BEng Individual Project - Final Report

Pointing, Acquisition and Tracking Device for Bidirectional Laser Communication

April 23rd, 2019

Maurice Rahme - s1557582 Word Count: 9547 LaTeX

Supervised by Dr. Aristides Kiprakis

Personal Statement

The PATBLC is a new project at The University of Edinburgh. It was supervised by Dr. Aristides Kiprakis, who provided feedback and advice during weekly meetings. Notably, he suggested the use of servo motors for the project, and helped troubleshoot a myRIO malfunction by identifying servo motor noise as the culprit. He also supplied the camera for this project from a previous project, and ordered components.

This project consulted and amalgamated various resources for inspiration, but to current knowledge is a unique combination of the various optical, kinematic and vision elements within. The aim was to design and manufacture a prototype that is robust and consistent using all available resources.

The PATBLC device was designed and built from scratch. Most elements were 3D Printed at the InnSpace Laboratory using a Prusa i3 Mk3. The bullseye was printed in the Sanderson Mechanical Laboratory using red PLA. The assembly was conducted in the Fleeming Jenkin Building as arranged by Dr. Kiprakis. The motor deconstruction and modification was also conducted in TLA using the soldering equipment there. The electronics workshop on the ground floor of TLA was also used for sourcing fasteners and for processing prints.

A Laser Safety Training course was attended at The University of Edinburgh on 28/02/2019, for which a certificate was obtained.

The focus lenses for the system were not received, because they were not purchased from official suppliers since those did not provide cost-effective options. Additionally, the second camera was not received, preventing the second PATBLC from being built and hampering the testing procedure, which instead relied on a human operator to hold the target with their 6 degree-of-freedom arm, which does not reflect system conditions.

This project has imparted knowledge on optics, computer vision, kalman filtering, inverse kinematics, and LabVIEW programming. These topics were novel, and their practical implementation was rewarding. Additionally, the servo motors' modification for position feedback granted in-depth knowledge on their operation. Finally, moving from the desktop programming environment to the myRIO was a an opportunity to learn memory management, which significantly affected system performance.

Summary

Pointing and Tracking Device for Bidirectional Laser Communication. Maurice Rahme - 23/04/2019.

This report records the design, manufacturing and testing for the Pointing and Tracking Device for Bidirectional Laser Communication (PATBLC) MKI. This device was conceptualised to supplement another ongoing project at The University of Edinburgh, the SquidROV. The PATBLC should supplement the SquidROV by granting it wireless communication in its submerged environment. The MKI prototype produced here, however, does not run underwater due to budget and time limitations.

The system is comprised mostly of 3D printed components and is actuated by two Tower Pro MG996R servo motors mounted perpendicularly to achieve a half-spherical workspace for the End-Effector. Inverse Kinematics is implemented to control the restricted motors whereby the End-Effector is flipped during one quarter-sphere of said workspace. The motors used here produce too much torque for the PATBLC, but were the only cost-effective option. Their torque is limited by using an ease-out function to dampen the motors' response near their commanded targets.

The End-Effector houses an optical breadboard carrying a laser, a dichroic mirror, a focus lens and a receiver - although the focus lens was not received for this prototype. The system sending the blue laser houses a short-pass dichroic mirror to pass its own beam out of the End-Effector and to reflect its twin system's beam onto its photodiode receiver for data reception. The alternate system uses a long-pass dichroic mirror and a red beam. A green beam was originally intended, but it was not possible to find one below the laser class III, which is unsuitable for un-enclosed use at the university.

The computer vision algorithm uses elliptical shape detection coupled with a Kalman Filter for skew and occlusion-resistant-tracking. The system is programmed in LabVIEW using a Queued Message Handler design pattern to actuate Queued State Machines. Unfortunately, only one camera was received, so only one system was built at this time.

The PATBLC performed well, with a link closing time of 0.23s/60° and 85% accuracy over 20 trials for a target at a 500mm distance. Accuracy over various distances could not be measured due to noise limitations as well as the lack of a second system. Some of the future-work recommendations include the implementation of an instrumentation amplifier for measurement noise reduction, and the use of a more powerful controller to extend the target identification range beyond the current 870mm.

Acknowledgements

I would like to start by thanking Dr. Aristides Kiprakis for giving me the opportunity to work on this project. I believe it has thoroughly prepared me for the graduate studies I plan to undertake in Robotics this fall. This project has truly amalgamated all of the skills I have acquired at The University of Edinburgh, and I am grateful for the experience.

I would also like to thank Ian Gold and Alasdair Chirstie for their indispensable support in the Electronics Laboratory. Likewise, I would like to thank Kevin Tierney for letting me borrow his equipment, and Douglas Carmichael for his LabVIEW support.

I also thank Dr. James Hopgood and Dr. Javier Escudero Rodriguez for helping me narrow my resources for learning computer vision by outlining criteria for adequate shape detection: skew, scale and rotation in-variance for diverse lighting conditions.

Additionally, I would like to thank Prof. Konstantin Kamenev for the feedback he provided during the interim meeting, which was crucial to the continuation of the project.

I would also like to thank Jing Zhang, who worked tirelessly on the optical data transfer portion of this project.

Finally, I would like to thank Mr. Daniel Adams from Edmund Optics for providing me with two dichroic mirrors at a generous 50% discount. This project would not have been possible without this contribution, which was vital is staying within the £150 budget.

Notation

θ_n: the rotation of the n^{th} frame about the z_{n-1} axis so that the x_n and x_{n-1} axes match in a Kinematic Diagram.

H_3^0: the Homogeneous Transformation Matrix for the PATBLC, relating frames 3 and 0.

c_n: an abbreviation of $\cos_{\theta\,n}$.

s_n: an abbreviation of $\sin_{\theta\,n}$.

ϕ_1: the maximum allowable error angle of the outgoing beam for each PATBLC.

ϕ_2: the maximum allowable error angle of the outgoing beam through the acrylic window.

δ: the beam deviation angle caused by passing through a flat medium.

ϕ_3: the maximum allowable error angle of the outgoing beam through the focus lens.

σ: standard deviation.

$\bar{x_t}$: the predicted next-state vector.

x_{t-1}: the current state vector.

ε_x: the Gaussian state error estimation, or process noise.

$\bar{z_t}$: the sensor prediction.

$\bar{x_t}$: the predicted state vector.

ε_z: the Gaussian sensor error or measurement noise

x_{EST}: the Kalman Filter output.

$(z_t - \bar{z_t})$: the Kalman correction term.

K_t: the Kalman Gain.

$\bar{\Sigma_t}$: the final covariance.

D: one pixel length.

E_{2a}: the Equivalent Ellipse Major Axis.

E_{2b} the Equivalent Ellipse Minor Axis.

ϕ: elevation in real-world coordinates.

θ: azimuth in real-world coordinates.

x_t: horizontal position in pixels.

y_t: vertical position in pixels.

$\dot{x}_t$: horizontal velocity in pixels/second.

$\dot{y}_t$: vertical velocity in pixels/second.

σ^2: variance.

μ_x: the data-set's mean.

X: the sample in the variance calculation.

N: the data-set's size in the variance calculation.

Summary of Resources

The University of Edinburgh's LabVIEW and MATLAB licenses were used for this project, in addition to a SolidWorks license provided by an extracurricular group. The main workshop used was the TLA Electronics Laboratory in the Fleeming Jenkin Building.

The Project InnSpace Laboratory was used for its 3D printing services. The Sanderson Mechanical Laboratory was used once to print the PATBLC's bullseye using red PLA.

Contents

1 Introduction

1.1 Motivation

This project is inspired by the Lincoln Laboratory (LL)'s recent field demonstration of narrow-beam laser technology for undersea optical communication. The LL notes that lack of communication with undersea rovers increases likelihood of drift and loss of the rovers themselves [1]. Current protocols include acoustic modems and radio frequency (RF), which suffer from low underwater data rates (100bytes/s) [2]. Conversely, the LL's field test recorded data rates up to 8.7Mb/s at a 7.8m distance with 0.25mW lasers. Extrapolated to higher-power lasers, this data indicates link closing distances of up to 450m [1].

Based on the Optical Payload for Lasercomm Science (OPALS) experiment, this project aims to design and manufacture two pointing, acquisition and tracking devices for bidirectional laser communication (PATBLC) [3]. This project runs in parallel with Jing Zhang's, which focuses on optical data transfer. The combination of these projects will supplement Dr. Aristides Kiprakis' autonomous underwater rover, SquidROV, with wireless communication capabilities. This should allow the SquidROV to explore environments beyond The University of Edinburgh's (UofE) WaveTank.

1.2 Project Scope

Considering the short timeline and £150 budget given to build two PATBLCs, it was decided to forego submerged testing. The encapsulation of control, imaging and optical systems in a waterproof package whilst maintaining dry-land performance is a project in its own right. That said, this system is designed with waterproofing in mind, and is adaptable to submersion.

1.3 Report Structure

This report begins with a project management strategy in chapter 2. It includes a project timeline in the form of a Gannt Chart along with a plan for managing setbacks.

Next, the literature review in chapter 3 presents an assessment of related works. Its main discussions pertain to the topics of inverse kinematics, optics, computer vision, and control.

Chapter 4 covers mechanical and electronic design. Chapter 5 details the computer vision and Kalman Filtering algorithms for this project both in theory by building upon the literature review, and in practice within LabVIEW. Next, chapter 6 concludes the Inverse Kinematics design for the system and introduces a motor easing algorithm for fine control. Chapter 7 describes the LabVIEW program.

Next, chapter 8 provides a performance analysis before drawing conclusions and recommendations for future work in chapter 9.

Finally, the references are provided and followed by the appendix.

2 Project Management

2.1 Philosophy and Execution

An error-tolerant project plan was developed to account for setbacks. The Gannt Chart in Appendix 1 is characterized by repeated tasks in red, acting as buffers to the project flow. The chart is easily modified if a task's difficulty is erroneously estimated. Sub-timelines split the project into five sections: report writing, administration, optical and physical system, image processing, control, and system integration. Each of these contains milestones, which, in addition to the percentage completion bars for each timeline, make challenges less daunting as their scale with respect to the project as a whole is evident.

The manufacturing and programming tasks for the PATBLC deviated from the interim report's plan. The servo motors were not received on time, so emphasis was placed on the LabVIEW program at earlier stages. Although this came with the drawback of identifying a mechanical design flaw at a late stage, - the End Effector's (EE) weight makes it ill-suited for fastening using servo motor horns, and a planetary gear link is recommended for future iterations - it was possible to spend more time programming to implement modular code with robust error-handling and communication protocols.

More data acquisition should have been done for a better understanding of system performance, but the decision to cease lab work on 12/04/2019 to focus on the report was respected.

3 Literature Review

3.1 System Specifications and Feasibility

The narrow-beam optical communication method proposed by the LL deviates from the traditional approach of wide-beam data transfer to avoid pointing and tracking (PAT) challenges. Therefore, the LL calls for an implementation of PAT to supplement data transfer [1]

The LL cites Petzold's measurements of absorption and scattering coefficients to identify the beam wavelengths with the lowest optical power attenuation due to submerged propagation [4]. They identify Green (517nm) wavelengths as ideal for the turbid harbor environment due to higher chlorophyll concentrations, and Blue (470nm) wavelengths as optimal for open ocean environments [1].

The field report's PAT section outlines criteria for effective link closing. The system should be robust to occlusion and intermittent outages from debris. This can be achieved with Kalman Filtering [5]. Beam jitter should be controlled with a Proportional, Integral, Derivative (PID) controller on each motorised degree of freedom (DOF) [1]. For the PATBLC, this is done using servo motors.

The LL suggests that the beams be controlled by a galvanometer (ThorLabs GVS-112) costing £2,500. Although these have $15\mu rad$ repeatability and $400\mu sec$ step response for a $0.02°$ step, their cost is prohibitive [6]. As shown in section 3.3, the accuracy required for the PATBLC is $0.1°$ for a submerged closing link of 5m.

3.2 Inverse Kinematics

Servo motors capped at 180° of rotation are used as joint actuators. Each acts as a revolute joint oriented at 90° from the other as seen in figure 1, enabling a half-spherical workspace [7].

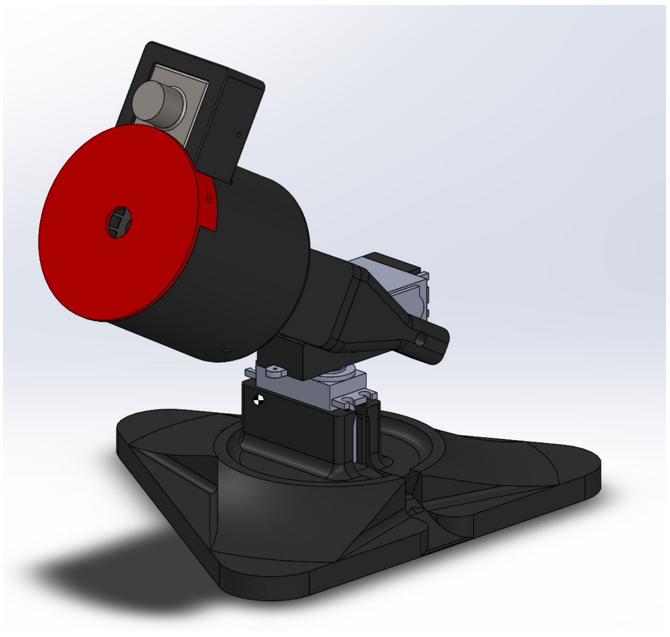


Figure 1: Isometric view of the CAD model in SolidWorks with the motors coloured in grey.

Inverse Kinematics (IK) can map the restricted joints to orientations that allow this half-spherical workspace. The derivation in Appendix 2 is based on the Denavit-Hartenberg Method for spherical wrists to find the system's Homogeneous Transformation Matrix (HTM):

$$H_3^0 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1c_2s_3 - s_1c_3 & c_1s_2 & c_1s_2d_3 \\ s_1c_2c_3 + c_1s_3 & -s_1c_2s_3 + c_1c_3 & s_1s_2 & s_1s_2d_3 \\ -s_2c_3 & s_2s_3 & c_2 & c_2d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

The rotation part of the HTM, comprised of the first 3x3 elements, can be compared to the Euler Angle Transformation (EAT) which describes the manipulator's desired orientation. The EAT is derived in Appendix 3. For the standard order of roll-pitch-yaw (X-Y-Z), it is:

$$R_{XYZ} = \begin{bmatrix} c_zc_y & -s_zc_x + c_zs_ys_x & s_zs_x + c_zs_x + c_zs_yc_x \\ s_zc_y & c_zc_x + s_zs_ys_x & -c_zs_x + s_zs_yc_x \\ -s_y & c_ys_x & c_yc_x \end{bmatrix} \tag{3.2}$$

This yields a system of nine equations with three unknowns. The compared elements are arbitrary. The simplest are: H_{33},H_{32} and H_{23} with R_{33}, R_{32}, and R_{23}, giving the following relationship for each joint relative to the desired orientation:

$$\theta_1 = \arcsin\left(\frac{R_{23}}{\sin(\theta_2)}\right) \tag{3.3}$$

$$\theta_2 = \arccos(R_{33}) \tag{3.4}$$

$$\theta_3 = \arcsin\left(\frac{R_{32}}{\sin(\theta_2)}\right) \tag{3.5}$$

θ_3 corresponds to the revolute joint on the EE whose rotation is inconsequential. With this relationship, the required joint angles can be computed for any given EAT orientation 3.2. The calculated joint angles are plugged back into equation 3.1's rotation component and compared with equation 3.2 for confirmation at each iteration using LabVIEW's robotics toolbox [8, 9].

3.3 Optical Design

The optical design's goal was to allow the incoming and outgoing beams on each device to share the same axis, permitting the IK in section 3.2. If these beams do not share an axis, the twin devices will only be aligned for a quarter-sphere range of motion. The only non-optical solution to this is to add a third revolute joint on the EE.

The OPALS' optical scheme, shown in figure 2 served as a good starting point for the optical design [3].

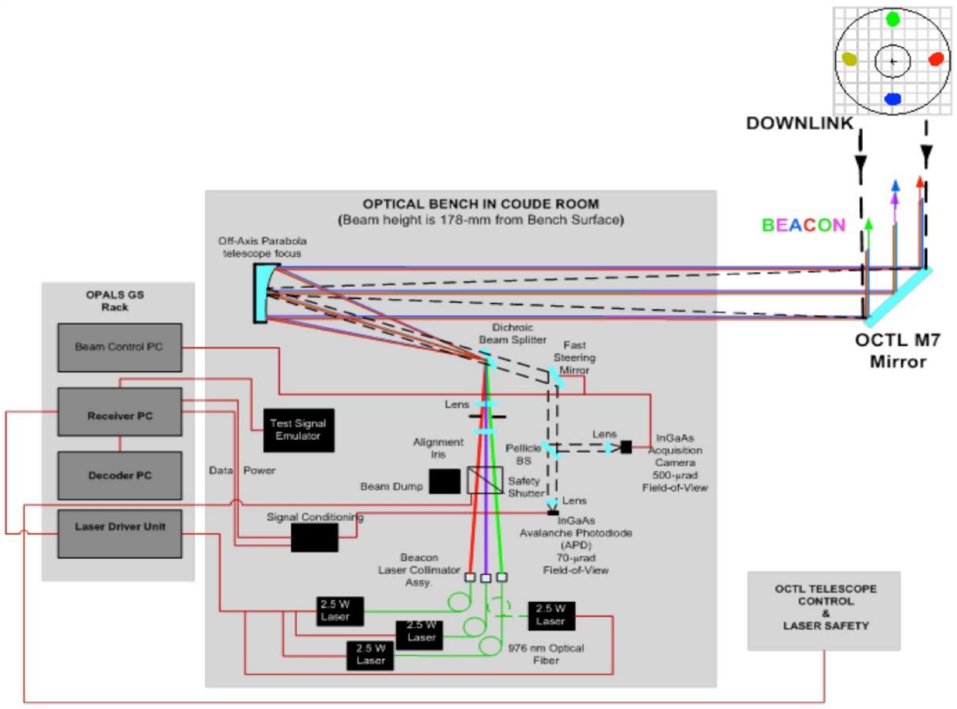


Figure 2: The OPALS Optical Scheme [3].

The Off-Axis Parabolic Focus Lens (OAP) guides incoming beams to a dichroic beam splitter, which passes the down-link beam (dotted line) to the photo-diode detector and camera through a spectral filter to block unwanted signals. The up-link beam is reflected by the dichroic beam splitter and is collimated by the OAP before being sent to the ground station. The OPALS' design enables the camera to be placed within the optical setup as the satellite knows roughly where to point thanks to its orbit and RF communication with the ground station [3].

A simpler design is proposed for the PATBLC:

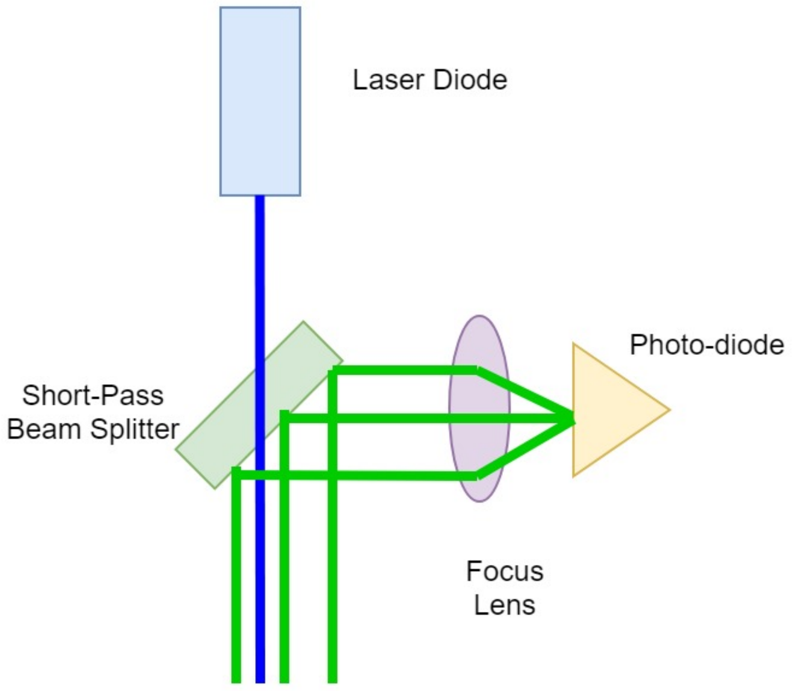


Figure 3: The PATBLC Optical Scheme

Referring back to section 3.1, one device transmits a Blue (470nm) beam, and the other transmits a Green (517nm) beam. In practice, it was only possible to acquire a green class III laser at cost, which is unsuitable for un-enclosed actuation at UofE. A Red class II beam was used instead. In figure 3, a short-pass dichroic beam splitter with a 500nm cut-off wavelength is used to transmit the outgoing Blue beam and reflect the incoming Green beam onto the focus lens and into photo-diode one focal length away. The alternate device uses a long-pass dichroic beam splitter [10].

Ray tracing was done to specify system tolerances for a submerged environment. Tolerances are looser in the dry-land prototype. Figure 4 shows the acrylic windows used for waterproofing. The following analysis finds the maximum ϕ_1 error angle of the outgoing beam for it to hit the 12.5mm target located at a maximum distance of 5000mm (the WaveTank depth) [11].

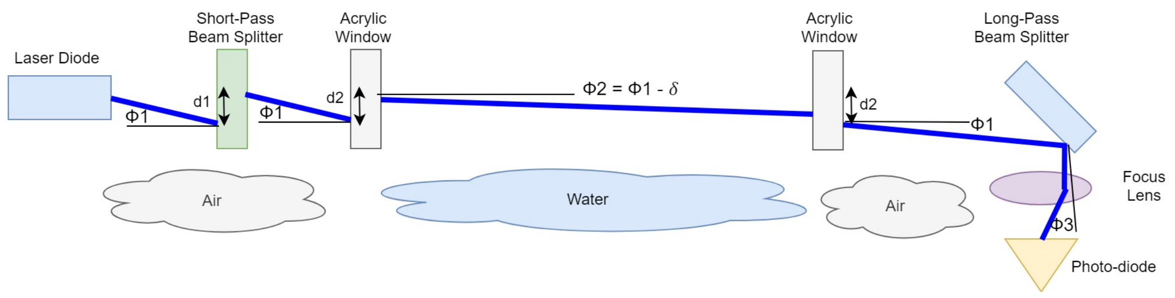


Figure 4: Ray Tracing for offset tolerance investigation

n_{air}	n_{water}	$n_{acrylic}$	n_{DM}
1	1.333	1.49	1.55

Table 1: Refraction indices for relevant mediums [10,12]

The displacement from passing through a flat medium (d_1 and d_2) is neglected in this worst-case analysis as it increases tolerance. Optical component distances are ignored since the 5000mm travel is an over-estimate. The diagram treats the dichroic mirror for the outgoing laser as perpendicular to the ray. This is done because the angle of incidence for passing beams is 45°, and this visualisation simplifies the analysis.

First, the maximum angle out of the acrylic window ϕ_2 is computed using trigonometry:

$$\phi_2 = \arctan\left(\frac{6.25}{5000}\right) = \pm 0.0716^\circ \tag{3.6}$$

The target diameter is 12.5mm, constrained by the beam splitter. Hence, the beam can deviate by a maximum of 6.25mm.

To find the beam offset angle and the deviation angle caused by passing through a flat medium, ϕ_1 and δ, system of two equations 3.8 and 3.7 with two unknowns was solved iteratively in MATLAB, starting with $\delta = 0$ and looping until the increments in δ and ϕ_1 became negligible [12]. The code is provided in Appendix 4.

$$\delta = \phi_1 - \arcsin\left(\frac{n_{air}}{n_{water}}\sin(\phi_1)\right) \tag{3.7}$$

$$\phi_1 = \phi_2 + \delta \tag{3.8}$$

After 1000 iterations, the following was recorded: $\delta = \pm 4.1625e{-4}$ rad, and hence $\phi_1 = \pm 0.0017$ rad $= \pm 0.098^\circ \approx \pm 0.1^\circ$.

To find the effect of this offset on the focus lens, ϕ_3 is computed, noting that passing the second acrylic window negates the effect of δ on ϕ_2, and that the incidence angle on the dichroic mirror is ϕ_1. Equating the angle of incidence with the angle of reflection for the mirror, we can equate ϕ_3 to ϕ_1, giving a final offset limit of 0.1° [12].

3.4 Computer Vision

A digital image is a matrix composed of i columns and j rows where each element is a pixel. A colour image is composed of three such matrices, denoting the Red, Blue, and Green planes of the captured image. The resolution is defined as the number of pixels in a plane. Each pixel can hold a value between 0 (black) and 255 (highest colour intensity) [13].

The tracking strategy for PATBLC was inspired by the OPALS experiment:

1. The brightest pixel in the FOV (640x480 pixels) is identified.
2. A centre-of-mass calculation is performed to find the bright spots centroid.
3. The background level is computed using a reduced 8x8 pixel subframe around the bright spot.
4. The flux of the Data Number (proportional to the number of detected photons) is summed in the 8x8 pixel subframe.
5. The background level is subtracted from the total flux to determine the beacon flux.

The beacon flux is thresholded to decide the transition open-loop to closed-loop tracking. This renders the system vulnerable false detection if bright noise is detected. Furthermore, undersea pointing accuracy has not been examined and the OPALS' beam diverges considerably to hit the target by the time it reaches the ground station, permitting lower accuracy [3].

The PATBLC's camera separate from the optical scheme, so they don't share the same frame, and the beam itself cannot be used as a target. Instead, a physical target is used. Coordinate transformation in the form of an elevation offset can be applied by computing the camera's HTM relative to the EE frame in the same manner outlined in section 5.2.5 [8,9].

To smooth the target for object detection, the algorithm employs Gaussian Blur, which uses a Gaussian convolution kernel to eliminate image noise by blurring each pixel. A convolution is an element-by-element multiplication of two matrices followed by summation [13].

To counter target occlusion, object tracking can be used to consider the object's position in the current frame and predict its position in the next frame. One such tracker is the Kalman Filter (KF), a recursive algorithm that assigns a confidence level to various state predictions and combines them to make a final estimate. The algorithm uses a state prediction equation and a sensor prediction equation, which are weighted and combined to make a final next-state estimate [5].

In the state prediction equation,

$$\bar{x_t} = Ax_{t-1} + B\mu_t + \varepsilon_x \tag{3.9}$$

$\bar{x_t}$ is the predicted next-state vector, and A and B within the plant model describe the system's previous state and the external input respectively. μ_t is the control signal, which is zero since its influence is unknown. x_{t-1} is the current state, and ε_x is the Gaussian state error estimation, or process noise. The state vector is a model of position and velocity in both x and y directions in the camera's FOV.

The state error, ε_x is written as the state covariance matrix 3.10 where the subscripts p and v denote position and velocity respectively.

$$E_x = \begin{bmatrix} \sigma_p^2 & \sigma_p\sigma_v \\ \sigma_v\sigma_p & \sigma_v^2 \end{bmatrix} \tag{3.10}$$

The sensor prediction equation predicts the next-state sensor measurement extrapolated from the next-state prediction:

$$\bar{z_t} = C\bar{x_t} + \varepsilon_z \tag{3.11}$$

$\bar{z_t}$ is the sensor prediction, $\bar{x_t}$ is the predicted state, and ε_z is the Gaussian sensor error or measurement noise. C denotes a relationship between the predicted state and the sensor measurement. It is set to 1 for correlated values and 0 otherwise.

The sensor error ε_z can be written as $E_z = \sigma_z^2$, the sensor covariance matrix. The KF requires a matrix solution because there are multiple states being analysed (position and velocity). Ez is calibrated based on expected sensor noise.

Finally, equations 3.9 and 3.11 are combined in the KF equation for the final state estimate,

$$x_{EST} = \bar{x_t} + K_t(z_t - \bar{z_t}) \tag{3.12}$$

where x_{EST} is the KF output, $(z_t - \bar{z_t})$ is the correction term. If $\bar{z_t}$ is equal to the sensor prediction, z_t, then only the state prediction is used in the final estimate, otherwise, it is corrected by a factor of K_t of the correction term. K_t is the Kalman Gain, which decides the importance of the correction term and is calculated using

$$K_t = \bar{\Sigma_t}C^T(C\bar{\Sigma_t}C^T + E_z)^-1 \tag{3.13}$$

denoting its behaviour as a measurement of sensor importance, since an increase in E_z leads to a reduction in K_t. $\bar{\Sigma_t}$ is the predicted covariance of the whole system:

$$\bar{\Sigma_t} = A\Sigma_{t-1}A^T + E_x \tag{3.14}$$

where Σ_{t-1} is the prior-state covariance calculated in the last iteration.

The final covariance, used as the prior in the next algorithm iteration, is:

$$\Sigma_t = (I - K_tC)\bar{\Sigma_t} \tag{3.15}$$

where I is the identity matrix. By multiplying $\bar{\Sigma}_t$ with the factor $(I - K_tC)$, it is evident that a more informative measurement, denoted by a higher K_t, has a lower system covariance. This is because an informative measurement indicates a more accurate final state estimation, x_{EST}.

Using these equations, the recursive KF algorithm is:

1. Prediction Step
   (a) Predict the next state 3.9
   (b) Predict the error covariance 3.14
2. Correction Step
   (a) Calculate the Kalman Gain 3.13
   (b) Correct the state estimate 3.9 using 3.12
   (c) Compute the final system covariance 3.15 and feed it back into 3.14 as Σ_{t-1} in step 1.b for the next algorithm iteration

The first iteration-state is distinguished in that its prior state, x_{t-1} is zero, and its prior state-covariance Σ_{t-1} is 1 as an assumption that the system contains noise [5].

