

PIER PORTAL

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Mechanical Engineering

by

Alec W. Hardy

June 2020

© 2020

Alec William Hardy

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: Pier Portal

AUTHOR: Alec W. Hardy

DATE SUBMITTED: June 2020

COMMITTEE CHAIR: John Ridgely, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: William Murray, Ph.D.
Professor of Mechanical Engineering

COMMITTEE MEMBER: Tom Moylan
Marine Operations Manager for the Center for
Coastal Marine Sciences

ABSTRACT

Pier Portal

Alec William Hardy

The Pier Portal marine monitoring system is an internet controlled underwater camera that will enable students, researchers, and the general public the ability to monitor underwater marine life and ocean conditions. Installed at Cal Poly's Center for Coastal Marine Science pier at Avila Beach, California, the camera can be commanded in real-time to raise or lower to any position between the top of the pier and the bottom of the ocean floor, while providing a live-stream video to the operator and general public. This thesis effort focused on the integration of the various subsystem components through software, and the development of an online interface to allow the remote control of the system and the ability to view the live-stream video from the camera. Missing or damaged mechanical and electrical components were successfully redesigned and replaced, and a more compact and serviceable winch and sheave system was designed and partially manufactured. A software set was written in Python, JavaScript, HTML, and IEC 61131-6 successfully connecting the system's control server, the winch's industrial PLC, the cameras inside the underwater pod, and the front-facing webserver. The system's internal network has been designed to allow internal components to communicate, and to allow external users the ability to securely control the system and view the video feed. Due to campus closure in early 2020, the final system was unable to be installed and tested, however, this document contains the entire system design and the next steps required to fully implement the system.

Keywords: underwater camera, marine monitoring, full stack development, industrial motion control, IEC 61131-3, Raspberry Pi, web server, Modbus TCP, client/server protocol, WebSocket, ASGI, Django, networks

ACKNOWLEDGMENTS

I would like to acknowledge Dr. Ridgely for his helpfulness and enthusiasm in this project – it was with his help and expertise that I was able to make so much progress on the Pier Portal system.

Additionally, I would like to thank Dr. Murray and Dr. Ridgely again for helping me discover my passion for robotics and mechatronics. It was after taking Introduction to Mechatronics with Dr. Murray in 2018 that I discovered my deep interest in computer systems and electronics, so much so that I pursued a year-long minor in computer science and a Master’s degree concentrated primarily in computer engineering after taking Dr. Ridgely’s Mechatronics and Control System Design classes.

Thank you to Mr. Tom Moylan for allowing me to work on your project and at your pier. Tom encouraged me to become certified in underwater SCUBA diving and taught me many things about the engineering problems in harsh marine environments.

Thank you to Mr. Rob Brewster for all your help in manufacturing the sheave and winch assembly. With my time spent working with Rob in the machine shop, I learned more practical shop skills than I had in my machining classes, and without his help the contributions to the mechanical system would not have been possible.

I would like to acknowledge and thank my family, especially my parents Marc and Tiffanne Hardy, for their support and motivation throughout my educational career. I am so proud to have them as my family; I try to work hard every day to make them proud, just as they have worked hard every day of their life to raise me to be the man I have become. I cannot thank you enough.

Lastly, I would like to acknowledge my grandpa, Frederick Miller, or grandpa “Dude” as I have called him. My grandpa has fostered my interest in the technological world, drawing diagrams of how refrigerators or airplanes or oil refinement works on napkins at the kitchen table since I was a small child. Thank you for sparking my everlasting interest in engineering.

TABLE OF CONTENTS

LIST OF TABLES	IX
LIST OF FIGURES	X
1. INTRODUCTION	1
1.1 Statement of Purpose.....	1
1.2 Description of System	1
1.3 Project Scope.....	2
2. BACKGROUND	3
2.1 Existing Underwater Monitoring Solutions	3
2.2 Summary of Previous Contributions	5
2.3 Starting System State	16
3. CONTRIBUTIONS AND DEVELOPMENT	17
3.1 Winch and Sheave Assembly.....	17
3.2 Winch Servo Control System.....	24
3.3 ServoPak and Servo Testing	26
3.3.1 PLC DIP switch configuration.....	26
3.3.2 PC Network Configuration	27
3.3.3 MotionWorks IEC Software	28
3.4 Servo Network Control	29
3.4.1 System Network Configuration	30
3.4.2 PLC Modbus Communication	30
3.4.3 PC Modbus Communication.....	33
3.4.4 PLC Servo Control Firmware	34
3.5 Pod System.....	38
3.5.1 Pod Internals	39
3.5.2 Electronic Interfacing.....	41
3.6 Pier Portal Server	44

3.6.1	Web Server.....	45
3.6.2	Remote Access.....	46
4.	DESIGN DETAILS	49
4.1	Winch and Sheave Assembly.....	49
4.2	Pier Portal Server	53
4.2.1	Dependency Installation.....	54
4.2.2	Server Start.....	54
4.2.3	Auto-start on boot	55
4.2.4	Web Interface.....	56
4.2.5	Django Project Overview.....	58
4.2.6	PPWebServer Application Details.....	58
4.2.7	PPLive Application Details - User Interface	61
4.2.8	PPLive Application Details – ASGI Application and Web Sockets.....	62
4.2.9	Winch Modbus Controller	67
4.2.10	Video Livestream.....	68
4.3	Camera Floodlight.....	69
4.4	Network Configuration	69
5.	FUTURE WORK.....	75
5.1	Final System Installation.....	75
5.2	Winch Servo Enclosure.....	76
5.3	Router Configuration.....	78
5.4	Apache/Nginx Web Server.....	78
5.5	TLS and Security.....	78
5.6	Mechanical System Safeguards.....	79
5.7	Additional Features	80
6.	PROJECT FILES	81
7.	REFERENCES	82

8. APPENDICES	84
Appendix A: Safety Operation Plan	84
Appendix B: Deflection and Square-stock selection of Sheave Holder Assembly	85
Appendix C: Winch Datasheet	87
Appendix D: Yaskawa ServoPak DIP Switch Configuration	88
Appendix E: New Winch Sprocket	89
Appendix F: Django pre-processed HTML source for user interface	90
Appendix G: Winch and ModbusConnection classes	92
Appendix H: Video Stream Source Code.....	94
Appendix I: Floodlight Control Code.....	96
Appendix J: List of TODO Items and Materials to Purchase.....	99

LIST OF TABLES

Table 1. Critical Buckling Loads for Sheave Holder Assembly.....	20
Table 2. Ladder Diagram Block Description.....	36
Table 3. Description of variables used in in Main V:Main.....	38
Table 4. Pod Computer Configurations.	41
Table 5. Pre-existing pod wiring arrangement.....	42
Table 6. Correct pinout of SubConn Connector	43

LIST OF FIGURES

Figure 1. Underwater track-fixed dive camera (Chen, 2008).	4
Figure 2. Underwater "MobyCam" used to track swimmers racing across a pool (Chen, 2008).	5
Figure 3. Track installation at Pier. (Balingit, Spieler, & Jen, 2013).	6
Figure 4. Close up rendering of first-generation sheave and winch assembly, on pier. (Balingit, Spieler, & Jen, 2013)	7
Figure 5. Underwater sea life growth on I-beam track after six months (Balingit, Spieler, & Jen, 2013).	8
Figure 6. Yaskawa Servo and Planetary Gearbox Reducer (Balingit, Spieler, & Jen, 2013).	8
Figure 7. First generation web-app to remotely control Pier Portal system (Klimaj, Noble, & Valdez).	9
Figure 8. Stand-alone track cleaning cart (Klimaj, Noble, & Valdez).	10
Figure 9. Pier2Pier Subsystem Flowchart (Nelson, Miller, Maalouf, & Shah, 2015).	11
Figure 10. Modbus TCP support in MP2600iec (Nelson, Miller, Maalouf, & Shah, 2015).	12
Figure 11. Pier2Pier Process Chart (Nelson, Miller, Maalouf, & Shah, 2015).	13
Figure 12. Pier Portal Pod (left) and cart (right). (Espinosa, Pier Portal: A Marine Monitoring System, 2017).	14
Figure 13. Winch Control Panel (Espinosa, Pier Portal: A Marine Monitoring System, 2017).	15
Figure 14. Completed 2017 Pier Portal work (Espinosa, Pier Portal Thesis Image Repository, 2017).	16

Figure 15. Original sheave holder assembly showing rust damage. (Espinosa, Pier Portal Thesis Image Repository, 2017)	17
Figure 16. Simple static mockup of sheave holder (left) and free body diagram of upright column (right).	18
Figure 17. Original Delrin sheave designed by first Pier Portal group. (Belis, et al., 2012).	21
Figure 18. CAD design of sheave holder.....	21
Figure 19. Reelcraft NORDIC series 2400 large-frame hose reel (Nelson, Miller, Maalouf, & Shah, 2015).....	22
Figure 20. Yaskawa ServoPak and planetary gear reducer (Nelson, Miller, Maalouf, & Shah, 2015).	23
Figure 21. Sheave holder and winch assembly.....	24
Figure 22. ServoPak Control System.....	24
Figure 23. C-curve (left) and D-curve (right) current-limiting breakers. (Automation Direct, 2020).	25
Figure 24. Configuration DIP switches on ServoPak PLC. (Yaskawa, 2018).	26
Figure 25. PC ethernet adapter configuration to allow Ad-Hoc communication with Yaskawa PLC.....	27
Figure 26. MotionWorks IEC2 Hardware Configuration window.	28
Figure 27. Ethernet switch inside servo control box.....	29
Figure 28. Pier Portal Network Configuration.....	30
Figure 29. PLC Modbus configuration.	31
Figure 30. Modbus FC input registers.	32

Figure 31. Two variables are assigned within the allowable Modbus FC register space.	32
Figure 32. Ladder Diagram of servo jog program with Modbus inputs.	35
Figure 33. Conversion of integer to LREAL datatype.....	37
Figure 34. Variables local to Main V:Main.	37
Figure 35. Pier Portal Pod, as received.	39
Figure 36. Pier Portal Pod High Level Electronic Schematic.....	40
Figure 37. Hosts file of PortalPi1 camera pod computer.....	40
Figure 38. Pin numbering scheme for the SubConn 12-contact underwater connector. ..	42
Figure 39. Ethernet Interface to Switch inside Pod.	44
Figure 40. Apache webserver working.	46
Figure 41. Dynamic DNS Record for pierportaltest.ddns.net.....	47
Figure 42. Domain name resolution for pierportaltest.ddns.net.	47
Figure 43. Port forwarding to allow public access to web server.	48
Figure 44. Completed Sheave Holder Assembly.....	49
Figure 45. Pod and cart cover can be easily attached and removed from the pier.....	50
Figure 46. Cart rubbing on track joint plate at position shown in red.	51
Figure 47. Assembly is bolted down to pier deck using J-bolts.	52
Figure 48. CAD Models of Winch Holder Assembly.....	52
Figure 49. Pier Portal winch, sheave holder, cart, pod, and track.	53
Figure 50. Pier Portal Server Application TLD.	54
Figure 51. Pier Portal Server bootup message showing ASGI server running.	55
Figure 52. Online interface pod control panel.	56
Figure 53. Pier Portal Web Interface.	57

Figure 54. Diagram of WebSocket data flow.	63
Figure 55. Live-stream webcam working during early development.	69
Figure 56. Ethernet interface configuration of pod computer.	70
Figure 57. Pod Pi computer communication over layer 2.	71
Figure 58. Pod switch (and winch panel) switch IP configuration. Note the switch is the first host on the 192.168.0.0/24 subnet.	71
Figure 59. Interface used to change IP address, Subnet mask, and Default Gateway of Yaskawa ServoPak winch controller PLC.	73
Figure 60. Internal IP addresses for each component.	73
Figure 61. Cable routing (left, (Google Maps, 2020)) and winch control panel location (right, (Espinosa, Pier Portal Thesis Image Repository, 2017)).	76
Figure 62. Enclosure for Yaskawa servo, shown in blue.	77
Figure 63. Mechanical safeguards to ensure proper tension (Espinosa, Pier Portal: A Marine Monitoring System, 2017).	80

1. INTRODUCTION

1.1 Statement of Purpose

The purpose of this thesis project is to complete the Pier Portal marine monitoring system to be used by scientists, researchers, students, educators, and the general public. The Pier Portal system allows users to remotely control an underwater camera attached to a vertical track at the Cal Poly Research Pier in Avila Beach, California. Users will be able to control the camera position and illumination and receive real-time video stream feedback, all through a standard web browser. In order to facilitate research efforts, video streams may be recorded and archived for later analysis. The project shall enable researchers to repeatably and accurately observe underwater ocean phenomena at fixed depths over time to monitor macroscopic ocean lifeforms and to track ocean conditions and climate change.

1.2 Description of System

The Pier Portal system is comprised of many mechanical, electrical, and software components that work together to achieve the system's purpose. The physical system consists of a vertical track affixed to the pier's support pylon, a pod attached to a cart which traverses the track, a sheave and a winch which raise and lower the cart/pod assembly, and an electrical control box which contains the winch's controller and the system's web server. This project is a continuation of the work completed by various mechanical engineering senior and thesis project groups; thus, the system has heavily evolved over time as each contributor has influenced the system. In particular, the project was most recently contributed to by Victor Espinosa III as partial fulfillment for his degree requirements for his master's in science in mechanical engineering in June 2017.

Most of the mechanical systems, barring the sheave and winch assembly, were adopted with little to no modification from Victor's efforts.

1.3 Project Scope

As most of the mechanical track, cart, and pod system has already been completed by previous contributors, the primary focus of this thesis project is to finish design and manufacture of the remaining mechanical systems and to tie all components together through tested, reliable software. The pre-existing winch and sheave assembly had been destroyed by the salty marine environment and have been redesigned and rebuilt for longevity. The system webserver, authentication system, video streaming and recording services, winch controller, and various additional complimentary components have also been designed and implemented.

2. BACKGROUND

2.1 Existing Underwater Monitoring Solutions

There are many existing solutions for underwater video monitoring. Modern solutions involve remote operated vehicles (ROVs) or fixed underwater data monitoring stations.

Engineers at the University of Washington have developed a “mechanical eye” that sits underneath the ocean’s surface and uses an array of sensors to monitor sea animals and collect data about the sea water. This system, called the “Adaptable Monitoring Package”, aka “AMP”, is a stationary system that uses a stereo camera system to collect image and video data, a hydrophone system to record underwater sounds, a small tidal turbine to collect ocean current data, and water quality sensors to gauge quality (McQuate, 2018). This system, however, is stationary and would not be able to provide the plant and sea-life growth monitoring capabilities that are needed at the Cal Poly pier to track growth over time on the pier’s support pylon.

One example of a track-fixed moving underwater camera is that of Garrett Brown, who designed the underwater dive camera used to capture Olympic high divers. One of Brown’s systems uses gravity to drop a camera-pod in an air-sealed tube to capture divers both in air and under water (Brown, 2017); the bottom part of the tube shown in Figure 1.



Figure 1. Underwater track-fixed dive camera (Chen, 2008).

Another one of Brown's underwater camera systems involves a horizontal track with pulleys attached to a camera cart on either side. The system, dubbed the underwater "MobyCam" tracks Olympic swimmers as they race across the pool. This system, unlike the gravity-powered vertical camera, uses cables to precisely control the position of the camera cart. The system is shown in Figure 2.

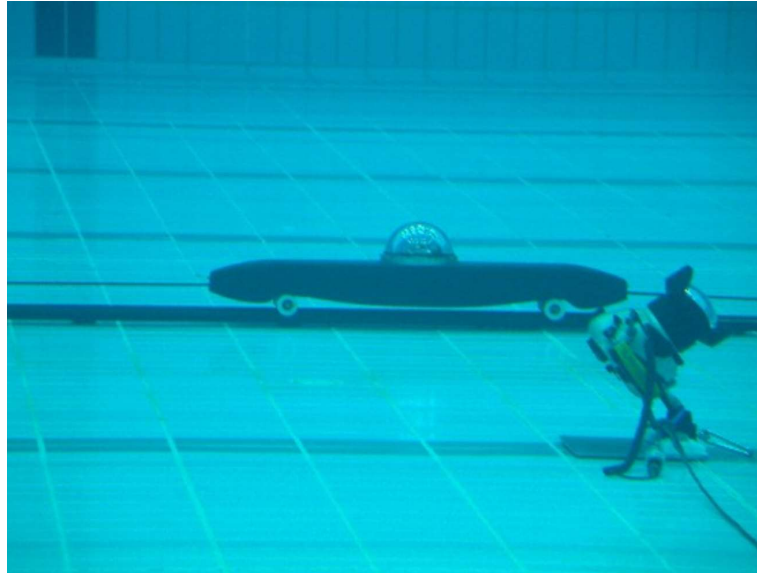


Figure 2. Underwater "Mobycam" used to track swimmers racing across a pool (Chen, 2008).

Because a lot of existing work has been completed on the Pier Portal project, the goal of this project is to utilize as much existing infrastructure as possible. As such, the existing track, cart, pod, and winch systems will be utilized. The current system uses a vertical track similar to Brown's vertical dive track, a sealed camera pod that is attached to a cart bound to the track and raised and lowered by a single cable, and a winch that raises and lowers the cart/pod assembly.

2.2 Summary of Previous Contributions

This thesis project is the continuation of the work performed by previous Cal Poly senior project groups and graduate students. Work began on the project in 2011 by six senior project group members: Andrew Belis, Andy Crafts, Jeremy DePangher, Aaron Hein, Michael Machado, and Aaron Poulos (Belis, et al., 2012). The project was sponsored by Tom Moylan, Cal Poly's Marine Operations Manager. This first group successfully designed and facilitated the install of the vertical FRP I-beam track and the first iterations

of the cart and pod systems. The track system, along with the first iteration of the sheave and winch assemblies, is shown in Figure 3. A close-up view of the first-generation winch and sheave assembly is included in Figure 4.

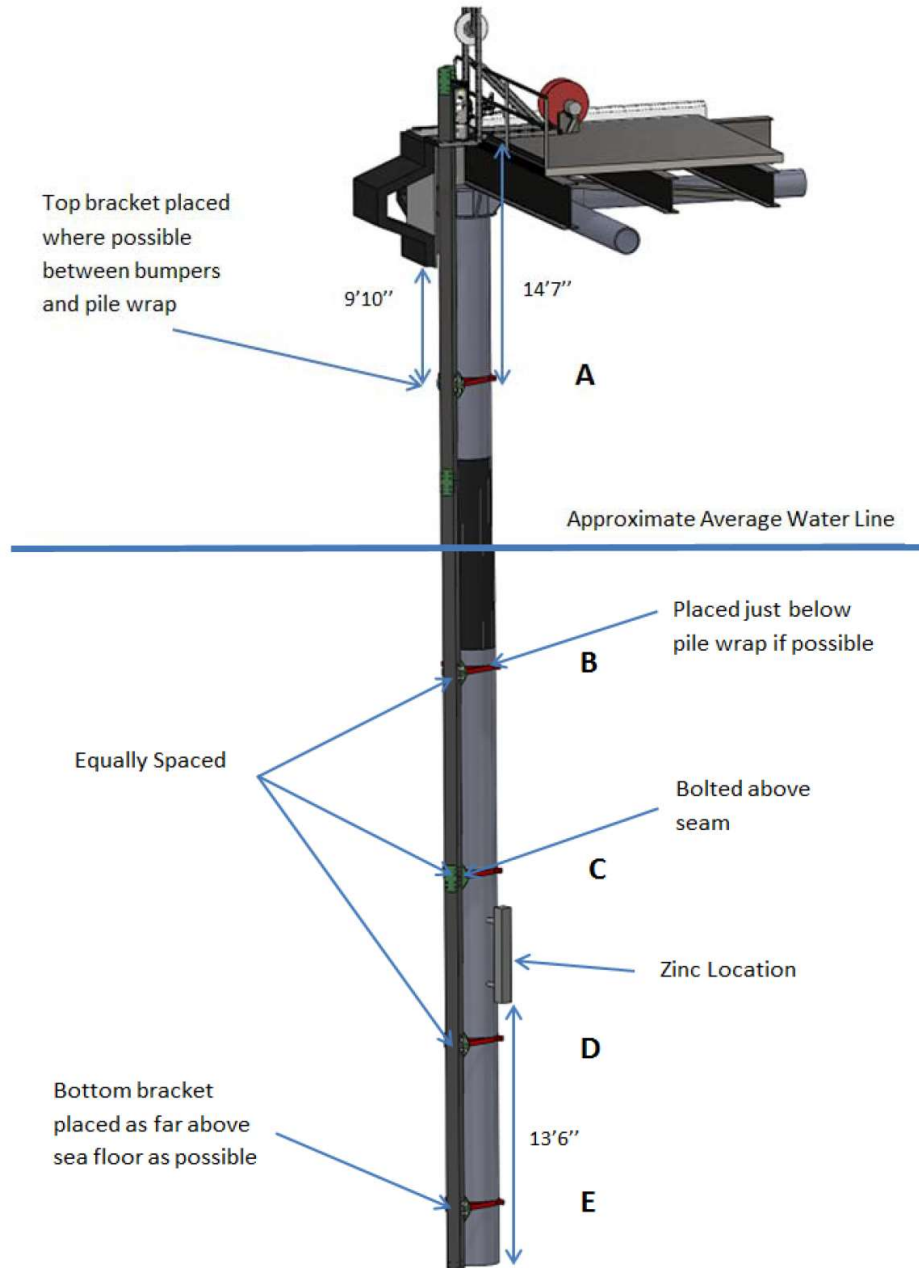


Figure 3. Track installation at Pier. (Balingit, Spieler, & Jen, 2013).

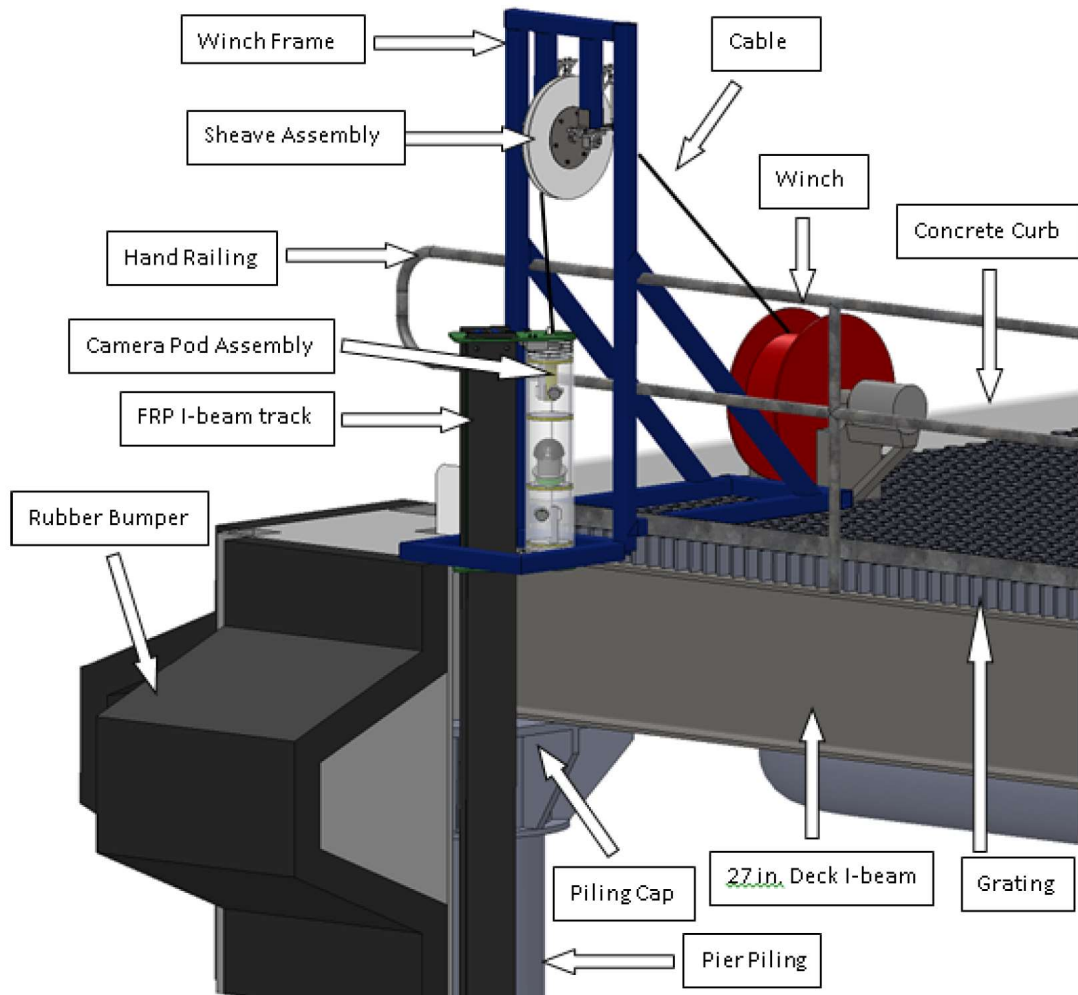


Figure 4. Close up rendering of first-generation sheave and winch assembly, on pier. (Balingit, Spieler, & Jen, 2013)

The following year, the Pier Portal project was taken over by Misha Balingit, Cory Spieler, and Aaron Jen as part of their 2013 senior project work. This group improved the pod's electronics by replacing them with a custom circuit board, named "Pawesome", and also developed a track cleaning solution to allow the cart's underwater movement to be unhindered by growing sea-life. It was determined that six months of undisturbed growth of underwater sea life would render the I-beam track completely useless without cleaning. A figure of the condition of the underwater track is shown in Figure 5.



