# IS305: Unmanned Monitoring and Maintenance Systems


EG3301R DCP Project
Individual Final Report

**Submitted by**

Parvathi Ranjith Menon, A0194448R

**Supervisors**

Dr. Edwin Koh

Sim Zhi Min

Gary Wilson

## Introduction

The following individual report will be discussing the electrical and software aspects of our team's prototype versions. This includes, for each prototype version, the electrical schematic which shows the detailed connections of the electrical components in the respective prototype, the power budget table that summarizes the amount of power required for each of the electrical components, a detailed explanation of the code used and any additional features with respect to the electrical and/or software side of the prototype.

## Prototype v1.0

### i. Electrical Schematic

Figure 1 in the next page shows an image of the electrical schematic of the first rendition of our prototype. In this version of the prototype, an analog water sensor was used to detect the presence and subsequently the percentage of water in the drain flow. The water sensor outputted analog values of 0 to 1023 corresponding to the voltage values 0V to 5V. However, the Raspberry Pi GPIO (General Purpose Input/Output) pins can only input and output digital values of 3V3 for high and 0V for low. Therefore, a logic shifter was needed to scale the voltage range of the output from the water sensor from 0V-5V to 0V-3.3V and an Analog-to-Digital Converter (ADC) was used to convert the analog signals into digital signals that could be recognized by the Raspberry Pi.

The other electrical components in the schematic include the Raspberry Pi camera, an accelerometer and the relays connecting to the two electric valves. The purpose of the Raspberry Pi camera was to monitor the drainage flow during each scheduled drain cycle to ensure that the draining process was taking place correctly. The accelerometer was used to measure the tilt of the system so as to stop the drainage when the tilt angle exceeded a certain threshold value. This threshold value would correspond to the angle at which the opening to the drainpipe would be covered by oil instead of water due to the tilt of the system. The relay module connected the Raspberry Pi to each of the valves. Depending on the signal outputted by the Raspberry Pi, either one or both relay channels would open and allow current to flow to the valves, thereby opening or closing each of them as required.

### ii. Power Budget Table

The power budget table, as seen in Figure 2 in the next to next page, is an overview of the total power consumption of the system. It shows the breakdown of the power consumption and helps us to analyze the required power to be supplied to the system and each of its electrical components accordingly so as to ensure the system works properly and no components are damaged. Furthermore, the power budget table allows us to evaluate the efficiency of the system and see where power could be saved.

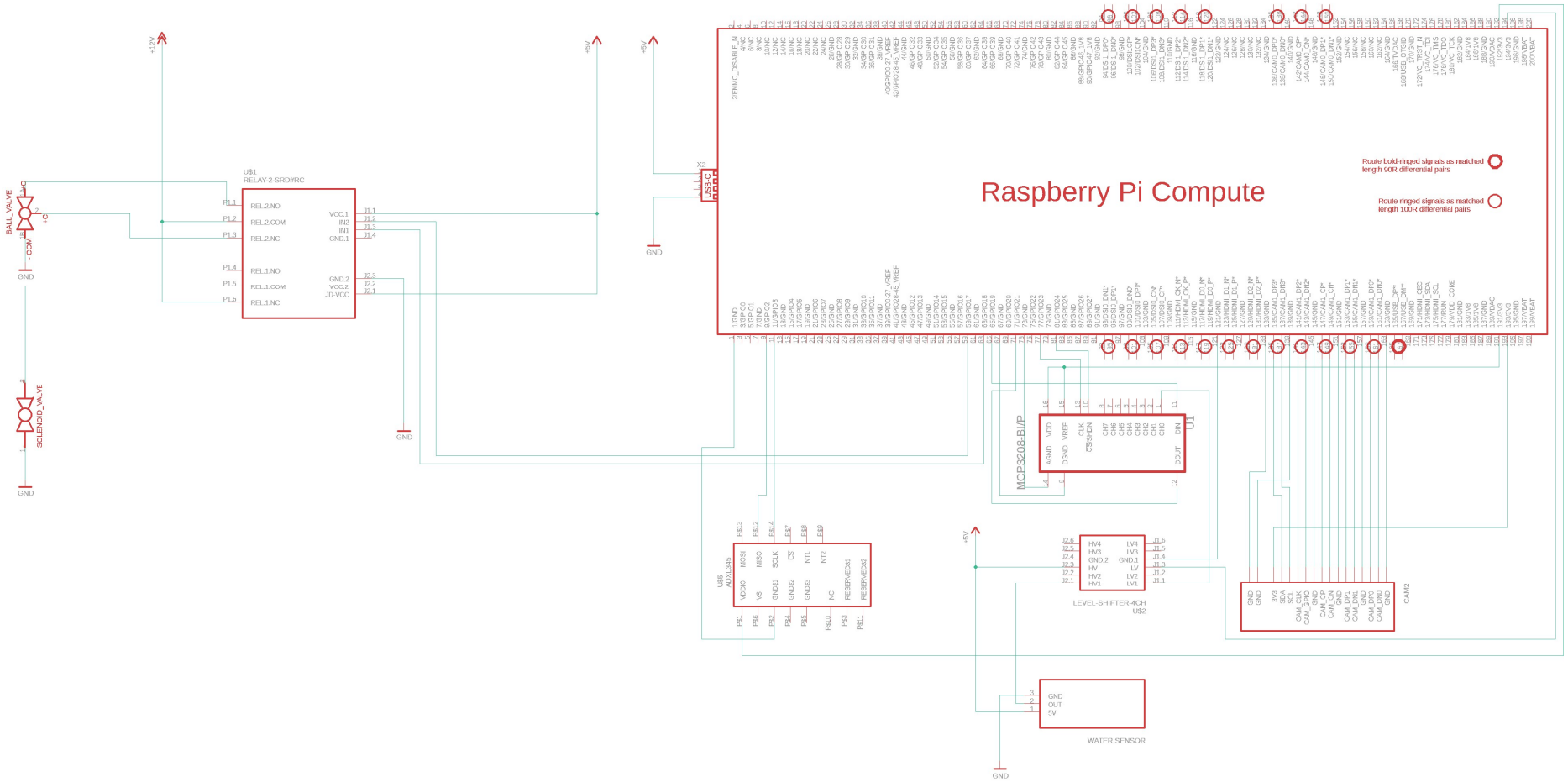


*Figure 1: Electrical Schematic for Prototype v1.0*

| COMPONENT | State | VOLTAGE (V) | CURRENT (mA) | QTY | POWER (W) | REMARKS |
|---|---|---|---|---|---|---|
| Microcontroller (Arduino Uno Atmega 328) | System , Active | 12 | 0.2 | 1 | 2.4 | On our designed system of 12V power (Also max recommended voltage) |
| Single Board Computer (Raspberry Pi 4) | Idle | 5 | 570 | 1 | 2.85 | Test Conditions: No power consumption from external USB devices connected to Pi |
| Single Board Computer (Raspberry Pi 4) | 100% X 4 CPU | 5 | 1280 | 1 | 6.4 | 4 |
| Raspberry V2 Camera | Operating | 3.3 | 250 | 1 | 0.825 | |
| Solenoid Valves NC (Ball) | ON | 12 | 500 | 1 | 6 | |
| Solenoid Valves NO (Diaphragm) | ON | 12 | 2900 | 1 | 34.8 | |
| Gyroscope/tilt sensor (ADXL345) | Standby | 2.5 | 0.0001 | 1 | 0.00025 | Negligible Power Consumption |
| Gyroscope/tilt sensor (ADXL345) | Measurement | 2.5 | 0.023 | | 0.0575 | Negligible Power Consumption |
| Water Level Sensor (raindrop detection module) | | 5 | 15 | 1 | 0.075 | |
| Relay | | 12 | | 1 | | |
| Step Down Converter | | - | - | 1 | | |
| Power supply | | | | 1 | | |
| Total: | | | | | 50.55775 | |

