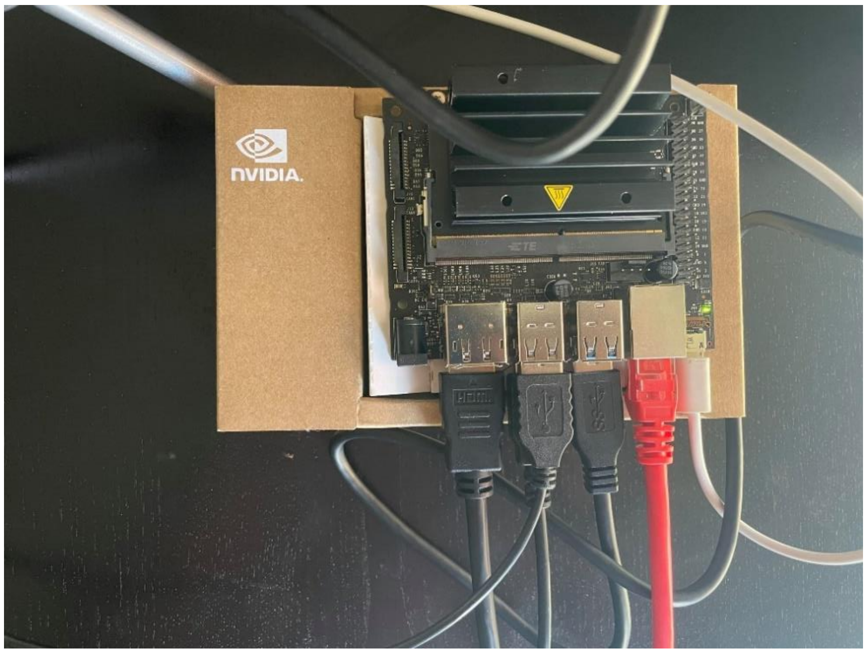


Figure 4.2: The NVIDIA Jetson Nano with all connections during the experiment.

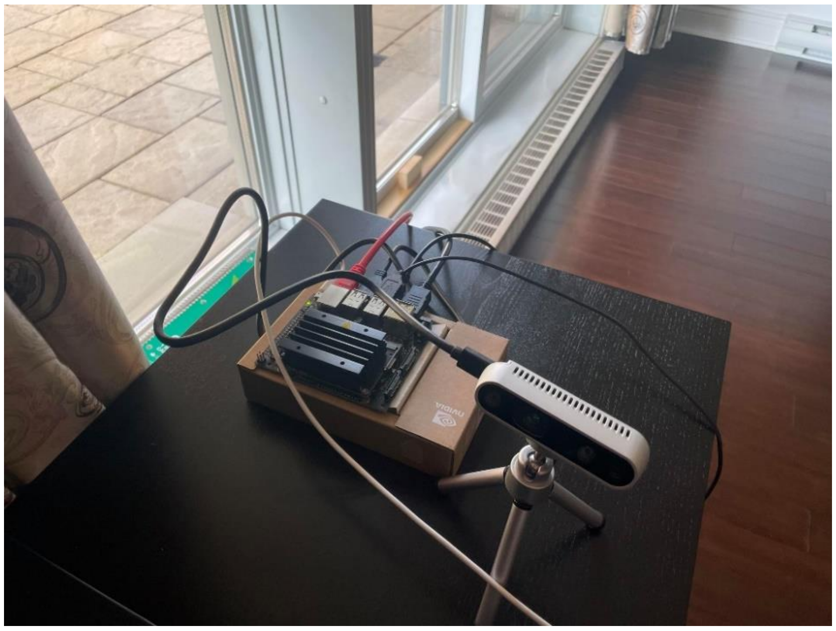


Figure 4.3: A wider view of the experimental setup with the camera.

To run the object detection component on the board, a YOLOv2 network using darkflow was installed on the board with the imported weights trained in MATLAB, rather than converting the MATLAB network to Python for execution on the board [92]. Using the weights trained on MATLAB allow for the MATLAB trained network to be replicated on the board. A separate script was used to transform from the pixel coordinates of the image to camera world coordinates. Again, the funnel, this time physical, was used.

To use the trained 3D regression network for 3D pose detection, the code and surrounding library was uploaded to GitLab, and was then downloaded to the board. All dependencies needed were installed on the board, such as TensorFlow and Keras to be able to use the developed network.

When combining the two networks into a singular pipeline and running the algorithm together, the 6DOF pose detection was capable of running at 3 fps, with spikes to 6 fps during some time intervals and with dips to 1 fps. The regression network is small, as described in Chapter 3, and therefore did not play a large effect on the performance of the overall pipeline. The YOLOv2 network from [92] was averaging approximately 4 fps for the original designer on the NVIDIA Jetson Nano.

This is not a feasible result, and therefore it is recommended that separate boards are used to run the 3D object localization component and the 3D orientation detection, and then the result being combined. This is possible due to the NVIDIA Jetson Nano’s small footprint and low power consumption. The YOLOv5 network can be used to replace the YOLOv2 network as well on the board as it is an improvement over the latter and is faster, providing up to 12 fps on the NVIDIA Jetson Nano [93].

Chapter 5: Creating Virtual Training and Testing Data

Summary: In this chapter, a detailed description of the methodology for rendering virtual data sets in varying lighting and backgrounds is given. Within the script for rendering images, the object of interest is placed at random orientations a set distance away from the camera and a light source is placed in random positions around the object of interest. Ground truth bounding boxes and quaternions are automatically generated. Using virtual data limits noise and is an effective method for training neural networks.

5.1 Introduction

The current thesis is motivated by the need for a computer vision algorithm used for effectively detecting the 6DOF pose of an object in varying environmental conditions. Training neural networks to be effective requires a wide variety and a large size of training data, which is difficult to obtain with real images. Along with the images, accurate ground truth labeling is needed. With real images, labeling tends to be completed by human input which has its own biases. One individual may bound an object differently than another or may bound an object in one image differently than in another image. For pose detection, labeling real images with the orientation of an object relative to the camera introduces human error into measurements. The errors in annotations are introduced into the neural network and can lead to overfitting. Real images also tend to be noisy as not all camera sensors are the same. Some images may be noisy with lower resolutions compared to others. With noise, overfitting is also introduced into a network. To overcome these limitations, an algorithm to render virtual images with accurate ground truth labels of any size in different lighting conditions and different object orientations was developed.

5.2 Advantages of Using Virtual Data

There are many advantages to using virtual data for training. The lighting position, lighting intensity, and light temperature can be adjusted in every rendered frame with code or by manually changing parameters in a software. Camera parameters such as focal length and resolution can be adjusted for more diversity in training datasets. The background can be changed, and occlusions can easily be added. Colors can be mixed, and material properties can be adjusted. Images can be saved in the PNG format for lossless saving of data, as compared to the compression formatting of JPEG images [94].

As discussed in Section 5.1, the virtual ground truth labels are accurate and do not incorporate human biases and error when automating labeling. Thousands of images can be generated with ground truth labels in minutes, as compared to the tedious work of manually labeling real images.

As mentioned above, some real images can be noisy and another advantage of using virtual data is the ability to decrease or eliminate noise. Prior to the use of virtual data, the following figures were used for training. These images were taken with a low-quality camera, the only available at the time, and introduced noise into the training images. The ground truth labels also needed to be constructed manually, introducing human error.

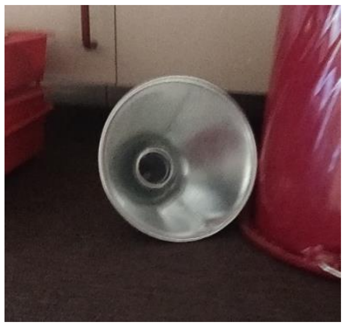


Figure 5.1: An example of real images used for initially training a network without ground truth labels.

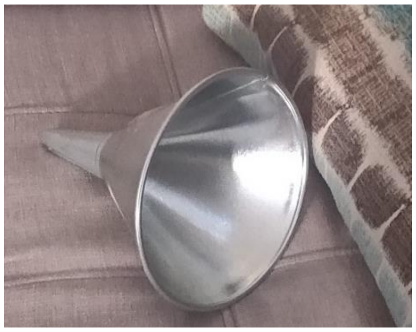


Figure 5.2: An example of real images used for initially training a network without ground truth labels.

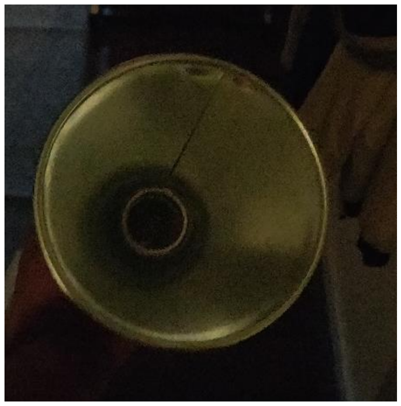


Figure 5.3: An example of real images used for initially training a network without ground truth labels.

5.3 Methodology

To virtually render images, initially a CAD model of the object of interest, a funnel, was developed using SolidWorks and an STL file was generated from it. The STL model can be seen in Figure 5.4.

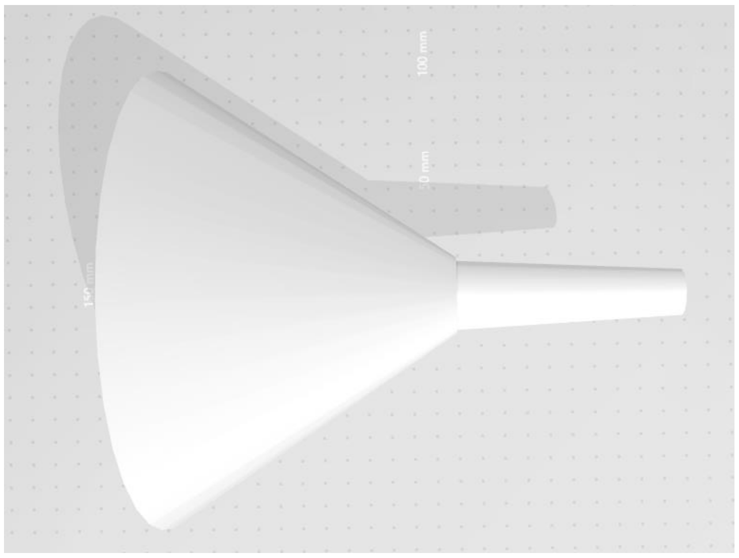


Figure 5.4: Custom STL file of a funnel.

The STL file was then imported into Blender, a software used for 3D modeling, animations, and rendering images of objects. In Blender, the color of the object was changed to match that of the real funnel seen in Figure 5.1. The reflective and absorptive properties were changed to resemble a steel funnel. The scaling of the object was changed as well to depict an accurate representation of the object in a real scene. Figure 5.5 shows the dimensions of the funnel.

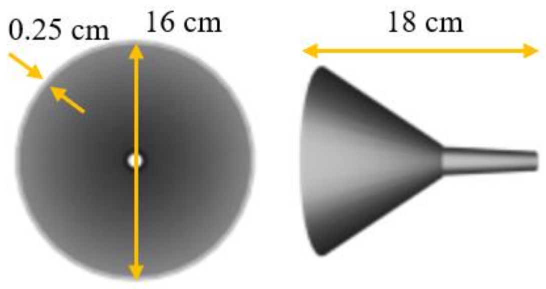


Figure 5.5: Scaling of the virtual funnel.

To automatically render images, Python was used to script functionality into Blender. The camera sensor was placed at 0.4 m away from the center of the funnel along the Z-axis of the Blender environment. Figure 5.6 shows the setup in the Blender world and Figure 5.7 shows the default orientation of the funnel.

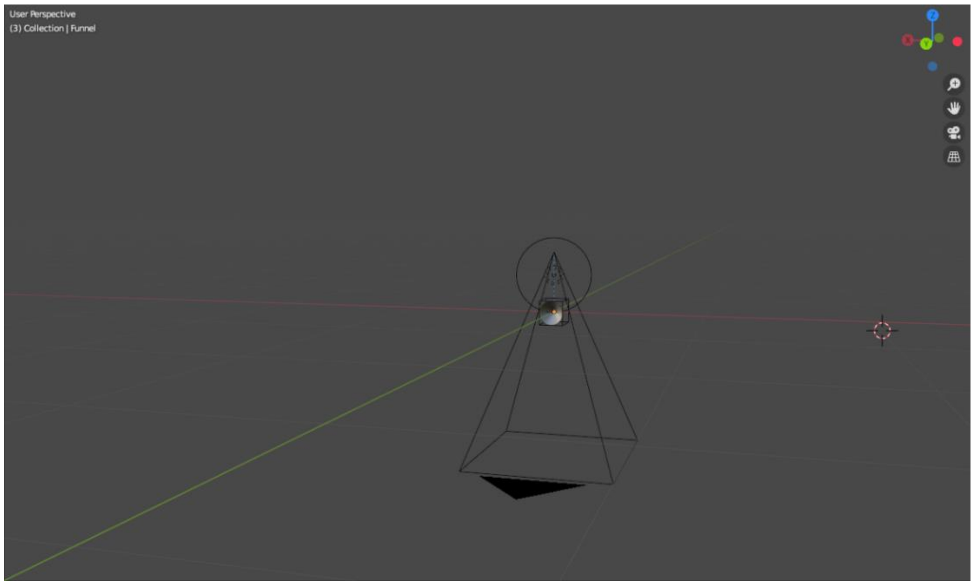


Figure 5.6: Environment setup in Blender. The red line is the X-axis, the green line is the Y-axis, and the blue line is the Z-axis.

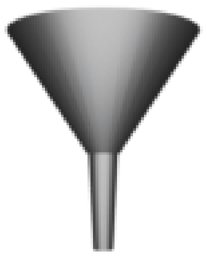


Figure 5.7: Default orientation of the funnel.

In Blender, the script automatically rotated the funnel using XYZ Euler transformations within the ranges listed in Table 5.1. The spotlight used for illumination was randomly placed at any point within the combination of the ranges listed in Table 5.2.

Table 5.1: The ranges of orientation of the funnel used in each axis during data generation.

AXIS	**RANGE**
X	-45˚to 45˚
Y	-45˚to 45˚
Z	0˚to 180˚

Table 5.2: The ranges of the position of the spotlight used in each axis during data generation.

AXIS	**RANGE**
X	-2 m to 2 m
Y	-2 m to 2 m
Z	0 m to 2 m

With changing the parameters i and j, the number of images generated can be found with Equation 5.1.

$$Number\ of\ images = i*(j+1)^2 \tag{5.1}$$

The image size can be changed to train networks of different sizes. With every render, the funnel's XYZ orientation was recorded along with each renders file path in separate text files.

To automate the generation of ground truth bounding box data, the images were all inputted into a MATLAB script using each image's file path. Since the images had transparent backgrounds, MATLAB automatically assigned the background pixels a value of 0. To find the bounding box coordinates; the left most, right most, top most, and bottom most pixel indices of value greater than 0 were found. Each image needed a training label consisting of the top left coordinate, width, and height. Table 5.3 lists how to find these values. Figure 5.8 below shows a ground truth label around the funnel.

Table 5.3: Method for finding the different coordinates of a bounding box.