Figure 5. Underwater sea life growth on I-beam track after six months (Balingit, Spieler, & Jen, 2013).

Most importantly, Balingit, Spieler and Jen were able to secure an AC Servo Motor from Yaskawa – the SGMV-05 500W, 1476rpm servo motor complete with the ServoPak PLC motor controller. This servo would be gear-reduced using the Shimpo Able VRL-120-20-K5-19EC1600 20:1 gear reducer, selected because the gear reducer is able to be attached directly onto the servo. The servo and gearbox are shown in Figure 6.

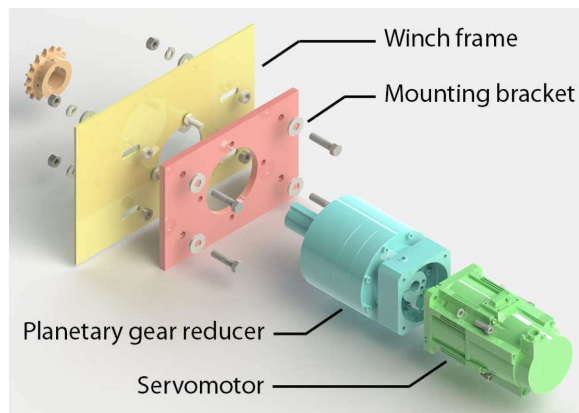


Figure 6. Yaskawa Servo and Planetary Gearbox Reducer (Balingit, Spieler, & Jen, 2013).

The third senior project group to work on the Pier Portal Project, Alex Klimaj, Patrick Noble, and Rudy Valdez, part of Pier Portal Phase 3, worked on making various improvements to the different subsystems, such as improvements to the cart cleaning brushes, stainless steel clamps to improve the rigidity of the I-beam, and a software redesign. The group created a simple web-app to be used for remote control of the system, as shown in Figure 7.



Figure 7. First generation web-app to remotely control Pier Portal system (Klimaj, Noble, & Valdez).

This group used a Raspberry Pi as their web server; and wrote their web server code in Python. Their code allowed for remote control of their system, as well as a local, administrative interface.

In addition to their work on a basic web interface, the Pier Portal Phase 3 group designed a stand-alone track cleaning cart, as shown in Figure 8, that was separate from the cart/pod assembly.



Figure 8. Stand-alone track cleaning cart (Klimaj, Noble, & Valdez).

The fourth group to work on the Pier Portal project in 2015, under the name “Pier2Pier”, consisting of Kevin Nelson, Tony Miller, Paul Maalouf, and Chahan Shah, primarily worked on the software to be used by the Pier Portal system. The group developed a web interface using Ruby on Rails and JavaScript that supported video-streaming and pod position control through a user front-facing system. During this iteration, the group used the existing “P^{awesome}” boards; their system flowchart is shown in Figure 9.

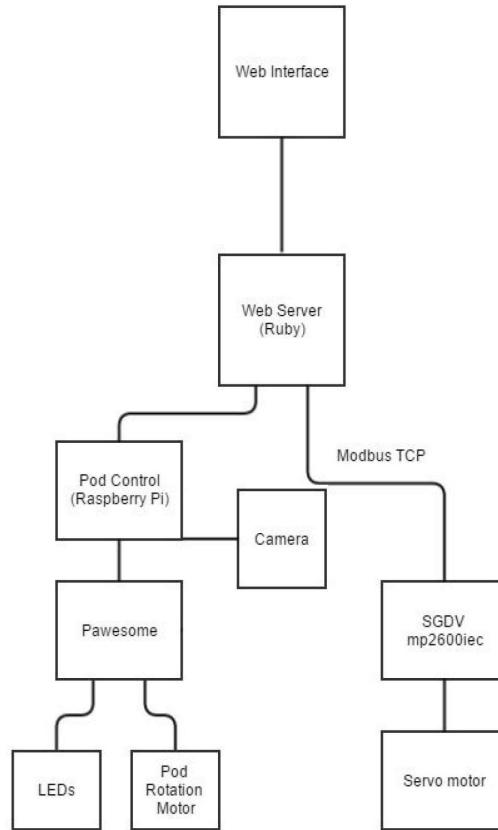


Figure 9. Pier2Pier Subsystem Flowchart (Nelson, Miller, Maalouf, & Shah, 2015).

The group performed a lot of underlying research of the best methods to achieve communication between the different subcomponents of the Pier Portal system, such as an analysis of the communication protocols that can be used between the web server and the servo controller (SGDV MP2600IEC, as shown in the diagram). As will be used in this iteration of the Pier Portal project, Modbus TCP was selected as the best candidate communication protocol, as the MP2600IEC PLC has built-in support for Modbus by linking variable addresses to a Modbus function, as shown in Figure 10.

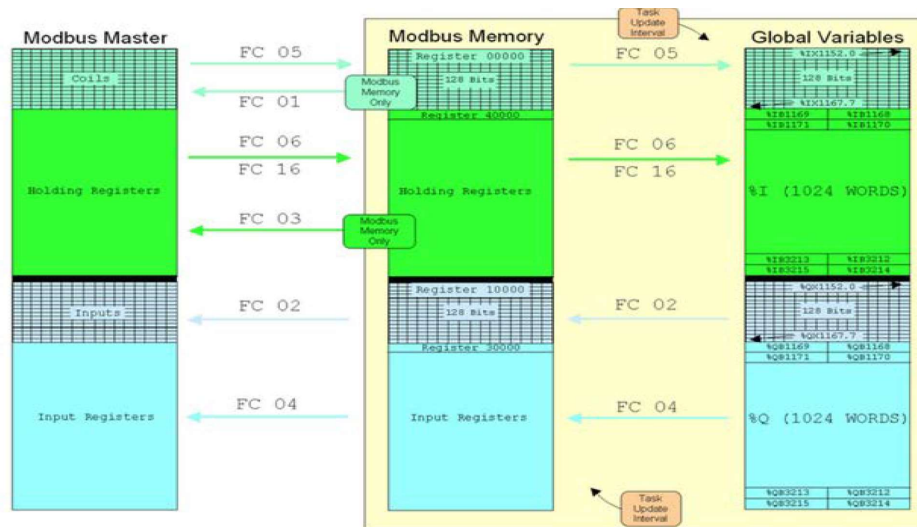


Figure 10. Modbus TCP support in MP2600iec (Nelson, Miller, Maalouf, & Shah, 2015).

The Pier2Pier “Pier Server” is “responsible for receiving requests/commands from web clients and delegating tasks to [the] various sub systems.” The system has different processes running on different pieces of hardware, all working in tandem to deliver the desired cohesive user experience. The web server, referred to as the “rails server” by the Pier2Pier group, is “the primary handler for user interaction” and “run[s] on a physical server located at the pier.” This server ran a “Servo Daemon”, which “receive[d] movement commands from the rails back end” web server and “act[ed] as a Modbus Client [to] format the commands into Modbus Packets to be sent to the Modbus Server running on the mp2600iec” (Nelson, Miller, Maalouf, & Shah, 2015).

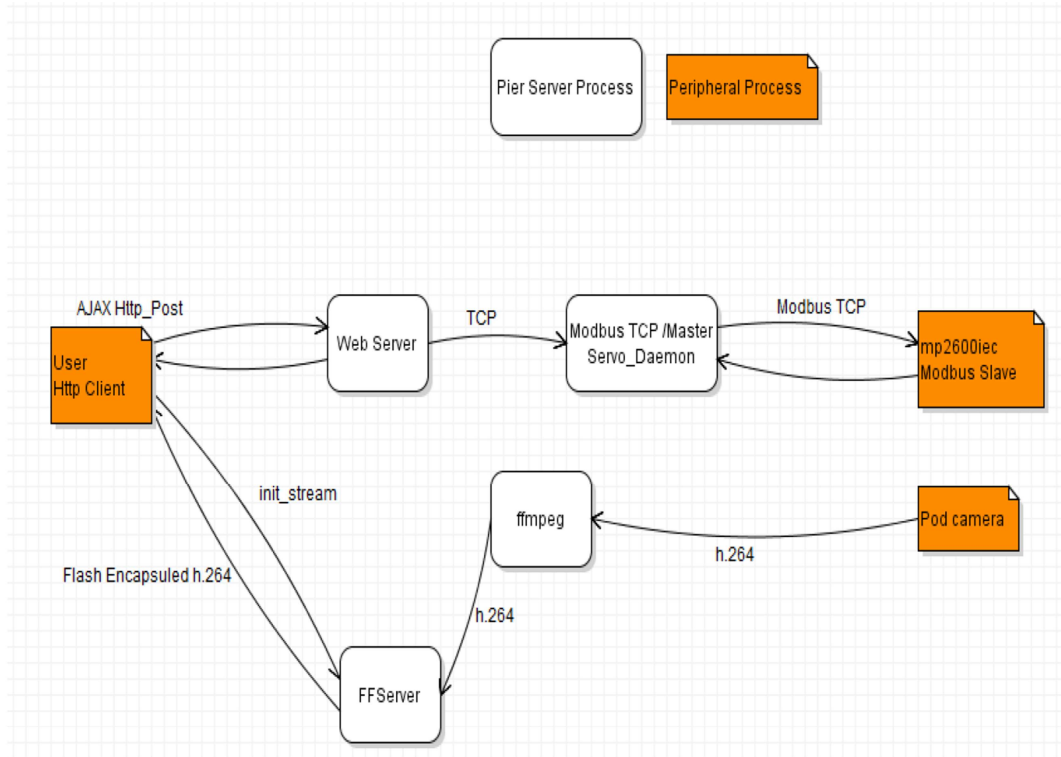


Figure 11. Pier2Pier Process Chart (Nelson, Miller, Maalouf, & Shah, 2015).

Most recently, in 2017 Victor Espinosa worked on the Pier Portal project as his master’s thesis project, “Pier Portal: A Marine Monitoring System”. Victor’s work consisted of modifications to the pod, cart, winch, sheave frame, and winch control panel electronics. At the end of Victor’s work, most of the mechanical and electrical systems were in working order and ready for software integration and implementation.

The size of the pod was greatly reduced and the internal electronics were significantly simplified. The new pod design featured three cameras at ninety-degree offsets, each connected to a Raspberry Pi computer through its onboard CSI-2 header. Rather than reusing the unsupported custom circuit boards and software developed by the previous groups, the Raspberry Pi computers were used to process and serve the live video stream from the cameras. Each camera was mounted onto a new custom PCB containing

floodlight LEDs and a temperature sensor, interfaced through GPIO and SPI pins. A rendering of the new pod, and the new cart to guide the pod along the pier's track, is shown below in Figure 12.

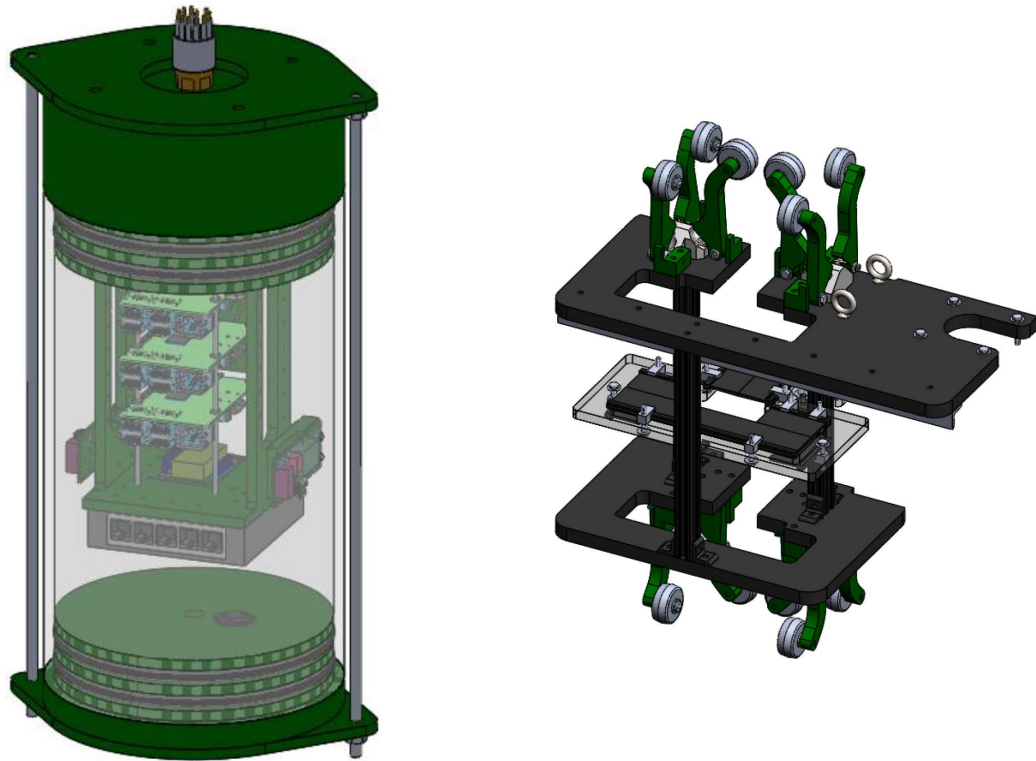


Figure 12. Pier Portal Pod (left) and cart (right). (Espinosa, Pier Portal: A Marine Monitoring System, 2017).

Work on the winch control panel electronics included the addition of a power filter, a magnetic disconnect circuit breaker, a magnetic contactor E-stop, and the addition of internal circuit breakers. Each of these additions to the electrical system provide increased safety and system longevity. The winch control system operates at high voltage, requiring 3-phase 208VAC. The digital electronics inside, such as the PLC and network switch, require the industry standard 24VDC to operate, thus a large, clean

power supply module is required as well. The internals of the control box, as completed in 2017, are shown in Figure 13.

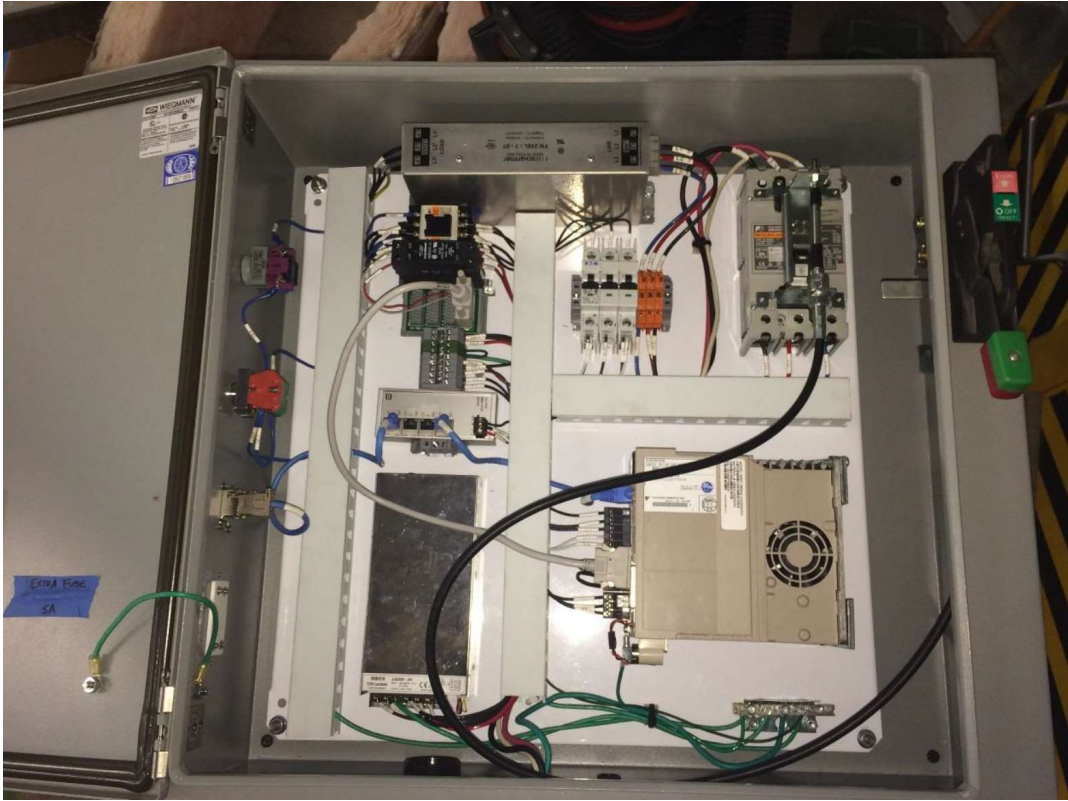


Figure 13. Winch Control Panel (Espinosa, Pier Portal: A Marine Monitoring System, 2017).

By the end of the 2017 work, the Pier Portal's mechanical and electrical system was ready to be installed. The track, cart, pod, sheave, and winch assemblies were all complete and independently verified working. The completed mechanical parts are shown in Figure 14. Very little software and no network planning had been completed, and while the software completed by the previous senior project groups could serve as a proof-of-concept, a more permanent and robust software solution is necessary to push the Pier Portal system into live production.

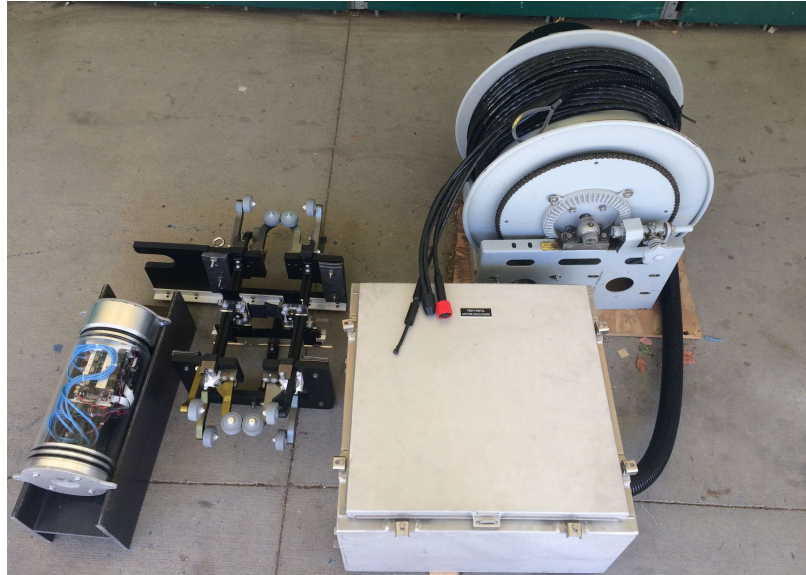


Figure 14. Completed 2017 Pier Portal work (Espinosa, Pier Portal Thesis Image Repository, 2017).

2.3 Starting System State

The camera pod, winch, and cart were found in on-campus storage in unknown condition. The camera pod, when connected to power, would not turn on, and when connected to ethernet given the pinout diagram found in previous thesis reports, no network connection could be established even when powering the internal switch directly. The winch servo control panel was found with a circuit schematic inside, but the internal breaker would trip upon power-up, and some internal wires were disconnected. The sheave holder assembly from previous projects had deteriorated from the harsh marine conditions and was disposed of, needing to be redesigned and rebuilt. The vertical track on the pier was intact but covered in sea life below the water level. The wiring on the pier had been destroyed. Other small components, such as the drive sprocket for the winch, have never been found.

3. CONTRIBUTIONS AND DEVELOPMENT

Development of the Pier Portal system requires work on many different subsystems in parallel. Because some of the systems involve heavy moving mechanical parts and high voltages, strict safety precautions are taken during all work on the project. This safety plan is outlined in Appendix A: Safety Operation Plan.

3.1 Winch and Sheave Assembly

An entirely new sheave and winch assembly needed to be designed and manufactured. The purpose of this assembly is to guide the winch cable so that it can pull the pod vertically up and down along the track. Analysis of the remaining sheave assembly components from the previous iteration shows that the new assembly must be more capable of withstanding the corrosive marine environment found at the Cal Poly pier. Figure 15 shows the original sheave assembly and corrosive damage that had occurred over time.



Figure 15. Original sheave holder assembly showing rust damage. (Espinosa, Pier Portal Thesis Image Repository, 2017)

The new sheave holder assembly has a design load of above 500 pounds and is manufactured out of 304 stainless steel in order to prevent rust and corrosion from the marine environment. This design load is extremely conservative, as the true weight of the cart and pod do not exceed 50 pounds; however, in the event of human or marine life interference with the cart or pod system, the sheave assembly is not a designed point of failure. Static analysis was performed on a mockup design based on the existing structure, shown in Figure 16.

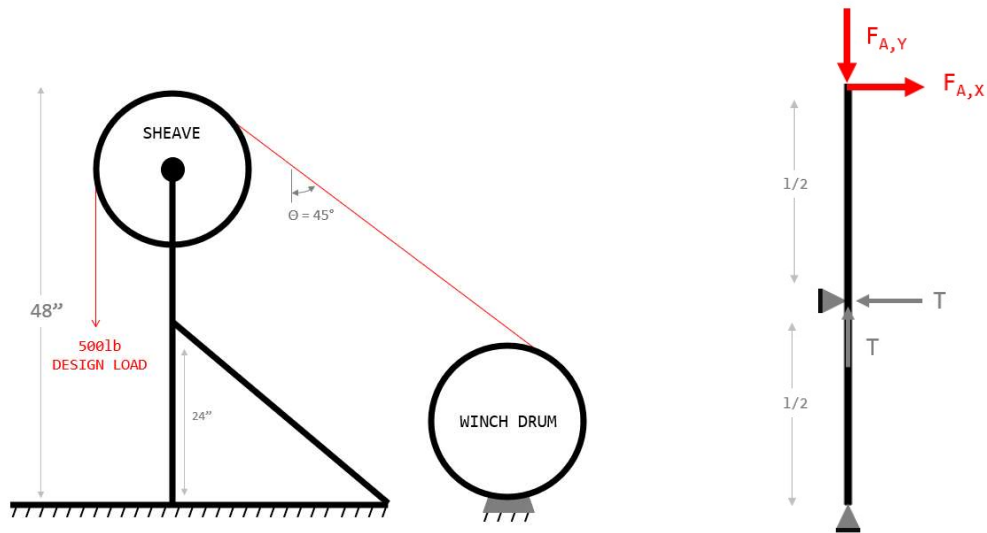


Figure 16. Simple static mockup of sheave holder (left) and free body diagram of upright column (right).

The forces within the upright column were analyzed using Castigliano's method to determine the maximum deflection of the sheave in the horizontal direction towards the winch drum. In this analysis, in order to account for the reaction forces of the diagonal support beam, and to make the analysis statically determinant, the intersection between the pin and the upright column was modeled as a pin. With the y-direction pointing

vertically upwards positive, and the x-direction horizontal positive to the right, the moment within the upright column at position y is given by Equation 1 below.

$$M(y) = \begin{cases} -F_{A,x}(y) & 0 \leq y \leq l/2 \\ -F_{A,x}(l-y) & \frac{l}{2} < y < l \end{cases} \quad (1)$$

The horizontal deflection is calculated as shown in Equation 2 below.

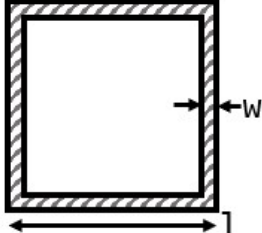
$$\delta_{F_{A,x}} = \frac{1}{EI} \left[\int_0^{l/2} (-F_{A,x}(y))(-y)dy + \int_{l/2}^l (F_{A,x}(l-y)^2)dy \right] \quad (2)$$

Equation 2 simplifies to $\delta_{F_{A,x}} = \frac{1}{EI} \left[\frac{F_{A,x}l^3}{12} \right]$. Assuming a modulus of elasticity of 28,000ksi for 304 stainless steel (ASM Aerospace Specification Metals Inc., 2020), 1” wall, 0.065” thick square stock would deflect by approximately four inches, and 2” wall, 0.12” thick square stock would deflect by approximately a quarter inch under the design load of 500 pounds.