*Figure 2: Power Budget Table for Prototype v1.0*

From Figure 2, it can be seen that the maximum amount of power consumed by the prototype during operation is approximately 51W. The component diagram shown in Figure 9 in the Appendix gives a more detailed, visual representation of the electrical signals and the corresponding connections between the different electrical components.

### iii. Integration of RPi

In order to start and boot up the Raspberry Pi, the Raspberry Pi OS (Raspbian) had to be installed onto it. Alongside this, several additional sources and libraries needed to be installed in order to accommodate for our prototype's system. These included Python 3 for writing the overall code, the Adafruit Python MCP3008 library for the use of the ADC and Python smbus library for the use of the accelerometer. Additional prerequisite settings included enabling SSH/VNC and Camera on the Raspberry Pi to access and control the Raspberry Pi and use the Raspberry Pi camera, respectively.

### iv. Code

Figure 10 in the Appendix shows the code that was written to be used for the prototype v1.0. This code was written to facilitate testing so it is not a fully refined code that can be implemented in the actual working system. The code allows for testing each electrical component, namely the Raspberry Pi camera, the relay, and the sensors – water sensor and accelerometer. This code would have been modified to accommodate for the threshold values and requirements with respect to the final prototype to be used for its actual implementation. Figure 3 in the next page is a flowchart showing the overall code logic for this prototype version.

Lines #1 to #10 in Figure 10 shows the code used to import the required libraries for each device – Raspberry Pi Camera, Accelerometer and Analog-to-Digital Converter (ADC) – and the multiprocessing feature to coordinate and run all the functions side by side. The following lines #12 to #19 is the initialization code for the accelerometer and ADC. Lines #22-#34, #35-#44 and #46-#55 are the function codes for testing the relay,

camera, and sensors, respectively. It does simple input-output functions for us to analyze the working of each electrical component and deciding the threshold values that would apply for our prototype system. Each of these functions is called in the 'main' function as processes for multiprocessing.

The 'main' function code corresponds to lines #57 to #66 in Figure 10. The 'main' function controls the multiprocessing of the devices' function code so that the processes can run side by side without interfering with one another. The code begins with defining each device function as a process, followed by starting each process so that it can run simultaneously and finally joining each process to complete the program. This multiprocessing method allows for reduced delays within the program and allows for a faster rate of polling and monitoring, thereby improving the efficiency of the drainage system.

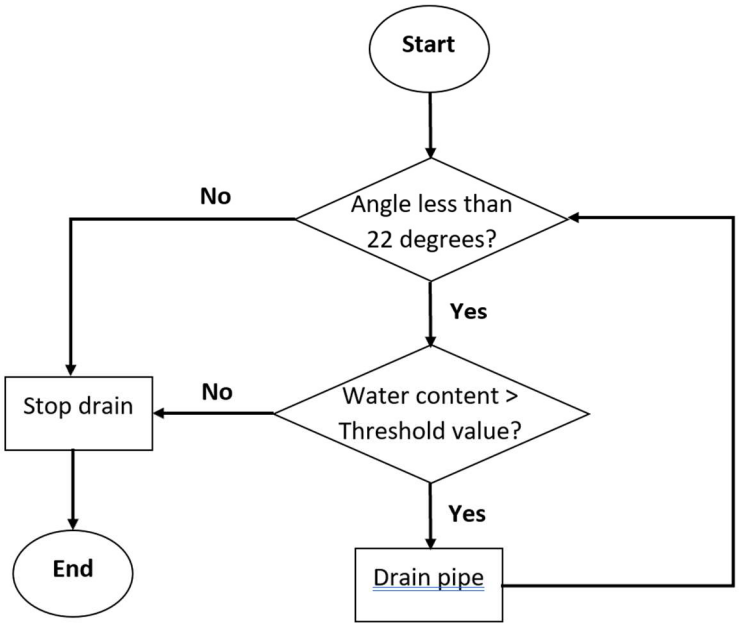


*Figure 3: Code Logic Flowchart for Prototype v1.0*

## v. PCB Design

In order to make our electrical circuit more compact and organized, a Printed Circuit Board (PCB) was designed using the Autodesk Eagle software. This helped us get rid of the several long wires and connections and instead solder our electrical components directly onto the PCB. The PCB was designed to be a shield for the Raspberry Pi as can be seen in Figure 12 in the Appendix. Figure 11 in the Appendix shows the schematic of the PCB and how each of the electrical components are connected to the power supply and one another on the board.

Special thanks to Mr. Ee Wei Han, Eugene for his webinar and advice on how to design a PCB.

## Prototype v2.0

### i. Electrical Schematic

Upon learning about the unreliability and inaccuracy of the analog water sensor after several rounds of testing, we reiterated our design thinking process and changed our prototype design to allow for better detection and differentiation of water and oil levels. Our prototype v2.0 design makes use of the Raspberry Pi camera to capture continuous images of the drainpipe and these images are analyzed against our machine learning model.

Our machine learning model has been trained to detect and distinguish the water and oil level from the image captured and decide whether or not it should drain from the settling tank *[Refer to Section 2: Royston Shieh]*. Depending on the decision made by the trained model, the Raspberry Pi sends the appropriate signal to the relay module through its GPIO pins, thereby opening or closing the electric valves as required.

Figure 5 in the next page shows the electrical schematic of this iteration of our prototype.

### ii. Power Budget Table

| COMPONENT | State | VOLTAGE (V) | CURRENT (mA) | QTY | POWER (W) | REMARKS |
|---|---|---|---|---|---|---|
| Single Board Computer (Raspberry Pi 4) | Idle | 5 | 570 | 1 | 2.85 | Test Conditions: No power consumption from external USB devices connected to Pi 4 |
| Single Board Computer (Raspberry Pi 4) | 100% X 4 CPU | 5 | 1280 | 1 | 6.4 | |
| Raspberry V2 Camera | Operating | 3.3 | 250 | 1 | 0.825 | |
| Solenoid Valves NC (Ball) | ON | 12 | 500 | 1 | 6 | |
| Solenoid Valves NO (Diaphragm) | ON | 12 | 2900 | 1 | 34.8 | |
| Relay | | 12 | | 1 | | |
| Power supply | | | | 1 | | |
| Total: | | | | | 48.025 | |

*Figure 4: Power Budget Table for Prototype v2.0*

Figure 4 above shows the power budget table for prototype v2.0 and Figure 13 in the Appendix shows the component diagram for prototype v2.0 with the corresponding connections and power signals for each electrical component.

