CHALMERS
UNIVERSITY OF TECHNOLOGY

SSRS Drone Launcher

SW/HW testing and optimization of the automated platform

Bachelor's thesis in Electrical Engineering

QUENTIN DEMORY
ABDULRAHMAN ALSAMEL

Department of Electrical Engineering
Division of Systems and control
CHALMERS UNIVERSITY OF TECHNOLOGY
Bachelor's thesis EENX20-V20
Gothenburg, Sweden 2020

BACHELOR'S THESIS EENX20-V20

SSRS Drone Launcher

SW/HW testing and optimization of the automated platform

Bachelor's thesis in Electrical Engineering

QUENTIN DEMORY
ABDULRAHMAN ALSAMEL

Department of Electrical Engineering
Division of Systems and control
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2020

SSRS Drone Launcher
SW/HW testing and optimization of the automated platform
QUENTIN DEMORY
ABDULRAHMAN ALSAMEL

Bachelor's thesis EENX20-V20
Department of Electrical Engineering
Division of Systems and control
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Colophon:
The thesis was created using LaTeX 2_ε and `biblatex` and edited on `www.overleaf.com`. The typesetting software was the TeX Live distribution. The text is set in Times New Roman. Graphs were created using the MATPLOTLIB library in python. All picture licenses are released under CC BY-NC 4.0. All codes under MIT license.

Cover:
The launcher on its final day of testing at the Swedish Sea Rescue Society (SSRS)

Chalmers Reproservice
Gothenburg, Sweden 2020

SSRS Drone Launcher
SW/HW testing and optimization of the automated platform
Bachelor's thesis in Electrical Engineering
QUENTIN DEMORY
ABDULRAHMAN ALSAMEL
Department of Electrical Engineering
Division of Systems and control
Chalmers University of Technology

ABSTRACT

The personnel at Swedish Sea Rescue Society (SSRS) faces a multitude of different scenarios during rescue operation which can require different types of rescue gear and resources. As information about the gravity of distress is limited, a better understanding and overview of the accident would facilitate better resources management and therefore dramatically improve response time and adequate support, specially for critical interventions where larger towing boats or air ambulances may be needed. In an attempt to solve this problem SSRS has conducted a study through several theses and internal projects, resulting in an unfinished prototype of a drone launcher, to be the basis of future improvements.

The scope of this project is, in the first place, to assess the current operations of the prototype. Run tests to evaluate and report on its electrical system and operating levels. Assess the software implementation and develop a comprehensive document of the equipment's current state.
In the second place, add new functionalities for safety, such as lights and sound to indicate motion but also functionalities to monitor the system's health, such as temperature and position data as well as deploy its own internet connection for remote operations to the previously established current state. Also part of the scope was the delivery of a User document supporting general users with safe operating guidelines and an overview of components and functionalities to guide engineers and the curious through the electrical system. Details and technicalities are provided throughout this document reporting on all tests, statuses, issues and implementations. In the conclusive stages of this project, the launcher was returned, to the SSRS, functioning with its new implementations. As it is a prototype, it is still some way away from seamless operations due to its powering solution and complex software interface but solving this issues falls outside of this particular thesis's scope.

Keywords: Drone Launcher, python

CONTENTS

List of Figures

Listings

PREFACE

We, Abdulrahman Alsamel and Quentin Demory chose to take on the drone launcher project and write our thesis about it as it seemed very exciting to us. The work is carried out on a prototype which offers a wide variety of implementations. It makes use of both electrical hardware and software in a way that is very relevant to our acquired skill set in the Electrical Engineering Bachelor's program at Chalmers University of Technology but at the same time presents many challenges that we had not previously been exposed to.

In writing this document, we have tried to shape our writing to help superseding project workers find the information needed quickly. We shaped the section and subsection with care so the reader can target its reading per topic. Along this document, we cover hardware and software implementations that are relevant to this project. We have not, however, covered it in details should the documentation be easily available or redundant from previous thesis work. Relevant web pages and documents are naturally made available in the reference section of this document.

Some of the challenges we faced were that we had not previously worked with some of the software and hardware components of the launcher but this turned out to be a great learning experience that we can take with us into the professional world. The corona virus outbreak also happened whilst carrying out this project but we feel that it did not impact the quality of this document although it has had some impact on external delivery times, leaving us with less time, than we would have wished, for system testing. We would like to thank Infotiv for letting us carry out this project on their behalf but also for their support and professionalism regarding follow-ups and hitting targets. We would like to specifically thank Niels Jonsson, our supervisor at Infotiv, for his support, encouragement and enthusiasm sharing his learnings with us and Nils Gangby, a fellow thesis student at Infotiv, who always made time for us and our varied software related topics. Lastly, we would like to thank Göran Hult, our examiner at Chalmers, for his help and support along the way.

We would also like to mention and thank Mattias Preuss and Edward Bergen, who built the drone's case, for their camaraderie and support and, Daniel Valero Beltrá who left a great project behind him, for us to work on, and made every effort to answer the many questions we had.

Abdulrahman Alsamel & Quentin Demory

Acronyms

CSS Cascading Style Sheets. 4, 12

EEPROM Electrically erasable programmable read-only memory. 3, 7, 17, 26

GPIO General Purpose Input Output. 3, 17

GUI Graphic User Interface. 2, 4, 9, 10, 13, 16, 18, 19, 21, 24, 26–28

HAT Hardware Attached on Top. 3, 19

HTML Hypertext Markup Language. 4, 12

HW Hardware. 14

I^2C Inter-integrated circuit. 19

IPD Infotiv - Product Development. 2

LED Light Emitting Diodes. 2–4, 13, 17

PIV Proportional Integral Velocity feedback. 3

RGB Red Green Blue. 2, 3, 17

SSRS Swedish Sea Rescue Society. i, 1, 2, 5, 21

SW Software. 2, 28

1 Introduction

The Swedish Sea Rescue Society (SSRS) is a voluntary organisation for rescuing operations at sea. The Society is financed by membership fees, donations and voluntary work. The Society has doubled the number of sea rescue stations in recent years, tripled the number of rescue volunteers available and built 230 modern rescue vessels. This expansion has enabled the Swedish Sea Rescue Society to meet its goal of departing within 15 minutes or less from the time an alarm is received. Crews live close to stations and conduct training several times a month. The sea rescue volunteers are willing to go out in any weather, at any time even during normal work hours or in the middle of the night. The personnel at SSRS face a multitude of different scenarios while in the rescue operation which requires a different type of rescue gear and resources. A better understanding and overview of the accident would facilitate better resource management.

Throughout this document, we will be focusing on the development of the drone launcher, the drone itself being the subject of a separate thesis. The purpose for the robotic launcher is two fold. The drone, set to be expelled from said launcher is a fixed-wings drone. This means that it is designed like a small plane, it has a triangular shape with long wings and a DC motor propeller at the back. The launcher is set to propel the drone from its platform with enough momentum for the drone to switch on its propeller and increase speed through the air. Such a contraption means that the drone itself can save weight on take off/landing gear. Added to this purely functional attribute, the launcher, when unused, is set to stand as an advertisement platform to invigorate donation capital. The SSRS finances itself from memberships but people can also make donations, used to purchase equipment such as buying a boat for the SSRS. The donor, in return, gets their name written on the very boat. The drone/drone launcher would be financed in the same way but at a lower cost offering new donation alternatives.

1.1 Background

The SSRS has a department for innovation where they are working with finding solutions to ease the prerequisites for the volunteers when performing rescue operations. To help the volunteers to get a better overview of the scene of the accident SSRS have conducted a study through several thesis and internal projects resulting in an unfinished first prototype (drone and launch mechanism) to be the basis future improvements. The purpose of this document is describe the steps taken to ensure the delivery of a functioning and robust prototype, tested to work for operation from April to October, along with a document for the user, describing normal operation but also a more descriptive approach to the Software and Hardware specifications enabling maintenance and further optimizations. The launcher when it stands ready for testing will also need several new functionalities, parts to enhance marketing visibility but also to ensure it has some safety features to make testing obvious to onlookers. Currently, the launcher does not track weather conditions and thus, needs some protocol to retrieve such information if it is to operate in automatic launch mode as wind conditions are of concern for drone launches.

1.2 Problem description

The prototype is described in a previous Master thesis report from 2018, written by Daniel Valero Beltrá, a student from the Mechatronic department of Chalmers. This document has been very helpful

and Daniel himself has been very forthcoming in offering his help. However, there were no detailed coupling schemes available to verify the electrical soundness of the launcher and thus safety of use could not be established without research and investigation. the hardware on the prototype presents other challenges as the department of Electrical Engineering does not offer courses specifically using Raspberry Pis or Motor controllers but the web-based community for this products' use and implementation is very active.
The Launcher is powered by a battery capable of delivering 100 A which safe handling and using also represented a learning curve.

1.3 General aim

The prototype should be designed to operate from April 1 to October 1, meaning that the drone will be verified to be operable in its manual setting. This would confirm good operation of the software and hardware states.
The motors operating the case open/close function will be implemented in the SW design. This process relies on another team, namely SSRS – IPD, and we will start implementing shaft/motor operation when they have defined specifications.
The prototype will be equipped with LED to provide back lighting for marketing and security purposes. We intend to use RGB LEDs so they can be customized to change color and act as warning when the launcher is in motion. This will be fitted in concurrence with a sound alarm to increase safety of usage. Temperature sensors for in/out temperature will be fitted to the Raspberry Pi to monitor potential damages due to out-of-range temperatures.
The prototype will be equipped with a 4G module. In the scope of our project, we require this module so that the embedded computer can collect weather information from SSRS's weather station. A measure of this would be presented on a GUI. The prototype will be equipped with a gyroscope/compass so the SW can resolve initial position and encoder/final position enabling it for a weighted launch.
The prototype needs some battery status monitoring and critical state feedback, as a launch under sub-critical battery reserves could cause damages to the system. We believe some solution to this might be readily applicable at Infotiv. This could be implemented on the same Raspberry Pi module used for the 4G application, giving us a flexible prototype before it gets assigned to the launcher.
As the Launcher may develop to be a finished product and as it has varied applications beyond its scope, a descriptive and detailed document will be generated to offer clarification, instruction and potential modulation of the SW/HW. We will refer to it in this text as "User Manual" Such a document is not currently available, and we feel it is important that the users have such a document, enabling them to correctly use the unit but also, have the right resources to maintenance it. On our end, this document is critical for testing which is necessary if the launcher is to grow out of its prototype phase.

1.4 Method / Outline

The material that was necessary for research and for implementation as some of the hardware, like sensors, was to be permanently added to the already existing one. the list of items is as follows:

HARDWARE

- 2× **Raspberry Pi**: The launcher operative system and server is run by this compact computer, specifically the Raspberry Pi 3 B+.

- $2\times$ **Roboclaw motor controllers** $2 \times 30A$: the original launcher is already equipped with two Roboclaw but two extra units were available for training and testing. One of these two spare units will be fitted to the permanent hardware as part of this thesis scope to open and close the launching platform's protective casing. Some of the features of the Roboclaw besides motor controls (speed, forward, backward, acceleration, PIV tuning) are current level sensors, temperature sensors, voltage clamping, interactive error reporting.
- **Globe Motors 537A354 DC Motor HEDL-5505 A06 30.3V**: Motor used for testing Roboclaws and concurrent operations before implementations on the Launcher.
- **Sense HAT**: This hardware sits on the Raspberry Pi's 40 GPIO breakout connection and is equipped with temperature sensor, humidity sensor, pressure sensor, gyroscope, accelerometer, compass, magnetometer and a 8×8 LED matrix.
- **Gyroscope, accelerometer MPU-92/65**: An alternative source of data for geo-positioning used for training purposes but also as secondary option for the real time launcher positioning.
- **Addressable RGB LED strip, 1m** ($60\times$ **LEDs) type WS2812**: Lightning designed to provide enough light to render the different states of the launcher clear to any user during testing or general use.
- **One-wire water resistant temperature sensor DS18B20**: sensor aimed at collecting temperature data from inside the launching platform casing.
- **HUAWEI 4G Dongle E3372**: Unit aimed at updating the launcher GUI from Local Network operation to Internet based network.
- **2** $\times$ **Step down "Buck" converters**: Used to take the battery supplied voltage from 24V to the required voltage for the light strip and sound horn.
- **Experiment Board, 3-links**: Board for soldering, aimed at creating robust module attached to the Raspberry Pi.
- **Oscilloscope Tektronix TDS 1001B**: For signal analysis; Mainly of motor encoder signals.
- **Others**:
  - Digital Multi meter: Controls
  - Voltage source: Testing
  - Resistors: Circuitry
  - Jumper wires: Circuitry
  - 40 GPIO break out flat cable: Circuitry
  - Extra long male connectors: Break out from HAT - Circuitry

SOFTWARE

- **BasicMicro Motion Studio**: Motor controller tuning. Read / Write EEPROM with tuning tests, encoder data and PIV settings before incorporating these data in python code.
- **Linux - Debian**: Raspberry Pi Operating System

PROGRAMMING LANGUAGES

- **python 2.7 & 3.7-8**: The former version has been used on the Launcher software. All operations will be updated to python 3.7-8 in this project. Libraries used in this project:
  - Roboclaw: Provided by BASICMICRO, makers of the controller.
  - Flask: Back-end application building
  - JSON: Used to transmit packages between the python code and the GUI
  - time: Timing functions
  - socket: Low-level networking interface
  - threading: parallel programming
  - neopixels: addressable LEDs
- **HTML**: GUI implementations, Web page rendering.
- **CSS**: GUI implementations, object formatting, style rendering.
- **JavaSript**: GUI implementations, dynamic rendering.

2 Research and preparation

In this section is described the state of the launcher as we took over the project. What were the knowns and unknowns and what was done to overcome these steps to take the project into its implementation phase.

2.1 Initial state

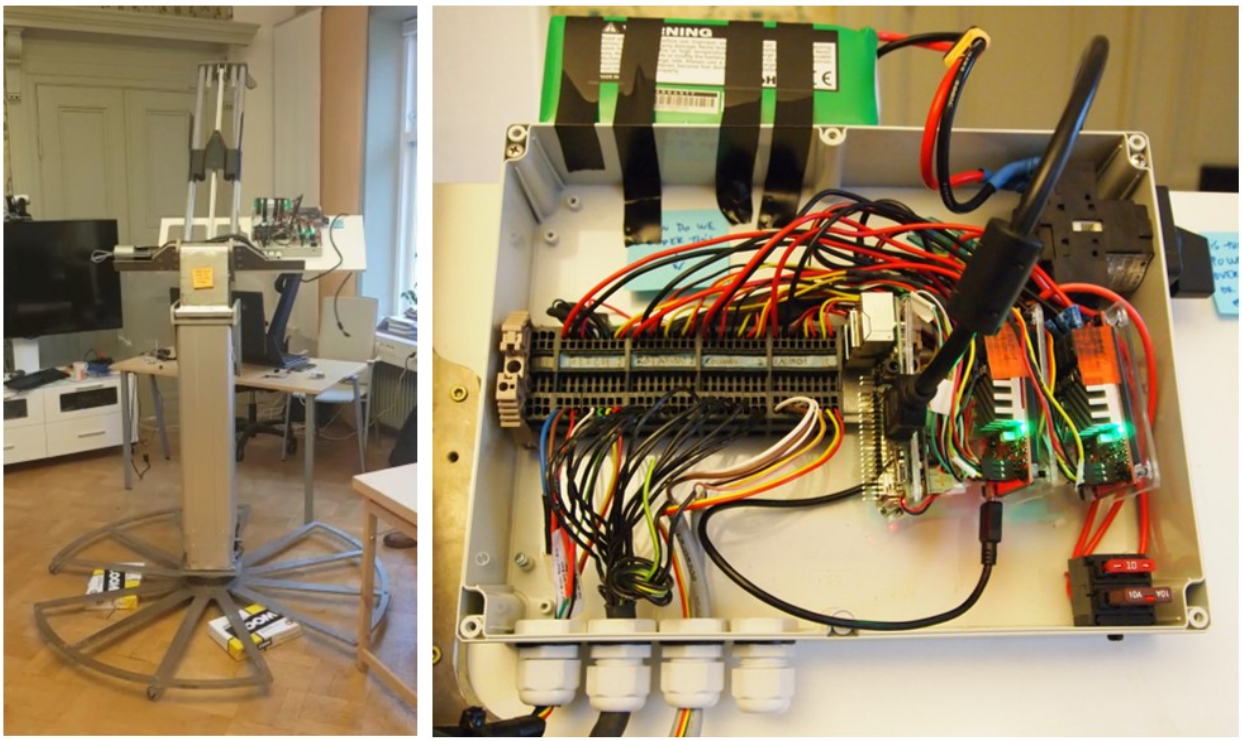


Figure 2.1: Initial state of the launcher robot and its electrical box

During the first meeting, at the parent company Infotiv, an introduction was made stating which documentation was available regarding the Swedish Sea Rescue Society (SSRS) drone launcher. This was a Master's thesis of a Chalmers mechatronic student responsible for most of the launcher's assembly. The physical launcher was available, as seen in figure 2.1, and so a quick overlook was carried out. It could be seen, at a glance, that the hardware was built with a Raspberry Pi at its core. Neither the authors of this documents had worked previously with Raspberry Pi. However, this was quickly overcome as Infotiv had training packages available in house. During this fist meeting, the project leader established working standards that were to be in line with larger companies standards and thus a detailed Project Description document was created as well as a Time Plan and a Risk assessment. To be planned also were five "Gates" which are meetings with management in which progress, achievement and next steps are discussed and the Gate approved should the outcome be satisfactory.

2.2 Latest Master's thesis

Daniel Valero Beltrá's report, "Design and mechatronic integration of a drone launcher" is a very detailed report on the construction of the launcher. A lot was learnt about the hardware going through it such as how many motors are currently operable in the launcher (as seen in figure 2.2), namely:

- the PITCH actuator: Changes the launching platform angle.
- the ROTATION motor: Engaged with gears and rotates the whole platform left or right.
- the COLUMN motor: Raises the platform high enough to remove any human interaction risks on launch.
- the LAUNCH motor: Actions the belt that launches the drone.

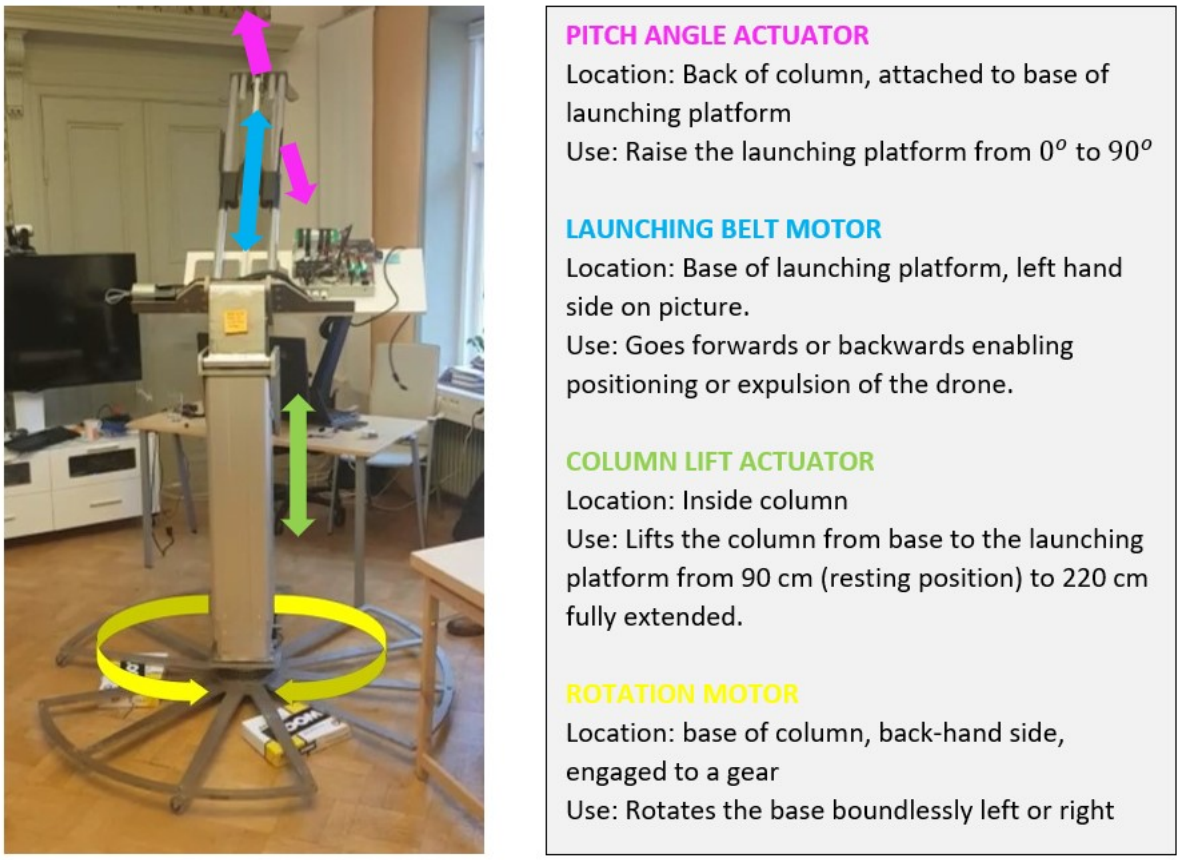


Figure 2.2: Visual overview of motor functionality

A breakdown of each motors peak and stall currents is also described in details and meant that operating the launcher was safe from an electrical power point-of-vue. A chapter of the thesis is aimed at electrical hardware and its software. We learnt that two motor controllers (Roboclaws) were needed to run the four motors incorporated in the launcher. We saw some basic schematics of how the system was coupled and we were walked through some of the code and it's functions.

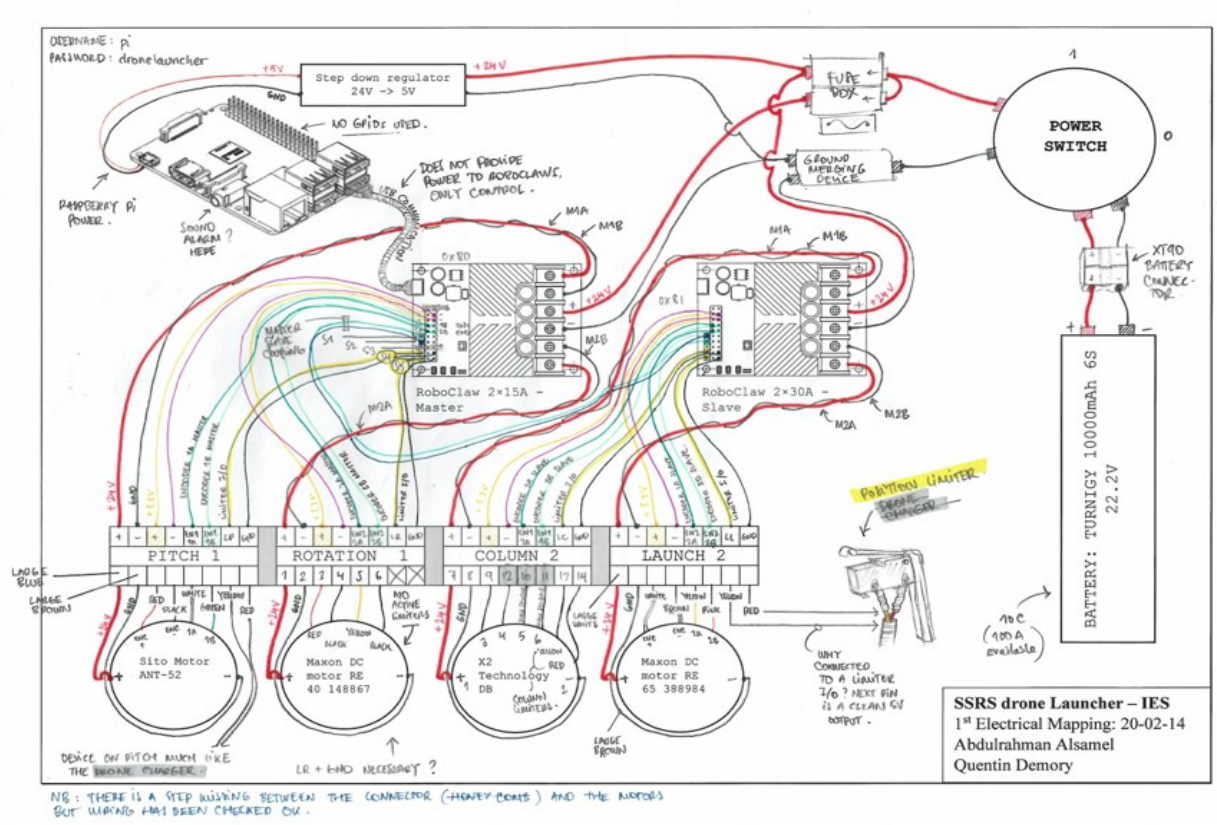


Figure 2.3: Completed electrical mapping

2.3 Electrical Mapping

The first task we set for this thesis was to create a comprehensive map for the electrical coupling of the launcher as it is not advisable to send energy into an electrical circuit without knowing that it is safely coupled. To do this we set ourselves to do an exhaustive drawing representing each components and their coupling to one another. as seen in figure 2.3, the system seemed to be correctly coupled and we were able to gain a greater understanding as to the hierarchy of the system.

2.4 BasicMicro - Roboclaw

The two motor controllers running the system are Roboclaws, we cover them here in a little more detail as they are a crucial part of the hardware. They hold the encoder communications between the Raspberry Pi and the motors. One further Roboclaw will be added within the scope of this project to open and close the launching platform casing and so closely understanding its functionality is paramount.