A buckling analysis was also performed on the mockup model to determine the critical pressure at which Euler Buckling of the upright column may occur. Because the column is fixed on the bottom and has a free end, the Euler Buckling k value of 2.0 was used for analysis. The critical load, P_{cr} , is given by the equation $P_{cr} = \frac{\pi^2 EI}{(kl)^2}$. For the readily-available square stock tubing sizes of 1”, 1.5”, and 2” walls and 0.065” and 0.12” thicknesses, the critical buckling loads were calculated and tabulated in Table 1.

Table 1. Critical Buckling Loads for Sheave Holder Assembly.

Wall l [in]	Thickness w [in]	Mass moment of Inertia I [in ⁴]	Critical Load P _{cr} [lbf]
1	0.065	0.03559	4169
1	0.12	0.05553	6660
1.5	0.065	0.12831	15390
1.5	0.12	0.21184	26608
2	0.065	0.31431	37699
2	0.12	0.53375	64020



Hand calculations for deflection using Castigliano’s method and Euler buckling are included in Appendix B: Deflection and Square-stock selection of Sheave Holder Assembly.

From the static analyses, it is proven that buckling is not a design concern but that the thicker wall square tubing is necessary to prevent deflection. In order to facilitate ease of welding, and because of a negligible price difference among sizes, the frame was manufactured out of 2-inch, 0.12-inch thick square stock tubing.

The original sheave, shown in Figure 17, was designed by the first Pier Portal contributors and was salvaged and able to be re-used as the cable guide. The sheave is made of Delrin (polyoxymethylene) plastic, is 20-inches in outer diameter, and it contains a centered contour to guide the winch cable.



Figure 17. Original Delrin sheave designed by first Pier Portal group. (Belis, et al., 2012).

A CAD design of the sheave holder assembly is shown below in Figure 18.



Figure 18. CAD design of sheave holder.

The winch that raises and lowers the pod consists of an off-the-shelf Reelcraft NORDIC series 2405 heavy duty large frame hose reel. The datasheet for the winch is included in Appendix C: Winch Datasheet. The winch is shown below in Figure 19.



Figure 19. Reelcraft NORDIC series 2400 large-frame hose reel (Nelson, Miller, Maalouf, & Shah, 2015).

Powering the winch is a Yaskawa SGD V MP2600iec ServoPak reduced by a 20:1 planetary gear reducer. The ServoPak and gear reducer are shown in Figure 20; note that the sprocket attached to the reducer's output has been replaced.



Figure 20. Yaskawa ServoPak and planetary gear reducer (Nelson, Miller, Maalouf, & Shah, 2015).

In order to provide ease of maintenance, cleaning, and removal, the winch will be attached to the sheave holder frame by steel gussets and bolts. Because the sheave holder frame is made of 304 stainless steel and the winch body is made of non-stainless steel; in order to provide galvanic isolation, plastic or another insulating material will be used as spacers between the winch frame and the horizontal winch holder attachment legs. These same spacers are used to separate the bottom of the assembly from the pier's decking, described further in section 4.1.

The winch will be covered by a “doghouse” enclosure to protect the ServoPak from the environment. A CAD rendering of the winch-attached sheave holder (uncovered) is shown below in Figure 21.

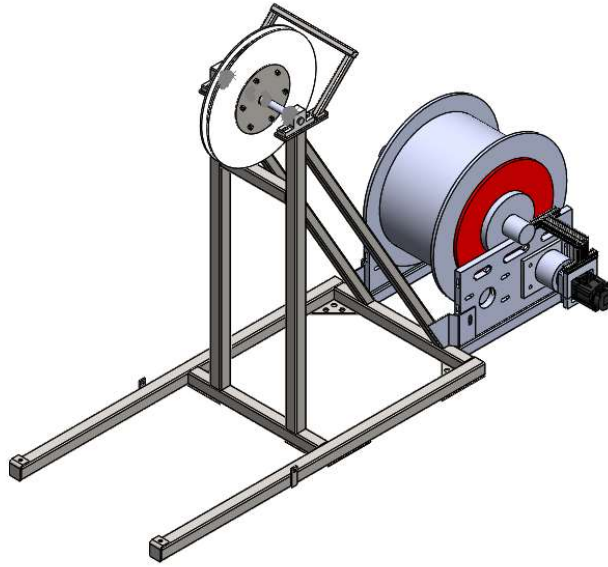


Figure 21. Sheave holder and winch assembly.

3.2 Winch Servo Control System

The electronics that control the ServoPak and allow for Pier Portal server communication with the underwater pod and ServoPak had been designed and provided by Victor Espinosa (Espinosa, Pier Portal: A Marine Monitoring System, 2017). The system, as found, is shown in Figure 22.

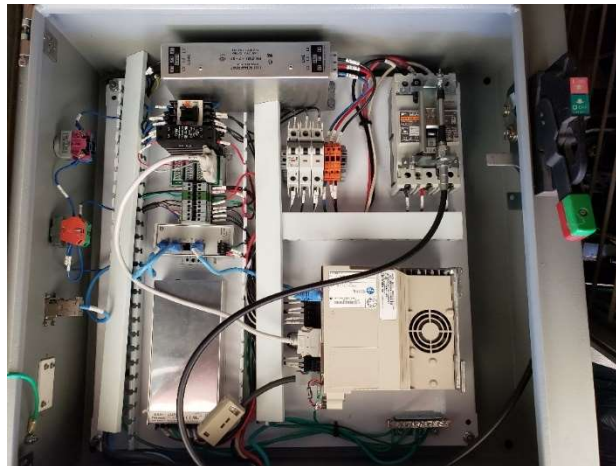


Figure 22. ServoPak Control System.

While testing the original control system with and without the servo attached, the internal circuit breakers would trip upon powerup. The inrush current required to power the ServoPak PLC controller was beyond the C-curve breaker's limit, and while under steady-state condition the 1.5A C-curve breaker sufficed, the large inrush current would trip the breakers. The problem was resolved by replacing the C-curve 1.5A breakers with a D-curve 3.0A breaker, model FAZ-D3-3-NA, as shown in Figure 23. After this replacement, the control system would power on correctly.

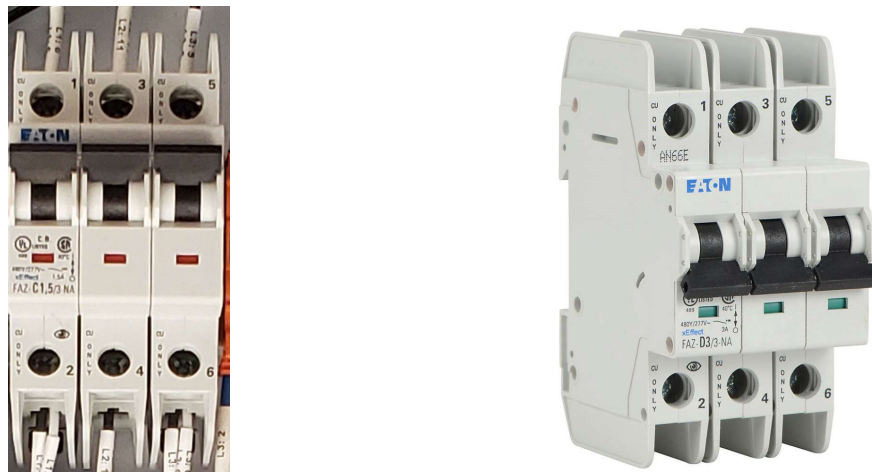


Figure 23. C-curve (left) and D-curve (right) current-limiting breakers. (Automation Direct, 2020).

The ServoPak PLC, which allows for programmable control of the winch servo, is ethernet-addressable through both of its two ethernet ports. PLC Programming over ethernet is made possible through Yaskawa's MotionWorks IEC software, and PLC communication is made possible through various industry standard communication protocols. Licenses for version 2 of MotionWorks IEC Professional and Express versions were provided to Cal Poly by Yaskawa.

3.3 ServoPak and Servo Testing

As a preliminary test of the servo and control system, a servo “Test Move” was executed to ensure that the PLC, motor, encoder, and network configuration were operating correctly. In order to perform this test, the ServoPak and the programming test computer needed to be configured.

3.3.1 PLC DIP switch configuration

The DIP switches on the ServoPak, shown in Figure 24, needed to be configured such that the PLC was in E-INIT (Ethernet Initialization) mode and that the SUP firmware programming mode was enabled. DIP switch configuration details are included in Appendix D: Yaskawa ServoPak DIP Switch Configuration. With switch #1, #2, and #4 set to on, the PLC is programmable and addressable via ethernet Port A at the IP address 192.168.1.1.



Figure 24. Configuration DIP switches on ServoPak PLC. (Yaskawa, 2018).

3.3.2 PC Network Configuration

In order to communicate over IP with the PLC set at 192.168.1.1, the programming computers (PC) needed an Ad-Hoc network. The PC's ethernet adapter was configured to enable TCP/IPv4 communication. Because the PLC's IP address is pre-set as 192.168.1.1, the IP address of the PC's ethernet adapter must be set so some IP within the same subnet but at a different address. The IP address 192.168.1.100 with the Subnet mask of 225.225.225.0 allowed communication. A screenshot of the configuration is shown in Windows 10 below in Figure 25.

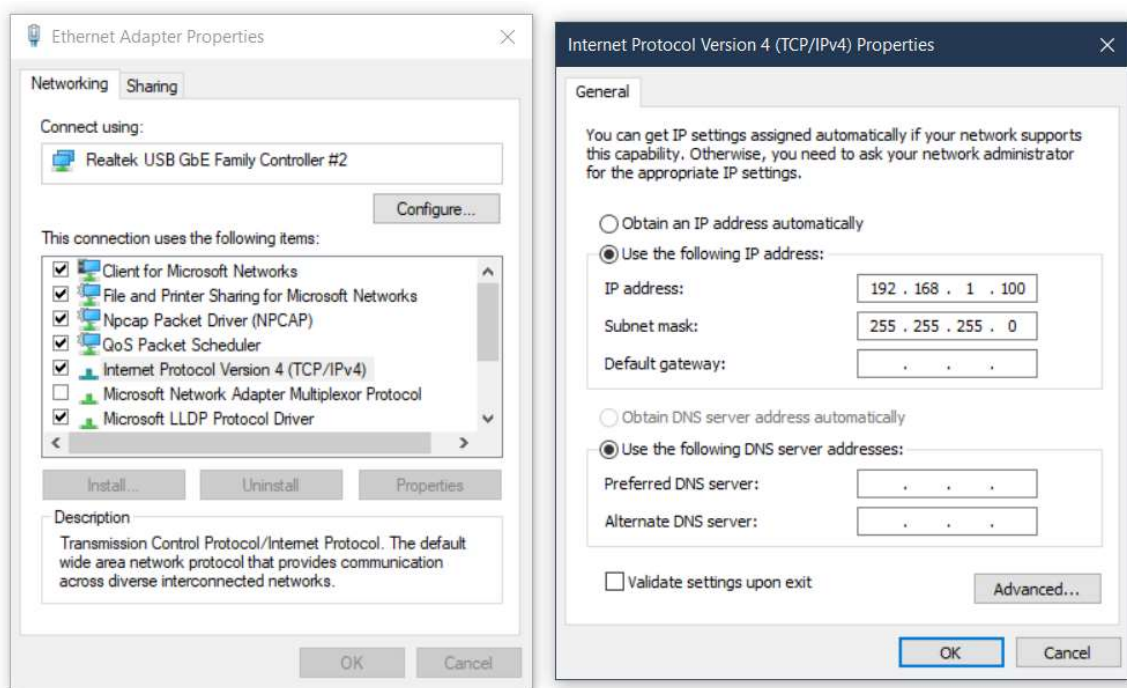


Figure 25. PC ethernet adapter configuration to allow Ad-Hoc communication with Yaskawa PLC.

A quick communication test can be accomplished by pinging the PLC's IP address through a command line window by issuing the "ping 192.168.1.1" command. Note that if the adapter's IP address was incorrectly set as 192.168.1.1, then the ping test would

still pass because the PC would be pinging itself via loopback (the same behavior as if pinging 127.0.0.1). This phenomenon was very problematic when debugging communication issues.

3.3.3 MotionWorks IEC Software

Yaskawa's MotionWorks IEC2 software is used to configure and program the winch controller PLC. During normal operation, this software is not needed nor used, however, it is extremely useful during debugging and testing. Following software installation and license activation, the Hardware Configuration window can be used to perform test moves of the attached (default configuration) servo, as shown in Figure 26. Note that in order to launch the MotionWorks software, the CrypKey License service must be running – make sure that this service is enabled and running before launching the software; launch services.msc and start the executable at *C:\windows\system32\crypserv.exe* as a service.

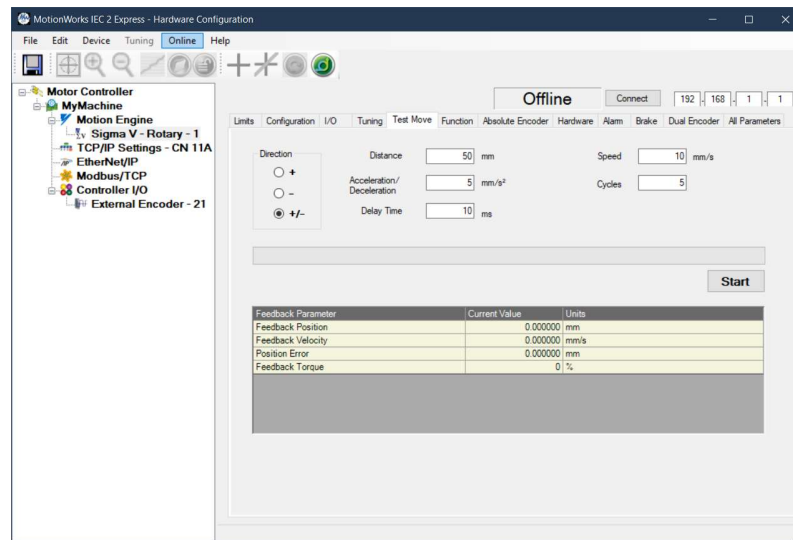


Figure 26. MotionWorks IEC2 Hardware Configuration window.

3.4 Servo Network Control

Once Ad-Hoc network communication is established and verified working, the PLC must be addressable in an infrastructure network environment so that it can be remotely controlled by the Pier Portal server. The servo control box contains a Ha-VIS eCon 2000 unmanaged ethernet switch, shown highlighted in Figure 27. In its current configuration, the switch is only connected to the PLC and has an auxiliary ethernet passthrough port on the side and a long ethernet cable to be attached and connected to the switch inside the underwater pod.

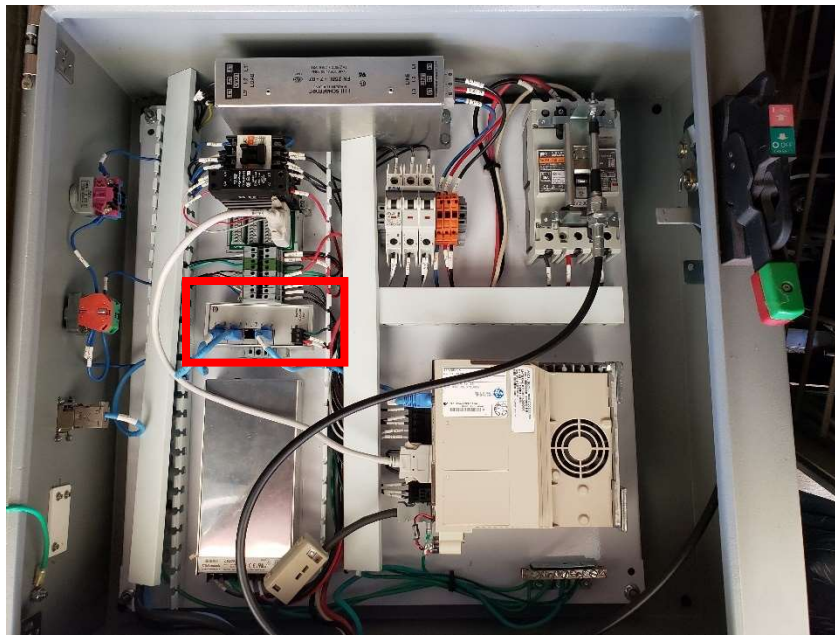


Figure 27. Ethernet switch inside servo control box.

The IP address, gateway, and subnet of the PLC are configurable through the Hardware Configuration window described previously. The PLC's gateway is to be configured to match the internal IP address of the router that the Pier Portal's server is connected to. Any IP address within the subnet range can be used for the PLC so long that it is not reused by any of the computers inside the pod or the Pier Portal server.

3.4.1 System Network Configuration

Once complete, the proposed network configuration for the Pier Portal system is as follows in Figure 28.

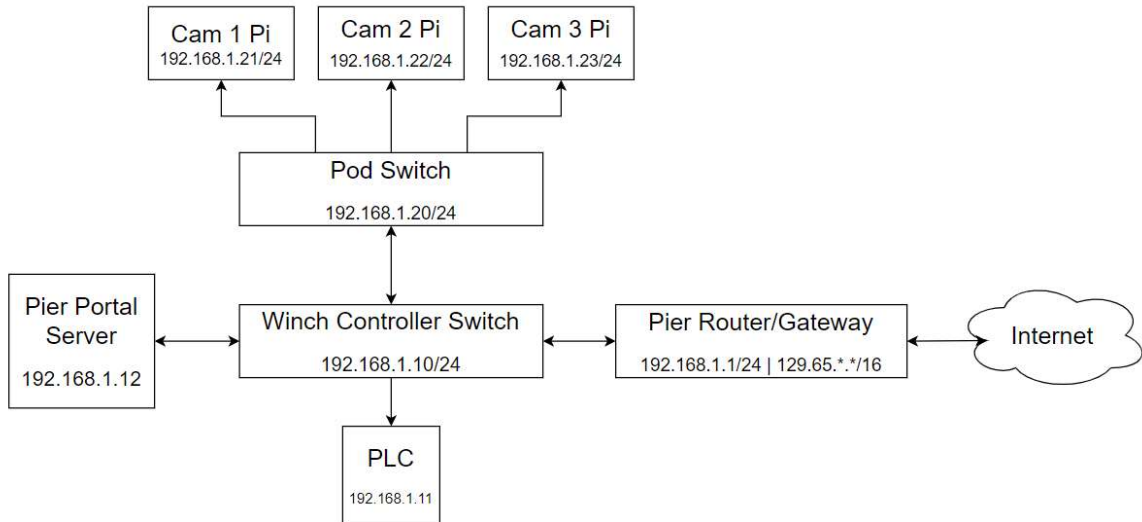


Figure 28. Pier Portal Network Configuration.

When programming or debugging the PLC, the auxiliary ethernet port on the side of the control box, connected to the winch control switch, shall be used in an Ad-Hoc configuration – again ensuring that the ethernet adapter’s IP address does not conflict with any of the device’s IP addresses. A simple network connectivity test consists of pinging each of the system’s computer systems from the winch controller switch connected programming computer (PC). Note that the PLC should not be reachable from the external gateway.

3.4.2 PLC Modbus Communication

In a limited configuration with only the Pier Portal Server, PLC, and programming computer (PC) connected through the winch control switch, commands can be sent from

the PC to the Pier Portal Server, and those commands will be relayed to the PLC. Once the web interface is complete, the commands provided by the users will mimic the commands being sent by the PC. Communication between the server and PLC can be accomplished using the Modbus protocol, with the PLC configured as a Modbus slave. This configuration can be set through the MotionWorks Hardware Configuration window, as shown in Figure 29.

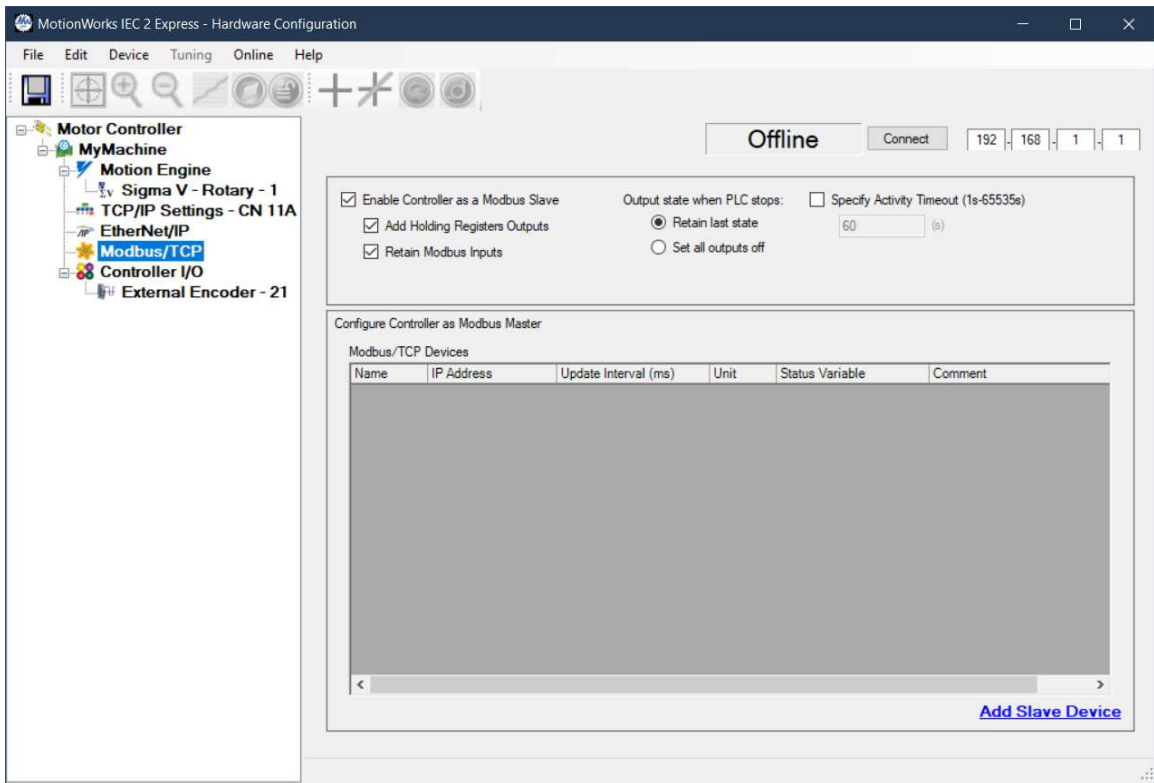


Figure 29. PLC Modbus configuration.

With Modbus, the contents of the PLC’s registers can be read and written to. This can be utilized to write servo setpoints to jog the servo/winch to the correct position such that the pod is displaced to its desired underwater position. When the PLC is configured to act as a Modbus slave, a Modbus FC block is created with 1024 registers. In a test program,

these registers defaulted to %IB28672-%IB30719, as shown in Figure 30. These registers are binary input registers that can be both read and written to with Modbus.

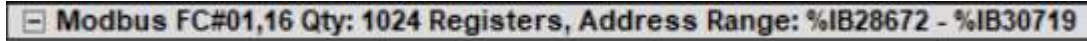


Figure 30. Modbus FC input registers.

Within the defined FC block register range, individual registers can be assigned to variables, such as the integer “*goToAbsPos*” and the Boolean “*axisDoExecute*” shown below in Figure 31. These variables are used in a simple Modbus-controlled servo setpoint program described in more detail in a later section. Note that integer *goToAbsPos* is assigned to register %IW28673, where the contents of *goToAbsPos* consist of an entire word’s length of input memory (hence the %IW prefix), and the boolean *axisDoExecute* is assigned to the zeroth bit of the binary input register %IX28672.0 (hence the %IX prefix and the .0 suffix).

☐ Modbus FC#01,16 Qty: 1024 Registers, Address Range: %IB28672 - %IB30719				
<i>goToAbsPos</i>	INT	VAR_GLOBAL		%IW28673
<i>axisDoExecute</i>	BOOL	VAR_GLOBAL		%IX28672.0

Figure 31. Two variables are assigned within the allowable Modbus FC register space.

Within the Modbus protocol, with the above configuration, the variable *axisDoExecute* can be referenced as Modbus register zero because it has the lowest defined register number within the Modbus FC range, and the variable *goToAbsPos* can be referenced as Modbus register one because it is the next register following *axisDoExecute*. To set *axisDoExecute* to true through Modbus, simply write the value 1 to register 0, and to assign an integer value to *goToAbsPos*, simply write the value of the desired setpoint to register 1.