From Figure 4, it can be seen that the maximum amount of power consumed by the prototype v2.0 system is 48.025W which is lesser than the 51W maximum power consumption of prototype v1.0 seen earlier. This shows that our second prototype iteration is not only more reliable and accurate than the first but also more power efficient.

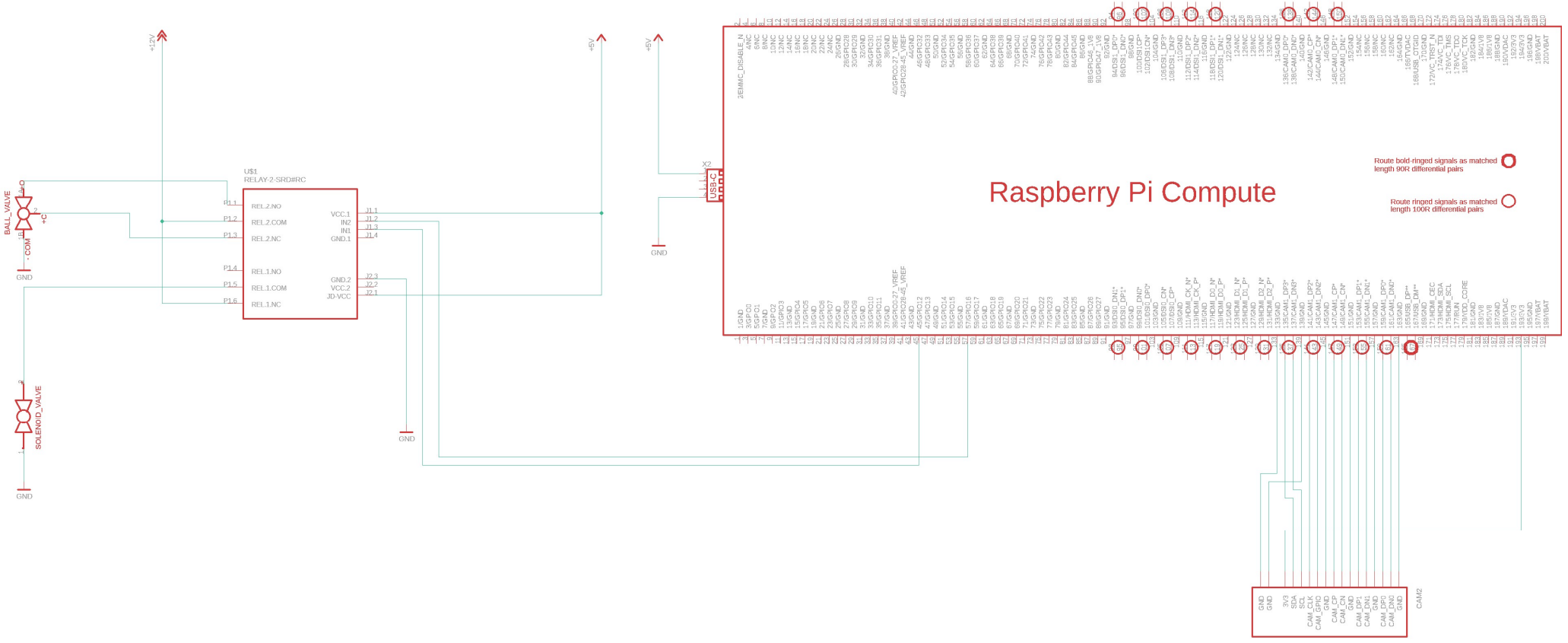


*Figure 5: Electrical Schematic for Prototype v2.0*

### iii. Integration of RPi

The initial basic start up and additional prerequisite settings is the same as that for prototype v1.0. This includes installing Raspbian and enabling SSH/VNC and Camera on the Raspberry Pi. The prototype v2.0 system required additional sources and libraries to function correctly this included Python 3 for writing and managing the overall code, Tensorflow, Keras and PIL to load and run the machine learning model and rclone for the Google Drive sync which will be discussed in more detail in the upcoming section, Cloud Storage.

### iv. Code

Figure 14 in the Appendix shows the python script that was used in our prototype v2.0 system.

Lines #1 to #11 in the code imports the Python libraries required to run the machine learning model, operate the Raspberry Pi camera and to use the GPIO pins for the relay module. The GPIO pins are setup in lines #13 to #18. GPIO pin 12 is connected to the ball valve whereas GPIO pin 16 is connected to the solenoid valve. The solenoid valve is a normally open valve and plays the role of a backup valve for our prototype system. The Raspberry Pi sends a continuous LOW signal of 0V to this backup solenoid valve as seen in line #18 of Figure 14 therefore keeping it constantly open. This valve snaps shut when there is a power failure hence acting as a fail-safe system.

The Raspberry Pi sends a HIGH signal of 3.3V to close and a LOW signal of 0V to open the ball valve. The ball valve is initially closed as seen in line #17 as the GPIO pin 12 corresponding to the ball valve is given a HIGH output. The ball valve is then opened for a few seconds at the beginning of every scheduled run to drain the pipe to get rid of the oil residue left behind from the previous run. This is shown in lines #27 to #30. It was chosen to keep the GPIO pin 12 at LOW for 15 seconds as seen in line #29 to account for the time taken for the valve to open as well as the time needed to clear out the pipe. This initial draining of the pipe allows for more accurate decisions to be made on whether to run the scheduled drain.

During each scheduled drain, the Raspberry Pi camera is used to take a picture at a polling rate of one second. The image taken is then used to compare against the machine learning model to decide whether to open or close the valves. Simultaneously, the image is saved and uploaded onto our team's Google Drive to be used for monitoring and inspection purposes. The process of pushing out data onto the drive is explained in more detail in the following section, Cloud Storage.

Lines #20-#24, #32-#33 and #47-#72 corresponds to the loading and using the machine learning model *[Refer to Section 2: Royston Shieh]*. Based on the prediction result made by the machine learning model, the output signal from GPIO pin 12 is changed as seen in the If-Else statement in lines #73-#81.

If the prediction result is equal to '1', then the GPIO pin 12 is made to output LOW to drain the pipe and when the prediction result is equal to '0', the GPIO pin outputs HIGH to stop draining the pipe. This conditional logic is summarized in Figure 6 below.

| Machine Learning Model Result | GPIO Pin 12 | Ball Valve |
|---|---|---|
| 1 | LOW | Open |
| 0 | HIGH | Close |

*Figure 6: Conditional Logic for Prototype v2.0*

To run the draining cycle on a scheduled timing, the code responsible for the drain-no drain process has been put into a function named 'cv' which is called in the 'main' function (line #83 to #90) every 90 seconds. We chose to run the schedule at 90 second intervals to mimic the real-life bi-hourly drain schedule onboard ships and for easier demonstration purposes.