3.5 Control

The pan-tilt system can be managed by controlling the motors responsible for azimuth and elevation independently. As detailed in section 4.4, servo motors have built-in PI controllers that ensure the commanded setpoint is reached with minimal overshoot and settling time. Here are the proposed control stages:

1. Open-Loop Tracking: each PATBLC performs a pan-tilt sequence detailed in section 7.3 to cover the half-sphere workspace. Once the target is detected, closed-loop tracking commences.
2. Closed-Loop-Tracking: the target's pixel coordinates are fed to the KF algorithm to compute an improved estimate before translating it into azimuth and elevation angles. subsequently, the servo motors in each device are actuated according to their IK descriptions to reach the correct attitude.

4 Design & Manufacturing

4.1 Mechanical Components

The mechanical design serves to:

1. Reduce physical 'wobble' from mounts and linkages.
2. Lower the Centre of Gravity (CoG) for a smaller impact on the SquidROV's stability.
3. Ensure a long lifespan by placing the PCB and myRIO inside the SquidROV and away from moving parts.

All components were 3D printed using black PLA except the bullseye, which was printed in red for identification. A calibration tool was printed to estimate tolerances. It prints in one piece with a series of cylinders housed in casings of varying tightness ranging from 0.05mm to 0.5mm. The cylinders spin if the tolerance is loose. It was found that a 0.25mm tolerance results in a manual press fit, although this was not always true for the smaller optical components.

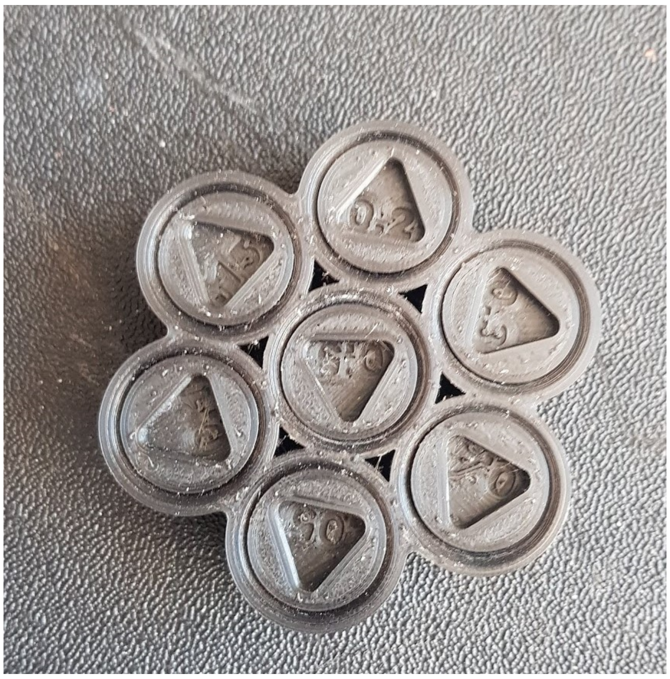


Figure 5: Calibration tool used to measure printing tolerance

The system has a static CoG of 42mm away from the centre horizontally. Since the EE's length was dictated by the length of the blue laser, an easing algorithm controls the motors' speed and torque to prevent toppling. The base's shape ensures that the edge cases from section 6.1 are better supported. The limited print-bed size meant that a triangular shape could be printed to fulfil this criterion but a circular one could not as it would protrude from the bed. The assembly instructions are provided in Appendix 8.

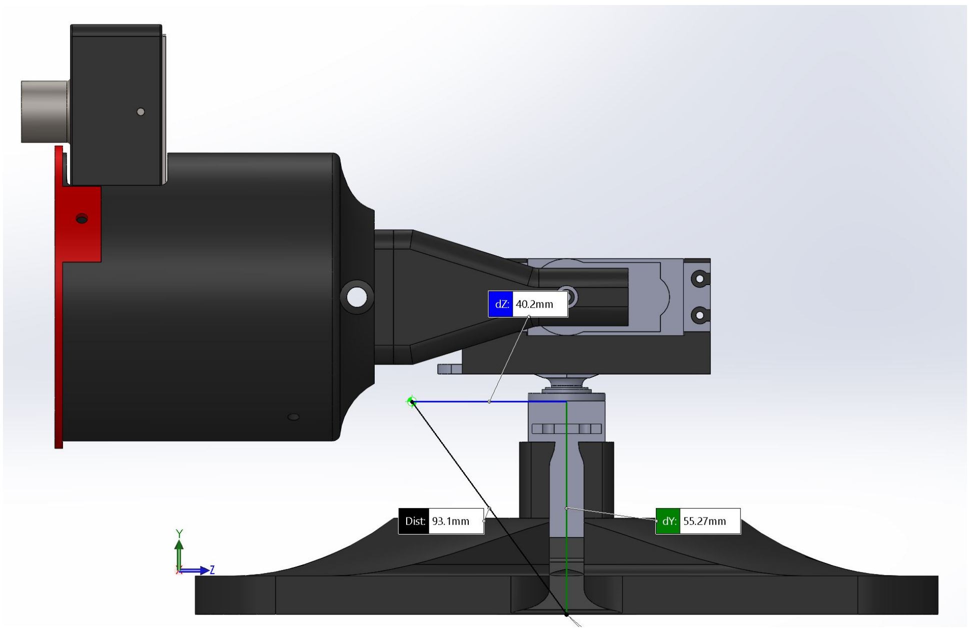


Figure 6: CAD Model with CoG

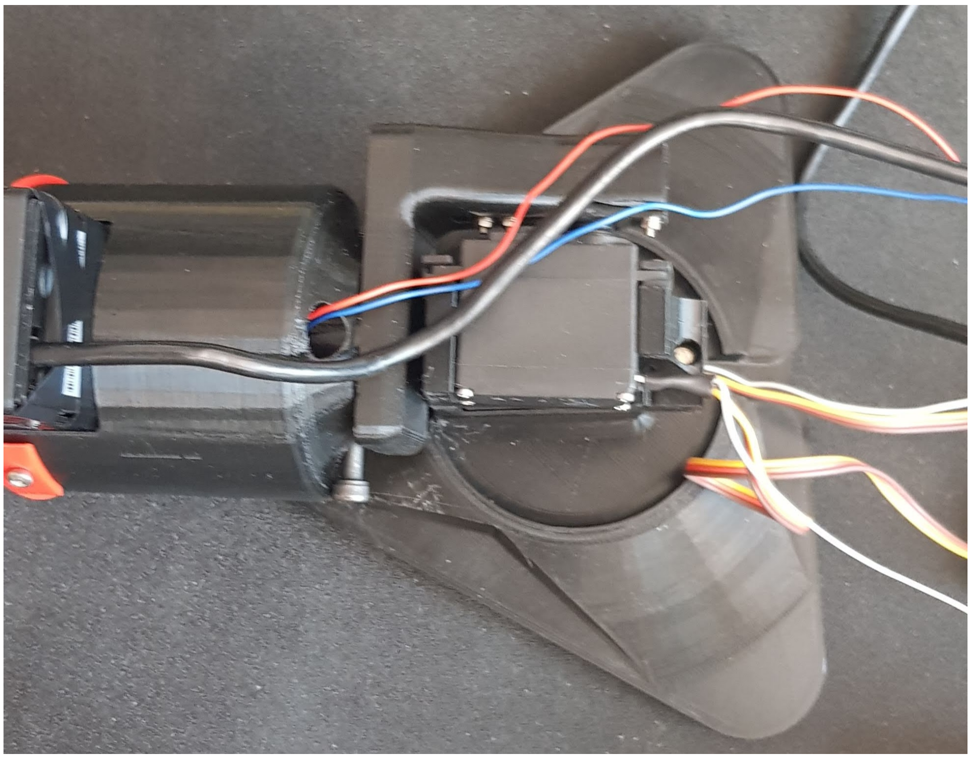


Figure 7: Triangular Base

4.1.1 End Effector & Camera Mount

These are the largest contributors to the PATBLC's CoG since the camera in its metal casing weighs 230 grams. The End Effector (EE) with its fasteners and optical breadboard brings the total weight to 413 grams.

A compromise was made between size and user-friendliness, since assembly and dis-assembly were performed regularly.

1. An extruded cut on the EE base allows the servo mounted arm to press-fit into the EE and be secured with an M5 bolt. A 12mm hole provides an egress point for the laser and receiver terminals.

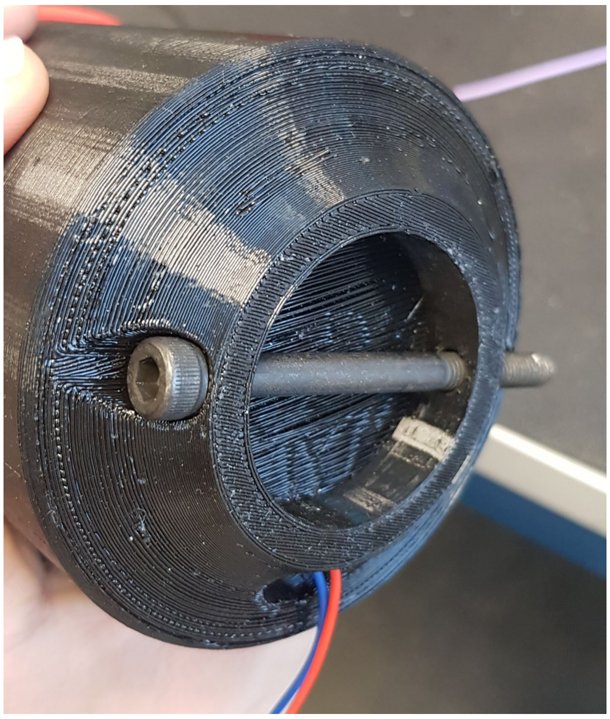


Figure 8: EE Base.

2. The EE contains two M3 slots to pass bolts to secure the optical breadboard from the outside.

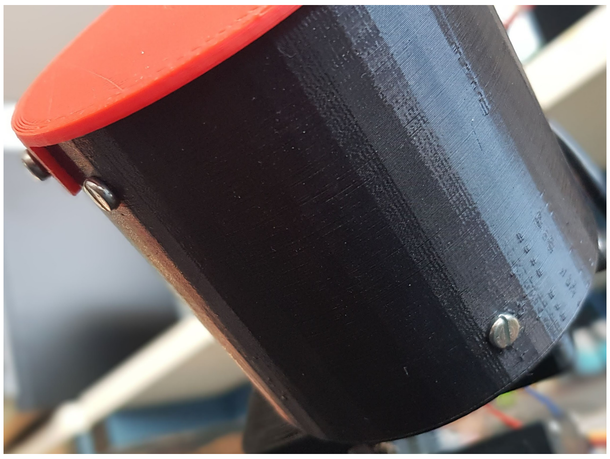


Figure 9: Slots to secure the breadboard to the EE.

3. A cavity with an octagonal slot secures the camera mount without fasteners.

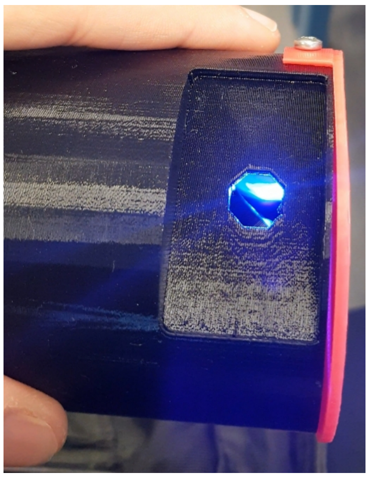


Figure 10: Slot to secure the camera to the EE.

4. On the EE's face, two 5mm deep holes with 5mm diameters house knurled inserts to fasten the bullseye.

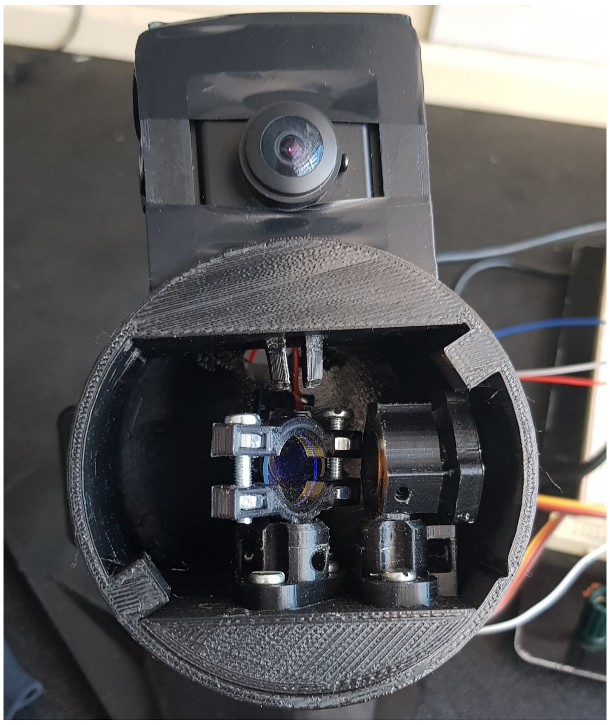


Figure 11: Knurled insert and camera mount extrusions.

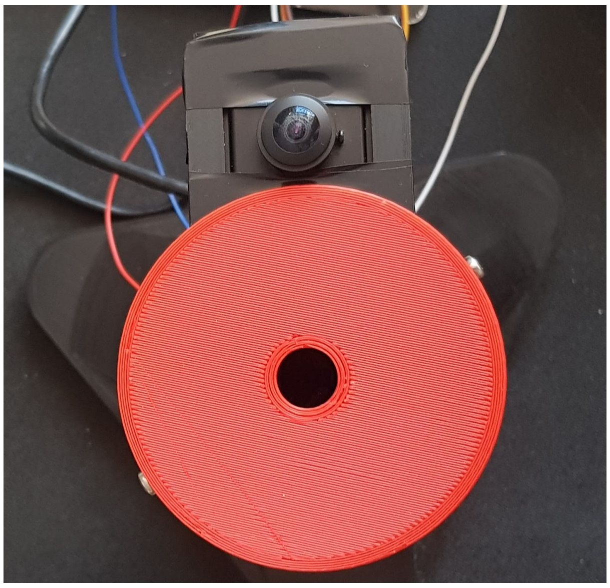


Figure 12: EE with bullseye.

The camera mount in figure 11 is secured to its metal casing via threaded holes. The mount has an octagonal extrusion without a 0.25mm tolerance relative to the EE slot. Its middle is cut out so that it bends to fit into the EE slot and click into place.

4.1.2 Base & Motor Mount

A common edge case where the EE flips and exerts high torques is from 0° to below 360° in azimuth, where one of the base's vertices provides additional support. An extruded frame houses M2 holes for the first motor's screws and a 10mm filleted slot on one side for its terminals. The fillet begins at 5mm from the screw entry point to expose enough threading material while limiting bending to the motor terminals during assembly.

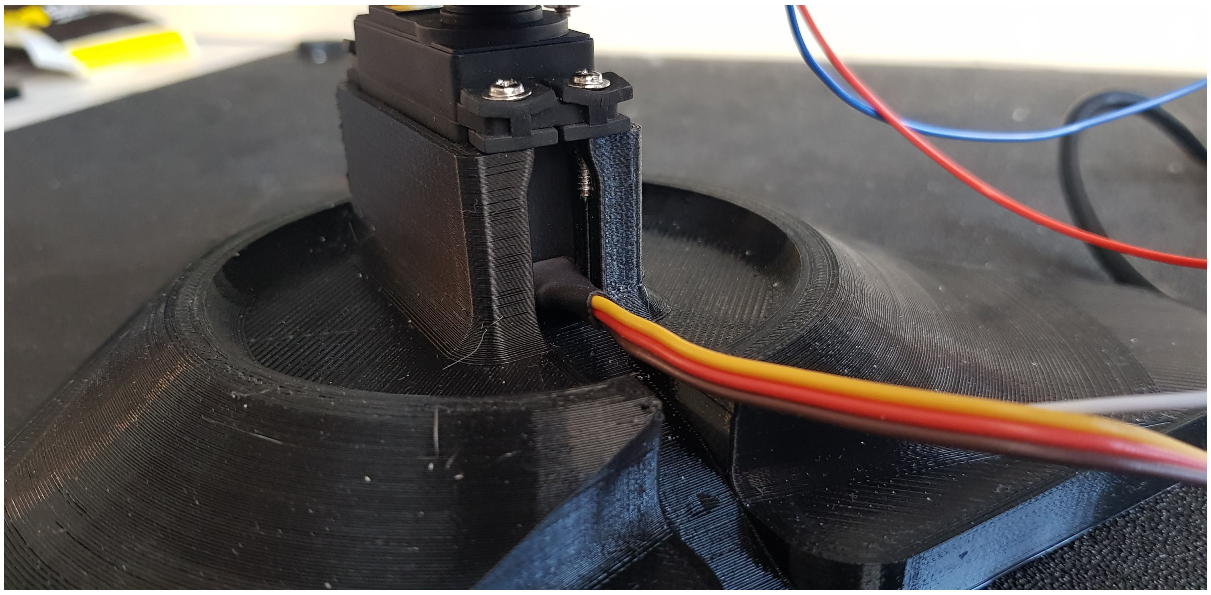


Figure 13: The base.

The motor mount aligns the second motor's shaft to first's centre of rotation, validating the IK in section 3.2. A cavity in the mount allows the first motor's horn to attach snugly.

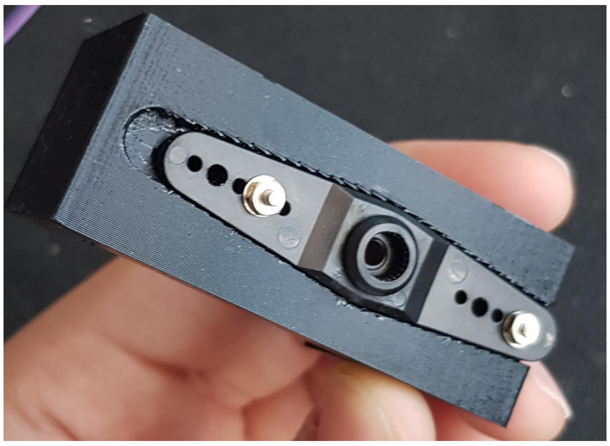


Figure 14: The motor mount.

Wobble was still experienced due to the EE's weight. For future iterations, this paper recommends the use of a 3D-printed planetary gear system instead of a servo horn attachment, which is coupled to the system's CoG. Unfortunately, by the time this flaw was noticed, it was too late to redesign and manufacture the part; the wobble was controlled using the easing algorithm.

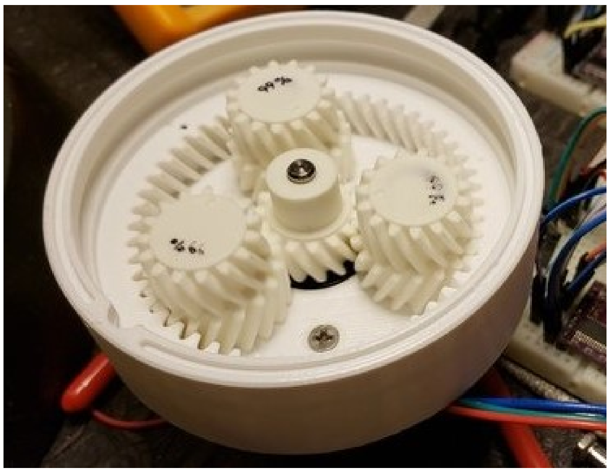


Figure 15: Example of a 3D-printed planetary gear system.

4.1.3 Servo Motor Arm

This link includes a cavity for its motor horn as well. Another cavity on the side of the arm creates the same 8.5mm thickness as the motor mount to allow the same bolts to be used in both components whilst maintaining the arm's thickness for load-bearing. Part of the horn is cut to avoid clashing mechanically with the first motor.

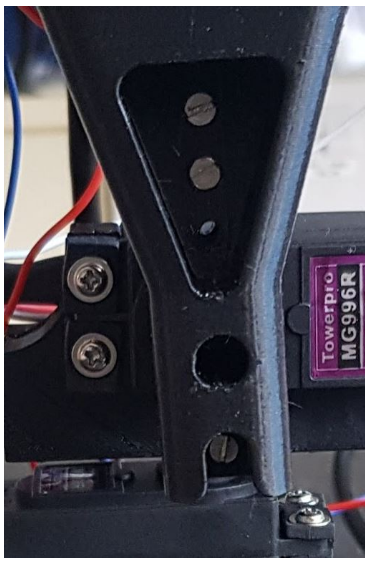


Figure 16: The servo arm exterior cavity.

The arm's face is filleted to its shoulder with a 5mm radius to minimize stress concentration. The cylindrical extrusion interfacing the EE originates on the centre of the first motor to validate the IK.

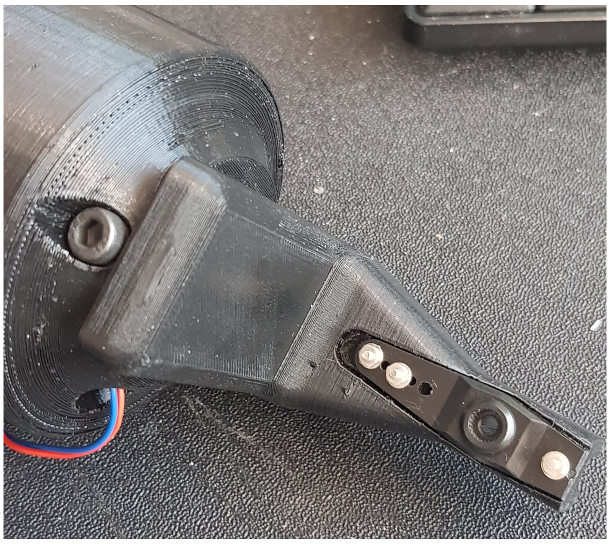


Figure 17: The servo arm fillet and interior cavity.

4.2 Optical Breadboard & Mounts

The optical breadboard houses knurled inserts. Two of these face the EE's inner surface, and the remaining six interface with the optical mounts.

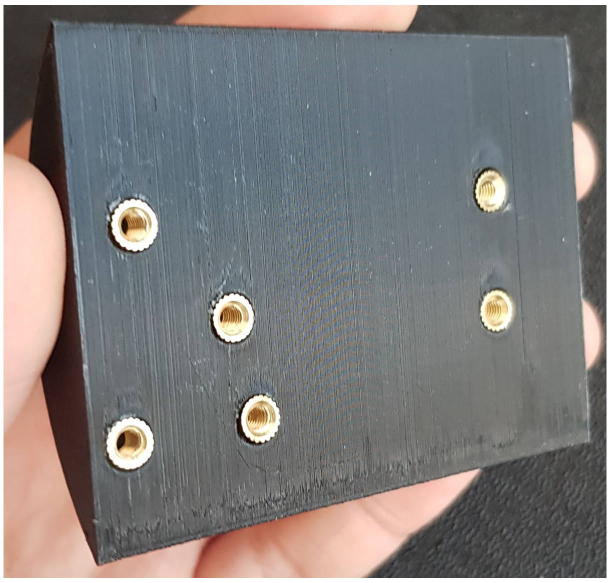


Figure 18: The breadboard with knurled inserts.

The post base houses two M3 holes to fasten to the breadboard along with an octagonal slot to fit the mounts' octagonal posts. An octagon was chosen as this permits mounting at 45°in case the breadboard design changes for size-reduction. Since 3D prints are not uniform, an M3 nut and slot housing allows a bolt to pass and secure the the octagonal posts.

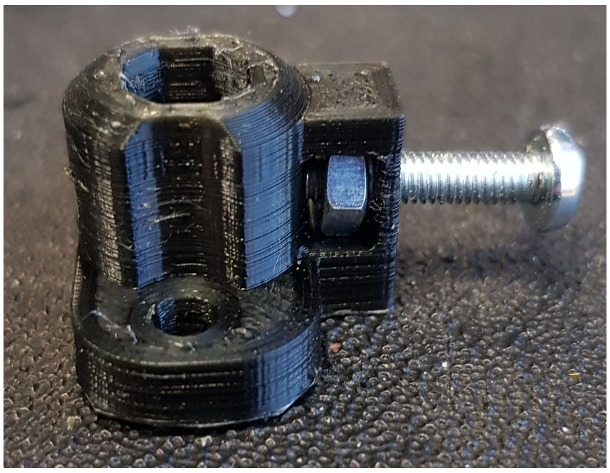


Figure 19: The post base.

The laser and focus lens mounts have octagonal extrusions to fit the post bases. The mount houses inner rings and fastener slots to secure focus lenses and diodes of varying radii externally if necessary.

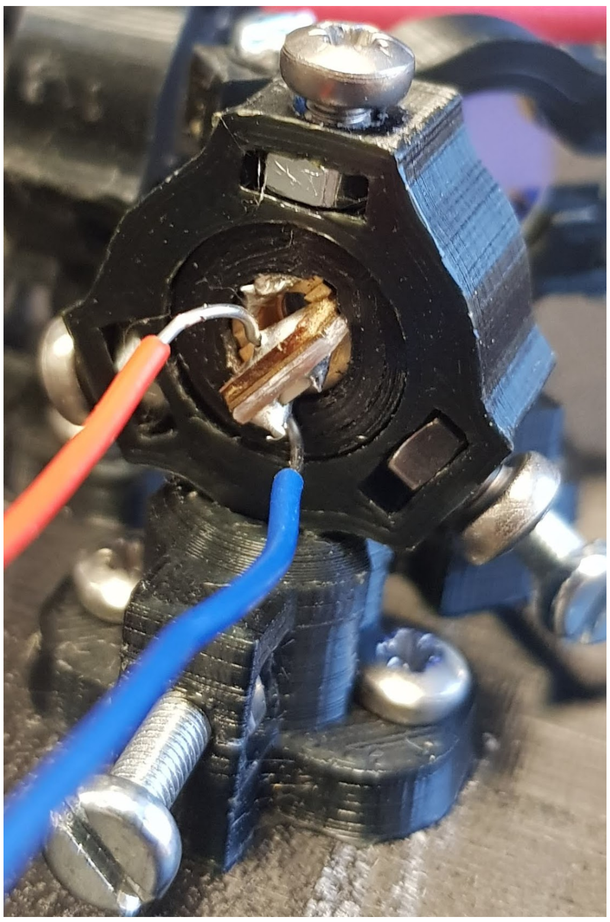


Figure 20: The red laser mounted with a small inner-ring and external fasteners.

Both parts of the dichroic mirror mount contain a small cavity of the lens' thickness for fitting, as well as protrusions on the side with M3 nut and bolt slots. The small cavity prevents skewed lens fastening and grants tactile feedback, generating an audible click during correct fitting.

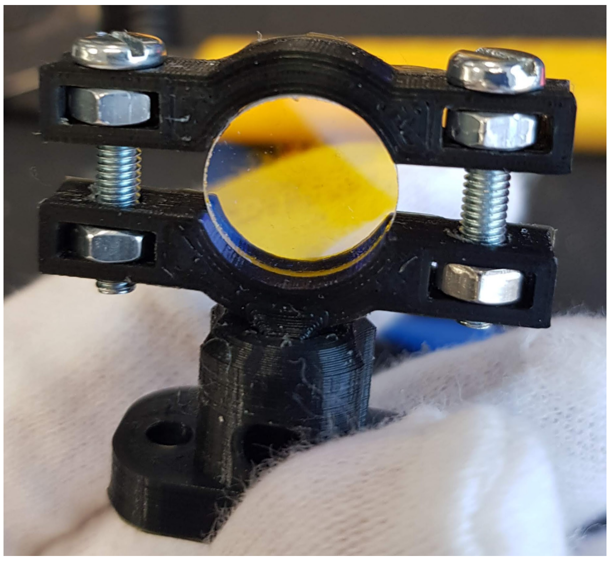


Figure 21: A dichroic mirror in its mount.

The receiver mounts to the back of a focus lens mount using extruded bridges with fastening slots. It houses the photo-diode receiver using epoxy glue.

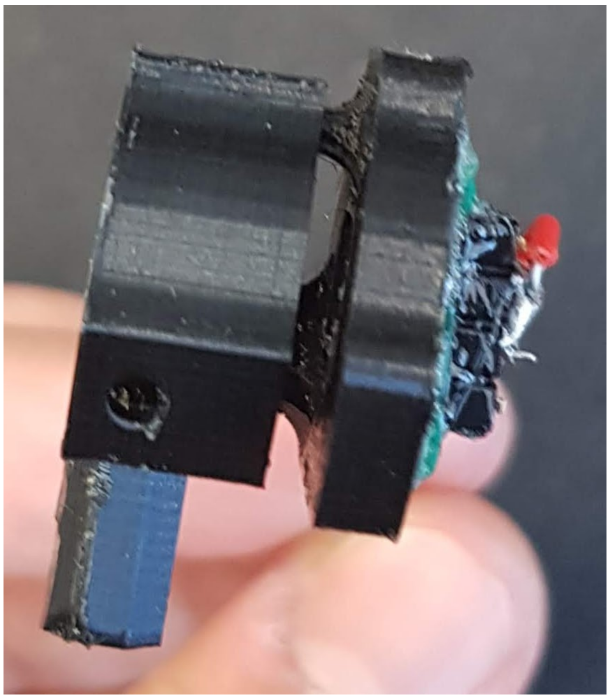


Figure 22: The receiver mount assembly.