TOP LEFT COORDINATE OF THE BOUNDING BOX	(Left most pixel index, Right most pixel index)
WIDTH	Right most pixel index – left most pixel index
HEIGHT	Bottom most pixel index – top most pixel index

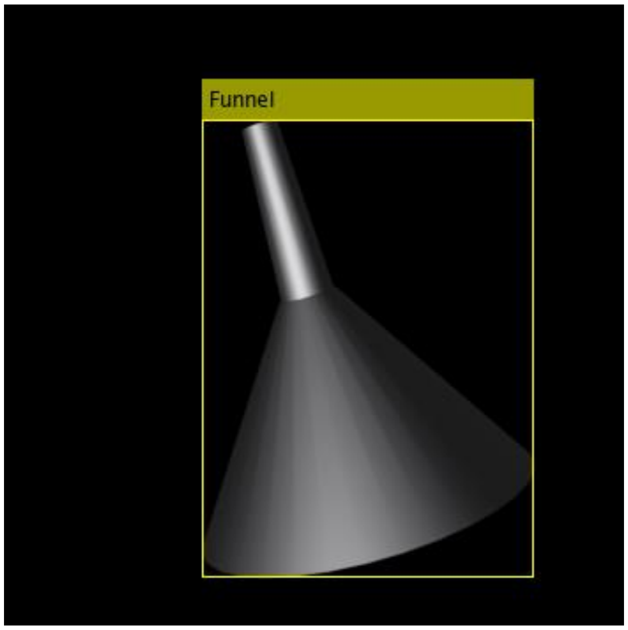
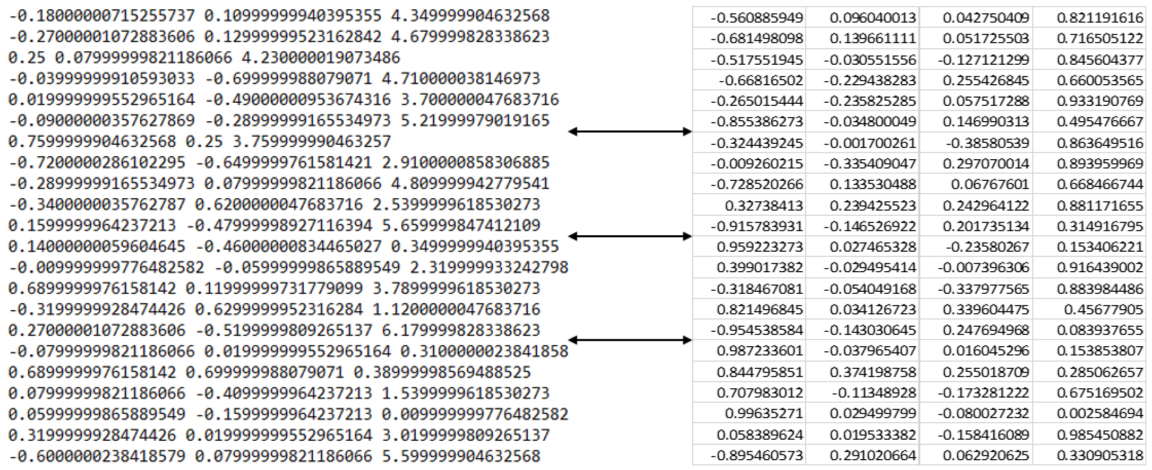


Figure 5.8: Automated ground truth label around the virtual funnel in a training image.

To generate quaternion ground truth labels, the XYZ Euler angles generated in Blender were inputted into a MATLAB script which used Equation 3.14 for the transformation. The quaternion labels were outputted into a Microsoft Excel file along with each image's file path. The Microsoft Excel file was inputted into a Python script for training the pose regression network discussed in Chapter 3.3. Figure 5.9 shows the format of XYZ Euler angles outputs from Blender in a text file and the quaternions in a Microsoft Excel spreadsheet.



```
-0.18000000715255737 0.10999999940395355 4.349999904632568
-0.27000001072883606 0.12999999523162842 4.679999828338623
0.25 0.07999999821186066 4.230000019073486
-0.03999999910593033 -0.699999988079071 4.710000038146973
0.019999999552965164 -0.49000000953674316 3.700000047683716
-0.09000000357627869 -0.28999999165534973 5.21999979019165
0.7599999904632568 0.25 3.759999990463257
-0.7200000286102295 -0.6499999761581421 2.9100000858306885
-0.28999999165534973 0.07999999821186066 4.809999942779541
-0.3400000035762787 0.6200000047683716 2.5399999618530273
0.1599999964237213 -0.47999998927116394 5.659999847412109
0.14000000059604645 -0.46000000834465027 0.3499999940395355
-0.009999999776482582 -0.05999999865889549 2.319999933242798
0.6899999976158142 0.11999999731779099 3.7899999618530273
-0.3199999928474426 0.6299999952316284 1.1200000047683716
0.27000001072883606 -0.5199999809265137 6.179999828338623
-0.07999999821186066 0.019999999552965164 0.3100000023841858
0.6899999976158142 0.699999988079071 0.38999998569488525
0.07999999821186066 -0.4099999964237213 1.5399999618530273
0.05999999865889549 -0.1599999964237213 0.009999999776482582
0.3199999928474426 0.019999999552965164 3.0199999809265137
-0.6000000238418579 0.07999999821186066 5.599999904632568
```

-0.560885949	0.096040013	0.042750409	0.821191616
-0.681498098	0.139661111	0.051725503	0.716505122
-0.517551945	-0.030551556	-0.127121299	0.845604377
-0.66816502	-0.229438283	0.255426845	0.660053565
-0.265015444	-0.235825285	0.057517288	0.933190769
-0.855386273	-0.034800049	0.146990313	0.495476667
-0.324439245	-0.001700261	-0.38580539	0.863649516
-0.009260215	-0.335409047	0.297070014	0.893959969
-0.728520266	0.133530488	0.06767601	0.668466744
0.32738413	0.239425523	0.242964122	0.881171655
-0.915783931	-0.146526922	0.201735134	0.314916795
0.959223273	0.027465328	-0.23580267	0.153406221
0.399017382	-0.029495414	-0.007396306	0.916439002
-0.318467081	-0.054049168	-0.337977565	0.883984486
0.821496845	0.034126723	0.339604475	0.45677905
-0.954538584	-0.143030645	0.247694968	0.083937655
0.987233601	-0.037965407	0.016045296	0.153853807
0.844795851	0.374198758	0.255018709	0.285062657
0.707983012	-0.11348928	-0.173281222	0.675169502
0.99635271	0.029499799	-0.080027232	0.002584694
0.058389624	0.019533382	-0.158416089	0.985450882
-0.895460573	0.291020664	0.062920625	0.330905318

Figure 5.9: An example of Euler outputs from Blender (left) converted to their respective Quaternions (right).

In conclusion, a technique was developed to generate an arbitrary number of virtual images for the purpose of training artificial neural networks with bounding box and orientation ground truth. It is shown in Chapter 3 that using virtual images expands the training set and allows neural networks to be effective.

Chapter 6: Robot Control

Summary: In this chapter, a detailed description of the FANUC M20iD/25 robot in a virtual environment is given. The physical specifications are described, and the robot's equations are derived using the Denavit-Hartenberg method. Limitations are placed on the robot joints in terms of range and velocity to keep the virtual setup consistent with the future Hardware-in-the-Loop setup. Finally, control methodologies are described for controlling the robot along in the Simulink and ROS environments for running experiments on the visual servo-controlled robot. Finally, the results are shown for the joint movements.

6.1 Introduction

The current thesis is motivated by the need for an effective robotic control technique for real-time object rendezvous and docking in the space environment. In Chapter 3, a computer vision algorithm was developed for the purpose of being paired with a robot to grasp a target object. To test real-time object rendezvous and docking, two industrial robots will be used in a Hardware-in-the-Loop simulation at the Space Engineering Design Lab at York University. A proposed general setup is seen in Figure 6.1. The two robots are FANUC M20iD/25 robots, one being the chaser robot paired with the computer vision algorithm and robotic controls algorithms, and a gripper; the second being the target robot equipped with a mock satellite and exhibiting free-flying motion.

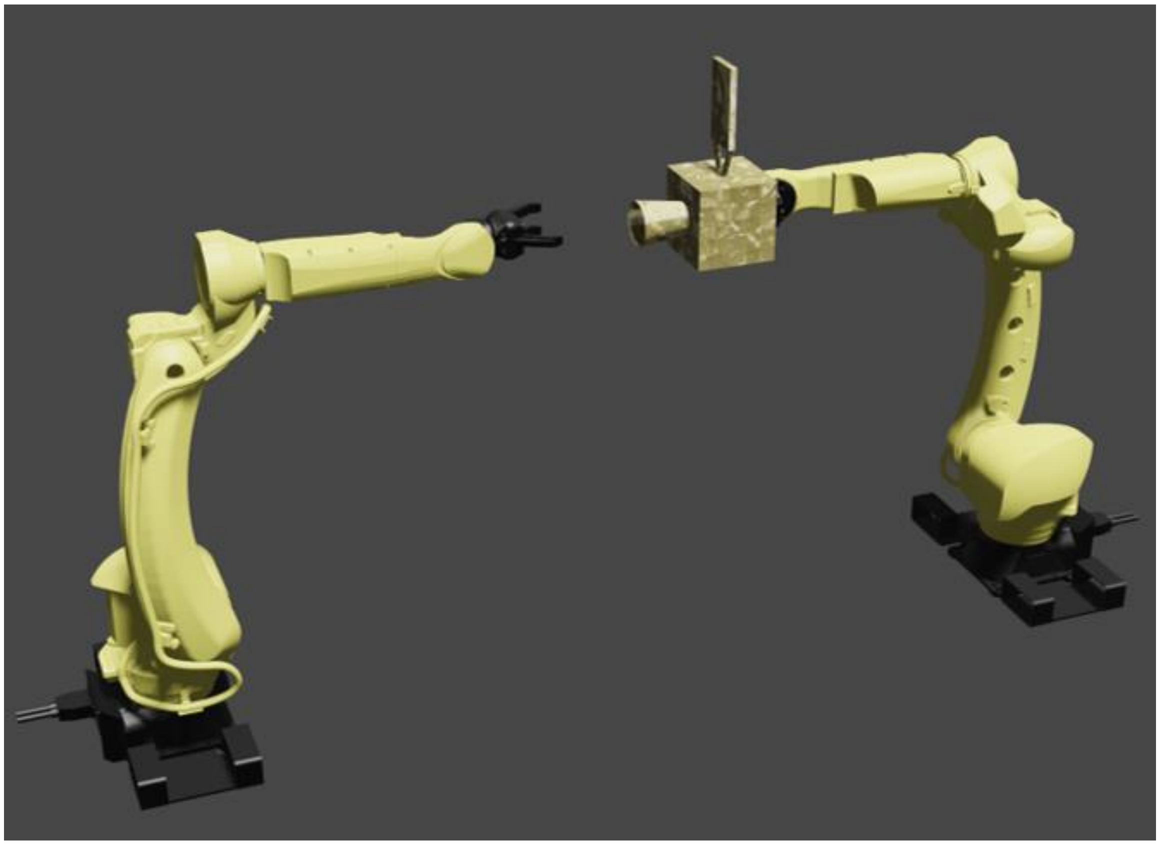


Figure 6.1: The proposed setup for the FANUC robots at the Space Engineering Design Lab at York University.

Unfortunately, due to the COVID-19 pandemic, lab access was restricted, resulting in the robotics component being completed in a virtual environment with a Software-in-the-Loop simulation. A kinematic representation of the robots is used to drive the chaser robot to the target after using the 6DOF pose detection technique to accurately locate the target. PI controllers are used to drive each joint to the desired position, in turn moving the end-effector. MATLAB Simulink and the Robot Operating System are used together to test docking with the target.

6.2 Robot Specifications

An image of the FANUC M20iD/25 robot can be seen in Figure 6.2.

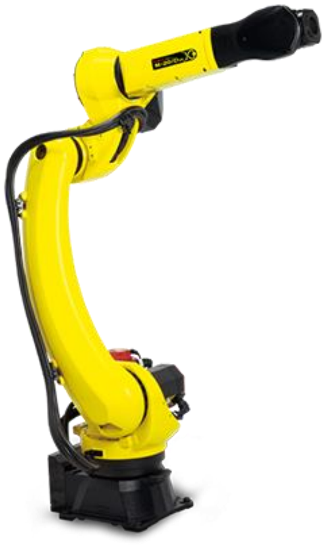


Figure 6.2: The FANUC M20iD/25 robot [95].

Table 6.1: Joint ranges and maximum joint speeds of the FANUC M20iD/25 robot.

JOINT	MAXIMUM RANGE	MAXIMUM SPEED
1	340°	210°/sec
2	260°	210°/sec
3	475°	265°/sec
4	400°	420°/sec
5	280°	420°/sec
6	540°	720°/sec

Table 6.1 shows the range of motion for this robot in all its six (6) axes along with the maximum rotational speed. The operational weight of one robot is 250 kg and can carry a maximum load of 25 kg, allowing for a mock satellite to easily be placed on the end effector of the target robot [96]. The robot's motion range can be seen in Figure 6.3.

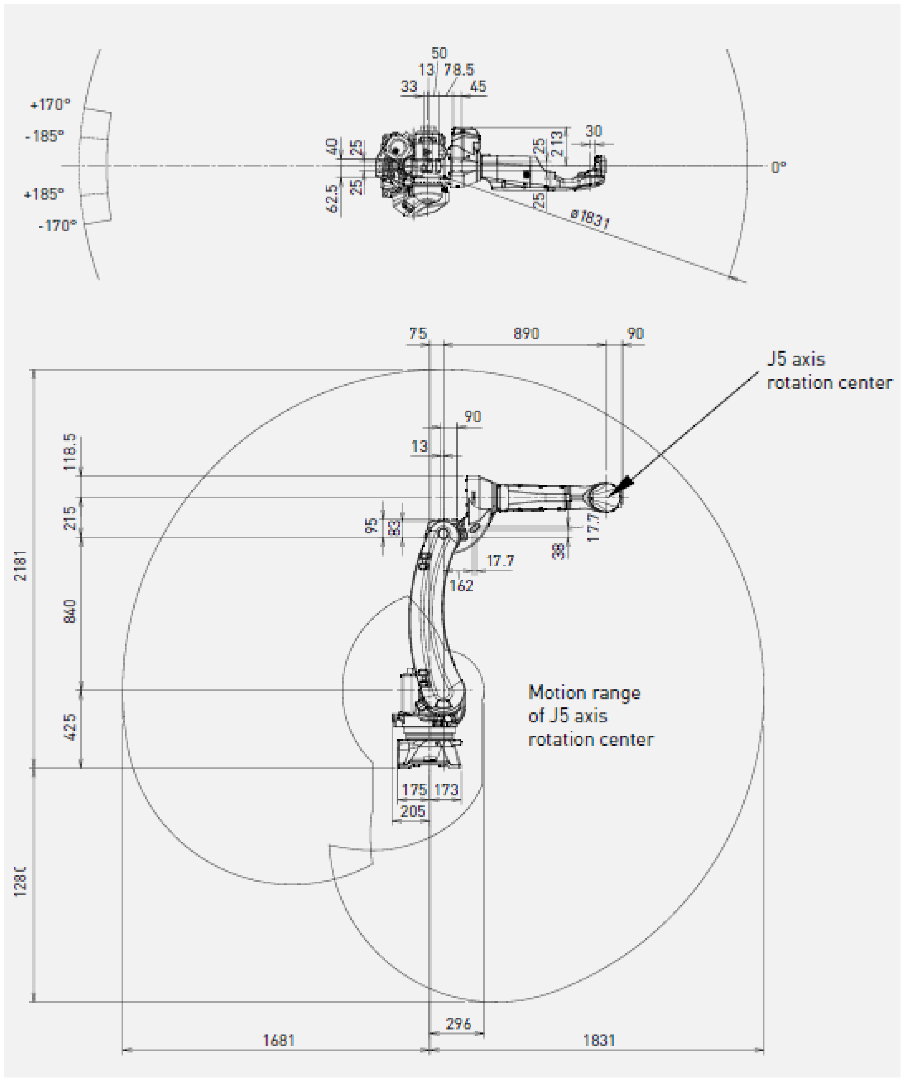


Figure 6.3: The motion range of the FANUC M20iD/25 robot [97].

6.3 Limitations on the Robot

The Software-in-the-Loop virtual simulation is meant to mimic the proposes Hardware-in-the-Loop simulation, therefore the limitations intended to be imposed on the hardware must be imposed on the virtual robot. Table 6.2 lists these limitations on each of the six (6) joints.

Table 6.2: Joint limits imposed on the FANUC M20iD/25 robot.