For a simpler overall understanding, Figure 7 below shows a flowchart of the code logic for this prototype version.

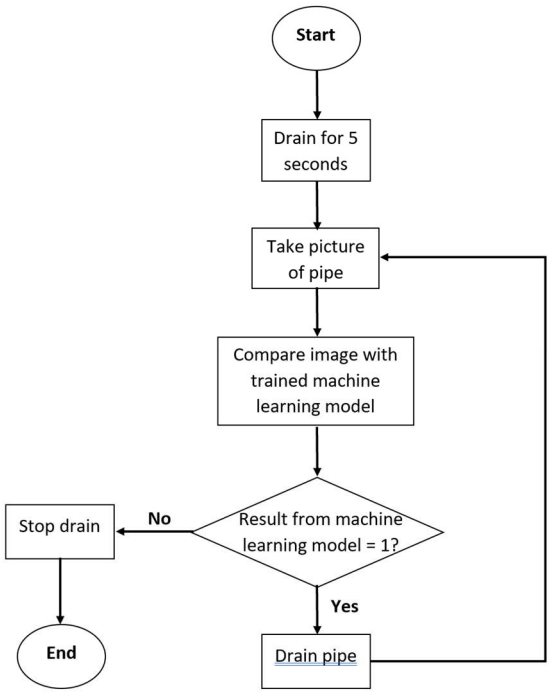


*Figure 7: Code Logic Flowchart for Prototype v2.0*

### v. Cloud Storage

The data and images collected from each scheduled drain can be very valuable when it comes to analyzing the drainage system as well as the state and behavior of the ship's settling tank. The data collected would include measurements such as the drain time for each scheduled run, the amount of water drained per run, the amount of oil lost during draining and other data collected by piggy-back sensors *[Refer to Section 6:*

*Thirunavukarasu Bhalaji]*. The images produced during each run can help to monitor whether each scheduled drain takes place as expected as well as the condition of the drainage pipe. Any anomalies, damage or clogs can be looked into and fixed accordingly. Furthermore, these data and images can be used to improve the timing of the scheduled drains and further improve the machine learning model to improve the accuracy of our draining system.

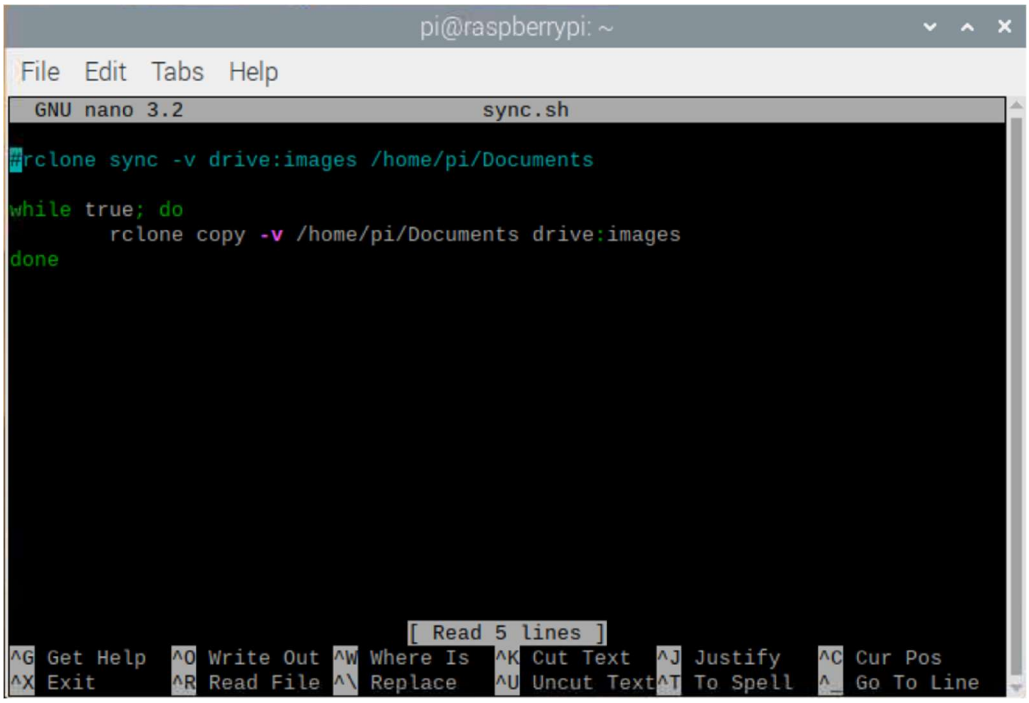


*Figure 8: Code for Syncing Files to Google Drive*

Therefore, to make all this useful information available on a real-time basis to the engineers on-shore, an executable code has been implemented in the prototype system to not only store the data locally but to also push-out the data to a cloud drive. This code, as seen in Figure 8 above, runs continuously and uploads any new images or data produced by the system onto the drive instantaneously. This way the engineers onshore can analyze the information immediately and report or take necessary actions if there are any faults detected. To mimic the real-life uploading of files and updating the drive, we are pushing the data into our team's shared Google Drive so that it can be accessed by each of our team members from any location at any time. Figure 15 in the Appendix shows the messages printed out on the command line terminal on the Raspberry Pi as the executable code runs and updates the drive. Figure 16 in the Appendix shows a snapshot of our team's Google Drive folder and how it appears with the images of the scheduled drains uploaded to it.

We understand that the ship would have several thousands of signals and data being sent to shore and the priority would be given to the data from the most crucial systems on board the ship. Furthermore, greater detail about a particular function system would only be transmitted in the case of a fault. Therefore, this idea of pushing out data from

the draining system is merely a concept for the time being and demonstrates what could be possible if the bandwidth permits.

### vi. Graphical User Interface (GUI)

For our future works, we hope to implement a Graphical User Interface (GUI) to portray the data and information from each scheduled drain as well as the status of the system. Figure 17 in the Appendix depicts our vision for the GUI. The GUI is created to open as a dashboard so as to allow engineers to get a quick overview of the situation of the system in one glance. The dashboard would show the latest information and status of the system including a live picture of the drainpipe, average time taken for each drain, last and next scheduled drain and the volume of water drained during each completed run. We also hope to implement additional features such as an emergency tab that will show any critical anomalies detected or issues that require immediate attention.

In reality, a ship would have its own GUI system in place. Therefore, our draining system would be providing the necessary output data in the format required by the ship's integrated system so that it can be displayed on its GUI. The above-mentioned idea is for our prototype version and to mimic the real GUI system of the ship.