2.4.1 Roboclaw hardware

The Roboclaw motor controller is packed with intelligent hardware and can be configured in many ways. For this project they are configured to communicate in serie, as master/slave. the master Roboclaw interacts with the Raspberry Pi via USB. The Roboclaw, seen in figure 2.4, is packed with current level sensor, temperature sensor, voltage controls for each channel and has an EEPROM

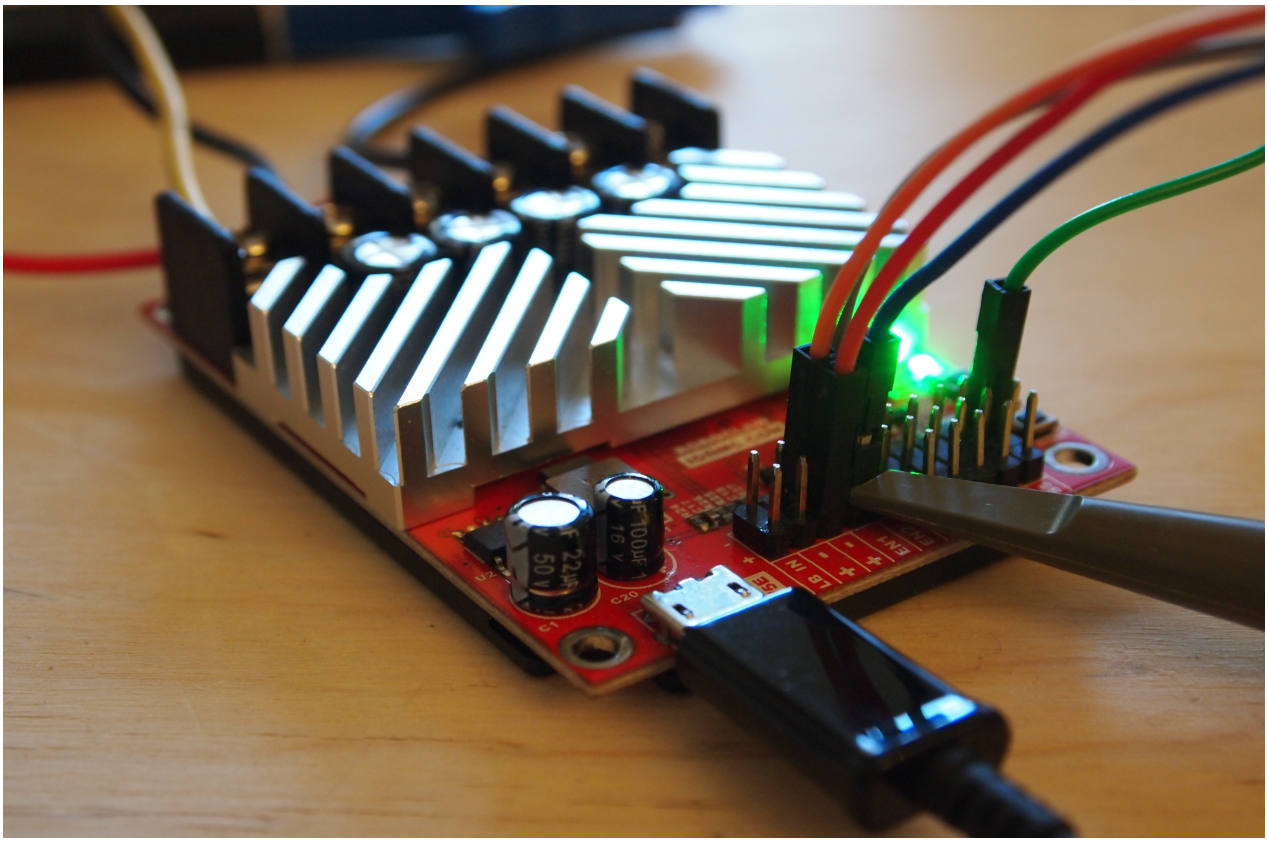


Figure 2.4: Roboclaw

available to read/write configuration data which are tuned using the BasicMicro Motion Studio software. They have a "Logics" side (3.3V) for communication between units and encoders. the encoders are set to values between 0 and 127, 7-bits, to allow for reverse driving (negative values). An example of their signals are shown in figure 2.5. The other side (24V) is for voltage response to the motors regulated by Pulse Width Modulation (PWM) as seen in figure 2.6. This side is coupled to large capacitors and cooling equipment to handle large peak currents. Roboclaws are designed to handle a set amount of current on each channel but can also use both channels to handle up to the sum of their current capacity. For example, a 2×15A Roboclaw can handle up to 30A at peak and a 2×30A can handle 60A at peak. Which is well within the range of our motors. Using the manual provided and looking at the Electrical Mapping we had created we were able to declare the system safe to boot as long as hardware was involved. We could not establish proper coupling of the PITCH actuator - and we will come back to that later in this text - as the documentation was missing and the encoder is tightly encapsulated. However, we knew the right voltages levels were going in the right place and there was no reason - at this stage - to think that they would not be coupled correctly.

2.4.2 BasicMicro Motion Studio

The tuning software is well described in the Roboclaw manuals. In figure 2.7 we see the general interface when one wants to tune or analyse response of a Roboclaw. We will not go into details of this interface here but it is relevant to note that the python code that is used to manipulate the Roboclaws can give us access to all the data that we can get access to on this polished interface, battery levels included.

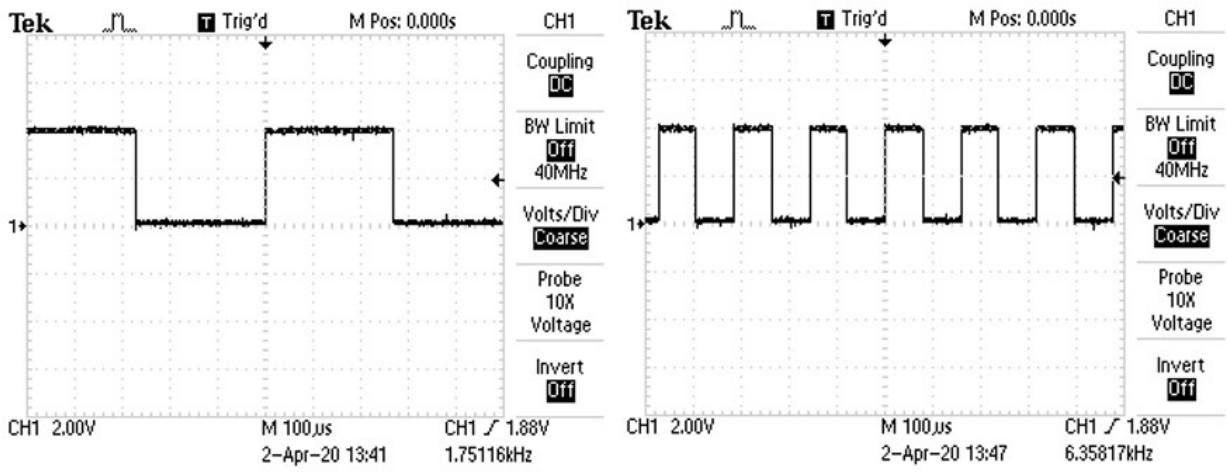


Figure 2.5: Encoder signals at two different speed

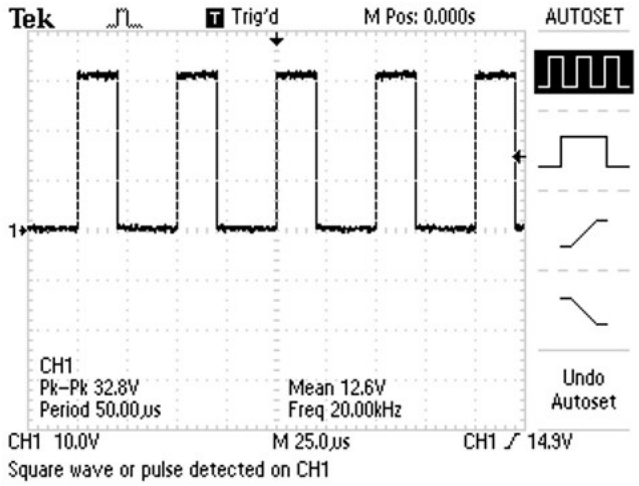


Figure 2.6: Speed alteration to a motor using PWM

2.5 Code assimilation

Setting to work with the codes proved to be quite involved. The codes were long and spread across several languages. As we both had some experience in coding, one of us specifically in python, we were able to take ourselves through the main code and observed the divide and conquer approach through our decoding stage. We were able to identify that some of these codes were issued by BasicMicro as they were a large class description for the Motion Studio functions. We also found that the HTML, CSS and JavaSript codes all related to the GUI interface and so we could put that to the side for later. In so doing we were left with one program `dronelauncher_python.py` and, still using the divide and conquer approach we set to break it down until we could make sense of each functions interaction with the hardware.

2.5.1 Launcher's hardware response coding

The python code as written by Daniel Valero Beltrá, `dronelauncher_python.py` is split in four parts. First we have all of the variables as obtained from measurements, as the column's length for

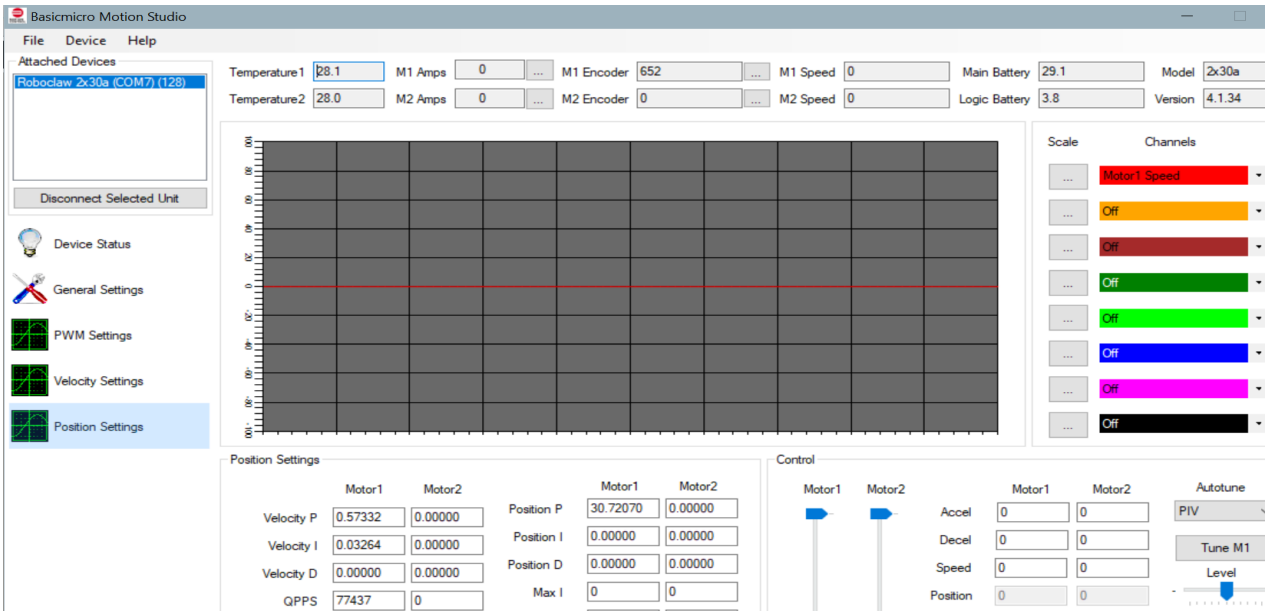


Figure 2.7: ION studio interface for Roboclaw test tuning

example or data collected from BasicMicro Motion Studio, such as the encoder variables. This is followed by Manual and encoder regulated commands for each of the motors. We should note at this point that each function starts with an `app.route()` syntax and that is a Flask specific syntax that we will overview in the following subsection. Next come the Automated functions which follow encoder regulated commands to reach preset position variables (set in the variables section). Last come variable updater functions to customise the Automated function variables for position, speed, acceleration without having to specifically change the variables at the top of the code.

2.5.2 Launcher's server protocol

Wireless communications with the Launcher is done with the Raspberry Pi's wifi integrated module by connecting to it using its IP address within a local network. Once the connection established the control panel, GUI can be reached to operate the launcher. Making this happen - opening a local web server and transferring commands remotely to the launcher - requires the use of the Flask module in python. We will not explain the functionality of Flask here, as it falls outside of the scope of this document, but only explain that the `app.route()` syntax in the code means that there is a transfer of information when it is called. In simple terms if you press a button on the GUI you are calling/taking a different "route" and that route is bound to a specific function. If we take the example of the pitch up function (line 148-162 in Annex A). If one was to press the up button on the GUI, it would activate the `app.route('app_pitch_up', methods=['POST'])` which is bound to the function `def function_pitch_up():`. The 'POST' method means that the user can interact with the data. Figure E.1 & E.2 in Annex E are a reference to help grasp the inner workings of the GUI

2.5.3 Launcher's GUI coding

To display the GUI, seen in figure 2.8, and interact with it, some other languages had to be involved. Following will be a succinct description of what those are and what they do. We make a point of presenting those as some new implementations will be implemented on the GUI later, using those programming tools. It should be noted here that the GUI only activates on the local network with the

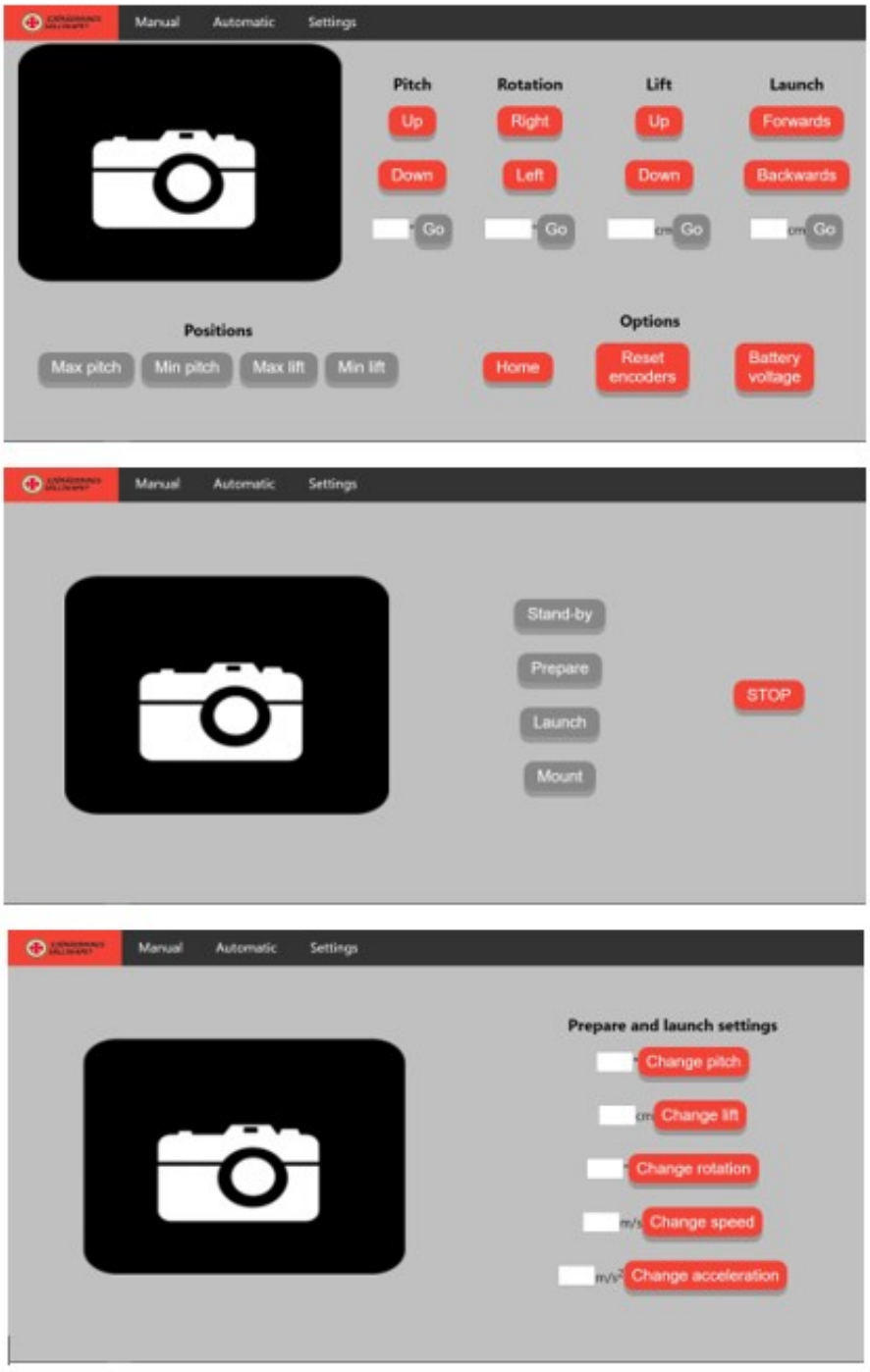


Figure 2.8: Original GUI

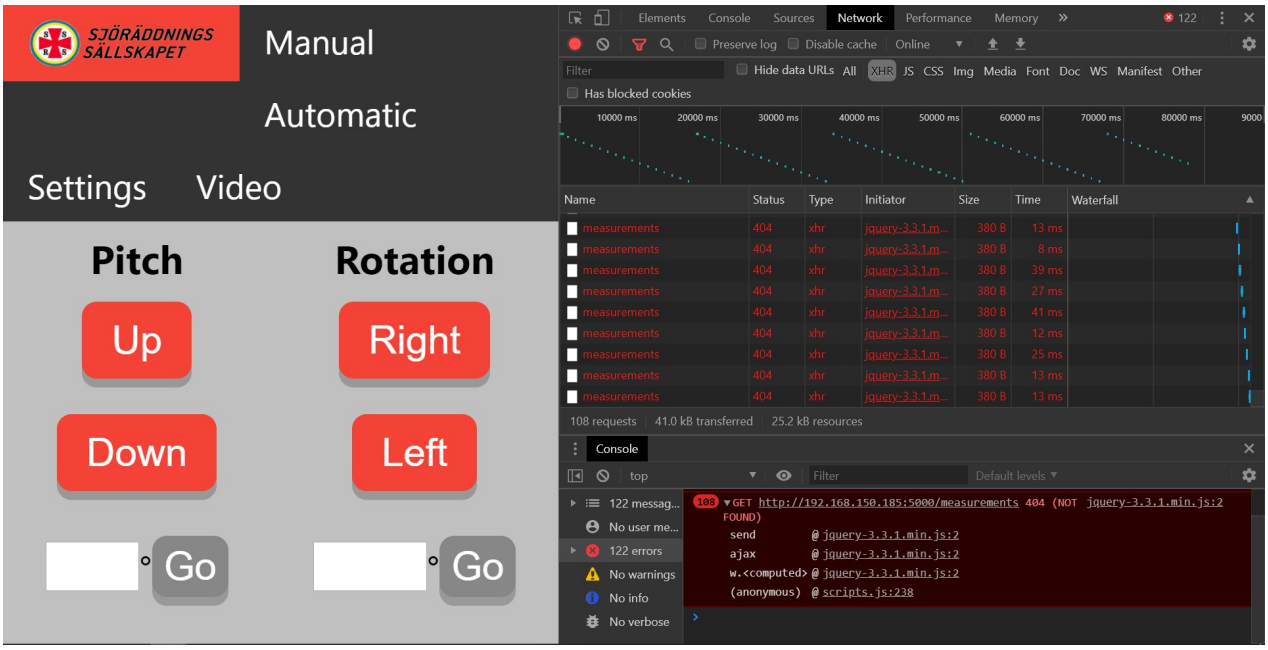


Figure 2.9: Debugging on failed temperature measurements from a browser

motor initiating lines #80 & #81 are commented off. Otherwise, the code will look for motors that are not activated (coupled to energy) and will therefore not find them and throw an error on boot. It is completely fine to start the Raspberry Pi on local power (mini-USB), then comment out the lines and run the program but it will boot in that configuration next time it is switched off and on again as the program automatically starts during the Raspberry Pi boot process. Using motors (when the battery is coupled to the launcher), you must make sure these lines are reactivated otherwise the GUI will appear to function properly but the motors will not receive any commands.

HTML

Hypertext Markup Language is the front end script, it is used to format how a web page will look and what its properties will be. Typically, one can get a glimpse of HTML syntax by right clicking on a page and choosing "view page source" on the pop up panel.

CSS

Cascading Style Sheets is a style formatting language which supports the HTML with colors, buttons and styles. It creates style objects that can be called into the HTML instead of exhaustively writing each styled components.

JavaScript

JavaScript enable the use of functions inside the HTML. In that way the computer running a web page supported by JavaScript can do some of the computation direct from the Users computer, reducing computations on the server side and therefore limiting the strain on communication between the server and the user. This enables the developer also to have access to some powerful debugging tools on the web-page itself, as seen in figure 2.9. This interface can be accessed by right clicking on a web-page and selecting "inspect"

2.6 First test - Controlled environment

As the Hardware and Software had been fully investigated, and there were no hazards highlighted. The power was put on to run some tests with the launcher and its GUI. Those test were successful but some issues were identified. At this stage, the successes included:

- Fully responding manual operations of all the motors, pitch, rotation, column lift and launch.
- Pressing two manual buttons on the GUI would make two motors move simultaneously. i.e: one could initiate a "Max lift" routine while rotating the launcher.

2.7 What issues were identified

2.7.1 python 2.7

Early in this project, when the codes running the launcher had been made available, it was identified that it was written in python 2.7. This was immediately logged as an issue as, going forward, python 2.7 is no longer maintained by the python organization. The new modules set to be added to the launcher all are developed using python 3 and therefore present an integrating issue.

2.7.2 Malfunctioning pitch angle

During tests all manual function were functioning as expected except the pitch position function. One enters the desired pitch angle, say 45^o and the launcher should go from its current position to that entered. Instead the pitch motor only went from 90^o to 0^o, so its maximum to minimum position without stopping at any values in between, no matter what the user would enter. This function ran successfully for the other three motors.

2.7.3 Current overflow between rotation and pitch

During a demonstration for management at Infotiv, the launcher was put through a stress test and failed when some motion was still detected on one of the motors, specifically the rotation motor in what was believed to be a current overflow between the pitch and the rotation (the two motors being used at the time of the test) as these two have their operation on the same Roboclaw unit. This, in turn highlighted the fact that there was no emergency stop button on the manual screen.

2.8 Column rotation freely movable

During the first assessments of the launcher hardware, it was identified that one can manually rotate the launcher without applying much force. This became a worry instantly when LEDs were lighting up on the Raspberry Pi and the Roboclaw units in the electrical box. This obviously went up at once on our risk assessment chart and was reported to the management and the launcher owner as the launcher is intended to be fixed outside for testing in areas exposed to winds. Motors made to rotate manually act as generators, creating energy which, as opposed to a motor, can not use this energy to move a load. Instead the electricity created races back in the system trying to find some way to dissipate. the LEDs in the electrical box would be one place and with small, slow motions this is not a problem but exposed to significant winds, it is hard to predict how much energy can be generated

and what those levels could do to the hardware. This issue was highlighted generously as creating some mechanical stop for this process falls outside of the scope of this particular project.

2.9 Electrical box and water protection

The electrical box that came along with the system was a small box $300 \times 220 \times 85$ mm that contained neatly the computing hardware. It was, however, too small to fit the battery pack in properly. It reported as a risk to the management and the owner as the scope of this project is to add new functionalities to the launcher, meaning that more hardware space would be required. The location of the electrical box, its needs for water proofing and its maximum size were all unbeknown. A request for a $600 \times 250 \times 85$ mm waterproof box has been made through the Infotiv channels but a box of this dimensions could not be found. The solution to this issue was resolved by ordering a second box of the same dimensions as the one already in use and fitting the HW through two boxes taped to the back plate of the launcher so it can be moved elsewhere in the future.

2.10 Removing blockers

2.10.1 python 3.7

First and foremost all the python 2.7 codes were moved to python 3.7. The makers of the Roboclaw have recently made available a python 3 release of their library (something that was not available when the codes were first written). It was then possible to adapt the `dronelauncher_python.py` to the python 3 library and start testing with it. Some of the neatness of the launcher's Raspberry Pi is that it starts the launcher script on boot. That way when the launcher is switched on, it is automatically ready to proceed with launcher specific execution. The move to python 3 stopped that particular feature but after some research it was identified that the shell script (`dronelauncher_python.sh`)that calls to the `dronelauncher_python.py` needed to be updated so that it knew that it should now be a python 3 file that should be passed to the interpreter. This routine was implemented on a second Raspberry Pi to test for robustness and it passed all the test required for us to guarantee that it is performing.

2.10.2 Pitch testing

Some of the difficulty with fixing this particular issue and the reason why we will be talking a lot about a "test Raspberry Pi", as we have done in the previous section, is that soon after our first testing, we lost access to the launcher as the team making its platform casing needed to take it apart. This meant that we were not able to test the signals sent by the encoders with an oscilloscope and this issue remained until late in the project. A code `dronify1.1.py`, listing C.1, was built to test if the pitch position issue was generated by some software issue and give us numerical values for what output should be expected when testing with an oscilloscope. It is built by setting specific exception handlers to show what the commands would be to the motors. This code proved that all four position functions were identical but properly tuned for each motor specifications and thus ruled out any software malfunction and meant the bug causing this unexpected behaviour is of an electrical nature. An example of the digital research performed on the pitch is presented below, in listing 2.1, in the form of console responses:

```
In [1]: runfile('dronify1.1.py', wdir='')
```

```
<bound method Roboclaw.ForwardM1 of <roboclaw_3.Roboclaw object at 0
    x0000022C99B61160>>
128 129 130

In [2]: pitch.up()
<bound method Roboclaw.ForwardM1 of <roboclaw_3.Roboclaw object at 0
    x0000022C99B61160>>

In [3]: rotation.up()
<bound method Roboclaw.ForwardM2 of <roboclaw_3.Roboclaw object at 0
    x0000022C99B61160>>

In [4]: caseR.down()
<bound method Roboclaw.BackwardM2 of <roboclaw_3.Roboclaw object at 0
    x0000022C99B61160>>

In [5]: pitch.address
Out[5]: 128

In [6]: pitch.position()

which position do you want to reach: 0
starting point: 44.99492957746479 degrees
I go -177480 steps
128, 7000, -177480

In [7]: pitch.position()

which position do you want to reach: 90
objective = 355000
starting point: 44.99492957746479 degrees
I go 177520 steps
128, 7000, 177520

In [8]: pitch.position()

which position do you want to reach: 55
objective = 216944
starting point: 44.99492957746479 degrees
I go 39464 steps
128, 7000, 39464

In [9]: pitch.position()

which position do you want to reach: 46
objective = 181444
starting point: 44.99492957746479 degrees
I go 3964 steps
128, 7000, 3964

In [10]:
```

Listing 2.1: Return values on adequate methods and encoder commands in IPython kernel

In listing 2.1, it is visible that the response for Forward and Backward of the Roboclaws happens on the right channels according to their physical coupling. Considering the `pitch.position()` results, we have a minimum raw encoder position of 0 (pulses) and a maximum raw position of 355000

(pulses) which translate to 0^o and 90^o. We purposefully set the starting position at 45^o, 177480 pulses to show that the commands relative to it would have the right sign if one tries to reach a lower or higher destination. 1 degree in this set up is then $355000/90 \approx 3944$ pulses. That result is observable in the last `pitch.position()` try, effectively ruling out a digital error.