3.4.3 PC Modbus Communication

The Github user “sourcepearl” wrote and maintains a Python library *pyModbusTCP* which allows for easy Modbus communication over TCP/IPv4 (sourcepearl, 2019). By simply importing *pyModbusTCP.client*, the module can be used to easily read and write the ServoPak PLC’s Modbus FC registers. A simple function to verify connectivity by reading the first 100 PLC registers is accomplished using the source below, where the PLC is in E-INIT mode so its IP address is 192.168.1.1.

```
def doTest():
    # TCP auto connect on modbus request, close after it
    c = ModbusClient(host="192.168.1.1", auto_open=True, auto_close=True)

    regs = c.read_holding_registers(0, 100)
    if regs:
        print(regs)
    else:
        print("read error")
```

With communication verified working, the following script below allows for writing servo setpoints to the aforementioned variables *goToAbsPos* and *axisDoExecute*. The script works by setting the boolean *axisDoExecute* to false (0), requesting a user input for a *goToAbsPos* integer if the script was successfully able to write to the *axisDoExecute* register, and then writing the input value to the *goToAbsPos* register and setting *axisDoExecute* to true again. With some firmware written to the PLC, this script can be used to jog the servo.

```

from pyModbusTCP.client import ModbusClient
import time

def doWrite(goToPos):
    """Commands PLC to move servo to a given position"""
    c = ModbusClient(host="192.168.1.1", auto_open=True, auto_close=True)

    if c.write_single_register(1,goToPos):
        print("write ok")
    else:
        print("write error")

if __name__=='__main__':
    c = ModbusClient(host="192.168.1.1", auto_open=True, auto_close=True)
    while(True):
        if c.write_single_register(0,0):
            print("Waiting for instruction")
        goto = int(input("Enter Go To Pos: "))
        doWrite(goto)
        if c.write_single_register(0,1):
            print("Going!")
        time.sleep(.1)

```

3.4.4 PLC Servo Control Firmware

A simple PLC program can be written using the IEC 61131-1 “Ladder Logic” language (Yaskawa, 2014) to energize and jog the servo to the position requested via Modbus. A cyclic main task is written which is always ran endlessly by the PLC. For the test move program, this cyclic task sets up and configures the motor controller by enabling power, and then sets the servo’s position setpoint to the user’s desired position as provided over Modbus. The ladder diagram for the main task is shown in Figure 32.

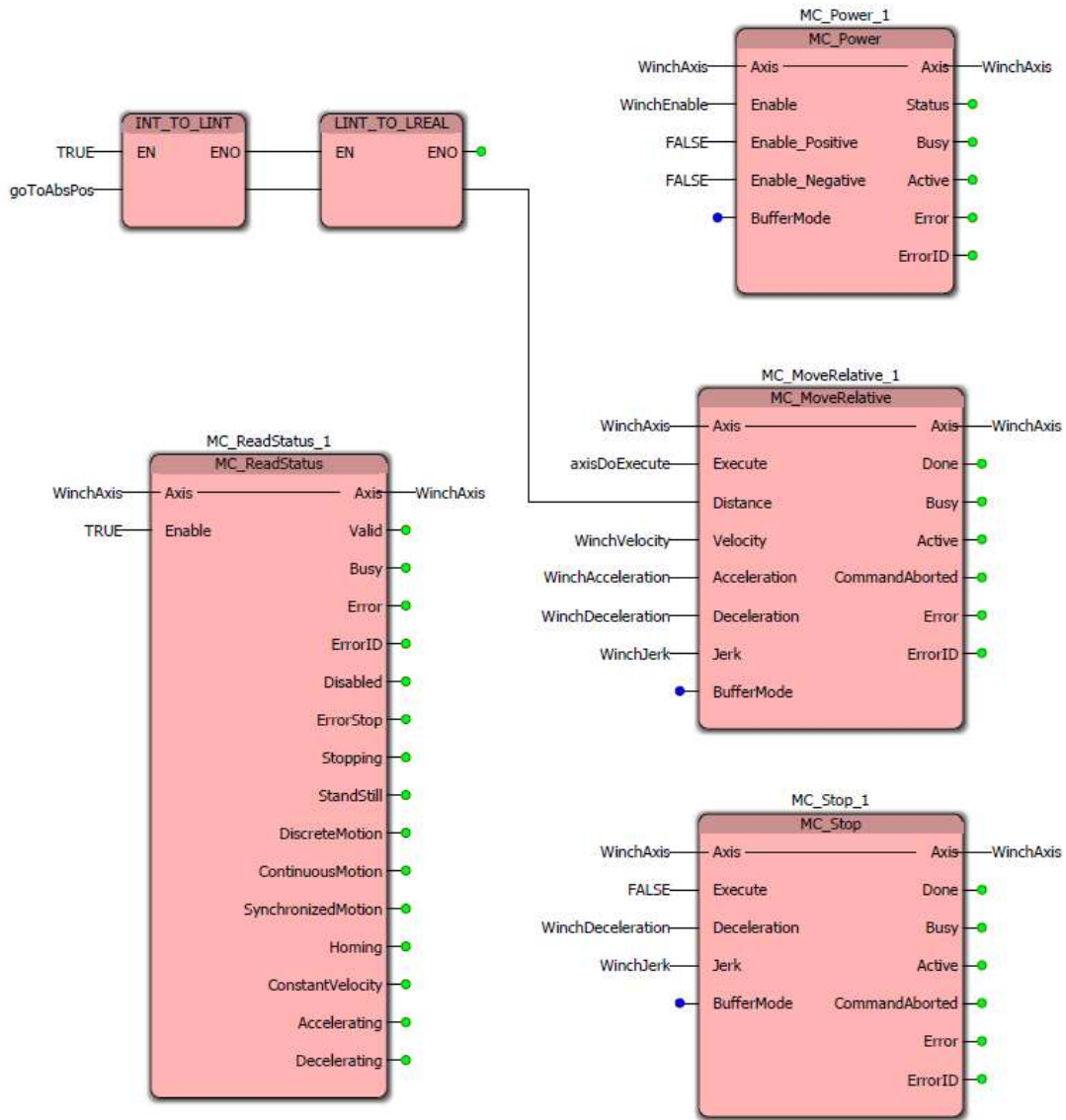
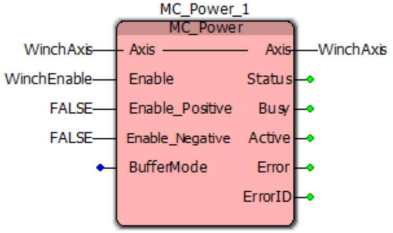
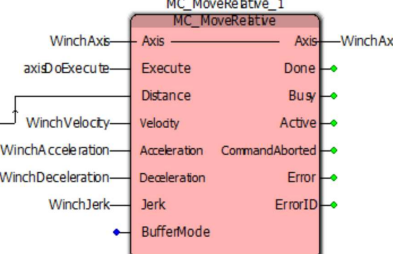
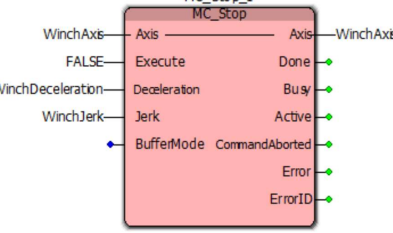
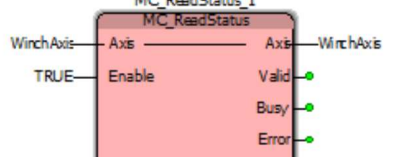


Figure 32. Ladder Diagram of servo jog program with Modbus inputs.

The individual blocks are described in detail in Table 2. Detailed variable descriptions follow.

Table 2. Ladder Diagram Block Description.

Block	Description	Relevant/Required Parameters
 <p>The diagram shows the MC_Power_1 block with inputs: WinchAxis (Axis), WinchEnable (Enable), FALSE (Enable_Positive), FALSE (Enable_Negative), and BufferMode. Outputs include WinchAxis (Axis), Status, Busy, Active, Error, and ErrorID.</p>	<p>Enables/Energizes the servo connected to the input Axis. In this case, the servo is connected to Axis 1.</p>	<p>--- UINT --- Axis --- BOOL --- Enable</p>
 <p>The diagram shows the MC_MoveRelative_1 block with inputs: WinchAxis (Axis), axisDoExecute (Execute), WinchVelocity (Velocity), WinchAcceleration (Acceleration), WinchDeceleration (Deceleration), WinchJerk (Jerk), and BufferMode. Outputs include WinchAxis (Axis), Done, Busy, Active, CommandAborted, Error, and ErrorID.</p>	<p>Moves the servo relative to its current position by the amount specified in Distance. Desired velocity, acceleration, and jerk must be supplied.</p>	<p>--- UINT --- Axis --- BOOL --- Execute --- LREAL --- Distance, Velocity, Acceleration, Deceleration, Jerk</p>
 <p>The diagram shows the MC_Stop_1 block with inputs: WinchAxis (Axis), WinchDeceleration (Deceleration), WinchJerk (Jerk), and BufferMode. Outputs include WinchAxis (Axis), Done, Busy, Active, CommandAborted, Error, and ErrorID.</p>	<p>Stops/Brakes the servo, by a desired deceleration and jerk if the servo is moving.</p>	<p>--- UINT --- Axis --- BOOL --- Execute --- LREAL --- Deceleration, Jerk</p>
 <p>The diagram shows the MC_ReadStatus_1 block with inputs: WinchAxis (Axis) and TRUE (Enable). Outputs include WinchAxis (Axis), Valid, Busy, and Error.</p> <p>*Additional output variables not shown</p>	<p>Extracts status data from motion controller. Used to determine servo state.</p>	<p>--- UINT --- Axis --- BOOL --- Enable</p>

Note that the ladder diagram displayed in Figure 32 includes two blocks in series to convert an integer input value from Modbus into the LREAL datatype. This ladder diagram is shown below in Figure 33.

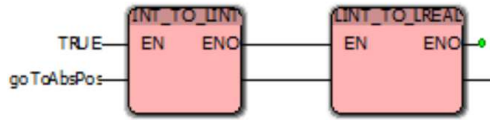


Figure 33. Conversion of integer to LREAL datatype.

The following local variables (under Main V:Main), shown in Figure 34, are used in the main task; additional details of some of the variables is found in Table 3. The WinchAxis variable is set to be retained as 1 even after a power-cycle, because the WinchAxis will never change.

Name	Type	Usage	Address /	Init	Retain
Default					
MC_MoveRelative_1	MC_MoveRelative	VAR			<input type="checkbox"/>
WinchAxis	UINT	VAR		1	<input checked="" type="checkbox"/>
WinchEnable	BOOL	VAR_EXTER...			<input type="checkbox"/>
WinchVelocity	LREAL	VAR		10.0	<input type="checkbox"/>
WinchAcceleration	LREAL	VAR		10.0	<input type="checkbox"/>
WinchDeceleration	LREAL	VAR		10.0	<input type="checkbox"/>
WinchJerk	LREAL	VAR		10.0	<input type="checkbox"/>
ENO_1	BOOL	VAR			<input type="checkbox"/>
MC_Power_1	MC_Power	VAR			<input type="checkbox"/>
MC_ReadStatus_1	MC_ReadStatus	VAR			<input type="checkbox"/>
MC_Stop_1	MC_Stop	VAR			<input type="checkbox"/>
WinchDoMove	BOOL	VAR			<input type="checkbox"/>
doWinchMove	BOOL	VAR			<input type="checkbox"/>
goToAbsPos	INT	VAR_EXTER...			<input type="checkbox"/>
axisDoExecute	BOOL	VAR_EXTER...			<input type="checkbox"/>
MC_TorqueControl_1	MC_TorqueControl	VAR			<input type="checkbox"/>

Figure 34. Variables local to Main V:Main.

Table 3. Description of variables used in in Main V:Main.

Variable	Type	Description
WinchAxis	UINT	Used if multiple motion controllers are being controlled by a single PLC. In this use case, we only have one servo connected to Axis 1.
WinchEnable	BOOL	Used to Enable the MC_Power block so that the servo is energized. The servo must be energized to perform MC_Move* operations.
WinchVelocity	LREAL	Maximum servo velocity.
WinchAcceleration	LREAL	Maximum servo acceleration.
WinchJerk	LREAL	Maximum servo jerk.
axisDoExecute	BOOL	The MC_MoveRelative and MC_MoveAbsolute blocks only execute if their Execute parameter is TRUE.
goToAbsPos	LREAL	Desired distance for servo to rotate. Note: In the ladder diagram, the MC_MoveRelative block is used, so the value assigned to this variable is the relative displacement that the servo will move from its previous position. The MC_MoveAbsolute block provides the expected absolute position behavior.

3.5 Pod System

The Pier Portal Pod, originally manufactured by Victor Espinosa (Espinosa, Pier Portal: A Marine Monitoring System, 2017), contains three cameras and additional electronics used to capture and transmit underwater video. The pod's electronics are seated inside a water-proof clear acrylic tube and milled aluminum sealing top and bottom parts. Electronics are interfaced through a twelve-conductor connector. The condition of the pod upon receipt is shown in Figure 35.



Figure 35. Pier Portal Pod, as received.

3.5.1 Pod Internals

The electronics within the pod are controlled by three Raspberry Pi computers. Each raspberry pi drives an individual camera and spotlight circuit. The interface between the computer and the spotlight circuit is a custom header consisting of SPI and GPIO interfaces. Each camera is connected to its respective computer through the Raspberry Pi's CSI-2 connector. Each computer is accessible over ethernet through the onboard 4-port switch. A custom power delivery board powers each of the computers, switch, floodlight boards, and cameras. A schematic of the pod's internals is shown below in Figure 36.

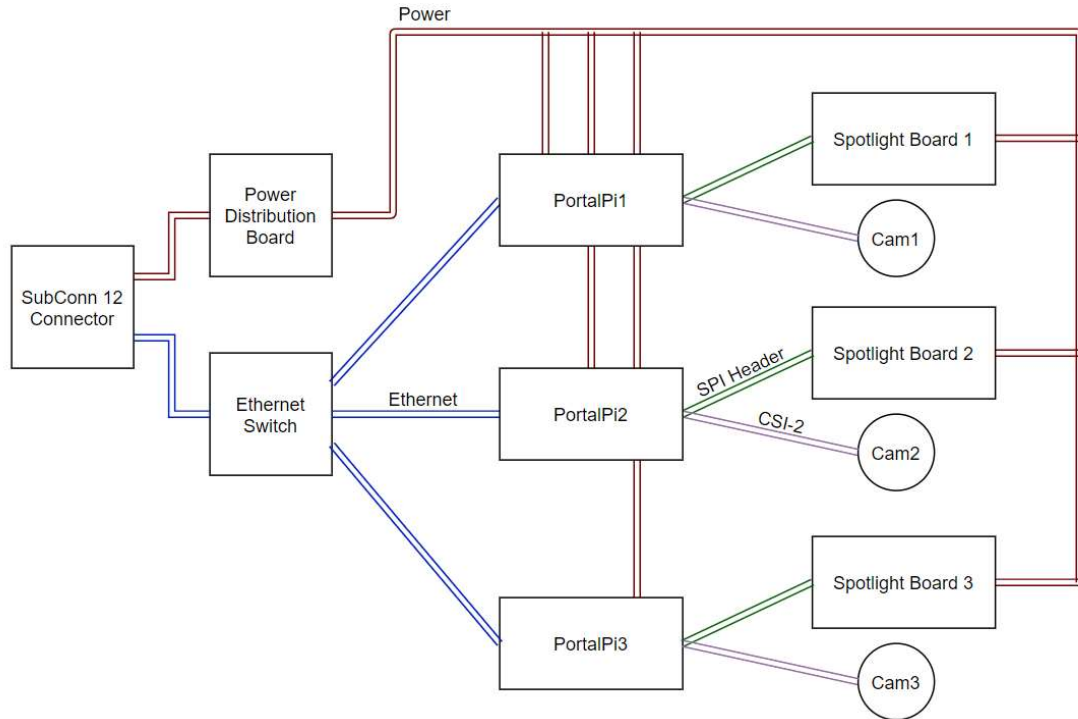


Figure 36. Pier Portal Pod High Level Electronic Schematic.

Each of the computers have been configured with different hostnames so that they can be accessed remotely over ethernet through the onboard switch. Each computer has multicast DNS (mDNS) enabled, so they can be interfaced without knowing the device's IP or MAC address. The host file from one of the pod computers is shown in Figure 37.

```

pi@PortalPi1:~/PierPortal $ cat /etc/hosts
127.0.0.1    localhost
::1        localhost ip6-localhost ip6-loopback
ff02::1    ip6-allnodes
ff02::2    ip6-allrouters

127.0.1.1   PortalPi1

```

Figure 37. Hosts file of PortalPi1 camera pod computer.

Each of the computers have also been assigned the usernames, passwords, and hostnames as follows in Table 4.

Table 4. Pod Computer Configurations.

	Computer 1	Computer 2	Computer 3
Username	pi	pi	pi
Password	portalpi1	portalpi2	portalpi3
Hostname	PortalPi1	PortalPi2	PortalPi3

It is important that these passwords are changed before putting the system online in production. These passwords were chosen for simplicity and ease of testing and development. This, and other security concerns, are addressed in section 5.5.

Each computer also has the following interfaces enabled:

- Camera (CSI-2)
- SSH
- SPI
- GPIO
- Serial

3.5.2 Electronic Interfacing

In order to interface with the electronics within the pod without needing to disassemble and break the top header connections, the first step was to determine the pinout of the SubConn circular connector. The connector has a pin numbering arrangement as shown below in Figure 38.



Figure 38. Pin numbering scheme for the SubConn 12-contact underwater connector.

The following wiring information was provided by Victor Espinosa (Espinosa, Pier Portal: A Marine Monitoring System, 2017), shown below in Table 5.

Table 5. Pre-existing pod wiring arrangement.

New Subconn	Old Subconn	Supply Header	Ethernet
Pod	Pod	Pod	Pod
3 Br	3 R/Bk	5 Br	1 O/W
7 Y	7 W/Bk	4 Y	2 O
5 O	5 O	3 O	3 Gn/W
6 Bl	6 Bl	2 Bl	4 Bl
2 W	2 W	6 W	5 Bl/W
8 Gy	8 R	7 Gy	6 Gn
1 P	1 Bk	8 P	7 Br/W
4 Gn	4 Gn	1 Gn	8 Br
9 R-18	9 Gn/Bk		
10 R-18	10 O/Bk		
11 Bk-18	11 Bl/Bk		
12 Bk-18	12 Bk/W		

Using the continuity testing function of a multimeter, it was determined that the pinout provided above was incorrect when connecting the pins of the SubConn connector directly to the NIC of any programming computer (PC). By plugging in a standard, straight-thru (aka. patch cable) ethernet cable into the output of the slip-ring of the winch drum, the following connections were determined, as shown in Table 6.

Table 6. Correct pinout of SubConn Connector

Cable/Connector	Pod	
Location	Direct Access	
Pin Connections <i>(num:color)</i>	1	Eth Brown/White
	2	Eth Blue/White
	3	Eth Green
	4	Eth Brown
	5	Eth Green/White
	6	Eth Blue
	7	Eth Brown/White
	8	Eth Brown
	9	+24V
	10	+24V
	11	GND
	12	GND

These connections were verified by splicing an ethernet cable, connecting the respective conductors, and attempting to interface the computers connected to the switch through an ethernet NIC. The connections were verified to be accurate after each computer inside the pod was able to be pinged using their hostnames (through mDNS). The wiring is shown below in Figure 39.



Figure 39. Ethernet Interface to Switch inside Pod.

After successfully pinging each of the computers inside the pod through the ethernet connection, login authentication was tested for each computer by attempting to connect to each computer over secure shell (SSH). Connection to each computer was successful.

3.6 Pier Portal Server

The Pier Portal main server is the user-facing master computer which provides the web interface that will allow users to view the live-streamed video from the cameras within the pod, as well as the system that will relay winch commands from the web-interface to the winch control PLC system. This server can be any computer connected directly to the internet, or to a router with an active default gateway. Because this computer will be handling all the web requests, it is recommended that the uplink speed of the default gateway that this computer is connected to is as high as possible. For cost reasons and for

testing purposes, this server will be run on a fourth Raspberry Pi computer, running a headless version of Raspian.

3.6.1 Web Server

The Pier Portal server has been configured such that its hostname is “PPMaster” (short for Pier Portal Master), default username of “pi”, and the password has been left as the default “raspberry”. Again, note that it is very important that this password is changed before the system is placed online live for production. The same interfaces that have been enabled in each of the PortalPi computers have been enabled, and apache2 and libapache2-mod-wsgi-py3 have both been installed to act as the system’s production webserver. Using a WSL interface, the installation procedure can be tested by entering the following commands, and the web server can be verified working by visiting the localhost address on the WSL PC. The commands used to install and run the production webserver are shown below:

```
sudo apt-get update
```

```
sudo apt-get install apache2 libapache2-mod-wsgi-py3
```

```
sudo service apache2 start
```

By visiting localhost, or the loopback IP 127.0.0.1, the following webpage, shown in Figure 40, verifies that the webserver is working on the standard HTTP port 80.

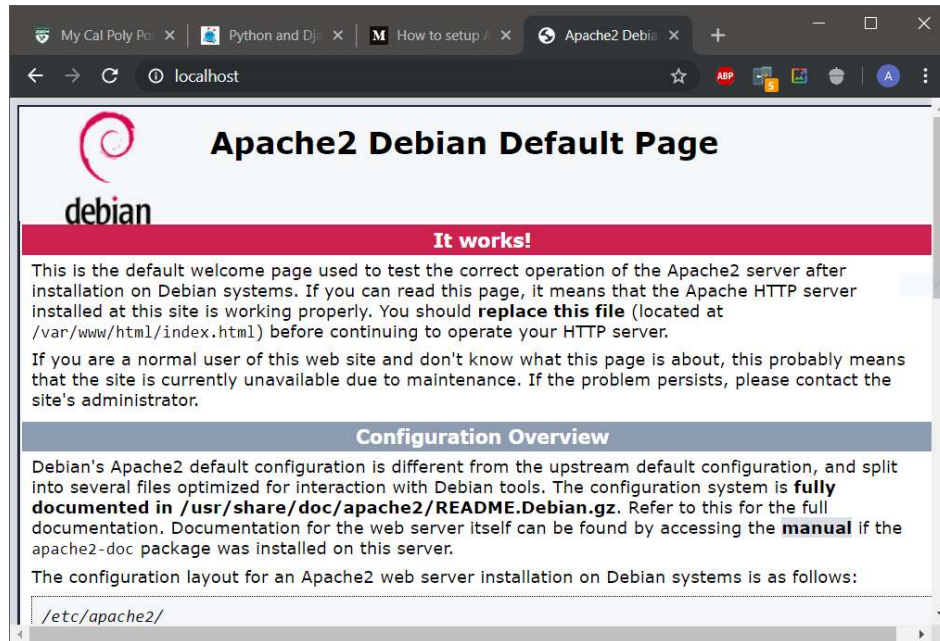


Figure 40. Apache webserver working.

3.6.2 Remote Access

In order to access the test webserver from a remote, public location, certain configurations needed to be made to the router that the server is connected to. During the final installation of Pier Portal, this will be the pier's router that the winch control panel's switch is plugged into. For testing purposes, the server is behind a typical home wireless router. Remote access is performed using the network's public IP address, or a domain name resolving to the same network public IP address, from a remote internet connection.

The domain name "pierportaltest.ddns.net" was registered with a dynamic DNS provider because the test network is assigned a dynamic IP address from its ISP. The DNS Type-A record (for IPv4) looks as follows in Figure 41. A quick nslookup confirms that the domain name was successfully registered and recognized, as per Figure 42.

Hostname ▲	Last Update	IP / Target	Type
pierportaltest.ddns.net Expires in 20 days	May 13, 2020 15:24 PDT	47.6.179.215	A

Figure 41. Dynamic DNS Record for pierportaltest.ddns.net.

```
C:\Users\Alec>nslookup pierportaltest.ddns.net
Server: rns01.charter.com
Address: 2607:f428:ffff:ffff::1

Non-authoritative answer:
Name: pierportaltest.ddns.net
Address: 47.6.179.215
```

Figure 42. Domain name resolution for pierportaltest.ddns.net.