JOINT	JOINT MINIMUM	JOINT MAXIMUM
1	-29.8 $^\circ$	29.8 $^\circ$
2	-20.1 $^\circ$	60.2 $^\circ$
3	-45.3 $^\circ$	45.3 $^\circ$
4	-90.0 $^\circ$	90.0 $^\circ$
5	-45.3 $^\circ$	45.3 $^\circ$
6	-90.0 $^\circ$	90.0 $^\circ$

The range limitations are used to avoid walls and other obstacles when operating in a small laboratory area during the Hardware-in-the-Loop simulation. The velocity limits were set to the maximum values presented in Table 6.1.

6.4 Robot Equations

The proposed control methodology is based on kinematics; therefore, the forward kinematics equations need to be derived for the FANUC M20iD/25 robot. The forward kinematics of a manipulator relates the end-effector position and orientation to the inputted joint parameters. To derive the forward kinematics equations, a geometric approach can be used, where each joint position is determined by adding the previous link length vectors and their orientations as shown in Figure 6.4.

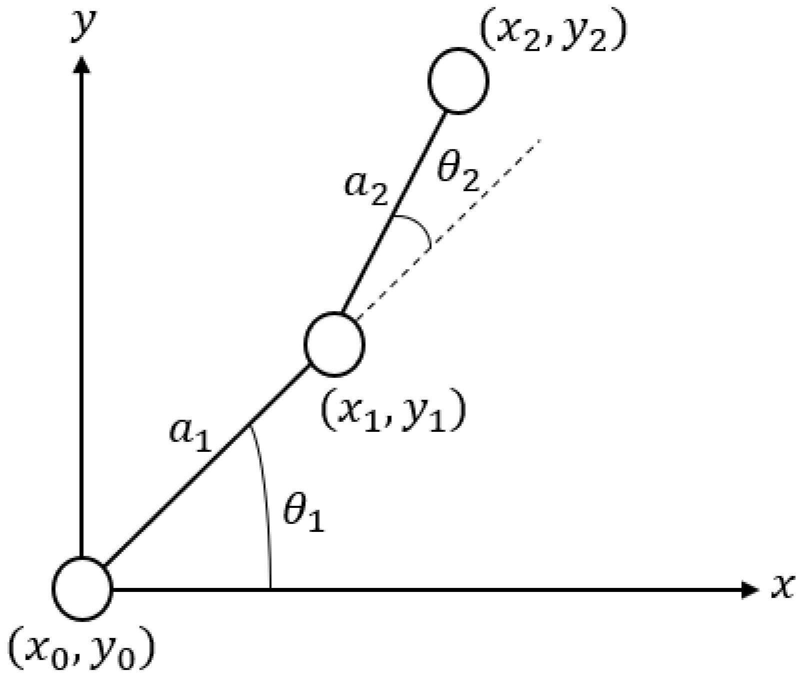


Figure 6.4: A planar 2 link kinematic arm.

In Figure 6.4, to get the end-effectors position (x_2, y_2) in terms of the base frame (x_0, y_0), Equations 6.1 and 6.2 are derived. This gets more complex as more robot links are added to the system and as the system expands into 3-dimensions.

$$x_2 = a_1 \cos(\theta_1) + a_2 \cos(\theta_1 + \theta_2) \tag{6.1}$$

$$y_2 = a_1 \sin(\theta_1) + a_2 \sin(\theta_1 + \theta_2) \tag{6.2}$$

An alternative to the geometric approach is using the Denavit-Hartenberg (DH) method for complex systems. The DH convention simplifies the forward kinematics problem by representing each homogeneous transformation from one joint to another with four (4) parameters associated with each link and joint. The four parameters are: (1) l_i: the link length, (2) b_i: the link offset, (3) α_i: the link twist, and (4) θ_i: the joint variable (θ_i for a revolute joint, d_i for a prismatic joint) [98]. Figure 6.5 shows a general DH frame assignment.

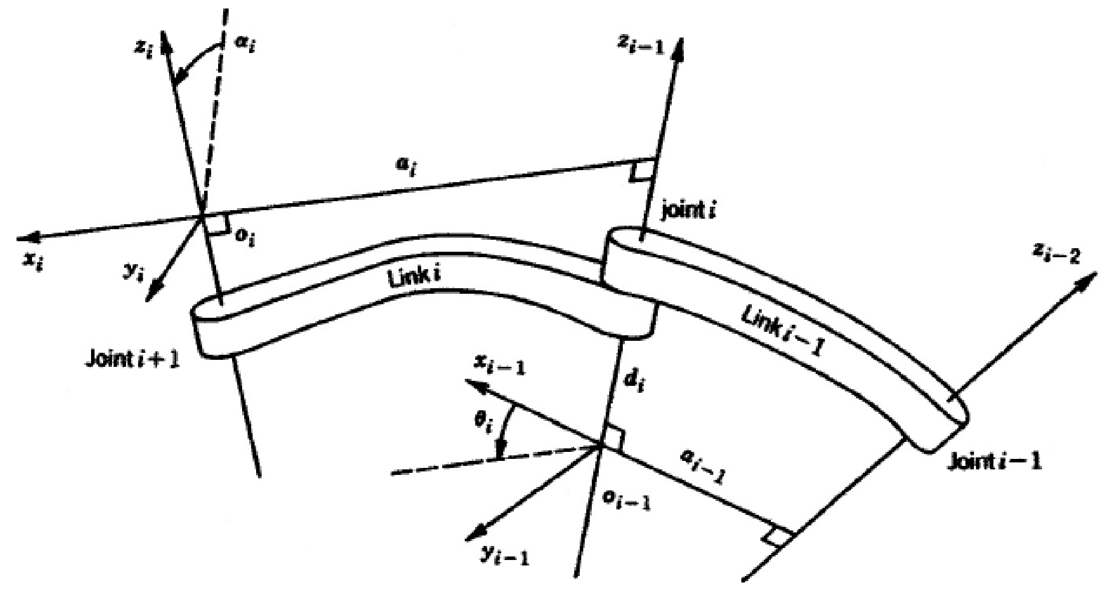


Figure 6.5: Parameters used with the DH method for forward kinematics [98].

As seen in Figure 6.5, θ_i is the angle between axis x_{i-1} and x_i from x_{i-1} to x_i along the z_{i-1} axis. The link twist, α_i is the angle between z_{i-1} and z_i from z_{i-1} to z_i along the x_i axis.

The calculations shown from this point forward will use the following notation in reference to Figure 6.5: axis i-1 in Figure 6.5 will be referenced as i and axis i in Figure 6.5 will be referenced as i+1. The link length will be referred to as l_i and the link offset will be referred to as b_i.

To retrieve the end-effector position and orientation in the base frame, the following equations are used.

$$Position = P = a_1 + Q_1 a_2 + Q_1 Q_2 a_3 + Q_1 Q_2 Q_3 a_4 + Q_1 Q_2 Q_3 Q_4 a_5 + Q_1 Q_2 Q_3 Q_4 Q_5 a_6 \quad (6.3)$$

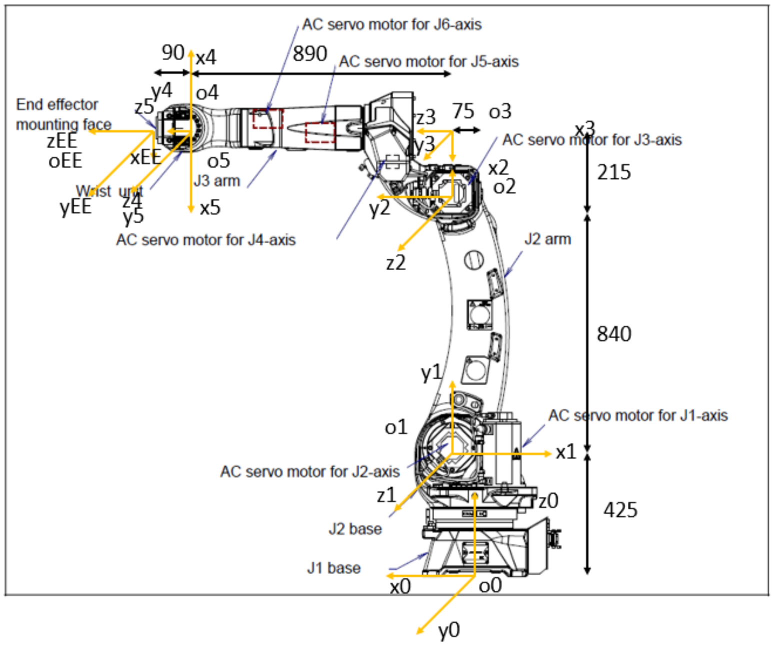
$$Orientation = Q = Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 \quad (6.4)$$

where

$$Q_i = \begin{bmatrix} cos\theta_i & -cos\alpha_i sin\theta_i & sin\alpha_i sin\theta_i \\ sin\theta_i & cos\alpha_i cos\theta_i & -sin\alpha_i cos\theta_i \\ 0 & sin\alpha_i & cos\alpha_i \end{bmatrix} \quad (6.5)$$

$$a_i = \begin{bmatrix} l_i cos\theta_i \\ l_i sin\theta_i \\ b_i \end{bmatrix} \quad (6.6)$$

Figures 6.6 and 6.7 show the DH frames for the FANUC M20iD/25 robot. Table 6.3 is the DH table for the FANUC M20iD/25, in which all the DH parameters are listed for each of the six (6) joints.



AC servo motor for J6-axis
90
x4
890
AC servo motor for J5-axis
y4
o4
End effector mounting face
z5
zEE
oEE
xEE
o5
J3 arm
Wrist unit
yEE
z4
y5
x5
AC servo motor for J4-axis
z3
75
o3
x3
AC servo motor for J3-axis
y3
x2
o2
215
y2
z2
J2 arm
840
y1
o1
AC servo motor for J1-axis
x1
AC servo motor for J2-axis
z1
z0
425
J2 base
J1 base
x0
o0
y0

Figure 6.6: The joint frames for the DH method. As seen in Figure 6.3, all frames originate in the same plane (x_0-z_0 plane).

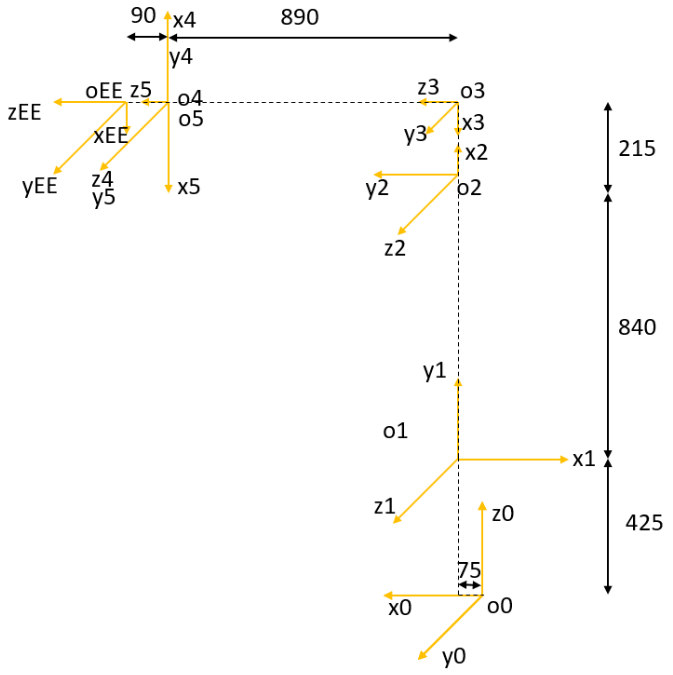


Figure 6.7: Another image of the derived DH frames for the FANUC M20iD/25 robot.

Table 6.3: The DH table for the FANUC M20iD/25 robot.

LINK	l_i (mm)	b_i (mm)	α_i	θ_i
1	75	425	+90°	θ_1
2	840	0	0°	θ_2
3	215	0	+90°	θ_3
4	0	890	+90°	θ_4
5	0	0	+90°	θ_5
6	90	0	0°	θ_6

The following are the homogeneous transformation matrices for each link and the a_i vectors, derived using Table 6.3 and Equations 6.5 and 6.6.

$$Q_1 = \begin{bmatrix} cos\theta_1 & 0 & sin\theta_1 \\ sin\theta_1 & 0 & -cos\theta_1 \\ 0 & 1 & 0 \end{bmatrix} \tag{6.7}$$

$$Q_2 = \begin{bmatrix} cos\theta_2 & -sin\theta_2 & 0 \\ sin\theta_2 & cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.8}$$

$$Q_3 = \begin{bmatrix} cos\theta_3 & 0 & sin\theta_3 \\ sin\theta_3 & 0 & -cos\theta_3 \\ 0 & 1 & 0 \end{bmatrix} \tag{6.9}$$

$$Q_4 = \begin{bmatrix} cos\theta_4 & 0 & sin\theta_4 \\ sin\theta_4 & 0 & -cos\theta_4 \\ 0 & 1 & 0 \end{bmatrix} \quad (6.10)$$

$$Q_5 = \begin{bmatrix} cos\theta_5 & 0 & sin\theta_5 \\ sin\theta_5 & 0 & -cos\theta_5 \\ 0 & 1 & 0 \end{bmatrix} \quad (6.11)$$

$$Q_6 = \begin{bmatrix} cos\theta_6 & -sin\theta_6 & 0 \\ sin\theta_6 & cos\theta_6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.12)$$

$$a_1 = \begin{bmatrix} 75cos\theta_1 \\ 75sin\theta_1 \\ 425 \end{bmatrix} \quad (6.13)$$

$$a_2 = \begin{bmatrix} 840cos\theta_2 \\ 840sin\theta_2 \\ 0 \end{bmatrix} \quad (6.14)$$

$$a_3 = \begin{bmatrix} 215cos\theta_3 \\ 215sin\theta_3 \\ 0 \end{bmatrix} \quad (6.15)$$

$$a_4 = \begin{bmatrix} 0 \\ 0 \\ 890 \end{bmatrix} \quad (6.16)$$

$$a_5 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (6.17)$$

$$a_6 = \begin{bmatrix} 90cos\theta_6 \\ 90sin\theta_6 \\ 0 \end{bmatrix} \quad (6.18)$$

Using Equation 6.3, the following are the expressions for the end-effector position with respect to the individual joint values in the robot base frame.

$$\begin{aligned}
&x = 75\cos(\theta_1) + 840\cos(\theta_1)\cos(\theta_2) + 215\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \\
&-215\cos(\theta_1)\sin(\theta_2)\sin(\theta_3) + 890\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\
&+890\cos(\theta_1)\sin(\theta_2)\sin(\theta_3) \\
&+90(((\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - \cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\
&+\sin(\theta_1)\sin(\theta_4))\cos(\theta_5) \\
&+(\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) + \cos(\theta_1)\sin(\theta_2)\cos(\theta_3))\sin(\theta_5))\cos(\theta_6) \\
&+90((\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - \cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\sin(\theta_4) \\
&-\sin(\theta_1)\cos(\theta_4))\sin(\theta_6)
\end{aligned} \quad (6.19)$$

$$\begin{aligned}
&y = 75\sin(\theta_1) + 840\sin(\theta_1)\sin(\theta_2) + 215\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\
&-215\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) + 890\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\
&+890\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \\
&+90(((\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\
&+\cos(\theta_1)\sin(\theta_4))\cos(\theta_5) \\
&+(\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) + \sin(\theta_1)\sin(\theta_2)\cos(\theta_3))\sin(\theta_5))\cos(\theta_6) \\
&+90((\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\sin(\theta_4) \\
&-\cos(\theta_1)\cos(\theta_4))\sin(\theta_6)
\end{aligned} \quad (6.20)$$

$$
\begin{aligned}
z &= 425+840\sin(\theta_2)+215\sin(\theta_2)\cos(\theta_3)+215\cos(\theta_2)\sin(\theta_3) \\
&+890\sin(\theta_2)\sin(\theta_3)-890\cos(\theta_2)\cos(\theta_3) \\
&+90\begin{pmatrix}(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\cos(\theta_4)\cos(\theta_5) \\ +(\sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3))\sin(\theta_5)\end{pmatrix}\cos(\theta_6) \\
&+90(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\sin(\theta_4)\sin(\theta_6)
\end{aligned}
\tag{6.21}
$$