## Conclusion

Through this report, the electrical, power and software aspects of the two prototype versions were vastly explored. It discussed the specific pin-to-pin connections in detail and how all the electrical components came together with the help of the electrical schematics. Furthermore, going in depth into the code analysis allowed for a better understanding of the code and conditional logic as well as the working and implementation of each of the electrical components within the software. Finally, additional features and future works concepts were discussed thereby depicting the potential of the prototype.

In conclusion, not only was the programming aspect of our final prototype explained in detail but also the evolution of the software and electrical schematic through each iteration of the prototyping process was observed.

## Appendix

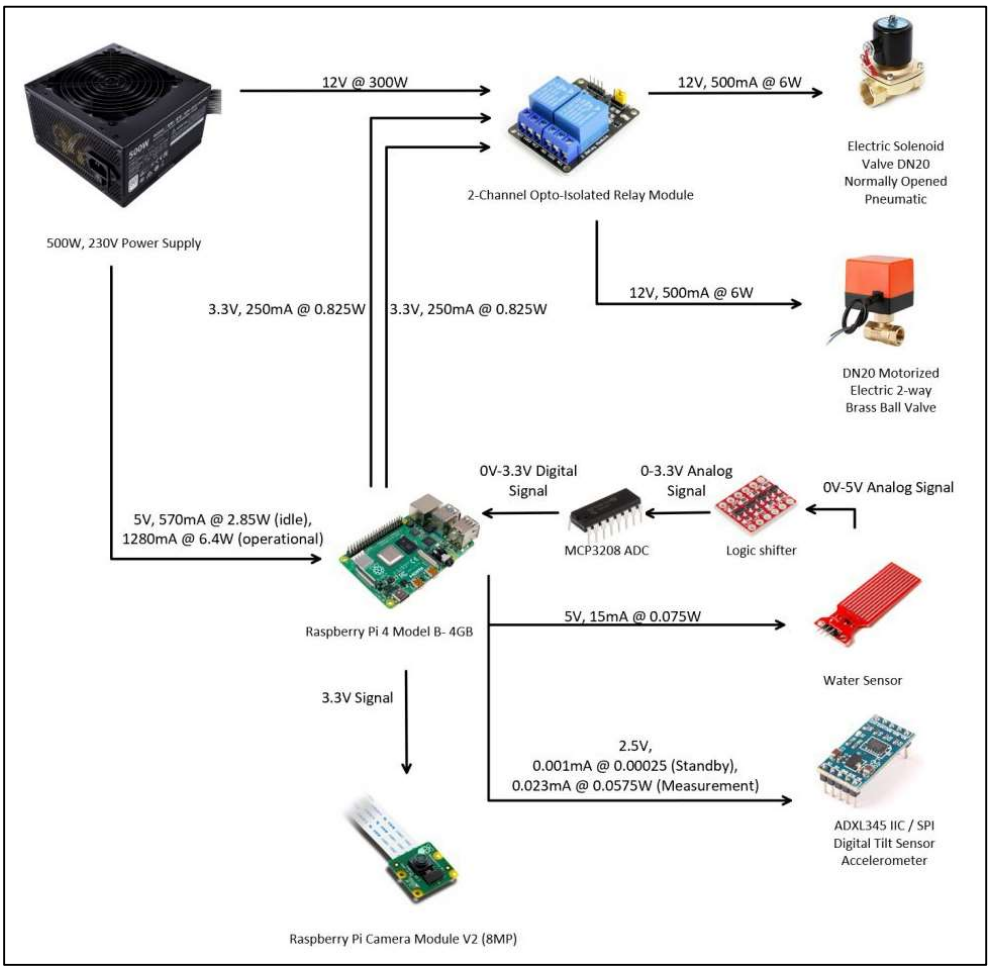


*Figure 9: Power Component Diagram for Prototype v1.0*

```
#1      import multiprocessing as mp
#2      import time
#3
#4      # Library for RPi Camera
#5      from picamera import PiCamera
#6
#7      # Libraries for accelerometer
#8      import board
#9      import busio
#10     import adafruit_adxl34x
#11
#12     # Initializing and reading accelerometer
#13     i2c = busio.I2C(board.SCL, board.SDA)
#14     accelerometer = adafruit_adxl34x.ADXL345(i2c)
#15
#16     # Initializing ADC
#17     SPI_PORT = 0
#18     SPI_DEVICE = 0
#19     mcp = Adafruit_MCP3008.MCP3008(spi=SPI.SpiDev(SPI_PORT,
SPI_DEVICE))
#20
#21     # Relay module to be modified based on scheduled cycles
#22     def relay():
#23             while True:
#24                     GPIO.setmode(GPIO.BCM)
#25                     GPIO.setup(12, GPIO.OUT)
#26                     GPIO.setup(16, GPIO.OUT)
#27                     GPIO.output(12, GPIO.HIGH)
#28                     GPIO.output(16, GPIO.HIGH)
#29                     time.sleep(2)
#30                     GPIO.output(12, GPIO.LOW) # First relay on
#31                     time.sleep(2)
#32                     GPIO.output(16, GPIO.LOW) # Second relay on
#33                     time.sleep(5)
#34                     GPIO.cleanup() # All relays off
```

```
#35     def camera():
#36             while True:
#37                     capture_time = time.strftime('%H:%M:%S')
#38                     camera = PiCamera()
#39                     camera.start_preview()
#40                     camera.start_recording('/home/pi/Desktop/' +
str(capture_time) + '.h264')
#41                     time.sleep(5) # To be changed based on scheduled cycles
#42                     camera.stop_recording()
#43                     camera.stop_preview()
#44                     camera.close()
#45
#46     def sensors():
#47             while True:
#48                     value = mcp.read_adc(0)
#49                     sensor_time = time.strftime('%H:%M:%S')
#50                     print('Red sensor at t = ' + str(sensor_time) + ', Reading = '
+ str(value) + ', X = %f Y = %f Z = %f' %accelerometer.acceleration)
#51                     file = open('/home/pi/data_log.csv', 'a')
#52                     file.write('Red sensor at t = ' + str(sensor_time) + ', Reading
= ' + str(value) + ', X = %f Y = %f Z = %f' %accelerometer.acceleration + '\n')
#53                     file.flush()
#54                     file.close()
#55                     time.sleep(0.5)
#56
#57     if __name__ == '__main__':
#58             p1 = mp.Process(target = camera)
#59             p2 = mp.Process(target = sensors)
#60             p3 = mp.Process(target = relay)
#61             p1.start()
#62             p2.start()
#63             p3.start()
#64             p1.join()
#65             p2.join()
#66             p3.join()
```