2.10.3 Stop all button on GUI

The issue concerning a motor continuing its motion although it was meant to stop is a serious safety issue. The way the GUI is set to work, from the JavaScript commands, is that when pressing continuously a manual button 'up' (for example), the motor should be proceeding with a motion order but if you release the button, it should instantly trigger a 'stop' order. This failed once and so we implemented a STOP ALL button on the manual interface, as seen in figure D.1, to make sure that if all else fails the user can still stop motors when required, reducing the safety risks significantly. So far we are unable to provide a sensible explanation for this behaviour. However, we have stress tested a motor we had available and found we were able to trigger that bug by pressing on and off the 'up' (or 'down') button several times very quickly. This rules out our earlier claim that an overflow would have occurred between two motors on a Roboclaw, as was thought to be the case when the bug was first discovered. Instead we now suspect that it may be a timing issue with very quick orders. The time required for the order to stop to settle must be violated by the next "on" order setting a logic high that can only be interrupted by a actual stop order. We believe that some sleep time after any stop order should be implemented to allow signals to stabilize before they are triggered again. We believe it to be so as the communication packages sent by the Raspberry Pi are relayed by FIFO units.

3 New modules implementation

At this stage, the launcher has been tested to work, with some issues to attend and the work on implementing the other deliverable can start. The original code already had an implementation for battery level control and the Roboclaws are set as part of their EEPROM settings to no longer operate if the batteries present less than 18V making the system safe from critical damage. It is recommended by our predecessor and again by us to recharge when the battery reaches 20.2V to insure all hardware are getting enough energy to perform. This is more critical now than it was before as the Raspberry Pi will be fitted with new functionalities coming out of its GPIOs meaning that a reliable supply of energy is adequate. This also means that every solution presented here has been developed trying to use the least amount of pins on the Raspberry Pi to keep its power stresses to a minimum.

3.1 LED strip

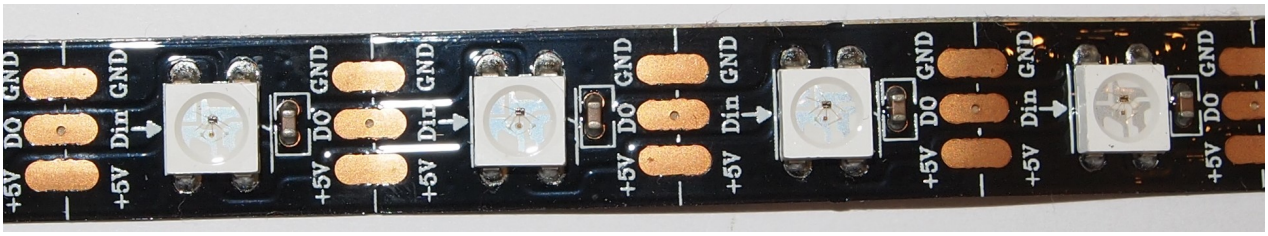


Figure 3.1: Strip data and voltage relay

The LED strip used in this project is an addressable 1 meter equipped with 60 × RGB LEDs. This strip is addressable using the WS2812 library for python and requires 5V, ground and in-data to control the LEDs. Coding for the light strip was done following "Connect and Control WS2812 RGB LED Strips via Raspberry Pi". Tests could be ran to make sure brightness and commands were responding well. The code used remained a test code but the strip commands embedded in the `dronelauncher_python.py` only have two states:

- Solid WHITE light: whenever no motion is detected.
- Solid RED light: whenever motion is detected.

A solid red light for motion was not deemed enough to draw awareness to the launcher and so a blinking solution was preferred. To do that a relay was used between the voltage source and the light strip. The relay takes 5V, ground and a signal, here set at PIN 24, from the Raspberry Pi. With this configuration, we can send in the data for color changes from the Raspberry Pi (on PIN 12, PWM enabled PIN) and also use the relay to turn the strip on and off.

3.2 Sound

When in motion the launcher is set to emit a loud beeping sound to warn any bystander. Making the beeping sound was done using the same routine as the blinking light and relay. A second relay was brought in and the blinking was assigned to PIN 23 on the GPIO header. The horn used is a DL129 car horn sounding with 105 dB. This horn is set to sound its loudest with 12V and 3A fed to it. However, delivering 5V and 1A was found to be loud enough to be acceptable. The sound module

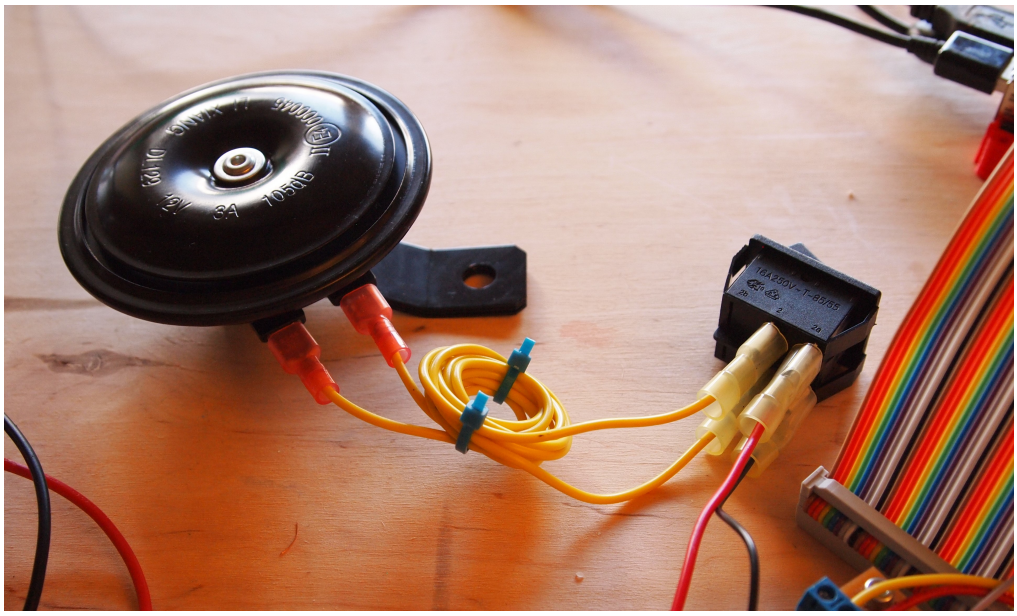


Figure 3.2: Launcher's horn and on/off switch

should be used at all times when operating the system outdoors but a switch was installed to override the sound during indoor use as the blinking light and the noise generated by the motors are loud enough to adequately warn for motion. As the project's test bench was set indoors, we controlled the voltage delivery through an oscilloscope, as seen in figure 3.3 to keep an eye out for issues.

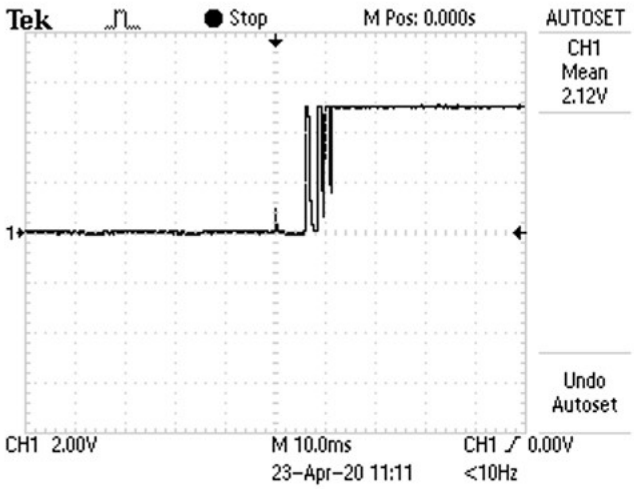


Figure 3.3: Controlling signals sent to the horn in real time

3.3 Thermometer

The launcher is set to be positioned outside during the summer. With the influence of the direct sun and what it could mean to the hardware, we have fitted three different types of thermometers pulling data to the GUI. More thermometer monitoring can occur as the Roboclaws each have a thermometer sensor on board, which would become relevant to use should the temperature be found to be sensitive

for the system as more knowledge is acquired during prolonged outside use (something that falls outside of this project's scope).
The three thermometers currently in function are the following:

- *Packaged CPU thermometer sensor*
  Monitoring the CPU temperature is important as the Raspberry Pi should not be let to work if the CPU temperature exceed 85^o C. The CPU is not subject to quite enough computational stresses to reach this temperature on its own. Indoor testing sees its average temperature at 55^o C. It is still necessary to keep this data available as no data is available for outdoor use. Coding to report on CPU was done as per Listing B.1. Raspberry Pis come with a sensor embedded in the CPU unit and the python code calls to it for measurements when prompted to do so by the JavaScript algorithm on the GUI end. This is the way in which each of the measurements are reported to the GUI so that computing load is off-loaded slightly to the users computer.

- *Water resistant thermometer type DS18B20 with 1-wire communication*
  The case developed to contain the launching platform and the drone is made of transparent plastic material with a metal alloy frame. The drone, as it has been reported, should not remain in the case above some temperature. A water resistant thermometer has been added to the launcher specifically to monitor temperatures inside the case. To do this we used a 1-wire communication process which can be implemented on Raspberry Pi as part of its configuration as indicated in "RASPBERRY PI DS18B20 TEMPERATURE SENSOR TUTORIAL", an online tutorial. The raw data returned by the sensor is hexadecimal and the code used calibrates this value to a simple decimal temperature in degrees.

- *Multiple Purpose Unit (MPU) 92/65 on-board temperature sensor*
  To monitor temperature inside the case we use the sensor available on the MPU 92/65 - Some more depth on the MPU 92/65 follows in the very next section. The temperature is retrieved as part of a bus communication between the Raspberry Pi and the MPU. Monitoring the electrical case is important as most components inside are rated against 21^o C and increased temperature within that case may mean for some changes in performance.

It should be noted that the roboclaws monitor their own temperatures as part of their operations and thus have their own securities in place should the temperature be found excessive for them. A reference for them is available on page 74 of "RoboClaw Series Brushed DC Motor Controllers - User Manual".

3.4 MPU 92/65 Gyroscope, Accelerometer, Magnetometer

During the research phase for a 9-axis device we quickly found a great cluster device called the Sense Hat, seen in figure 3.4 (left), which we acquired and mounted to the Raspberry Pi. This device was set to give us a lot of functionality with just one piece of hardware but, it covered all of the output pins and used too many of them to use other sensors along with it. The Sensor HAT was kept for prototyping, research and calibration but as new devices were being investigated, work was started on an Arduino device, the MPU 92/65, seen in figure 3.4 (right). We found resources, "Accelerometer, Gyroscope, and Magnetometer Analysis with Raspberry Pi Part 1: Basic Readings", to connect this device to the Raspberry Pi and get information from it. We, quickly, could get the right data from this device, which uses I^2C (bus) to communicate with its various components, as stated in "MPU-9250 Product Specification Revision 1.1". This makes it very "pin-economical" and thus the preferred

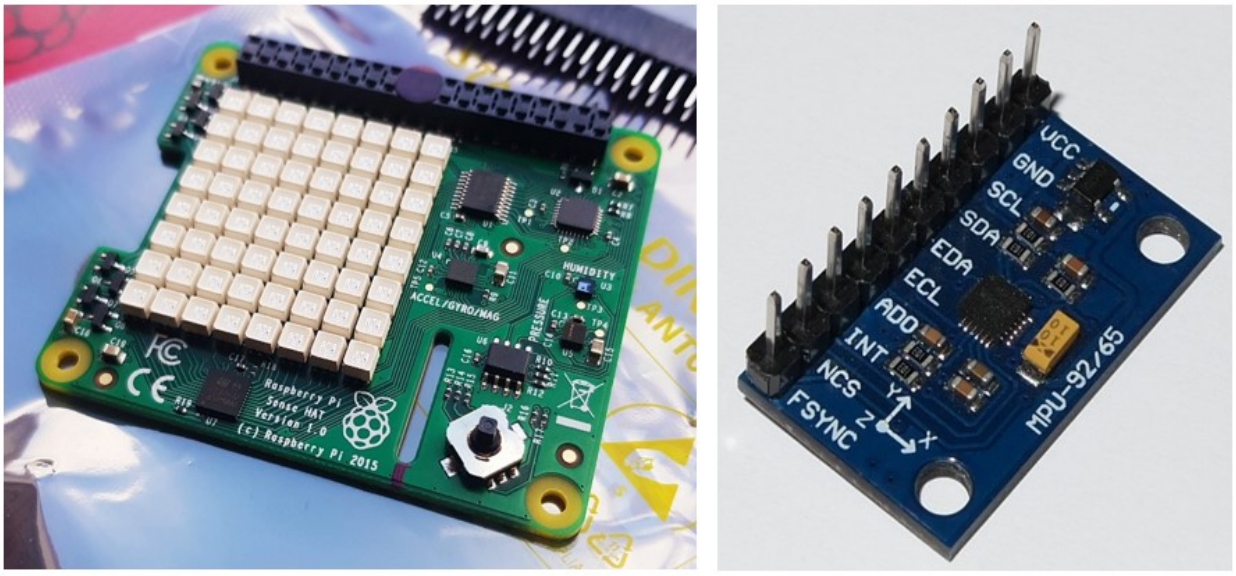


Figure 3.4: Raspberry Pi's Sense Hat and the MPU 92/65

choice. Tuning the device was not straight forward. The Gyroscope and Accelerometer send raw data that is easy to understand and scale to get 3-axis positions for each the sensors but the magnetometer needs more attention as it is influenced by all sources of metal and the launcher is entirely made of metal. During testing in a disturbance free area, as seen in figure 3.5 we got the lateral angle to function as expected. Time ran out to properly tune this vertical angle on the launcher but as the position of the Magnetometer relative to the launcher is always the same, a "hard iron" tuning - where one uses averages maximum and minimum data to re-calibrate the "0", rest position - should be ran.

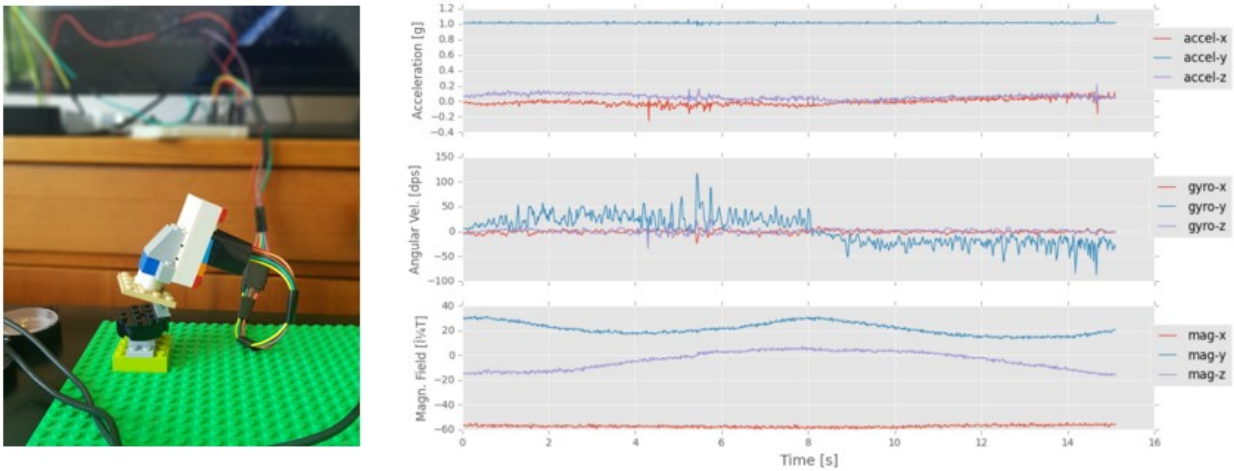


Figure 3.5: Mini launcher fitted with a MPU92/65 & data collected for calibration

3.5 GUI

A challenging problem that we kept coming back to in this project has been the GUI. It has slowed us down considerably in testing and prototyping. We have had to organize ourselves as Hardware

developer and Software developer to speed the process up as all of the processes we implemented here had to have a GUI representation for us to see that they worked in the larger code. We, for example, had to implement the case actuators and so, one of us sat and programmed and wired them to have an open and close routine to test, while the other installed and connected virtual buttons across 3 languages for us to actually run tests as they would be for the actual launcher.

3.6 From local network to internet access

The SSRS drone launcher's GUI, as it was handed out, could be accessed and used through local network. This means that if two computers are coupled to the same network and one of them runs a web-page accessible through a web browser, the other computer can connect and interact with that page. A computer outside of that particular network coverage would not be able to access such a link. As the launcher is set to stand outside of SSRS's network, the GUI had to be moved from a local network web application to a internet accessible web application. This was done using Ngrok, an application that creates a tunnel, to route the web application to the internet. When properly set up, as described in "Ngrok documentation", it was possible to operate the test area through the GUI from a remote location. Changing lights and data monitoring all work without any lag on tests. The second challenge was to launch a Ngrok link at boot. This was done following "Run a Program On Your Raspberry Pi At Startup" and was tuned to launch at boot but wait 10 seconds to activate to create the tunnel for the main code to launch 5 seconds later. This functionality as been thoroughly tested and verified. It should be noted that Ngrok requires and active membership to create a stable address (link), the free version generates random links at set up which is impossible to keep track off if the Raspberry Pi is ran "headless", as it is meant to be.

3.7 4G with HUAWEI E3372

With the web application reachable on the internet, the next challenge was not only that the user could reach the launcher from a remote location but that the launcher could itself be in a remote location. Such as it is when the launcher stands out of wifi range in Långedrag. To do this, a HUAWEI 4G Dongle was used for the launcher to deploy its own wifi access. It was installed, using "Huawei E3372 Product Description", and tested with the same results as previously obtained with the local network and remote access (internet).

To summarize, at this stage, when the launcher is switched on, it deploys its wifi network and posts the GUI to its Ngrok link ready for use.

3.8 Blinking light & beeping sound

Getting the light and sound to work was relatively straight forward. Having them blink and beep while the launcher is moving was a bigger issue that took time to solve. The problem with blinking lights (beeping sound have the exact same routine and so it will not be mentioned further in the rest of this text) is that they need a sleep routine. The launcher, however, can not have operation blocked by sleep statements. If a motor is active, we can not wait for a sleep statement to finish before we can activate the emergency stop for example. It is also necessary to write the routine for the blinking lights nested in a `while` loop as we can not set timing constraints in relation to motions and exhaustively writing light routines through the code would be counter productive and render the code impossible

to read. We knew that we needed for the blinking light routine to be listening for motion triggers and operate outside of the main code. During our research, we came across two python libraries that could do the trick, `threading` and `multiprocessing` (see “Python 3.7.7 documentation” for further reading). The `threading` library allows the programmer to create threads within the code that will "simulate" parallel programming by using some smart scheduling to run other operations during sleep statements. The `multiprocessing` library allows the programmer to run an entirely separate script in parallel to the main script. This means that we would need more than one multiprocessor core to run on the Raspberry Pi and, a Raspberry Pi has four of those. `multiprocessing` is useful for core intensive computations and speed, neither of which is relevant to our project so we chose to program further with the `threading` library at this stage. When trying to implement the blinking along with `threading` we could get the lights to blink as we had hoped but we were not able to make them stop as we could start our thread but we were unable to interact with it. At this stage, a short lived attempt to make an analog timer was started using a circuit with capacitor and resistances but the results were more "beaming" than blinking as the capacitors discharged and that did not have the right effect. the `threading` library was picked up again and we searched deeper until we came across a sub-library called `Event` which seemed to set and listen to Boolean flags anywhere in the running threads (including the main thread). When this routine was implemented, we were able to operate and stop motors while a test LED was switching on and off during operation, the test code capturing this functionality can be found in the Appendix C.2. Obviously the LED Strip need 5V feed steady that can not be provided by the Raspberry Pi and the horn needs 12V for maximum output. What we did though is to bring the two circuits already built and install relays between the source and load so that our LED `threading` routine could just become two separate relay `threading` routines on pin 23 and 24 demanding no more power that the Raspberry Pi can provide. the threading routine also changes the light settings to red during motion and back to white when not in motion. The case thermometer also uses a `threading` routine as it also uses a closed while loop and sleep routine to make its measurements.

3.9 Launching platform case actuators

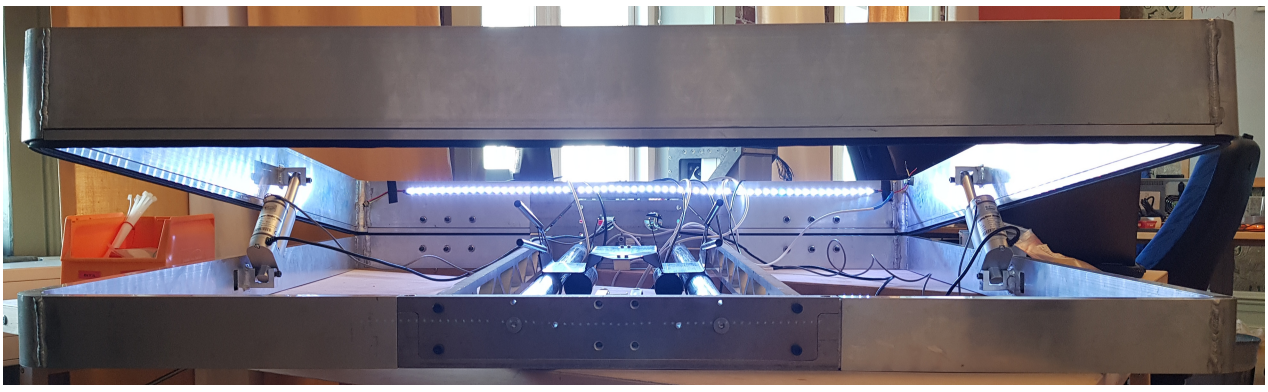


Figure 3.6: Open case and actuators

The actuators ST1500 operating the case open and close functions, seen on figure 3.6, arrived on our bench at the end week 15. At this stage setting up motors on ION studio, the BasicMicro software was a straight forward manipulation. We however did not have a proper data sheet covering the encoders coupling and so we made connection with what we believed to be correct and got results that seem correct when reading the encoders and activating the motors. It has not been possible to

tune the actuators like we can the other motors which is not a problem as they are only supposed to open and close and as they are, they will be able to perform that task as expected. At the very start of the following week, we were able to run both actuators with a python code (listing C.3) and so we set to work and incorporated them to the test system we had running. This was good for demonstration purposes but also, we had not set two roboclaws to communicate as master and slave, something that should be performed in the real launcher system where it will be made of a master and two slaves. The exercise was helpful as setting up this communication meant that we could observe the data packages being transferred between the master and slave, as seen in figure 3.7, helping us get a deeper understanding of the hardware but also an edge on debugging down the line.

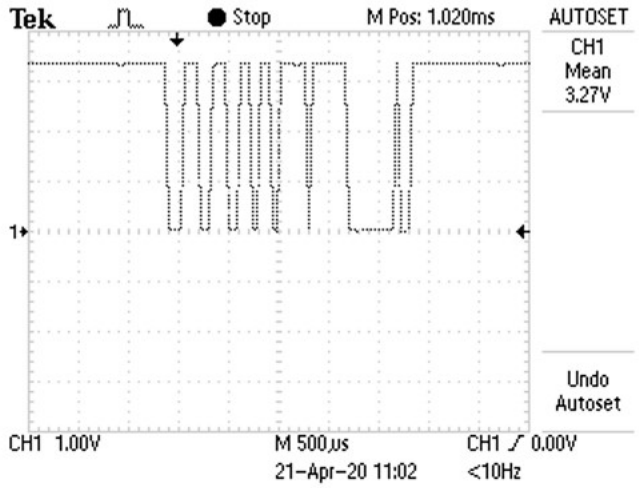


Figure 3.7: Communication package between two roboclaws

4 Results

4.1 Compiling modules

4.1.1 Software compiling

As each of the components got tested with their own code to measure success, integration in the `dronelauncher_python.py` had also been tested. When the time came to pull all of modules together, tests had already been run against the entire code for each component, and is available in its entirety in Appendix A (Listing A.1). The GUI also got some updates and all four updated screens can be seen in Appendix D, figure D.1, D.2, D.3 & D.4.
While the hardware was getting fitted - as described in the following section - work got underway to move the software, with it's new implementations and the extra Roboclaw directives to the launcher's Raspberry Pi. This meant changing the configuration of the Raspberry Pi in order to allow for 1-wire communications for the thermometer, as well as opening the bus communication necessary to interact with the MPU92/65. The system was tested at that stage for response and when these were satisfactory, work got underway to configure the booting routines so that Ngrok (the web-page link provider), the correct python file and the 4G would all get set up correctly when the launcher is switched on. The Roboclaw communications with one master and two slaves were also tested, although the master and first slave did not have motors attached to them, and we collected good communication results for the second slave, the case motors Roboclaw. It should be mentioned here that a 1 kΩ pull-up resistor was added between the master to slaves communication to present a sustained signal reaching these two loads.