In order to access the web server publicly, the router must be configured to route all web requests to the Pier Portal server. Because the layer 2 network that the pier portal test server is part of has other devices, all using the same outward-facing public IP address, the IP address that the domain name “pierportaltest.ddns.net” resolves is the IP address of the router’s default gateway. Thus, when a web request packet reaches the router, and because the packet’s destination is the default gateway (the router), the router must have additional instruction on which layer 2 device to send the web request packet. The router can be configured to port-forward any received packets from a TCP/UDP/HTTP port to an internal device (by the internal device’s local IP address). An example of this configuration is shown in Figure 43.

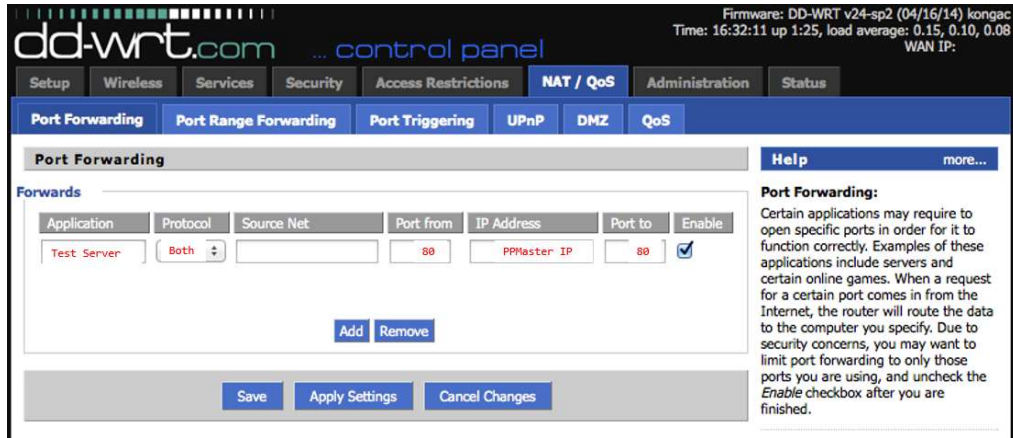


Figure 43. Port forwarding to allow public access to web server.

With port forwarding correctly configured, visiting “pierportaltest.ddns.net” from a non-local device successfully displays the Apache webserver’s test page. This same port-forwarding procedure will be used for each of the three internal pod cameras as a method to address each individual Raspberry Pi.

4. DESIGN DETAILS

4.1 Winch and Sheave Assembly

The winch and sheave design proposed above in Section 3.1 was manufactured with the help of Rob Brewster, CSM Equipment Technician at Cal Poly. Initially, the sheave holder without the winch attachment was manufactured and relocated to test mounting and installation placement at the Pier. The completed sheave holder assembly is shown below in Figure 44.



Figure 44. Completed Sheave Holder Assembly.

Notice the vertical tabs on the horizontal bottom beams, and the upwards-facing L-bracket endcaps. These tabs will allow for the easy installation and removal of a lightweight cover for the pod and cart while it is not underwater. A fresh water rinsing station can be attached to this cover. A close-up rendering is included in Figure 45.

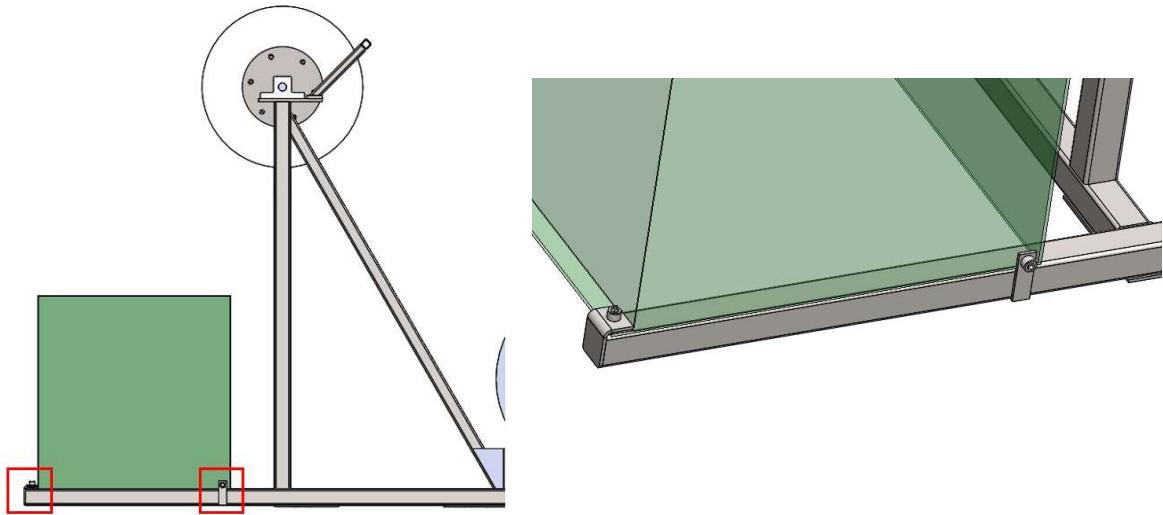


Figure 45. Pod and cart cover can be easily attached and removed from the pier.

On the pier, the sheave holder and the state of the track and cart were preliminarily tested by tying a long section of rope to the cart, placing the cart on the track, and using the sheave holder as a pulley to lower and raise the cart to and from water level. This test verified that the sheave holder assembly could successfully guide the winch cable attached to the cart down the length of the track, and that the cart and track assemblies operated as expected. Minor clearance issues at one of the track joints may need to be evaluated when the track is cleaned and the system is finally installed, as the cart rubbed on one of the track joining plates just above the water's surface. This interference is shown circled in red in Figure 46. The cart was able to be freed during this test by simply lowering it and re-raising it; there are some minor scratches on the black C-shaped part of the cart that faces towards the track, likely from one of the attachment bolt heads. This interference was only noticed during one cycle of manual cart lowering and raising.

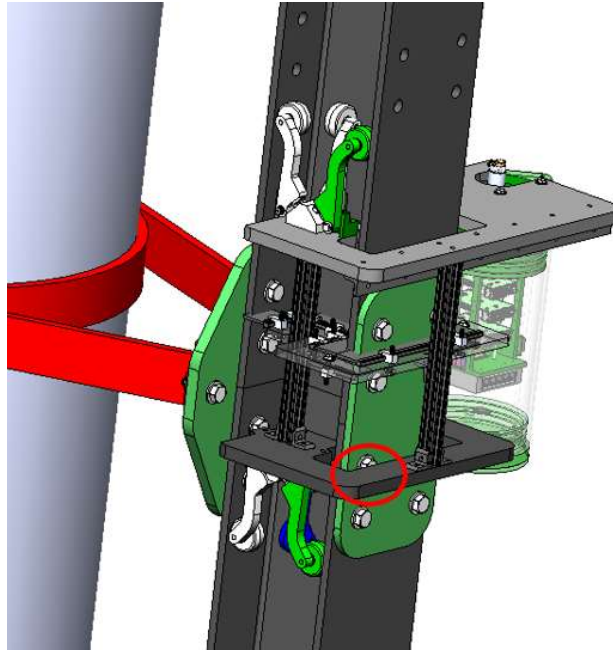


Figure 46. Cart rubbing on track joint plate at position shown in red.

Mounting the assembly to the pier is achieved using the gussets welded to the bottom of the frame. J-bolts are attached underneath the floor of the pier with the threaded end coming upwards out from underneath the bottom of the pier decking. The threaded end goes through the hole in the gussets and is bolted down, as shown in Figure 47. Scrap pieces of Delrin plastic are used directly underneath the gussets like large washers to keep the gussets separated from the metal grating. Without the plastic spacers, the interface between the metal gussets and the pier decking would rust or corrode because of the salty, electrolytic water from the ocean.



Figure 47. Assembly is bolted down to pier deck using J-bolts.

The manufacture of the rest of the winch holder assembly was cut short due to campus closure in early 2020, however, all mechanical pieces are cut to size, cleaned up, and ready for welding onto the sheave holder assembly. A new sprocket was manufactured to fit onto the output shaft of the 1:20 planetary gearbox, and is shown in Appendix E: New Winch Sprocket. Additional views of the winch holder assembly, to be manufactured, are shown in Figure 48.

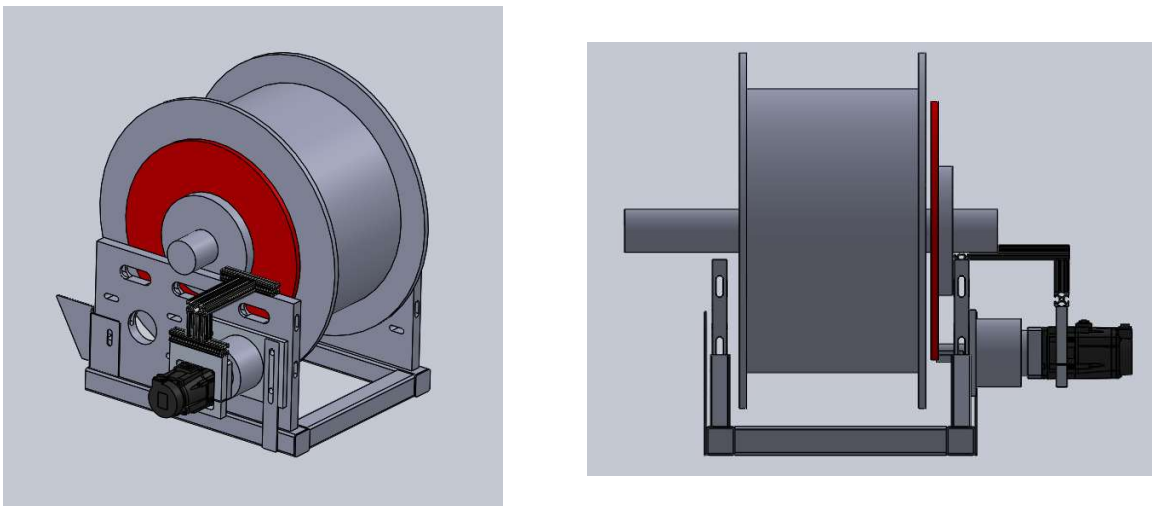


Figure 48. CAD Models of Winch Holder Assembly.

A finalized view of the shave and winch assembly on the pier, showing the plastic pod cover “doghouse” as transparent, is shown in Figure 49.

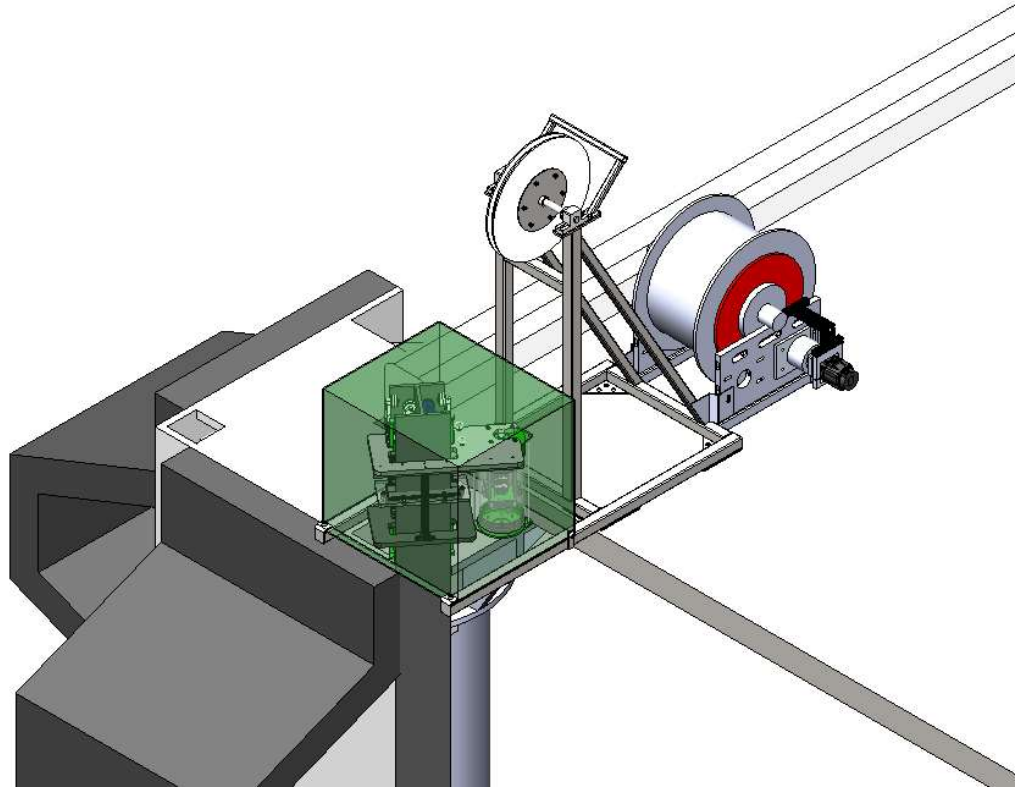


Figure 49. Pier Portal winch, sheave holder, cart, pod, and track.

4.2 Pier Portal Server

The Pier Portal main server, as discussed in section 3.6, runs on an independent Raspberry Pi computer connected to the same network as the rest of the Pier Portal electronics. The main server application is written in Python 3 using Django version 2.2. The application provides the web interface for public users, configuration and administrative setting access for development and debugging purposes, and as the software link between the pod and winch controller PLC. The application requires Python version 3.7 or greater, python3-django version 2.2 or greater, and the python3-django-channels plugin. The Django project is referred to as “PPWebServer”.

4.2.1 Dependency Installation

The most up-to-date versions of Python, Django, and Django channels can be installed using the following commands:

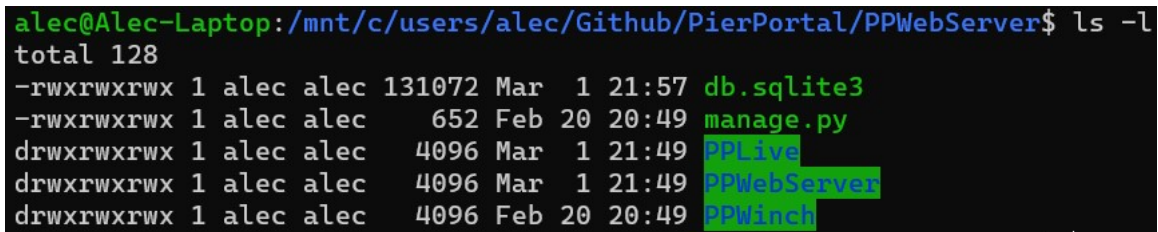
```
sudo apt-get install python3 python3-pip
```

```
sudo pip3 install -U django
```

```
sudo pip3 install -U channels
```

4.2.2 Server Start

The server is started by first setting the current working directory to the directory containing the project files, “PPWebServer”. This top-level directory should contain the following files and folders, shown in Figure 50.

A terminal window showing the command 'ls -l' in a directory. The output lists several files and folders with their permissions, owner, group, size, date, and time. The files listed are db.sqlite3, manage.py, PPLive, PPWebServer, and PPWinch. The PPWebServer and PPWinch folders are highlighted in green in the original image.

```
alec@Alec-Laptop:/mnt/c/users/alec/Github/PierPortal/PPWebServer$ ls -l
total 128
-rwxrwxrwx 1 alec alec 131072 Mar  1 21:57 db.sqlite3
-rwxrwxrwx 1 alec alec   652 Feb 20 20:49 manage.py
drwxrwxrwx 1 alec alec  4096 Mar  1 21:49 PPLive
drwxrwxrwx 1 alec alec  4096 Mar  1 21:49 PPWebServer
drwxrwxrwx 1 alec alec  4096 Feb 20 20:49 PPWinch
```

Figure 50. Pier Portal Server Application TLD.

The server is then started using the following command:

```
python3 manage.py runserver
```

It is very important that when the server is started, that the startup message reads as follows in Figure 51. If the line specifying “ASGI/Channels...server at [hostname]” is not seen, then the web-socket server is not started, and winch control will not be possible. If the Channels server is not running, then the startup message will only read “Django version 3.0.6, using settings 'PPWebServer.settings'.”

```
alec@Alec-Laptop:/mnt/c/users/alec/Github/PierPortal/PPWebServer$ python3 manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
May 19, 2020 - 22:32:49
Django version 3.0.6, using settings 'PPWebServer.settings'
Starting ASGI/Channels version 2.4.0 development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Figure 51. Pier Portal Server bootup message showing ASGI server running.

4.2.3 Auto-start on boot

In order to guarantee that the Pier Portal server auto-starts on system bootup, or upon a server reboot, the server application can be registered as a service. This is accomplished by using a system file located at `/lib/systemd/system/pierserver.service`. This file must have a permission number of 0644 or greater. The contents of the file to launch the server is shown below:

```
[Unit]
Description=Pier Portal Server
After=multi-user.target
[Service]
Type=idle
ExecStart=python3 /home/pi/PPWebServer/manage.py runserver
[Install]
WantedBy=multi-user.target
```

Running the following commands will enable the service to launch at boot-time:

```
sudo systemctl daemon-reload

sudo systemctl enable pierserver.service
```

4.2.4 Web Interface

The user web interface is shown in Figure 53. This is provided by the server application by an HTTP request from `http://{hostname}/live`. Using the test domain name discussed in section 3.6.2, `pierportaltest.ddns.net`, the web interface would be accessible at `http://pierportaltest.ddns.net/live`. Note that when the system is live, the “Camera Currently Unavailable” image is replaced with the live stream from the currently selected camera. The user interface to control the pod is shown in Figure 52.

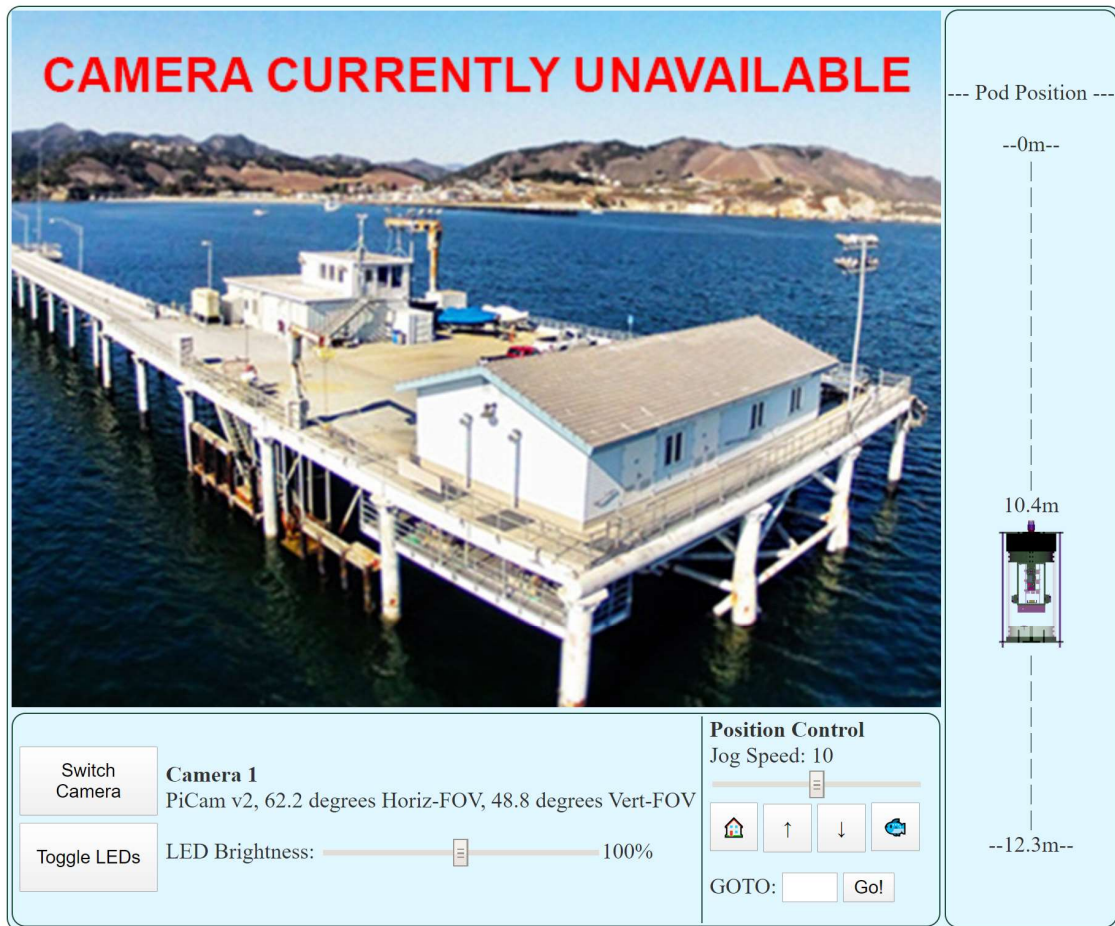



Figure 52. Online interface pod control panel.




CAL POLY

Pier Portal

Real-Time Underwater Camera

The "Pier Portal" system provides public real-time footage from the pier at Cal Poly's Center for Coastal Marine Sciences. The Underwater pod contains three cameras; two 8MP standard webcams and one 8MP No-IR webcam. The system is user-controlled and available 24/7. If the system is currently in use or reserved, use the link on the right sidebar to reserve a time.



Position Control

Jog Speed: 10

Position: 10

Home | Left Arrow | Right Arrow | Go

GOTO:

Pod Position ---

---0m---

10.4m

---12.3m---

Pier Portal Live

Reserve Control Time

Stream Source

About Pier Portal

Copyright © 2020 Alec Hardy.
 This project is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. ([CC BY-NC 4.0](https://creativecommons.org/licenses/by-nc/4.0/))

PierPortal is brought to the public by Cal Poly Mechanical Engineers. Software and control system designed and implemented by Alec Hardy. Camera pod and cart designed by Victor Espinosa. Special thanks to Dr. John Ridgely and Tom Moylan. Additional mechanical systems and iterations by Andrew Belis, Andy Crafts, Jeremy DePangher, Aaron Hein, Michael Machado, Aaron Poulos, Misha Ballingit, Cory Spieler, Aaron Jen, Alex Klimaj, Patrick Noble, Rudy Valdez, Kevin Nelson, Tony Miller, Paul Maalouf, Chahan Shah.

Figure 53. Pier Portal Web Interface.

4.2.5 Django Project Overview

The Django project that acts as the Pier Portal server contains two separate internal applications, PPWebServer and PPLive. The web server application handles web HTTP requests and acts as the front-facing user interface and master server controller. The PPLive application is internal and acts as the bridge between the user interface site (through an Asynchronous Server Gateway Interface, aka. ASGI) and the winch controller PLC, in addition to providing the webpage allowing for direct control of the video pod and winch system. In the system's current state, the web server application simply routes web requests to the front-end of the PPLive application. If additional webpages need to be added to the system, they can easily be added to the application's URL routing file. A detailed description of the two applications follows.

4.2.6 PPWebServer Application Details

The PPWebServer application serves as the master controller for the entire server software set. When a user accesses the Pier Portal webpage, the webserver application responds to the request by sending the user the webpage and allowing for an additional winch-control connection to be made. When a user requests a specific URL from the Pier Portal server, it is the PPWebServer application that parses the request and delivers the correct response. Within the application directory, the file `urls.py` contains an array named "urlpatterns" which consists of all recognized URL paths and what function to execute in order to fulfill the request at that URL. For now, because the system only handles the PPLive application, the file reads as follows:

```

from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('live/', include('PPLive.urls'))
]

```

When a URL request to /live is made, the PPLive application takes over for further processing. Note this URL pattern allows for any URL with the top-level directory “live” to be handled by the PPLive application. Further details about the PPLive application is found in section 4.2.7.

The web server application also contains Pier Portal Server specific settings, in its settings.py file. This file specifies the ASGI application to allow winch control, and error and info logging setting configuration. The ASGI application is registered with the following line:

```
1 ASGI_APPLICATION = 'PPWebServer.routing.application'
```

Separate logging configurations are set for the main web server application and the PPLive application. In order to allow logging information to be displayed in the server’s console, the following code needed to be added to the PPWebServer settings file:

```

1 # Enable logging of logger.info to display in console
2 # From https://docs.djangoproject.com/en/3.0/topics/logging/
3 LOGGING = {
4     'version': 1,
5     'disable_existing_loggers': False,
6     'handlers': {
7         'console': {
8             'class': 'logging.StreamHandler',
9         },
10    },
11    'root': {
12        'handlers': ['console'],
13        'level': 'WARNING',
14    },

```



```

15     'loggers': {
16         'django': {
17             'handlers': ['console'],
18             'level': os.getenv('DJANGO_LOG_LEVEL', 'INFO'),
19             'propagate': False,
20         },
21         # Enables logging of info level logs to console
22         'PPLive': {
23             'handlers': ['console'],
24             'level': os.getenv('DJANGO_LOG_LEVEL', 'INFO'),
25             'propagate': False,
26         },
27     },
28 }

```

Logging configuration specific to PPWebServer is encapsulated by the ‘django’ namespace shown in lines 16-20, and logging performed in the PPLive namespace is explicitly handled by the ‘console’ handler for the logging level of ‘INFO’, as specified in lines 22-26.

It is important to note that, while the PPWebServer application can serve web requests, commercial-grade web server applications provide tighter security and run with more efficiency, especially in a production case. Apache or Nginx can support the Django application through WSGI, specifically through the module `mod_wsgi`. The PPWebServer application has been configured to allow WSGI integration so transition into production is seamless. Within the PPWebServer application, the file `wsgi.py` contains the following configuration:

```

import os
from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'PPWebServer.settings')
application = get_wsgi_application()

```

In order to switch to a production environment, Apache or Nginx can be installed with `mod_wsgi` pointing to the Django application.

4.2.7 PPLive Application Details - User Interface

When a web HTTP request for the PPLive application is made, the request is first processed by the application's `url.py` file:

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.LiveView.as_view(), name='index'),
6 ]
```

This configuration launches the LiveView web view, a “generic template view” as specified in `view.py`:

```
1 from django.shortcuts import render
2 from django.views import generic
3
4 class LiveView(generic.TemplateView):
5     template_name = "PPLive/liveView.html"
```

The generic template view parses the specified page template file from within the PPLive templates. The `liveView.html` file contains the HTML/CSS/Javascript code necessary to display the user interface page. The template file also contains Django-specific tags so that pre-processing of page elements can be achieved. For instance, the first few lines of the `liveView.html` template file are shown:

```
1 {% load static %}<html>
2     <head>
3         <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
4         <meta http-equiv="Pragma" content="no-cache" />
5         <meta http-equiv="Expires" content="0" />
6         <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
```

```
7      <!--THIS SHOULD ONLY SHOW UP IF ACTIVELY CONTROLLING WINCH!!!! -->
8      <script type="text/javascript" src="{% static 'PPLive/control.js' %}"></script>
9
10     <link rel="stylesheet" type="text/css" href="{% static 'PPLive/style.css' %}">
11     <script type="text/javascript">
```

The first line, “{% load static %}”, allows the application to grab and host static images, stylesheets, and Javascript code from the application’s static file directory. These files are not pre-processed by Django, PPLive, or PPWebServer, and are simply hosted as-is. The <script> tags on lines 8 and 10 call for the inclusion of a link to these static files, as specified by the {% static * } lines. The rest of the liveView.html file contains the source code for the user-interfaced webpage. The entirety of the source code is found in Appendix F: *Django pre-processed HTML source for user interface*.