To go along with the end-effector positions, the end-effector orientation matrix can be found using Equation 6.4. Due to the size of the elements, each of the nine (9) components are listed separately below.

$$
\begin{aligned}
Q_{11} = &\begin{pmatrix}\begin{pmatrix}\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)- \\ \cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4) \\ +\sin(\theta_1)\sin(\theta_4)\end{pmatrix}\cos(\theta_5) \\ +\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{pmatrix}\sin(\theta_5)\end{pmatrix}\cos(\theta_6) \\
&+\begin{pmatrix}\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4) \\ -\sin(\theta_1)\cos(\theta_4)\end{pmatrix}\sin(\theta_6)
\end{aligned}
\tag{6.22}
$$

$$
\begin{aligned}
Q_{12} = -&\begin{pmatrix}\begin{pmatrix}\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)- \\ \cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4) \\ +\sin(\theta_1)\sin(\theta_4)\end{pmatrix}\cos(\theta_5)+ \\ \begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{pmatrix}\sin(\theta_5)\end{pmatrix}\sin(\theta_6) \\
&+(\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4) \\
&-\sin(\theta_1)\cos(\theta_4))\cos(\theta_6)
\end{aligned}
\tag{6.23}
$$

$$Q_{13} = ((\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - \cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ +\sin(\theta_1)\sin(\theta_4))\sin(\theta_5) - (\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \tag{6.24}$$

$$Q_{21} = \begin{pmatrix} \begin{pmatrix} \begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) \end{pmatrix}\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4) \end{pmatrix}\cos(\theta_5) \\ +\begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \end{pmatrix}\sin(\theta_5) \end{pmatrix}\cos(\theta_6) \\ +\begin{pmatrix} \begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) \end{pmatrix}\sin(\theta_4) \\ +\cos(\theta_1)\cos(\theta_4) \end{pmatrix}\sin(\theta_6) \tag{6.25}$$

$$Q_{22} = -\begin{pmatrix} \begin{pmatrix} \begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) \end{pmatrix}\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4) \end{pmatrix}\cos(\theta_5) \\ +\begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \end{pmatrix}\sin(\theta_5) \end{pmatrix}\sin(\theta_6) \\ +\begin{pmatrix} \begin{pmatrix} \sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) \end{pmatrix}\sin(\theta_4) \\ +\cos(\theta_1)\cos(\theta_4) \end{pmatrix}\cos(\theta_6) \tag{6.26}$$

$$Q_{23} = ((\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4))\sin(\theta_5) - (\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \tag{6.27}$$

$$Q_{31} = \begin{pmatrix} \left(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3)\right)\cos(\theta_4)\cos(\theta_5) \\ +(\sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3))\sin(\theta_5) \end{pmatrix}\cos(\theta_6) \\ +(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\sin(\theta_4)\sin(\theta_6) \tag{6.28}$$

$$Q_{32} = -\begin{pmatrix} \left(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3)\right)\cos(\theta_4)\cos(\theta_5) \\ +(\sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3))\sin(\theta_5) \end{pmatrix}\sin(\theta_6) \\ +(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\sin(\theta_4)\cos(\theta_6) \tag{6.29}$$

$$Q_{33} = \left(\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3)\right)\cos(\theta_4)\sin(\theta_5) \\ -(\sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3))\cos(\theta_5) \tag{6.30}$$

The forward kinematics equations can be used to find the Jacobian matrix of a manipulator, which is used to map the joint velocities to both the end-effector linear and angular velocities. This matrix is also important in the development of control laws for smooth path planning, for deriving the dynamics equations of a manipulator, and for transforming forces and torques from the end-effector to the joints [98]. The dexterity of a manipulator and manipulator singularities can also be determined with the Jacobian matrix.

The following equation is used in relating the end-effector velocities to the joint rates.

$$\dot{X} = J\dot{q} \tag{6.31}$$

where $\dot{X}$ is the 6x1 column vector of the end-effector velocities (linear and angular), J is the 6x6 Jacobian matrix, and $\dot{q}$ is the 6x1 column vector of the joint velocities.

The Jacobian matrix can be further broken down into two (2) components, a linear and angular component, as seen in Equation 6.32.

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix} \tag{6.32}$$

J_v is the linear component and relates the linear end-effector velocities to the joint velocities. The form for this matrix is the following.

$$J_v = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \cdots & \frac{\partial x}{\partial q_n} \\ \frac{\partial y}{\partial q_1} & \ddots & \frac{\partial y}{\partial q_n} \\ \frac{\partial z}{\partial q_1} & \cdots & \frac{\partial z}{\partial q_n} \end{bmatrix} \tag{6.33}$$

where x, y, z are the end-effector positions determined from the forward kinematics problem, and n is the DOF of the robot.

The angular component of the Jacobian, J_ω, relates the angular end-effector velocities to the robot's joint velocities. To find each component of the angular Jacobian, the spin axis of each joint relative to the base frame is needed. The following is the form of the angular Jacobian.

$$J_\omega = \begin{bmatrix} \hat{\omega}_1^{base} & \cdots & \hat{\omega}_n^{base} \end{bmatrix} \tag{6.34}$$

where $\hat{\omega}_i^{base}$ is a 3x1 column vector. To find this column vector for each joint, a transformation from the base to each respective joint needs to be completed, then the spin axis component is extracted [96]. From Figure 6.7, the DH parameters are set up in such a way to ensure each joint revolves around its respective z-axis.

Due to the size of the parameters inside of the Jacobian, each of the 18 components of the linear half Jacobian is listed separately below from Equation 6.35 to Equation 6.52.

$$\begin{aligned}
J_{v11} = J_{11} &= -75 sin(\theta_1) - 840 sin(\theta_1)\cos(\theta_2) \\
&-215 sin(\theta_1)\cos(\theta_2)\cos(\theta_3) + 215\cos(\theta_1)\sin(\theta_2) sin(\theta_3) \\
&-890 sin(\theta_1) cos(\theta_2) sin(\theta_3) - 890 sin(\theta_1) sin(\theta_2)\cos(\theta_3) \\
&+90\left(\left(\left(\begin{matrix}-\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\+\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\cos(\theta_4) \atop +\cos(\theta_1)\sin(\theta_4)\right)\cos(\theta_5) \atop +\left(\begin{matrix}-\sin(\theta_1)\cos(\theta_2) sin(\theta_3)\\-\sin(\theta_1) sin(\theta_2) cos(\theta_3)\end{matrix}\right) sin(\theta_5)\right)\cos(\theta_6) \\
&+90\left(\left(\begin{matrix}-\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\+\sin(\theta_1) sin(\theta_2) sin(\theta_3)\end{matrix}\right)\sin(\theta_4) \atop -\cos(\theta_1)\cos(\theta_4)\right)\sin(\theta_6)
\end{aligned} \tag{6.35}$$

$$\begin{aligned}
J_{v12} = J_{12} &= -840\cos(\theta_1) sin(\theta_2) - 215\cos(\theta_1) sin(\theta_2)\cos(\theta_3) \\
&-215\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) - 890\cos(\theta_1)\sin(\theta_2)\sin(\theta_3) \\
&+890\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) \\
&+90\left(\left(\begin{matrix}-\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\\-\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)\end{matrix}\right)\cos(\theta_4)\cos(\theta_5) \atop +\left(\begin{matrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\-\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\sin(\theta_5)\right)\cos(\theta_6) \\
&+90\left(\begin{matrix}-\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\\-\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)\end{matrix}\right)\sin(\theta_4)\sin(\theta_6)
\end{aligned} \tag{6.36}$$

$$J_{v13} = J_{13} = -215\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) - 215\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)$$
$$+890\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - 890\cos(\theta_1)\sin(\theta_2)\,sin(\theta_3)$$
$$+90\begin{pmatrix}\begin{pmatrix}-\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\cos(\theta_5)\\ +\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_5)\end{pmatrix}\cos(\theta_6) \tag{6.37}$$
$$+90\begin{pmatrix}-\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4)\sin(\theta_6)$$

$$J_{v14} = J_{14} = 90\begin{pmatrix}-\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4)\\ +\sin(\theta_1)\cos(\theta_4)\end{pmatrix}\cos(\theta_5)\cos(\theta_6)$$
$$+90\begin{pmatrix}\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\\ +\sin(\theta_1)\sin(\theta_4)\end{pmatrix}\sin(\theta_6) \tag{6.38}$$

$$J_{v15} = J_{15} = 90(-\begin{pmatrix}\begin{pmatrix}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\\ +\sin(\theta_1)\sin(\theta_4)\end{pmatrix}\sin(\theta_5)$$
$$+(\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \tag{6.39}$$
$$+\cos(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5))\cos(\theta_6)$$

$$J_{v16} = J_{16} = -90\left(\left(\left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\cos(\theta_4) \atop +\sin(\theta_1)\sin(\theta_4)\right)\cos(\theta_5) + \left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\sin(\theta_3)\\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{array}\right)\sin(\theta_5)\right)\sin(\theta_6)$$

$$+90(\left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\sin(\theta_4) \qquad (6.40)$$

$$-\sin(\theta_1)\cos(\theta_4))\cos(\theta_6)$$

$$J_{v11} = J_{11} = 75\cos(\theta_1) + 840\cos(\theta_1)\cos(\theta_2)$$
$$+215\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - 215\cos(\theta_1)\sin(\theta_2)\mathit{sin}(\theta_3)$$
$$+890\cos(\theta_1)\mathit{cos}(\theta_2)\mathit{sin}(\theta_3) + 890\cos(\theta_1)\mathit{sin}(\theta_2)\cos(\theta_3)$$
$$+90\left(\left(\left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\cos(\theta_4) \atop +\sin(\theta_1)\sin(\theta_4)\right)\cos(\theta_5) + \left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\mathit{sin}(\theta_3)\\ \cos(\theta_1)\mathit{sin}(\theta_2)\mathit{cos}(\theta_3)\end{array}\right)\mathit{sin}(\theta_5)\right)\cos(\theta_6)$$
$$+90\left(\left(\begin{array}{l}\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)\\ -\cos(\theta_1)\mathit{sin}(\theta_2)\mathit{sin}(\theta_3)\end{array}\right)\sin(\theta_4) \atop -\sin(\theta_1)\cos(\theta_4)\right)\sin(\theta_6) \qquad (6.41)$$

$$J_{v22} = J_{22} = -840\sin(\theta_1)\,sin(\theta_2) - 215\sin(\theta_1)\,sin(\theta_2)\cos(\theta_3)$$
$$-215\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) - 890\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)$$
$$+890\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)$$
$$+90\left(\begin{array}{l}\left(\begin{array}{l}-\sin(\theta_1)\cos(\theta_2)\sin(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{array}\right)\cos(\theta_4)\cos(\theta_5)\\+\left(\begin{array}{l}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\sin(\theta_5)\end{array}\right)\cos(\theta_6)$$
$$+90\left(\begin{array}{l}-\sin(\theta_1)\cos(\theta_2)\sin(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{array}\right)\sin(\theta_4)\sin(\theta_6) \tag{6.42}$$

$$J_{v23} = J_{23} = -215\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) - 215\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)$$
$$+890\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - 890\sin(\theta_1)\sin(\theta_2)\,sin(\theta_3)$$
$$+90\left(\begin{array}{l}\left(\begin{array}{l}-\sin(\theta_1)\cos(\theta_2)\sin(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{array}\right)\cos(\theta_4)\cos(\theta_5)\\+\left(\begin{array}{l}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\sin(\theta_5)\end{array}\right)\cos(\theta_6) \tag{6.43}$$
$$+90\left(\begin{array}{l}-\sin(\theta_1)\cos(\theta_2)\sin(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{array}\right)\sin(\theta_4)\sin(\theta_6)$$

$$J_{v24} = J_{24} = 90\left(\begin{array}{l}-\left(\begin{array}{l}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\sin(\theta_4)\\-\cos(\theta_1)\cos(\theta_4)\end{array}\right)\cos(\theta_5)\cos(\theta_6)$$
$$+90\left(\begin{array}{l}\left(\begin{array}{l}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)\\-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{array}\right)\cos(\theta_4)\\-\cos(\theta_1)\sin(\theta 4)\end{array}\right)\sin(\theta_6) \tag{6.44}$$

$$J_{v25} = J_{25} = 90(-\left(\left(\begin{matrix}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\cos(\theta_4) \atop -\cos(\theta_1)\sin(\theta_4)\right)\sin(\theta_5)$$
$$+(\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \tag{6.45}$$
$$+\sin(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5))\cos(\theta_6)$$

$$J_{v26} = J_{26} = -90\left(\begin{matrix}\left(\begin{matrix}\left(\begin{matrix}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4)\end{matrix}\right)\cos(\theta_5) \\ +\left(\begin{matrix}\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3)\end{matrix}\right)\sin(\theta_5)\end{matrix}\right)\sin(\theta_6)$$
$$+90(\left(\begin{matrix}\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\sin(\theta_4) \tag{6.46}$$
$$+\cos(\theta_1)\cos(\theta_4))\cos(\theta_6)$$

$$J_{v31} = J_{31} = 0 \tag{6.47}$$

$$J_{v32} = J_{32} = 840\cos(\theta_2) + 215\cos(\theta_2)\cos(\theta_3)$$
$$-215\sin(\theta_2)\sin(\theta_3) + 890\cos(\theta_2)\sin(\theta_3)$$
$$+890\cos(\theta_2)\cos(\theta_3)$$
$$+90\left(\begin{matrix}\left(\begin{matrix}\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\cos(\theta_4)\cos(\theta_5) \\ +\left(\begin{matrix}\sin(\theta_2)\cos(\theta_3) \\ +\cos(\theta_2)\sin(\theta_3)\end{matrix}\right)\sin(\theta_5)\end{matrix}\right)\cos(\theta_6)$$
$$+90\left(\begin{matrix}\cos(\theta_2)\cos(\theta_3) \\ -\sin(\theta_2)\sin(\theta_3)\end{matrix}\right)\sin(\theta_4)\sin(\theta_6) \tag{6.48}$$

$$J_{v33} = J_{33} = -215\sin(\theta_2)\sin(\theta_3) + 215\cos(\theta_2)\cos(\theta_3)$$
$$+890\sin(\theta_2)\cos(\theta_3) + 890\cos(\theta_2)sin(\theta_3)$$
$$+90\left(\begin{pmatrix}\cos(\theta_2)\cos(\theta_3)\\ -\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\cos(\theta_5) + \begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_5)\right)\cos(\theta_6) \tag{6.49}$$
$$+90\begin{pmatrix}\cos(\theta_2)\cos(\theta_3)\\ -\sin(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4)\sin(\theta_6)$$

$$J_{v34} = J_{34} = -90\left(\begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4)\right)\cos(\theta_5)\cos(\theta_6)$$
$$+90\left(\begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\right)\sin(\theta_6) \tag{6.50}$$

$$J_{v35} = J_{35} = 90(-\begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\sin(\theta_5)$$
$$+(\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3))\cos(\theta_5))\cos(\theta_6) \tag{6.51}$$

$$J_{v36} = J_{36} = -90\left(\left(\begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\cos(\theta_4)\right)\cos(\theta_5) + \begin{pmatrix}\sin(\theta_2)\sin(\theta_3)\\ -\cos(\theta_2)\cos(\theta_3)\end{pmatrix}\sin(\theta_5)\right)\sin(\theta_6) \tag{6.52}$$
$$+90\begin{pmatrix}\sin(\theta_2)\cos(\theta_3)\\ +\cos(\theta_2)\sin(\theta_3)\end{pmatrix}\sin(\theta_4)\cos(\theta_6)$$

Equations 6.53 to 6.58 are the components consisting of the angular half of the Jacobian matrix.