4.1.2 Soldering and boxing

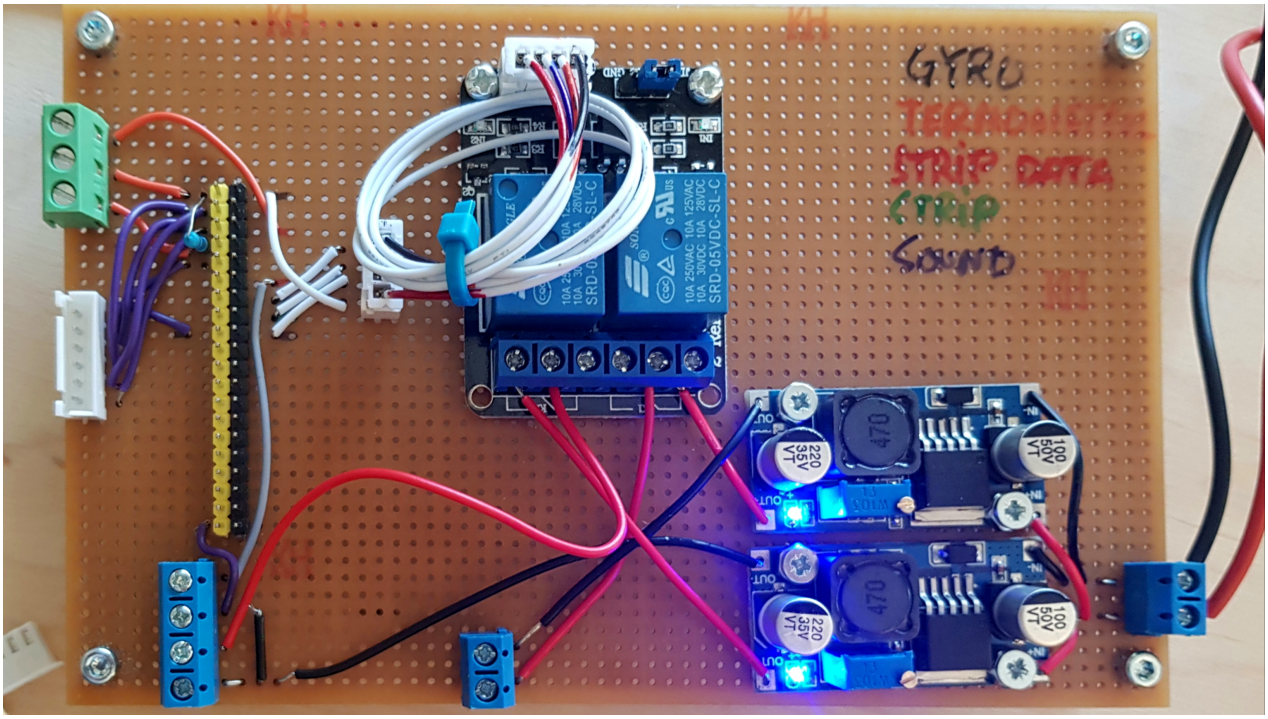


Figure 4.1: Assembled Experimental board

Soldering all the components together was necessary to provide a robust, albeit temporary, solution for the hardware, as seen in figure 4.1. A long term solution would come in the form of a consolidated PCB. The purpose of soldering is as stated, primarily to reinforce coupling of the components. It is not as robust as one could wish or completely bug free. After soldering, some issues with the system took place with the case thermometer, throwing out all of the code. However, having test codes for each of the components, it was possible to isolate where an issue was and electrically debug it but, also to check for the other modules health separately before they could be tested together once again. Once the soldering was over and tests had been successfully ran, all the attention moved to re-boxing the entire system. It has already been mentioned in the Electrical Box section of this document that the wiring had to be done across two electrical boxes. The boxing, seen in 4.2 took most of a day between planning, installing and testing each connection with a multi-meter. The result is a system that works but a permanent, robust solution should be sought to give the hardware a little more flexibility without risking decoupling some of the more experimental connections.

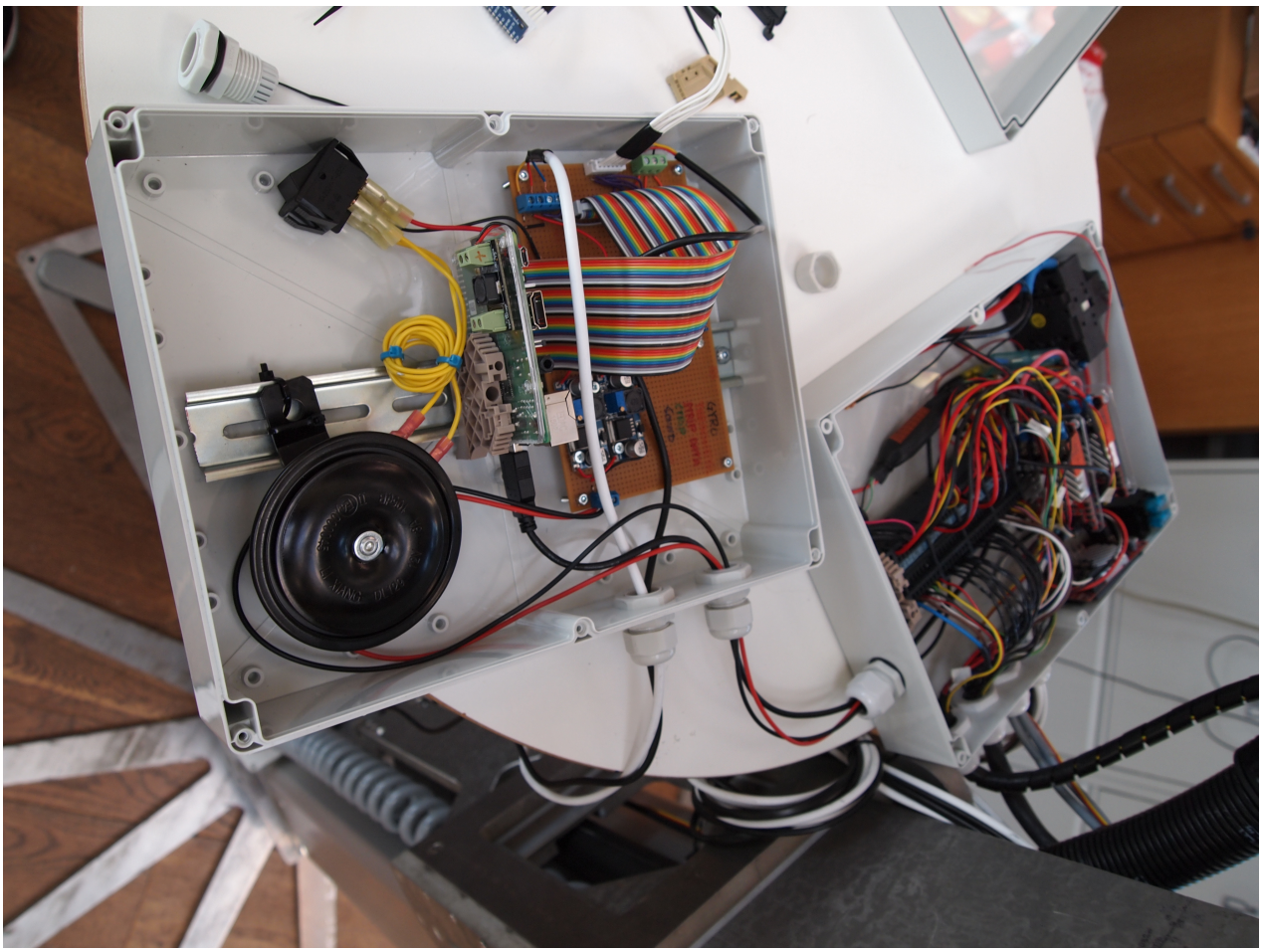


Figure 4.2: Boxing complete

4.2 Testing - Controlled environment

4.2.1 Manual command testing

Manual functions on the launcher worked as expected. They are however, much slower than they have been before as extra weight has been added to the case. The added weight of the case is (without the front display, heavy piece of plastic) another 30 kg from what it originally was and although it is no trouble at all for the column lift, it is very noticeable that the pitch actuator is slow, lifting the launching platform from its minimum position to its maximum position. The electrical bug responsible for the pitch position failure has not been identified during this set of testing and therefore, in this project. The encoders signals were connected to an oscilloscope but no expected encoder signal was detected.

4.2.2 Automatic command testing

The automatic functions - Prepare, Launch, Standby - had actually never been tested until the very last day, where the launcher was put together inside a building where tests were not limited (as it was previously inside the office at Infotiv with limited space). From the original code, it was visible that, apart from the problem with the Pitch angle, all function were using components of the "manual position" functions which had all been tested and vetted. During the tests, at SSRS, we experienced a lag in communication that we could not explain. This threw out the automatic functions and meant that the testing was unsuccessful. The error received from the system seemed like a "pipelining" issue causing delays, pointing to some bottle neck in the Roboclaw communications. Upon returning for some last tests before the project came to an end, we re-configured the Roboclaws as we had reason to suspect that their EEPROM settings were not correct. After doing that, all problems of latency seemed resolved. We experienced some lag later on during the test but their nature seemed to point to the "prototype" coupling of the Roboclaws (single plug in wires) which is not reliable enough to prevent communication errors.

4.2.3 Launch

The launch function suffers from the same pains as the other automatic functions and therefore could not be demonstrated successfully. On the GUI's "settings" page, it is possible to update the speed of launch but this did not show any significant results. A run through all `dronelauncher_python.py` showed that there should not be any problems with the functionalities of these functions. Hard-coding a higher launching base speed will mean that the launcher can manually launch but it has a negative impact on user experience. As we returned for our last test session, we found that there was a switch uncoupled that created the erratic response of the launcher. That mechanical switch is supposed to act as a limiter defining the "0" pulse position of the launcher motor and when it was activated we launched successfully.

4.3 Testing - Outdoors

The time frame to put together this project and test means that we have not been able to run any tests outside, as we only had the full system to test for 2 days. The version of the prototype, we leave behind us, is not waterproof and too heavy to easily transport to an outdoor testing position and back inside, on rainy days. Testing and analysing data will have to be part of an other study.

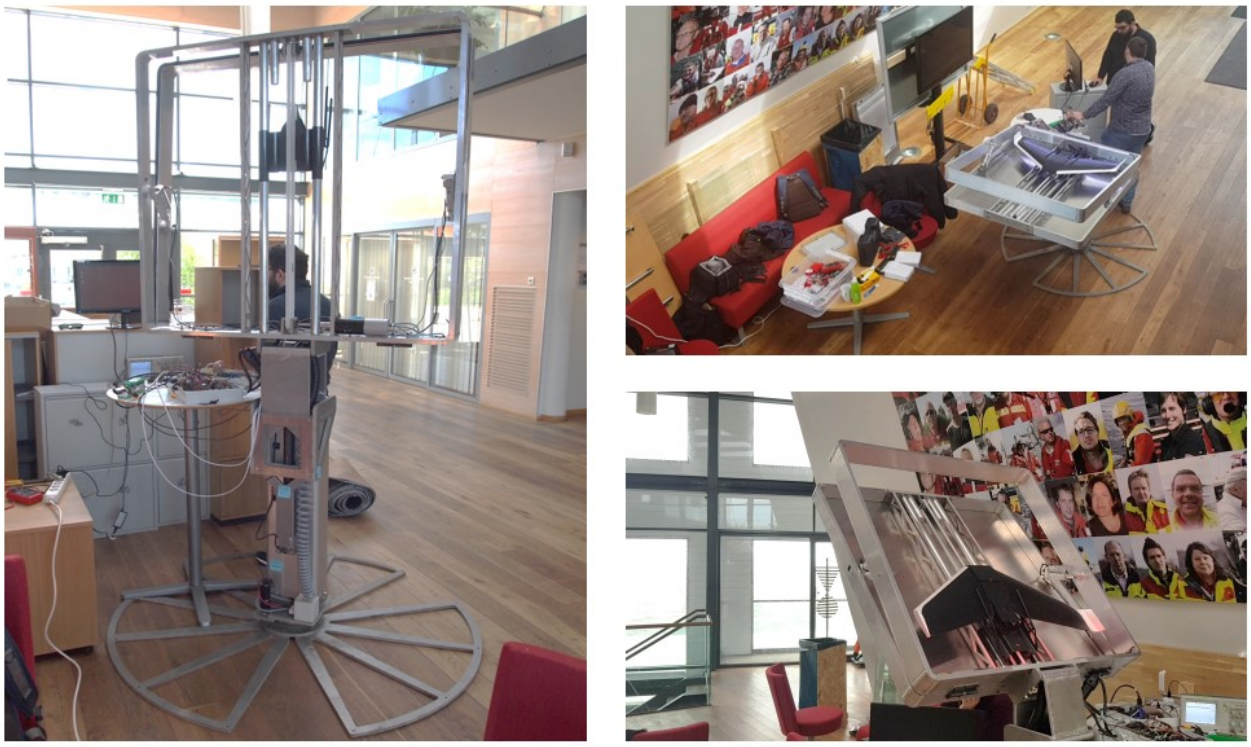


Figure 4.3: Launcher re-assembled and tested on site

4.4 Code robustness

The code, as it runs on the launcher now, is an extension of the original code that was provided to us. This means that we did not altered the code in any way but built new functionalities around it. It is robust as it is exhaustively written but it is not practical as the launcher function are nested inside a flask/JavaScript routine, with independent routes as highlighted in figure E.1, making the launcher inoperable without an access to the GUI and that GUI in turn relies on JavaScript standards being correctly interpreted and/or not going out of date by the internet browsers.

4.5 Code testing from Yrgo

During the development phase, an annex project was created. This was initiated within Infotiv and the aim was to provide a working GUI to students within the test field. This required some meetings and organizing as the GUI is reliant on the Roboclaws being connected for the GUI to give sensible responses rather than connection errors. Some work that been put in, to create a standalone code which would dynamically recognize its Roboclaw connections and has an exception handling routine to "mock" motor functionality, listing C.1.
The root of this exchange between this thesis and the test students was to get feedback on issues that could arise while using the GUI. The feedback received was welcome as it proved to fix some issues notably by improving the booting routines of the system.

5 Conclusions

5.1 Hardware

The hardware was successfully implemented and all deliveries set at the top of this project were delivered as well as some extra functionalities to make usability and the exhaustive tests more intuitive. An example of that is the switch for the horn making testing indoors comfortable for the user. Work has been put in to try and find some more robust solution for all the sensor connections leaving or feeding the Raspberry Pi and cutting the use of weak prototyping cable connections. A robust experimental board was soldered just to that effect. It is, however, not the most robust solution. We did our end of project electrical drawings (see Appendix F) in the software KiCad. There, one would be able to draw plans for a PCB to be made instead of our board. Exchanging the current board for a PCB would make the electrical design more compact but also more reliable and robust. It would also make the addition of new modules easier as it would be an extension of the KiCad drawing.

5.2 Software

The software that is left at the end of this project works as intended and has presented very little to no issues. It is however, exhaustively written and bloated across four languages. This makes the SW hard to use and understand and hard to make further implementations to. As far as the scope of this project is concerned, the SW has the functionalities that were set out as part of our deliveries and in that way it is a success.

5.3 System health monitoring

The launcher, as it may stand outside as a robust plan for temperature monitoring which is valuable information and will help to weed out any potential problems as we set to learn more about the launcher's behaviour in its new environment. However, the data reported by the GUI is limited to "current" values, making it difficult to draw conclusions on the system's health. To remedy to that it is important to send all data to databases so data history, daily maximums, daily minimums and such could be retrieved allowing the user to understand the robot further but also to use this data to make future decisions.

Figure 5.1: The prototype, as it stands in SSRS

6 Hand-over documentation

6.1 User Manual

As this project was taken over, there was no such document as a User Manual. This, in essence, was not an issue as the only potential users were going to be engineers. As the project is due to be delivered and tested by a larger array of individuals, it is critical that the launcher would have with it a User Manual to help users get started with operations but also to highlight and advice on safe use of the rig. A copy of that User Manual can be found in Appendix G. There are several moving parts on the launcher that carry a fair amount of weight. The motors can handle larger loads making physical contact with its moving parts dangerous. however slow moving they might be. There are clamping risks with the cases open and close routines. There is electrical hardware that must be switched off before any investigating can occur...
Further to a short and concise first acquaintance with the system, a second section has been designed to help any engineer attend to system first aid, such as what to look for when faced with system errors and where to find information. A listing of components and their description is available as well as notes on potential issues and improvements that gain in being highlighted to future project workers, engineers and potential users.

6.2 Future improvements

Part of our deliverable have been to try and think beyond our specific project and think about which improvements could be made by future teams working to improve the launcher. Following are two such ideas.

6.2.1 Mini project proposal # 1 - In-case mechanical drone charger

As the drone will sit inside the case during its idle, display time, some solution should be found to have it automatically charge instead of charging it in a separate location. Adding a new battery and/or wiring for it should be fairly straight forward but delivering the energy to the drone "automatically" is less evident. A solution could be to use a wireless charging device posted under the drone as it rests but this would mean adding hardware to facilitate charging by induction in the drone, something that might cause weight issues for the drone. An other solution could be to create a flexible mechanical arm that would be long enough to extend to the charging connection of the drone. The connector should be equipped with a magnetic snap connection enabling an "automatic-yet-mechanical" coupling. This would be a cheap alternative, add no, or very little, weight to the drone itself. the flexibility of the arm should only be in one direction, the direction of launch. That way the coupling would always be right as the drone backs back into its base during standby. During launch, the arm could flex and snap off the magnetic connection without influencing the launch itself.

6.2.2 Mini project proposal # 2 - Solar power

We believe, going forward that some solar panel solution should be sought for to power the launcher. This would mean extra autonomy for the unit. The idea would be to create a close circuit between batteries and solar panels. Placement for such panels would have to be worked out as leaving them unattended to the side of a launcher could prove problematic as interaction with it would be too easy.

Putting it on the launcher itself would be safer but the platform of the final launcher already weights a lot and adding more weight to it not advised unless the pitch motor is changed for a higher voltage motor to limit the current spikes which are already high. This would be a great solution, in-line with the objectives of the launcher, being that it is built to assist people.

6.3 Code readability improvements

The code `dronelauncher_python.py`, as we received it, had some good comments but was missing `"""docstrings"""`. `"""docstrings"""` are used in python in the following line after a function declaration, to briefly indicate what the function can be expected to do. This a helpful tool as the `"""docstrings"""` can be accessed from a `help` search on the code from the command line. This returns each function name and their `"""docstrings"""` without having to look at the actual code. This make the conception of the code easier to assimilate. as seen in figure 6.1 these `"""docstrings"""` are also valuable to individuals that want to comprehend functionalities without interacting with the code.

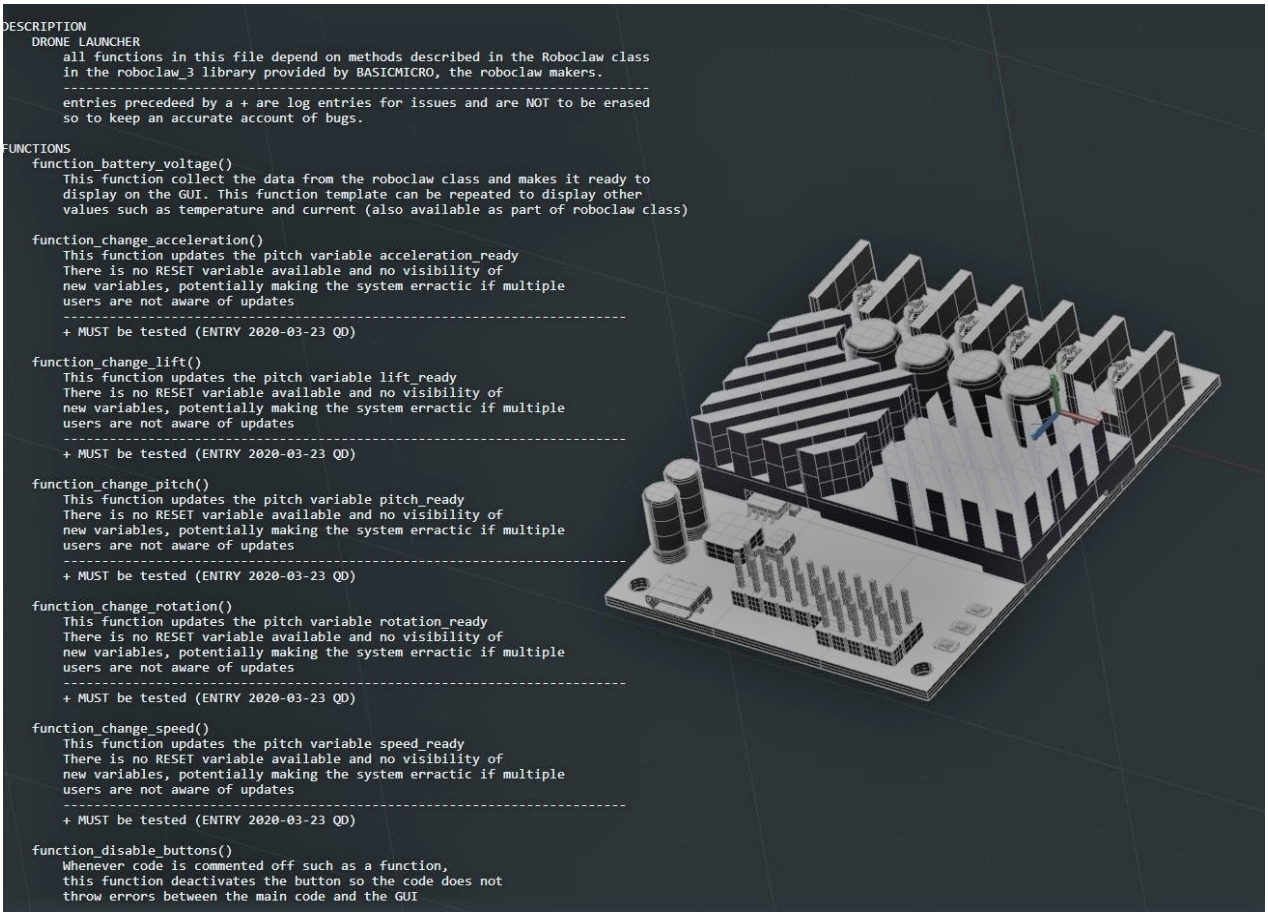


Figure 6.1: Access to docstrings outside of the code

7 Discussion

7.1 Learning curve

When starting this project we were both feeling comfortable with our Electrical Engineering knowledge and capabilities. We felt that we had some ideas about writing codes for hardware and we could reasonably write python code. There were, however, many unknown. Some of which had to do with the nature of the project itself. Quickly, we realised the size of challenge when the python codes were using a flask syntax which we had never seen before, the codes spread across three more languages HTML, CSS and JavaScript, also new to us. No one could confirm or deny that the launcher would work if the battery pack was coupled in and the power turned on. No one could tell us how to get access to the Application that ran the launcher. We had never used Raspberry Pis before.
So we started by enumerating each area of concern we had and proceeded to research and learn about each of the items on that list. We broke down the codes to snippets and understood what each section did. We took up two internet courses on the edX platform to learn about python, flask, HTML, CSS and JavaScript. We found and collected data sheets for each of the electrical components in our rig. Read the Roboclaw User Manual from cover to cover. We found ourselves a Raspberry Pi each ,lovingly named testPi and the phoenyx, as we set to split the workload between us.

7.2 Hitting our stride

When we were done with our learning objectives, we moved on apace to running tests on our stations and sharing successes between us. We set targets for the day together and went about creating our own modules or test our own hardware before integrating together before the day was out. That team work and seeking exchange and/or help when stuck did create the relentless drive needed through this project. Many obstacles were laid in front of us, some set backs were met along the way, mostly as the launcher was unmounted and unavailable from the first test run up to a week before our practical work was over, but the willingness to carry on finding solutions and moving forward was the overriding force. We have learnt more than we could have hoped for but mostly it was incredible fun to be let to play with a robot for 20 weeks.

7.3 Ethics and footprint

Considering the work that has been carried out during this thesis work, no issues of ethics have stalled the process. The launcher could be seen as a medium to projectile objects other than a drone and therefore might be seen as a security issue or some surveillance device. The SSRS launcher, however, is being developed and built to send a drone on its way to assess rescue needs. Further projects, as is understood at this time, are to further develop this technology so it is reliable and safe enough to help the SSRS support its crews across Sweden. The launcher's footprint, or the added launcher footprint of this particular thesis is low as the components added to the current system have been chosen to be as small and energy effective as possible. they are also modular and can be swapped individually without taking a large piece of hardware out of commission. The separate parts of hardware have not been soldered together meaning that if the hardware is to change in the future, components can be recycled easily into other systems rather than being thrown away.

7.4 Next steps

We believe that the next natural steps for the launcher is to extract all the launcher's python code from the flask routines. This code can and should be consolidated as a stand alone class of some type so that an engineer could run all the functions directly from the python code or wire functions to a remote controller to manipulate the rig in a "headless" (without screen) format. That would greatly improve control over the code, it would make tests and prototyping much faster and installing securities could be done in a more general fashion. Some work in that direction is already underway as we had to find some ways to make testing faster and customised to our test area. Later, once the class is organised and it works, it should not be a problem to add separately the flask routine that would call to the methods within the class, much in the same way that a remote controller would. Another step, we feel should be taken is regarding the power supply of the launcher. It is currently power by a battery giving it approximately 30 min of sustained use which is not enough for proper usage if the launcher is to stay on standby until call to action, let alone testing. The launcher should have a power solution that allows it to have constant reliable power as long as it is switched on. In doing that, it would be strongly advised to look at current requirements and regenerative power sent back from the Roboclaws. We believe some of the issues we faced during our testing was related to inconsistent power. Otherwise, we would refer you to the mini proposal #1 and #2.

References

Beltrá, D. V. (2018). *Design and mechatronic integration of a drone launcher* (Master's thesis, Chalmers University of Technology, Divison of Mechatronics). (Cited on page 6).

Connect and Control WS2812 RGB LED Strips via Raspberry Pi. (no date). Retrieved February 20, 2020, from https://tutorials-raspberrypi.com/connect-control-raspberry-pi-ws2812-rgb-led-strips/. (Cited on page 17)

Hrisko, J. (2019, November 15). Accelerometer, gyroscope, and magnetometer analysis with raspberry pi part 1: Basic readings. Retrieved April 5, 2020, from https://makersportal.com/blog/2019/11/11/raspberry-pi-python-accelerometer-gyroscope-magnetometer. (Cited on page 19)

Huawei e3372 product description. (no date). Retrieved April 2, 2020, from https://www.manualslib.com/manual/1336636/Huawei-E3372.html#manual. (Cited on page 21)

Mpu-9250 product specification revision 1.1. (2016, June 20). InvenSense. Retrieved April 10, 2020, from https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf. (Cited on page 19)

Ngrok documentation. (no date). Retrieved March 22, 2020, from https://ngrok.com/docs. (Cited on page 21)

Python 3.7.7 documentation. (no date). Retrieved April 15, 2020, from https://docs.python.org/3.7/. (Cited on page 22)

Raspberry pi ds18b20 temperature sensor tutorial. (no date). Retrieved February 18, 2020, from https://www.circuitbasics.com/raspberry-pi-ds18b20-temperature-sensor-tutorial/. (Cited on page 19)

Roboclaw series brushed dc motor controllers - user manual. (2014). BasicMicro. (Cited on page 19).