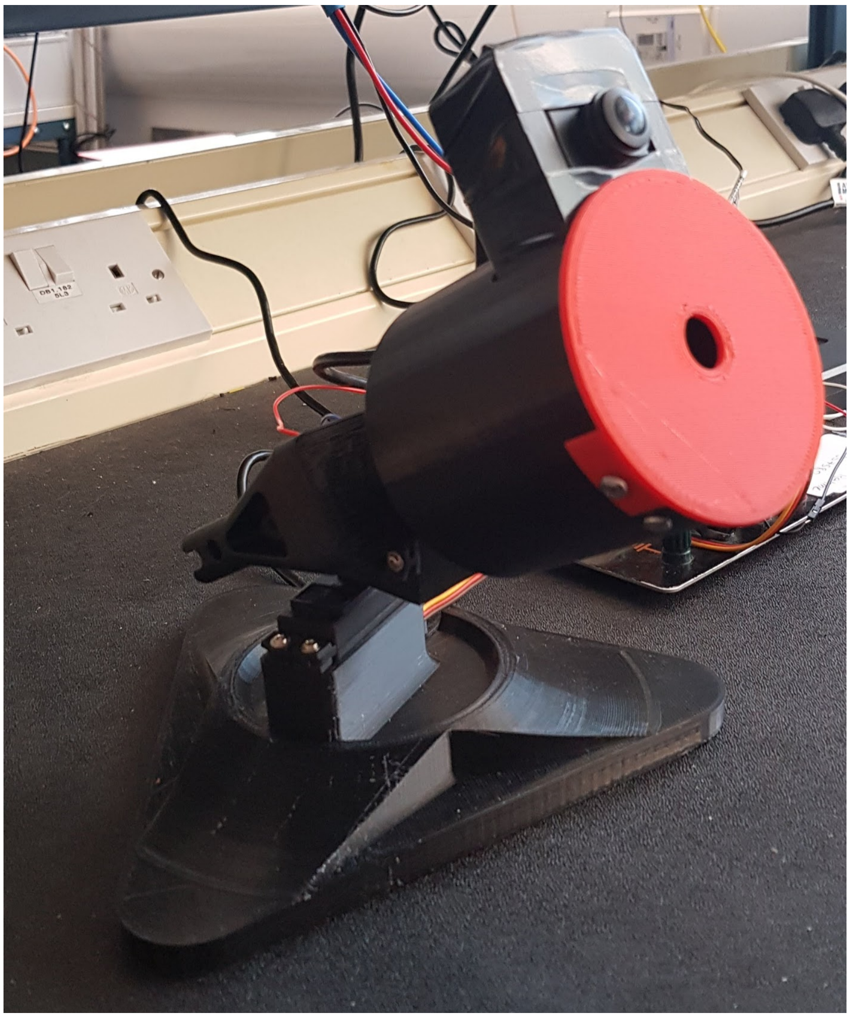


Figure 23: The full system.

4.3 Electronic Components

The myRIO interfaces on channel B are the following:

1. Pin 3, the ground pin, links to the external power supply used to power the lasers and motors.

2. Pins 27 and 29 are PWM outputs.

3. Pins 5, 7 and 9 are analogue inputs. Pins 5 and 7 measure the motor potentiometer voltages, and pin 9 measures the receiver signal.

4. Pin 11 is the digital output that controls the N-Channel MOSFET to actuate the laser.

Since the myRIO can only supply 100mA per connector, the motors, requiring up to 2.5A each, and laser are connected to an external supply [14,15]. The servos' DC motors are noisy and interfere with the position feedback measurements as these are drawn from a potentiometer instead of an encoder, which is a more expensive but more reliable sensor. To attenuate noise, the breadboard houses four $100\mu F$ shunt capacitors on the power rails [15].

The laser diode's positive terminal connects to the power supply and its negative terminal connects to the MOSFET drain. The MOFSET source connects to ground, and a pulldown resistor links its gate to ground to prevent actuation due to parasitic gate capacitance, which is instead discharged through the resistor. Upon actuating the laser, the motor position measurements swung significantly, up to ±5°. This is likely due to the MOSFET's 290pF parasitic capaticance [16]. Four $100nF$ shunt capacitors on each position feedback terminal reduce this swing to ±2°. It can be minimised using a cleaner power supply.

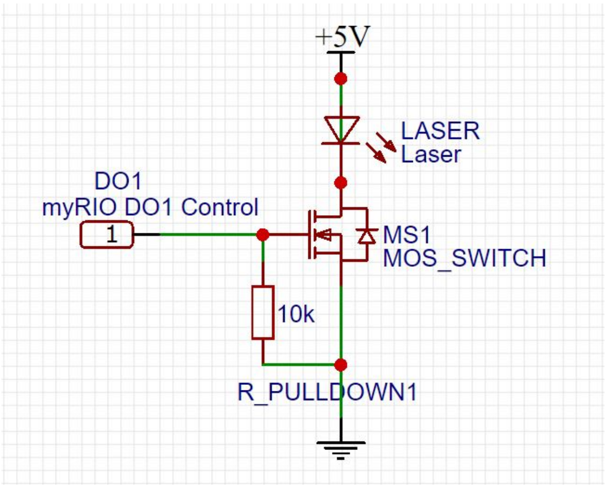


Figure 24: The Laser and MOSFET connection diagram

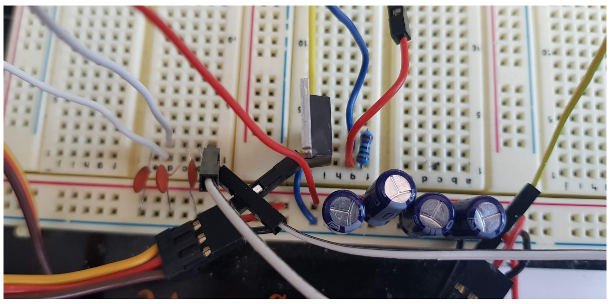


Figure 25: The breadboard connections.

4.4 Servo Motor Operation & Deconstruction for Position Feedback

The PATBLC uses two servo motors instead of stepper motors. Servo motors are faster and more accurate than stepper motors despite their reduced holding torque, which, for the Tower Pro MG996Rs, at 9.4kg/cm, is not an issue [17].

The servo motors' PI closed-loop feedback control is achieved by reading the motor's position using its on-board potentiometer which returns a voltage relating to the output shaft's position. Servo motors lack the ability to rotate up to 360°, but can re-adjust if their position slips due to high loading, whereas a stepper motor gains an offset for the current power cycle [18].

Servo Motors provide the user three terminals: the positive, negative, and Pulse-Width-Modulation (PWM) inputs. The PWM signal consists of minimum and maximum pulse widths at frequency. At 50Hz, a 1ms pulse width rotates the shaft to -90° , and a 2ms pulse width rotates it to 90 °. Hence, the motor can rotate up to 180 ° and its neutral position is at a 1.5ms. These numbers are not rigid; the typical shaft rotation range is approximately 200°, and the duty cycle values for each motor are found experimentally. The calibration procedure is given in Appendix 5 and is used to find the relationship between the PWM signal and shaft command. This map is given in Appendix 7 for each motor [18].

The command for each motor depends on the target coordinates relative to the EE added to the current motor position. The only way to estimate motor positions was to assume they are equal to the commanded position. This assumption is flawed because the commanded position is not achieved instantly, and the target's position relative to the EE changes in real-time, resulting in erratic behaviour.

The solution is to modify the motors by adding an additional terminal to the potentiometer's position feedback node, which can be mapped to a shaft position. The deconstruction and modification is outlined in Appendix 6.

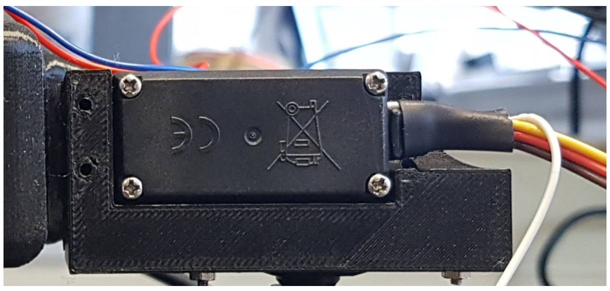


Figure 26: The added white terminal is used to measure position.

Similarly to the Duty Cycle calibration, the returned voltage from the potentiometer is mapped to an output shaft angle at 12-bit resolution, or a 1.2mV step size for a 5V signal. According to the map, 1.2mV corresponds to 1.14E-4°. The potentiometer terminal connects to pin 5 on the myRIO, and the voltage map is calculated using $y = mx + b$.

Motor 1 example:

y = Motor Position°, x = Measured Voltage,

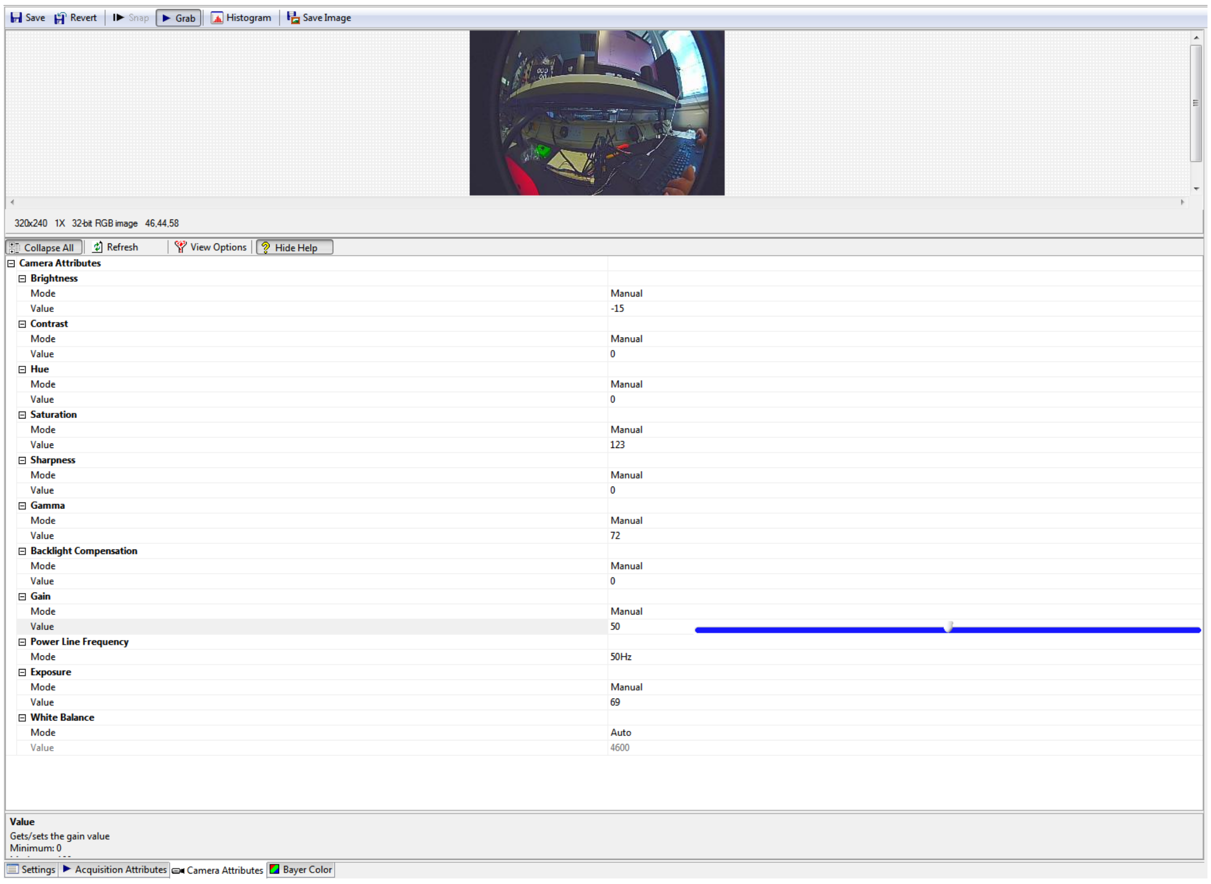
$m = \frac{180-0}{180_{VOLTS} - 0_{VOLTS}}$ and $b = -m * 0.0952148$.

The voltage maps are given in Appendix 7.

5 Computer Vision & Kalman Filtering

5.1 Camera Capture Settings

The camera was calibrated within NI MAX to ensure the red bullseye's visibility in all lighting conditions. The settings are shown in figure 27. The chosen resolution was 320x240 pixels at 30 frames-per-second (FPS). This was done to minimise computing time as the myRIO has a limited RAM of 512Mb [14].



320x240 1X 32-bit RGB image 46,44,58

Camera Attributes	
Brightness	
Mode	Manual
Value	-15
Contrast	
Mode	Manual
Value	0
Hue	
Mode	Manual
Value	0
Saturation	
Mode	Manual
Value	123
Sharpness	
Mode	Manual
Value	0
Gamma	
Mode	Manual
Value	72
Backlight Compensation	
Mode	Manual
Value	0
Gain	
Mode	Manual
Value	50
Power Line Frequency	
Mode	50Hz
Exposure	
Mode	Manual
Value	69
White Balance	
Mode	Auto
Value	4600

Value
Gets/sets the gain value
Minimum: 0

Figure 27: Camera settings.

The vision algorithm extracts the RGB colourspace within LabVIEW. LabVIEW's 'ImaqCreate' function, which creates space for image memory, works natively in RGB. An HSV extraction requires conversion from RGB, and colour-image manipulation is three times as costly as binary. The finished algorithm was found to run at 30 FPS using RGB-extraction and at 15FPS using HSV.

The camera in figure 7 is fitted upside-down for its USB 2.0 cable to exit the case. This was remedied using the 'Imaq Symmetry' function in figure 37 [19].

5.2 Computer Vision Methods

5.2.1 Gaussian Convolution Kernel

As mentioned in section 3.4, a Gaussian convolution eliminates image noise and blurs particle edges. This linear low-pass filter replaces each pixel by a weighted sum of its neighbours determined by the convolution kernel. The convolution smooths particle edges before the shape detection step, which is useful when processing 320x240 images [19].

The Gaussian kernel is:

$$\begin{bmatrix} a & d & c \\ b & \mathbf{x} & b \\ c & d & a \end{bmatrix} \tag{5.1}$$

where $\mathbf{x} > 1$ and the other elements are positive integers dependent on the kernel size. Each pixel becomes a weighted average of its neighbours, noting that higher-coefficient neighbours carry more weight. Since $\mathbf{x} > 1$, its original value has a greater effect on its weighted sum than for the Smoothing kernel, where x has a value of 0 or 1. The Gaussian kernel creates a more subtle smoothing effect which helps conserve shapes [19].

The minimum kernel size is 3x3. A larger kernel increases smoothing alongside processing time. The 3x3 kernel showed negligible reduction in performance whilst maximising processing speed [19].

5.2.2 Heywood Circularity Particle Filter

The Heywood Circularity Particle Filter (HCPF) deletes non-circular particles grouped using pixel connectivity. After RGB-thresholding, the image elements have intensity values of 0 or 1, and elements with value 1 can be grouped. This is done using connectivity modes 4 or 8 in LabVIEW. Connectivity mode 4 groups adjacent pixels into particles whereas mode 8 groups both adjacent and corner pixels of intensity value 1 [19].

In a 2D image, each pixel P_0 has eight neighbours:

$$\begin{bmatrix} P_8 & P_1 & P_2 \\ P_7 & \mathbf{P}_0 & P_3 \\ P_6 & P_5 & P_4 \end{bmatrix} \tag{5.2}$$

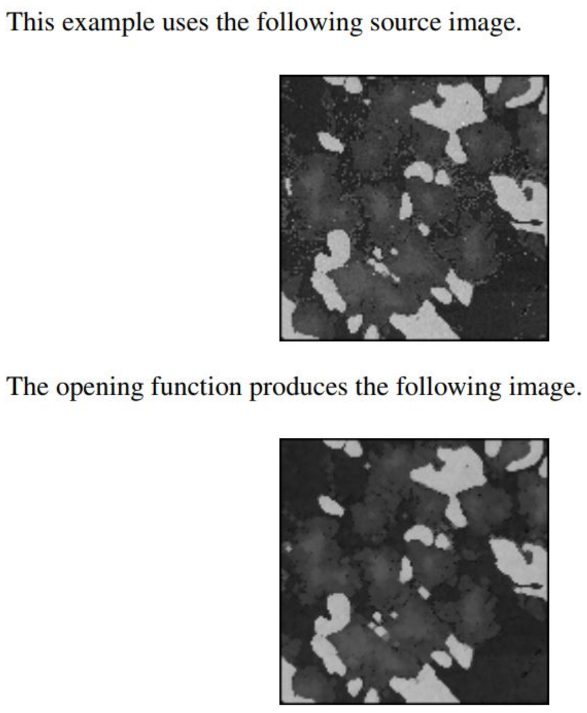
where P_0 is closer to P_1, P_3, P_5, P_7 than to P_2, P_4, P_6, P_8. It is distance D (one pixel length) away from the former, and $\sqrt{2}D$ from the latter. Connectivity mode 4 thus groups pixels of distance D from P_0 whereas connectivity mode 8 groups pixels of up to distance $\sqrt{2}D$ [19].

The HCPF assesses particle circularity using:

$$HCPF = \frac{P}{2\sqrt{\pi A}} \tag{5.3}$$

where the perimeter P in pixels is divided by the the circumference using area, A. The closer a particle is to a disk, the closer its HCPF is to 1. LabVIEW lets the user select a threshold for rejecting particles. Experimentally, to account for skew, it was set from 0 to 1.5 [19].

The filter also deletes particles that contact the image border. After unwanted particles are deleted, those remaining have their holes filled using a closing function, which is a dilation followed by an erosion that smooths particle boundaries. Since dilation and erosion are morphological complements, the particle's expansion due to dilation is reversed with erosion, but the filled holes are not removed [19].


This example uses the following source image.

The opening function produces the following image.

Figure 28: Opening function [19].

5.2.3 Small Element Elimination

Finally, an opening function is used to eliminate remnant small particles. It is an erosion followed by a dilation where the eroded borders are restored and smoothed using dilation, but the rejected particles remain lost. Two iterations are enough to maintain performance and speed. Any more eliminates the bullseye if it is too far [19].

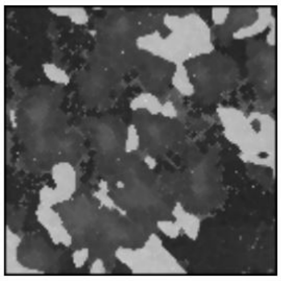
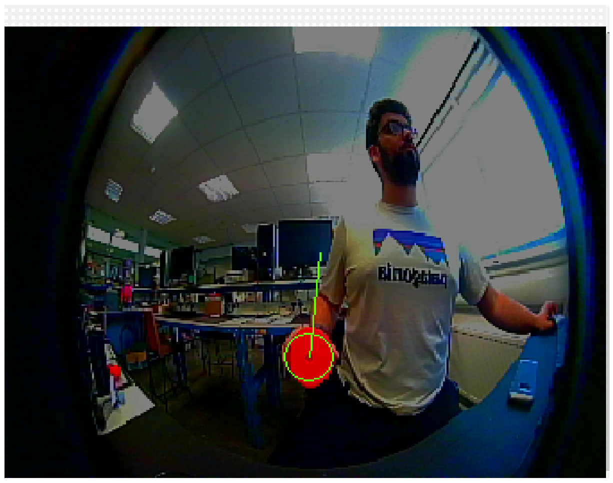


Figure 29: Closing function result [19].

5.2.4 Shape Detection

Finally, LabVIEW's Elliptical Shape Detection algorithm is used to find the bullsey's centre instead of using the OPALS' brightness-concentration method. Hence, should part of the target (up to 50%) be occluded or skewed, or should there be additional particles uncleared by image processing, the bullseye has a higher likelihood of being identified.


(a) Ideal conditions.

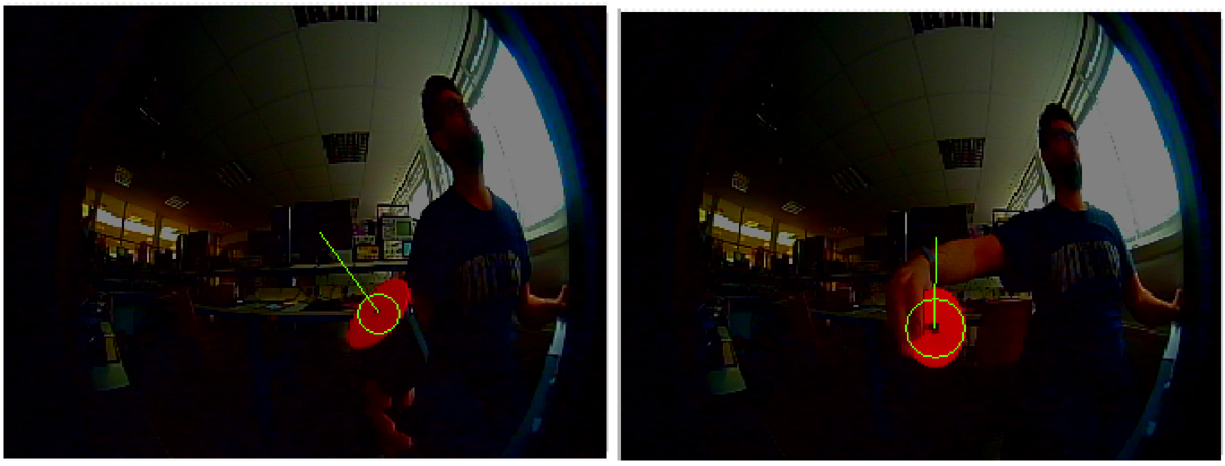

(b) Skewed in dark lighting. (c) Occluded in dark lighting.

Figure 30: Various identification scenarios.

The method employed by LabVIEW is curve extraction. National Instruments defines a curve as 'a set of edge points that are connected to form a continuous contour'. Curves are extracted in three steps [19]:

1. Finding Curve Seed Points: a curve seed cannot belong to an existing curve. Additionally, it must have a contrast value greater than the edge threshold, a function relating the examined pixel's intensity value with that of its neighbours.

   $P_{i,j}$ is the pixel intensity value P at coordinates (i, j). The edge threshold is:

   $$\sqrt{(P_{i-1,j} - P_{i+1,j})^2 + (P_{i,j-1} - Pi_{,j+1})^2} \tag{5.4}$$

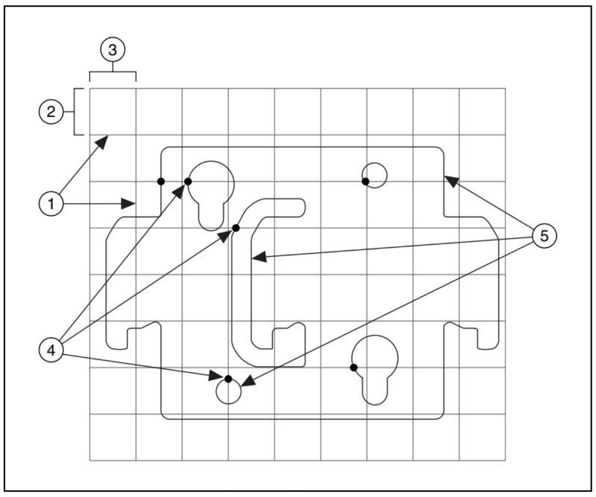
   and can vary between 0 and 360 for an 8-bit image. The image processing steps should ensure high scores.

   The number of examined pixels is limited by a user-defined step size. A higher value results in a faster algorithm that can miss smaller curves, so it must be smaller than the smallest expected curve to work well. In LabVIEW, the smallest identifiable elliptical radius is 3 pixels. Hence, a step size of 6 was chosen.

The algorithm first examines pixel $(0,0)$ for a seed point. If the above criteria are met, the curve begins tracing. Otherwise, the next pixel in the row is evaluated until the end of the row is reached. If by the end of the row, no seed is found, the algorithm skips 'step size' rows and repeats the process until the last row is reached. Then, the same process is repeated through the columns.

2. Curve Tracing: The curve is traced from the seed point by adding neighbouring pixels with the highest edge threshold - higher than the limit - to the curve. When no more pixels can be added in one direction, the process is reiterated in another. When the tracing is completed, a new seed point is sought out.

3. Refinement: After all curves are traced, the ones with ends in close proximity considering 50% occlusion are combined. Open curves are closed if the result meets the minimum curve size of 3 pixels. Otherwise, they are deleted.



1 Scan Lines	3 Column Step	5 Curves
2 Row Step	4 Curve Seeds	

Figure 31: Curve Extraction [19].

The closed curve is compared to an elliptical template with a minimum radius size of 3 pixels and a maximum of 160 for the major radius and 120 for the minor radius; half the 320x240 resolution. The template is defined by:

$$Area(A) = \pi ab \tag{5.5}$$

and

$$Perimeter(P) = \pi\sqrt{2(a^2+b^2)} \tag{5.6}$$

where a and b are half the total length of the major and minor axes of ellipses with the same area and perimeter as a particle. $a = \frac{E_{2a}}{2}$ and $b = \frac{E_{2b}}{2}$, where E_{2a} and E_{2b} are the Equivalent Ellipse Major and Minor Axes [19].

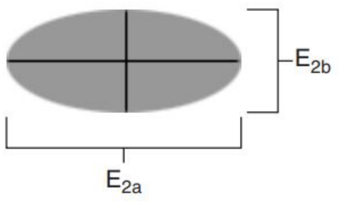


Figure 32: Elliptical template.

Rearranging for E_{2a} and E_{2b}:

$$E_{2a} = \sqrt{\frac{P^2}{2\pi^2} + \frac{2A}{\pi}} + \sqrt{\frac{P^2}{2\pi^2} - \frac{2A}{\pi}} \tag{5.7}$$

$$E_{2b} = \sqrt{\frac{P^2}{2\pi^2} + \frac{2A}{\pi}} - \sqrt{\frac{P^2}{2\pi^2} - \frac{2A}{\pi}} \tag{5.8}$$

The matching process is comprised of three steps:

1. Feature Correspondence Matching: High-level geometric features are identified from the extracted curves using polygon approximations. These are ordered by feature type, - i.e circle - feature representation strength, and template representation strength. Then, potential matches from the image are classed based on these criteria. Their scale, rotation and position is noted.

2. Template Model Matching: The template is superimposed onto the potential match for confirmation. If the match is confirmed, additional features such as scale and rotation are compared with the template for a more detailed match.

3. Match Refinement: Further matching is carried out for precise positional, scalar and angular information. Curves extracted from the inspected image are compared to the template's curves to ensure accurate matching before returning a match score, which is an average of the scores for each matched feature.

The user-defined match score threshold ranges from 0 to 1000, with 1000 indicating a perfect match. A higher threshold means fewer potential matches are carried to steps 2 and 3 above, reducing processing time, but increasing the likelihood of missing the bullseye. The target was placed in non-ideal conditions, and a match score threshold of 800 was derived. The bullseye should be the highest-scoring match, so the match coordinates are zero-indexed [19].

Remnant particles can classify as false matches and increase processing time. Multi-step functions such as HCPF hence increased the processing rate from 8FPS to 15FPS. By converting from HSV to RGB extraction, this rate was further increased to 30FPS.

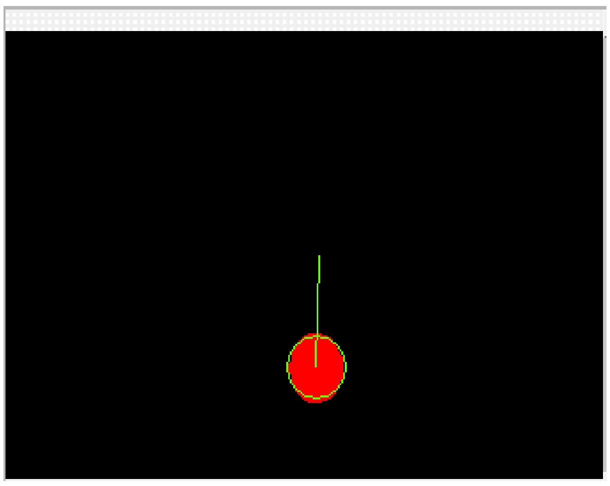


Figure 33: Resultant image after processing and detection.

5.2.5 Pixel to Spherical Coordinates

The camera used here is an ELP-USBFHD01M-L180 and has a spherical 'fisheye' lens with a 180° FOV. The captured image is distorted, and real-world coordinates cannot be mapped without manipulation.

According to Dr. Paul Bourke, assuming two identical spherical lens halves; a good assumption considering the glass-spun manufacturing [20].

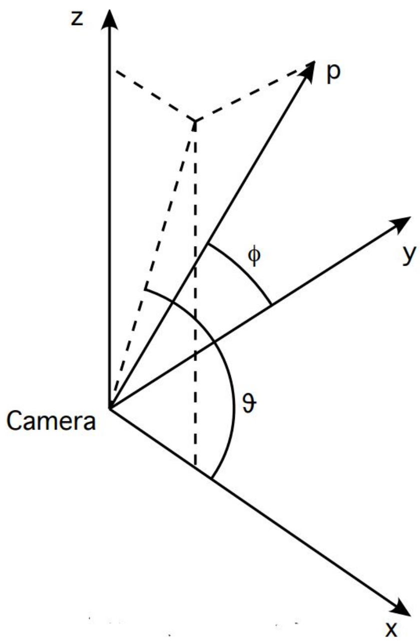


Figure 34: Real-world coordinates [21].

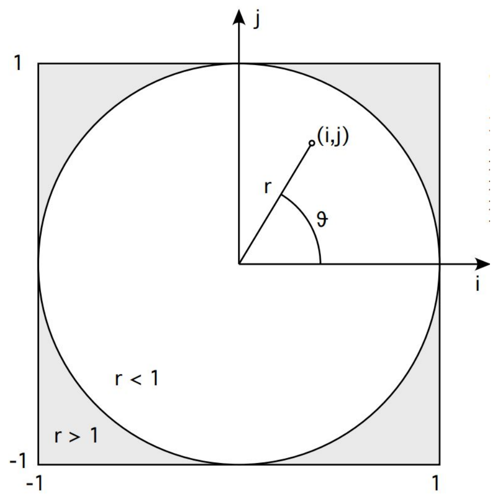


Figure 35: Image coordinates [21].

Here, $r = 1$, ϕ is elevation and θ is azimuth in real-world coordinates and i and j are normalised pixel distances from the centre of the fisheye image. The translation from pixel to spherical coordinates is as follows [21]:

$$\phi = \arctan(\frac{\sqrt{p_x^2 + p_z^2}}{p_y}) \tag{5.9}$$

where p_y is found using a calibration map for radius-size-to-distance in Appendix 7.

Additionally,

$$\theta = (i/j) \tag{5.10}$$

and

$$p_x = cos\theta * sin(\frac{\pi i}{2cos\theta}) \tag{5.11}$$

and

$$p_z = sin\theta * sin(\frac{\pi i}{2cos\theta}) \tag{5.12}$$

These can be used to find the bullseye's spherical coordinates relative to the camera, which are added to the camera' elevation offset to find the result relative to the EE [21].