$$J_{\omega_1} = \hat{\omega}_1^{base} = \begin{Bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{Bmatrix} \tag{6.53}$$

$$J_{\omega_2} = \hat{\omega}_2^{base} = \begin{Bmatrix} \sin(\theta_1) \\ -\cos(\theta_1) \\ 0 \end{Bmatrix} \tag{6.54}$$

$$J_{\omega_3} = \hat{\omega}_3^{base} = \begin{Bmatrix} \cos(\theta_1)\cos(\theta_2)\sin(\theta_3)+\cos(\theta_1)\sin(\theta_2)\cos(\theta_3) \\ \sin(\theta_1)\cos(\theta_2)\sin(\theta_3)+\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) \\ \sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3) \end{Bmatrix} \tag{6.55}$$

$$J_{\omega_4} = \hat{\omega}_4^{base} = \begin{Bmatrix} (\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)-\cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\sin(\theta_4) \\ -\sin(\theta_1)\cos(\theta_4) \\ (\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\sin(\theta_4) \\ +\cos(\theta_1)\cos(\theta_4) \\ (\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\sin(\theta_4) \end{Bmatrix} \tag{6.56}$$

$$J_{\omega_5} = \hat{\omega}_5^{base} = \begin{Bmatrix} ((\cos(\theta_1)\cos(\theta_2)\cos(\theta_3)-\cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ +\sin(\theta_1)\sin(\theta_4))\sin(\theta_5)-(\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \\ ((\sin(\theta_1)\cos(\theta_2)\cos(\theta_3)-\sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4))\sin(\theta_5)-(\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \\ (\sin(\theta_2)\cos(\theta_3)+\cos(\theta_2)\sin(\theta_3))\cos(\theta_4)\sin(\theta_5) \\ -(\sin(\theta_2)\sin(\theta_3)-\cos(\theta_2)\cos(\theta_3))\cos(\theta_5) \end{Bmatrix} \tag{6.57}$$

$$J_{\omega_6} = \hat{\omega}_6^{base} = \begin{Bmatrix} ((\cos(\theta_1)\cos(\theta_2)\cos(\theta_3) - \cos(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ +\sin(\theta_1)\sin(\theta_4))\sin(\theta_5) - (\cos(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\cos(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \\ ((\sin(\theta_1)\cos(\theta_2)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\cos(\theta_4) \\ -\cos(\theta_1)\sin(\theta_4))\sin(\theta_5) - (\sin(\theta_1)\cos(\theta_2)\sin(\theta_3) \\ +\sin(\theta_1)\sin(\theta_2)\cos(\theta_3))\cos(\theta_5) \\ (\sin(\theta_2)\cos(\theta_3) + \cos(\theta_2)\sin(\theta_3))\cos(\theta_4)\sin(\theta_5) \\ -(\sin(\theta_2)\sin(\theta_3) - \cos(\theta_2)\cos(\theta_3))\cos(\theta_5) \end{Bmatrix} \quad (6.58)$$

6.5 Control

The main control technique used was combining visual servo and linear control. Visual servo uses computer vision data to drive a robot to a place and/or orientation by combining image processing, computer vision, and control theory. There are two (2) visual servo methods, both aiming to minimize the error between the current position of a camera sensor and the desired position of the camera. Equation 6.59 is the error term being minimized with visual servo control

$$e(t) = s(m(t), a) - s^* \quad (6.59)$$

where m(t) is the set of image measurements used to determine a vector of *k* visual features s(m(t), a), a is the set of parameters representing the system, and s* is the desired values of visual features [99].

The first visual servo method is Image-based Visual Servo Control (IBVS) where the visual features vector, s, consists of a set of features available in a 2D image. Position-based Visual Servo control (PBVS) consists of the feature vector, s, being a vector of 3D parameters estimated from image measurements [99].

IBVS uses image coordinates to define the visual features vector s(m, a), with m being the pixel coordinates of a feature, and a being the intrinsic parameters of the camera. To control a 6-DOF robot, three (3) points are needed. With a 3D point $\bar{x} = (x, y, Z)$ in the camera world frame, Equations 6.60 and 6.61 translate this point to the world frame

$$X = \frac{u - c_u}{f\alpha} \tag{6.60}$$

$$Y = \frac{v - c_v}{f} \tag{6.61}$$

where m = (u,v), a = (c_u, c_v, f, α) and α is the ratio of pixel dimensions. The velocity of the 3D point can be related to the camera spatial velocity as seen in Equation 6.62 and the set of equations from Equation 6.63 to Equation 6.65.

$$\dot{\mathbf{X}} = -\mathbf{v_c} - \omega_c \times \mathbf{X} \tag{6.62}$$

$$\dot{X} = -v_x - \omega_y Z + \omega_z Y \tag{6.63}$$

$$\dot{Y} = -v_y - \omega_z X + \omega_x Z \tag{6.64}$$

$$\dot{Z} = -v_z - \omega_x Y + \omega_y X \tag{6.65}$$

Using Equation 6.66, the camera's velocity can be controlled with L_x is the interaction matrix as seen in Equation 6.67. To create a 6x6 interaction matrix, three (3) points are needed as mentioned earlier [99].

$$\dot{\mathbf{x}} = \mathbf{L}_{\mathbf{x}} \mathbf{v}_{\mathbf{c}} \tag{6.66}$$

$$\mathbf{L}_{\mathbf{x}} = \begin{bmatrix} -\frac{1}{Z} & 0 & \frac{x}{Z} & xy & -(1+x^2) & y \\ 0 & -\frac{1}{Z} & \frac{y}{Z} & 1+y^2 & -xy & -x \end{bmatrix} \tag{6.67}$$

PBVS uses the camera’s pose with respect to a reference coordinate system to define a visual features vector. For this, the camera’s intrinsic matrix is needed along with a 3D object model of the target [99].

There are four (4) different robot control types: (1) point-to-point control, (2) continuous-path control, (3) controlled-path control, and (4) stop-to-stop control. Point-to-point control requires locations for the robot to move towards to be loaded into memory as the robot is only able to move to and from specific points. Continuous-path control allows for a robot to move along a prescribed path continuously and all points along this path must be stored in robot memory. Controlled-path control allows the robot to take any path to get from a starting point to the end point. Stop-to-stop control is open looped control where the position and velocity of the robot is not known to the controller [100].

The control technique used to control the FANUC M20iD/25 robot to dock with a target combined IBVS, linear control and controlled-path control. The controllers were implemented kinematically, rather than dynamically. Only the joint values were driven by controllers, rather than torques. Kinematic modeling studies robot motion without considering forces and torques acting on the robot and causing motion as the end effector pose is dependent on just the joint values. Dynamic modeling uses a relationship between applied torques and forces and the movement of a robot [101].

To control the joints, Proportional-Integral (PI) controllers were used. An image of a typical PI controller can be seen in Figure 6.8.

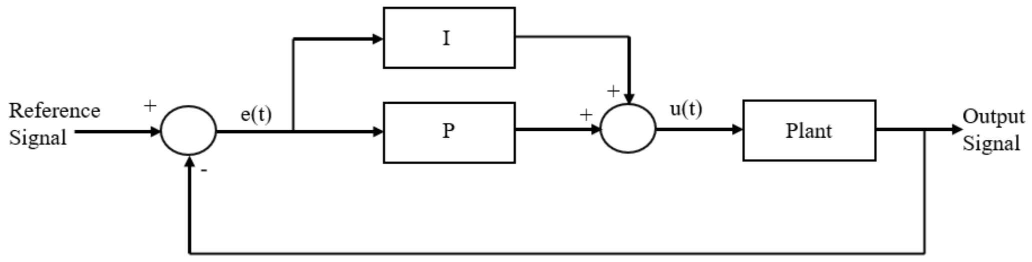


Figure 6.8: The general form of a PI controller.

PI controllers are feedback controllers which use a proportional term, P, and an integral term, I, to reduce the error of a system. The proportional term outputs the proportional error, which can be adjusted with a proportional gain term. The integral term uses both the magnitude of the error and considers the time over which the error takes place. This term tends to improve the steady-state error which is present in just a proportional (P) controller [102]. The proportional and integral terms can be seen in Equations 6.68 and 6.69.

$$P = k_p e(t) \tag{6.68}$$

$$I = K_I \int e(t)dt \tag{6.69}$$

The PI controllers used provided desired responses. These responses, as seen in Figure 6.18 to Figure 6.23 had fast rise times, did not oscillate about the steady state, and were able to stabilize about the steady state. As seen in Figure 6.9 below, responses from controllers that are overdamped are not able to reach the desired position which is not desirable for a robotic arm. With an underdamped system, the oscillations can cause vibrations in the manipulator and can fluctuate past the joint limits. A response similar to critically damped is desired.

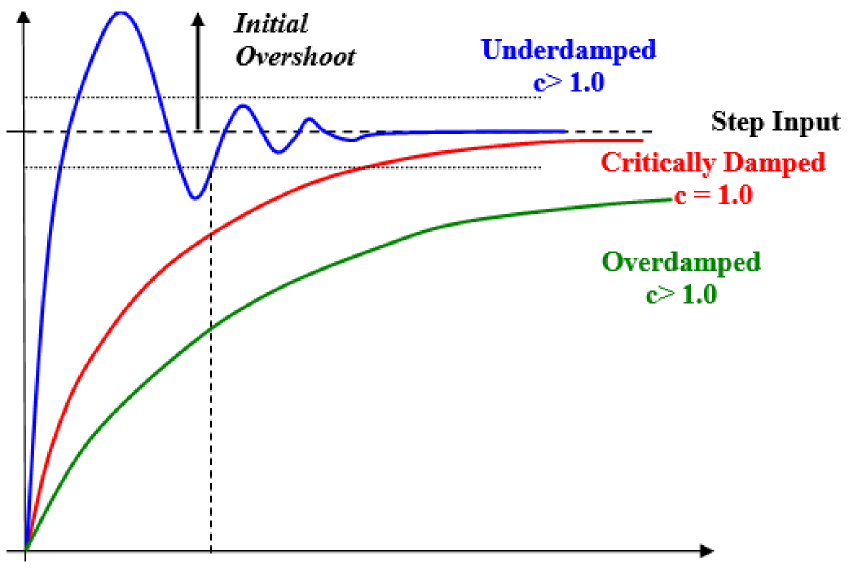


Figure 6.9: The various types of responses from a controller output [103].

The controllers were implemented in MATLAB Simulink, where the software simulated the movement and responses of the robot joints. These joint values were sent as commands to ROS Gazebo, where a Kinect camera was acting as the end effector and sending image data to Simulink for computer vision processing. Figure 6.10 is the method of operations of the algorithm. The software implementation is discussed further in Chapter 6.6.

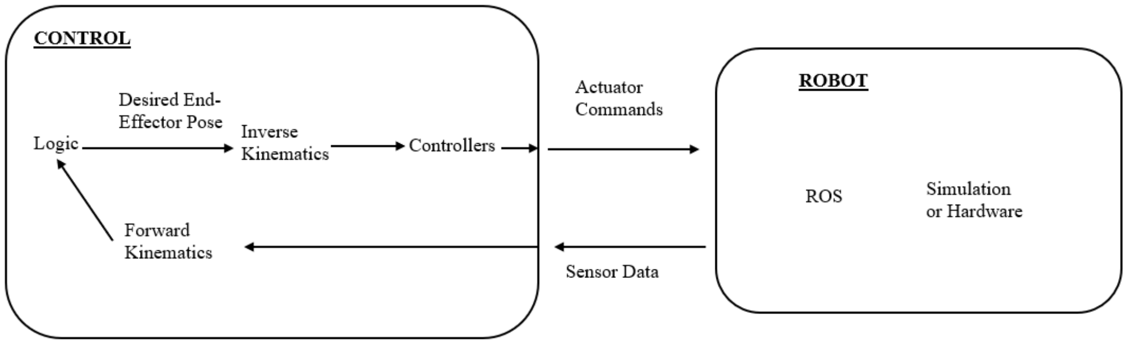


Figure 6.10: Method of operations of the developed simulation algorithm.

6.6 Simulink Environment Setup

To control the robot, MATLAB Simulink was used in tandem with ROS Gazebo. Figures 6.11, 6.12, and 6.13 show the developed Simulink program. In Figure 6.11, two (2) ROS subscriber blocks were used. ROS uses messages to communicate among nodes on a network, a subscriber receives data on the network. The two subscribers were subscribed to RGB image data and depth image data from ROS, respectively. In ROS Gazebo, a virtual camera (Kinect camera) was set up to take images of a target while the simulation ran as seen in Figure 3.8. These images were inputted into Simulink. The raw RGB and depth data was forwarded to the respective *Read Colour Image* and R*ead Depth Image* blocks. These blocks were predefined blocks available in Simulink for converting raw image data from ROS to viewable images used for computer vision. The *getCoordinates* block was a user-defined function that reads in the RGB image and depth image to retrieve the target's world coordinates in the camera world frame and the pose of the target in the camera world frame. The world coordinates were then transformed to the base frame of the robot and combined with the target pose relative to the base frame as a 4x4 transform matrix. The transform matrix states where the target object is oriented in 6-dimensions relative to the base of the robot.

In Figure 6.12, the transform matrix was inputted into the *Inverse Kinematics* block, which determined the joint values needed to position and orient the end-effector at the target's location. The joint states were first sent to a controller block, which was user-defined and contained the six (6) PI controllers used to control each of the six (6) joints. The outputs from the controller block were inputted into the *Fanuc Robot* block, which contained the robot's joints, links, actuators, and the environment the robot was functioning in. The PI controller block used negative feedback, therefore the joint values from the *Fanuc Robot* block were inputted into the *PI Joint Controller* block. Figures 6.14 and 6.15 show *PI Joint Controller* and *Fanuc Robot* blocks in expanded detail.

In Figure 6.13, the joint values from the *Fanuc Robot* block were inputted as a vector into the *Forward Kinematics1* block. This was also a predefined block that outputted the end-effector position of a robot when joint values were inputted. The end-effector position and the XYZ pose output were inputted into a user-defined block called *sendMessages*. This function block was created to organize the 6DOF end effector pose to allow for easy publishing to the ROS network. The outputs from this block were all the values being sent to the ROS network to move the camera to the desired position in the ROS Gazebo simulation. The *Publish kinects pose1* block was the publisher block, the counterpart to the subscriber blocks seen in Figure 6. Rather than receiving data from nodes, this block sent data to other nodes across the ROS network.

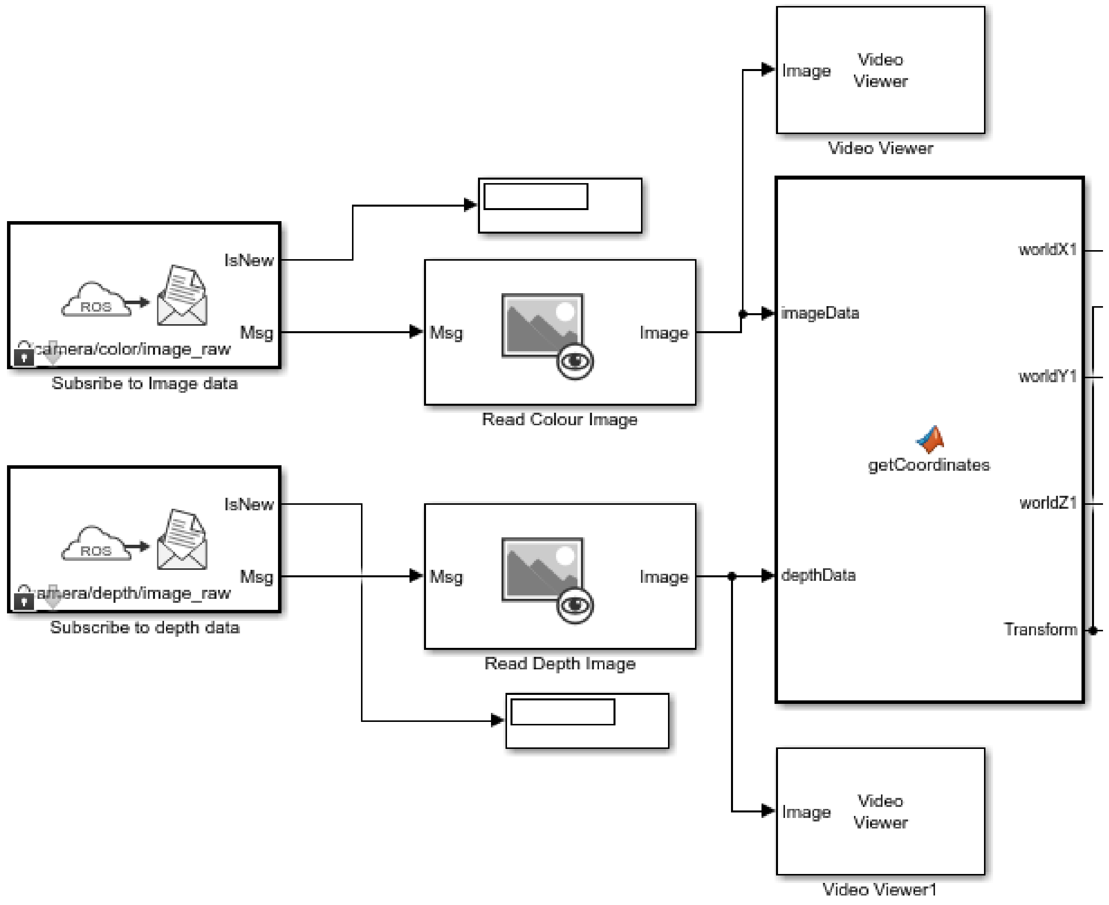


Figure 6.11: The first part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot.