Run a Program On Your Raspberry Pi At Startup. (no date). Retrieved March 14, 2020, from https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup/. (Cited on page 21)

A Launcher codes

```
"""
DRONE LAUNCHER

    Created by Daniel Valero Beltra 2018
    @contributors: Quentin Demory & Abdulrahman Alsamel 2020
    @license: MIT license

    All functions in this file depend on methods described in the Roboclaw class
    in the roboclaw_3 library provided by BASICMICRO, the roboclaw makers.

    ------------------------------------------------------------------------

    Entries precedeed by a + are log entries for issues and are NOT to be erased
    so to keep an accurate account of bugs.
"""

#Import modules
from flask import Flask, render_template, request, jsonify
from roboclaw import Roboclaw
import time
import socket
from neopixel import *
import argparse
import threading
import thermo
import MPU9250
import RPi.GPIO as GPIO
from time import sleep
from threading import Thread, Event

"""
#1 SET UP AND VARIABLES
-------------------------------------------------------------------------------

"""

# LED strip configuration:
LED_COUNT      = 60      # Number of LED pixels.
LED_PIN        = 18      # GPIO pin connected to the pixels (18 uses PWM!).
#LED_PIN        = 10      # GPIO pin connected to the pixels (10 uses SPI /dev/spidev0.0).
LED_FREQ_HZ    = 800000  # LED signal frequency in hertz (usually 800khz)
LED_DMA        = 10      # DMA channel to use for generating signal (try 10)
LED_BRIGHTNESS = 255     # Set to 0 for darkest and 255 for brightest
LED_INVERT     = False   # True to invert the signal (when using NPN transistor level shift)
LED_CHANNEL    = 0       # set to '1' for GPIOs 13, 19, 41, 45 or 53

# Create NeoPixel object with appropriate configuration.
```

```
strip = Adafruit_NeoPixel(LED_COUNT, LED_PIN, LED_FREQ_HZ, LED_DMA,
    LED_INVERT, LED_BRIGHTNESS, LED_CHANNEL)

#Setup for the pins for threading the lights and the sound
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.OUT, initial = GPIO.LOW) #Sound
GPIO.setup(24, GPIO.OUT, initial = GPIO.LOW) #Lights

# Define functions which animate LEDs in various ways.
def colorWipe(strip, color, wait_ms=50):
    """Wipe color across display a pixel at a time."""
    for i in range(strip.numPixels()):
        strip.setPixelColor(i, color)
        strip.show()
        #time.sleep(wait_ms/1000.0) #This sleep in case we want to have a
    longer intervall between each led lighting


#Function to thread the lights and the sound while the motor is moving
def relay_activate():
    """
    Threading routine, to activate or reset lights and sound when motor
    functions are active
    """
    while True:
        event.wait()
        while event.is_set():
            GPIO.output(24, GPIO.HIGH)
            GPIO.output(23, GPIO.HIGH)
            sleep(1)
            GPIO.output(24, GPIO.LOW)
            GPIO.output(23, GPIO.LOW)
            colorWipe(strip, Color(0, 255, 0))
            sleep(1)
        colorWipe(strip, Color(255, 255, 255))

#Open serial port
#Linux comport name
rc = Roboclaw("/dev/ttyACM0",115200)
#Windows comport name
# rc = Roboclaw("COM8",115200)
rc.Open()

#Specify IP address and port for the server
host=(([ip for ip in socket.gethostbyname_ex(socket.gethostname())[2] if
    not ip.startswith("127.")] or [[(s.connect(("8.8.8.8", 53)), s.
    getsockname()[0], s.close()) for s in [socket.socket(socket.AF_INET,
    socket.SOCK_DGRAM)]][0][1]]) + ["no IP found"])[0]
port=5000

address = 0x80                  #Controller 1, M1=Pitch, M2=Rotation
address_2 = 0x81                #Controller 2, M1=Lift, M2=Launch
address_3 = 0x82                #Controller 3, M1=Case Open, M2=Case Close

pitch_pulses=355000             #Encoder pulses from the linear actuator
pitch_length=90.0               #Degrees
```

```
pitch_speed_pulses=7000        #Pulses per second
pitch_speed_manual=50          #From 0 to 127
pitch_ready=80.0               #Pitch degrees for the launch (temporary)

rotation_pulses=950000         #Encoder pulses from the rotation motor
rotation_length=180.0          #Degrees
rotation_speed_pulses=16000    #Pulses per second
rotation_speed_manual=15       #From 0 to 127
rotation_ready=5.0            #Rotation degress for the launch (temporary)

lift_pulses=19000              #Encoder pulses from the lifting colum
lift_length=20.0              #cm
lift_speed_pulses=420          #Pulses per second
lift_speed_manual=127          #From 0 to 127
lift_ready=lift_length         #Lift lenght for the launch (temporary)

launch_pulses=14800            #Encoder pulses from the launch motor
launch_length=111.0            #cm
launch_speed_pulses=6*13400    #Pulses per second during launch (145000 max)
    (13400 pulses/m)
launch_speed_pulses_slow=2500 #Pulses per second during preparation
launch_speed_manual=40         #From 0 to 127
launch_acceleration=(launch_speed_pulses**2)/13400 #Acceleration during
   launch (pulses/second2)
launch_max_speed=10            #Maximum launch speed
launch_min_speed=1             #Minimum launch speed
launch_max_acceleration=48     #Maximum launch acceleration
launch_min_acceleration=1      #Minimum launch acceleration
launch_standby=8000            #Drone position during stand-by
launch_mount=17000             #Drone position during mounting
launch_break=21000             #Belt position during breaking
launch_bottom=0                #Drone position at the back part of the
   capsule
launch_connect=2190            #Belt position for touching the upper part

encoders_ready = 0             #At the beggining, the encoders are not ready

#Create an instance of the Flask class for the web app
app = Flask(__name__)
app.debug = True

#Render HTML template
@app.route("/")
def index():
    return render_template('dronelauncher_web_test.html')

#Motor controller functions
#rc.ForwardM2(address, rotation_speed_manual)
#rc.ForwardM2(address,0) #Both commands are used to avoid rotation

"""
#2 MANUAL FUNCTIONS
-------------------------------------------------------------------------------

"""

@app.route('/app_pitch_up', methods=['POST'])
```

```
149 def function_pitch_up():
150     """
151     This function uses the Backward motion described in roboclaw_3 on the
    PITCH actuator.
152     It does so for UP because the actuator coupling is inverted.
153     i.e: 24V connected to ground and ground to 24V. This was done by design
     according
154     to Daniel Beltra. He reported that the right way around had no results.
155 
    ----------------------------------------------------------------------------

156     + At this stage we assume it is a bad coupling on the manufacturing
    side
157     but it is pending further investigation. (ENTRY: 2020-03-20 QD)
158     + At close of project, this was still unsolved but it is of an
    electrical nature
159     """
160     event.set()
161     rc.BackwardM1(address, pitch_speed_manual)
162     return (''), 204 #Returns an empty response
163 
164 @app.route('/app_pitch_down', methods=['POST'])
165 def function_pitch_down():
166     """
167     This function uses the Forward motion described in roboclaw_3 on the
    PITCH actuator.
168     It does so for DOWN because the actuator coupling is inverted.
169     i.e: 24V connected to ground and ground to 24V. This was done by design
     according
170     to Daniel Beltra. He reported that the right way around had no results.
171 
    ----------------------------------------------------------------------------

172     + At this stage we assume it is a bad coupling on the manufacturing
    side
173     but it is pending further investigation. (ENTRY: 2020-03-20 QD)
174     + Same as above
175     """
176     event.set()
177     rc.ForwardM1(address, pitch_speed_manual)
178     return (''), 204
179 
180 @app.route('/app_pitch_position', methods=['POST'])
181 def function_pitch_position():
182     """
183     This function uses current encoder position relative to a desired
    encoder
184     position entered by the user. Depending on its current position,
    activates
185     the up or down motion to reach the desired position and stop.
186 
    ----------------------------------------------------------------------------

187     + This particular function does not stop as intended for the same
    wiring mishaps
188     it is also pending further investigation. (ENTRY: 2020-03-20 QD)
189     """
```

```
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    pitch_position = request.form.get('pitch_position', type=int)
    if pitch_position > pitch_length or pitch_position < 0:
        return (''), 400
    elif pitch_position == 0:
        pitch_objective = 0
    else:
        pitch_objective = int(pitch_pulses/(pitch_length/pitch_position))
    pitch_actual = rc.ReadEncM1(address)[1]
    pitch_increment = pitch_objective-pitch_actual
    if pitch_increment >= 0:
        rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204
    else:
        rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204


@app.route('/app_pitch_stop', methods=['POST'])
def function_pitch_stop():
    """
    Pitch FORCE stop
    """
    rc.ForwardM1(address,0)
    event.clear()
    return (''), 204


@app.route('/app_rotation_right', methods=['POST'])
def function_rotation_right():
    """
    This function uses the Forward motion described in roboclaw_3 on the ROTATION
    motor to proceed left
    """
    event.set()
    rc.ForwardM2(address, rotation_speed_manual)
    return (''), 204

@app.route('/app_rotation_left', methods=['POST'])
def function_rotation_left():
    """
    This function uses the Backwards motion described in roboclaw_3 on the ROTATION
    motor to proceed right
    """
    event.set()
    rc.BackwardM2(address, rotation_speed_manual)
    return (''), 204

```

```
@app.route('/app_rotation_position', methods=['POST'])
def function_rotation_position():
    """
    This function uses current encoder position relative to a desired encoder
    position entered by the user. Depending on its current position, activates
    the left or right motion to reach the desired position and stop.
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    rotation_position = request.form.get('rotation_position', type=int)
    if rotation_position > rotation_length or rotation_position < -rotation_length:
        return (''), 400
    elif rotation_position == 0:
        rotation_objective = 0
    else:
        rotation_objective = int((rotation_pulses/(rotation_length/rotation_position))/2)
    rotation_actual = rc.ReadEncM2(address)[1]
    rotation_increment = rotation_objective-rotation_actual
    if rotation_increment >= 0:
        rc.SpeedDistanceM2(address,rotation_speed_pulses,rotation_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204
    else:
        rc.SpeedDistanceM2(address,-rotation_speed_pulses,-rotation_increment,1)
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204

@app.route('/app_rotation_stop', methods=['POST'])
def function_rotation_stop():
    """
    Rotation FORCE stop
    """
    rc.ForwardM2(address,0)
    event.clear()
    return (''), 204


@app.route('/app_lift_up', methods=['POST'])
def function_lift_up():
    """
    This function uses the Forward motion described in roboclaw_3 on the LIFT/COLUMN motor to proceed UP
    """
    event.set()
    rc.ForwardM1(address_2, lift_speed_manual)
    return (''), 204

```

```
@app.route('/app_lift_down', methods=['POST'])
def function_lift_down():
    """
    This function uses the Backward motion described in roboclaw_3 on the LIFT/COLUMN motor to proceed DOWN
    """
    event.set()
    rc.BackwardM1(address_2, lift_speed_manual)
    return (''), 204

@app.route('/app_lift_position', methods=['POST'])
def function_lift_position():
    """
    This function uses current encoder position relative to a desired encoder
    position entered by the user. Depending on its current position, activates
    the up or down motion to reach the desired position and stop.
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    lift_position = request.form.get('lift_position', type=int)
    if lift_position > lift_length or lift_position < 0:
        return (''), 400
    elif lift_position == 0:
        lift_objective = 0
    else:
        lift_objective = int(lift_pulses/(lift_length/lift_position))
    lift_actual = rc.ReadEncM1(address_2)[1]
    lift_increment = lift_objective-lift_actual
    if lift_increment >= 0:
        rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204
    else:
        rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204


@app.route('/app_lift_stop', methods=['POST'])
def function_lift_stop():
    """
    Lift/Column FORCE stop
    """
    rc.ForwardM1(address_2,0)
    event.clear()
    return (''), 204



@app.route('/app_launch_forwards', methods=['POST'])
def function_launch_forwards():
```

```
    """
    This function uses the Forward motion described in roboclaw_3 on the LAUNCH motor to proceed Outwards
    """
    event.set()
    rc.ForwardM2(address_2, launch_speed_manual)
    #rc.SpeedM2(address_2,launch_speed_pulses_slow) #Using the speed control instead of the duty cycle because the friction changes in the tube
    return (''), 204

@app.route('/app_launch_backwards', methods=['POST'])
def function_launch_backwards():
    """
    This function uses the Backward motion described in roboclaw_3 on the LAUNCH motor to proceed Inwards
    """
    event.set()
    rc.BackwardM2(address_2, launch_speed_manual)
    #rc.SpeedM2(address_2,-launch_speed_pulses_slow) #Using the speed control instead of the duty cycle because the friction changes in the tube
    return (''), 204

@app.route('/app_launch_position', methods=['POST'])
def function_launch_position():
    """
    This function uses current encoder position relative to a desired encoder
    position entered by the user. Depending on its current position, activates
    the Outwards or Inwards motion to reach the desired position and stop.
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    launch_position = request.form.get('launch_position', type=int)
    if launch_position > launch_length or launch_position < 0:
        return (''), 400
    else:
        launch_objective = launch_bottom
        launch_actual = rc.ReadEncM2(address_2)[1]
        launch_increment = launch_objective-launch_actual
        if launch_increment >= 0:
            rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow, launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
            rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
        else:
            rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-launch_increment,1)
            rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration

        buffer_2 = (0,0,0)
        while(buffer_2[2]!=0x80):   #Loop until all movements are completed
            buffer_2 = rc.ReadBuffers(address_2)

```

```
392         if launch_position == 0:
393             launch_objective = 0
394         else:
395             launch_objective = int(launch_pulses/(launch_length/
    launch_position))
396         launch_actual = rc.ReadEncM2(address_2)[1]
397         launch_increment = launch_objective-launch_actual+launch_connect
398         if launch_increment >= 0:
399             rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,
    launch_increment,0) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
400             rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
401             event.clear()
402             return (''), 204
403         else:
404             rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-
    launch_increment,0)
405             rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
406             event.clear()
407             return (''), 204
408
409
410 @app.route('/app_launch_stop', methods=['POST'])
411 def function_launch_stop():
412     """
413     launch FORCE stop
414     """
415     rc.ForwardM2(address_2,0)
416     event.clear()
417     return (''), 204
418
419
420 @app.route('/app_max_pitch', methods=['POST'])
421 def function_max_pitch():
422     """
423     This function evaluates current (encoder) pitch position relative to
    the max position and proceeds to it.
424     """
425     if encoders_ready == 0: #Not execute if the encoders are not ready
426         return (''), 403
427     event.set()
428     pitch_objective = pitch_pulses
429     pitch_actual = rc.ReadEncM1(address)[1]
430     pitch_increment = pitch_objective-pitch_actual
431     if pitch_increment >= 0:
432         rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(
    address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
433         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
434         event.clear()
435         return (''), 204
436     else:
437         rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
438         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
439         event.clear()
440         return (''), 204
441
442 @app.route('/app_min_pitch', methods=['POST'])
```

```
443 def function_min_pitch():
444     """
445     This function evaluates current (encoder) pitch position relative to
    the min position and proceeds to it.
446     ------------------------------------------------------------
447     + As for other pitch functions, this is malfunctioning as it seems
448     to not interpret encoder once it is in motion. maybe the bad coupling
449     referenced in other log means that the motors and the robocalws are
450     communicating on the wrong channel - LOG OPEN (ENTRY 2020-03-20 QD)
451     """
452     if encoders_ready == 0: #Not execute if the encoders are not ready
453         return (''), 403
454     event.set()
455     pitch_objective = 0
456     pitch_actual = rc.ReadEncM1(address)[1]
457     pitch_increment = pitch_objective-pitch_actual
458     if pitch_increment >= 0:
459         rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(
    address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
460         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
461         event.clear()
462         return (''), 204
463     else:
464         rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
465         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
466         event.clear()
467         return (''), 204
468
469 @app.route('/app_max_lift', methods=['POST'])
470 def function_max_lift():
471     """
472     This function evaluates current (encoder) column position relative to
    the max position and proceeds to it.
473     """
474     if encoders_ready == 0: #Not execute if the encoders are not ready
475         return (''), 403
476     event.set()
477     lift_objective = lift_pulses
478     lift_actual = rc.ReadEncM1(address_2)[1]
479     lift_increment = lift_objective-lift_actual
480     if lift_increment >= 0:
481         rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(
    address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
482         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
483         event.clear()
484         return (''), 204
485     else:
486         rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
487         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
488         event.clear()
489         return (''), 204
490
491 @app.route('/app_min_lift', methods=['POST'])
492 def function_min_lift():
493     """
494     This function evaluates current (encoder) column position relative to
    the min position and proceeds to it.
```

```
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    lift_objective = 0
    lift_actual = rc.ReadEncM1(address_2)[1]
    lift_increment = lift_objective-lift_actual
    if lift_increment >= 0:
        rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(
   address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204
    else:
        rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
        event.clear()
        return (''), 204


@app.route('/app_case_open', methods=['POST'])
def function_case_open():
    """
    This function opens the case on full speed as an absolute order Forward
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    else:
        rc.ForwardM1(address_3, 127)
        rc.ForwardM2(address_3, 127)
    return (''), 204

@app.route('/app_case_close', methods=['POST'])
def function_case_close():
    """
    This function closes the case on a low speed absolute order Backwards
    to prevent for clamping hazards should the stop all command ever fail.
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    else:
        rc.BackwardM1(address_3, 50)
        rc.BackwardM2(address_3, 50)
    return (''), 204


@app.route('/app_home', methods=['POST'])
def function_home():
    """
    This function uses Backwards functions from the roboclaw_3 class
    and limiters on the launcher to return to base (absolut 0) position
    PITCH       90 Degrees
    LIFT        120 cm
    LAUNCH      BACK OF CASE
    ----------------------------------------------------------------------
    + In the current state, the drone's launcher holder falls back all
```

```
    the way and would interact with the limiter. potentially harmless to the system
    but should be tested (ENTRY 2020-03-20)
    """
    event.set()
    rc.BackwardM1(address, pitch_speed_manual)
    rc.BackwardM1(address_2, lift_speed_manual)
    rc.BackwardM2(address_2, launch_speed_manual)

    #rc.SpeedM2(address_2,-launch_speed_pulses_slow) #Using the speed control instead of the duty cycle because the friction changes in the tube
    #Missing rotation limit switch
    event.clear()
    return (''), 204


@app.route('/app_reset_encoders', methods=['POST'])
def function_reset_encoders():
    """
    reset encoders function should only be used if the GUI requests it.
    It usually requests it when the launcher has returned to home and the User
    wishes to use it again. Then the system requests refreshed encoders.
    ----------------------------------------------------------------------
    + Unclear if pressing reset encoders at any time can create some positioning
    issue if the system was not in home position (ENTRY 2020-03-20)
    """
    #rc.ResetEncoders(address)
    #rc.ResetEncoders(address_2)
    global encoders_ready
    encoders_ready = 1 #Encoders have been reset
    return (''), 204


@app.route('/app_battery_voltage', methods=['POST'])
def function_battery_voltage():
    """
    This function collect the data from the roboclaw class and makes it ready to
    display on the GUI. This function template can be repeated to display other
    values such as temperature and current (also available as part of roboclaw class)
    """
    voltage = round(0.1*rc.ReadMainBatteryVoltage(address)[1],2)
    return jsonify(voltage=voltage)


@app.route('/measurements', methods=['GET'])
def data_display():
    '''
    This function gets the data from the gyrscope and the temp sensor send them to the webpage
    '''
```

```
    # threading.Thread(target = thermo.portread_loop, daemon = False).start()
    temp = thermo.read_temp()
    x_rotation = MPU9250.gyro_data()[0]
    y_rotation = MPU9250.gyro_data()[1]
    angle = MPU9250.gyro_data()[2]
    gyro_temp = MPU9250.gyro_data()[3]
    cpu_temp = MPU9250.gyro_data()[4]

    return jsonify(temperature=temp,x_rotation=x_rotation,
                   y_rotation=y_rotation, angle=angle,
                   gyro_temp=gyro_temp, cpu_temp=cpu_temp)
                   #Don't forget to add the return variables

# The following 3 functions were an attempt to collect data from each of the roboclaws
# but was too demanding between the roboclaws and the GUI so they were put aside.
# They worked but created a lag that was unmanageable
#
# @app.route('/rc1_data', methods=['GET'])
# def rc1_data():
#     temp_rc1 = "NaN"
#     pitch_current = "NaN"
#     rotation_current = "NaN"
#     errors_rc1 = "NaN"
#     return jsonify(temp_rc1=temp_rc1,pitch_current=pitch_current,
#                    rotation_current=rotation_current,errors_rc1=errors_rc1)

# @app.route('/rc2_data', methods=['GET'])
# def rc2_data():
#     temp_rc2 = "NaN"
#     column_current = "NaN"
#     launch_current = "NaN"
#     errors_rc2 = "NaN"
#     return jsonify(temp_rc2=temp_rc2, column_current=column_current,
#                    launch_current=launch_current, errors_rc2=errors_rc2)

# @app.route('/rc3_data', methods=['GET'])
# def rc3_data():
#     temp_rc3 = "NaN"
#     case_open_current = "NaN"
#     case_close_current = "NaN"
#     errors_rc3 = "NaN"
#     return jsonify(temp_rc3=temp_rc3,case_open_current=case_open_current,
#                    case_close_current=case_close_current,errors_rc3=errors_rc3)



@app.route('/app_stop', methods=['POST'])
def function_stop():
    """
    FORCE STOP all operations
    """
    rc.ForwardM1(address, 0)
```

```
    rc.ForwardM2(address, 0)
    rc.ForwardM1(address_2, 0)
    rc.ForwardM2(address_2, 0)
    rc.ForwardM1(address_3, 0)
    rc.ForwardM2(address_3, 0)
    event.clear()
    return (''), 204

"""
# 3 AUTOMATED FUNCTIONS
--------------------------------------------------------------------------------

"""

@app.route('/app_standby', methods=['POST'])
def function_standby():
    """
    This function is technically a more complex and customizable version of
    the HOME function.
    All systems return to home position
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()
    print ('STANDBY MODE')
    colorWipe(strip, Color(255, 255, 255))
    pitch_objective = 0
    pitch_actual = rc.ReadEncM1(address)[1]
    pitch_increment = pitch_objective-pitch_actual
    if pitch_increment >= 0:
        rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(
   address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration

    rotation_objective = 0
    rotation_actual = rc.ReadEncM2(address)[1]
    rotation_increment = rotation_objective-rotation_actual
    if rotation_increment >= 0:
        rc.SpeedDistanceM2(address,rotation_speed_pulses,rotation_increment
   ,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately
   ))
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM2(address,-rotation_speed_pulses,-
   rotation_increment,1)
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration

    lift_objective = 0
    lift_actual = rc.ReadEncM1(address_2)[1]
    lift_increment = lift_objective-lift_actual
    if lift_increment >= 0:
        rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(
   address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
```

```
    else:
        rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration

    launch_objective = launch_bottom
    launch_actual = rc.ReadEncM2(address_2)[1]
    launch_increment = launch_objective-launch_actual
    if launch_increment >= 0:
        rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,
    launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-
    launch_increment,1)
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
    rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_standby,0)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
    rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
    event.clear()
    return (''), 204

@app.route('/app_prepare', methods=['POST'])
def function_prepare():
    """
    This function takes position target from the variable list and operates
    the launcher
    from any random position back to it.
    The variables can be updated for a different position using the change_
    ..() functions
    The updates do not override this files variables but rather keeps the
    updates on the
    local machine used for the command.

    ----------------------------------------------------------------------------

    + PITCH could present a problem here and should. The routine should be
    tested to report
    actual results (ENTRY 2020-03-23)
    + ATTEMPT to change variables on one device and test prepare on another
    to see that the above
    "theoretical" statement is correct. (ENTRY 2020-03-23)
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()

    if pitch_ready == 0:
        pitch_objective = 0
    else:
        pitch_objective = int(pitch_pulses/(pitch_length/pitch_ready))
    pitch_actual = rc.ReadEncM1(address)[1]
    pitch_increment = pitch_objective-pitch_actual
    if pitch_increment >= 0:
        rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(
    address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
```

```
    else:
        rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
        rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration

    if rotation_ready == 0:
        rotation_objective = 0
    else:
        rotation_objective = int(rotation_pulses/(rotation_length/
    rotation_ready))
    rotation_actual = rc.ReadEncM2(address)[1]
    rotation_increment = rotation_objective-rotation_actual
    if rotation_increment >= 0:
        rc.SpeedDistanceM2(address,rotation_speed_pulses,rotation_increment
    ,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately
    ))
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM2(address,-rotation_speed_pulses,-
    rotation_increment,1)
        rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration

    if lift_ready == 0:
        lift_objective = 0
    else:
        lift_objective = int(lift_pulses/(lift_length/lift_ready))
    lift_actual = rc.ReadEncM1(address_2)[1]
    lift_increment = lift_objective-lift_actual
    if lift_increment >= 0:
        rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(
    address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
        rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration

    launch_objective = launch_bottom
    launch_actual = rc.ReadEncM2(address_2)[1]
    launch_increment = launch_objective-launch_actual
    if launch_increment >= 0:
        rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,
    launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-
    launch_increment,1)
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
        event.clear()
    return (''), 204

@app.route('/app_launch', methods=['POST'])
def function_launch():
    """
    This function operates the launch motor + belt
    speed can be updated from change_...() functions

    ------------------------------------------------------------------------
```

```
    + This function caused a problem during testing as the mechanical parts of the
    prototype were not able to "connect" and therefore could not operate above a
    certain speed (ENTRY 2020-03-23 QD)
    + During FULL testing we could get the launch to work but the switch limiter
    had to be properly re-fitted for that to happen.
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()

    launch_objective = launch_bottom
    launch_actual = rc.ReadEncM2(address_2)[1]
    launch_increment = launch_objective-launch_actual
    if launch_increment >= 0:
        rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
    else:
        rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-launch_increment,1)
        rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration

    rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_connect,0)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
    rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration

    launch_objective = launch_break
    launch_actual = launch_connect
    launch_increment = launch_objective-launch_actual
    rc.SpeedAccelDistanceM2(address_2,launch_acceleration,launch_speed_pulses,launch_increment,0)
    rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration

    event.clear()
    return (''), 204

@app.route('/app_mount', methods=['POST'])
def function_mount():
    """
    This function ensures that the drone holder comes to the top of the launching rails
    The column goes to lower position
    The Launching platform goes to 0 degrees
    """
    if encoders_ready == 0: #Not execute if the encoders are not ready
        return (''), 403
    event.set()

    pitch_objective = pitch_pulses
    pitch_actual = rc.ReadEncM1(address)[1]
    pitch_increment = pitch_objective-pitch_actual
    if pitch_increment >= 0:
```

```
840         rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
841         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
842     else:
843         rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment,1)
844         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
845 
846     rotation_objective = 0
847     rotation_actual = rc.ReadEncM2(address)[1]
848     rotation_increment = rotation_objective-rotation_actual
849     if rotation_increment >= 0:
850         rc.SpeedDistanceM2(address,rotation_speed_pulses,rotation_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
851         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
852     else:
853         rc.SpeedDistanceM2(address,-rotation_speed_pulses,-rotation_increment,1)
854         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
855 
856     lift_objective = 0
857     lift_actual = rc.ReadEncM1(address_2)[1]
858     lift_increment = lift_objective-lift_actual
859     if lift_increment >= 0:
860         rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
861         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
862     else:
863         rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
864         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
865 
866     launch_objective = launch_bottom
867     launch_actual = rc.ReadEncM2(address_2)[1]
868     launch_increment = launch_objective-launch_actual
869     if launch_increment >= 0:
870         rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
871         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
872     else:
873         rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-launch_increment,1)
874         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
875     rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_mount,0) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
876     rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
877 
878     event.clear()
879     return (''), 204
880 
881 # Automatic launch works, but it is disabled becuase the loop while prevents
882 # the motors to stop when the button Stop is pressed, making it dangerous
883 
884 ##@app.route('/app_automatic_launch', methods=['POST'])
885 ##def function_automatic_launch():
886 ##    if encoders_ready == 0: #Not execute if the encoders are not ready
```

```
##         return (''), 403
##
##     #Prepare
##     if pitch_ready == 0:
##         pitch_objective = 0
##     else:
##         pitch_objective = int(pitch_pulses/(pitch_length/pitch_ready))
##     pitch_actual = rc.ReadEncM1(address)[1]
##     pitch_increment = pitch_objective-pitch_actual
##     if pitch_increment >= 0:
##         rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment
    ,1)
##         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
##
##     if rotation_ready == 0:
##         rotation_objective = 0
##     else:
##         rotation_objective = int(rotation_pulses/(rotation_length/
    rotation_ready))
##     rotation_actual = rc.ReadEncM2(address)[1]
##     rotation_increment = rotation_objective-rotation_actual
##     if rotation_increment >= 0:
##         rc.SpeedDistanceM2(address,rotation_speed_pulses,
    rotation_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
##         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM2(address,-rotation_speed_pulses,-
    rotation_increment,1)
##         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
##
##     if lift_ready == 0:
##         lift_objective = 0
##     else:
##         lift_objective = int(lift_pulses/(lift_length/lift_ready))
##     lift_actual = rc.ReadEncM1(address_2)[1]
##     lift_increment = lift_objective-lift_actual
##     if lift_increment >= 0:
##         rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment
    ,1)
##         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
##
##     launch_objective = launch_bottom
##     launch_actual = rc.ReadEncM2(address_2)[1]
##     launch_increment = launch_objective-launch_actual
##     if launch_increment >= 0:
##         rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,
    launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
```

```
##         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-
    launch_increment,1)
##         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##     rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_connect
    ,0) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately
    ))
##     rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##
##     buffer_1 = (0,0,0)
##     buffer_2 = (0,0,0)
##     while(buffer_1[1]!=0x80): #Loop until pitch is completed
##         buffer_1 = rc.ReadBuffers(address)
##     while(buffer_1[2]!=0x80): #Loop until rotation is completed
##         buffer_1 = rc.ReadBuffers(address)
##     while(buffer_2[1]!=0x80): #Loop until lift is completed
##         buffer_2 = rc.ReadBuffers(address_2)
##     while(buffer_2[2]!=0x80): #Loop until launch is completed
##         buffer_2 = rc.ReadBuffers(address_2)
##     #The loop does not work with AND conditions
##     time.sleep(2)
##
##     #Launch
##     launch_objective = launch_break
##     launch_actual = rc.ReadEncM2(address_2)[1]
##     launch_increment = launch_objective-launch_actual
##     rc.SpeedDistanceM2(address_2,launch_speed_pulses,launch_increment,0)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##     rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##
##     while(buffer_2[2]!=0x80): #Loop until launch is completed
##         buffer_2 = rc.ReadBuffers(address_2)
##     #The loop does not work with AND conditions
##     time.sleep(2)
##
##     #Standby
##     pitch_objective = 0
##     pitch_actual = rc.ReadEncM1(address)[1]
##     pitch_increment = pitch_objective-pitch_actual
##     if pitch_increment >= 0:
##         rc.SpeedDistanceM1(address,pitch_speed_pulses,pitch_increment,1)
    #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM1(address,-pitch_speed_pulses,-pitch_increment
    ,1)
##         rc.SpeedDistanceM1(address,0,0,0) #To avoid deceleration
##
##     rotation_objective = 0
##     rotation_actual = rc.ReadEncM2(address)[1]
##     rotation_increment = rotation_objective-rotation_actual
##     if rotation_increment >= 0:
##         rc.SpeedDistanceM2(address,rotation_speed_pulses,
    rotation_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=
    Execute immediately))
##         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
```

```
##     else:
##         rc.SpeedDistanceM2(address,-rotation_speed_pulses,-rotation_increment,1)
##         rc.SpeedDistanceM2(address,0,0,0) #To avoid deceleration
##
##     lift_objective = 0
##     lift_actual = rc.ReadEncM1(address_2)[1]
##     lift_increment = lift_objective-lift_actual
##     if lift_increment >= 0:
##         rc.SpeedDistanceM1(address_2,lift_speed_pulses,lift_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM1(address_2,-lift_speed_pulses,-lift_increment,1)
##         rc.SpeedDistanceM1(address_2,0,0,0) #To avoid deceleration
##
##     launch_objective = launch_bottom
##     launch_actual = rc.ReadEncM2(address_2)[1]
##     launch_increment = launch_objective-launch_actual
##     if launch_increment >= 0:
##         rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##     else:
##         rc.SpeedDistanceM2(address_2,-launch_speed_pulses_slow,-launch_increment,1)
##         rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##     rc.SpeedDistanceM2(address_2,launch_speed_pulses_slow,launch_standby,0) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute immediately))
##     rc.SpeedDistanceM2(address_2,0,0,0) #To avoid deceleration
##
##     return (''), 204

"""
#4 VARIABLE UPDATES
-------------------------------------------------------------------------------

This area of the code did not seem to update variables during the final FULL tests
it needs to be investigated further in the future.
"""

@app.route('/app_change_pitch', methods=['POST'])
def function_change_pitch():
    """
    This function updates the pitch variable pitch_ready
    There is no RESET variable available and no visibility of
    new variables, potentially making the system erractic if multiple
    users are not aware of updates

    ---------------------------------------------------------------------

    + MUST be tested (ENTRY 2020-03-23 QD)
    """
```

```
    pitch_position_prepare = request.form.get('pitch_position_prepare', type=int)
    if pitch_position_prepare > pitch_length or pitch_position_prepare < 0:
        return (''), 400
    global pitch_ready
    pitch_ready = float(pitch_position_prepare)
    return (''), 204

@app.route('/app_change_lift', methods=['POST'])
def function_change_lift():
    """
    This function updates the pitch variable lift_ready
    There is no RESET variable available and no visibility of
    new variables, potentially making the system erractic if multiple
    users are not aware of updates

    ------------------------------------------------------------------------

    + MUST be tested (ENTRY 2020-03-23 QD)
    """
    lift_position_prepare = request.form.get('lift_position_prepare', type=int)
    if lift_position_prepare > lift_length or lift_position_prepare < 0:
        return (''), 400
    global lift_ready
    lift_ready = float(lift_position_prepare)
    return (''), 204

@app.route('/app_change_rotation', methods=['POST'])
def function_change_rotation():
    """
    This function updates the pitch variable rotation_ready
    There is no RESET variable available and no visibility of
    new variables, potentially making the system erractic if multiple
    users are not aware of updates

    ------------------------------------------------------------------------

    + MUST be tested (ENTRY 2020-03-23 QD)
    """
    rotation_position_prepare = request.form.get('rotation_position_prepare', type=int)
    if rotation_position_prepare > rotation_length or rotation_position_prepare < 0:
        return (''), 400
    global rotation_ready
    rotation_ready = float(rotation_position_prepare)
    return (''), 204

@app.route('/app_change_speed', methods=['POST'])
def function_change_speed():
    """
    This function updates the pitch variable speed_ready
    There is no RESET variable available and no visibility of
    new variables, potentially making the system erractic if multiple
    users are not aware of updates
```

```
    ------------------------------------------------------------------------

    + MUST be tested (ENTRY 2020-03-23 QD)
    """
    speed = request.form.get('speed', type=int)
    if speed > launch_max_speed or speed < launch_min_speed:
        return (''), 400
    global launch_speed_pulses
    global launch_acceleration
    if speed > 7:
        launch_speed_pulses = speed*13400
        launch_acceleration = 655360 #Maximum value
        return (''), 204
    else:
        launch_speed_pulses = speed*13400
        launch_acceleration = (launch_speed_pulses**2)/13400
        return (''), 204

@app.route('/app_change_acceleration', methods=['POST'])
def function_change_acceleration():
    """
    This function updates the pitch variable acceleration_ready
    There is no RESET variable available and no visibility of
    new variables, potentially making the system erractic if multiple
    users are not aware of updates

    ------------------------------------------------------------------------

    + MUST be tested (ENTRY 2020-03-23 QD)
    """
    acceleration = request.form.get('acceleration', type=int)
    if acceleration > launch_max_acceleration or acceleration <
   launch_min_acceleration:
        return (''), 400
    acceleration = acceleration*13400
    global launch_acceleration
    launch_acceleration = acceleration
    return (''), 204


@app.route('/app_disable_buttons', methods=['POST'])
def function_disable_buttons():
    """
    Whenever code is commented off such as a function,
    this function deactivates the button so the code does not
    throw errors between the main code and the GUI

    ------------------------------------------------------------------------

    this works perfectly well and can be tested by commenting out an app.
   route()
    and seeing the button on GUI disappear, or uncommentinmg the automatic
   launch
    and seeing that button appear.
    """
    return jsonify(encoders_ready=encoders_ready)
```

```

# Testing having the lights and temperature as a part of the backend code
#threading.Thread(target = thermo.read_loop, daemon = False).start()
parser = argparse.ArgumentParser()
parser.add_argument('-c', '--clear', action='store_true', help='clear the display on exit')
args = parser.parse_args()
strip.begin()
print ('STANDBY MODE')
colorWipe(strip, Color(255, 255, 255))

#For starting the thread when booting
event = Event()
Thread(target=relay_activate).start()

if __name__ == "__main__":
    try:
        # Deploying main program for the lights:
        # Intialize the library (must be called once before other functions).
#         threading.Thread(target = thermo.read_loop, daemon = False).start()
#         parser = argparse.ArgumentParser()
#         parser.add_argument('-c', '--clear', action='store_true', help='clear the display on exit')
#         args = parser.parse_args()
#         strip.begin()
#         print ('STANDBY MODE')
#         colorWipe(strip, Color(255, 255, 255))  # White wipe

        #app.run(host=host,port=port) #starting the server for the webinterface
        app.run('localhost',port=5000, debug=True)
        #app.run(debug=True)
    except KeyboardInterrupt:
        if args.clear:
            colorWipe(strip, Color(0, 0, 0), 10)
```