*Figure 10: Code for Prototype v1.0*

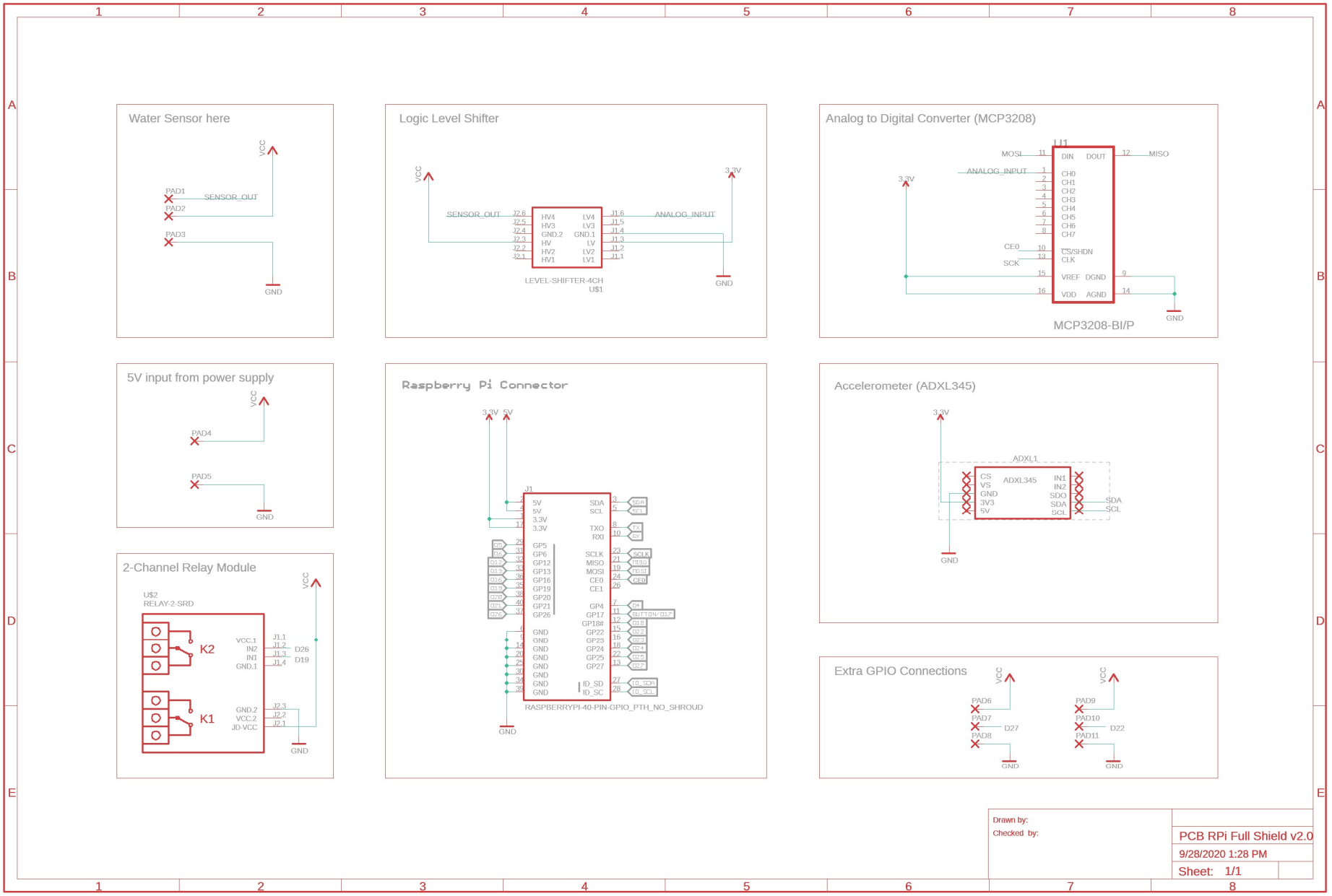


*Figure 11: PCB Schematic for Prototype v1.0*

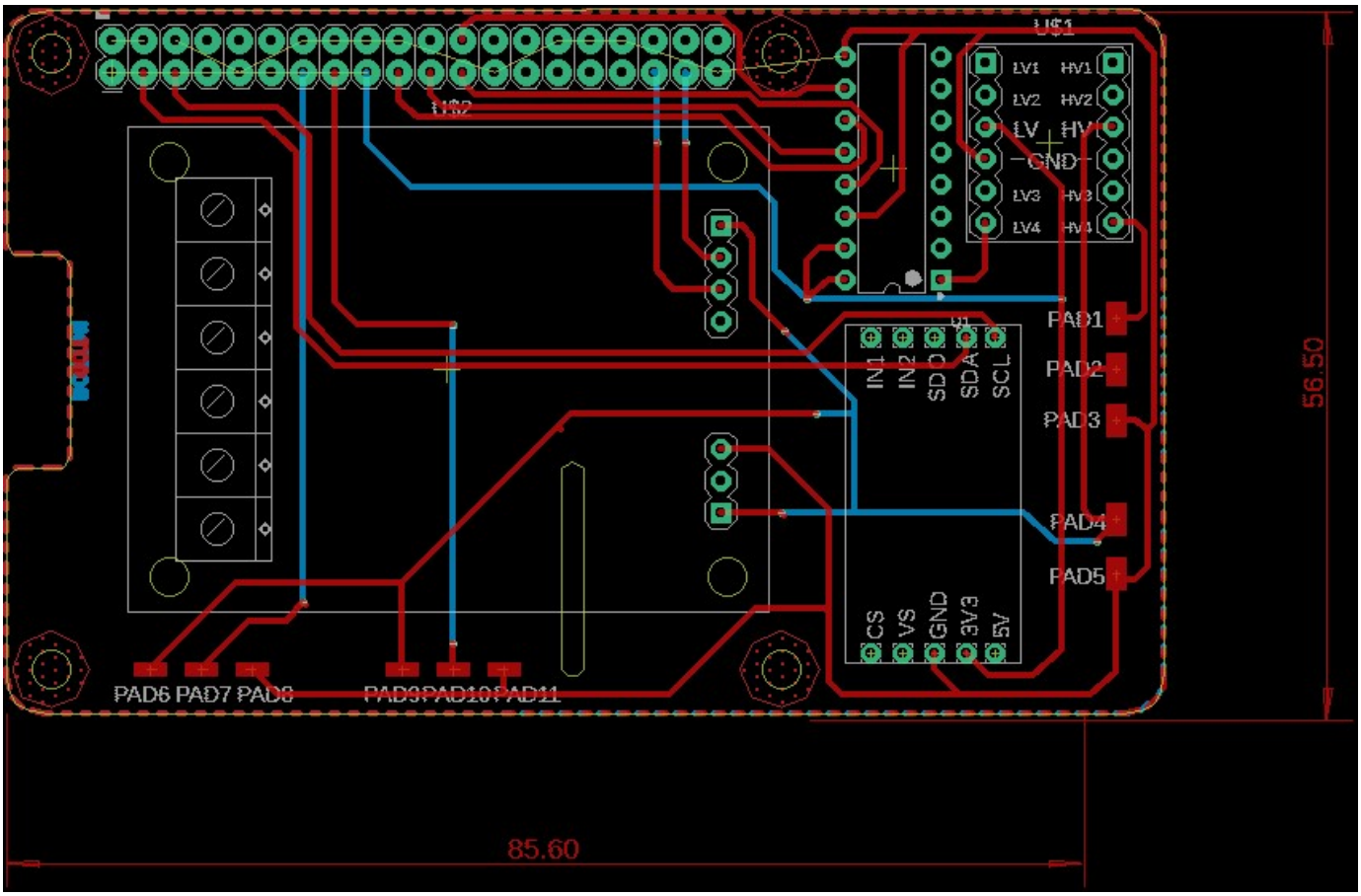


*Figure 12: PCB Board for Prototype v1.0*

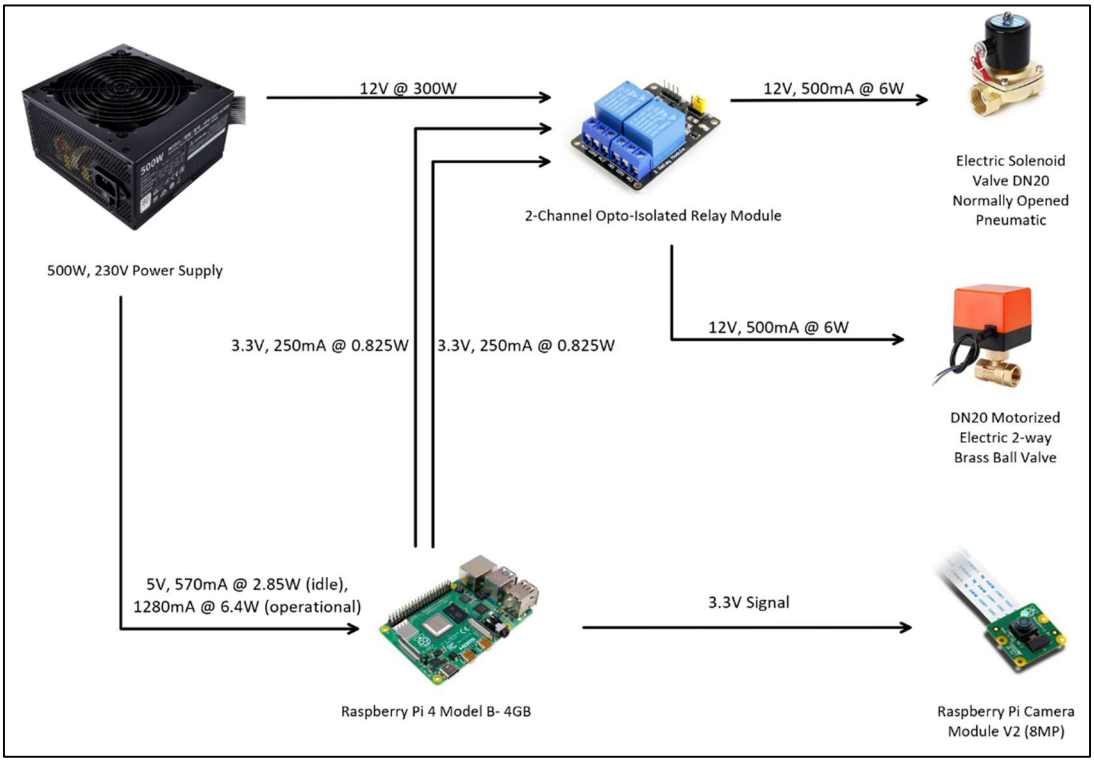


*Figure 13: Power Component Diagram for Prototype v2.0*

```
#1      import tensorflow.keras
#2      from PIL import Image, ImageOps
#3      import numpy as np
#4      import time
#5
#6      # Libraries for GPIO pins for relay
#7      import RPi.GPIO as GPIO
#8
#9      # Libraries for RPi Camera
#10     from picamera import PiCamera
#11     from time import sleep
#12
#13     # Setting up the relay
#14     GPIO.setmode(GPIO.BCM)
#15     GPIO.setup(12, GPIO.OUT)
#16     GPIO.setup(16, GPIO.OUT)
#17     GPIO.output(12, GPIO.HIGH)
#18     GPIO.output(16, GPIO.LOW)
#19
#20     # Disable scientific notation for clarity
#21     np.set_printoptions(suppress=True)
#22
#23     # Load the model
#24     model =
tensorflow.keras.models.load_model('/home/pi/Desktop/tf_pi/keras_model.h5')
#25
#26     def cv():
#27             # Initial drain to clear pipe
#28             GPIO.output(12, GPIO.LOW)
#29             sleep(15)
#30             GPIO.output(12, GPIO.HIGH)
#31             while True:
#32                     data = np.ndarray(shape=(1, 224, 224, 3), dtype=np.float32)
#33                     Ptest_list = []
#34                     # Take picture with RPi camera
#35                     camera = PiCamera()
#36                     capture_time = time.strftime('%H:%M:%S')
#37                     camera.rotation = 180
#38                     camera.resolution = (2592, 1944)
#39                     camera.framerate = 15
#40                     camera.start_preview()
#41                     sleep(1)
#42                     camera.capture('/home/pi/Desktop/image.jpg')
#43                     camera.capture('/home/pi/Documents/' +
str(capture_time) +'.jpg')