4.2.8 PPLive Application Details – ASGI Application and Web Sockets

In order to allow a user to control the winch and pod system remotely, a fast and reliable data-exchange protocol must be implemented. Web requests are handled using HTTP, and data exchange could take place by performing a new HTTP request each time data needs to be sent or new data received. However, this would break the user experience as the webpage would need to be refreshed frequently. Loading a simple HTML-only webpage hosted at Cal Poly, <http://wind.calpoly.edu>, for example, takes approximately 200ms when un-cached (without hitting 100% network utilization and over 30 network hops). This latency would render the user interface unusable, or undesired at best. The winch “jog” functionality would suffer a large amount of latency, the poor experience even further compounded by the latency in video transmission after the pod starts moving.

Asynchronous Javascript and XML (AJAX) can be used to provide data exchange to and from the Pier Portal server without requiring that the webpage be refreshed, however, each AJAX request takes nearly as long for the server to process as a separate HTTP request. Accordingly, if no data exchange needs to happen, if polling is used to refresh the pod's position on the client's webpage, unnecessary load may be placed on the server if the pod is stationary. This technique is not scalable, especially since the server is hosted on a low-power Raspberry Pi computer serving the webpage from data stored on an SD card.

In order to provide near-real-time winch control without placing unnecessary CPU burden on the server, web sockets are used to perform low-latency data exchange between clients and the Pier Portal server. Using a WebSocket also prevents the need to refresh the web interface every time data needs to be reloaded or a new command is issued. The web socket is kept open and alive for the entirety of the client's use of the winch system. Data is exchanged through WebSocket PDUs and transmitted over TCP/IP. Figure 54 below diagrams the WebSocket data flow.

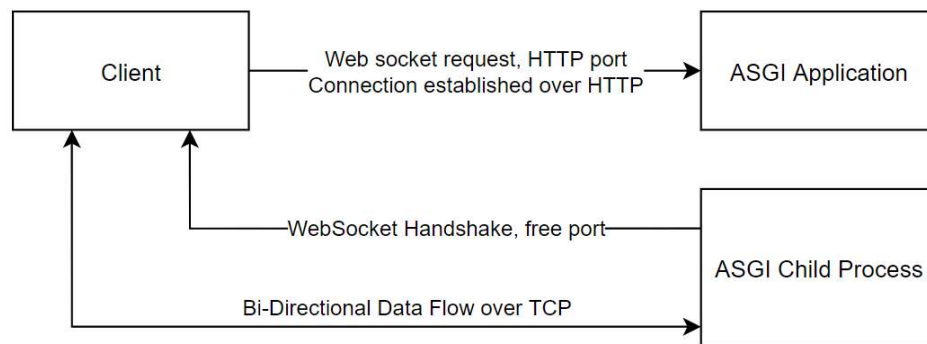


Figure 54. Diagram of WebSocket data flow.

Data sent to the client is handled by JavaScript, through the WebSocket's onmessage() callback function. On the server side, the PPLive ASGI application handles incoming WebSocket connections on the HTTP port, accepting new clients and spawning a child process or thread to handle the bidirectional data exchange between the server and client for the entire duration that the client is controlling the system. With this system, it is even possible to have multiple clients controlling the winch at once, as each client connection would spawn a child WebSocket handling process on the server.

On the client, the JavaScript code required to create a WebSocket connection is as simple as the single line:

```
var ctrlSocket = new WebSocket('ws://' + window.location.host + '/ws/ctrl/');
```

Now, messages can be sent or received on the socket (if the connection is successful) using `ctrlSocket.send("message");`. The Javascript code used to jog the winch is shown below:

```
1 var SOCKET_WRITE_DELAY = 100;
2
3 var ctrlSocket = new WebSocket('ws://' + window.location.host +
4 '/ws/ctrl/');
5
6 ctrlSocket.onmessage = function(data) {
7     var message = data['message'];
8     // Messages can be handled from the server here
9 };
10
11 ctrlSocket.onclose = function(data) {
12     console.error('Socket closed unexpectedly');
13 };
14
15
16 function getTravelSpeed() {
17     return $("#jogSpeedRange").val();
18 }
19
20 // Winch Control Buttons
21 var timeoutId = 0;
22 function handleWinchCtrlButton(event) {
```

```

23
24 var speed = String($("#jogSpeedRange").val());
25 var cmdString = "";
26 switch(event.target.id){
27     case "winchUp":
28         // Jog Up Speed *
29         cmdString = "JUS" + speed;
30         break;
31     case "winchDown":
32         // Jog Down Speed *
33         cmdString = "JDS" + speed;
34         break;
35     default:
36         // Add additional functionality here
37         break;
38 }
39 console.log(cmdString);
40 ctrlSocket.send(cmdString);
41
42 timeoutId = setTimeout(handleWinchCtrlButton, 100, event);
43 }
44 $(document).ready(function() {
45     // Handle winch ctrl buttons
46     $(".winchCtrl").on('mousedown', function(event) {
47         handleWinchCtrlButton(event);
48     }).on('mouseup mouseleave', function() {
49         clearTimeout(timeoutId);
50     });
51 });

```

On the server side, the WebSocket handler ASGI application uses the Django Channels library. As explained in section 4.2.6, the ASGI application is registered as 'PPWebServer.routing.application'. The routing.py file contains the following code, which registers any WebSocket request at ws://{pier portal hostname}/ws/ctrl to be handled by the PPLive consumer:

```

from django.urls import re_path
from . import consumers

websocket_urlpatterns = [
    re_path(r'ws/ctrl/', consumers.CtrlConsumer),
]

```

A Django channel consumer is the code that handles WebSocket connections and data exchange. The consumer file contains the CtrlConsumer class, specific to PPLive, which also imports the winch controller Python code that communicates with the Yaskawa PLC over Modbus. The CtrlConsumer class contains function handles for connect(), disconnect(), and receive(), and those functions are called anytime a WebSocket is connected, disconnected, or receives data from a client. The code is shown below:

```
1 from channels.generic.websocket import WebsocketConsumer
2 from . import winch
3
4 import re
5 import logging
6
7 logger = logging.getLogger(__name__)
8
9 class CtrlConsumer(WebsocketConsumer):
10
11     jog_regex = re.compile('J[UD]S\d')
12     home_regex = re.compile('[HB]')
13
14     # The winch class uses class methods
15     # We will never have more than one winch controller operating
16     winch = winch.Winch()
17
18     def connect(self):
19         self.accept()
20
21     def disconnect(self, close_code):
22         pass
23
24     def receive(self, text_data):
25         if self.jog_regex.match(text_data) is not None:
26             speed = int(text_data[3:])
27             if text_data[1] == 'U':
28                 # We are jogging up
29                 self.winch.jog(-speed)
30             if text_data[1] == 'D':
31                 # We are jogging down
32                 self.winch.jog(speed)
33
34         elif self.home_regex.match(text_data) is not None:
35             pass
36         else:
37             print("Weird command received: " + text_data)
38
```

Client commands to jog the winch arrive as messages formatted as “J[UD]S[1-9]”, so the command to jog the winch up at Speed 6 would be received as “JUS6”. To jog down at speed 2, the command would be “JDS2”. The above code logs the command with a logger level of ‘INFO,’ and passes the command to the Winch handling script through its instance of a Winch object.

4.2.9 Winch Modbus Controller

The winch class that performs Modbus communication with the Yaskawa winch PLC keeps track of positioning information and allows jogging and movement to specific setpoints. The Modbus connection functionality is a subclass of the Winch controlling class, abstracting away any Modbus specific communication code for easy use and extensibility. Winch methods consist of `jog()`, `go_to_position()`, `get_position()`, `saturate_position()`, etc. while the `ModbusConnection` class handles functionality such as `testConnection()`, `readRegister()`, `writeRegister()`, etc. The Winch class handles conversion of real-world position and winch servo encoder ticks. For instance, the `go_to_position_raw()` function, shown below, is used to drive the winch to a raw encoder position.

```
1     def go_to_position_raw(self, position):
2         desired_position = self.saturate_position(position)
3         logger.info("Going to raw position " + str(desired_position))
4         if not self.OFFLINE and position > 0:
5             self.ModbusConnection.goToPos(desired_position)
6         if self.OFFLINE:
7             self.position = desired_position
```

The `get_position()` function, for example, is simply a single line of code that calls upon the `get_position_raw()` function and converts the raw encoder value position to the actual real-world position.


```

1     def get_position(self):
2         return self.get_position_raw() / self.TICKS_PER_METER

```

The `get_position_raw()` function grabs the encoder position directly from the PLC without processing, as shown:

```

1     def get_position_raw(self):
2         return self.get_position_PLC() - self.pos_offset

```

The `get_position_PLC()` function uses the `ModbusConnection` subclass to read the position from the PLC's registers:

```

position = int(self.ModbusConnection.readRegister(self.ModbusConnection.POSITION_REG))

```

The different PLC registers that correspond to values of interest are stored as constants within the `ModbusConnection` class. These values are register offsets from within the Modbus FC block on the PLC. These must be configured in the IEC 61131-3 code on the Yaskawa ServoPak PLC. Information about this is covered in section 3.4.4. The entirety of the `Winch` class and `ModbusConnection` subclass is included in Appendix G: `Winch` and `ModbusConnection` classes.

4.2.10 Video Livestream

Each of the three computers inside the camera pod is running a web server which constantly streams a motion JPEG over HTTP. The motion JPEG is created from the input of the CSI-2 port. Using the python module “picamera”, the method “`start_recording()`” is capable of recording directly to a mjpeg format. This MJPG file is included in the hosted webpage and can be embedded by the pier control webpage. The source code used to create this webserver has been adapted from the code found on the `picamera` `readthedocs` webpage (Jones, 2020). The entire adapted source code can be found in Appendix H: Video Stream Source Code. An example of the web cam

livestream, in an early development state while the user interface was still being developed, is shown in Figure 55.

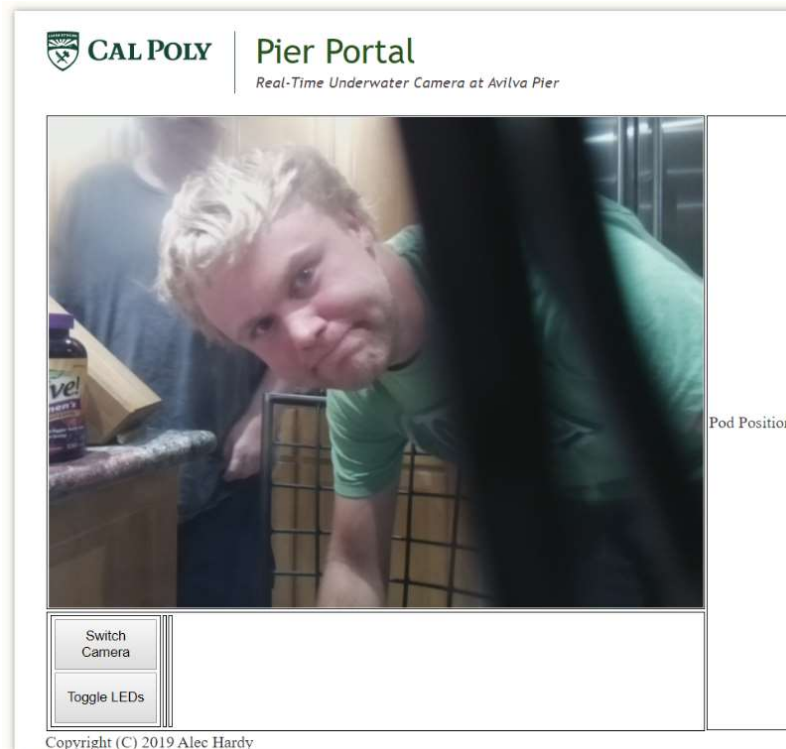


Figure 55. Live-stream webcam working during early development.

4.3 Camera Floodlight

The code required to control the floodlight LEDs was written by Victor Espinosa and is described in his thesis report. Integration into the PPLive application is as simple as including the floodlight python script in the CtrlConsumer class described in section 4.2.8. The provided code was complete and working during testing. The code is found in its entirety in Appendix I: Floodlight Control Code.

4.4 Network Configuration

In order to interface every ethernet enabled device within the Pier Portal system, the networking devices must all be configured properly to route and forward packets to their

destination computer. This must be accomplished in a secure manner to prevent unauthorized access to any computer system, such as the PLC winch controller. The entire system has a router acting as its gateway to the internet. This router will either be connected to Cal Poly's network or through a cellular hotspot not under the jurisdiction of Cal Poly's IT policies. The second case will make configuration and setup significantly easier. Within Cal Poly's network, certain simple protocols are disabled, such as ICMP between computers even on the same subnet.

The pod video computers do not have direct internet access. A look at their ARP tables does not return any results. Connecting an ethernet cable directly into a single pod Pi computer to the NIC card of a testing personal computer did not provide a working network connection. A connection could only be made by interfacing the pod computers through the pod's internal switch. Running the `ifconfig` (the Unix version of Window's `ipconfig` program) returns the following, shown in Figure 56. The IPv4 address beginning with 169.254 indicates the computer had not been assigned a static IP address and that the DHCP server (which was confirmed to be running) was unable to lease a valid IP address.

```
eth0    Link encap:Ethernet  HWaddr b8:27:eb:72:79:19
        inet addr:169.254.130.248  Bcast:169.254.255.255  Mask:255.255.0.0
        inet6 addr: fe80::d51b:26d6:b0bc:85b9/64  Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
        RX packets:1205  errors:0  dropped:0  overruns:0  frame:0
        TX packets:125  errors:0  dropped:0  overruns:0  carrier:0
        collisions:0  txqueuelen:1000
        RX bytes:97406 (95.1 KiB)  TX bytes:20036 (19.5 KiB)
```

Figure 56. Ethernet interface configuration of pod computer.

The pod Pi computer was only interfaceable by using its hostname because the computer has mDNS enabled, and communication was established over IPv6. With the pod switch

connected directly to the test PC, as shown in Figure 57, the test PC is able to use the address resolution protocol to interface the Pi's through layer 2 communication after the mDNS resolution through IPv6.

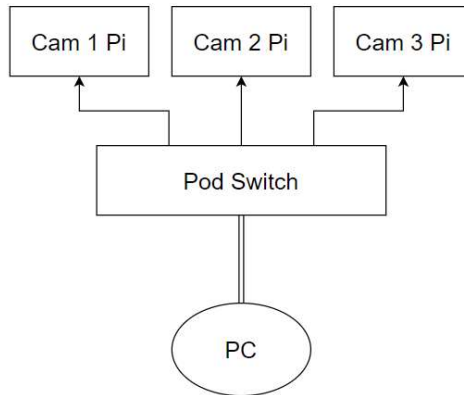


Figure 57. Pod Pi computer communication over layer 2.

Note, communicating with the pod's computers is only possible when configuring the PC's NIC to have the same subnet mask and same subnet as the pod's internal switch. The pod's switch IP address configuration and system information is shown in Figure 58. It is imperative that the PC's NIC IP address is on the same subnet but at a different host address than the switch, otherwise it is unreachable. These same configuration rules apply to the switch internal to the winch control panel.

IP Address Setting

DHCP Setting	Disable ▾
IP Address	192.168.0.1
Subnet Mask	255.255.255.0
Default Gateway	0.0.0.0

Apply Help

Figure 58. Pod switch (and winch panel) switch IP configuration. Note the switch is the first host on the 192.168.0.0/24 subnet.

The Pier Portal system will have one public-facing IP address, likely the same address as the system's default gateway. If the system is connected to Cal Poly's network, Cal Poly owns the address range 129.65.0.0-129.65.255.255 (129.65.0.0/16). All the internal computers will have local IP's assigned to them by the outward-facing router's dynamic host configuration protocol or from manually specified static internal IP addresses within the correct internal subnet. For all the system's internal computers and network devices to be able to communicate, it is very important that all devices are on the same subnet. For instance, the eth0 configuration of the pod Pi computer in Figure 56 specified that the ethernet adapter is on subnet 255.255.0.0, while the switch is on subnet 255.255.255.0. With this conflicting configuration, data will not be transmittable over IP, only over layer 1 and layer 2 communication protocols. TCP and UDP communication will not be possible. Because of ARP, communication will be possible using the device's hostnames, as was used to test the video streaming capabilities in section 4.2.10.

The Yaskawa ServoPak PLC can have its IP address and subnet configured through the Hardware Configuration window, as shown in Figure 59. As with the IP and subnet of the pod computers, the subnet must match the subnet used by all the Pier Portal systems, and the assigned IP address must be within the subnet specified. Suppose the 192.168.1.0/24 subnet is used by the outward facing router, and the default gateway and router's IP address is 192.168.1.1/24, then an acceptable IP address for PLC would be 192.168.1.11 with the 255.255.255.0 subnet. This IP address must be used in PPLive winch.py file for Modbus communication.

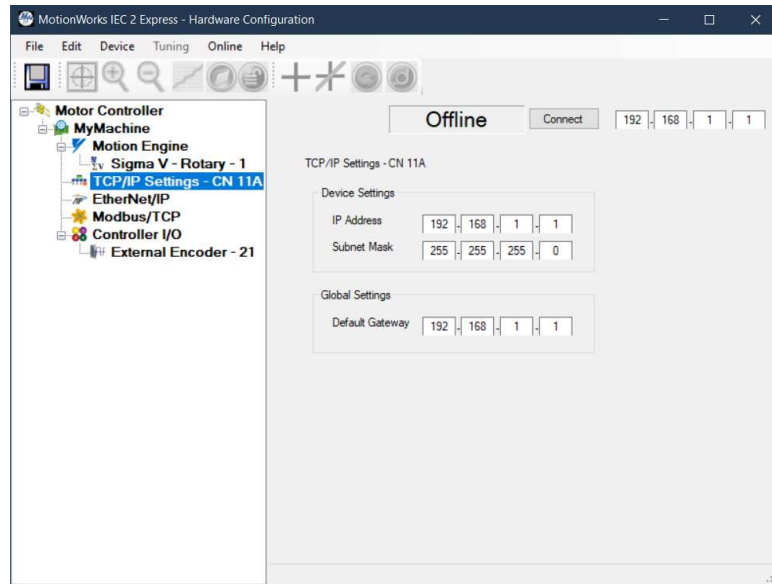


Figure 59. Interface used to change IP address, Subnet mask, and Default Gateway of Yaskawa ServoPak winch controller PLC.

A figure showing valid IP addresses that can be used to allow all devices to communicate within the same subnet is shown below in Figure 60. With this configuration, DHCP is not required by the frontward facing router because each device has its IP address manually specified. Alternatively, the hosts 2-7 could be used (on the 192.168.1.0/24 subnet) for the two switches, server, and pod Pi's. This would keep the higher hosts available for other network devices used on the pier.

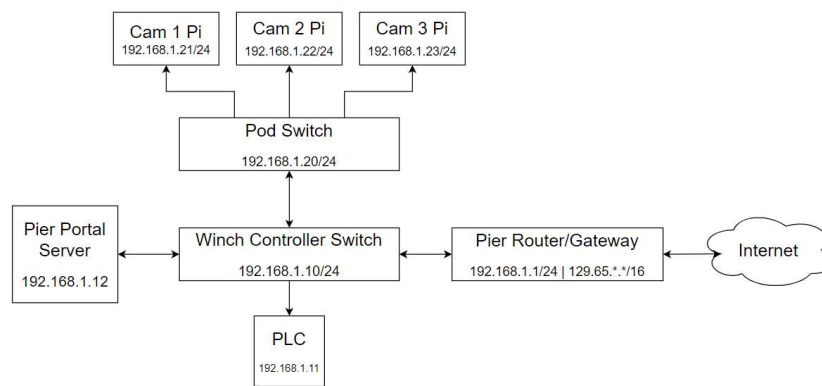


Figure 60. Internal IP addresses for each component.

Publicly interfacing the web server will be accomplished through port forwarding, as described in section 3.6.2 starting on page 46 and shown in Figure 43. This same setup must be used to publicly interface the video stream from the three pod Pi computers running webservers. Either the ports of each pod Pi webserver must be different (some port number greater than 1024 so that root access is not required for binding), or when port forwarding, the port must be re-directed such that a request to `http://{pier portal host}/video_stream.mjpg:8001` is forwarded to PortalPi1 on port 8000, a request to `http://{pier portal host}/video_stream.mjpg:8002` is forwarded to PortalPi2, etc. This is the easiest and most secure way to publicly access the webservers internal to the system. Otherwise, if the video streams were accessed like `http://{pier portal host}/stream1.mjpg`, `http://{pier portal host}/stream2.mjpg`, etc., then Django would need to perform some processing and render and host the video streams internally – the video streaming would not be able to be handled by the pod Pi hardware. This would provide additional CPU load on the main server that would increase linearly with each additional connected client viewing the stream.

5. FUTURE WORK

Due to an unexpected campus closure in early 2020, the complete installation of the new mechanical hardware and new cabling was unable to take place. As such, this thesis document shall serve as a guide for a future student or students to use to complete the system. Many of the mechanical systems have been tested and are working; the majority of the work required to put the system online, at least in a state such that all system functions can be tested, requires software integration and network configuration. Each individual software piece has been tested, and the entire layer 2 network consisting of the pod computers, server, switches, and PLC has been tested and confirmed to all work in tandem. However, the system has not yet been tested remotely through its gateway on the internet. Extensive network knowledge may be useful when configuring the network components, as the information in section 4.4 may be difficult to understand or implement without background knowledge in networking or IT.

The action items below outline work which must be completed on the Pier Portal system. Appendix J: List of TODO Items and Materials to Purchase, contains bulleted lists of the steps which must be taken, and the materials still required to be purchased to have the system operating.

5.1 Final System Installation

The winch and sheave assembly is currently on Cal Poly's campus. The sheave assembly is complete and tested, however the winch-holder extension needs to finish being welded to the base of the sheave assembly. Once these last welds are complete, the assembly needs to be transported back to the pier and set in place and secured to the decking.