The camera's offset from the EE is calculated using $\phi_{OFFSET} = (\frac{y}{x})$ where y is the vertical distance from the centre of the EE to the centre of the camera, and x is the horizontal distance from the edge of the lens to the system's centre of rotation.

5.3 Image Processing in LabVIEW

The 'RGB Process' VI conducts image processing and shape detection. The constants and controls passed to each function - performed in connectivity mode 8 - correspond to the values from section 5.2.

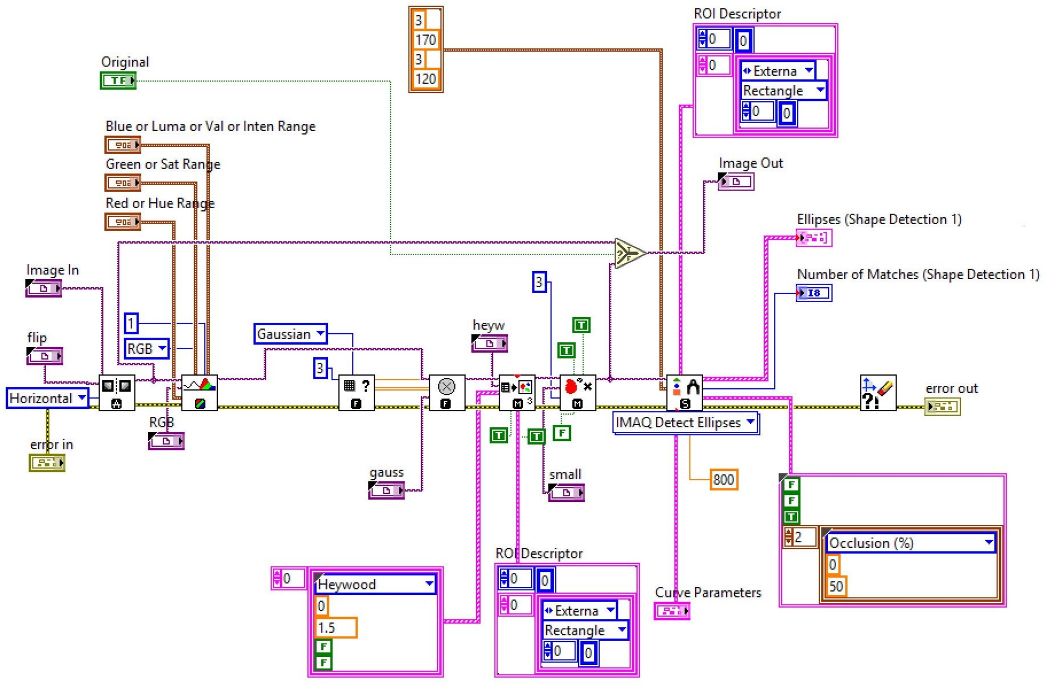


Figure 36: The 'RGB Process' VI.

The 'Image Dst' inputs reference memory where the destination image after each processing step is stored. The memory type reflects the post-processed image format. Memory is established on first-run using the 'ImaqCreate' function to reduce processing time by avoiding a re-initialisation in every loop as seen in figure 58.

The 'Pixel to Spherical Coordinates' VI converts the KF pixel coordinates to azimuth and elevation using the equations in section 5.2.5. The Y_{EST} coordinate is divided by 1.35 to normalise the vertical distortion discrepancy identified in the 320x240 format. This was calibrated by laying a circle parallel to the camera. Additionally, the azimuth coordinate is negated if elevation is above 90°. Azimuth is also limited to 360 degree using a modulo operation.

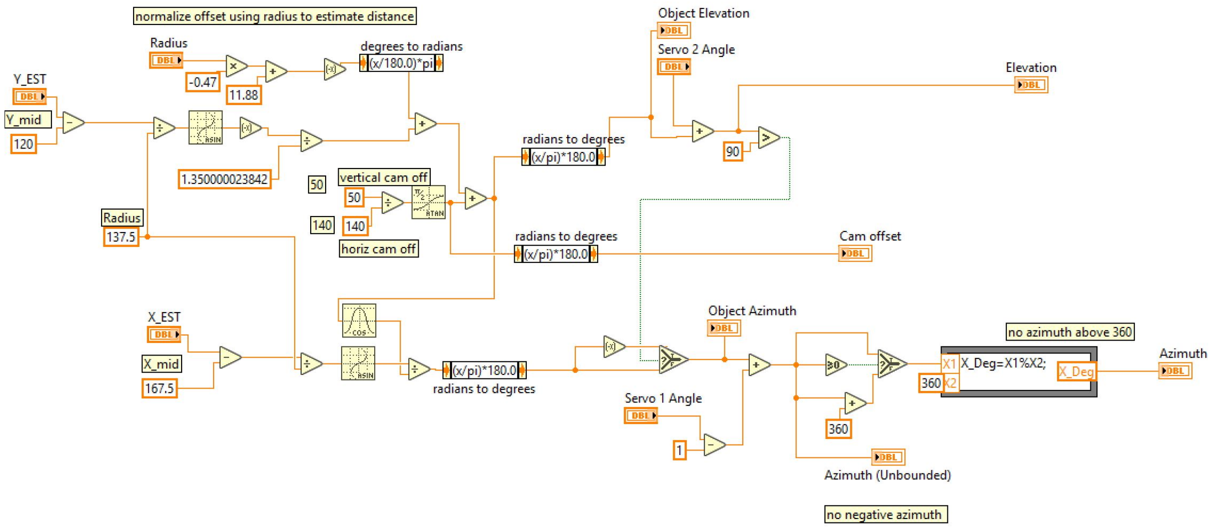


Figure 37: The 'Pixel to Spherical Coordinates' VI.

5.4 Kalman Filter

5.4.1 Sate-Space Modeling

A state-space model is crucial to the implementation of a KF. Here, the equation of motion of a free particle is used. A state-space model is modular and allows element separation to simplify multiple-input-multiple-output (MIMO) systems. In the future, the model should be upgraded to MIMO by implementing the SquidROV's equations of motion (EOM) as well as those of the hub-mounted PATBLC. During the interim meeting, Professor Kamenev highlighted the challenges posed by the latter PATBLC bobbing in the water. Measurements from the myRIO's internal measurement unit (IMU) can be subsumed into a single model using sensor fusion with the SquidROV's EOM to tackle the issue [22].

Since the target's coordinates are x and y pixels, its next-step motion in each direction can be described using:

$$x_t = x_{t-1} + \dot{x}_{t-1}t + \frac{1}{2}at^2 \tag{5.13}$$

$$\dot{x}_t = \dot{x}_{t-1} + at \tag{5.14}$$

The same equations is applied to vertical motion, where t is the discrete KF's time step of 34ms, equal to the task loop time in section 7.3.

A combined state-space model is constructed for equation (3.9). It has a 4-dimensional state vector for horizontal and vertical position and velocity [23]:

$$\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} \frac{t^2}{2} \\ \frac{t^2}{2} \\ t \\ t \end{bmatrix} \mu + \varepsilon_x \tag{5.15}$$

where the process error is

$$\varepsilon_x = \begin{bmatrix} \frac{t^4}{4} & 0 & \frac{t^3}{2} & 0 \\ 0 & \frac{t^4}{4} & 0 & \frac{t^3}{2} \\ \frac{t^3}{2} & 0 & t^2 & 0 \\ 0 & \frac{t^3}{2} & 0 & t^2 \end{bmatrix} \tag{5.16}$$

and is derived from the system's input by multiplying each element [23]:

$$\begin{array}{c|cccc|} & \mathrm{x} & \mathrm{y} & \dot{x} & \dot{y} \\ \mathrm{x} & & & & \\ \mathrm{y} & & & & \\ \dot{x} & & & & \\ \dot{y} & & & & \end{array} \tag{5.17}$$

The sensor model's state-matrix equivalent to equation 3.11, is described noting that the camera only measures position:

$$\bar{z}_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \bar{x}_t + \varepsilon_z \tag{5.18}$$

where the sensor error is

$$\varepsilon_z = \begin{bmatrix} \sigma_x & 0 \\ 0 & \sigma_y \end{bmatrix} \tag{5.19}$$

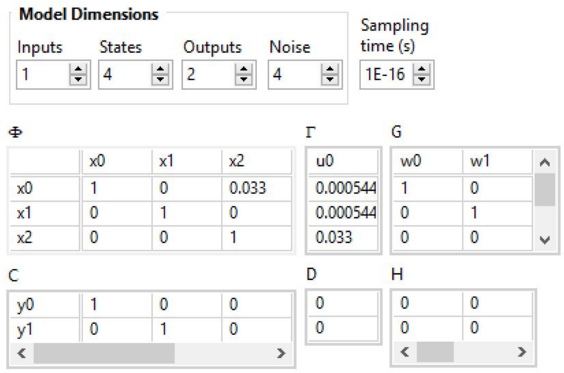
and σ_x and σ_y are equal to the variance of a set of sensor measurements [5]. Referring the USB-provided sheet 'Sensor Noise.xlsx', it was found as 8.95E-5 using the variance equation:

$$\sigma^2 = \frac{\Sigma(X - \mu_x)^2}{N} \tag{5.20}$$

where σ^2 is variance, X is the sample, μ_x is the data-set's mean, and N is its size. Equations 5.15 to 5.19 are subsequently implemented in the Kalman Algorithm outlined in section 3.4.

5.4.2 Kalman Filtering in LabVIEW

In the 'Discrete Kalman Filter' block, the user configures the system state-space model, including process and sensor noise, along with the intial system estimate, which is the centre of the camera's FOV.



Model Dimensions

Inputs	States	Outputs	Noise
1	4	2	4

Sampling time (s): 1E-16

Φ

	x0	x1	x2
x0	1	0	0.033
x1	0	1	0
x2	0	0	1

Γ

u0
0.000544
0.000544
0.033

G

w0	w1
1	0
0	1
0	0

C

y0	1	0	0
y1	0	1	0

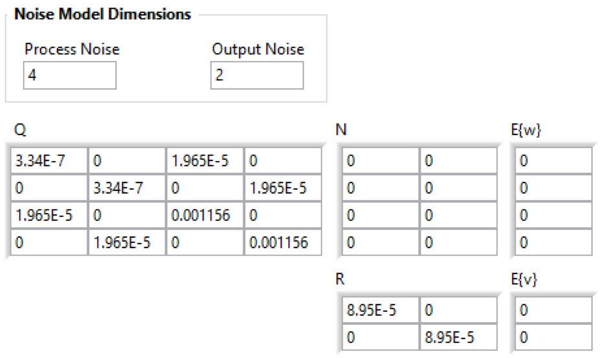
D

0
0

H

0	0
0	0

(a) State-Space Model.



Noise Model Dimensions

Process Noise	Output Noise
4	2

Q

3.34E-7	0	1.965E-5	0
0	3.34E-7	0	1.965E-5
1.965E-5	0	0.001156	0
0	1.965E-5	0	0.001156

N

0	0
0	0
0	0
0	0

E{w}

0
0
0
0

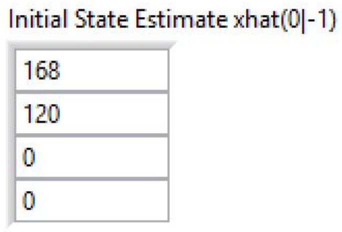
R

8.95E-5	0
0	8.95E-5

E{v}

0
0

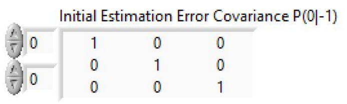
(b) Noise Matrices.



Initial State Estimate xhat(0|-1)

168
120
0
0

(c) Initial state estimate



Initial Estimation Error Covariance P(0|-1)

1	0	0
0	1	0
0	0	1

(d) Initial state co-variance matrix

Figure 38: Discrete Kalman Filter parameters in LabVIEW

If no object is detected, the shape detection code returns (0,0). This operation skews the KF's estimations, so a feedback node in the 'Overlay Circle' VI in figure 60 inputs the last-estimated state back to the KF block in this case, allowing occlusion-resistant motion estimation.

6 Kinematics & Control

6.1 Inverse Kinematics in LabVIEW

The process of verifying the IK's system of 9 equations with gradient-descent for joint limitations using LabVIEW's robotics toolbox was costly alongside computer vision, causing significant lag. As the system only has 2DOF, it was possible to map IK heuristically to derive a quadrant-based relationship [8,9].

The 'Quadrant Selector' VI accepts azimuth and elevation inputs between 0°-360° and 0°-180° from the 'Pixel to Spherical Coordinates' VI. At the $(0°, 0°)$ position, should the target be to the right of the EE, then the system should point to somewhere below 360°in azimuth. For elevation, allowing a range up to 180°provides an alternate route for reaching the second quadrant.

The quadrant selector evaluates azimuth and elevation to determine which of the four quadrants the system should operate in. 'Select' functions are fed booleans determined by 'AND' operators with range conditions. If a set of azimuth and elevation values is identified in-range, an integer related to a quadrant is passed. Otherwise, a value of zero is passed. All four 'Select' outputs are summed and passed out of the VI to the case selector that maps the joints.

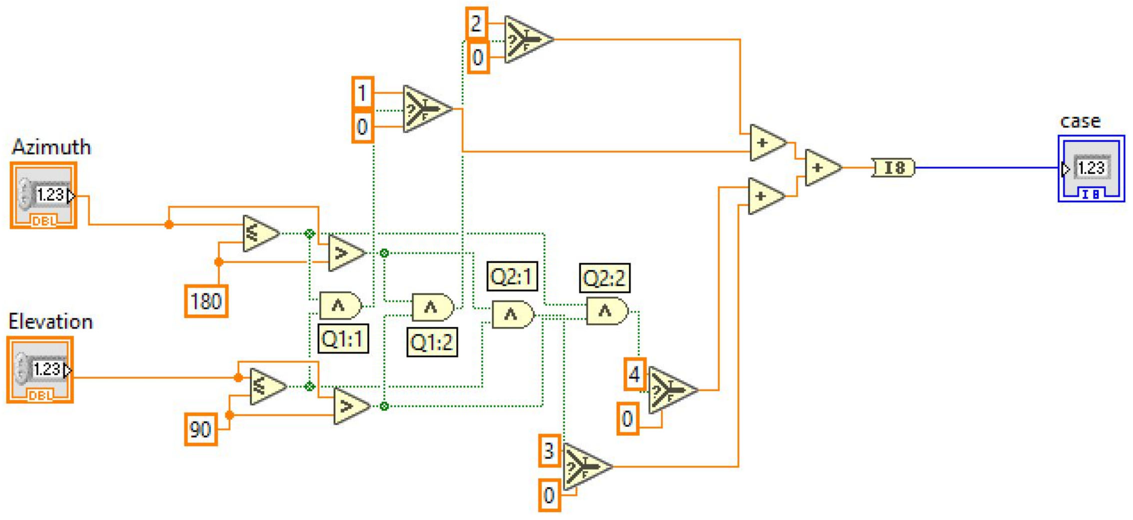


Figure 39: 'Quarant Selector' VI.

In the case selector, each quadrant commands the motors as follows:

1. Quadrant 1:1
   Azimuth <= 180°, Elevation <= 90°
   Azimuth and elevation are passed directly to motors 1 and 2 respectively.

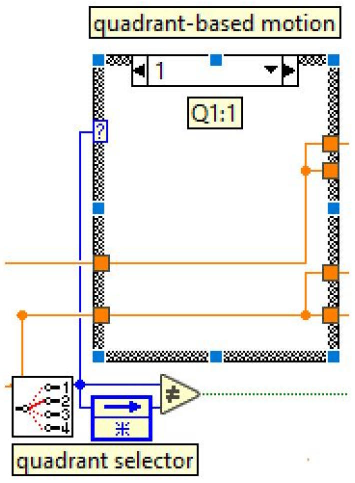


Figure 40: Quadrant 1:1.

2. Quadrant: 1:2
   Azimuth >180°, Elevation >90°
   An attempt to return to quadrant 1:1 is made by subtracting 180° from azimuth and subtracting elevation from 180° and passing the modified values to motors 1 and 2.

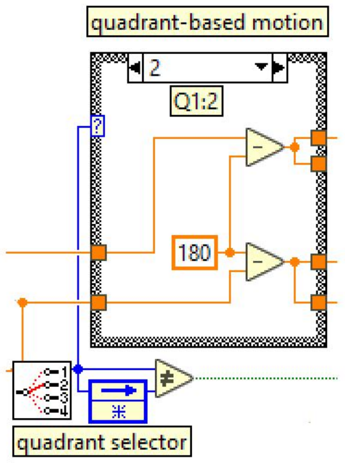


Figure 41: Quadrant 1:2.

3. Quadrant 2:1
   Azimuth >180°, Elevation <= 90°
   The EE and camera flip to continue tracking the target. this is done by subtracting 180° from azimuth and subtracting elevation from 180°.

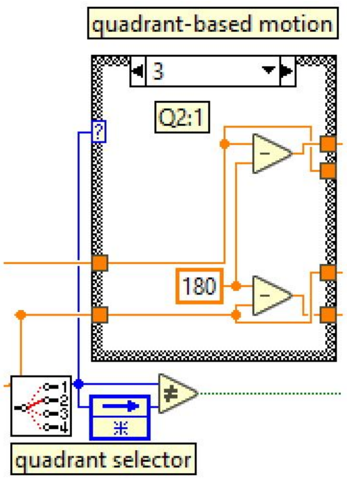


Figure 42: Quadrant 2:1.

4. Quadrant 2:2
   Azimuth <= 180°, Elevation >90°
   Commands are passed directly to the motors since the 'Pixel to Spherical Coordinates' VI negates azimuth if elevation exceeds 90°to ensure the correct path. The value manipulation is to optionally display coordinates in a user-friendly format.

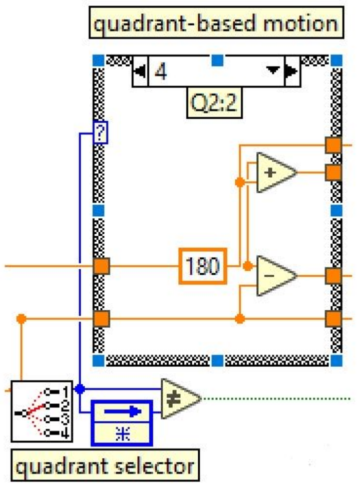


Figure 43: Quadrant 2:2.

Quadrant 2 can be entered from quadrant 1 through azimuth exceeding 180°(2:1) or through elevation exceeding 180°(2:2), and vice-versa.

6.2 Easing Algorithm

A motor easing algorithm was employed to avoid toppling the system during edge cases as the purchased motors produce a minimum torque of 9.4kg/cm. They were the only motors available which supplied enough torque at cost.

The easing algorithm used here, is an 'Ease-Out' algorithm [24]. It reduces the motor's positional rate of change the closer it is to its target. The change-magnitude is determined by a filter in the following equation based on the time-dependent Quadratic Ease-Out function:

$$Setpoint = Current * (1 - Filter) + (Target * Filter) \tag{6.1}$$

This low pass filter adaptation uses current versus target position instead since it iteration-based. The filter can be set between 1 (no filter) and 0.001. Experimentally, 0.4 was chosen [25].

Within LabVIEW, this is implemented using the identical 'Smooth Pan' and 'Smooth Tilt' VIs pictured below.

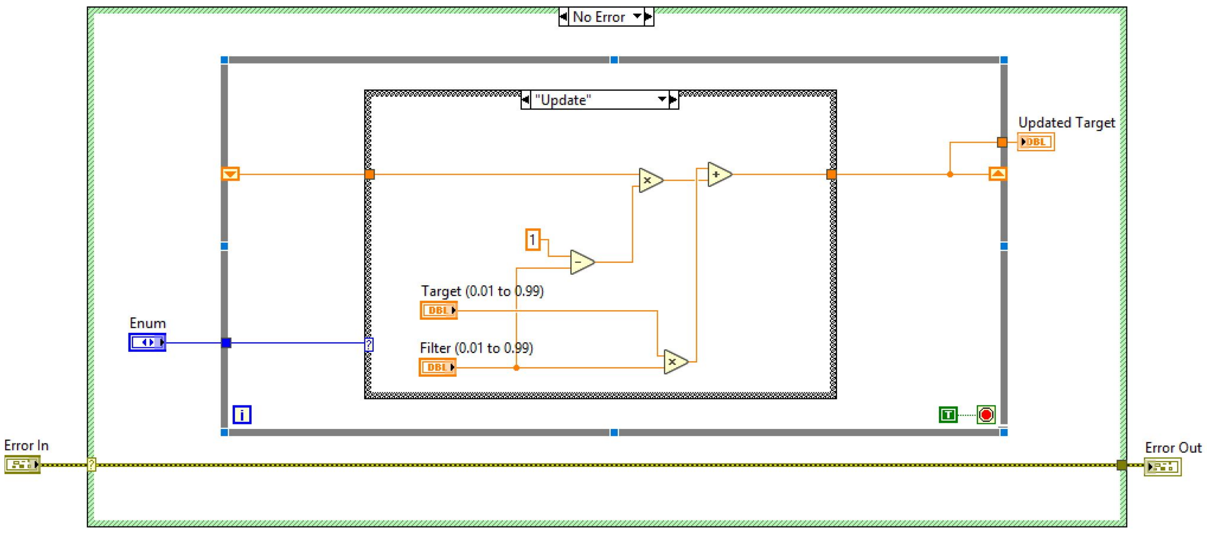


Figure 44: 'Smooth' VI.

7 LabVIEW

7.1 Graphical User Interface

The Graphical User Interface (GUI) on the PC Host shown below.

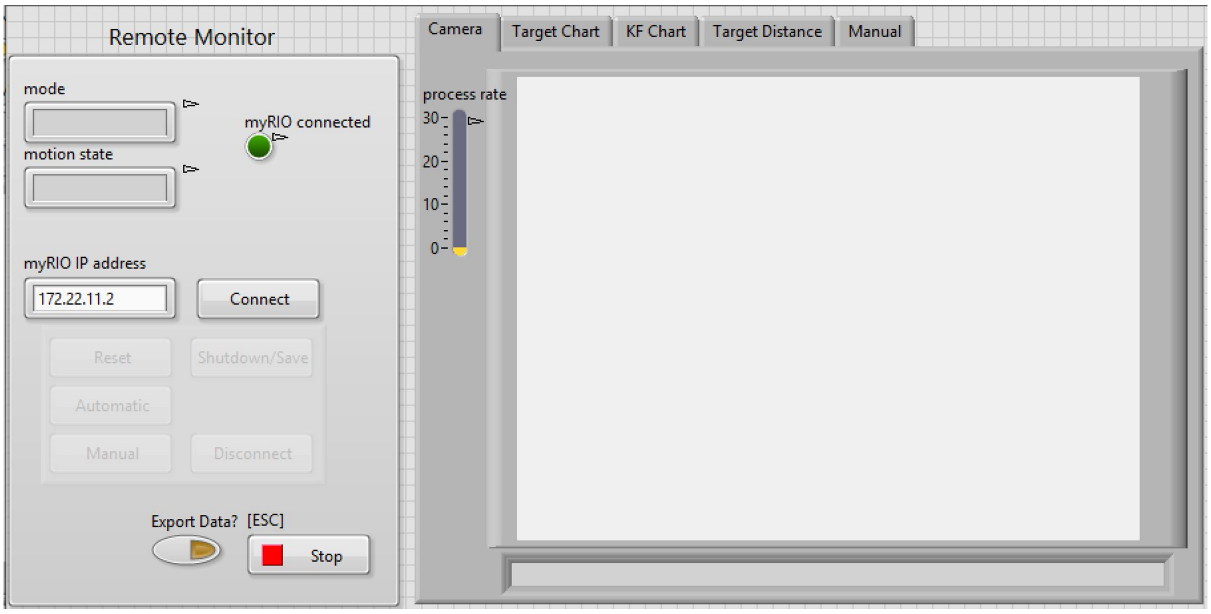


Figure 45: GUI.

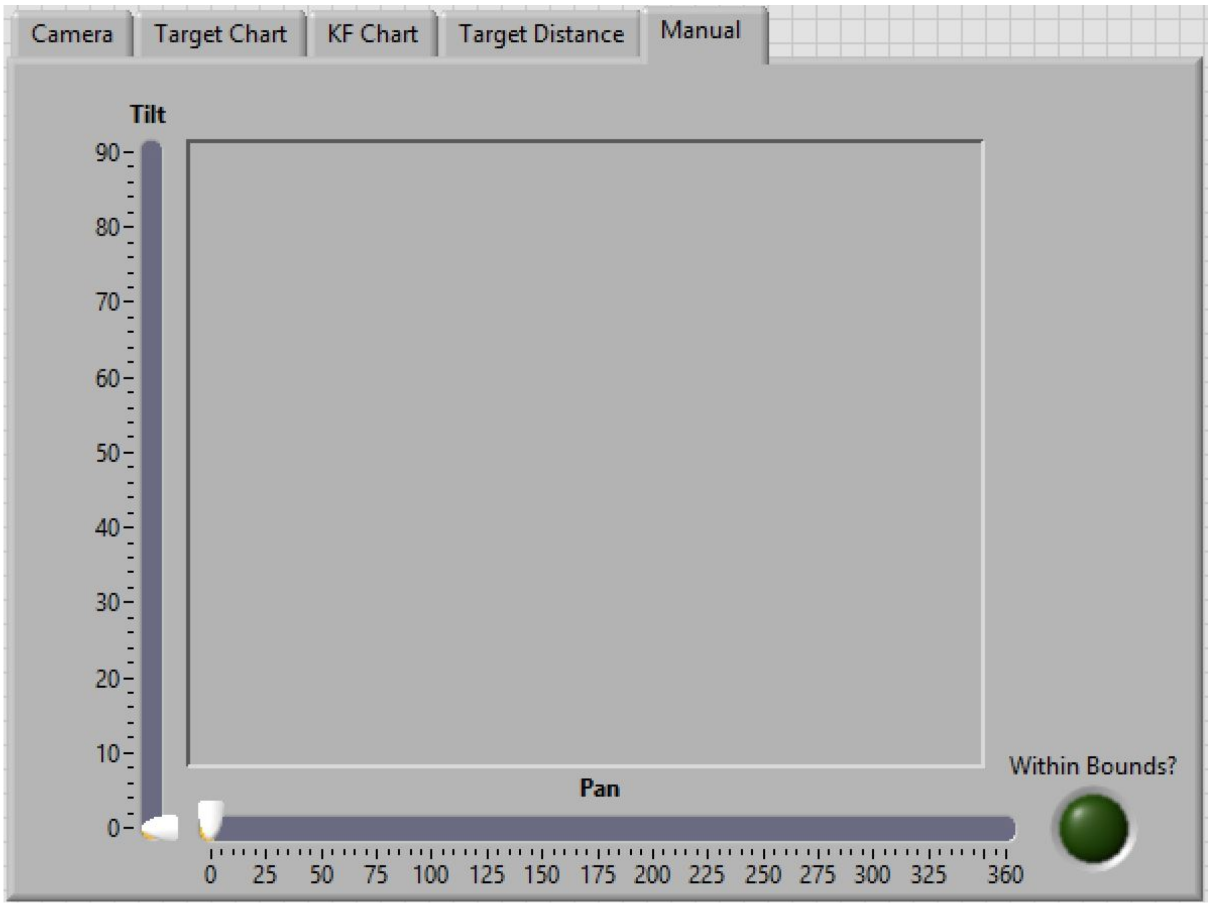


Figure 46: The Manual control tab explained in Appendix 9.

7.1.1 Consumer/Producer to Command Parser

The 'PC Host' VI uses a Producer-Consumer architecture [26]. The Producer Loop houses an event-case-structure for each front panel button followed by a guard clause that priority-enqueues the 'Shutdown' state using 'Enqueue at Opposite End' in case of an error. Queue operations are detailed in section 7.2. In the Consumer Loop, the guard clause dequeues the Producer command if there is no error, and passes the 'Shutdown' state otherwise to stop all task loops. The 'Shutdown' state is also enqueued if the 'ESC' button is pressed. In the Consumer, the 'Shutdown' state triggers a virtual press of the 'ESC' stop button to stop the Producer Loop and sets the stop boolean for its task loop to true. When both loops are stopped, the Queue is destroyed and a 'Simple Error Handler' function ensues.

The 'Connect' event in the Producer enqueues the 'connect' state to the Consumer to call the 'Create Network Stream Reader Endpoint' function with a 500ms timeout. If there is no error, meaning that a connection has been made using the myRIO IP address, the 'Connect' button is grayed-out and disabled and the Tab housing the other buttons is enabled using invoke nodes. If the timeout is exceeded, an error is injected into the error cluster [26].