#44                     camera.stop_preview()
#45                     camera.close()
#46
#47                     # Replace this with the path to your image
#48                     image = Image.open('/home/pi/Desktop/image.jpg')
#54
#55                     #resize the image to a 224x224 with the same
strategy as in TM2:
#56                     #resizing the image to be at least 224x224 and then
cropping from the center
#57                     size = (224, 224)
#58                     image = ImageOps.fit(image, size, Image.ANTIALIAS)
#59
#60                     #turn the image into a numpy array
#61                     image_array = np.asarray(image)
#62
#63                     # Normalize the image
#64                     normalized_image_array =
(image_array.astype(np.float32) / 127.0) - 1
#65
#66                     # Load the image into the array
#67                     data[0] = normalized_image_array
```

*Figure 14: Code for drain/no drain for Prototype v2.0*

```
#68                     # Run the inference
#69                     prediction = model.predict(data)
#70                     predictioncls = [[1 if y == max(x) else 0 for y in x] for x in prediction]
#71                     predicclsarray = np.array(predictioncls)
#72
#73                     if predicclsarray[0][0] == 1:
#74                             # drain
#75                             GPIO.output(12, GPIO.LOW)
#76                             print("draining...")
#77                     else:
#78                             # stop drain
#79                             GPIO.output(12, GPIO.HIGH)
#80                             print("drain stopped...")
#81                             break
#82
#83      if __name__ == '__main__':
#84              try:
#85                      while True:
#86                              cv()
#87                              time.sleep(90)
#89              except KeyboardInterrupt:
#90                      GPIO.cleanup()
```