Listing A.1: Main python 3 code running the launcher - dronelauncher_python.py

B Temperature monitoring

```
"""
Written by Quentin Demory
License: MIT license
"""
from gpiozero import CPUTemperature

cpu = CPUTemperature()
print("current temperature of the CPU is: {} C".format(cpu.temperature))
```

Listing B.1: CPU temperature reader code

C Testing/Research codes

```
# -*- coding: utf-8 -*-
"""
Created on Tue Apr 14 19:42:47 2020
@author: Quentin Demory
@license: MIT license
"""
from flask import Flask, render_template, request, jsonify
from roboclaw_3 import Roboclaw # This throws a warning but it works fine
import time
import socket
import logging

#logger = logging.getLogger(__name__)
#logger.setLevel(logging.INFO)
#
#formatter = logging.Formatter('%(asctime)s:%(name)s:%(levelname)s:%(
    message)s')
#
#file_handler = logging.FileHandler('launcher.log')
#file_handler.setLevel(logging.INFO)
#file_handler.setFormatter(formatter)
#
#logger.addHandler(file_handler)

logging.basicConfig(filename='launcher.log',level=logging.DEBUG)

#Open serial port
#Linux comport name
#rc = Roboclaw("/dev/ttyACM0",115200)
#Windows comport name
rc = Roboclaw("COM8",115200)
rc.Open()
encoders_ready = 1 # set to 1 so that the position method can be tested

origo = [90.0, 0, 0, 0]
actual_coordonates = origo[:]
case_open = 0

class motor():

    def __init__(self, address, channel, pulses, length, speed_pulses,
    speed_manual, ready):
        self.address = address
        self.channel = channel
        self.pulses = pulses
        self.length = length
        self.speed_pulses = speed_pulses
        self.speed_manual = speed_manual
        self.ready = ready

    def up(self):
        command = [rc.ForwardM1, rc.ForwardM2]
        try:
```

```
52              command[self.channel](self.address, self.speed_manual)
53          except AttributeError:
54              print(f"{command[self.channel]}")
55
56      def down(self):
57          command = [rc.BackwardM1, rc.BackwardM2]
58          try:
59              command[self.channel](self.address, self.speed_manual)
60          except AttributeError:
61              print(f"{command[self.channel]}")
62
63      def position(self):
64          """
65          This method is still bound to pitch in a way but it is good for
    testing
66          """
67          command = [rc.SpeedDistanceM1, rc.SpeedDistanceM2, rc.ReadEncM1, rc
    .ReadEncM2]
68          if encoders_ready == 0: #Not execute if the encoders are not ready
69              return (''), 403
70          try:
71              position = request.form.get('pitch_position', type=int)
72          except RuntimeError:
73              position = int(input('which position do you want to reach: '))
74          if position > self.length or position < 0:
75              return ('Out of bounds'), 400
76          elif position == 0:
77              objective = 0
78          else:
79              objective = int(self.pulses/(self.length/position))
80              print(f'objective = {objective}')
81          try:
82              actual = command[self.channel+2](self.address)[1]
83          except AttributeError:
84              actual = 0 # Set this to any value between 0 and self.pulses
85              print(f'starting point: {actual/(self.pulses/self.length)}
    degrees')
86          increment = objective-actual
87          if increment >= 0:
88              try:
89                  command[self.channel](self.address,self.speed_pulses,
    increment,1) #(address, +-speed, pulses, buffer(0=buffered, 1=Execute
    immediately))
90                  command[self.channel](self.address,0,0,0) #To avoid
    deceleration
91              except AttributeError:
92                  print(f'I go {increment} steps')
93                  print(f'{self.address}, {self.speed_pulses}, {increment}')
94  #                logger.info(f'{command[self.channel]}')
95          else:
96              try:
97                  command[self.channel](self.address,-self.speed_pulses,-
    increment,1)
98                  command[self.channel](self.address,0,0,0) #To avoid
    deceleration
99              except AttributeError:
100                 print(f'I go {increment} steps')
```

```
                print(f'{self.address}, {self.speed_pulses}, {increment}')
#                logger.info(f'{command[self.channel]}')

    def stop(self):
        command = [rc.ForwardM1, rc.ForwardM2]
        try:
            command[self.channel](self.address, 0)
        except AttributeError:
            print(f"{command[self.channel]}")

pitch = motor(0x80, 0, 355000, 90.0, 7000, 127, 70.0)   # pitch
rotation = motor(0x80, 1, 950000, 180.0, 16000, 15, 10.0)  # rotation
lift = motor(0x81, 0, 19000, 130.0, 420, 127, 130.0)    # lift
launch = motor(0x81, 1, 14800, 111.0, 6*13400, 12, 0.0)   # launch has more
    variables...
caseL = motor(0x82, 0, 5000, 5.0, 200, 127, 5.0)  # case left
caseR = motor(0x82, 1, 5000, 5.0, 200, 127, 5.0)  # case right

pitch.up()
#Master_M1.down()
#Slave_M2.position()
print(pitch.address, lift.address, caseL.address)

"""launch_acceleration=(launch_speed_pulses**2)/13400 #Acceleration during
   launch (pulses/second2)
    launch_max_speed=10           #Maximum launch speed
    launch_min_speed=1            #Minimum launch speed
    launch_max_acceleration=48    #Maximum launch acceleration
    launch_min_acceleration=1     #Minimum launch acceleration
    launch_standby=8000           #Drone position during stand-by
    launch_mount=17000            #Drone position during mounting
    launch_break=21000            #Belt position during breaking
    launch_bottom=0               #Drone position at the back part of the
   capsule
    launch_connect=2190           #Belt position for touching the upper
   part """
```

Listing C.1: Dynamic drone launcher program - dronify1.1.py

```
"""
@created: 2020-04-10
@author: Quentin Demory
@licence: MIT license

This code simulates concurrent operations
between a motor and a blinking light which
occur simulteanously although their sleep
times are different.
"""

import RPi.GPIO as GPIO
from time import sleep
from threading import Thread, Event

#initialize GPIOs as LOW (LEDs OFF)
#PIN 23 -> blinking light
#PIN 24 -> solid light representing motor on/off
```

```
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.OUT, initial = GPIO.LOW)
GPIO.setup(24, GPIO.OUT, initial = GPIO.LOW)

def relay_func():
    """
    This function activates the relay on PIN 23
    to switch on and off every second.
    Notice that the code is nested in a while
    loop and therefore listens on the background
    for flags -> event.set() and/or event.clear()
    """
    while True:
        event.wait()
        while event.is_set():
            GPIO.output(23, GPIO.HIGH)
            print("glurb")
            sleep(1)
            GPIO.output(23, GPIO.LOW)
            print("flurb")
            sleep(1)

#set up the event flag
event = Event()
#launch thread
Thread(target=relay_func).start()

#set blinking light, motor on, wait 5 sec
event.set()
GPIO.output(24, GPIO.HIGH)
sleep(5)
#clear light, motor off, wait 5 sec
event.clear()
GPIO.output(24, GPIO.LOW)
sleep(5)
#set blinking light, motor on, wait 5 sec
event.set()
GPIO.output(24, GPIO.HIGH)
sleep(5)
#clear light, motor off, wait 5 sec
event.clear()
GPIO.output(24, GPIO.LOW)
```

Listing C.2: Threading events test code

```
"""
@author: Quentin Demory
@license: MIT license
"""

from roboclaw_3 import Roboclaw
from time import sleep, time

#Windows comport name
#rc = Roboclaw("COM3",115200)
#Linux comport name
```

```
rc = Roboclaw("/dev/ttyACM0",115200)

"""
Whats are the encoders min, max?
M1 --> if 0 then 4958
M2 --> if 0 then 5043
"""

rc.Open()
address = 0x80
address_3 = 0x82

start_time = time()

"""Launch test motor"""
rc.ForwardM1(address, 60)
"""Open case"""
rc.ForwardM1(address_3, 127)
rc.ForwardM2(address_3, 127)
sleep(5) #adequate timing for full open
"""Stop motors"""
rc.ForwardM1(address_3, 0)
rc.ForwardM2(address_3, 0)
sleep(1)
"""Close case"""
rc.BackwardM1(address_3, 127)
rc.BackwardM2(address_3, 127)
sleep(5) #adequate timing for full close
"""Stop motors"""
rc.ForwardM1(address_3, 0)
rc.ForwardM2(address_3, 0)
##sleep(1)
"""Stop test motor"""
rc.ForwardM1(address, 0)

print(time()-start_time)
```