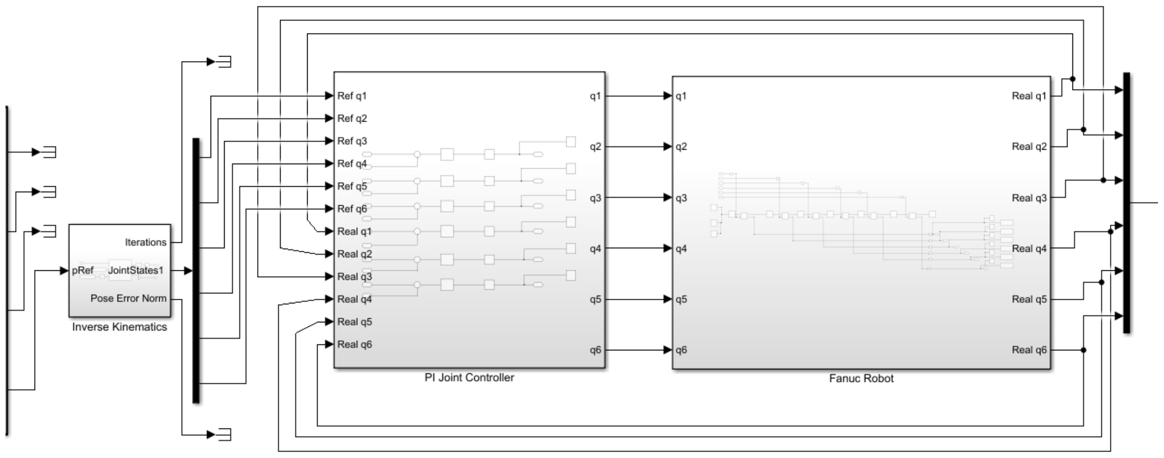


Figure 6.12: The second part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot.

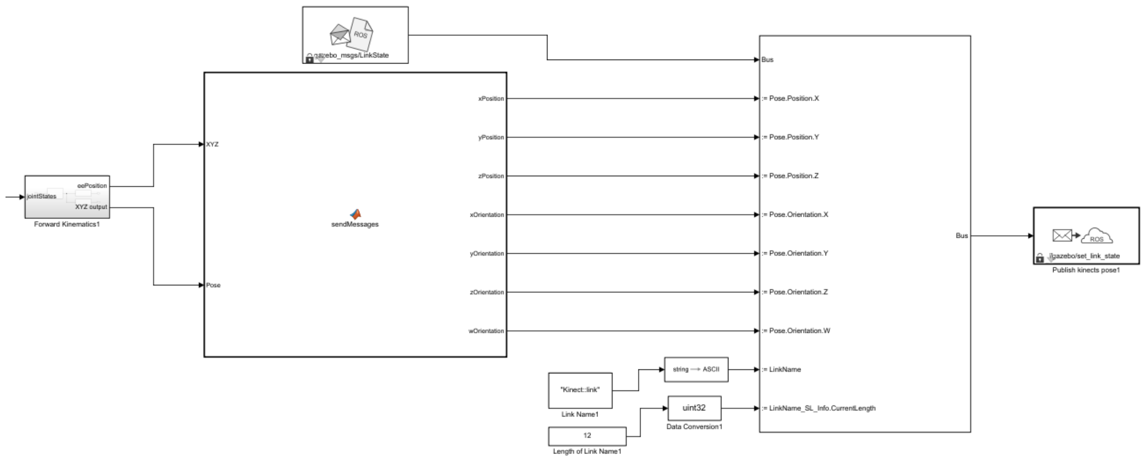


Figure 6.13: The third part of the Simulink block diagram to communicate with ROS Gazebo and to control the robot.

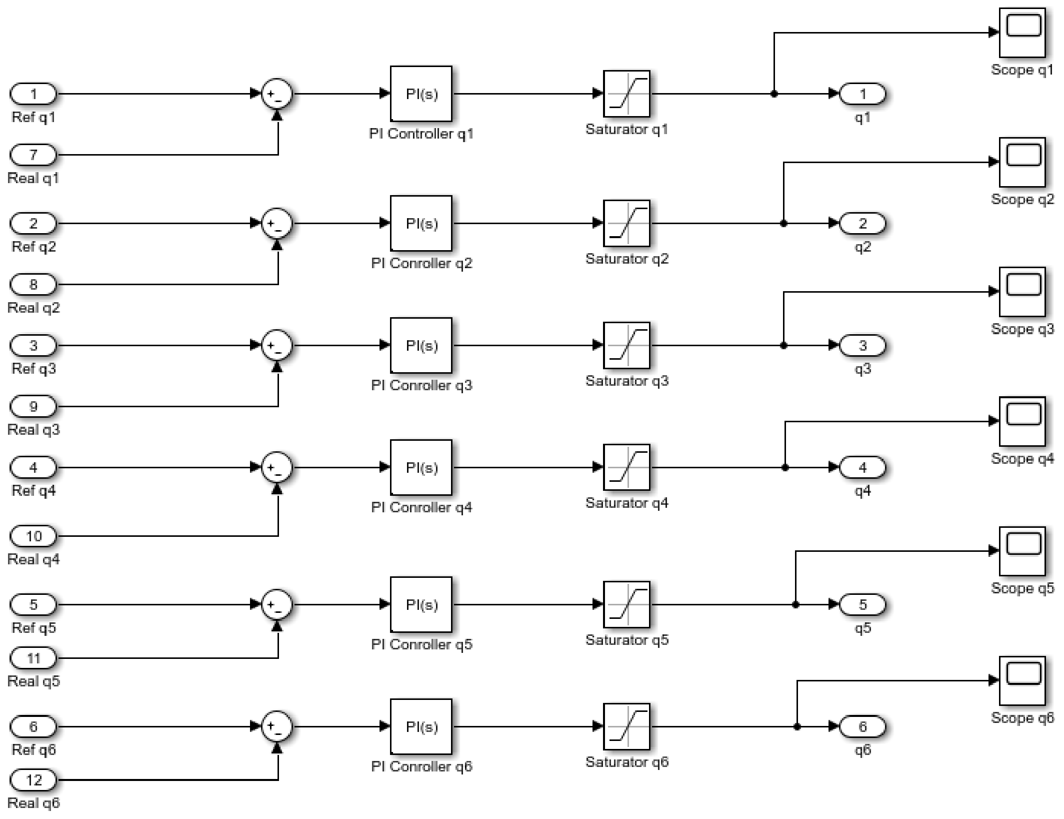


Figure 6.14: The PI Joint Controller block in more detail.

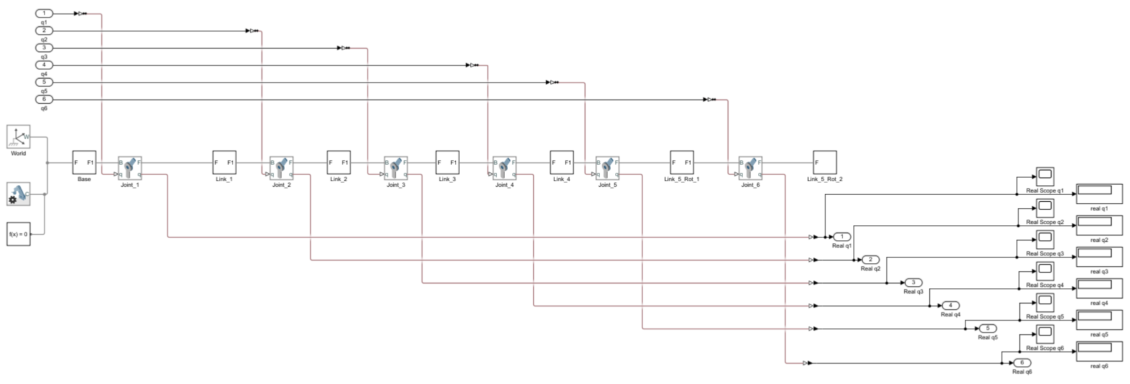


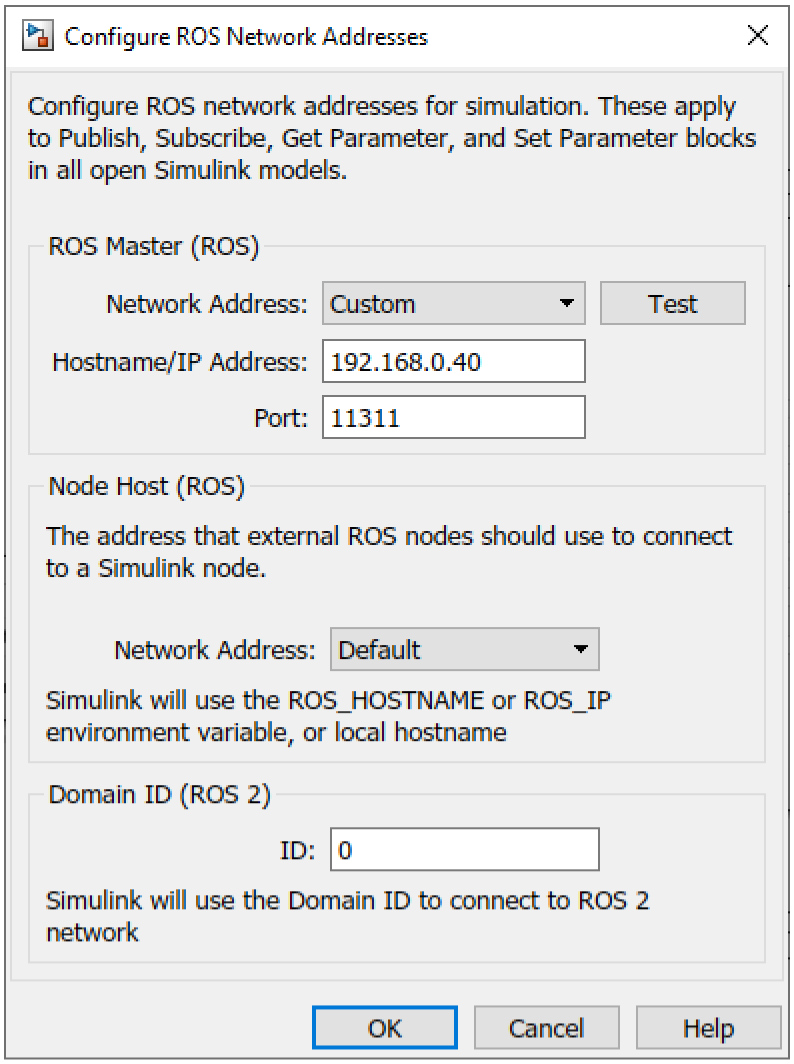
Figure 6.15: The Fanuc Robot block expanded in more detail.

Figure 6.14 shows the *PI Joint Controller* block in detail. Included are the inputs from the

Inverse Kinematics block and the feedback from the *Fanuc Robot* block. Each of the controllers were tuned using the Tuner App provided with MATLAB. A saturator was used for each joint to limit the joint values to the joint limits stated in Table 6.2.

Figure 6.15 shows the *Fanuc Robot* block in detail. The joint values from the *PI Joint Controller* block were inputted to the respective joints. The *Mechanism Configuration* block on the left contains the gravity vector the robot functions with. Gravitational acceleration was set to 0 in all three dimensions as for a Free-Flying robot, gravity is not considered. Each *joint* block contained the joint limits, joint velocity limits, and origins. Each *link* block contained the STL files of the respective link to visualize the robot in simulation. The joint transforms were also included between the two joints on either end of the link in each *link* block. The mass moment of inertias of each link were also included for when a dynamic simulation is used.

To connect with ROS, the ROS environment was started on a virtual machine on the same computer as the Simulink program. When running, the ROS Master node had its own IP address and port number. The Simulink simulation was connected to this IP address using the *Configure ROS Network Addresses* function provided with the ROS Toolbox in Simulink as seen in Figure 6.16.


Configure ROS Network Addresses

Configure ROS network addresses for simulation. These apply to Publish, Subscribe, Get Parameter, and Set Parameter blocks in all open Simulink models.

ROS Master (ROS)

Network Address: Custom

Hostname/IP Address: 192.168.0.40

Port: 11311

Node Host (ROS)

The address that external ROS nodes should use to connect to a Simulink node.

Network Address: Default

Simulink will use the ROS_HOSTNAME or ROS_IP environment variable, or local hostname

Domain ID (ROS 2)

ID: 0

Simulink will use the Domain ID to connect to ROS 2 network

Figure 6.16: The Configure ROS Network Address function in detail.

Once the Simulink environment was connected to the ROS Master node, the subscriber and publisher blocks could be configured to act as nodes over the ROS network and could receive or send messages, respectively.

6.7 Experimentations and Results

Figure 3.8 shows the Kinect camera in the ROS Gazebo environment which takes images and sends them to Simulink over the ROS network. The experiment that was run for the robotics simulation consisted of having the target placed away from the camera, the camera taking an RGB image and depth image of the target, and then the images being sent to Simulink. In Simulink, computer vision was used to retrieve the targets 6DOF pose, which was transformed into a 4x4 matrix. Inverse kinematics calculations were done on the 4x4 transform and the derived joint values were sent to the joint controllers. The goal of the joint controllers was to get a smooth but fast response in getting the robot’s end-effector from its home position to the target for docking. Based on the robot’s movement, the goal was to move the camera in the ROS Gazebo environment as if the robot’s end-effector had moved. Due to hardware constraints, only the Kinect camera was placed in ROS Gazebo as using the full Fanuc robot was computationally expensive with a virtual machine.

As seen in Figure 3.8, the camera was able to dock with the target as if the end-effector of the robot had docked. The plots seen in Figures 6.18 to 6.23 are the responses of each of the joints outputted from the controllers. All the controllers provided a smooth response to its end goal, which is the ideal response. Figure 6.17 shows the images taken by the Kinect camera and Figure 3.8 shows the cameras position relative to the target.

Figure 6.17: The images taken by the Kinect Camera inside ROS Gazebo.

Table 6.4 shows the coordinates of the target relative to the camera and the coordinate of the target in the robot's base frame. Table 6.5 shows the joint values obtained from the inverse kinematics block in Simulink.

Table 6.4: The coordinates of the target relative to the camera and robot base in the setup seen in Figure 3.8.

AXIS	CAMERA WORLD FRAME	ROBOT BASE FRAME
X	-0.01961	2.00
Y	0.002801	-0.0196
Z	1.035	1.463

To convert from the camera world frame to the robot base frame, the following equations were used.

$$X_{robot} = Z_{camera} + 0.965 \tag{6.70}$$

$$Y_{robot} = X_{camera} + 9.624x10^{-6} \tag{6.71}$$

$$Z_{robot} = Y_{camera} + 1.48 \tag{6.72}$$

Table 6.5: The joint values obtained from the inverse kinematics block in Simulink from the setup seen in Figure 3.8.

JOINT	VALUE [RAD]
1	-0.009806
2	0.7824
3	-0.79
4	-7.89 x 10^{-6}
5	-0.79
6	0.02725

Figures 6.18 to 6.23 are of the six (6) joint controller responses.