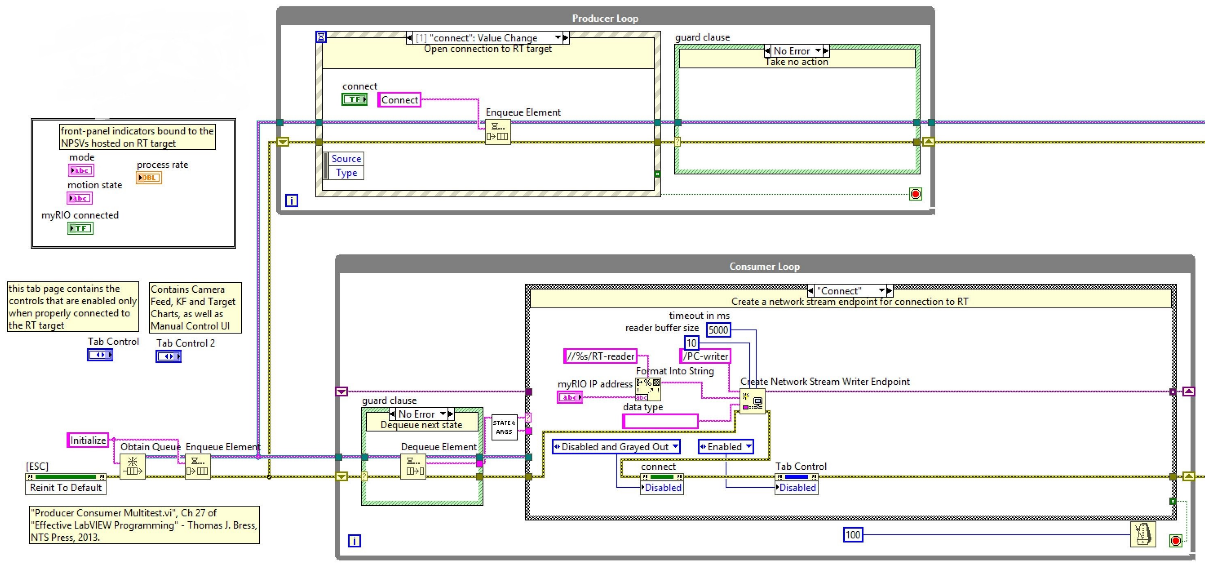

Producer Loop
[1] "connect": Value Change
Open connection to RT target
connect
Connect
Enqueue Element
Source
Type
guard clause
No Error
Take no action
front-panel indicators bound to the NPSVs hosted on RT target
mode
process rate
motion state
myRIO connected
Consumer Loop
"Connect"
Create a network stream endpoint for connection to RT
timeout in ms
reader buffer size
5000
10
//%s/RT-reader
/PC-writer
Format Into String
myRIO IP address
Create Network Stream Writer Endpoint
data type
this tab page contains the controls that are enabled only when properly connected to the RT target
Contains Camera Feed, KF and Target Charts, as well as Manual Control UI
Tab Control
Tab Control 2
guard clause
No Error
Dequeue next state
Dequeue Element
STATE & ARGS
Disabled and Grayed Out
Enabled
connect
Tab Control
Disabled
Disabled
Initialize
Obtain Queue
Enqueue Element
[ESC]
Reinit To Default
"Producer Consumer Multitest.vi", Ch 27 of "Effective LabVIEW Programming" - Thomas J. Bress, NTS Press, 2013.
100

Figure 47: PC Host 'Connect' state.

Similarly, the 'Disconnect' event triggers the 'disconnect' state in the Consumer to send the 'Disconnect' to the RT Target by flushing the stream to ensure passage. Then, the 'Destroy Stream Endpoint' function ends communication. Finally, the GUI button Tab is grayed-out and disabled, and the 'Connect' button is enabled [26].

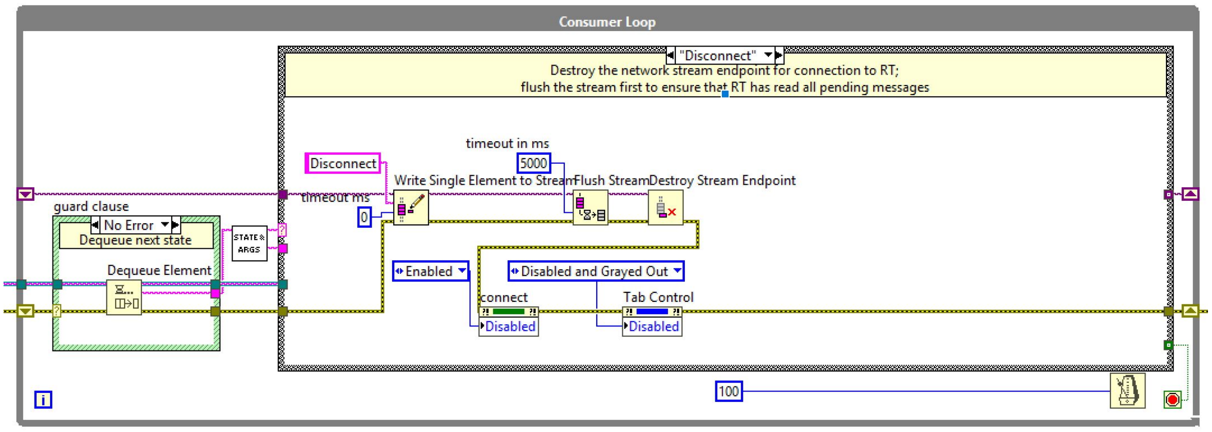


Figure 48: PC Host 'Disconnect' state.

The 'Consumer's 'Send' state has a driving Producer event for each front panel button. If a button is pressed after connecting to the myRIO, the 'Send:Message' string is enqueued. For example, pressing the 'Reset' button enqueues 'Send:SysMgr/Intialize'. As explained in section 7.2, the ':' delimeter splits the string into the 'Send' Consumer state and the 'SysMgr/Intialize' argument. The 'Send' case sends this argument to the RT Target using 'Write Single Element to Stream' [26].

Within 'RT Main', the Command Parser kernel moves from its 'Initialize' state to the 'wait for connection' state where it calls 'Create Network Stream Reader Endpoint'. If a connection is established, the true case-selector state is engaged, saving the network stream reader endpoint to the data highway and enabling the 'PC Host Connected' boolean to enqueue the 'connected' state. If the timeout is exceeded, the flase case is selected, enqueuing the 'wait for connection' state again and clearing the generated error to avoid the shutdown sequence in 7.4.2 [26].

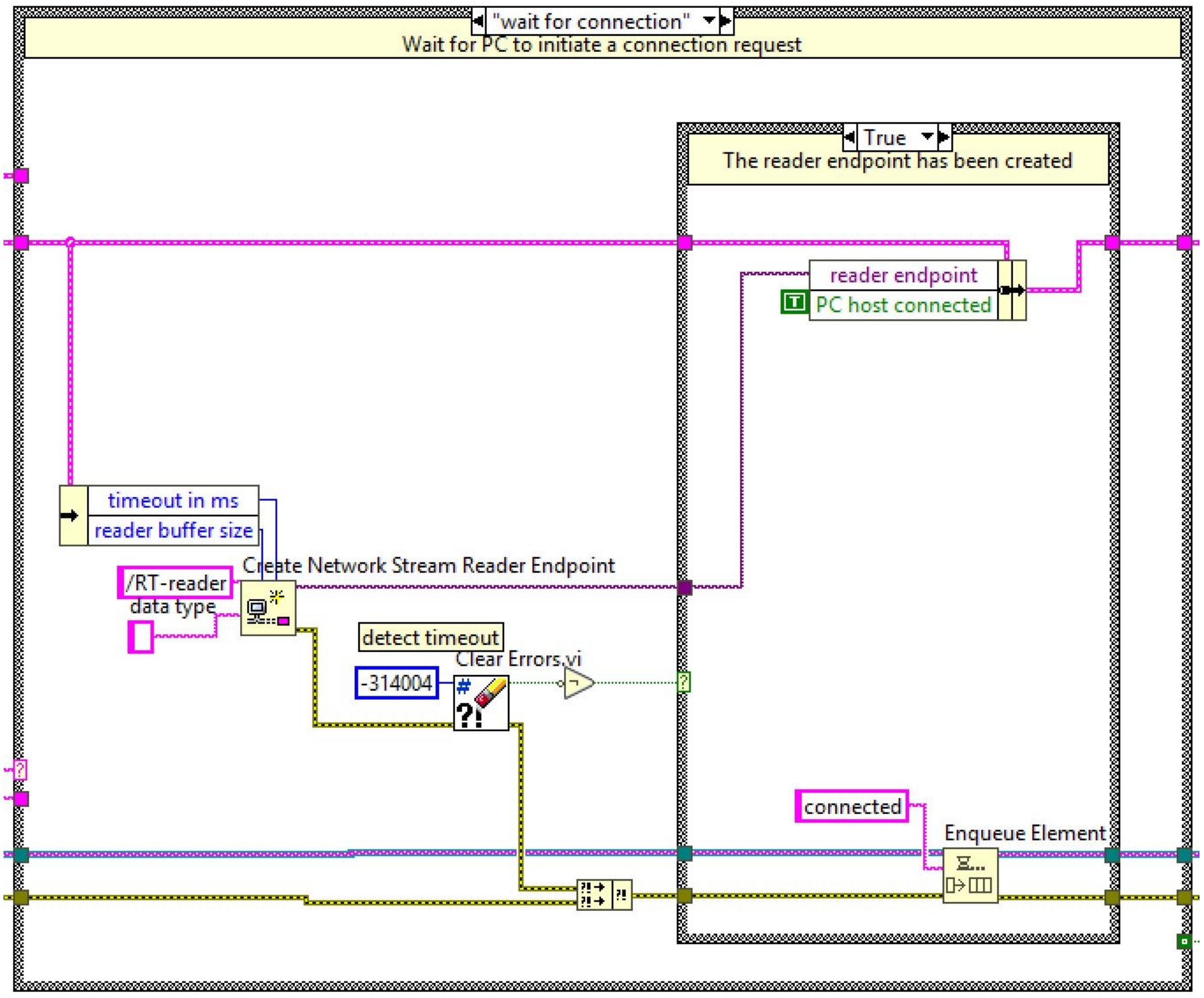


Figure 49: RT 'wait for connection' state.

In the 'connected' state, messages from the PC Host are read using 'Read Single Element From Stream'. If the timeout is exceeded, no message has been received and the error is cleared before the 'connected' state is enqueued. Otherwise, the received message is checked to see if it is 'Disconnect', in which case the 'disconnect' state is enqueued. Otherwise, the 'connected' state is enqueued and the received message is passed using the 'Enqueue Single Message' VI discussed later in this section. The 'SysMgr/Initialize' message is split into the 'SysMgr' queue and the 'Initialize' message using the '/' delimeter [26].

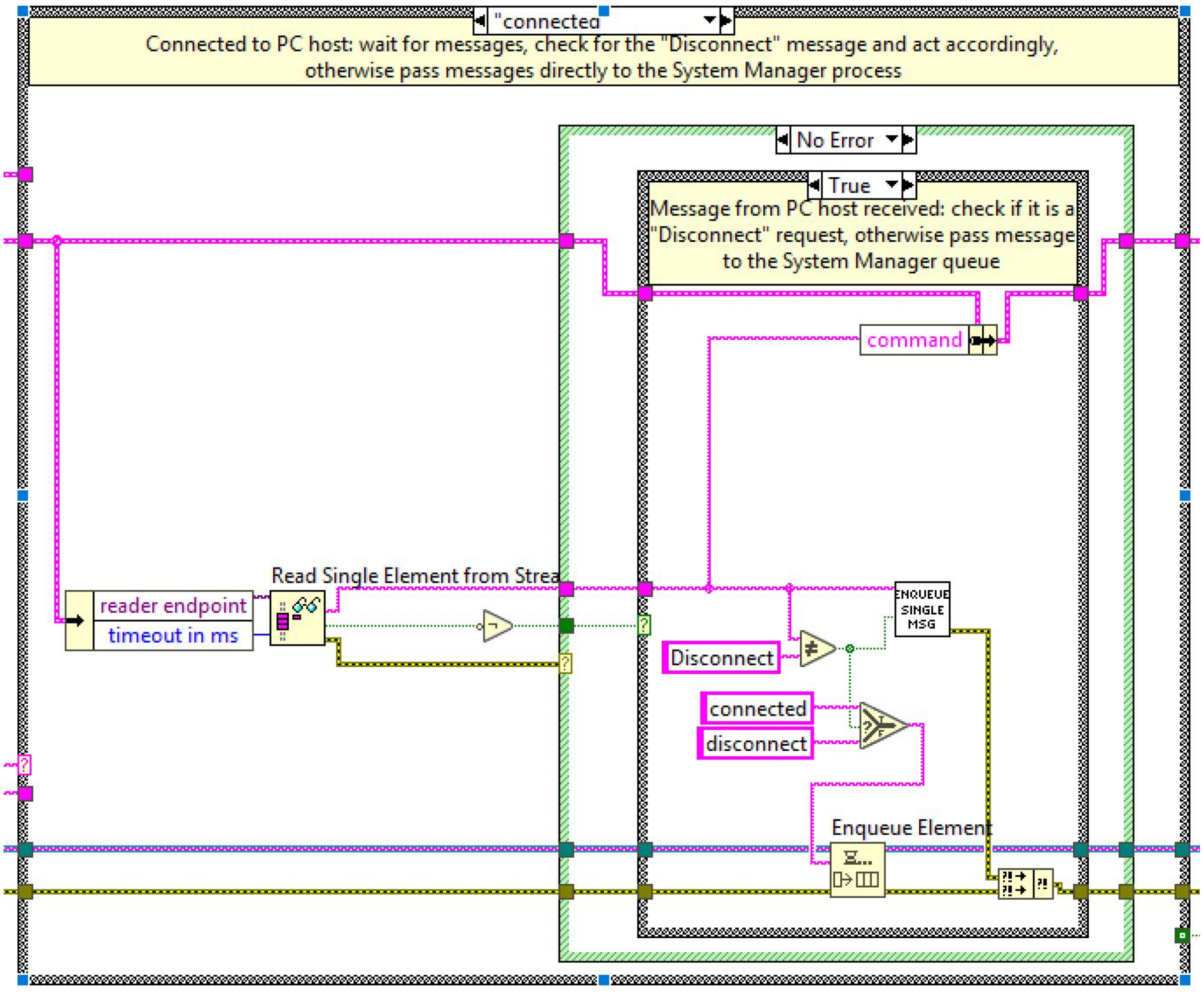

"connected"
Connected to PC host: wait for messages, check for the "Disconnect" message and act accordingly, otherwise pass messages directly to the System Manager process
No Error
True
Message from PC host received: check if it is a "Disconnect" request, otherwise pass message to the System Manager queue
command
Read Single Element from Strea
reader endpoint
timeout in ms
ENQUEUE SINGLE MSG
Disconnect
connected
disconnect
Enqueue Element

Figure 50: RT 'connected' state.

In the 'disconnect' state, the 'Destroy Stream Endpoint' function is called, the reader endpoint is emptied, and the 'PC Host Connected' boolean is set to false before enqueueing the 'wait for connection' state [26].

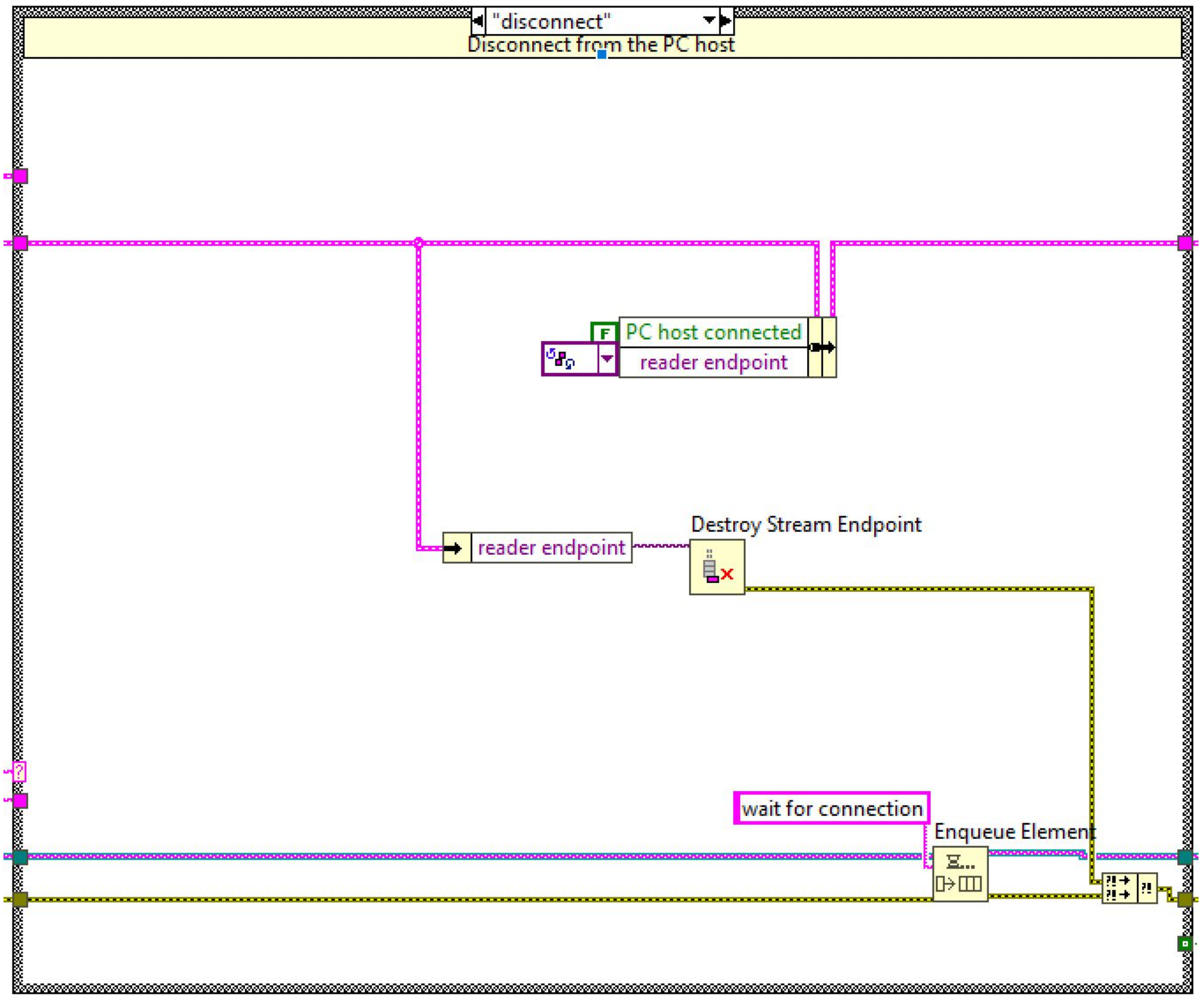


Figure 51: RT 'disconnected' state.

7.2 Queued Message Handler

This LabVIEW program runs on the Queued-Message-Handler (QMH) design pattern. The QMH houses multiple task loops running synchronously in parallel at 34ms/iteration with queue-based intercommunication.

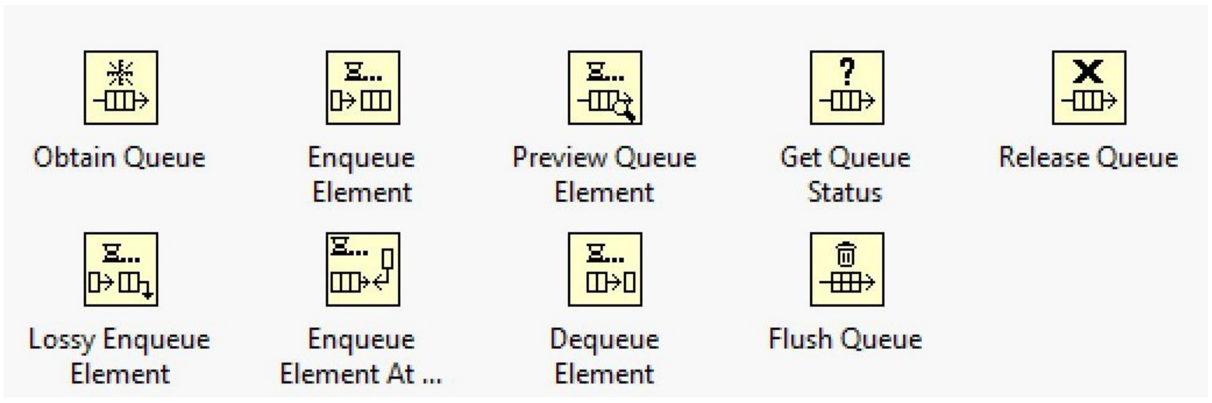
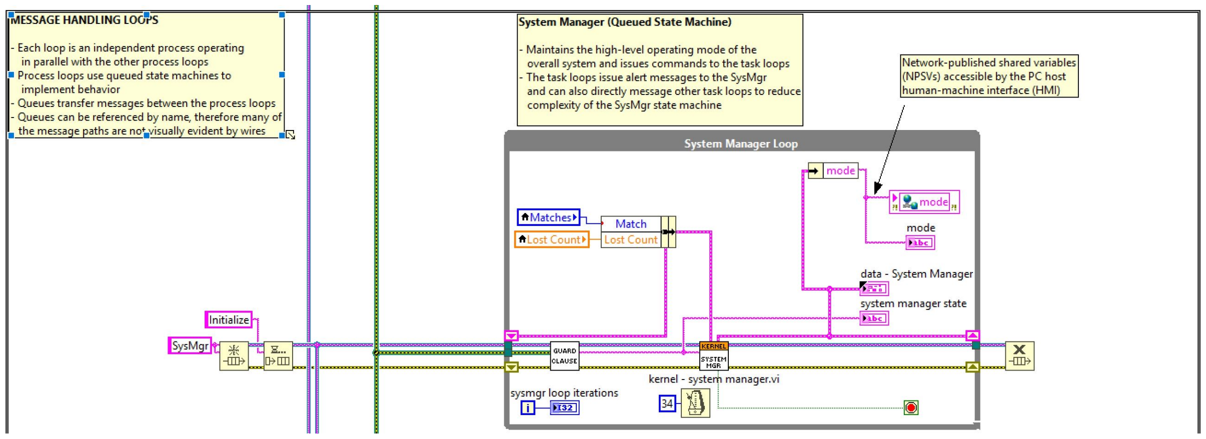


Figure 52: Queue functions.

The System Manager loop, a Queued State Machine (QSM), manages every loop in the QMH except the Command Parser and Error Handler. The 'Obtain Queue' function initializes the queue, and the queue's name 'SysMgr' is fed as the data type. This eliminates most wires from the program. The only by-wire queue references are linked to the Error Handler in section 7.4 [26].



MESSAGE HANDLING LOOPS

- Each loop is an independent process operating in parallel with the other process loops
- Process loops use queued state machines to implement behavior
- Queues transfer messages between the process loops
- Queues can be referenced by name, therefore many of the message paths are not visually evident by wires

System Manager (Queued State Machine)

- Maintains the high-level operating mode of the overall system and issues commands to the task loops
- The task loops issue alert messages to the SysMgr and can also directly message other task loops to reduce complexity of the SysMgr state machine

Network-published shared variables (NPSVs) accessible by the PC host human-machine interface (HMI)

Figure 53: The System Manager loop.

The 'Enqueue Element' function passes 'Initialize' state-message on first-run to the 'kernel' VI inside the loop. First, the message passes through the 'state&args' VI which splits it into two strings using the ':' delimiter. This is useful for passing sub-cases as is done in the 'Send' state in section 7.1.1. This VI is used in every task loop, so it is set to 'pre-allocated clone re-entrant' for each instance to run independently [26].

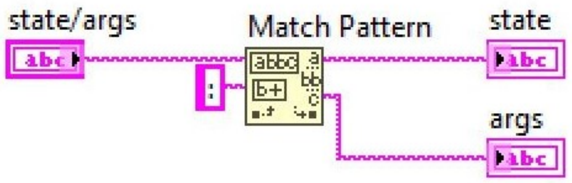


Figure 54: The 'state & args' VI.

The Video and Vision Loops self-enqueue states. This is done using the 'Enqueue Element' function through the internal queue reference wire. The System Manager Loop cannot communicate with other loops using its own queue. To control other loops, it calls the 'Enqueue Single Message' VI which uses the '/' delimeter to separate the queue's name and carried message in the format 'name/message'. Coupled with the 'state & args' VI, the result is 'name/message:argument'. It uses the queue's name to obtain the queue and enqueues its message and argument using 'Enqueue at Opposite End' for priority before destroying the VI's queue reference to clear data. To enqueue messages to multiple loops, the 'Enqueue Multiple Messages' VI passes an array of 'name/message:argument' strings into a for loop which calls the 'Enqueue Single Message' VI for each element. These two VIs are also set to 'pre-allocated clone re-entrant' [26].

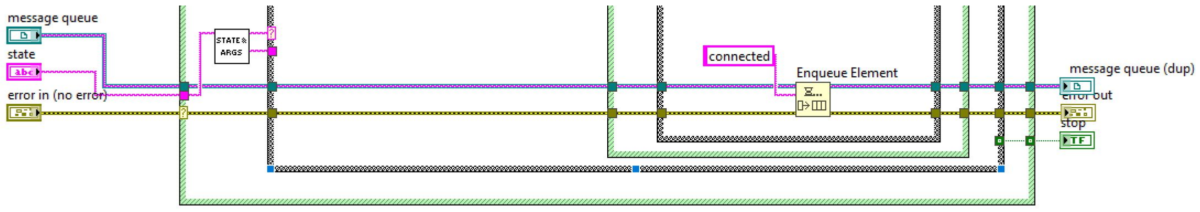


Figure 55: Self-enqueueing states.

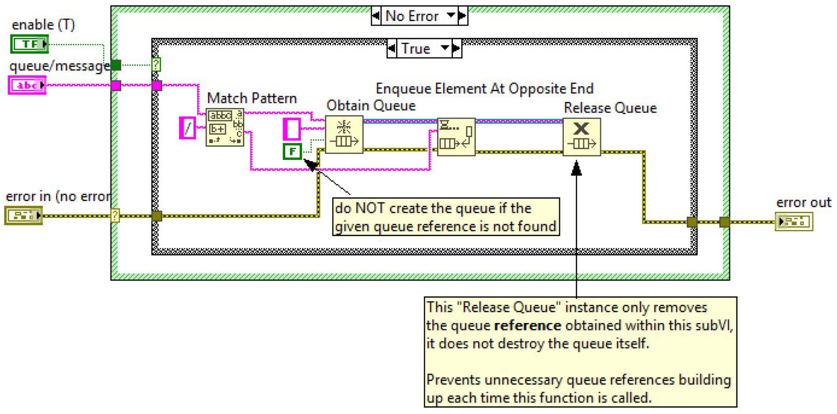

No Error
True
enable (T)
queue/message
Match Pattern
Enqueue Element At Opposite End
Obtain Queue
Release Queue
error in (no error
error out
do NOT create the queue if the given queue reference is not found
This "Release Queue" instance only removes the queue reference obtained within this subVI, it does not destroy the queue itself.
Prevents unnecessary queue references building up each time this function is called.

Figure 56: The 'Enqueue Single Element' VI.

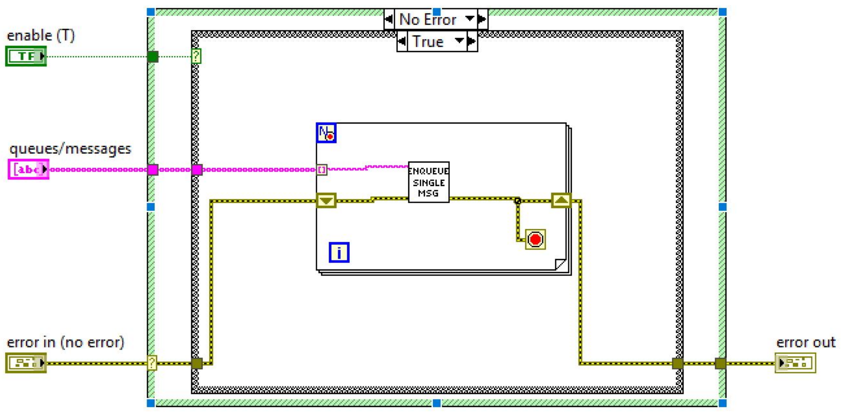


Figure 57: The 'Enqueue Multiple Elements' VI.

Queues are also used to pass time-critical data between loops, such as image frames from the 'Video Loop' to the 'Vision Loop', and pixel coordinates from the 'Vision Loop' to the 'Motion Loop'. These queues are separate from the QMH loop but are implemented in the same way [26].

7.3 Queued State Machines & Kernel VIs

The QSM loops use shift registers to provide persistent data for each iteration. This data is passed via variable clusters bundled into strict type definitions known as the data highway [26].

7.3.1 Video Loop

The video loop self-enqueues the 'run' state post-initialization. The frame capture and storage functions are performed inside the loop but outside the kernel as the time needed to wait for state-message and error identification using the Guard Caluse VI in section 7.4.1 causes lag. The kernel enables the 'Shutdown' state which stops the loop.

This loop initializes the QMH queue, and the image-frame transmission queue labeled 'Video Feed' with the 'Image' data type. On first-run, the 'Open Camera' function with a reference to 'cam0' is called before the 'Configure Grab' function, referencing the frame capture settings from NI MAX. The 'video session' is passed through both functions into the while loop. Finally, the 'ImaqCreate' function is called to create memory for the captured image.

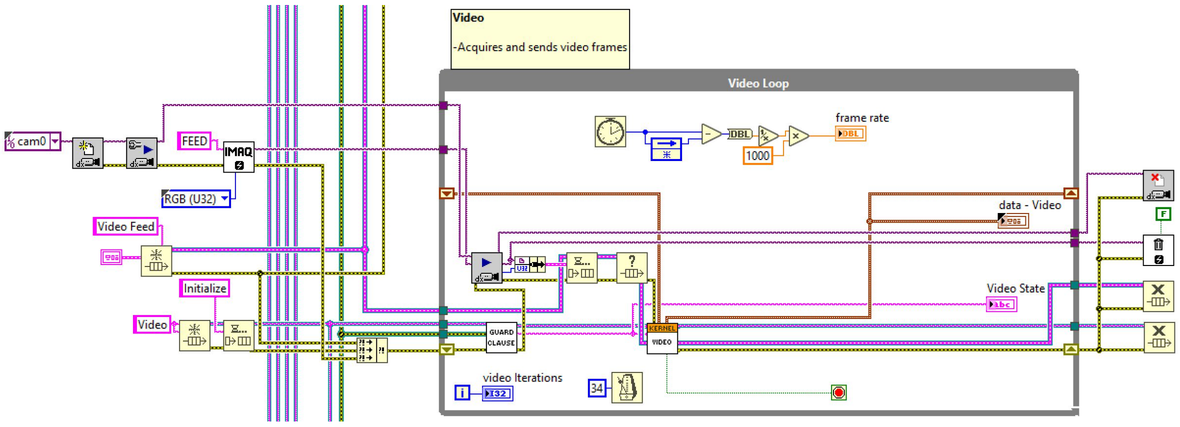


Figure 58: The Video loop.