Ethernet and cabling from the winch servo control panel must be run underneath the pier deck from the servo control panel to the winch and sheave assembly. The winch servo control panel needs to be mounted inside the pier network control room, as shown in Figure 61. 24V and ground must be connected from inside the servo control panel to the wires indicated from the winch slipping assembly. With the sheave and winch in place and the cabling routed, the pod can be attached to the carrier cart and the assembly can be placed onto the track secured by the winch cable. The vertical track must be cleaned beneath the water's surface and the cart must be tested along the length of the entire track to ensure that no mechanical damage had taken place.



Figure 61. Cable routing (left, (Google Maps, 2020)) and winch control panel location (right, (Espinosa, Pier Portal Thesis Image Repository, 2017)).

5.2 Winch Servo Enclosure

There are no enclosures protecting the high-voltage servo or the winch from the environment. As such, enclosures need to be installed covering both the winch drum and

the servo. It would be best to design a small, water-proof enclosure specifically for the servo, as shown in blue in Figure 62, and then cover the entire winch assembly in a larger enclosure. The larger enclosure does not need to be as waterproof as the servo enclosure, but it does need to protect the winch from the environment and weather.

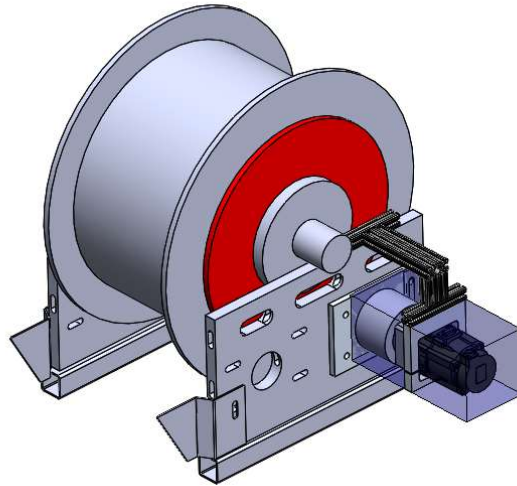


Figure 62. Enclosure for Yaskawa servo, shown in blue.

Because the pier environment has high humidity and large temperature swings, condensation inside the servo enclosure may be problematic. This can be resolved by using a small resistive heating element inside the servo enclosure. The servo will generate heat during use, so a temperature feedback system using a small temperature sensor (e.g. a thermocouple) can be implemented to limit the time the heating element needs to be used.

Because the wet winch cable may drag water from the ocean while the pod is ascending, a squeegee should be implemented to dry the winch cable before it is coiled on the winch reel.

5.3 Router Configuration

As described in section 4.4, each network device needs to have its appropriate IP address, subnet mask, and default gateway configured. If the entire system uses Cal Poly's network as its default gateway (i.e. a 129.65.0.0/16 IP address), then the system needs to be pin-holed by Cal Poly's IT department to allow external traffic to the device without the need of a VPN or tunneling. If the system uses an internet connection external to Cal Poly's network, then there should be no problem so long that the ISP's TOS allows web hosting. Also, if in the event that an email messaging function is to be added to the system, and if the POP3 protocol is to be used, the ISP must allow outgoing POP3 traffic on port 25 (SMTP should be used for this use case, and there will be no ISP port blocking issues with this protocol). Port forwarding for the Pier Portal server and each of the pod Pi computers must be set up and configured by the router. DHCP should not be used to assign IP addresses to any of the Pier Portal system components.

5.4 Apache/Nginx Web Server

Django should not be used for the production HTTP web server. The raw socket functions used to host the MJPEG live-stream videos should suffice for a production run of the system if all patches and updates are applied, but for the primary controlling server, a more robust and secure web serving software must be used. The conversion to Apache or Nginx web server is covered in section 4.2.6.

5.5 TLS and Security

Transport Layer Security (TLS) should be implemented on the four web servers in the system before being put into production. TLS will allow for secure, encrypted connections to the user interface by using the secure HTTPS rather than standard HTTP.

Furthermore, it is important to consider the security concerns of using a WebSocket ASGI application to control the winch. Because the WebSocket protocol is not restricted by the same-origin policy, without a robust authentication method, hijackers could take over control of the winch while another user is operating the system. A safety procedure when working on or servicing the winch system is to disable the ASGI application or cut power to the winch and winch control panel entirely.

It is extremely important to keep all servers up to date with the most recent versions of software installed on a regular basis. This can easily be accomplished by running “`sudo apt-get update`”, followed by “`sudo apt-get upgrade`” on every computer. Changing the account passwords on each of the individual computers on a regular basis would also prevent unauthorized access to the system.

5.6 Mechanical System Safeguards

Systems to ensure that the cart and pod do not exceed the maximum allowable tension or fall beneath the minimum expected tension must be set in place so that the system can react correctly if the pod becomes stuck or lost underwater. These safeguards have been evaluated by previous students (see Figure 63) but were unable to be tested.

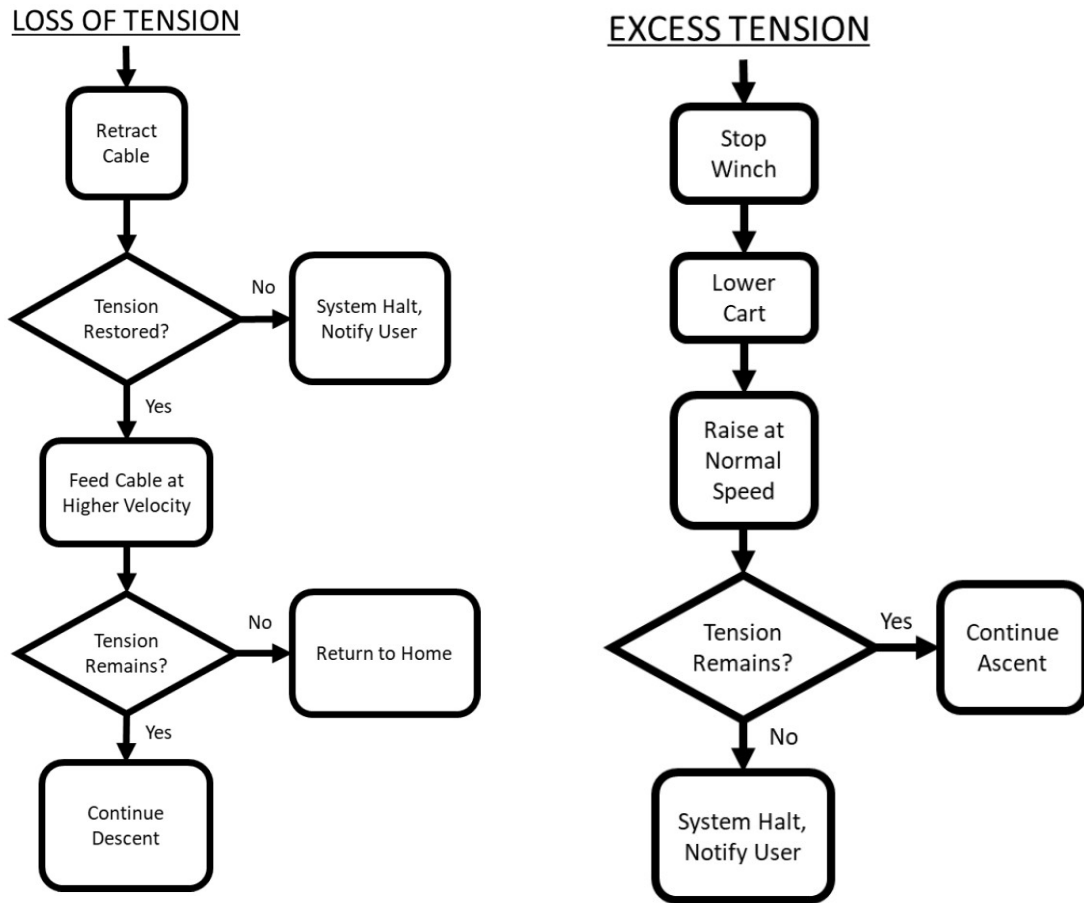


Figure 63. Mechanical safeguards to ensure proper tension (Espinosa, Pier Portal: A Marine Monitoring System, 2017).

5.7 Additional Features

The future work discussed above describes the work which must take place to make the system operable and secure. However, there are various other features that shall be added to the system once basic functionality is verified. The most important future item is a method of repelling sea life from the underwater track. This problem is discussed extensively in the reports provided by the previous work groups. Second to sea life repelling, a freshwater rinse needs to be implemented at the pier in order to keep the acrylic pod capsule clean. The addition of this should be extremely simple, hoses are in

place at the pier, a pin breakout is provided inside the winch servo control panel to allow PLC interface to a 24V valve, and the software integration required to do so has been extensively outlined in this document.

6. PROJECT FILES

The code repository for this project is accessible on Github at the following link:

<https://github.com/alecwhardy/PierPortal>

7. REFERENCES

ASM Aerospace Specification Metals Inc. (2020, June 1). *AISI Type 304 Stainless Steel*.

Retrieved from

<http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=mq304a>

Automation Direct. (2020, April 25). *Current-limiting Miniature Circuit Breaker: 3A, D*

curve. Retrieved from

[https://www.automationdirect.com/adc/shopping/catalog/circuit_protection_-z-_fuses_-z-_disconnects/ul_489_miniature_circuit_breakers/triple_pole_mini_circuit_breakers_\(0.5a-40a,_faz-na_series\)/d_curve_\(0.5a-40a,_faz-dxx-3-na\)/faz-d3-3-na](https://www.automationdirect.com/adc/shopping/catalog/circuit_protection_-z-_fuses_-z-_disconnects/ul_489_miniature_circuit_breakers/triple_pole_mini_circuit_breakers_(0.5a-40a,_faz-na_series)/d_curve_(0.5a-40a,_faz-dxx-3-na)/faz-d3-3-na)

Balingit, M., Spieler, C., & Jen, A. (2013). *Pier Portal II*. San Luis Obispo: California Polytechnic State University.

Belis, A., Crafts, A., DePangher, J., Hein, A., Machado, M., & Poulos, A. (2012). *Pier Portal*. San Luis Obispo: California Polytechnic State University.

Brown, G. (2017). *Deploying the Dive Cam*. Retrieved from Garrett Brown: Filmmaker and Inventor: <https://www.garrettcam.com/divecam>

Chen, B. (2008, August 7). *Garrett Brown Gives You a 'God's Eye' View of the Olympics*. Retrieved from Wired: <https://www.wired.com/2008/08/garrett-smith/>

Espinosa, V. E. (2017). *Pier Portal Thesis Image Repository*. San Luis Obispo.

Espinosa, V. E. (2017). *Pier Portal: A Marine Monitoring System*. San Luis Obispo: California Polytechnic State University.

Google Maps. (2020, May 23). *Google Maps*. Retrieved from <http://maps.google.com>

Jones, D. (2020). 4. *Advanced Recipes - Picamera 1.13 Documentation*. Retrieved from Picamera: <https://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming>

Klimaj, A., Noble, P., & Valdez, R. (n.d.). *Pier Portal Phase 3*. San Luis Obispo: California Polytechnic State University.

McMaster Carr. (n.d.). Machinable-Bore Corrosion Resistant Sprocket.

McQuate, S. (2018, December 13). *Underwater sensors for monitoring sea life (and where to find them)*. Retrieved from UW News:

<https://www.washington.edu/news/2018/12/13/new-underwater-sensors-for-sea-life/>

Nelson, K., Miller, T., Maalouf, P., & Shah, C. (2015). *Pier2Pier: Final Report*. San Luis Obispo: California Polytechnic State University.

sourceperl. (2019, May 20). sourceperl/pyModbusTCP: A simple Modbus/TCP library for Python.

Yaskawa. (2014, March 6). IEC 61131-3 Basics with MotionWorks IEC. (Rev 1.10).

Yaskawa. (2018, August). MP2600iec Hardware Manual. (YEA-SIA-IEC-6). Yaskawa.

8. APPENDICES

Appendix A: Safety Operation Plan

Pier Portal - Safety Operational Plan

Updated 5/5/2019

This document outlines known hazards and safety guidelines to mitigate risk from such hazards for the Pier Portal system. The following guidelines below are set in place to ensure safe development and operation of the Pier Portal mechanical and electrical systems. Operation of any Pier Portal system without strict adherence to all of these guidelines is strictly prohibited.

Mechanical Hazards:

Hazard	Risk Mitigation / Safe Operating Procedure
Heavy Components and Machinery	<ol style="list-style-type: none">1. Appropriate PPE and attire must be worn when necessary. Closed toe shoes must be worn when carrying or moving heavy or sharp mechanical components.2. Heavy components must be lifted properly and transported with the help of other persons or mechanical assistance if too heavy.
Rotating Mechanical Components	<ol style="list-style-type: none">1. Rotating components must be marked with tape or some other visual marking such that it is obvious when rotatable components are rotating.2. Where a possibility of persons or clothing being caught in machinery exists, the machinery must have a guard attached which prevents access.

Electrical Hazards:

Hazard	Risk Mitigation / Safe Operating Procedure
Electrical Shock	<ol style="list-style-type: none">1. At least two persons must be notified and present when operating high-voltage electrical equipment and systems.2. Power to high-voltage electrical enclosures must be unplugged before and while the enclosures are open.3. High-voltage enclosures must remain closed while connected to any high voltage (over 24 volt) power source.
High Voltage	<ol style="list-style-type: none">1. Any modifications to high-voltage electronics must be verified safe by a certified electrician prior to operation.
High Temperatures	<ol style="list-style-type: none">1. Some components such as heat sinks may be hot while in operation. Such components must be protected by screens or guards to prevent access by untrained persons.
Electrostatic Discharge	<ol style="list-style-type: none">1. ESD protection protocols must be followed when using ESD-prone components.

Appendix B: Deflection and Square-stock selection of Sheave Holder Assembly

ALEC HARDY
10/9/19

SQUARE STOCK
SELECTION

SQUARE STOCK SELECTION

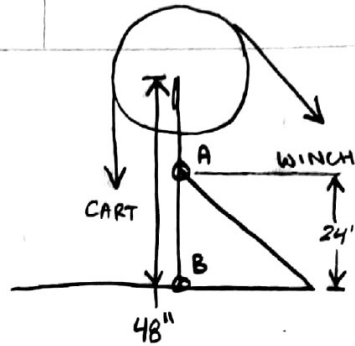
USE EULER BUCKLING

FROM QA → SHEAVE



$K=2.0$
 $L=24"$

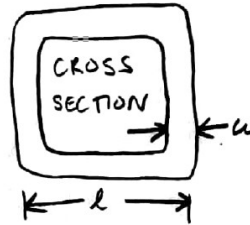
$$P_{CR} = \frac{\pi^2 EI}{(KL)^2}$$



304 STEEL: $E=28000 \text{ ksi}$

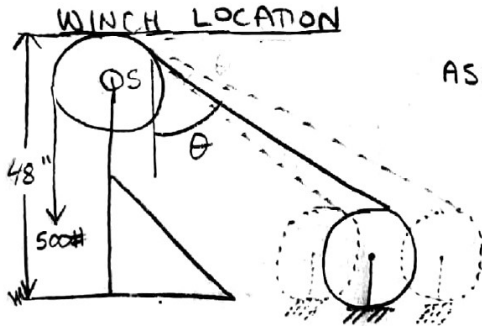
CRITICAL PRESSURES

$\frac{L}{W}$	W	$I \text{ (in}^4\text{)}$	$P_{CR} \text{ (lbf)}$
1"	.065"	.03559	4,169
1"	.12"	.05553	6,660
1.5"	.065"	.12831	15,390
1.5"	.12"	.21184	26,608
2"	.065"	.31431	37,699
2"	.12"	.53375	64,020

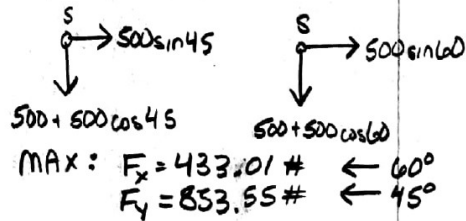


ANY OF THESE WILL WORK

→ SELECTED 1" W/ 0.12" WALL SIMPLY B/C IT IS EASIER TO WELD



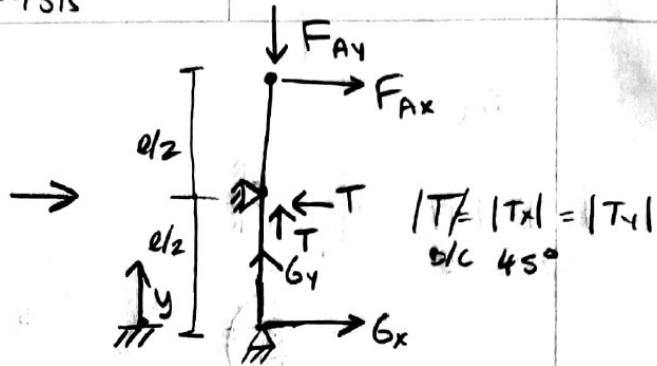
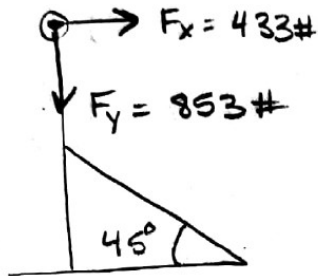
ASSUME • $45^\circ < \theta < 60^\circ$
• 500# MAX LOAD (HUGE F.S.)



→ PERFORM FEA W/ MAX F_x, F_y

ALEC HARDY
10/21/19

DEFLECTION
ANALYSIS



$$\sum F_x = 0: F_{Ax} - T + G_x = 0$$

$$\sum F_y = 0: G_y + T - F_{Ay} = 0$$

$$\sum M_G = 0: \frac{Tl}{2} - F_{Ax}l = 0$$

$$\Rightarrow T = 2F_{Ax}$$

$$\Rightarrow G_x = F_{Ax}$$

ASSUME PINS
TO MAKE
PROBLEM
STATICALLY
DETERMINANT

USE CASTIGLIANO'S METHOD TO FIND DEFLECTION

$$\delta_i = \frac{\partial U}{\partial F_i} = \int \frac{1}{EI} \left(M \frac{\partial M}{\partial F_i} \right) dy$$

$$M(y) = \begin{cases} -F_{Ax}(y) & 0 \leq y \leq l/2 \\ -F_{Ax}(l-y) & l/2 \leq y \leq l \end{cases}$$

$$\delta_{F_{Ax}} = \frac{1}{EI} \left[\int_0^{l/2} (-F_{Ax}y)(-y) dy + \int_{l/2}^l (-F_{Ax}(l-y))^2 dy \right]$$

$$= \frac{1}{EI} \left[\int_0^{l/2} F_{Ax}y^2 dy + \int_{l/2}^l F_{Ax}(l-y)^2 dy \right]$$

$$= \frac{1}{EI} \left[F_{Ax} \frac{l^3}{12} \right]$$

\Rightarrow FOR 1" WALL 0.005" THICK
 $\delta_{max} = 4.004"$
FOR 2" WALL 0.12" THICK
 $\delta_{max} = 0.267"$

Appendix C: Winch Datasheet

Large Frame Hose Reels | Series 2400

Heavy Duty Large Frame Reels Series 2400

Made in USA 

1/2", 3/4" or 1" fluid path for 1/2" through 1" I.D. hose

Rewind: Gear-drive rewind or chain and sprocket drive powered by A.C. or D.C. electric motor, compressed air motor or hydraulic motor.

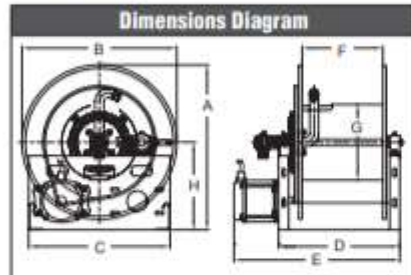
Inlet: 1/2" up to 1" female/male NPT 90° swivel joint. Straight swivel joint may be specified at extra cost.

Gooseneck: 1/2" up to 1" female/male NPT. Other sizes and/or threads, including male NST and male garden hose thread, can be furnished when specified.



1/2" pitch chain standard

Shown with optional bevel crank.



Fluid Path Material: Ductile iron with E-Coat standard; carbon steel and stainless steel available.

Seal Material: Buna-N standard; AFLAS®, Viton®, FKM, EPR, and low-temperature Buna-N available. Standard temperatures range from -20 °F (-29 °C) to 225 °F (107 °C). Standard pressures to 3000 psi.

The standard color for this series is red. See page 49 for color options.

Popular pre-configured reels listed

Don't see what you need? See page 27 to configure a reel for your needs. Available options and accessories listed on page 28.

Hose I.D.	Hose Capacity			Specifications - Hose Capacity and Reel Dimensions											Retraction Time*						
	in	mm	ft	HC = Hand Crank; EP = Explosion Proof 12 V DC Motor																	
Hose O.D.	in	mm	mm	Model without Hose		Hose Length		Weight		A	B	C	D	E	F	G	H	Spool RPM	min:sec		
1/2"	13	19	25	HC2400-19-16-10FF	ft	285	110	60	lb	70	in	22.5	21	21	22	34	16	11.25	13	N/A	N/A
				m	87	34	18	kg	32	mm	572	533	533	559	864	406	286	330	N/A	N/A	
3/4"	18	25	33	HC2400-23-14-10FF	ft	365	175	100	lb	74	in	24.5	23	21	20	32	14	11.25	13	N/A	N/A
				m	111	53	30	kg	34	mm	622	584	533	508	813	356	286	330	N/A	N/A	
1"	22	33	40	EP2405-23-18-10FF	ft	490	250	155	lb	78	in	24.5	23	21	24	30.5	18	11.25	13	13.39	3:15
				m	149	76	47	kg	35	mm	622	584	533	610	775	457	286	330	13.39	3:15	
1"	22	33	40	EP2405-23-20-10FF	ft	540	265	170	lb	80	in	24.5	23	21	26	32.5	20	11.25	13	13.39	3:30
				m	165	81	52	kg	36	mm	622	584	533	660	826	508	286	330	13.39	3:30	
1"	22	33	40	EP2405-23-24-10FF	ft	660	320	210	lb	84	in	24.5	23	21	30	36.5	24	11.25	13	13.39	4:30
				m	201	98	64	kg	38	mm	622	584	533	762	927	610	286	330	13.39	4:30	
1"	22	33	40	EP2405-25-8-10FF	ft	265	135	95	lb	72	in	26.5	25	21	14	20.5	8	11.25	14	40.18	0:45
				m	81	41	29	kg	33	mm	673	635	533	356	521	203	286	356	40.18	0:45	
1"	22	33	40	EP2405-25-14-10FF	ft	440	225	150	lb	74	in	26.5	25	21	20	26.5	14	11.25	14	12.05	3:45
				m	134	69	46	kg	34	mm	673	635	533	508	673	356	286	356	12.05	3:45	
1"	22	33	40	EP2405-31-8-10FF	ft	465	205	155	lb	74	in	32.5	31	26	14	20.5	8	11.25	17	12.05	3:00
				m	142	62	47	kg	34	mm	826	787	610	356	521	203	286	432	12.05	3:00	
1"	22	33	40	EP2405-31-10-10FF	ft	570	240	185	lb	76	in	32.5	31	24	16	22.5	10	11.25	17	12.05	3:30
				m	174	73	56	kg	34	mm	826	787	610	406	572	254	286	432	12.05	3:30	
1"	22	33	40	EP2405-31-16-10FF	ft	830	405	305	lb	80	in	32.5	31	24	22	28.5	16	11.25	17	12.05	5:45
				m	283	123	93	kg	36	mm	826	787	610	559	724	406	286	432	12.05	5:45	

Prop 65:  **WARNING:**
Cancer and Reproductive Harm
www.P65Warnings.ca.gov

Built to order, not returnable.

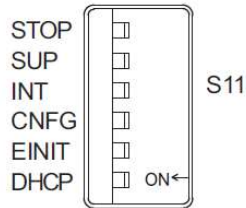
*Hose retraction times, based on largest size hose listed, are approximate and will vary depending on how hose piles on drum. Speed control options also available.