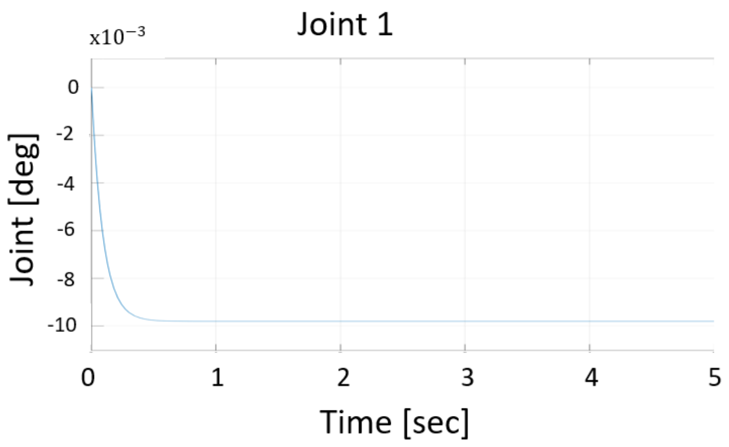


Figure 6.18: Joint controller response for joint 1.

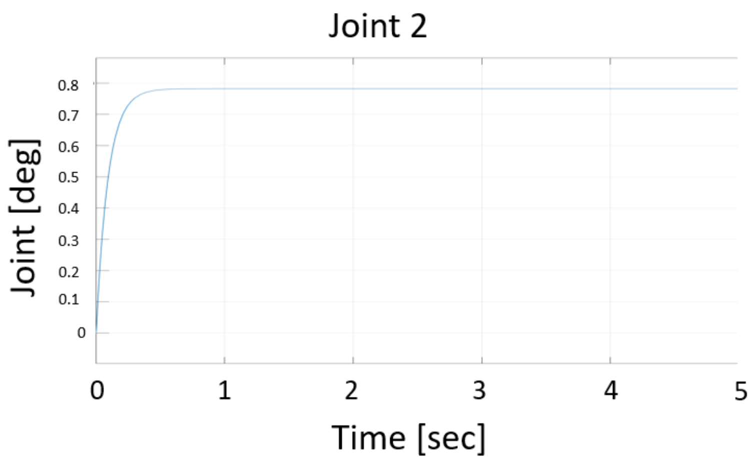


Figure 6.19: Joint controller response for joint 2.

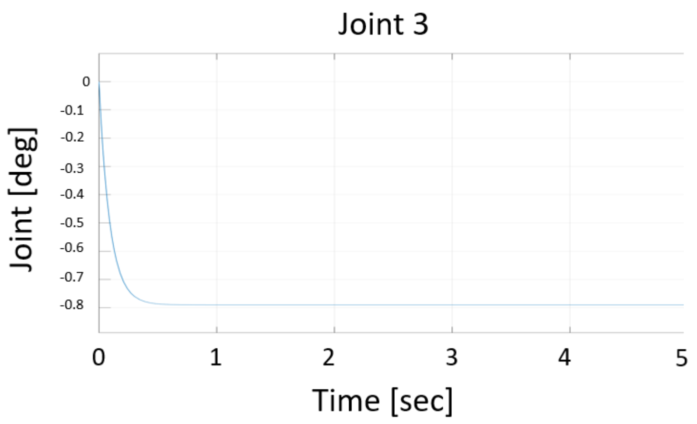


Figure 6.20: Joint controller response for joint 3.

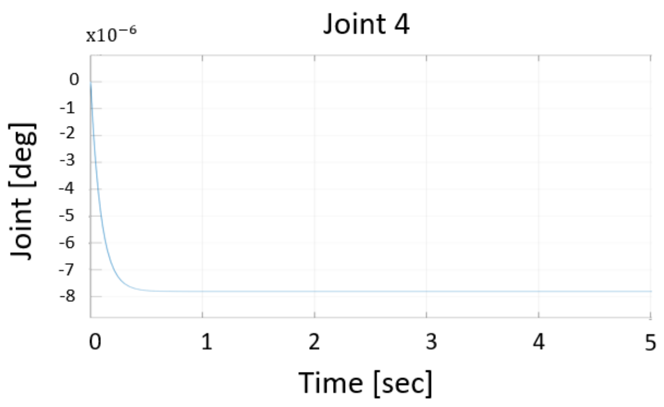


Figure 6.21: Joint controller response for joint 4.

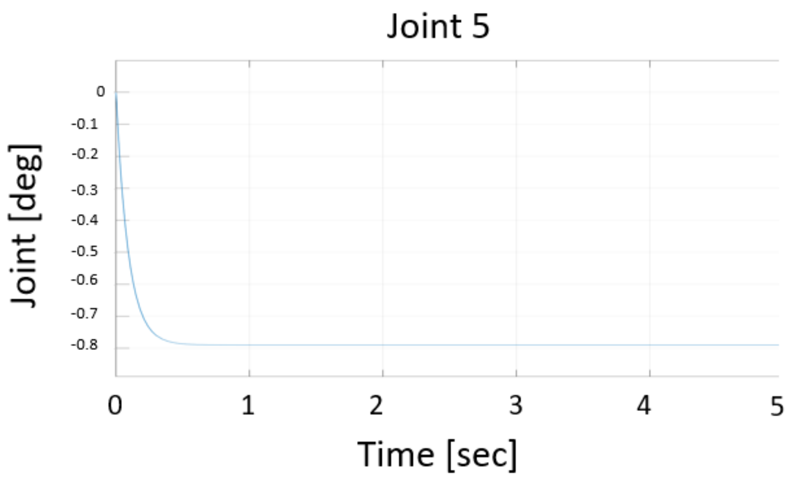


Figure 6.22: Joint controller response for joint 5.

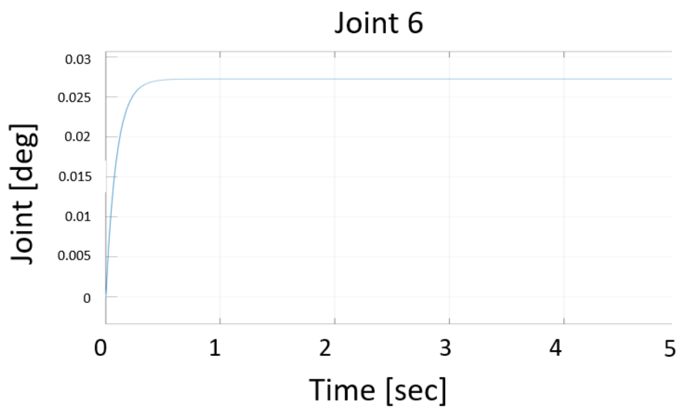


Figure 6.23: Joint controller response for joint 6.

In conclusion, a robotics rendezvous and docking simulation was successfully created with the integration of a 6DOF pose detection algorithm to accurately locate a target object. This exercise was a successful precursor to implementing robotic rendezvous and docking in a Hardware-in-the-Loop simulation.

Chapter 7: Conclusion and Future Work

Summary: This chapter consists of three sections. First, a summary of the overall results and general conclusions from this thesis are given. Second, the contributions from this thesis are provided. Finally, future work recommendations that can further improve the abilities to test space robotics rendezvous and docking are given.

7.1 General Conclusions

7.1.1 Thesis Accomplishments

7.1.1.1 Neural Network for Object Detection

The thesis was divided into two sections, one consisted of developing a computer vision algorithm for 6DOF pose detection, and the second consisting of developing visual servo control for robotic rendezvous and docking. The computer vision algorithm was developed using artificial intelligence, as the current approaches in the space environment use traditional computer vision techniques. A you-only-look-once network was used as the backbone for object detection due to its real-time object tracking capabilities. Using AI and a depth camera, this thesis work demonstrated that trained networks could be used to detect the position of an object in real time relative to the camera for real time tracking purposes.

7.1.1.2 Development of a Custom Neural Network for Pose Detection

To complete the 6DOF pose detection network, a custom 3D pose detection neural network was developed. This network was able to detect the orientation of an object in quaternion form relative to a camera sensor, which could then be converted to Euler Angles using an XYZ transformation. Both the object detection and pose detection neural networks were combined to create one single pipeline for input data. In this pipeline, the image is initially inputted into the YOLO network to localize the object and to obtain its 3D position coordinates. Then the localized object is inputted into the pose detection network to obtain the orientation. The thesis work validated the real time 6DOF pose detection capabilities of the custom computer vision algorithm.

7.1.1.3 Deployment of 6DOF Pose Detection Network on a Microcomputer

To test the capabilities of the 6DOF pose detection network on a space grade board, the algorithm was deployed on an NVIDIA Jetson Nano board. The pipeline was functional and provided the 6DOF pose of a target in real time on the board, although at low speeds. Therefore, recommendations for improvement were provided which including using multiple boards or deploying an updated YOLO network for faster performance.

7.1.1.4 Development of an Algorithm for Custom Virtual Data Generation

To compensate for the lack of available training data for a funnel and to have accurate ground truth data to assist in training an effective neural network for 6DOF pose detection, an algorithm was developed to generate custom virtual training data in Blender and Python. The object of interest was able to be placed in any orientation and with chosen lighting conditions. The thesis confirmed the ability to train an effective network with custom virtual data.

7.1.1.5 Validation of a Software-in-the-Loop Simulation for Robotic Rendezvous and Docking

Finally, the thesis work was wrapped up with the development of a Software-in-the-Loop simulation for robotic rendezvous and docking in Simulink and ROS Gazebo with the integration of visual servo. For visual servo, the 6DOF pose detection algorithm was used. It was shown that given the predictions of an object's 6DOF pose, a robot could be driven to dock with the object using kinematic joint control.

7.2 Contributions of Thesis Work

The thesis work developed a 6DOF pose detection algorithm using artificial intelligence. This work was motivated by the need to actively remove space debris, which is becoming a growing problem in the space environment, and by the expansion of the on-orbit servicing industry. Objects move and rotate quickly in the space environment, and traditional computer vision techniques struggle when not able to detect certain edges or certain colors. The space environment also has changing lighting conditions as objects move through their orbits. An artificially intelligent computer vision algorithm can be trained for these changes and track an object in real-time. By using a depth sensor and the intrinsic matrix of the depth camera, we can estimate the 3D location of a target object relative to the camera frame and after a transformation, relative to the base of a robot. Without the need for another camera sensor, we can estimate the orientation of the object relative to the camera frame and after a transformation, relative to the base of a robot. We can use this custom computer vision pipeline to create visual servo control laws to drive the joints of a robot to a desired location for rendezvous and docking, exhibited by the virtual simulation of the robot in Simulink and ROS Gazebo in Chapter 6.

This work was presented at the Canadian Society for Mechanical Engineering International Congress Conference in 2021. This work will contribute to the engineering design and scientific study of space robotic rendezvous and docking.

7.3 Future Work

From the contributions of this thesis, the following are considered for future studies:

7.3.1 Hardware Setup and Integration

Initially, the plan for this thesis was to program industrial robots in a Hardware-in-the-Loop simulation to test the capabilities of control techniques with artificially intelligent computer vision algorithms for space rendezvous and docking. Due to the COVID-19 pandemic, all in-person studies on campus at York University were halted, causing the initial plan to pivot. Due to the pandemic, the industrial robots were not installed at the Space Engineering Design Laboratory and the rendezvous and docking simulations were done virtually. A virtual simulation is useful for testing software before deployment on hardware and can mimic simulations meant to be taken place, but a virtual scenario tends to portray the optimal situation. The virtual robot controllers were also kinematic controllers, whereas actual robots will need dynamic controllers, allowing for better control of the manipulators. Setting up the industrial robots is important for integration of all subsystems working together for testing rendezvous and docking. The laboratory setup will also allow for changing the lighting conditions and setting them to mimic the space environment with the help of star trackers on the ceiling, black curtains placed for creating darkness, and with spotlights to mimic the Sun.

The integration of the computer vision algorithms developed as part of this thesis with the industrial robots is an important next step for testing rendezvous and docking too. The computer vision algorithm was only used in a virtual simulation, on virtual images, in set lighting conditions, and on models of a target with material conditions that are not perfect. There was minimal noise in the images tested on the computer vision algorithm alone and in the robot simulation. Using the computer vision algorithm in the Hardware-in-the-Loop simulation will truly test its capabilities in different lighting, in noisy images, and on objects with complicated reflective properties not mimicked in a virtual setting.

Figure 7.1 below is an example of the DLR’s OOS docking simulation setup, similar to what the plan is for the setup in the Space Engineering Design Laboratory at York University, but with an extra manipulator on the chaser satellite.

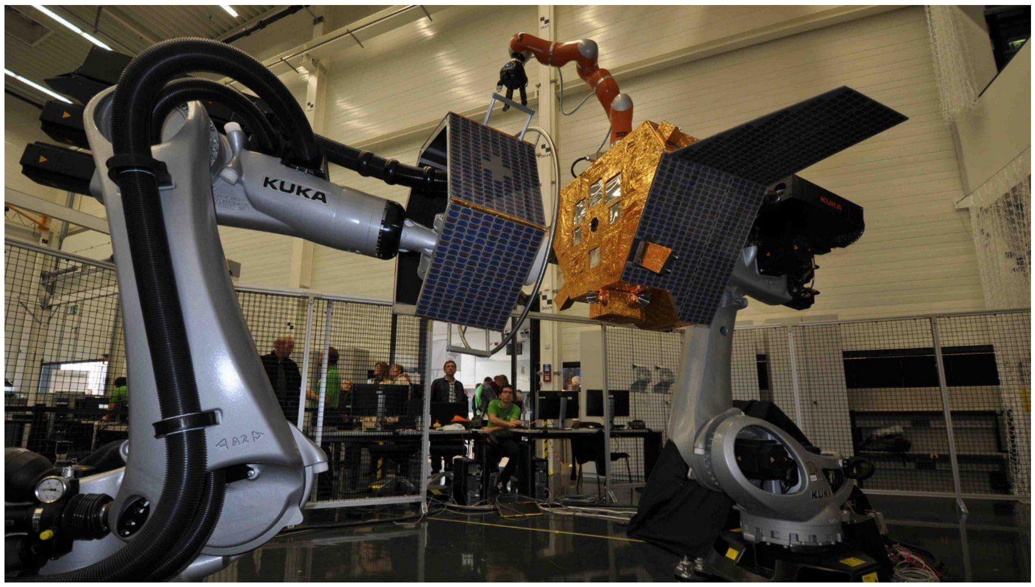


Figure 7.1: The DLR's OOS docking simulation setup, similar to what is being planned at the Space Engineering Design Laboratory at York University [104].

7.3.2 Re-Training the Object and Pose Detection Network

The current computer vision algorithm used in the virtual docking simulation was only trained to detect a funnel as a funnel was initially planned as the docking point on the mock satellite for the Hardware-in-the-Loop simulation. This was due to a funnel's similarity to a thruster nozzle of a satellite. In reality, different areas of a satellite can be used for docking depending on orientation, such as the solar panels, adapter rings, thruster nozzles, and corners. Re-training the computer vision algorithm to detect multiple objects and to detect various areas of the same object, rather than a general representation of the object, will improve the effectiveness of the algorithm and allow it to be more adaptable in different conditions.

7.3.3 Obstacle Avoidance

The current control scheme does not include obstacle avoidance, which is necessary to avoid docking with sensitive areas of a target satellite. Obstacle avoidance will also assist in avoiding collisions between the chaser satellite and the target satellite, the target colliding with different areas of the manipulator, i.e., different joints or links; and the manipulator colliding with the chaser satellite it is mounted on.

7.3.4 Improving Joint Controllers

In the current thesis, linear joint controllers were tested. In the future, non-linear controllers should be tested and studied to gauge their effectiveness at driving the torque inputs to each joint. To add to this, intelligent control techniques should be tested as well. Fuzzy logic can be tested in the future for joint control, which allows for logic to be partially true, rather than strictly true or false. Techniques such as reinforcement learning, a subset of machine learning, can be advantageous for path planning and determining the optimal path to the target for low energy consumption as well.