Listing C.3: Open/Close case routine

D Final GUI overview

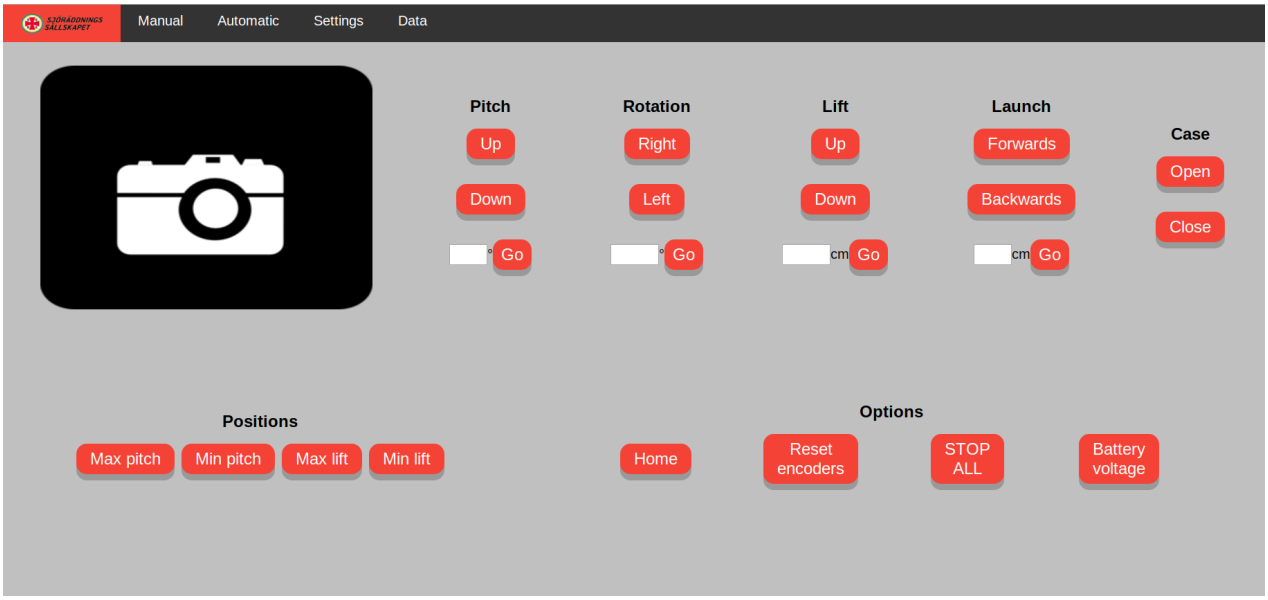


Figure D.1: Manual screen with STOP ALL and open/close case implemented

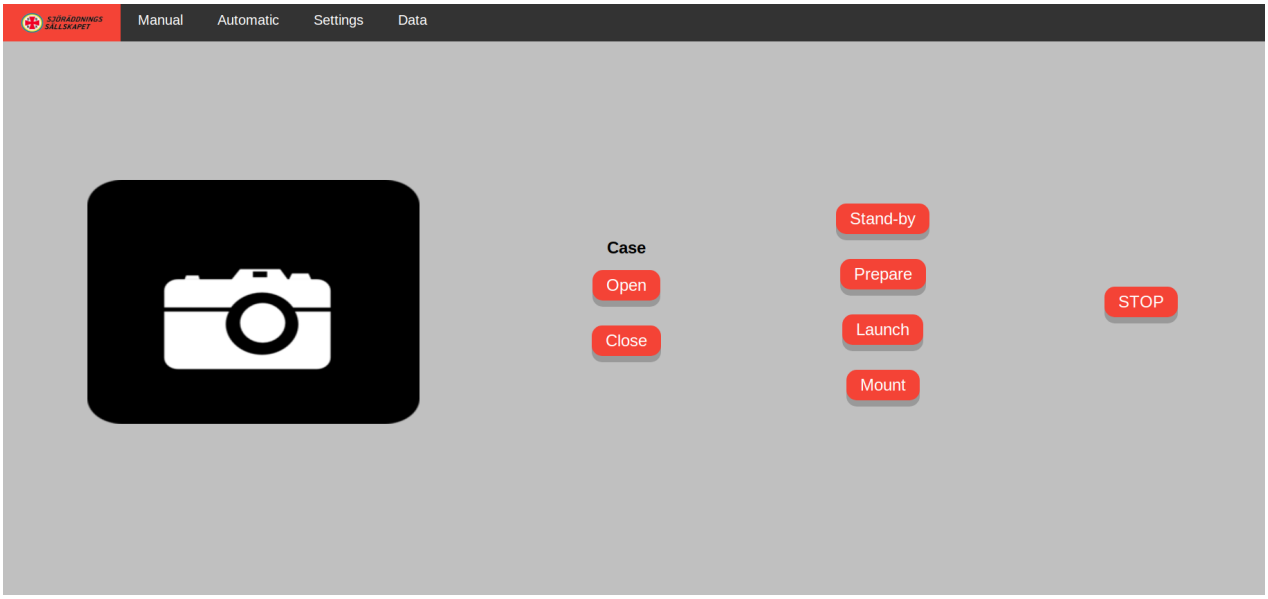


Figure D.2: Automatic screen with open/close case implemented

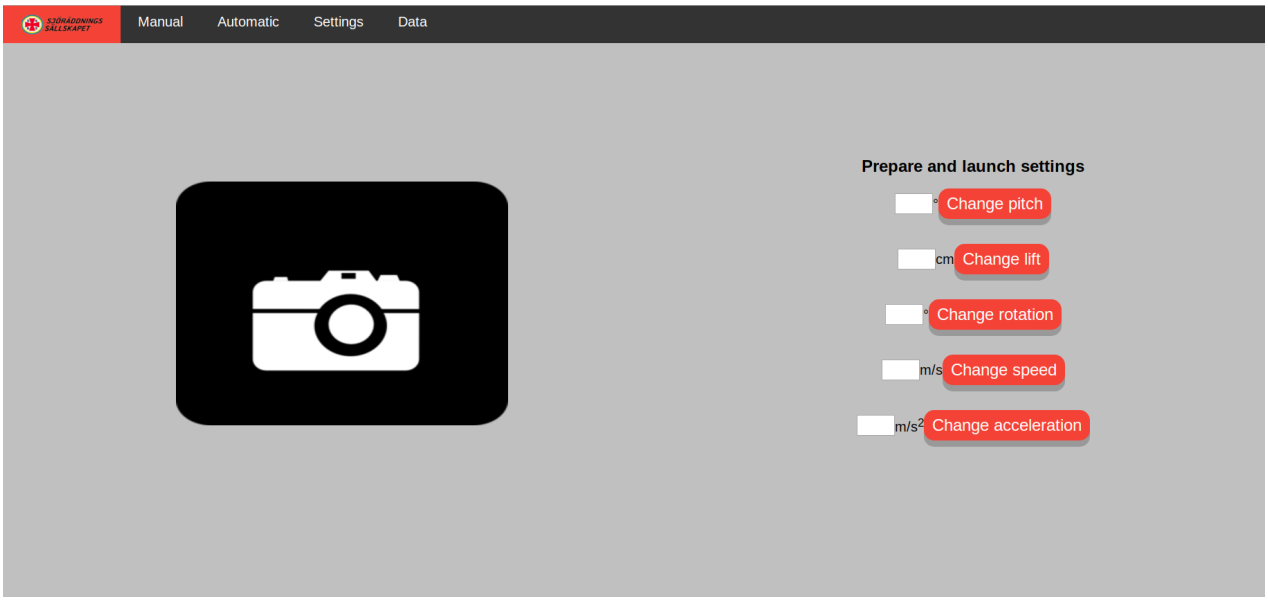


Figure D.3: Settings screen, no changes applied

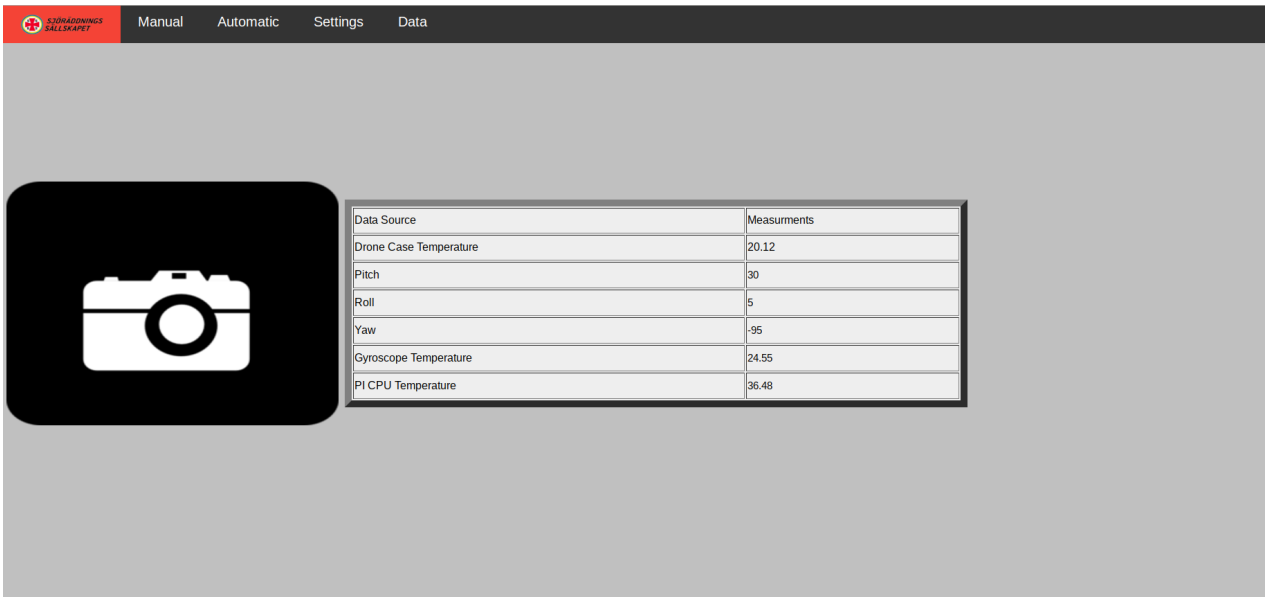


Figure D.4: Data screen with all the live data from sensors available

E Digital transmissions

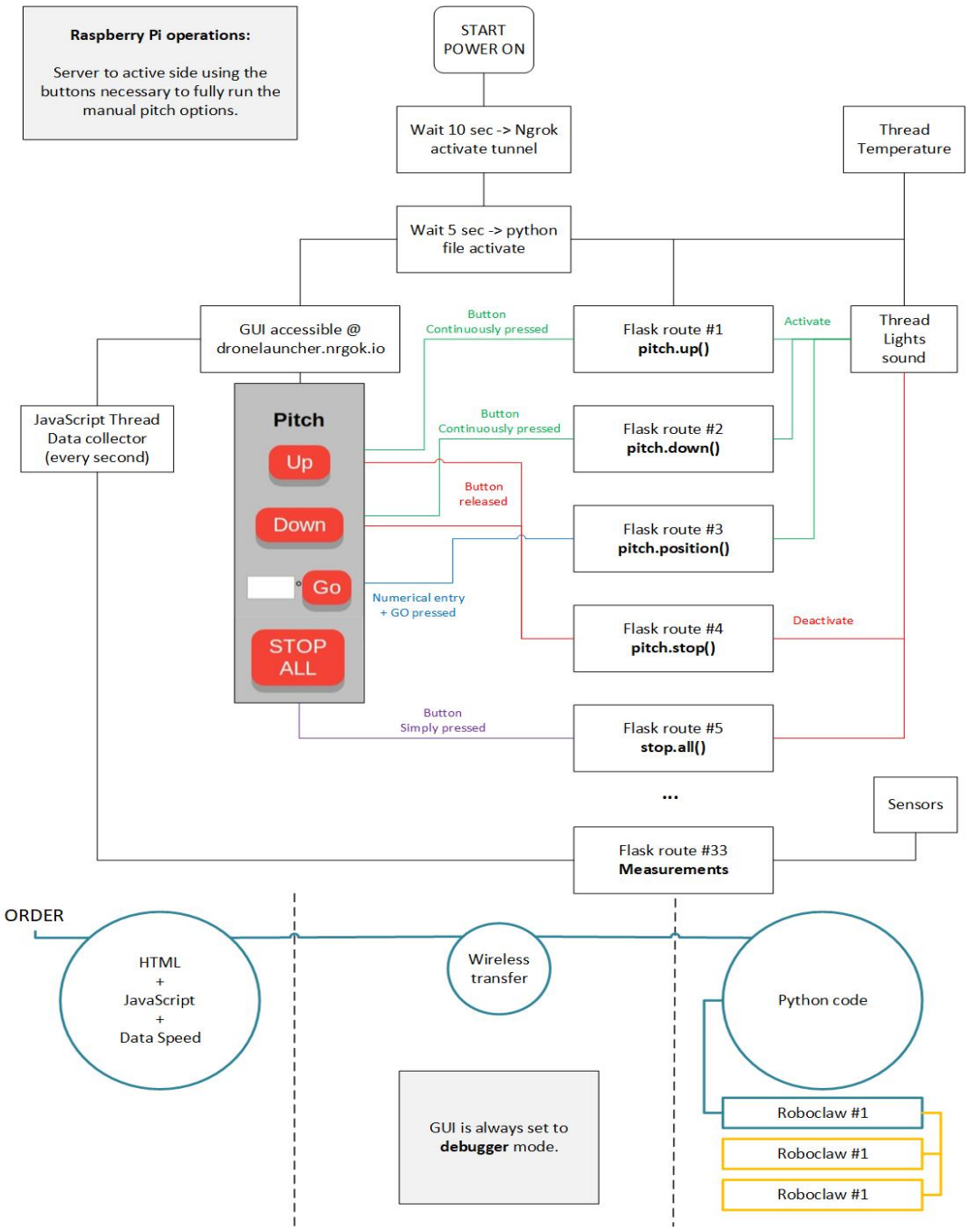


Figure E.1: Exchange between GUI and python files

Figure E.2: A detailed look from GUI to roboclaw order

F End of project Electrical mapping

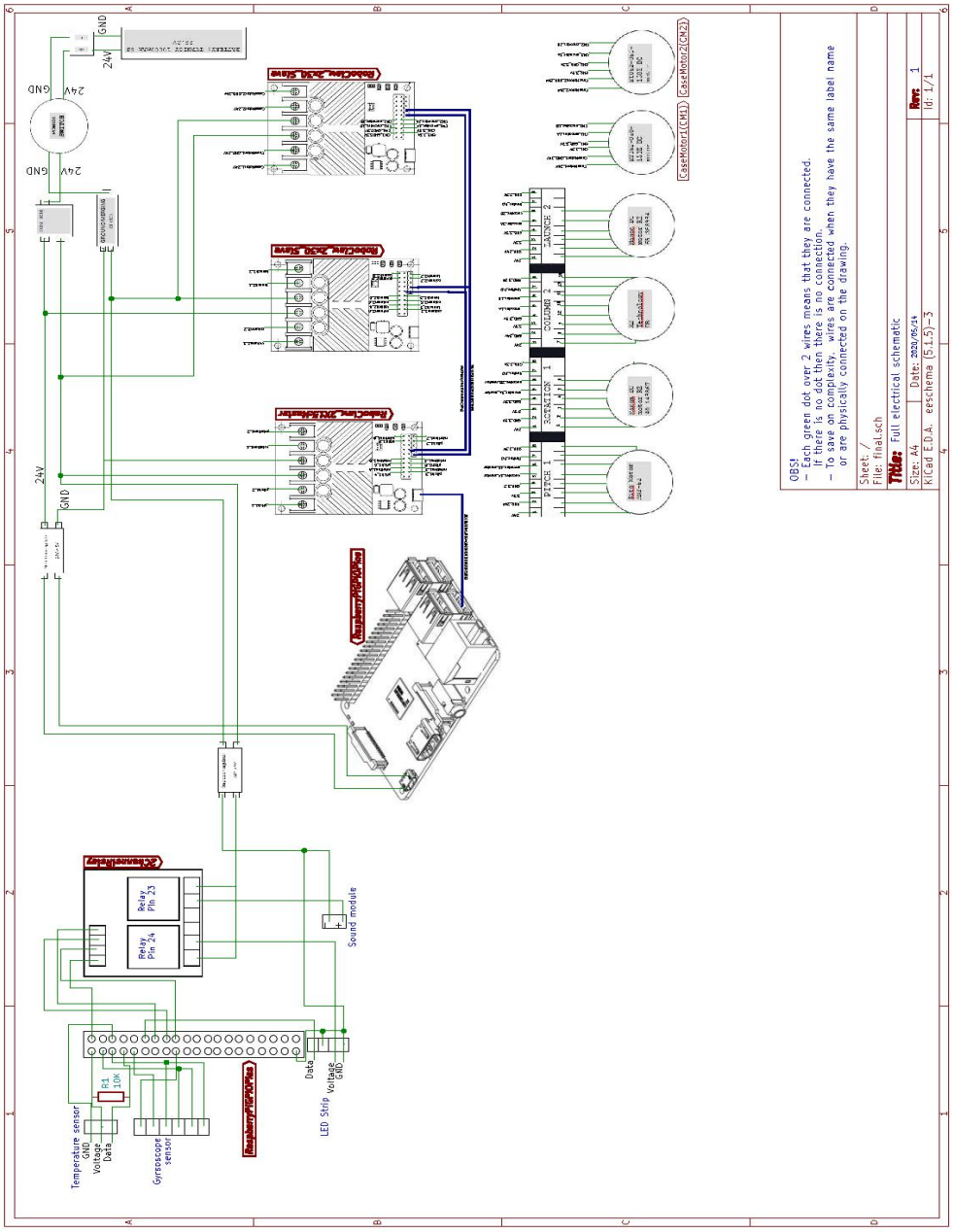


Figure F.1: Full electrical schematics

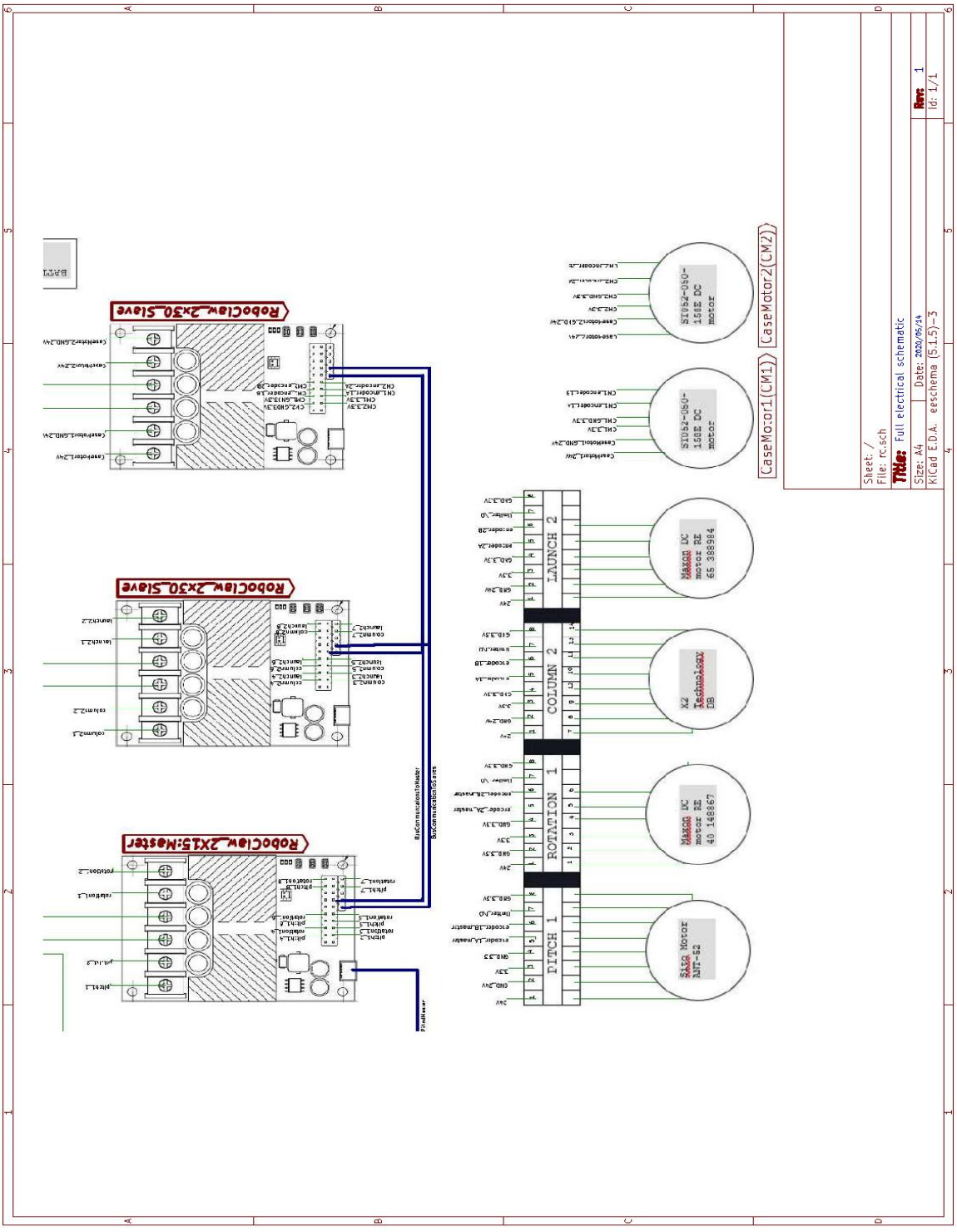


Figure F.2: Roboclaw coupling in focus

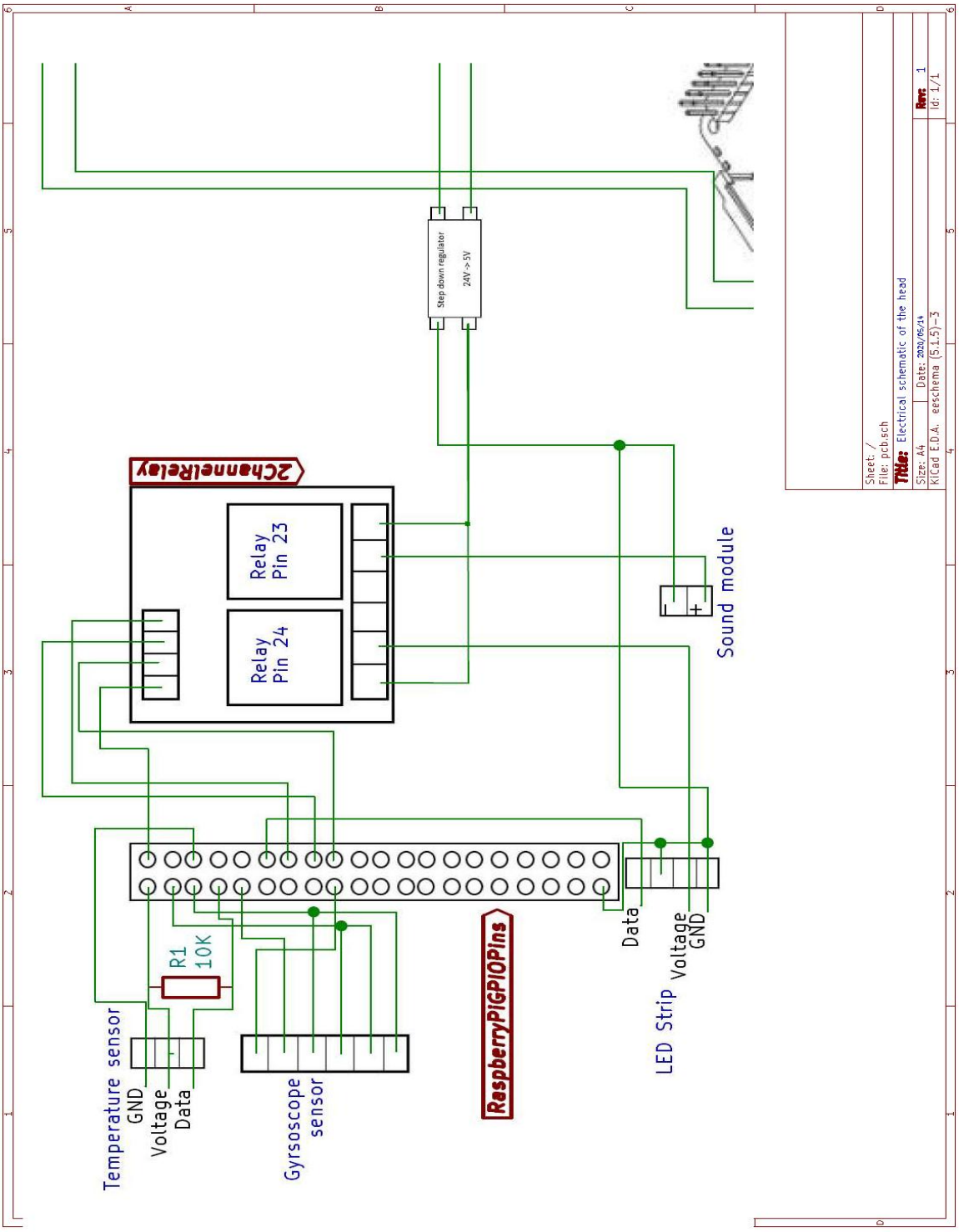


Figure F.3: Break out header coupling in focus

G User Manual (redacted)

As part of the scope of this project, we were tasked to provide some more accessible way of learning about the launcher's system and functionalities. Following, is that document. Non public information has been redacted.

USER MANUAL

SSRS Drone Launcher
Version 1.0

The following document is written by students from Chalmers, as part of their last year thesis work and under the supervision of INFOTIV in Gothenburg. The aim of this document is to help users of the launcher with the operations of the SSRS Drone Launcher. This document will contain two parts. Part I will cover normal use, set up and general safety of the Launcher. Part II will aim to bring as much information to the fore for maintenance, testing and debugging. Every effort has been made to make this document clear and concise to assist users and future developers in their respective tasks.

It should be reinforced here that the prototype of the launcher is still a work in progress and as it is a prototype, extra care should be taken when using it. It is not advised to use this launcher without have read and understood part I of this document.

We believe that we have attended to all the topics and issues that have been met within the scope of our project, in this document, but testing on the whole system was limited in time. This can mean that some issues that can develop, can have gone undetected and so it is primordial that the user/engineer be careful around the hardware, when powered, for themselves and others.

PART I

- Set up
- Normal use
- General safety

Set up

The launcher hardware is essentially composed of a Raspberry pi and 3 × Roboclaw motor controllers which in turn control 6 × motors. The flowchart presented in *figure 1* shows this concept graphically along with a gross communication protocol – in the real system all communications are two way. As a user your interaction with the launcher will be through the Web interface where you will be able to connect to system and test each of the motors and later perform a launch.

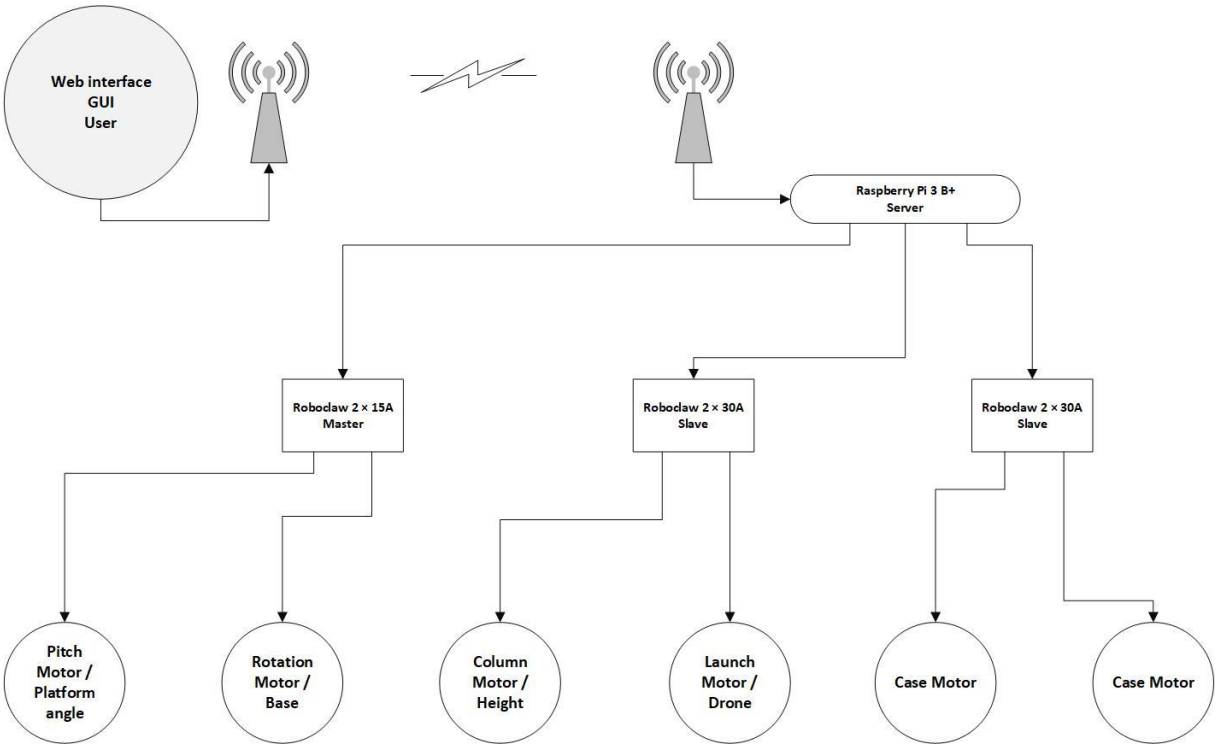


Figure 1: Launcher communication hierarchy.

User Web interface:

It is highly advisable to watch the YouTube video on the following link before interacting with the launcher and the web interface to get a better understanding of how the launcher and the interface interacts with each other but also which motor is involved in which motion:

https://www.youtube.com/watch?v=tI3ZHQFg-mg&t=28s

When using the launcher, after it has been switched on and the light strip has lit up with a solid color white, your only interaction with the system is the Web interface (seen figures below).

To get access to the web interface, please follow these steps:

- Visit the following link on any device from any web browser: [redacted]
- Then you will be prompted to write a username and a password: [redacted]
- At this point, you will see the user interface as shown in the picture below with 4 different pages: Manual, Automatic, Settings and Data.
- Now you will be able to control all the functions of the launcher.
- In order to control one part of the launcher to test each motor for example, then the manual page will be used. The manual page provides control of all the 6 motors: Pitch, Rotation, Lift, Launch and opening/closing the case.
- The Automatic page is responsible for putting the launcher to a certain task or function without the need of controlling each motor. These tasks as shown in the picture can be: Standby, Prepare, Launch and Mount.
- The Settings page is responsible for modifying the Automatic values such as: speed, acceleration, rotation, pitch, and lift. However, as of this version, tests on these variables did not seem to interact with the code, meaning that customizing these variables was unsuccessful.
- The Data page on the other hand is only responsible for presenting live data to monitor the launcher's state. The data that is being pulled are: Case temperature, Lateral angle, PI CPU temperature and the gyroscope temperature.