Inside the loop, 'Grab' sends each captured frame to the 'Video Feed' queue. Simultaneously, the capture rate is measured. A 'Tick Count' measures the milliseconds taken to complete the loop. Using a feedback node, the previous loop time is subtracted from this value. The result is inverted and multiplied by 1000 to output the capture rate in FPS. This is also done within the Vision Loop.

7.3.2 Vision Loop

The Vision Loop calls a set of 'ImaqCreate' functions as explained in section 5.3. The 'Video Feed Queue' is wired into the loop, and an element is dequeued at each iteration into 'Vision', which houses the 'RGB Process' VI. The queue is flushed post-image-processing to ensure that the next enqueued element is the most recent frame; the order of operations is ensured using the error cluster. Finally, the 'matches' output from 'Vision' increments the 'Lost Count' indicator if it is equal to zero, and resets it to zero otherwise.

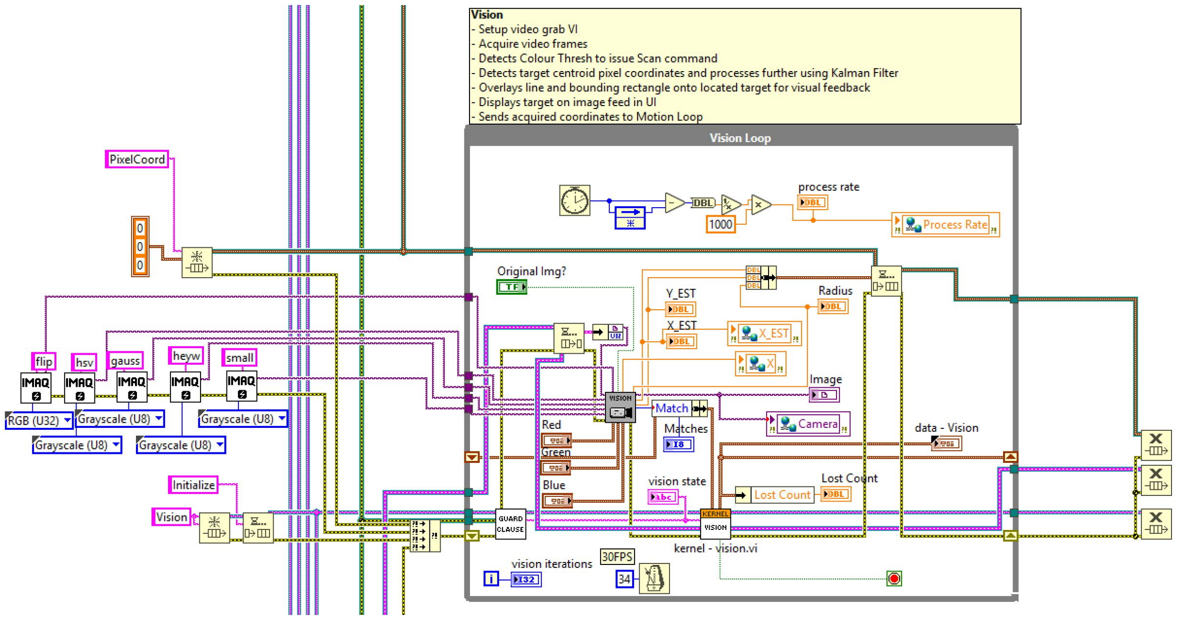
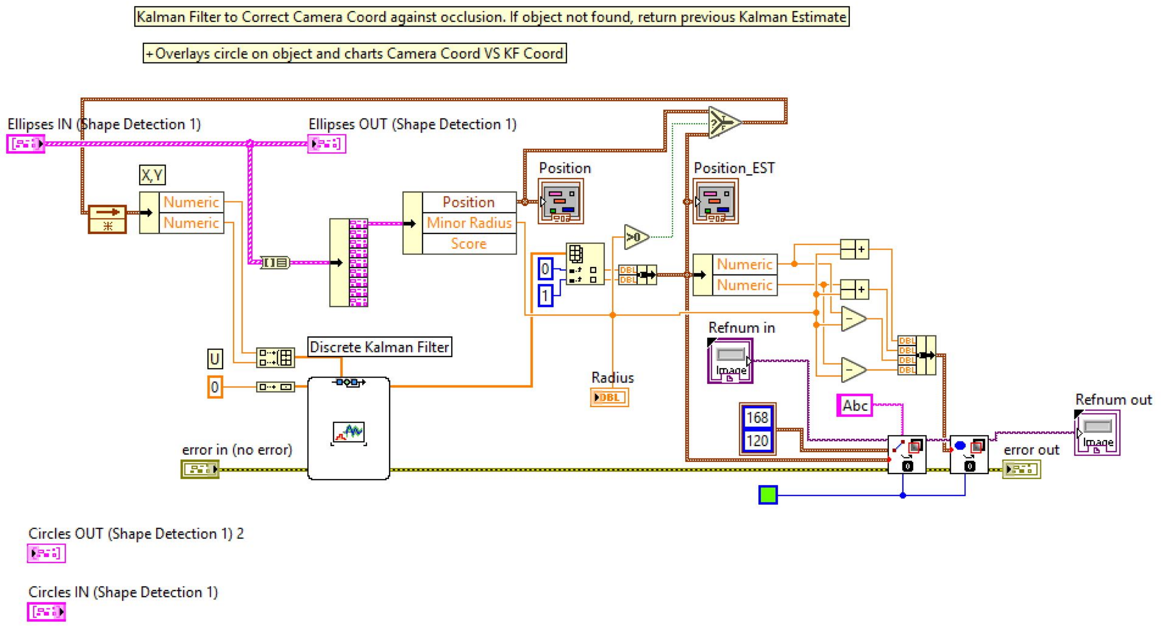

Vision
- Setup video grab VI
- Acquire video frames
- Detects Colour Thresh to issue Scan command
- Detects target centroid pixel coordinates and processes further using Kalman Filter
- Overlays line and bounding rectangle onto located target for visual feedback
- Displays target on image feed in UI
- Sends acquired coordinates to Motion Loop

Figure 59: The Vision loop.

The ’RGB Process’ VI passes the target’s x and y coordinates to the ’Overlay Circle’ VI, which implements the KF from section 5.4.2 and passes the target’s modified pixel coordinates and radius to an ’Overlay Line’ and ’Overlay Oval’ function. The former receives a ’start point’ input of $(168, 120)$, while the ’end point’ input is the KF output: (X_{EST}, Y_{EST}). The latter receives a ’bouding rectangle’ input of $(X_{EST} + Radius)$, $(Y_{EST} + Radius)$, $(X_{EST} - Radius)$, $(Y_{EST} - Radius)$. Together, they draw a line from the centre of the fisheye to the target, along with a circle corresponding to its minor radius like in figure 33. The choice to use the minor radius is explained in section 7.3.4.


Kalman Filter to Correct Camera Coord against occlusion. If object not found, return previous Kalman Estimate

+Overlays circle on object and charts Camera Coord VS KF Coord

Figure 60: The ’Overlay Circle’ VI.

Finally, the KF and raw camera outputs, along with the target's minor radius are passed out of the loop and enqueued into the 'PixelCoord' queue which is wired to the Motion loop.

7.3.3 Motion Loop

On first-run, 'RT Main' calls 'Open RIO AI' for three analogue input channels, two of which are passed to the Motion Loop to record potentiometer voltages from each motor. The 'PixelCoord' queue is wired into the Motion Loop to dequeue the target's pixel coordinates and radius. After each iteration, this queue is flushed to ensure current data.

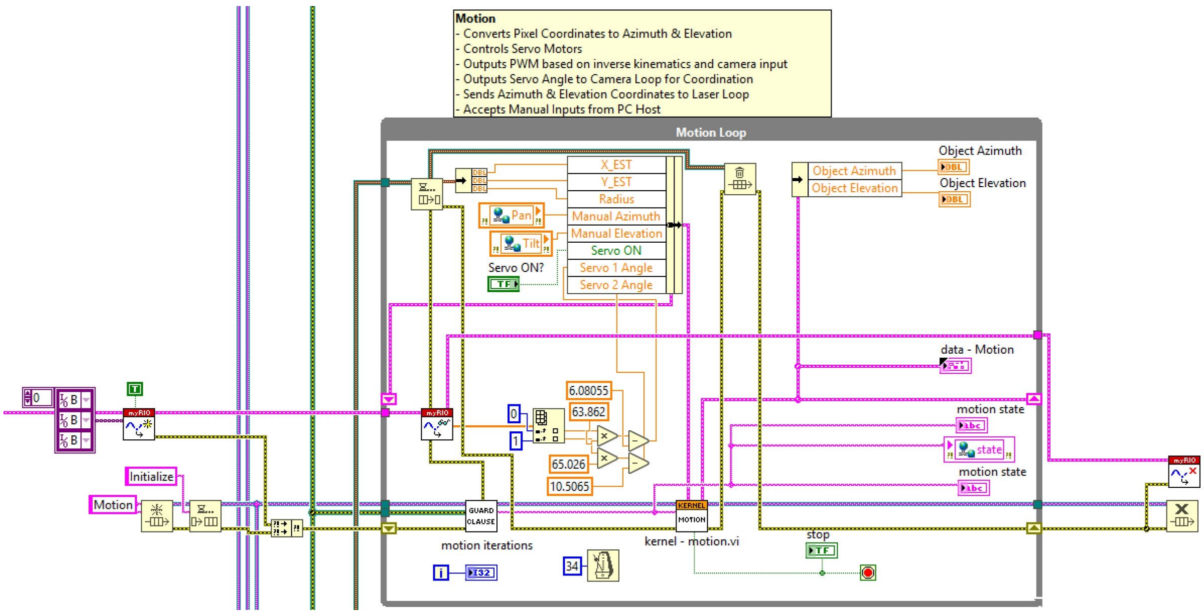
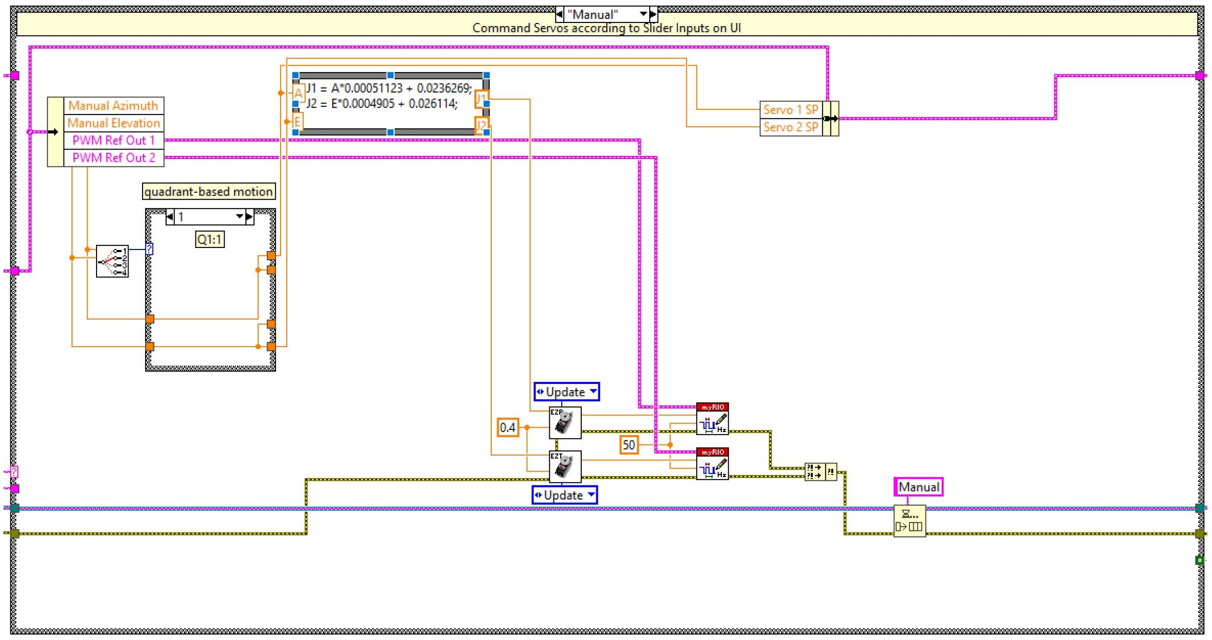

Motion
- Converts Pixel Coordinates to Azimuth & Elevation
- Controls Servo Motors
- Outputs PWM based on inverse kinematics and camera input
- Outputs Servo Angle to Camera Loop for Coordination
- Sends Azimuth & Elevation Coordinates to Laser Loop
- Accepts Manual Inputs from PC Host

Figure 61: The Motion loop.

This loop operates in three states: 'Scanning', 'Tracking' and 'Manual'. Referencing the PC Host GUI, the first two states fall under the 'Automatic' button, while the latter falls under 'Manual'.

The loop's 'Initialize' state establishes PWM terminals on Pins 27 and 29 by calling 'myRIO PWM Open'. An initial PWM value mapped to 0° for each motor is passed at 50Hz using 'Write Duty Cycle and Frequency' through the 'Smooth Pan' and 'Smooth Tilt' VIs in their 'Initialize' to set up the easing algorithm from section 6.2. Additionally, the data highway variables are set to zero or false.

In the 'Manual' state, azimuth and elevation setpoints provided by the PC Host's mouse coordinates as described in Appendix 9 are passed to a case selector which determines oint angles based on section 6.1. These are converted into duty cycles using the calibration map from Appendix 7. Finally, each motor is actuated trhough its easing algorithm VI. The 'Manual' state self-enqueues until a priority command is received from the System Manager Loop.

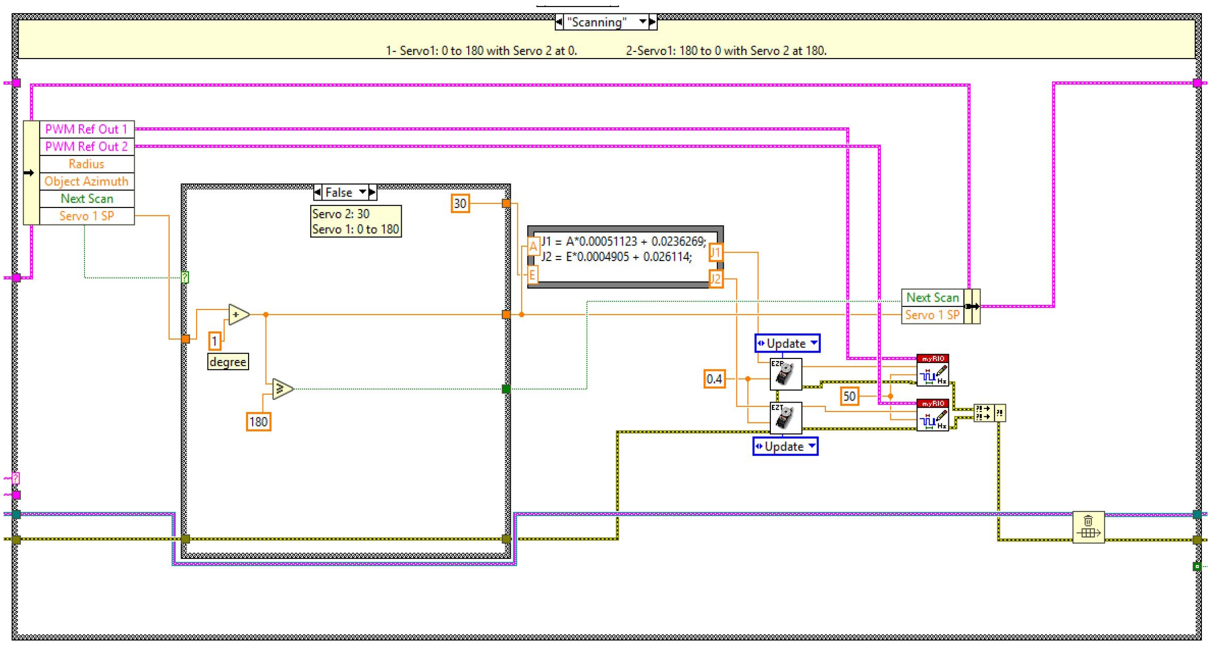


```
J1 = A*0.00051123 + 0.0236269;
J2 = E*0.0004905 + 0.026114;
```

Figure 62: The 'Manual' state.

The 'Scanning' state sets the joint angles as follows:

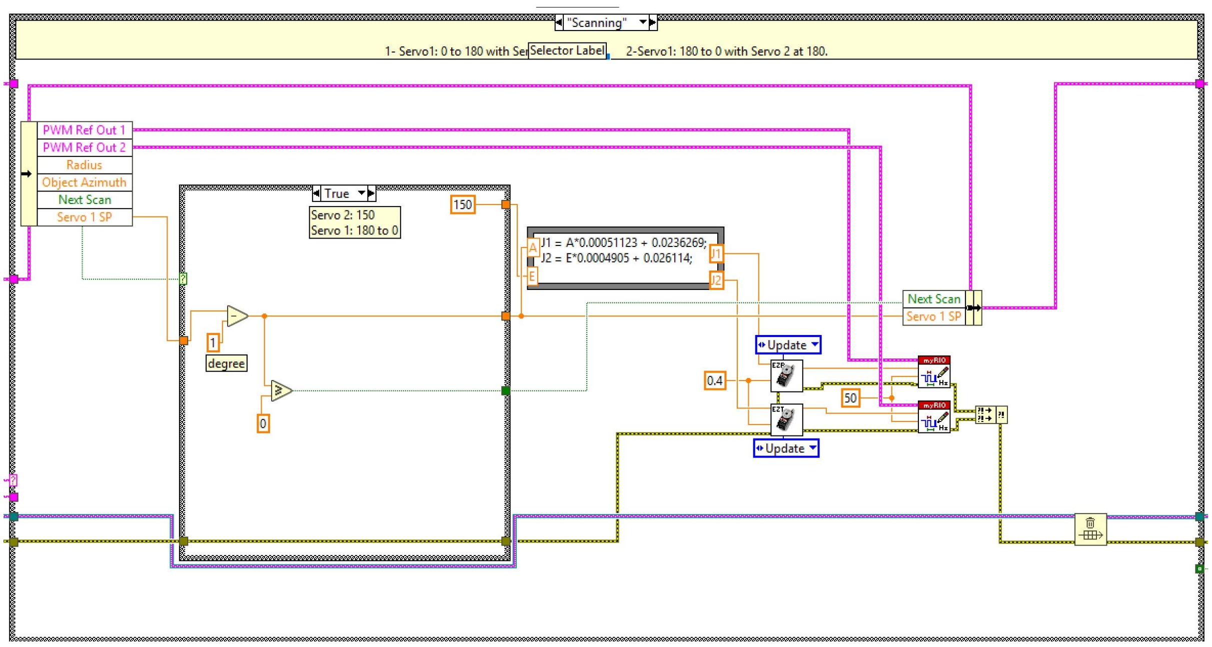
1. Scanning:1
   On first-run, the false case is engaged, setting the elevation motor to 30° and incrementing the azimuth motor by 1° each iteration until the next exceeds 180°, changing the 'Next Scan' boolean to true. This works by sending the azimuth command to 'Servo SP1' through the loop's shift register and reading it in the next iteration.



```
J1 = A*0.00051123 + 0.0236269;
J2 = E*0.0004905 + 0.026114;
```

Figure 63: The False case.

2. Scanning:2 The true case is engaged, setting the elevation motor to 150° and decrementing the azimuth motor by 1° each iteration until the next moves below 0°, changing the 'Next Scan' boolean to false.



```
J1 = A*0.00051123 + 0.0236269;
J2 = E*0.0004905 + 0.026114;
```

Figure 64: The True case.

This state is enqueued by the 'automatic' state within the System Manager Loop. the automatic state sends the the 'run' state to the Laser and Data Loops, and the 'Scanning' state to the Motion Loop while simultaneously enqueueing itself into the 'scan' state. Here, the 'Matches' variable is monitored. If it is equal to zero, the above-mentioned states are re-enqueued, otherwise, 'Tracking' is enqueued to the Motion Loop, and 'track' is self-enqueued to the System Manager Loop.

Within the System Manager Loop's 'track' state, the 'Lost Count' variable is monitored. If it exceeds 100, meaning that the target has been lost for 3.4 seconds, the Scanning operation states are enqueued. Otherwise, the Tracking operation states are re-enqueued.

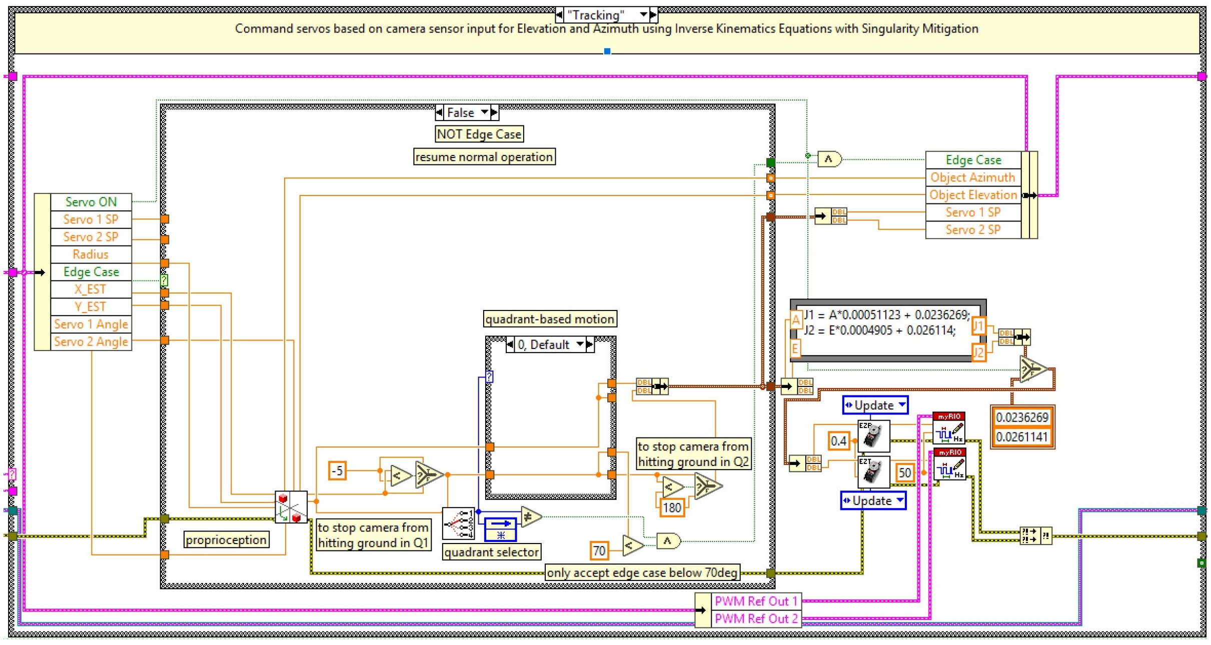
In the Motion 'Tracking' state, the camera-referenced azimuth and elevation values from the 'Pixel to Spherical Coordinates' VI are passed to the Laser Loop as 'Object Azimuth' and 'Object Elevation' via Local Variables before summing the motor positions to them and completing the system's proprioception. Elevation is limited between -5° and 180° to avoid ground collisions as shown in figure 65.

An issue was identified at edge cases where the PATBLC must flip the EE to access the half-sphere's second quadrant. During this swing, if the target's elevation is below 70°, the bullseye is erroneously picked up in the subsequent frame, resulting in a motor update that locks the system into back-and forth motion. The following solution was implemented:

1. Edge Case False:
   The operating quadrant is identified as described in 6.1, and a feedback node is used to determine if the quadrant has changed. If this is true, elevation is below 70°, and the

'Servo ON' boolean is set to true from the GUI, then the 'Edge Case' boolean is set to true for the next iteration. Otherwise, it is set to false and the same operation occurs in the next iteration.

Adjacently, the motors are commanded as described in the 'Manual' state. This only happens if the 'Servo ON' boolean is set to true, otherwise, the (0°,0°) position is set. The motor commands are also passed to the shift register through the 'Servo SP1' and 'Servo SP2' indicators.



```
J1 = A*0.00051123 + 0.0236269;
J2 = E*0.0004905 + 0.026114;
```

Figure 65: 'Edge Case' is False.

2. Edge Case True:
   Here, 'Servo SP1' and 'Servo SP2' from the flase scenario are passed as motor commands and compared to current motor positions. If these are within $\pm3°$ of each other, the 'Edge Case' boolean is set to false and normal operation ensues. Otherwise, it is set to true again, and the last commanded position is passed at every iteration until the proximity condition is met.

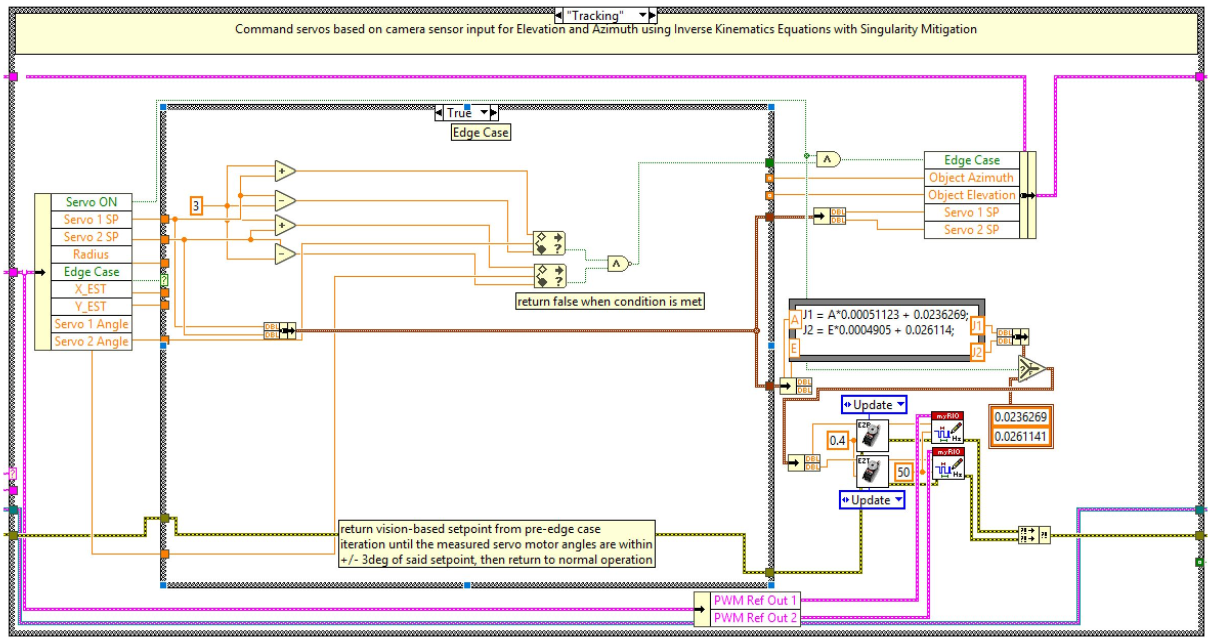

"Tracking"
Command servos based on camera sensor input for Elevation and Azimuth using Inverse Kinematics Equations with Singularity Mitigation
True
Edge Case
Servo ON
Servo 1 SP
Servo 2 SP
Radius
Edge Case
X_EST
Y_EST
Servo 1 Angle
Servo 2 Angle
3
return false when condition is met
return vision-based setpoint from pre-edge case iteration until the measured servo motor angles are within +/- 3deg of said setpoint, then return to normal operation
Edge Case
Object Azimuth
Object Elevation
Servo 1 SP
Servo 2 SP
J1 = A*0.00051123 + 0.0236269;
J2 = E*0.0004905 + 0.026114;
0.0236269
0.0261141
Update
Update
0.4
50
PWM Ref Out 1
PWM Ref Out 2

Figure 66: 'Edge Case' is True.

During the Motion 'Shutdown' state, the command to return to the (0°,0°) position is given before closing the PWM reference using 'PWM Close'.

7.3.4 Laser Loop

The Laser Loop calculates whether the EE is pointing within the bullseye through the 'Proximity' VI and actuates the laser using digital output pin 11 if the 'Laser ON?' control is true.

The 'Proximity' VI receives 'Object Azimuth', 'Object Elevation', and 'Target Radius'. First, the radius is converted into azimuth and elevation similarly to the 'Pixel to Spherical Coordinates' VI method. Then, it is fed as the limits for two 'In Range and Coerce' functions with 'Object Azimuth' and 'Object Elevation' inputs.

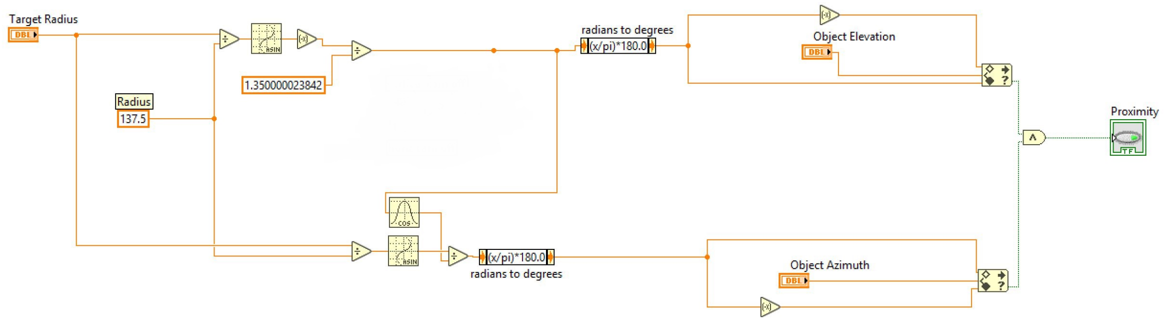


Figure 67: The 'Proximity' VI.

The 'Centered' VI determines whether the EE is centered on the bullseye as above, except the 'In Range and Coerce' limits are 1 and -1. The 'Pixel to Spherical Coordinates' VI outputs 'Object Elevation' and 'Object Azimuth' values of 0° if the EE and bullseye are centered.