References

[1] A. Flores-Abad, O. Ma, K. Pham and S. UIrich, "A review of space robotics technologies for on-orbit servicing," *Science Direct,* vol. 68, pp. 1-26, 2014.

[2] T. Rybus, "Obstacle avoidance in space robotics: Review of major challenges and proposed solutions," *ScienceDirect,* vol. 101, pp. 31-48, 2018.

[3] P. Putz, "Space Robotics in Europe: A Survey," *Robotics and Autonomous Systems,* 1998.

[4] M. Shan, J. Guo and E. Gill, "Review and comparison of active space debris capturing and removal methods," *Progress in Aerospace Sciences,* 2016.

[5] I. Rekleitis, E. Martin, G. Rouleau, R. L'Archevêque, K. Parsa and E. Dupuis, "Autonomous capture of a tumbling satellite," *IEEE,* 2006.

[6] DLR, "European Proximity Operations Simulator (EPOS 2.0)," DLR. [Online].

[7] G. Hirzinger, "Robots in Space - A Survey," *Advanced Robotics,* 1994.

[8] S. Dubowsky and E. Papadopoulos, "The Kinematics, Dynamics, and Control of Free-Flying and Free-Floating Space Robotic Systems," *TRANSACTIONS ON ROBOTICS AND AUTOMATION,* 1993.

[9] C. Dossowski and L. Notash, "A survey of technical issues in space robotics," *CSME,* 2001.

[10] R. Lampariello and G. Hirzinger, "Generating Feasible Trajectories for Autonomous On-Orbit Grasping of Spinning Debris in a Useful Time," *International Conference on,* 2013.

[11] N. Lucas, A. Pandya and R. Ellis, "Review of multi-robot taxonomy, trends, and applications for defense and space," *SPIE,* 2012.

[12] T. Schildknecht, "Optical surveys for space debris," *The Astronomy and Astrophysics Review,* vol. 14, pp. 41-111, 2007.

[13] S.-I. Nishida, S. Kawamoto, Y. Okawa, F. Terui and S. Kitamura, "Space debris removal system using a small satellite," *Science Direct,* Vols. 1-2, pp. 95-102, 2009.

[14] L. Hall, "The History of Space Debris," *Embry-Riddle Aeronautical University,* 2014.

[15] D. Mehrholz, L. Leushacke, W. Flury, R. Jehn, H. Klinkrad and M. Landgraf, "Detecting, Tracking and Imaging Space Debris," *ESA,* 2002.

[16] A. Ellery, J. Kreisel and B. Sommer, "The case for robotic on-orbit servicing of spacecraft: Spacecraft reliability is a myth," *Acta Astronautica,* 2008.

[17] M. Garcia, "Space Debris and Human Spacecraft," NASA, 7 August 2017. [Online]. Available: https://www.nasa.gov/mission_pages/station/news/orbital_debris.html.

[18] C. R. Binz, M. A. Davis, B. E. Kelm and C. I. Moore, "Optical Survey of the Tumble Rates of Retired GEO Satellites".

[19] Z. He, W. Feng, X. Zhao and Y. Lv, "6D Pose Estimation of Objects: Recent Technologies and Challenges," 2020.

[20] N. O. Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan and J. Walsh, "Deep Learning vs. Traditional Computer VIsion".

[21] M. M. Mijwil, "Artificial Neural Networks Advantages and Disadvantages," Linkedin, 27 January 2018. [Online]. Available: https://www.linkedin.com/pulse/artificial-neural-networks-advantages-disadvantages-maad-m-mijwel.

[22] G. S. Aglietti, "Current Challenges and Opportunities for Space Technologies," *Frontiers in Space Technologies,* 2020.

[23] L. Iess, C. Bruno and C. Ulivieri, "Satellite de-orbiting by means of electrodynamic tethers - Part II: System configuration and performance," *Acta Astronautica,* 2002.

[24] M. Shan, J. Guo and E. Gill, "Deployment dynamics of tethered-net for space debris removal," *Acta Astronautica,* 2017.

[25] J. Reed and S. Barraclough, "Deployment of harpoon system for capturing space debris".

[26] N. D. Tyson and M. Jah, *STARTalk Radio,* 2021.

[27] J. Langlois, H. Mouchère, N. Normand and C. Viard-Gaudin, "3D Orientation Estimation of Industrial Parts from 2D Images using Neural Networks," *ICPRAM,* 2018.

[28] T. S. Huang, "Computer Vision: Evolution and Promise", 1996.

[29] A. Lee, "Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics," *Semantic Scholar,* 2016.

[30] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," *ECCV,* 2006.

[31] E. Karami, M. Shehata and A. Smith, "Image Identification Using SIFT Algorithm: Performance Analysis against Different Image Deformations," *arVix,* 2017.

[32] H. Bay, T. Tuytelaars and L. V. Gool, "SURF: Speeded Up Robust Features," *ECCV,* 2006.

[33] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *arVix,* 2014.

[34] R. Girshick, "Fast R-CNN," *arVix,* 2015.

[35] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arVix,* 2016.

[36] S. Zhang, L. Wen, X. Bian, Z. Lei and S. Z. Li, "Single-Shot Refinement Neural Network for Object Detection," *arVix,* 2018.

[37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, "SSD: Single Shot MultiBox Detector," *arVix,* 2016.

[38] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arVix,* 2016.

[39] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *arVix,* 2016.

[40] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arVix,* 2018.

[41] G. Boesch, "Object Detection in 2021: The Definitive Guide," viso.ai, 9 July 2021. [Online]. Available: https://viso.ai/deep-learning/object-detection/.

[42] Y. Xiang, T. Schmidt, V. Narayanan and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *arXiv,* 2018.

[43] C. Capellen, M. Schwarz and S. Behnke, "ConvPoseCNN: Dense Convolutional 6D Object Pose Estimation," *arVix,* 2019.

[44] M. Kisantal, S. Sharma, T. H. Park, D. Izzo, M. Märtens and S. D'Amico, "Satellite Pose Estimation Challenge: Dataset, Competition Design and Results," *arVix*, 2020.

[45] B. Chen, J. Cao, A. Parra and T.-J. Chin, "Satellite Pose Estimation with Deep Landmark Regression and Nonlinear Pose Refinement," *arVix*, 2019.

[46] Y. LIU, Z. XIE and H. LIU, "Three-line structured light vision system for non-cooperative satellites in proximity operations," *Chinese Journal of Aeronautics*, 2020.

[47] M. A. Post, X. T. Yan, J. Li and C. Clark, "Visual pose estimation system for autonomous rendezvous of spacecraft," *ICRA*, 2015.

[48] P. Tsarouchi, S.-A. Matthaiakis, G. Michalos, S. Makris and G. Chryssolouris, "A method for detection of randomly placed objects for robotic handling," *CIRP Journal of Manufacturing Science and Technology*, 2016.

[49] F. Casado, Y. L. lapido, D. P. Losada and A. Santana-Alonso, "Pose Estimation and Object Tracking Using 2D Images," *Procedia Manufacturing,* 2017.

[50] B. Barrois and C. Wöhler, "Spatio-temporal 3D Pose Estimation of Objects in Stereo Images," *ICVS,* 2008.

[51] N. Blodow, D. Gossow, S. Gedikli, R. B. Rusu and G. Bradski, "CAD-model recognition and 6dof pose estimation using 3D cues," *IEEE,* 2011 .

[52] J. J. Lim, A. Khosla and A. Torralba, "FPM: Fine pose Parts-based Model," *European Conference on Computer Vision*, 2014.

[53] A. Krull, E. Brachmann, F. Michel, M. Y. Yang, S. Gumhold and C. Rother, "Learning Analysis-by-Synthesis for 6D Pose Estimation in RGB-D Images," *ICCV.*

[54] G. Georgakis, S. Karanam and J. Kosecka, "Matching RGB Images to CAD Models for Object Pose Estimation," *arVix,* 2018.

[55] T. M. Debidatta Dwibedi, V. Badrinarayanan and A. Rabinovich, "Deep Cuboid Detection: Beyond 2D Bounding Boxes," *arVix*, 2016.

[56] M. Rad and V. Lepetit, "BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth," *arVix*, 2018.

[57] A. Crivellaro, M. Rad, Y. Verdie, K. M. Yi, P. Fua and V. Lepetit, "A Novel Representation of Parts for Accurate 3D Object Detection and Tracking in Monocular Images," *IEEE*, 2015.

[58] A. Grabner, P. M. Roth and V. Lepetit, "3D Pose Estimation and 3D Model Retrieval for Objects in the Wild," *arVix*, 2018.

[59] T.-T. Do, M. Cai, T. Pham and I. Reid, "Deep-6DPose: Recovering 6D Object Pose from a Single RGB Image," *arVix*, 2018.

[60] F. Guo, Y. He and L. Guan, "RGB-D camera pose estimation using deep neural network," *IEEE*, 2017.

[61] I. Melekhov, J. Ylioinas, J. Kannala and E. Rahtu, "Relative Camera Pose Estimation Using Convolutional Neural Networks," *arVix*, 2017.

[62] J. Liu and S. He, "6D Object Pose Estimation Based on 2D Bounding Box," *arVix*, 2019.

[63] S. Mahendran, H. Ali and R. Vidal, "3D Pose Regression using Convolutional Neural Networks," *arVix*, 2017.

[64] K. Yoshida and H. Nakanishi, "The TAKO (Target Collaborativize) Flyer: a New Concept for Future Satellite Servicing," 2002.

[65] W. McMahan, V. Chitrakaran, M. Csencsits, D. Dawson, I. Walker, B. Jones, M. Pritts, D. Dienno, M. Grissom and C. Rahn, "Field trials and testing of the OctArm continuum manipulator," *IEEE*, 2006.

[66] "RObotic GEostationary orbit Restorer (ROGER)," ESA. [Online].

[67] M. Lavagna, R. Armellin, A. Bombelli and R. Benvenuto, "Debris removal mechanism based on tethered nets," *iSAIRAS*, 2012.

[68] N. Zinner, A. Williamson, K. Brenner, J. Curran, A. Isaak, M. Knoch, A. Leppek and J. Lestishen, "Junk Hunter: Autonomous Rendezvous, Capture, and De-Orbit of Orbital Debris," *ARC,* 2012.

[69] U. Bindra, L. Murugathasan, V. Jain, G. Li, J. Kang, C. Du, Z. H. Zhu, F. T. Newland, M. Alger, O. Shonibare and A. d. Ruiter, "DESCENT: Mission Architecture and Design Overview," *ARC,* 2017.

[70] K. Wormnes, R. L. Letty, L. Summerer, R. Schonenborg, O. Dubois-Matra, E. Luraschi and A. Cropp, "ESA technologies for space debris remediation," 2013.

[71] D. Reintsema, J. Thaeter, A. Rathke, W. Naumann, P. Rank and J. Sommer, "DEOS – The German Robotics Approach to Secure and De-Orbit Malfunctioned Satellites from Low Earth Orbits".

[72] D. Reintsema, K. Landzettel and G. Hirzinger, "DLR's Advanced Telerobotic Concepts and Experiments for On-Orbit Servicing", *Advances in Telerobotics,* 2007.

[73] "About Canadarm," Government of Canada, [Online]. Available: https://www.asc-csa.gc.ca/eng/canadarm/about.asp.

[74] "About Dextre," Government of Canada, [Online]. Available: https://www.asc-csa.gc.ca/eng/iss/dextre/about.asp.

[75] T. Boge, T. Wimmer, O. Ma and M. Zebenay, "EPOS−A Robotics-Based Hardware-in-the-Loop Simulator for Simulating Satellite RvD Operations," 2010.

[76] K. Seweryn, J. Baran, T. Barciński and P. Colmenarejo, "The prototype of space manipulator WMS LEMUR dedicated to capture tumbling satellites in on-orbit environment," *IEEE,* 2017.

[77] K. Seweryn, T. Rybus, P. Colmenarejo, G. Novelli, J. Oleś, M. Pietras, J. Z. Sasiadek, M. Scheper and K. Tarenko, "Validation of the Robot Rendezvous and Grasping Manoeuvre Using Microgravity Simulators," *IEEE,* 2018.

[78] "Anchor Boxes for Object Detection," MathWorks, [Online]. Available: https://www.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html.

[79] "Data Augmentation," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Data_augmentation. [Accessed 2021].

[80] I. C. Education, "Overfitting," IBM, 3 March 2021. [Online]. Available: https://www.ibm.com/cloud/learn/overfitting.

[81] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," Machine Learning Mastery, 3 July 2017. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[82] P. Lynch, "The many modern uses of quaternions," The Irish Times, 4 October 2018. [Online]. Available: https://www.irishtimes.com/news/science/the-many-modern-uses-of-quaternions-1.3642385.

[83] "Quaternion," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Quaternion. [Accessed 2021].

[84] D. Kostyaev, "Better rotation representations for accurate pose estimation," Towards Data Science, 13 November 2020. [Online].

[85] A. Al-Masri, "How Does Back-Propagation in Artificial Neural Networks Work?," Towards Data Science, 29 January 2019. [Online]. Available: https://towardsdatascience.com/how-does-back-propagation-in-artificial-neural-networks-work-c7cad873ea7.

[86] "Conversion between quaternions and Euler angles," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles. [Accessed 2020].

[87] Narayan, "An overview of on-board computer (OBC) systems available on the global space marketplace," Sat Search, 14 October 2021. [Online]. Available: https://blog.satsearch.co/2020-03-11-an-overview-of-on-board-computer-obc-systems-available-on-the-global-space-marketplace.

[88] "The Transfer of Heat Energy," National Weather Service, [Online]. Available: https://www.weather.gov/jetstream/heat. [Accessed 2021].

[89] *NVIDIA Jetson Nano System-on-Module - Data Sheet.*

[90] "NVIDIA Jetson Nano," NVIDIA, [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/. [Accessed 2020].

[91] "Install RealSense Camera in 5 minutes – Jetson Nano," Jetson Hacks, 22 December 2019. [Online]. Available: https://www.jetsonhacks.com/2019/12/22/install-realsense-camera-in-5-minutes-jetson-nano/.

[92] "jetson-nano-yolov2-darkflow," GitHub, [Online]. Available: https://github.com/valdivj/jetson-nano-yolov2-darkflow.

[93] A. Herdarian, "Yolov5 Object Detection on NVIDIA Jetson Nano," Towards Data Science, 31 July 2021. [Online]. Available: https://towardsdatascience.com/yolov5-object-detection-on-nvidia-jetson-nano-148cfa21a024.

[94] Y. Movshovitz-Attias, T. Kanade and Y. Sheikh, "How useful is photo-realistic rendering for visual learning?," *arVix*, 2016.

[95] "FANUC," FANUC, [Online]. Available: https://www.fanucamerica.com/products/robots/series/m-20/m-20id-25. [Accessed 2020].

[96] "Jacobian," ROS Robotics Learning, [Online]. Available: https://www.rosroboticslearning.com/jacobian. [Accessed 2021].

[97] *FANUC M-20iD/25 - Data Sheet,* FANUC.

[98] M. Spong, S. Hutchinson and M. Vidyasagar, Robot Modeling and Control, 2005.

[99] F. Chaumette and S. Hutchinson, "Visual servo control, Part 1: Basic Approaches," *HAL,* 2009.

[100] "Four types of robot control," Brain Kart, [Online]. Available: https://www.brainkart.com/article/Four-types-of-robot-control_5131/. [Accessed 2020].

[101] C. Yang, H. Ma and M. Fu, "Robot Kinematics and Dynamics Modeling," *Springer Link,* 2016.

[102] "PID Controller," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/PID_controller. [Accessed 2020].

[103] "Dynamics: Overdamped vibration have damping ratio greater then 1.0," Eng-Tips, [Online]. Available: https://www.eng-tips.com/viewthread.cfm?qid=434790.

[104] H. Benninghoff, F. Rems, M. Gnat, R. Faller, R. Krenn, M. Stelzer, B. Brunner and G. Panin, "End-to-End Simulation and Verification of Rendezvous and

Docking/Berthing Systems using Robotics," in *5th International Workshop on Verification and Testing of Space Systems*, Torino, 2016.