Appendix D: Yaskawa ServoPak DIP Switch Configuration

5.1 Switch Settings

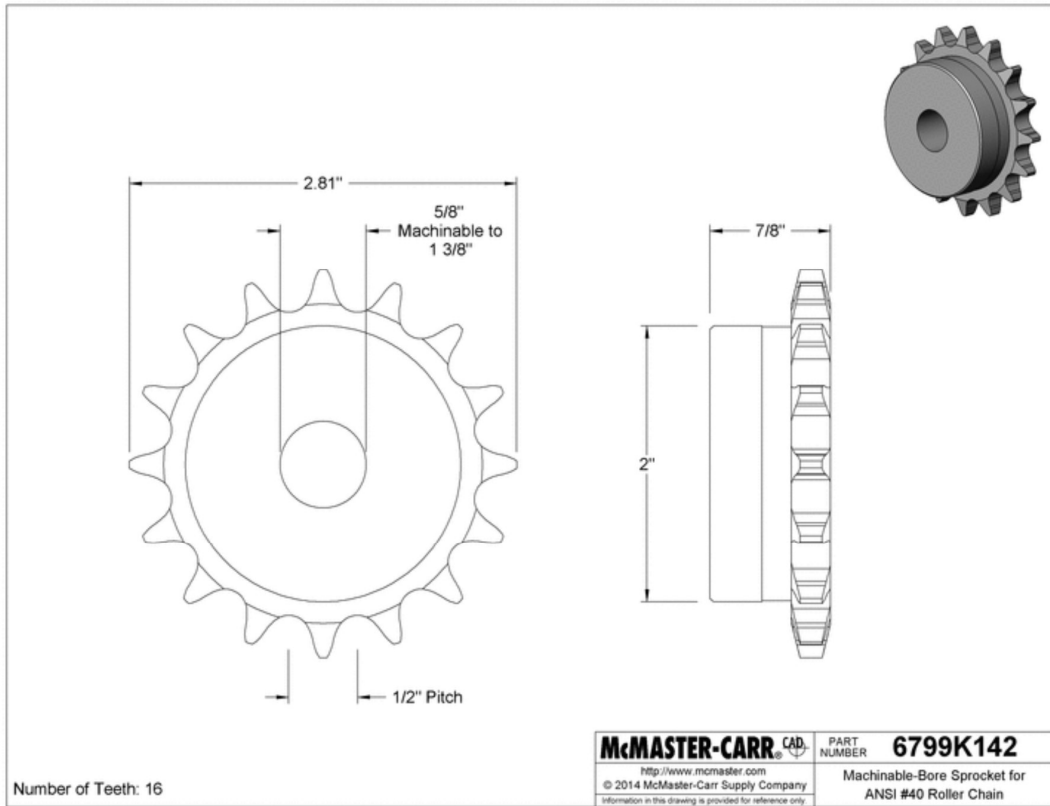
5 DIP Switches

5.1 Switch Settings

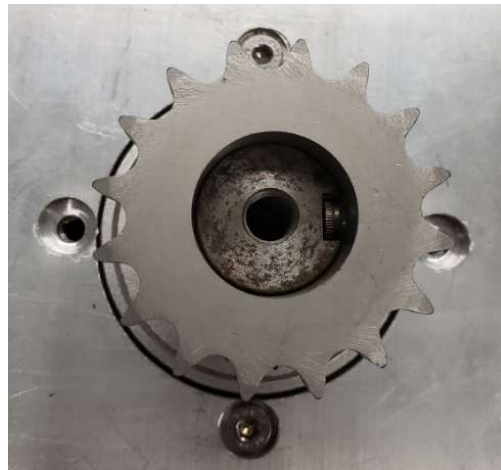


Switch	Name	Setting	Operating Mode	Setting for Normal Operation	Details
1	STOP	ON	User program execution inhibited	OFF	Inhibits user program execution
		OFF	Normal operation		
2	SUP	ON	Firmware programming mode	OFF	Enables controller firmware programming. (See Section 11)
		OFF	Normal operation		
3	INIT	ON	SRAM/clock initialization and configuration bypass	OFF	The INIT switch bypasses the loading of the saved configuration and initializes the SRAM contents and clock settings. Use after backup power has been lost, or if there are configuration issues causing alarms on the controller that may prevent connecting to the MPiEC controller. If the INIT switch is ON and the CNFG switch is OFF, the controller will show no axes connected.
		OFF	Normal operation		
4	CNFG	ON	Normal operation	ON	Always set to ON
		OFF	Do not set (reserved for future use)		
5	E-INIT	ON	Force Ethernet address setting for Port A to 192.168.1.1 and Port B to 192.168.2.1	OFF	Enables use of the default Ethernet addresses
		OFF	Normal Operation		
6	DHCP	ON	DHCP-configured IP settings	OFF	Enables use of DHCP for IP setting configuration
		OFF	Manually-configured IP settings		

Appendix E: New Winch Sprocket



Drawing from stock part, purchased from McMaster Carr (McMaster Carr).



16-tooth winch sprocket bored to fit output of planetary gearbox (left),
shown attached to gearbox (right).

Appendix F: Django pre-processed HTML source for user interface

```
{% load static %}<html>
<head>
  <meta http-equiv="Cache-Control" content="no-cache, no-store, must-revalidate" />
  <meta http-equiv="Pragma" content="no-cache" />
  <meta http-equiv="Expires" content="0" />
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
  <!-- TODO: ONLY ALLOW THIS TO SHOW UP IF ACTIVELY CONTROLLING WINCH!!!! -->
  <script type="text/javascript" src="{% static 'PPLive/control.js' %}"></script>

  <link rel="stylesheet" type="text/css" href="{% static 'PPLive/style.css' %}">
  <script type="text/javascript">

    var curCam = 1;
    var waterDepth = 13.0; //m
    var cameraDepth = 5.0; //m //TODO: Update this with AJAX every second if no websocket controller

    function doCamUnavailable(event){
      event.srcElement.src = "{% static 'PPLive/CamUnavailable.jpg' %}";
    }
    function updateDescription(camNum){
      $("#cameraNum").text("Camera " + String(camNum));
      switch(camNum){
        case 1:
        case 2:
          $("#cameraDesc").text("PiCam v2, 62.2 degrees Horiz-FOV, 48.8 degrees Vert-FOV");
          break;
        case 3:
          $("#cameraDesc").text("PiCam v2 NoIR, 62.2 degrees Horiz-FOV, 48.8 degrees Vert-FOV. No IR
Filter.");
          break;
      }
    }

    $(document).ready(function() {

      // Handle "Switch Camera button"
      $("#switchCamera").click(function() {
        // Increment current camera count, 1 - 3. Damn, this is some nice syntax.
        ++curCam > 3 ? curCam=1 : null;
        var src = "http://PortalPi" + String(curCam) + ":8000/stream.mjpg";
        // Update camera description
        updateDescription(curCam);
        // Fade image, change camera src
        $("#camImg").fadeTo(1000,0.30, function() {
          $("#camImg").attr("src",src);
        }).fadeTo(500,1);
        return false;
      });
    });

  </script>
</head>
<body>
  <div id="wrapper">
    <div id="header">
      <div id="CPLogo">
        
      </div>
      <div id="PageName">
        <h1>Pier Portal</h1>
        <h2>Real-Time Underwater Camera</h2>
      </div>
    </div>
    <div id="content">
      <p>The "Pier Portal" system provides public real-time footage from the pier at Cal Poly's Center for Coastal Marine Sciences. The Underwater pod contains three cameras: two 8MP standard webcams and one 8MP No-IR webcam. The system is user-controlled and available 24/7. If the system is currently in use or reserved, use the link on the right sidebar to reserve a time.</p>
      <div id="CamContents">
        <table id="CamCtrlContentTbl">
          <tr>
            <td>
              <!-- Camera Image Source-->
              
            </td>
            <td rowspan="2" id="podPosTbl">
              <!-- Pod Position-->
              --- Pod Position ---
              <div id="podPosition">

```


Appendix G: Winch and ModbusConnection classes

```
1 from .pyModbusTCP.client import ModbusClient
2 import logging
3
4 logger = logging.getLogger(__name__)
5
6 class Winch():
7
8     # If a connection to the PLC can't be made, messages will be displayed
9     DEBUG = 1
10    OFFLINE = 1
11
12    """Jog speed is between 1-10, so let's multiply by some factor
13        so we don't move super slow"""
14    JOG_MULTIPLIER = 100
15    TICKS_PER_METER = 1000 # TODO: Check this value
16
17    # Home position
18    POS_MIN = 0
19    # Bottom of the ocean. Need to test to find this value.
20    POS_MAX = 99999
21
22    def __init__(self):
23        # First, see if we have a working connection to the PLC
24        if not self.OFFLINE and not self.ModbusConnection.testConnection():
25            logger.error("Unable to connect to winch PLC")
26
27        """Let self.position have a default value of 0 in case we are OFFLINE"""
28        self.position = 0
29        self.pos_offset = 0
30
31        self.position = self.get_position()
32
33    def jog(self, speed):
34        logger.info("Jogging at speed " + str(speed))
35        self.go_to_position_raw(self.position + (speed*self.JOG_MULTIPLIER))
36
37    def go_to_position_raw(self, position):
38        desired_position = self.saturate_position(position)
39        logger.info("Going to raw position " + str(desired_position))
40        if not self.OFFLINE and position > 0:
41            self.ModbusConnection.goToPos(desired_position)
42        if self.OFFLINE:
43            self.position = desired_position
44
45    def saturate_position(self, position):
46        if position > self.POS_MAX:
47            return self.POS_MAX
48        elif position < self.POS_MIN:
49            return self.POS_MIN
50        else:
51            return position
52
53    """Returns the position as indicated by the PLC, if offline, returns known position"""
54    def get_position_PLC(self):
55        position = self.position
56        if not self.OFFLINE:
57            position = int(self.ModbusConnection.readRegister(self.ModbusConnection.POSITION_REG))
58        return position
59
60    def set_zero_position(self):
61        self.pos_offset = self.position
62
63    def home(self):
64        self.position = 0
65
66    def get_position_raw(self):
67        return self.get_position_PLC() - self.pos_offset
68
69    def get_position(self):
70        return self.get_position_raw() / self.TICKS_PER_METER
71
72
73
74    class ModbusConnection():
75
```

```

76     """Modbus FC offsets, see PLC code"""
77     WINCH_ENABLE = 0
78     SETPOINT_REG = 1
79     POSITION_REG = 5
80
81     winch_host = "192.168.1.1"
82     client = ModbusClient(host=winch_host, auto_open=True, auto_close=True)
83
84     @classmethod
85     def testConnection(cls):
86         # Attempts to read from 100 registers, starting at Modbus register 0
87         if Winch.OFFLINE:
88             return True
89         return cls.client.read_holding_registers(0, 100) is not None
90
91     @classmethod
92     def writeRegister(cls, reg, data):
93         if not cls.client.write_single_register(reg,data):
94             logger.error("Unable to write to Winch PLC register: " + reg)
95
96     @classmethod
97     def readRegister(cls, reg):
98         return cls.client.read_holding_registers(reg, 1)
99
100    @classmethod
101    def readRegisterRange(cls, reg, numRegs):
102        return cls.client.read_holding_registers(reg, numRegs)
103
104    @classmethod
105    def goToPos(cls, pos):
106        cls.writeRegister(cls.POSITION_REG, pos)
107        cls.writeRegister(cls.WINCH_ENABLE, 1)
108
109    @classmethod
110    def homeWinch(cls):
111        # TODO: There will probably be some extra PLC-side check here
112        cls.goToPos(0)

```

Appendix H: Video Stream Source Code

```
1 # Web streaming example
2 # Source code from the official PiCamera package
3 # http://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming
4
5 import io
6 import picamera
7 import logging
8 import socketserver
9 from threading import Condition
10 from http import server
11
12 PAGE="""\
13 <html>
14 <head>
15 <title>Raspberry Pi - Surveillance Camera</title>
16 </head>
17 <body>
18 <center><h1>Raspberry Pi - Surveillance Camera</h1></center>
19 <center></center>
20 </body>
21 </html>
22 """
23
24 class StreamingOutput(object):
25     def __init__(self):
26         self.frame = None
27         self.buffer = io.BytesIO()
28         self.condition = Condition()
29
30     def write(self, buf):
31         if buf.startswith(b'\xff\xd8'):
32             # New frame, copy the existing buffer's content and notify all
33             # clients it's available
34             self.buffer.truncate()
35             with self.condition:
36                 self.frame = self.buffer.getvalue()
37                 self.condition.notify_all()
38             self.buffer.seek(0)
39         return self.buffer.write(buf)
40
41 class StreamingHandler(server.BaseHTTPRequestHandler):
42     def do_GET(self):
43         if self.path == '/':
44             self.send_response(301)
45             self.send_header('Location', '/index.html')
46             self.end_headers()
47         elif self.path == '/index.html':
48             content = PAGE.encode('utf-8')
49             self.send_response(200)
50             self.send_header('Content-Type', 'text/html')
```

```

51         self.send_header('Content-Length', len(content))
52         self.end_headers()
53         self.wfile.write(content)
54     elif self.path == '/stream.mjpg':
55         self.send_response(200)
56         self.send_header('Age', 0)
57         self.send_header('Cache-Control', 'no-cache, private')
58         self.send_header('Pragma', 'no-cache')
59         self.send_header('Content-Type', 'multipart/x-mixed-replace;
boundary=FRAME')
60         self.end_headers()
61         try:
62             while True:
63                 with output.condition:
64                     output.condition.wait()
65                     frame = output.frame
66                 self.wfile.write(b'--FRAME\r\n')
67                 self.send_header('Content-Type', 'image/jpeg')
68                 self.send_header('Content-Length', len(frame))
69                 self.end_headers()
70                 self.wfile.write(frame)
71                 self.wfile.write(b'\r\n')
72         except Exception as e:
73             logging.warning(
74                 'Removed streaming client %s: %s',
75                 self.client_address, str(e))
76     else:
77         self.send_error(404)
78         self.end_headers()
79
80 class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):
81     allow_reuse_address = True
82     daemon_threads = True
83
84 with picamera.PiCamera(resolution='640x480', framerate=24) as camera:
85     output = StreamingOutput()
86     #Uncomment the next line to change your Pi's Camera rotation (in degrees)
87     #camera.rotation = 90
88     camera.start_recording(output, format='mjpeg')
89     try:
90         address = ('', 8001)
91         server = StreamingServer(address, StreamingHandler)
92         server.serve_forever()
93     finally:
94         camera.stop_recording()

```

Appendix I: Floodlight Control Code

The following code was written by Victor Espinosa (Espinosa, Pier Portal: A Marine Monitoring System, 2017).

```
1 #*****
2 """ \file client_RPi1.py
3
4     Brief Description:
5     This client RPI firmware is intended to be run on the primary RPI in the pod
6     network. It creates a TCP socket connection with the server PC, and sends a status
7     string in response to an LED brightness command from the server. The status depends
8     on the information gathered from the Teensy MCU climate board and the temperature
9     data from the TC77 sensor onboard the LED spotlight.
10
11     Revisions:
12             06-05-2017           VEE           Original file.
13             10-05-2017 VEE           Added setBrightness routine to invert brightness logic.
14
15     License:
16             This file is copyright 2016 by V. Espinosa and released under the Lesser GNU
17             Public License, version 2. It intended for educational use only, but its use
18             is not limited thereto. """
19 """ THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
20     AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
21     IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
22     ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
23     LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUEN-
24     TIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
25     OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
26     CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
27     OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
28     OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. """
29 #*****
30
31 import socket           # Enables TCP socket connection protocol API
32 import time
33 import serial           # Enables RX/TX serial port API
34 import sys
35 import spidev
36 import string           # Usefull since status indicators are strings
37 import RPi.GPIO as GPIO
38
39 HOST = ' ' # Enter IP or Hostname of your server
40 PORT = 59900 # Pick an open Port (1000+ recommended), must match the server port
41
42 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # TCP connection, not UDP
43 s.connect((HOST,PORT))
44
45 # Configure serial port comms with Teensy climate monitoring board
```

```

46 ser = serial.Serial(
47     port='/dev/ttyUSB0',
48     baudrate = 9600,
49     parity=serial.PARITY_NONE,
50     stopbits=serial.STOPBITS_ONE,
51     bytesize=serial.EIGHTBITS,
52     timeout=1
53 )
54
55 # Configure two SPI channels, one for the TC77 temp sensor and a second for the
56 # AD5160 digital potentiometer.
57 spi_TC77 = spidev.SpiDev()
58 spi_AD5160 = spidev.SpiDev()
59
60 spi_TC77.open(0,0)           # Attach the object to the SPI hardware
61 spi_AD5160.open(0,1)
62
63 spi_TC77.max_speed_hz = 500000 # Limit is 2 MHz
64 spi_AD5160.max_speed_hz = 50000 # Limit is ~600 kHz
65 spi_TC77.mode = 0b00 # 0b00 corresponds to CPOL = 0, CPHA = 0
66 spi_AD5160.mode = 0b00
67
68 # Configuration of GPIO for LED enable pin, which turns on LED Solid-State Relay
69 ledPin = 22
70 GPIO.setmode(GPIO.BCM)
71 GPIO.setup(ledPin, GPIO.OUT)
72
73 # For now, LED is always enabled (would be disabled if RPI is off)
74 GPIO.output(ledPin, GPIO.HIGH)
75
76
77 print 'Socket, serial, and SPI all configured.'
78
79 def readTC77():
80     rawtemp = spi_TC77.readbytes(2) # Read two bytes of temp data
81     temp = (rawTemp >> 3) * 0.0625 # First three LSB are always logic 1, and
82     return temp # must be
83 'deleted'. Also 1 LSB = 0.0625 degC
84
85 def setBrightness(brightness):
86     # Make sure brightness value is between 0-255 since AD5160 potentiometer only
87     # has an 8bit DAC. 0 = ground, 255 = Vref
88     if (brightness < 0):
89         brightness = 0
90     elif (brightness > 255):
91         brightness = 255
92
93     # The RCD-24 LED driver attributes an input voltage of 0 = MAX ON,
94     # which is counter-intuitive. Inverse logic to make 0 = OFF
95     brightness = 255 - brightness
96
97     # Write modified brightness value to the digital potentiometer via SPI

```

```

98         spi_AD5160.writebytes([brightness])           # writebyte method expects list
99
100 if __name__ == '__main__':
101     #Lets loop awaiting for your input
102     while True:
103         # First read climate board status. "good" and "bad" should be the responses
104         climate_status = ser.readline() # Status is a string:'good' or 'bad'
105
106         # Next check the TC77 temperature sensor on the LED PCB for excessive temps
107         temp_C = readTC77()
108         if temp_C > 70:
109             temp_status = 'bad' # Temps greater than 70C signify overheating
110         else:
111             temp_status = 'good'
112
113         # If either the pod climate is bad or the LED is hot, then signal problem
114         # to the pier-side PC server.
115         if (climate_status == 'bad' || temp_status == 'bad'):
116             status = 'bad'
117         elif (climate_status == 'good' && temp_status == 'good'):
118             status = 'good'
119         else:# If neither of the conditions are true, then
120             status = 'standby' # there is an issue with string signals.
121
122         # Next cycle TCP socket data from the webserver PC
123         identifier = 'Rpi_1'
124         prompt = identifier + ',' + status + '\n'
125
126         s.sendall(str.encode(prompt))
127         reply = s.recv(1024) # Still needs to be decoded!
128
129         # Note that the expected server reply is a comma delimited string of
130         # the form '<identifier>,<brightness 0-255>'. Must parse incoming packet
131         reply_decoded = reply.decode('utf-8') # reply_parsed[0] = identifier
132         reply_parsed = reply_decoded.split(',')# reply_parsed[1] = 0-255
133
134         # Assign requested LED brightness to the LED control
135         brightness = int(reply_parsed[1])
136         setBrightness(brightness)
137
138         if reply_parsed[0] == 'Terminate':
139             break
140         print reply
141
142     # Close necessary methods upon termination
143     spi_TC77.close()
144     spi_AD5160.close()
145     s.close()
146     GPIO.cleanup()
147
148 # EOF

```

Appendix J: List of TODO Items and Materials to Purchase

Steps to complete sheave/winch assembly:

1. Finish welding gussets and winch mount tabs to sheave holder assembly.
2. Mount planetary gear reducer to winch frame with drive sprocket installed. Gear reducer output shaft may need to be lengthened or drive sprocket may protrude over the end of the shaft. Ensure sprocket engages with winch drive chain.
3. Secure servo using supplied aluminum mounting arms and 80-20 extrusion, as shown in CAD model.
4. Design and install waterproof servo cover and heater system (This step can happen in parallel). Note this system can connect directly into 24VDC that is available to the pod electronics through the slipring.
5. Secure sheave/winch holder assembly to pier using J-hooks and Delrin spacers.
6. Secure winch to sheave/winch holder assembly using bolts through vertical mount tabs.

Materials needed to complete sheave/winch assembly:

1. Metal/Plastic enclosure for servo (to be designed).
2. Metal/Plastic enclosure for winch (to be designed).
3. Resistive heating element for servo enclosure.
4. Thermocouple or thermistor for servo enclosure.
5. Small control board to control heating element (if under closed-loop temperature control).

Steps to complete wiring:

1. Finalize location of winch control box in pier control room. Ensure E-Stop switches are easily accessible and that cables will not be strained. Use slotted hole strut channel if mounting enclosure to wall.
2. Run cables and wires from winch control box to winch. The following cabling needs to be run:
 - a. 24VDC and GND – Connect to slipring input.
 - b. Ethernet – Connect to ethernet header on slipring input.
 - c. 3-phase cable for winch power – Connect directly to servo.
 - d. Encoder cabling (available from Yaskawa or can be custom-made with Molex connectors. See link: <https://www.yaskawa.com/products/motion/sigma-5-servo-products>) – Connect directly to servo.
 - e. (Optional, for additional features such as fresh-water rinse). Additional 2-conductor pair for 24VDC appliances, 3-conductor pair for 24VDC sensors.

Long sections of these cables are already attached directly to the winch control box. They may be long enough, may need to be extended, or may need to be replaced entirely. Approximate length from control room to winch: 35m, so 50m segments are recommended for purchase to ensure enough cabling.



3. Connect camera pod to cart, secure cart to strain relief of winch cable, connect SubConn electrical connector on pod to end of winch cable. Verify ethernet and 24VDC connectivity from winch controller.
4. Perform test move of servo from MotionWorks IEC software from a computer attached to winch controller switch. There are two methods to complete this test:
 - a. Set winch controller switch to /24 subnet through admin interface (accessible at 192.168.0.1). Set ServoPak to E-INIT mode. Set test computer's NIC address to 192.168.1.2/24. Connect to PLC at 192.168.1.1. Perform test move through hardware configuration window.
 - b. Connect directly to ServoPak PLC (not through switch). Set PLC to E-INIT mode. Set test computer's NIC address to 192.168.1.2/24. Change PLC's IP address to an address on the 192.168.0.0/24 subnet (i.e. 192.168.0.100/24). Take PLC out of E-INIT mode by toggling the DIP switch. Change test computer's NIC address to 192.168.0.200/24. Connect test computer's NIC through winch controller switch. Perform test move through hardware configuration window.

Materials needed to complete wiring:

1. 50m ethernet cable
2. 50m 2-conductor 24+VDC cable
3. 50m 3-conductor high-voltage cable
4. 50m Yaskawa encoder cable (see above)
 - Or make custom cable, see Sigma 5 servo product link. In this case, need to purchase Molex connectors.

Steps to complete network configuration:

1. Determine Pier's router configuration. Are we connected to Cal Poly's campus network? If so, contact IT, a pinhole will need to be enabled to allow external communication.

2. Determine subnet of pier's internal network. Set all Pier Portal network devices to be on the same subnet. If pier's subnet is 192.168.1.0/24, then proposed IP addresses from report can be used. If subnet is 192.168.0.0/24, replace third octet of each proposed IP address with 0, still use /24 subnet. Ensure IP addresses do not conflict with existing pier devices.
3. Reflect changed ServoPak PLC IP address in winch.py code file.
4. Configure port forwarding for ports 80, 8001, 8002, and 8003 to forward incoming packets to the correct Pier Portal system device. Use IP addresses determined from previous step. Allow traffic over TCP and UDP. Note: If TLS is enabled, use port 443 and port 4431, 4332, and 4333, respectively.
5. Create domain name. Point domain name to pier's router's public IP address.
6. Reflect changed port numbers and domain name in static PPLive JavaScript files

Steps to test software:

1. Start Django web application on main Pier Portal server (Raspberry Pi).
2. Update all software on server
3. Connect to webpage (if on internal network, use internal IP address, otherwise use domain name). Perform test move. Verify working functionality.

Additional peripherals:

- Freshwater Rinse (Freshwater tubing/garden hose, sprayer valve, 24VDC solenoid to turn on and off water).
- Winch cable squeegee.