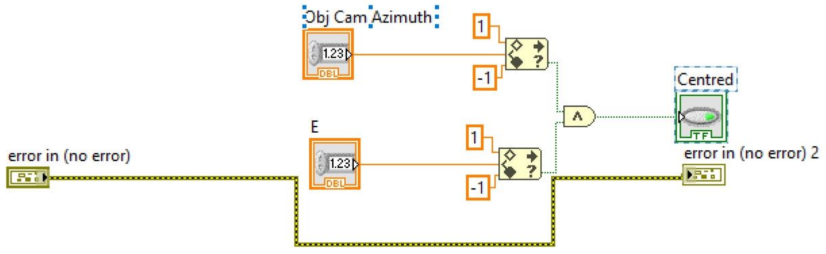


Figure 68: The 'Centred' VI.

7.3.5 Data Loop

The data loop reads the receiver using analogue input pin 9 and sends it to the PC Host alongside the 'Centered' boolean. The target's pixel radius is converted into a mm distance using the map in Appendix 7. A 'Centered + Rx' boolean is true when both the 'Centered' boolean is true and the receive signal is greater than 2V.

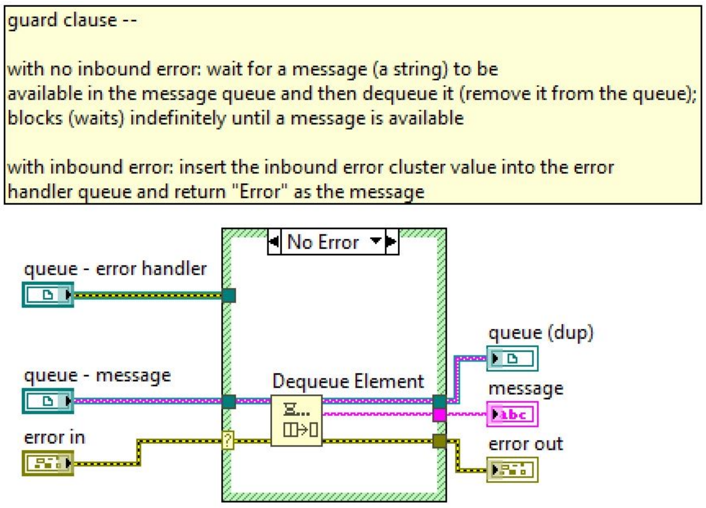
All data transmission between the RT Target and PC Host is done using Network Publishes Shared Variables (NPSV). These run on the Shared Variable Engine and are sparsely used due to computational cost.

The NPSVs from the RT to the PC Host are charted using the fourth loop in the 'PC Main' VI. It logs the data in the GUI's charts and uses a local variable of the 'ESC' stop button to end the loop. On stop, the data in each chart is exported to Excel using invoke nodes. On first-run, the charts are cleared by injecting a blank into their 'History' property node.

7.4 Error Handling

7.4.1 Guard Clause VI

The Guard clause VI is the same as in section 7.2.



guard clause --

with no inbound error: wait for a message (a string) to be available in the message queue and then dequeue it (remove it from the queue); blocks (waits) indefinitely until a message is available

with inbound error: insert the inbound error cluster value into the error handler queue and return "Error" as the message

Figure 69: Error-less 'Guard Clause' VI state.

Messages are dequeued by the VI and returned as strings. If an error is passed, it is injected into the error cluster, and the message queue is ignored [26].

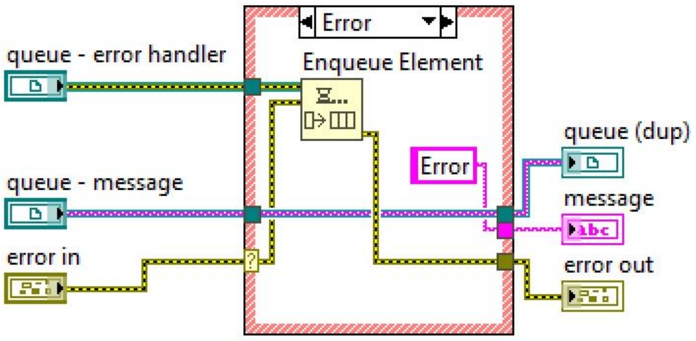
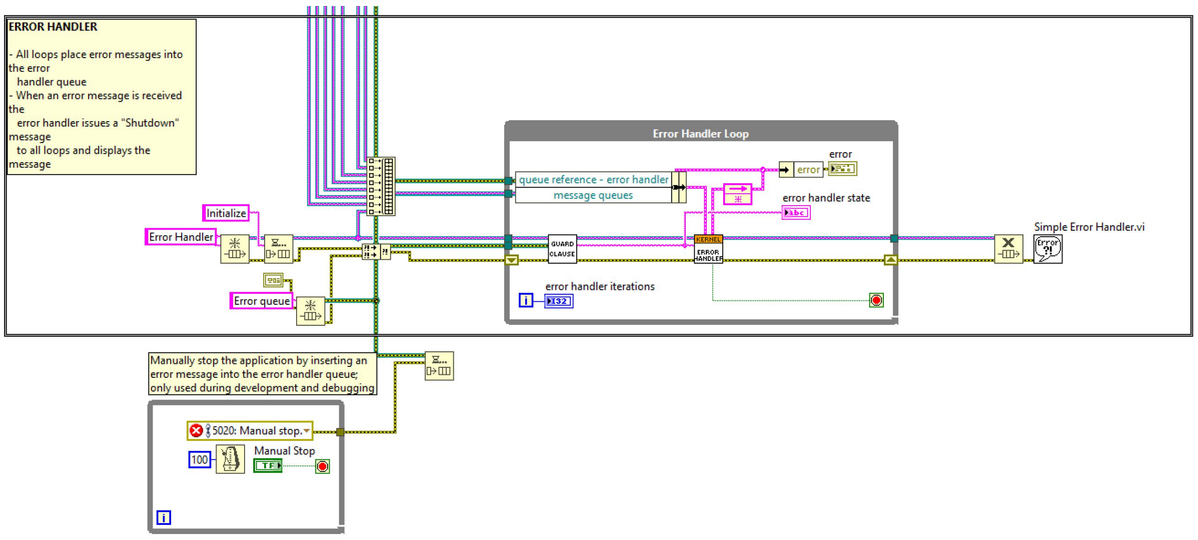


Figure 70: 'Guard Clause' VI in error state.

7.4.2 Shutdown Sequence

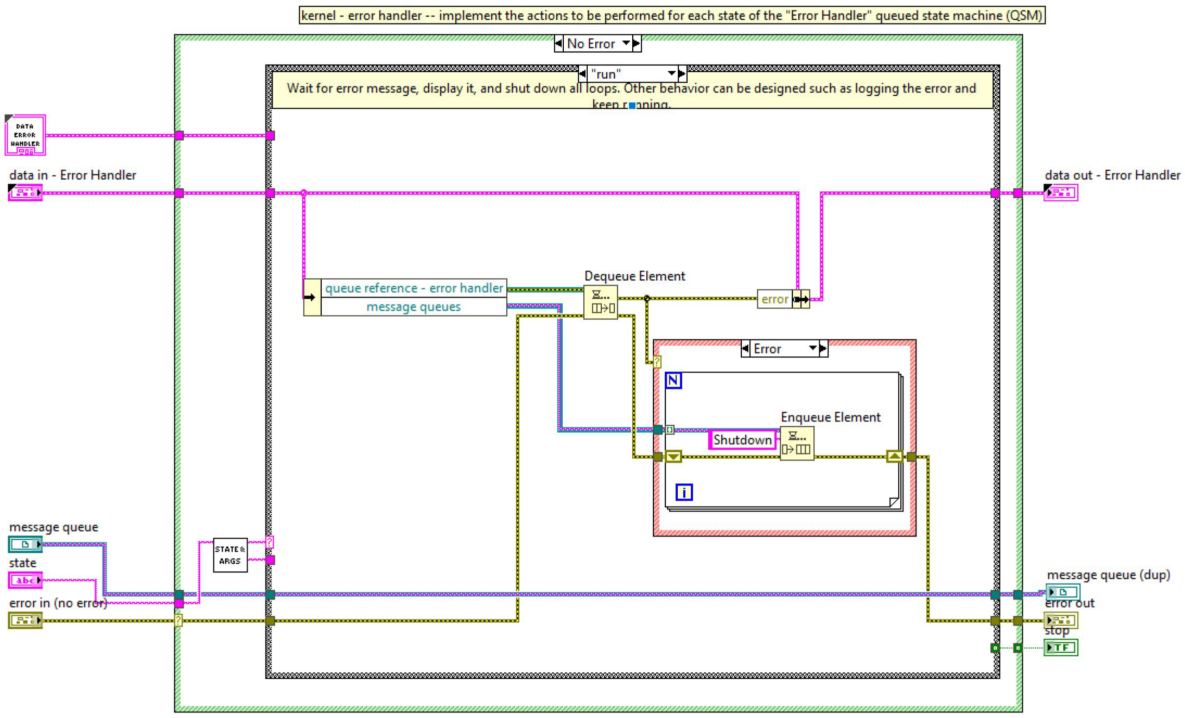
The Error Handler Loop initiates the shutdown sequence upon receipt of an error. It extracts all queue names through the merged queue reference input and priority-enqueues the 'Shutdown' message to each. It passes the error to a 'Simple Error Handler' before stopping the loop [26].


ERROR HANDLER

- All loops place error messages into the error handler queue
- When an error message is received the error handler issues a "Shutdown" message to all loops and displays the message

Manually stop the application by inserting an error message into the error handler queue; only used during development and debugging

(a) Error Handler.


kernel - error handler -- implement the actions to be performed for each state of the "Error Handler" queued state machine (QSM)

Wait for error message, display it, and shut down all loops. Other behavior can be designed such as logging the error and keep running.

(b) Error Handler 'run' state.

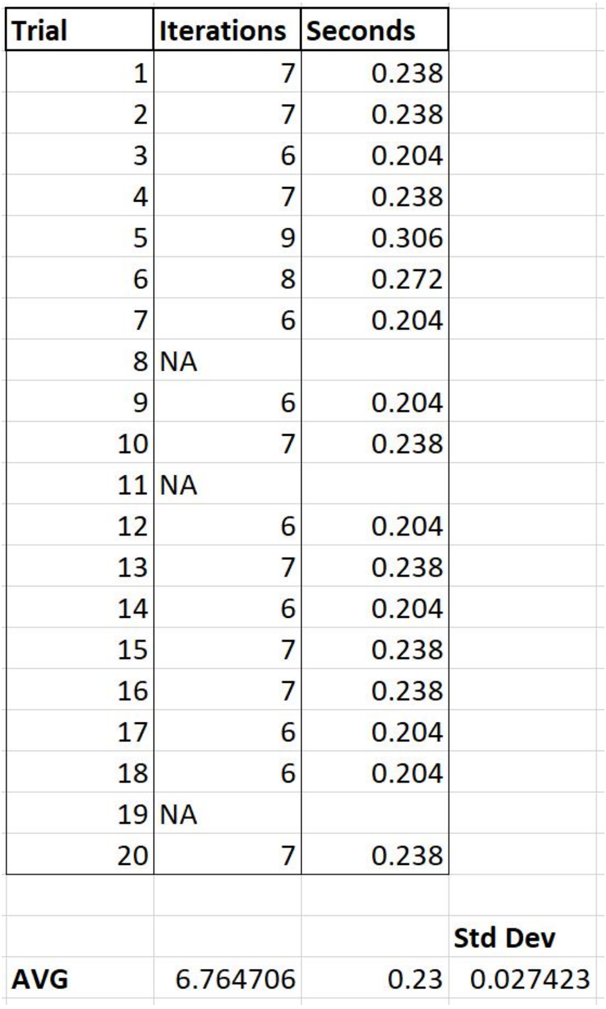
Figure 71: The Error Handler loop

The system can be shut down remotely using the PC Host 'Shutdown' command which injects an error through the System Manager Loop. There, each kernel VI stops its loop. Then, queues are destroyed, myRIO I/O are closed, and image memory is cleared.

8 Performance

8.1 Quantitative Analysis

The first test places the target 60° in azimuth, 500mm away from the EE to evaluate the time to close the beam-link. This was repeated 20 times by placing the target and verifying azimuth on the front panel before engaging the 'Servo ON' boolean. A summary of the data provided via USB ('Time.xlsx') is given below, where 'NA' indicates a no-link-close condition:



Trial	Iterations	Seconds	
1	7	0.238	
2	7	0.238	
3	6	0.204	
4	7	0.238	
5	9	0.306	
6	8	0.272	
7	6	0.204	
8	NA		
9	6	0.204	
10	7	0.238	
11	NA		
12	6	0.204	
13	7	0.238	
14	6	0.204	
15	7	0.238	
16	7	0.238	
17	6	0.204	
18	6	0.204	
19	NA		
20	7	0.238	
			Std Dev
AVG	6.764706	0.23	0.027423

Figure 72: Test 1 summmary.

The second test evaluates system's accuracy at various distances by achieving a close-range uplink and moving the target away. Unless the second system is built, it isn't possible to conduct a representative test as increasing the receiver wire length increases circuit noise and decreases pointing accuracy. Furthermore, since the distance map from section 5.2.5 was found to be insufficient, distance measurements could not be trusted. The raw data is provided via USB ('Distance.xlsx') but is not discussed for the reliability reasons above.

8.2 Qualitative Analysis & Discussion

1. The motor position measurements error of $\pm 2^\circ$ caused unnecessary re-centring.
2. The maximum target identification distance was 870mm, limited by pixel resolution.
3. The spherical lens is non-linear, so a linear map of radius to distance was insufficient.
4. The 3D printed optical mounts are expected to generate pointing error.
5. The system's wobble was dampened using the easing algorithm to avoid degrading system performance. However, a mechanical solution should be used to reduce easing for quicker motion.

On average, the system took 0.23s to centre on a target 60° away, just 21.0526% below the motor's speed (0.19s/60° at 4.8V) because of motor easing, the EE weight and process lag. In three trials, the system failed to close the link but still pointed the laser at the bullseye, indicating 85% accuracy due to the linear lens map and optical mount tolerances. The population's standard deviation is 27.423ms, which is smaller than the 34ms loop time. This data is incomplete as the second (unfinished) PATBLC must be implemented to evaluate the whole system.

9 Debriefing

9.1 Future Work

Recommendations for future work assume the PATBLC should be improved before adapting it to a submerged environment:

1. Correct the EE wobble by replace the horn attachments with a planetary gear system from section 4.1.
2. Minimise motor measurement error using a cleaner power supply and an encoder. Alternatively, an instrumentation amplifier is a low-cost alternative.
3. Fit the focus lenses and the second camera to complete the system.
4. Map the spherical lens using a 4^{th} order polynomial for target-size versus distance [20].
5. Use a cRIO or RoboRIO to exploit the camera's full 1920x1080 resolution, theoretically increasing target identification distance by a factor of 27.
6. Modify the KF state-space model to include the SquidROV's and Hub's EOM.
7. Design a waterproof motor case since the feedback modification ruins pre-existing water-tightness.

9.2 Final Thoughts

Compared to the galvanometer suggested by the LL in section 3, the PATBLC behaves poorly with a motor sensor accuracy of $\pm 2°$, and a maximum pointing distance of 870mm due to a 320x240 resolution and inadequate spherical lens mapping. However, considering that each system costs £75 and was developed in 12 weeks, these figures are more respectable. Notably, the vision system behaved consistently in all lighting conditions at 30FPS.

The KF successfully offset the camera's measurement noise while providing occlusion resistance, and the IK was implemented reliably at low computational cost, resulting in robust behaviour at edge cases. The easing algorithm prevented the base from toppling, and minimised the EE's wobble to close beam-links quickly, at 0.23s/60° on average, just 21.0526% below rated motor speed. In a 20-trial sample, the PATBLC closed links at 85% accuracy, and seldom broke them unless a noise-spike caused erroneous re-centring.

The second system could not be built without the second camera. The focus lenses were also not received, and are expected to improve accuracy.

The modular LabVIEW code written for this project makes it possible to subsume the SquidROV code into one project. The recommended future work in section 9.1 is based on current system observations. However, underwater vision will likely be the biggest submersion challenge.

References

[1] Hamilton, Scott A., et al. Undersea Narrow-Beam Optical Communications Field Demonstration. Ocean Sensing and Monitoring IX, 2017, doi:10.1117/12.2264733.

[2] Bahr, Alexander. Cooperative Localization for Autonomous Underwater Vehicles. Intl.Journal of Robotics Research, 2009, pp. 714728., doi:10.1575/1912/2852.

[3]Abrahamson, Matthew J., et al. Achieving Operational Two-Way Laser Acquisition for OPALS Payload on the International Space Station. Free-Space Laser Communication and Atmospheric Propagation XXVII, 2015, doi:10.1117/12.2182473.

[4] Petzold, Theodore J. Volume Scattering Functions for Selected Ocean Waters. 1972, doi:10.21236/ad0753474.

[5] Kalman, R. E. A New Approach to Linear Filtering and Prediction Problems. Journal of Basic Engineering, vol. 82, no. 1, 1960, p. 35., doi:10.1115/1.3662552.

[6] Thorlabs - HEP3965 Broadband IR Tungsten Bulb, 1900 K, TO-8 Can, www.thorlabs.com/.cfm?partnumber

[7] Servo Shop, www.servoshop.co.uk/index.php?pid=PAAS700area=Servo.

[8] Modern Robotics: Mechanics, Planning, and Control. Modern Robotics: Mechanics, Planning, and Control, by Kevin M. Lynch and Frank C. Park, University Press, 2017, pp. 77237.

[9] Spong, Mark W. Robot Modeling and Control. John Wiley Sons Inc, 2012, pp. 41 - 125

[10] Understanding Optical Specifications — Edmund Optics. Edmund Optics Worldwide, www.edmundoptics.com/resources/application-notes/optics/understanding-optical-specifications/

[11] Wave Tank Moving Forward. The University of Edinburgh, 29 Mar. 2016, www.ed.ac.uk/unpublished/news/2012/flowave-281112.

[12] Geometrical Optics. Optics. Hecht, by Eugene Hecht, Addison-Wesley, 1998, pp. 160256.

[13] Wei, Ching-Chuan, et al. Design of a Solar Tracking System Using the Brightest Region in the Sky Image Sensor. Sensors, vol. 16, no. 12, 2016, p. 1995., doi:10.3390/s16121995.

[14] N. Team, NI myRIO, Natl. Instruments, 2013.

[15] "Servo Motor Glossary of Terms". Oriental Motor U.S.A. Corp, 2019. https://www.orientalmotor.com/servo-motors/technology/servo-motor-glossary.html.

[16] "1A06AA Datasheet". Sparkfun, January 2004. https://www.sparkfun.com/datasheets/Components/General/RFP30N06LE.pdf.

[17] "Tower Pro MG996R Servo Specifications". Servo Database, 2019.

https://servodatabase.com/servo/towerpro/mg996r

[18] ”Industrial Automation Resources: Steppers vs Servos”. Advanced Micro Controls Inc, 2015.
https://www.amci.com/industrial-automation-resources/plc-automation-tutorials/stepper-vs-servo/

[19] N. Team, NI Vision Concepts Manual, Natl. Instruments, November 2005.

[20] ”Fisheye mathematics”. Dr. Paul Bourke, 2019.
http://paulbourke.net/dome/fisheyecorrect/fisheyerectify.pdf

[21] ”Fisheye, Real World to Screen Coordinates”. Dr. Paul Bourke, 2019.
http://paulbourke.net/dome/fisheye.pdf

[22] Westenberger, Antje, et al. Time-to-Collision Estimation in Automotive Multi-Sensor Fusion with Delayed Measurements. Advanced Microsystems for Automotive Applications 2013 Lecture Notes in Mobility, 2013, pp. 1320., doi:10.1007/978-3-319-00476-1_2.

[23] Introduction: State-Space Methods for Controller Design. Control Tutorials for MATLAB and Simulink.
http://ctms.engin.umich.edu/CTMS/index.php?example=Introductionsection=ControlStateSpace

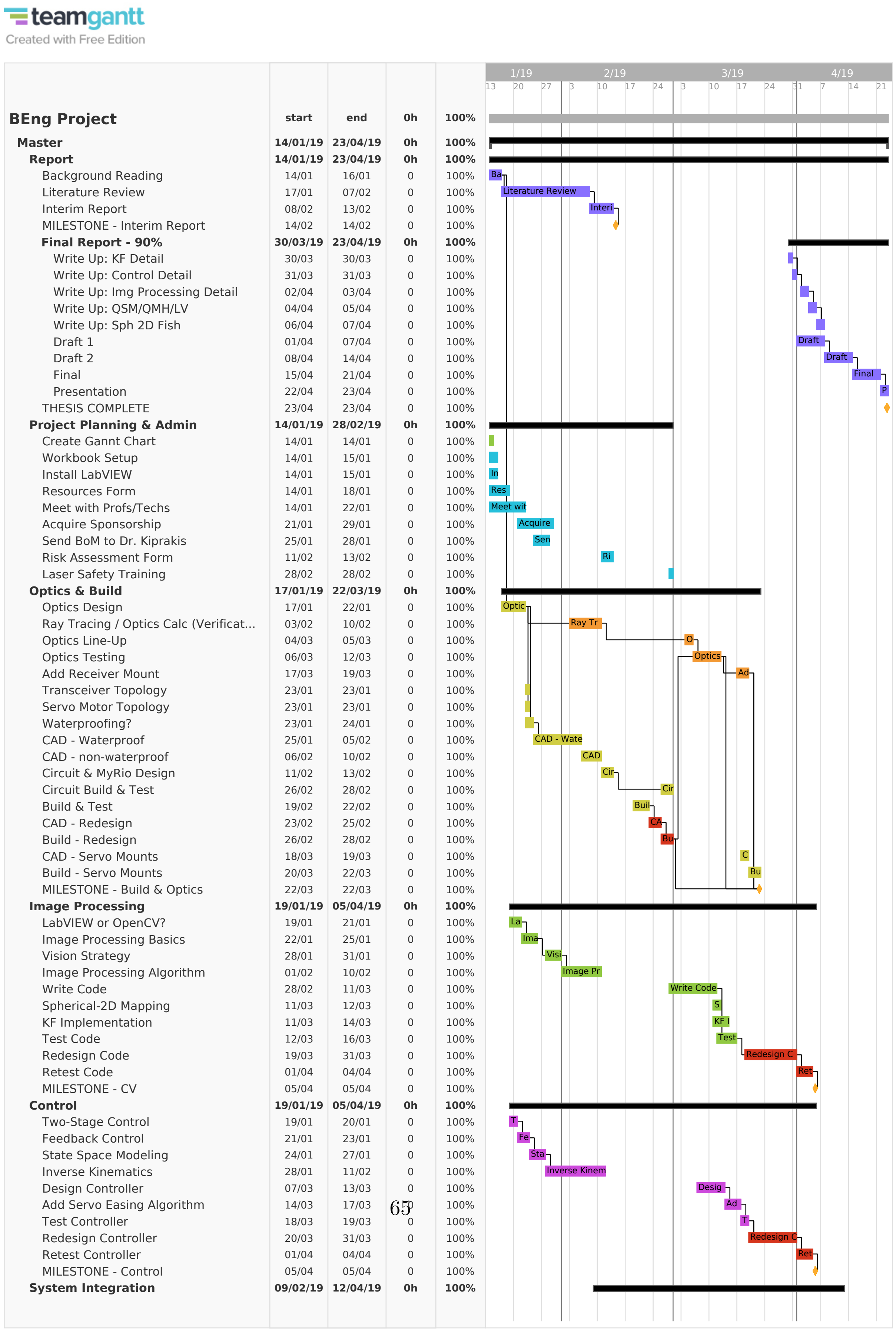
[24] Prenner, Robert. Motion, Tweening, and Easing. Dynamic Visuals, pp. 193237.

[25] ”Filtering Servo Movements”. Guilherme Martins, August 2009.
http://lab.guilhermemartins.net/2009/08/21/filtering-servo-movements/

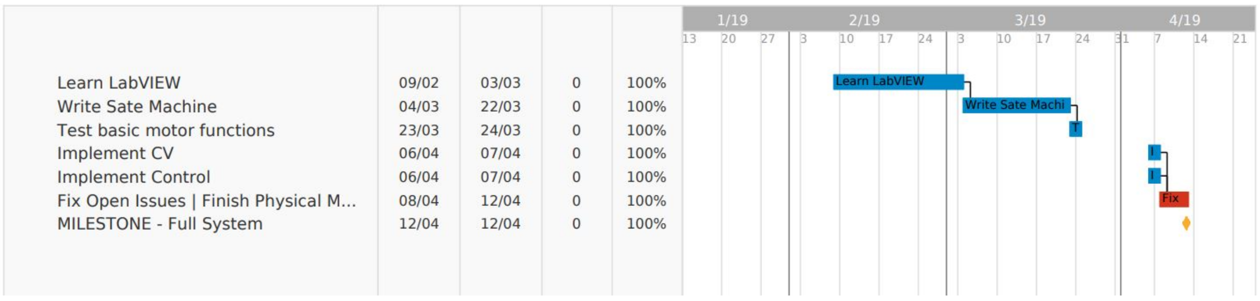
[26] ”System controller example: Home Security System”. Ed Doering, 2018.
https://learn-cf.ni.com/teach/riodevguide/code/rt-pc_home-security-system.html

Appendix

1 - Gantt Chart



BEng Project	start	end	0h	100%
Master	**14/01/19**	**23/04/19**	**0h**	**100%**
Report	**14/01/19**	**23/04/19**	**0h**	**100%**
Background Reading	14/01	16/01	0	100%
Literature Review	17/01	07/02	0	100%
Interim Report	08/02	13/02	0	100%
MILESTONE - Interim Report	14/02	14/02	0	100%
Final Report - 90%	**30/03/19**	**23/04/19**	**0h**	**100%**
Write Up: KF Detail	30/03	30/03	0	100%
Write Up: Control Detail	31/03	31/03	0	100%
Write Up: Img Processing Detail	02/04	03/04	0	100%
Write Up: QSM/QMH/LV	04/04	05/04	0	100%
Write Up: Sph 2D Fish	06/04	07/04	0	100%
Draft 1	01/04	07/04	0	100%
Draft 2	08/04	14/04	0	100%
Final	15/04	21/04	0	100%
Presentation	22/04	23/04	0	100%
THESIS COMPLETE	23/04	23/04	0	100%
Project Planning & Admin	**14/01/19**	**28/02/19**	**0h**	**100%**
Create Gannt Chart	14/01	14/01	0	100%
Workbook Setup	14/01	15/01	0	100%
Install LabVIEW	14/01	15/01	0	100%
Resources Form	14/01	18/01	0	100%
Meet with Profs/Techs	14/01	22/01	0	100%
Acquire Sponsorship	21/01	29/01	0	100%
Send BoM to Dr. Kiprakis	25/01	28/01	0	100%
Risk Assessment Form	11/02	13/02	0	100%
Laser Safety Training	28/02	28/02	0	100%
Optics & Build	**17/01/19**	**22/03/19**	**0h**	**100%**
Optics Design	17/01	22/01	0	100%
Ray Tracing / Optics Calc (Verificat...	03/02	10/02	0	100%
Optics Line-Up	04/03	05/03	0	100%
Optics Testing	06/03	12/03	0	100%
Add Receiver Mount	17/03	19/03	0	100%
Transceiver Topology	23/01	23/01	0	100%
Servo Motor Topology	23/01	23/01	0	100%
Waterproofing?	23/01	24/01	0	100%
CAD - Waterproof	25/01	05/02	0	100%
CAD - non-waterproof	06/02	10/02	0	100%
Circuit & MyRio Design	11/02	13/02	0	100%
Circuit Build & Test	26/02	28/02	0	100%
Build & Test	19/02	22/02	0	100%
CAD - Redesign	23/02	25/02	0	100%
Build - Redesign	26/02	28/02	0	100%
CAD - Servo Mounts	18/03	19/03	0	100%
Build - Servo Mounts	20/03	22/03	0	100%
MILESTONE - Build & Optics	22/03	22/03	0	100%
Image Processing	**19/01/19**	**05/04/19**	**0h**	**100%**
LabVIEW or OpenCV?	19/01	21/01	0	100%
Image Processing Basics	22/01	25/01	0	100%
Vision Strategy	28/01	31/01	0	100%
Image Processing Algorithm	01/02	10/02	0	100%
Write Code	28/02	11/03	0	100%
Spherical-2D Mapping	11/03	12/03	0	100%
KF Implementation	11/03	14/03	0	100%
Test Code	12/03	16/03	0	100%
Redesign Code	19/03	31/03	0	100%
Retest Code	01/04	04/04	0	100%
MILESTONE - CV	05/04	05/04	0	100%
Control	**19/01/19**	**05/04/19**	**0h**	**100%**
Two-Stage Control	19/01	20/01	0	100%
Feedback Control	21/01	23/01	0	100%
State Space Modeling	24/01	27/01	0	100%
Inverse Kinematics	28/01	11/02	0	100%
Design Controller	07/03	13/03	0	100%
Add Servo Easing Algorithm	14/03	17/03	0	100%
Test Controller	18/03	19/03	0	100%
Redesign Controller	20/03	31/03	0	100%
Retest Controller	01/04	04/04	0	100%
MILESTONE - Control	05/04	05/04	0	100%
System Integration	**09/02/19**	**12/04/19**	**0h**	**100%**



Learn LabVIEW	09/02	03/03	0	100%
Write Sate Machine	04/03	22/03	0	100%
Test basic motor functions	23/03	24/03	0	100%
Implement CV	06/04	07/04	0	100%
Implement Control	06/04	07/04	0	100%
Fix Open Issues | Finish Physical M...	08/04	12/04	0	100%
MILESTONE - Full System	12/04	12/04	0	100%

2 - Homogeneous Transformation Matrix

The Inverse Kinematics (IK) of a system is a relationship between joint angles and the End-Effector (EE) (the part of the manipulator that interacts with the workspace) position in space [8,9].