*Figure 14: Code for drain/no drain for Prototype v2.0 (Continued)*

```
pi@raspberrypi:~ $ sh sync.sh
2020/11/02 13:39:28 INFO  : 13:13:36.jpg: Copied (new)
2020/11/02 13:39:29 INFO  : 13:14:58.jpg: Copied (new)
2020/11/02 13:39:30 INFO  : 13:14:55.jpg: Copied (new)
2020/11/02 13:39:31 INFO  : 13:14:48.jpg: Copied (new)
2020/11/02 13:39:32 INFO  : 13:15:01.jpg: Copied (new)
2020/11/02 13:39:49 INFO  :
Transferred:        7.599M / 119.192 MBytes, 6%, 132.807 kBytes/s, ETA 14m20s
Checks:               117 / 117, 100%
Transferred:            5 / 51, 10%
Elapsed time:       1m3.5s
Transferring:
 *                                  13:15:04.jpg:  0% /2.149M, 0/s, -
 *                                  13:15:07.jpg:  0% /2.179M, 0/s, -
 *                                  13:15:10.jpg:  0% /2.186M, 0/s, -
 *                                  13:15:13.jpg:  0% /2.407M, 0/s, -

2020/11/02 13:39:57 INFO  : 13:15:10.jpg: Copied (new)
2020/11/02 13:39:58 INFO  : 13:15:04.jpg: Copied (new)
2020/11/02 13:39:58 INFO  : 13:15:07.jpg: Copied (new)
2020/11/02 13:39:59 INFO  : 13:15:13.jpg: Copied (new)
2020/11/02 13:40:02 INFO  : 13:15:16.jpg: Copied (new)
2020/11/02 13:40:04 INFO  : 13:17:04.jpg: Copied (new)
2020/11/02 13:40:04 INFO  : 13:17:07.jpg: Copied (new)
```

*Figure 15: Pushing out data & images to drive for Prototype v2.0*

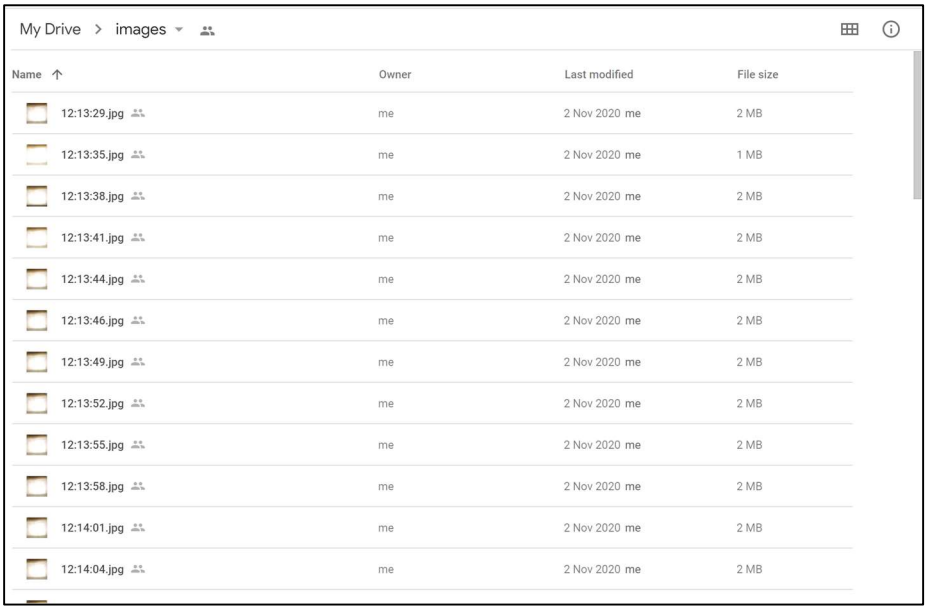


*Figure 15: Images from the Raspberry Pi uploaded onto our team's Google Drive folder*

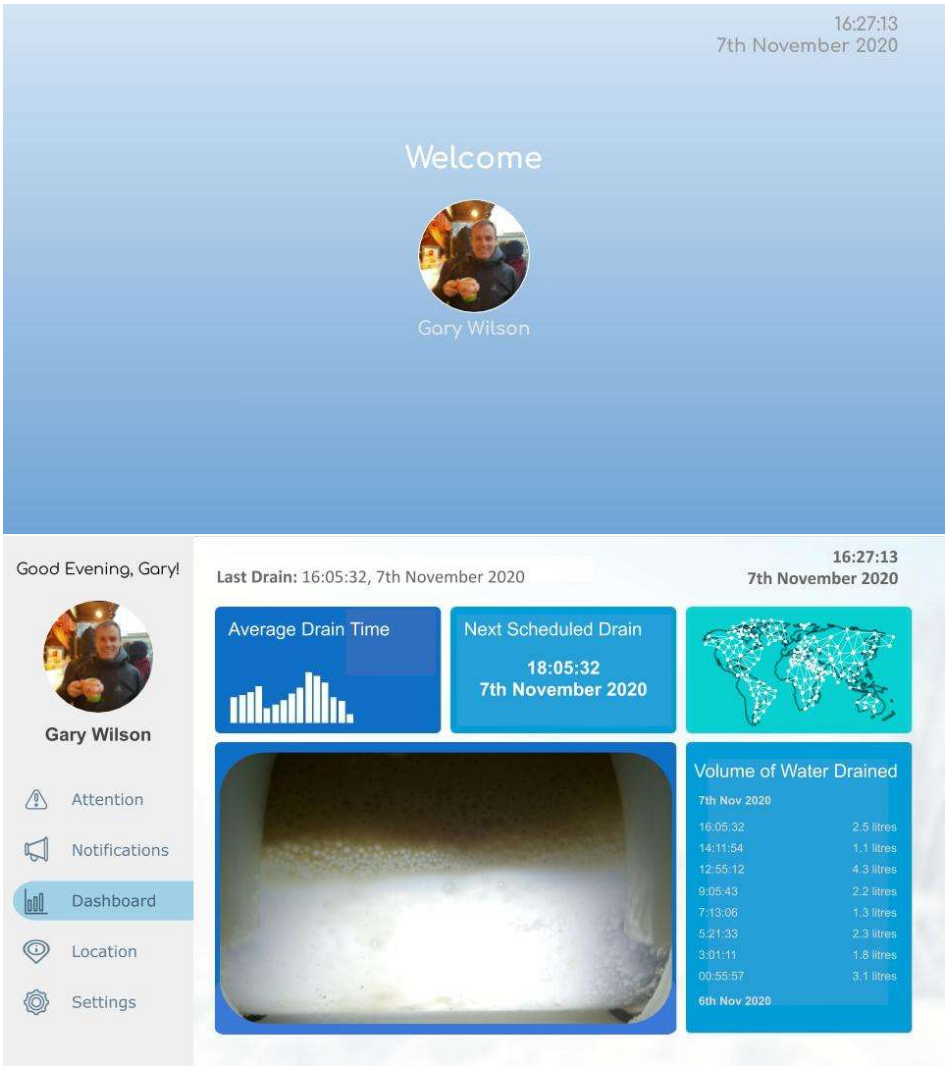


*Figure 16: Graphical User Interface Design*