A good way of finding this system's IK is to treat it as a 3DOF spherical wrist with three revolute joints if the laser transceiver is treated as the third DOF. This conserves the systems integrity as the incoming and outgoing beams travel the same line of action, and a rotation of the EE has no consequence on the systems pointing angle.

The first step in finding the IK is to identify the Homogeneous Transformation Matrix (HTM), a 4x4 matrix defining rotation and translation between subsequent frames in a Kinematic Diagram (KD), where the rotation matrix is described by the first 3x3 elements of the HTM, and the displacement vector by the first three row elements of the fourth column. These are called the rotation and displacement parts of the HTM. Rotation matrices are commutative, but displacement vectors are not. The last row of the HTM is $\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}$ to allow commutative matrix multiplication between frames by turning the HTM into a square matrix and circumventing the displacement vector's non-commutative property to reach a full description of the EE frame relative to the base frame in this manner:

$$H_3^0 = H_1^0 H_2^1 H_3^2 \tag{9.1}$$

The HTM describes the subscript 3 (EE) frame relative to the superscript 0 (base) frame [8,9].

There are two methods for finding the HTM: the Denavit-Hartenberg Method (DHM), and the Product of Exponentials (PoE) method. Dr. Kevin Lynch argues that while the DHM constitutes fewer parameters, its computational benefit is offset by lower EE pointing accuracy in high-DOF open-chain manipulators due to its vulnerability to alignment errors. Given the low-DOFs of the PATBLC and the high computational power required due to object tracking through computer vision, the DHM was chosen for the interim report. The PoE method may be implemented in a later trial if accuracy issues present themselves while using the DHM [8].

The DHM method is as follows. The first step is to draw the Kinematic Diagram (KD) and to assign frames according to the following DHM rules [8,9]:

1. The z-axis is the axis of rotation of a revolute joint, or the direction of motion of a prismatic joint. The end-effector frame follows the last joint frame.

2. The x-axis must be perpendicular to the z-axis of its own frame and to the z-axis of the frame before it.

3. All frames must follow the right-hand-rule, whereby the thumb points in the z-direction, the palm points in the y-direction, and the fingers point in the x-direction. That is to say, the axis represented diagonally on the page must always be drawn from left to right with a positive slope.

4. Each x-axis must intersect the z-axis of the previous frame in three-dimensional space.

True to these rules, here is the PATBLC's KD:

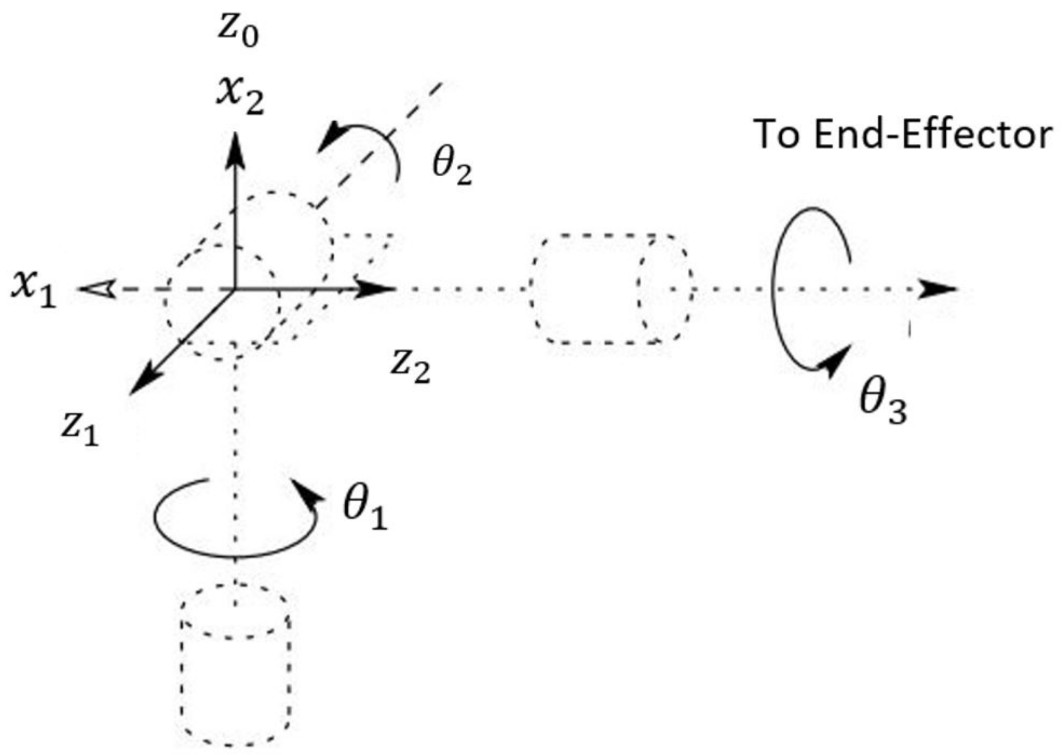


Figure 73: Kinematic Diagram for the PATBLC Spherical Wrist interpretation [9].

The second step is to fill the DHM parameter table. Below is the PATBLC's populated table:

$Link_n$	θ_n	α_n	r_n	d_n
1	θ_1	$-90°$	0	0
2	θ_2	$90°$	0	0
3	θ_3	$0°$	0	d_3

Table 2: The first column, θ_n, indicates the rotation of the n^{th} frame about the z_{n-1} axis so that the x_n and x_{n-1} axes match. The second column, α_n, indicates the rotation of the $n-1^{th}$ frame about the x_n axis so that z_{n-1} and z_n match. r_n indicates the distance between the centres of frames n and $n-1$ in the x_n direction. Finally, d_n indicates the distance between the centres of frames n and $n-1$ in the z_{n-1} direction [9].

The third step is to insert the table's values into equation 9.2 relating the n^{th} frame to the $n-1^{th}$ frame for each row of the DH parameter table, with cos and sin replaced by c and s respectively for clarity:

$$H^{n-1}_{\ \ n} = \begin{bmatrix} c_{\theta n} & -s_{\theta n}c_{\alpha n} & s_{\theta n}s_{\alpha n} & r_n c_{\theta n} \\ s_{\theta n} & c_{\theta n}c_{\alpha n} & -c_{\theta n}s_{\alpha n} & r_n s_{\theta n} \\ 0 & s_{\alpha n} & c_{\alpha n} & d_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.2}$$

This gives

$$H_1^0 = \begin{bmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.3}$$

$$H_2^1 = \begin{bmatrix} c_2 & 0 & s_2 & 0 \\ s_2 & 0 & -c_2 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.4}$$

$$H_3^2 = \begin{bmatrix} c_3 & -s_3 & 0 & 0 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.5}$$

Note that $\cos_{\theta\, n}$ and $\sin_{\theta\, n}$ are abbreviated as c_n and s_n.

Finally, the HTM can be computed by plugging equations 9.3, 9.4, and 9.5 into 9.1 to give:

$$H_3^0 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1c_2s_3 - s_1c_3 & c_1s_2 & c_1s_2d_3 \\ s_1c_2c_3 + c_1s_3 & -s_1c_2s_3 + c_1c_3 & s_1s_2 & s_1s_2d_3 \\ -s_2c_3 & s_2s_3 & c_2 & c_2d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.6}$$

3 - Euler Angle Transformation

Euler Angle Transformations depend on the chosen order of operations. For this system, the standard order of roll, then pitch, then yaw was chosen to describe rotational motion. These are right-multiplied as follows:

$$R_{XYZ} = R_z(z)R_y(y)R_x(x) \tag{9.7}$$

where

$$R_z(z) = \begin{bmatrix} c_z & -s_z & 0 \\ s_z & c_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9.8}$$

$$R_y(y) = \begin{bmatrix} c_y & 0 & s_y \\ 0 & 1 & 0 \\ -s_y & 0 & c_y \end{bmatrix} \tag{9.9}$$

$$R_x(x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & -s_x \\ 0 & s_x & c_x \end{bmatrix} \tag{9.10}$$

to yield

$$R_{XYZ} = \begin{bmatrix} c_z c_y & -s_z c_x + c_z s_y s_x & s_z s_x + c_z s_x + c_z s_y c_x \\ s_z c_y & c_z c_x + s_z s_y s_x & -c_z s_x + s_z s_y c_x \\ -s_y & c_y s_x & c_y c_x \end{bmatrix} \tag{9.11}$$

Note that $\cos_{\theta\,n}$ and $\sin_{\theta\,n}$ are abbreviated as c_n and s_n.

4 - MATLAB Code

```
%Optics.m

loop = 100000000;

t2 = 0.00125;
d1=0;

for repeat = 1:loop
    t1 = t2 + d1;
    d1 = t1-asin((1/1.333)*sin(t1));
end
```

5 - Servo Motor Calibration

1. Follow the assembly instruction for each revolute joint as shown in Appendix 8.
2. Press-fit the provided splined motor horn onto the output shaft, then run the VI and control the 'Duty Cycle' input at 50Hz until the shaft is at its minimum position. Then, remove it and re-fit it just behind the 0° line. This will be accounted for later, and is done because the output shaft's splines do not usually align parallel to the side of the motor.
3. Using small increments, increase the value of 'Duty Cycle' input until the servo horn is parallel to the sides of the motor. The Duty Cycle value for this position should be recorded.
4. By changing the VI control again, rotate the servo horn to roughly 180° from the previous point, then, using small increments, ensure that the horn is again parallel to the sides of the motor and record the value.
5. Finally, using these values, a linear relationship between Duty Cycle and shaft position can be found using the following:

$$y = mx + b \tag{9.12}$$

Where y = Duty Cycle and x = Desired Position°.

$m = \frac{0.115648-0.0236265}{180-0}$ and $b = 0.0236265$ for the azimuth motor. The remaining maps are given in Appendix 7.

6 - Servo Motor Position Feedback Modification

Below are the steps for the position feedback modification done to the servo motors. Note that this will likely void the warranty.

1. Remove the screws from the bottom of the servo casing. Be sure to hold the top of the casing securely once this is done to prevent the gear assembly from falling out.

2. Carefully extract the exposed integrated circuit (IC) from the casing. In this model, the white wire on the underside of the IC corresponds to the potentiometer's position feedback node, which returns a voltage corresponding to the shaft's position. Note that every motor's IC is different, and the node for this model was found experimentally.

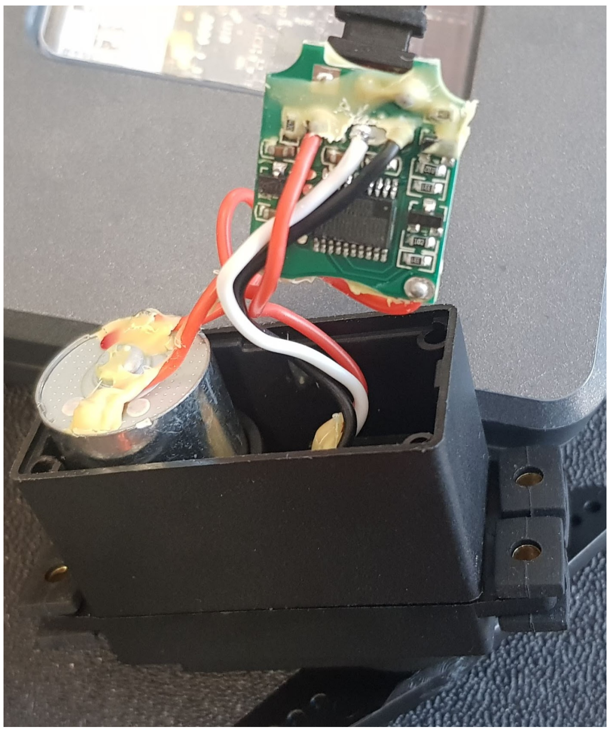


Figure 74: The motor IC with its white potentiometer terminal.

3. Secure the motor casing and IC using some blue putty and carefully remove the yellow shielding paste around the position feedback note and solder an additional white wire onto the white position feedback node. Be sure not to remove too much of the yellow paste as it protects the IC from shorts and also gives it mechanical strength.

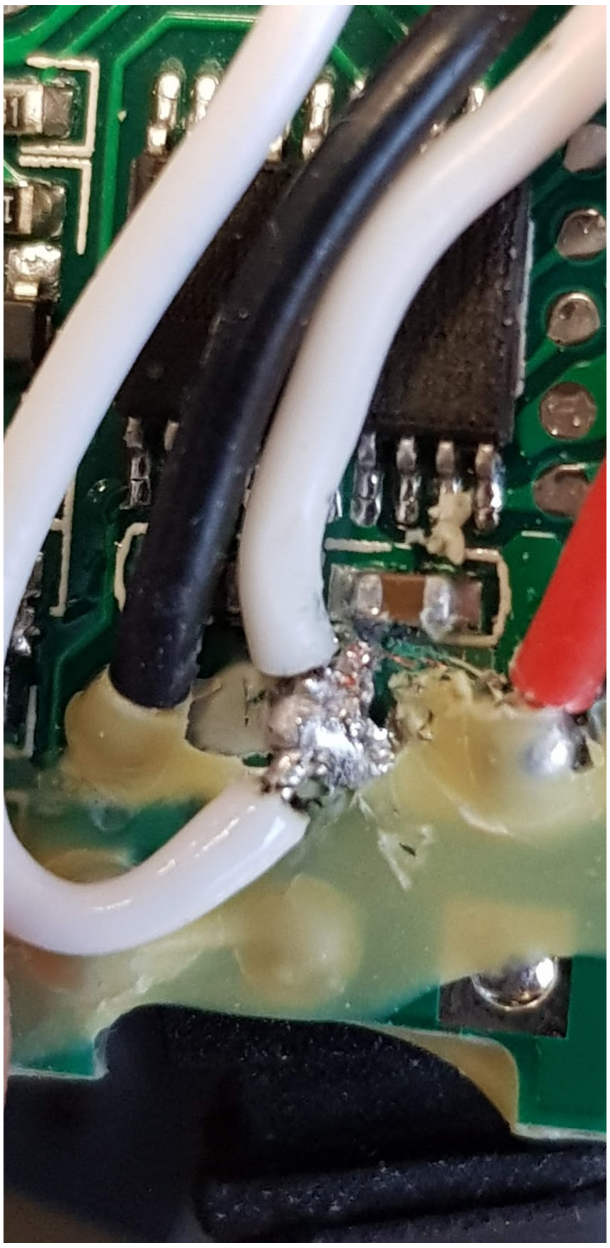


Figure 75: The additional soldered feedback terminal.

4. Using a carving knife or small box-cutter, cut an additional cavity into the motor's cover to allow the newly soldered wire to exit the motor casing.

5. Finally, carefully re-insert the IC into its original position and pass the newly soldered wire out of the new cavity, and place the original terminal grommet adjacent to it before re-fitting the motor's cover and re-inserting and tightening its screws. Then, wrap some heat shrink around all the terminals and secure them together using a heat gun on the lowest setting.

7 - Maps

$MapName$	Map
$Servo1_{PWM}$	$PWM = (5.1123E-4)x^\circ + 0.0236265$
$Servo2_{PWM}$	$PWM = (5.1123E-4)x^\circ + 0.0236265$
$Servo1_{VOLTAGE}$	$^\circ = 63.862V - 6.08055$
$Servo2_{VOLTAGE}$	$^\circ = 65.028V - 10.5065$
$PixelRadiustoDistance$	$^\circ = -0.47Radius(px) + 11.88$

Table 3: Maps

8 - Assembly Instructions

The assembly instructions can be broken up into the following steps for each systems:

1. Revolute One - Base (1x), Motor (1x), Motor Wood Screws (4x).
   (a) Fit the rubber and brass inserts onto the servo motor.
   (b) Fit the servo motor onto the base by first pushing the outgoing terminals into the base's vertical slot at an angle, then pushing the motor down after straightening it.
   (c) Fit each of the four wood screws onto the motor and drive them into the base using a phillips-head screwdriver.
2. Revolute Two - Motor Mount (1x), Motor Horn (2x), Motor (1x), Motor Arm (1x), M2 bolts (6x), M2 Washers (6x), M2 Nuts (6x), Motor Wood Screws (4x), Motor Spline Bolt (2x).
   (a) Secure the motor horn onto the motor mount using three of the M2 bolts, washers and nuts.
   (b) Place the above assembly onto the first motor (assembled in part 1) and follow the calibration instructions in section 4.4. Then, secure it using one of the motor spline bolts.
   (c) Secure the second servo motor to the motor mount using four wood screws as in part 1.
   (d) Cut the second motor horn in the middle of its its second hole on one side, then secure it to the motor arm using the remaining three M2 bolts, washers and nuts. If any part of the horn still protrudes from the arm, trim it.
   (e) Secure the motor arm assembly onto the second motor, following the calibration instructions in section 4.4 and subsequently securing it using the last motor spline bolt.
3. End-Effector - End Effector Cylinder (1x), Camera Mount (1x) Optical Breadboard (1x), Post Base (3x), Focus Lens Mount (2x), Focus Lens Mount Inner Ring (1x), Mirror Mount (1x), Receiver Mount (1x), Epoxy Resin (1 Tube), Cotton Gloves, Bullseye (1x), Dichroic Mirror (1x), Laser Diode (1x), Focus Lens (1x), Receiver (1x), Disposable Mixing Utensil (1x) 6mm M3 Bolts (16x), 10mm M3 Bolts (5x), M3 Nuts (13x), 5mmx5mm Knurled Inserts (10x), Camera Mount M1 Screws (2x), M5 Bolt (1x), M5 Nut(1x).
   (a) Using a pair of snips, hold one of the knurled inserts onto one of the optical breadboard's holes (there are six on the flat surface, and two on the curved surface) and heat it using a solding iron. Once the insert is hot, it will begin to melt into the hole, and the snips will prevent it from going any further. Release the snips and gently push the insert into the hole using the soldering iron, ensuring that it is flush with the breadboard's surface. Repeat this seven more times until all the holes are filled. Additionally, repeat this step twice at the front of the EE cylinder. **Be sure to wear safety goggles during this step!**

(b) Glue the receiver onto the receiver mount from the back, ensuring that the photodiode is centered. To do this, mix some epoxy using a disposable mixing utensil (scrap plastic/metal/wood) and apply it without making direct contact with the substance. **Be sure to wear safety goggles during this step!**

(c) Grab one of the focus lens mounts and press-fit the laser diode into it. If the laser is small, there should be a small ring that can be fitted first to accomodate its size. If either the ring or laser is loose, fit three of the M3 nuts into the mount, and use three of the 6mm M3 bolts to fasten the assembly evenly. Repeat this for the focus lens, ensuring that the inner ring is fastened first, and that the receiver mount is attached to the lens mount's slots before any bolts are secured if necessary. The nuts are not needed here since the friction from the tightly-toleranced receiver mount's holes will grant the bolts a gripping surface. In fact, it may be necessary to open the hole slightly using a carving knife.

(d) Using a pair of cotton gloves, set aside of the dichroic mirrors. If the red or green laser was fastened to the laser mount, use the high-pass mirror, otherwise, if the blue laser was fitted, use the low-pass mirror. Then, fit four of the M3 nuts into the bottom and top portions of the mirror mount. Then, fit two of the 10mm M3 bolts onto the top of the mount, driving one all the way done, and the other halfway as pictured below.

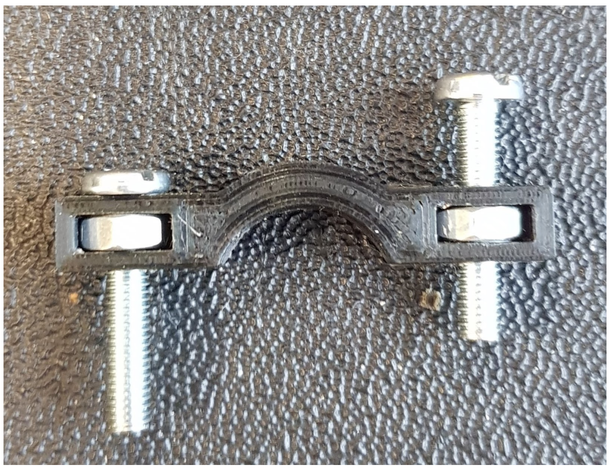


Figure 76: Fitting the top of the mirror mount.

Then, through the fully-driven bolt, drive the top mount into the bottom mount, leaving just enough space to squeeze the mirror in.

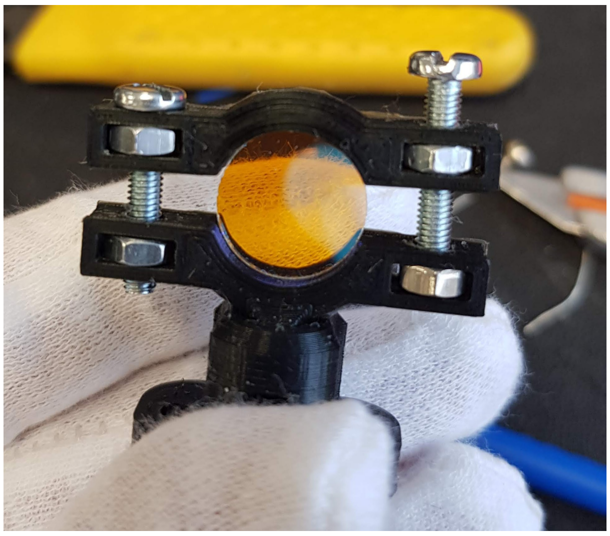


Figure 77: Slotting the mirror into the assembly.

Then, fit the mirror until a 'click' sound is heard, and drive the other bolt all the way down until both bolts are securely fastened.

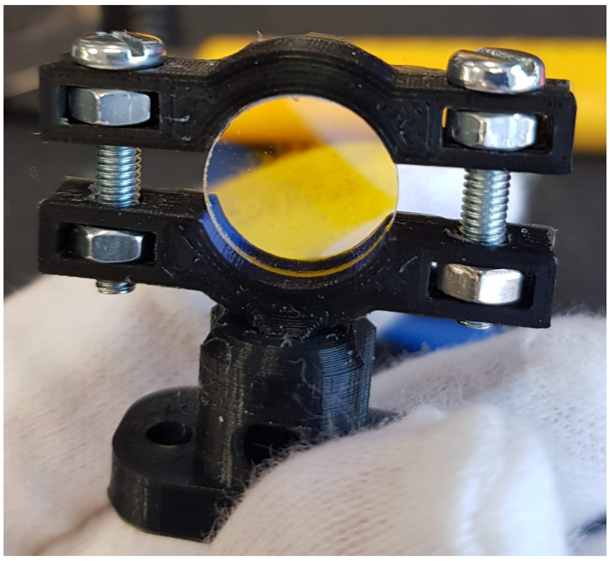


Figure 78: The final dichroic mirror assembly.

(e) Secure each of the post bases to the breadboard using six of the 6mm M3 bolts through the knurled inserts. Then, fit each of the mounts into a post base. If the fit is not tight, push an M3 nut into the post base and drive one of the 10mm M3 bolts through to ensure a secure fit between the mounts and the post base.

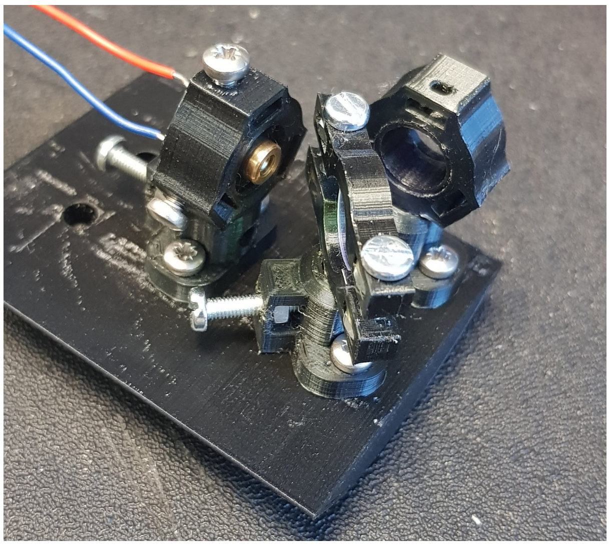


Figure 79: Isometric view of the optical breadboard.

(f) Slide the finished breadboard into the EE cylinder while pulling the diode and receiver's terminals out of the large hole in the back, and secure it to the bottom of the cylinder using two of the 6mm M3 bolts. Secure the rearmost bolt first, making sure to push the breadboard into the cylinder using a finger while doing so.

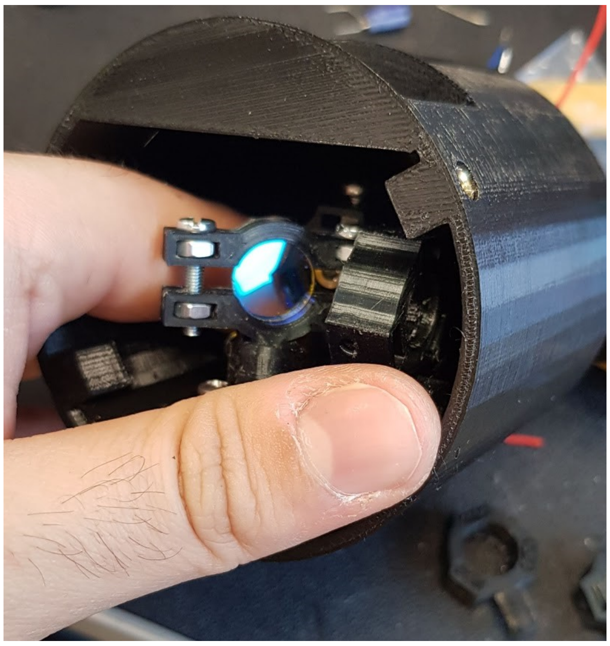


Figure 80: Fitting the breadboard to the EE.

(g) Fasten the camera mount to its metal case using the two provided camera screws. Then, push the mount's octagonal extrusion into the hole in the EE cylinder until a 'click' is heard. To unfasten it, the extrusion can be pinched to release the camera and push it back out.

4. Finally, secure the EE to the motor arm using the M5 nut and bolt, and secure the bullseye to the EE using the two remaining 6mm M3 bolts.

   The final result should look like figure 23.

9 - Manual Mouse Control

This VI is described in the appendix because it is non-essential to the project. It was developed as a means of verifying the IK implementation described in section 6.1. It was left in the final project as it provides additional functionality to the system.

The 'Manual' tab loop reads the mouse coordinates when the mouse moves on the front panel. If these coordinates fall within the indicator's bounds, they are translated into azimuth and elevation commands and sent to the Real-Time (RT) Target using NPSVs.

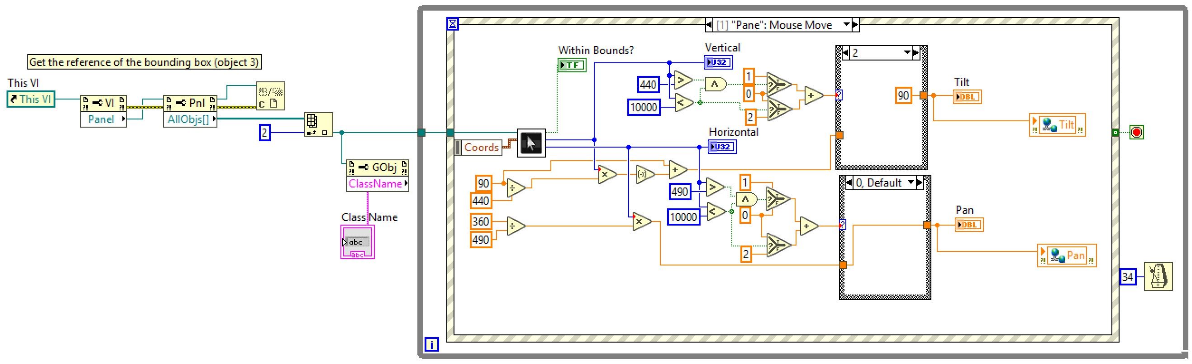


Figure 81: The 'Manual' tab loop.

On first-run, a property node references 'This VI' to pick out the Tab indicator from the front panel elements. It is indexed from the reference array and passed into the 'Bounding Box' VI through the process loop along with the mouse coordinates from the event structure.

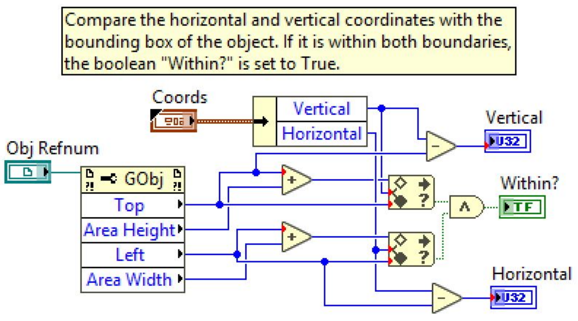

Compare the horizontal and vertical coordinates with the bounding box of the object. If it is within both boundaries, the boolean "Within?" is set to True.
Coords
Vertical
Horizontal
Obj Refnum
GObj
Top
Area Height
Left
Area Width
Vertical
Within?
Horizontal

Figure 82: The 'Bounding Box' VI.

Within this VI, the mouse coordinates are unbundled into vertical and horizontal and the object's bounds are established by summing 'Top' with 'Area Height' and 'Left' with 'Area Width'. The bounds are used as limits for two 'In Range and Coerce' functions which take in the mouse's coordinates to show the user if their mouse is within the perscribed bounds.

The VI's final operation is to subtract the 'Top' and 'Left' coordinates from the mouse's vertical and horizontal coordinates for normalization.

Within the process loop, the 'Bounding Box' VI's output coordinates are normalized by dividing them by the height and width of the tab control, and multiplying them by the maximum value they are to attain. Additionally, for the 'Vertical' coordinate, the value is negated and added to 90 to initialize it from the bottom instead of the top.

Using a case selector, the loop identifies whether the mouse coordinate beyond the lower or upper bound of the Tab Control in the x or y direction. In the former case, the value zero is passed to the elevation or azimuth indicators. In the latter case, 90 is passed to elevation if the bound is exceeded vertically, and 360 is passed to azimuth if it is exceeded horizontally. Otherwise, the bounded value is passed.