Figure 2: Manual screen

Figure 3: Automatic screen

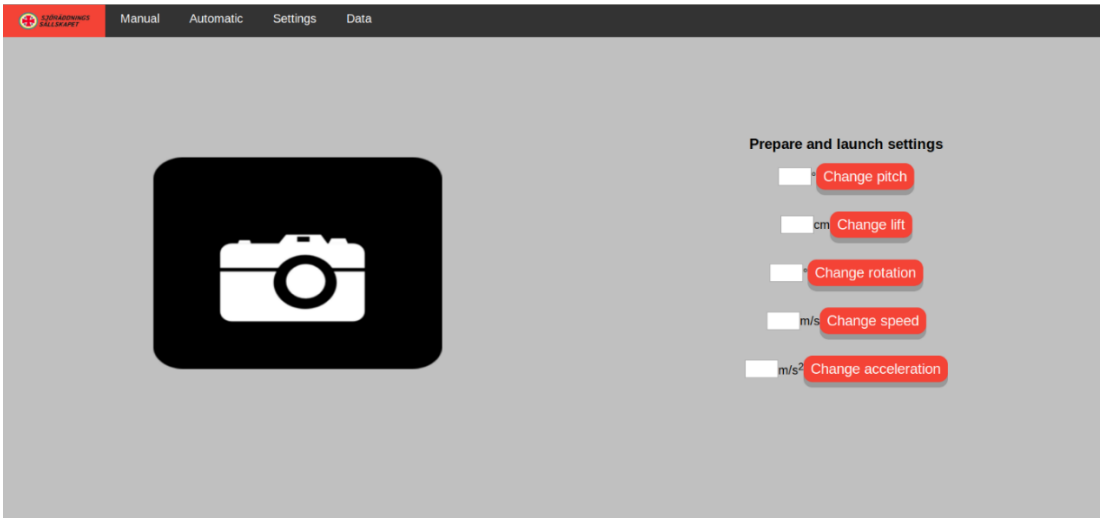


Figure 4: Settings screen

Figure 5: Data screen

Normal use

Testing the motors:

Recommended as part of weekly checks. Testing the motors, as well as Launch testing has been made easy for you. When the Launcher systems are on, the On-Board computer will start acting as a server allowing you to interface in the system. The interface is extremely easy and as several safety features protecting it from out of bounds requests. In other words, it will not break if you ask it to go up although it is already at maximum length. First time users are encouraged to test functions to understand how the launcher works but it is important to keep an eye on battery levels which preferably should not drop under 20.2V and will be critical for the hardware at 18V. DO NOT operate the launcher if people are, or could quickly be within a 2 meter radius from the column.

In the automatic page, there are a couple of functions that already have defined values. When pressing on standby for example, the pitch will go all the way up, the lift will go all the way down (rotation is not part of the standby and can be adjusted manually).

The prepare function will adjust the launcher based on defined values. These values can be changed in the settings page where you can change the pitch, the rotation, and the lift.

The mount function is helpful when there is some maintenance that needs to be done or when putting the drone back. Before pressing on mount (in case the drone case is closed), press on open button and then press mount. The mount function will put the pitch all the way down and the lift will go all the way down to make it easier to reach to the launcher.

Open and close routines for the launching case have not been included as part of the automatic functions but left to be manual instead to leave that control to the user.

General safety

General safety, of the equipment and bodies around the equipment is paramount. As the launcher has moving parts and electricals it is important that safe distances are kept and respected if the launcher is active. As the launcher is set to be used outside, we would ask that the user takes extra care when changing batteries or looking at electricals. The switch should be off and any suspicion of water infiltration or condensation inside the electrical box should be reported immediately and the launcher should be left unused until it has been declared safe to use by a qualified entity. It should also be noted that if the launcher seems to experience delays in response, i.e: a button is pressed but no immediate response is seen, this could indicate a communication error at which point it is best to manual power off and on again to avoid any “backed up commands” from executing while someone could be close. This is especially important when considering the launcher belt and motor as, opposed to the other motors, it moves very fast.

Moving parts: The launcher carries a lot of weight with it and although it is equipped with stopping securities, analog and digital, it is important to keep away from it when operating it as well as make sure that all individuals close to site are aware of its use. The open and close routine for the case works on absolute orders, this means that it will fully close and fully open upon summon which is obviously a dangerous clamping hazard and the user should keep all body parts away from the case when it is being operated (it has been tuned to close very slowly to reduce risks). ***Whenever use is intended, indoor or outdoor, ONLY use the***

launcher if it has enough clearing to operate its full range of motions, this includes walls, roof… but also individuals.

Switches: The power Switch sitting on the outside of the electrical box can handle up to 13kW, way more power than will be running at any time inside the launcher. It is therefore safe to use for power in on or off but also for emergency stops. ***DO NOT open the electrical box or attend to any electricals unless the power switch is off.***

Fuses: In case of an emergency stop or some defect, a fuse box has been installed between the battery/power switch and the hardware. If you find that after some event, the power does not seem to be running in the launcher when it should. Turn the Power Switch off, look to see if the electricals are intact and proceed to check the fuse. If the fuses are intact, it is advisable to call an engineer.

Battery: The battery use for the system is a Turingy 10000mAh 6S with a 10C rating that can deliver 22.2 V to power the entire system. This is a very heavy-duty battery that should not present any problem upon use. As a safety measure it is highly recommended to make sure it charges as it should, and it looks to be intact when looking at it. This battery can discharge up to 100 A (product of the 10C rating and the milli Ampère hour (mAh) value). ***DO NOT plug this battery in the hardware if you are not sure that the power switch is turned off.***

Recognizing error signals on the hardware: The Roboclaw microcontroller are equipped with a lot of gizmos enabling them to check on the health of their circuit. These health reports are not unfortunately interactive with our system as it is but diodes on the Roboclaw indicate errors in the system. Three diodes are positioned along the edge of the controller as shown in figure 2. From top to bottom, as per picture, we have STAT1 & STAT2 normal operation diodes for Channel 1 & 2. The last diode is the ERROR diode. If the error is flashing and the motors are unresponsive, switch the system off and on again to reset the system.

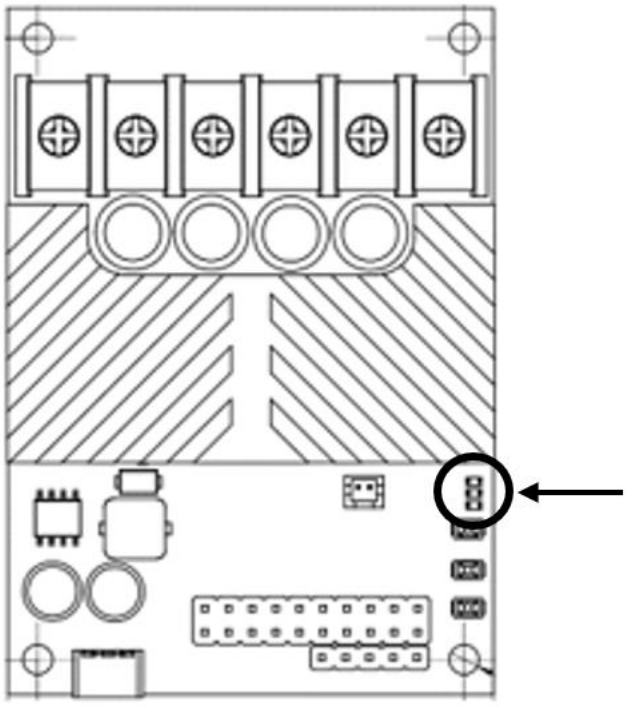


Figure 6: Onboard LEDs, top to bottom, STAT1, STAT2, ERROR

If the errors persist the Roboclaw User Manual provided by BasicMicro has a table of sequencing for these diodes displaying various warnings and we recommend looking through it to understand the root cause of the problem (on max and min positions, the red LED will switch on because it has reached its limit, proceeding in the other direction clears it).

Part II

Nuts and bolts.

This part is a more detailed overview of the components of the launcher. It is aimed at engineers for maintenance, testing and future improvements but we strongly encourage regular users to take a look through it at their leisure as we will still try to make it as understandable and concise as we can so that you can know the prototype better.

Any Engineer interested in launcher maintenance or improvements is advised to read through the Thesis report of Daniel Valero Beltra, “Design and mechatronic integration of a drone launcher” as well as “SSRS Drone launcher” from Abdulrahman Alsamel and Quentin Demory as well as the BasicMicro Roboclaw User Manual. These documents present detailed information on the different parts of the launcher and this document works as a guide based on these works.

Raspberry Pi 3 B+

The Raspberry Pi in the launcher is robust and is preferred over the Raspberry Pi 4 version as speed is not an important factor for the CPU and the slower version, the 3B+ runs cooler. There are no version specific set up in the launcher and the Buck converter used to power the RPi can deliver 3A (3B+ runs on 2.5A whereas 4 runs on 3A) so swapping out for a Raspberry Pi is possible. An aluminum heat sink has been fitted on the CPU and its temperature is being tracked on the GUI as a Raspberry Pi should not be kept on if its CPU runs hotter than 85℃. Should the CPU be found to run hotter, a better cooling solution should be sought. Active cooling is a good suggestion as there are no fans fitting in with the system. Active cooling (heat sink + fan) does very well in CPU stress tests and may also be helpful here although the heat source would be generated from the surrounding environment to the Pi.

3 × roboclaws

3 × roboclaws, the roboclaws control the motors as presented in the section above. They take 24V from the battery to drive the motors and will automatically shut operation if they sense less that 18V available from the same battery. Roboclaws are intelligent motor controllers that have their own monitoring system. I the launcher code, we use their battery sensor capabilities, but they also have current sensors (per motor), temperature sensor (per roboclaw), limitations up and down for voltage, current, temperature, reviewable system status, and more. The python code running the launcher is based on a roboclaw library issued by the manufacturer. There you will find the extent of the roboclaw capabilities. It should be noted that the roboclaws have a pre-installed slot for small fans to facilitate active cooling. These can be set to switch on and switch off at desired temperatures. The current on/off settings for fans can be found in the BasicMicro User Manual.

Changing malfunctioning roboclaws is not difficult. BasicMicro, the roboclaws manufacturers have a download section where the ION Studio can be retrieved. The software is easy to use and allows the user to tune the motors as well as set their address as indicated in “Design and mechatronic integration of a drone launcher” . For example, the address set as of the writing of this document is 128 for the master roboclaw, 129 for the first slave and 130 for the second slave (case motors).

6 × motors

Also available in “Design and mechatronic integration of a drone launcher” is the list of the motors in use for the column and the platform pitch, the launcher rotation, the column lift, and the launcher belt. The other two motors used in the case are an addition form the latest thesis work. All 6 motors are tough and should not present any problems. The only area to be monitored is current usage. The motors when starting motion require a large amount of current that settles quickly but using all the motors concurrently could prove damaging to the roboclaws or the battery pack (fuses have been implemented to secure the battery). As a measure of security, it is recommended to use maximum two motors at once and only the launcher motor alone as it requires a lot of current when running at high speed.

As the system is currently, the rotation motor does not have a home position and must be manually reset. A problem that can be later fixed by creating an origo position and making all movements relative to it. The rotation motor is set in the python code to a low speed. Its rotation speed can be increased but it is advised to keep it low as the upper arm carries a lot of weight and abruptly stopping could damage the system. The rotation motor as no limiters either as those were creating issues with the roboclaws and burn one of its channels. It is therefore recommended to manually.

3 × Step-down converters

Three step-down converters have been used as there are two different voltage levels being used through the system. 24V for the roboclaws and motors and 5V for the Raspberry Pi, the horn, and the light strip. The system could have been designed with only two step-down converters as, the Raspberry Pi needs 2,5 A and therefore its own converter (the converters in question deliver up to 3A), the sound horn was measured to draw 200mA and the light needs 600mA and therefore could be coupled together. It has been wired as it is now as the user can increase the volume on the sound horn by upping the output to 12V and then 3A, should this be required, without impacting the light strip. Changing the output is done by measuring the output with a multimeter and unscrewing the “gold” screw sitting on the blue cap of the converter.

Thermometer DS18B20

The case temperature is monitored by a DS18B20 device set up on a 1-wire communication protocol with the Raspberry Pi. This is a very straight forward set up that requires the 1-wire communication to be set up in the configuration of the Raspberry Pi. This temperature sensor is waterproof as it stands to take temperature outside. It has been wired the system using simple connectors and is therefore easy to remove and replace if found damaged.

MPU 92/65

The MPU 92/65 is the gyroscope, accelerometer, barometer, and temperature unit. It uses the SCL and SCA bus communication in the Raspberry Pi to communicate with all the sensors. This enables the pulling of position data to the Web application. This data, as of writing this

document does not interact with the code, it is only coupled in as a reference. In the future, the system could use the MPU unit to return to a home position and know its current position in space versus the home position.

2 × Relay unit

The two relay units are used to render the blinking lights and beeping sound when the launcher object is in motion. Flags called" Events" have been placed throughout the code to action the relays and stop them when the motion is stopped. The lights are dependent on orders rather than motion which is something that should be fixed to improve reliability and avoid exhaustive coding. As the open and close case functions work on absolute orders, they have not been fitted with blinking light and beeping sound routines. Creating a separate "threading" event with a wait statement between motion and stop state for this purpose would solve the issue.

RGB LED strip (1 m)

The led strip used in the launcher is a Luxoparts LED strip which works with the same libraries as the Adafruit RGB strips. Changing it for another unit is quite straight forward but one needs to update the total sum of LEDs in the python script, should that come to change (the current strip counts 60 LEDs). The strip has been fitted inside a clear tube to make it water resistant as it is not an IPXX rated product.

Sound horn

The sound horn in use in the launcher is a static car horn. It takes up to 12V/3A (60W) and 106 dB. On-board, the voltage has been reduced to 5V and 200mA has the sound level were deemed enough. This can be amended as describe in the step-down converter part of this text.

2 × Power switch

Two switches are found on the launcher. The large unit to the side of the electrical box is the power switch, a circuit breaker, used to turn the launcher on and off. This switch can handle 13kW a rating that the system could not achieve as $13\,000\ \mathrm{W} / 24\ \mathrm{V} = 542\ \mathrm{A}$ which much larger than the maximum 100 A that the battery pack can deliver and more than what the peak current could produce should they all happen at the same time (an occurrence that should never happen or be let to happen).

The second switch has been fitted to switch on and off the horn as its loudness can make it difficult to test comfortably. It should be noted that operating the launcher without sound alarm is not recommended at any time.

Battery pack TURNIGY 10000mAh

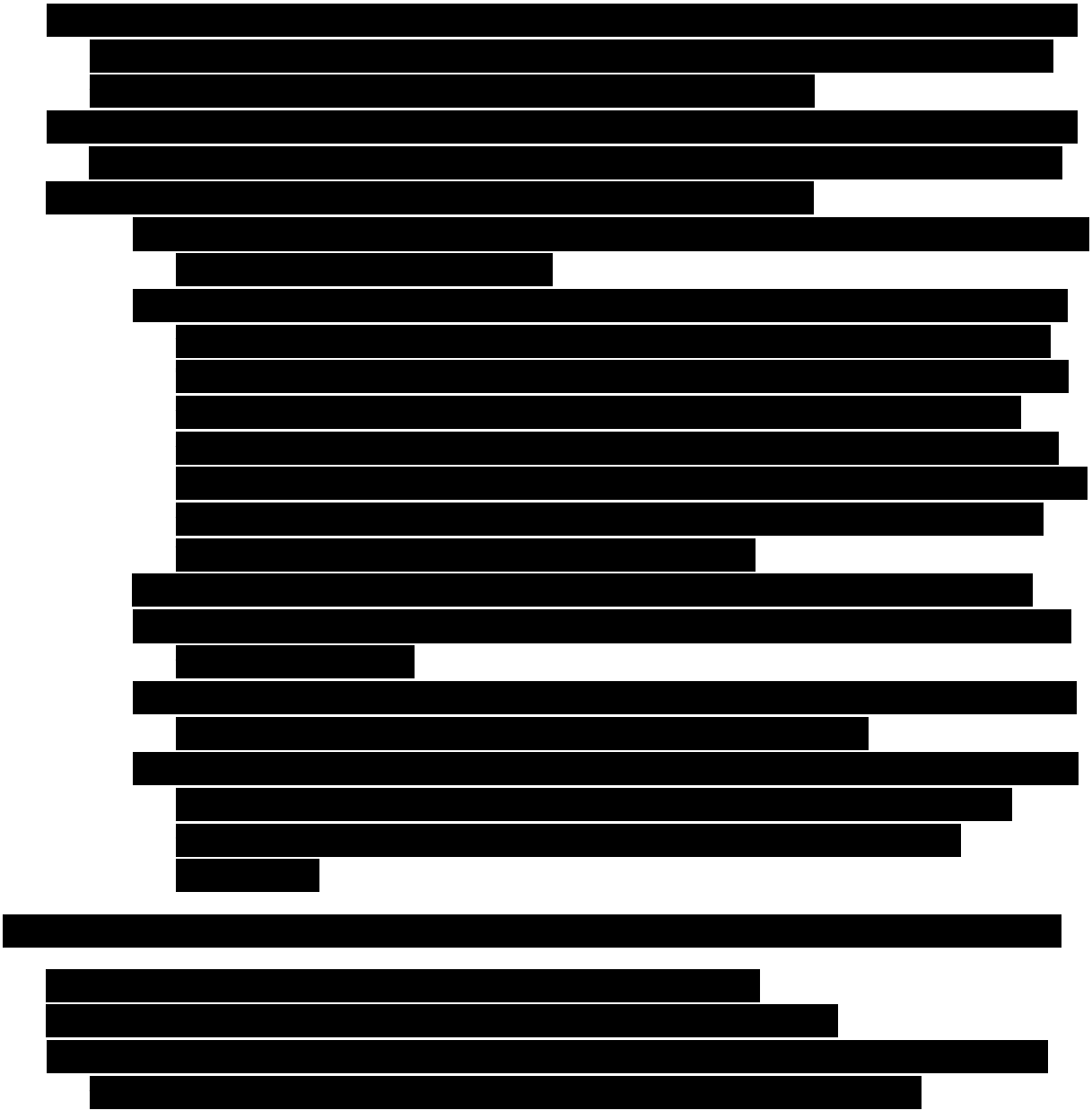
The battery feeding the system is a TURNIGY 22.2 V / 10 000 mAh, 6S (6 cells) with a 10 C rating. This effectively means that the battery can discharge up to 10 C × 10 000 mA = 100 000 mA = 100 A. This battery pack is enough for reasonable use of the launcher. No data is available currently as to timing performances for prolonged use. The only point of reference is

the battery level on the GUI which should be monitored so that the system is not set to function under 18 V. A better solution should be sought when thinking about powering such as getting the power from the mains or using some kind of battery pack to solar panel solution. The battery pack as it is now is inconvenient as it demands regular swapping and recharging making performances suboptimal.

Ngrok tunneling

Ngrok is the service that is responsible for making the server on the local host of the raspberry pi public by providing a tunnel that is protected by a username and password. It is a paid membership which makes it possible to have the same link each time the tunnel starts.

The configurations of Ngrok are saved locally on the raspberry pi and in order to access them for future modifications such as changing the username or the password, follow these steps:



Other devices

- Fuses
  A set of fuses are installed on the system to provide extra securities between the hardware and the battery. This installation was recommended as good practice from the roboclaw user manual and should be observed even if the system moves to a different power source. An extra fuse has been installed on the power output of the battery itself , this is an INFOTIV security mounted on all the battery packs used inside the main office.

- Connectors
  Several types of connectors have been used through the electrical boxes. Apart from a "T" connector is used to join grounds, the connectors are FT terminal block push-in TS35 (part # XW5T-P1.5-1.1-1) which are used to keep the electrical box wiring confined to the box itself. Then all external hardware fixed to the system can be wired to the electrical box whilst keeping coupling and decoupling neat, should the electrical box be moved, resized or taken down.

- Low pass filter
  A low pass filter is fitted between the Master roboclaw and the pitch motor. This was fitted to filter out noise that was making motor hard to analyze with an oscilloscope and therefore hard to tune correctly during the building phases, a further discussion on this component can be found in "Design and mechatronic integration of a drone launcher".

Growing pains

The launcher has a set of potential issues that the user/engineer should be aware of.

Prototype wiring: An effort has been made to wire the newly added system with tougher wiring connections. However, many prototyping connections remain, specially on the roboclaws. These connections are weak and have caused issues with some of the system's behavior during testing. They can look to be properly connected but a harder look with an oscilloscope shows that the connections are not clean enough for the system to effectively use the information. If the system motors seem to behave strangely, check these connections thoroughly. These have, for example, generated a failure in the pitch encoders properly reading positions and, therefore, rendering 'position stop' on the pitch impossible. It only goes from its maximum position to its minimum position and vice-versa.

Rotation motor: The rotation motor has no means to keep itself locked in a position. That means that a push on the launcher will freely rotate the motor and generate stray currents running through the hardware (visible as the Raspberry Pi LEDs and roboclaws LED will light up) which could be harmful to the system, should more current be generated compared to the maximum ratings of the hardware. It also presents a health hazard as the launcher is a very heavy object with variable width making uncontrolled motion dangerous for by standers. A potential solution (which would require constant electrical feed) could be to use the encoder position and ask the rotation motor to return to that position if it varies. Another idea would be a simple mechanical stop that could be fitted when the rotation as reached a position that is satisfactory, but this would require the user manually locking the system.

Case open/close motors: The actuators used for the case did not come with a data sheet, neither could we find a comprehensive one. The encoders were coupled and tested with an oscilloscope, but it was not possible to tune them automatically with the ION studio. Therefore, the motors use absolute orders to open and close the case which is not the most efficient way to open and close, making them reach a position and then stop would be more adequate and easier to install in the code. As it currently stands, the case does not have a light & sound routine when it opens and closes because of these absolute orders and that should be attended to.

Raspberry Pi shutoff routine: currently, when the system is switched off, from the main switch, it cuts all power to the Raspberry Pi and switches it off. Some securities are put in place on the Raspberry Pi, in form of capacitors, to make sure vital systems do not shutoff at once but this is not the way to switch off computers if it can be helped. Some solution to switch off the Raspberry Pi (from the GUI or otherwise) before the main is switch off would go a long way to support the Raspberry Pi's longevity.

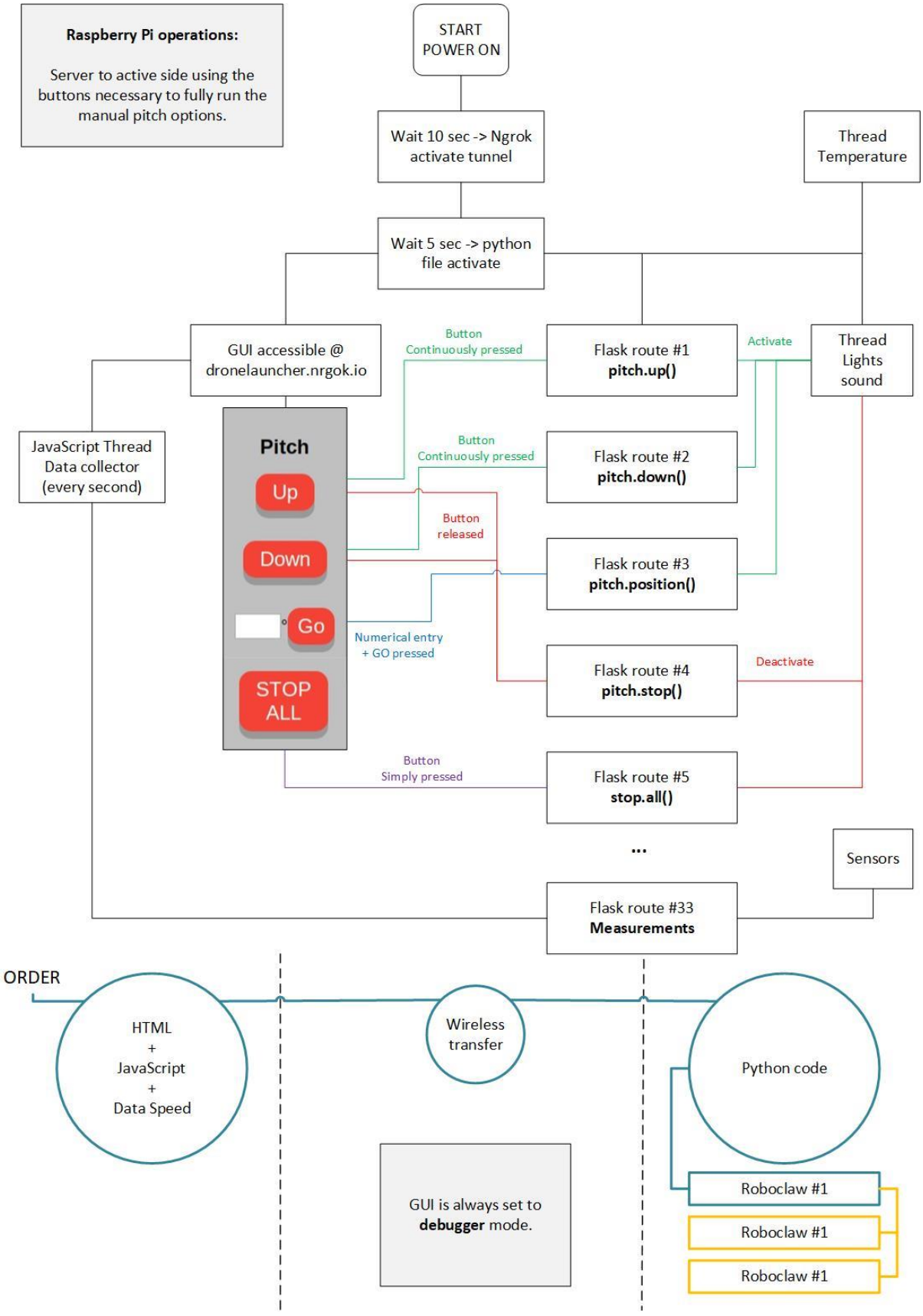
The python code running the launcher: The current code running the launcher is nested inside a Flask application making the system hard to use. Prototyping is also impacted as each new addition needs to be implemented on the GUI (across HTML/CSS/JavaScript). To do well, the system should instead be constructed as an object actionable from the python script without any need for a GUI. Like a plug and play. The methods of the object could then be wired to a Flask interface if that is the preferred way to use the launcher but would not be relying on it as it does today.

No locks on GUI: As it is set today, the GUI can be used by several devices without lock securities. In other words, if two people have access to the webpage at the same time, they can operate it through the GUI concurrently. This is a safety issue as some unexpected motion could be triggered. The web page is fit with a password to access it and so long as there is no locks, we ask of the user to make sure that there is a clear communication when more than one device is logged onto the system.

Monitoring data needs history: The data on the launcher as it is today is immediate. This makes monitoring for system's health and limitations inefficient. The system should store the data so analysis of minimums and maximum at different times and different weather conditions can give us valuable information and prevent unnecessary break down or stop use when oncoming weather is recognized to be weather dangerous for the system.

Magnetometer tuning: the magnetometer data is not currently tuned making the data from it still relative to the launcher's hard iron. Time has not been sufficient to attend to this issue but ways to tune this can be readily found on the internet.

Overview of communication protocols

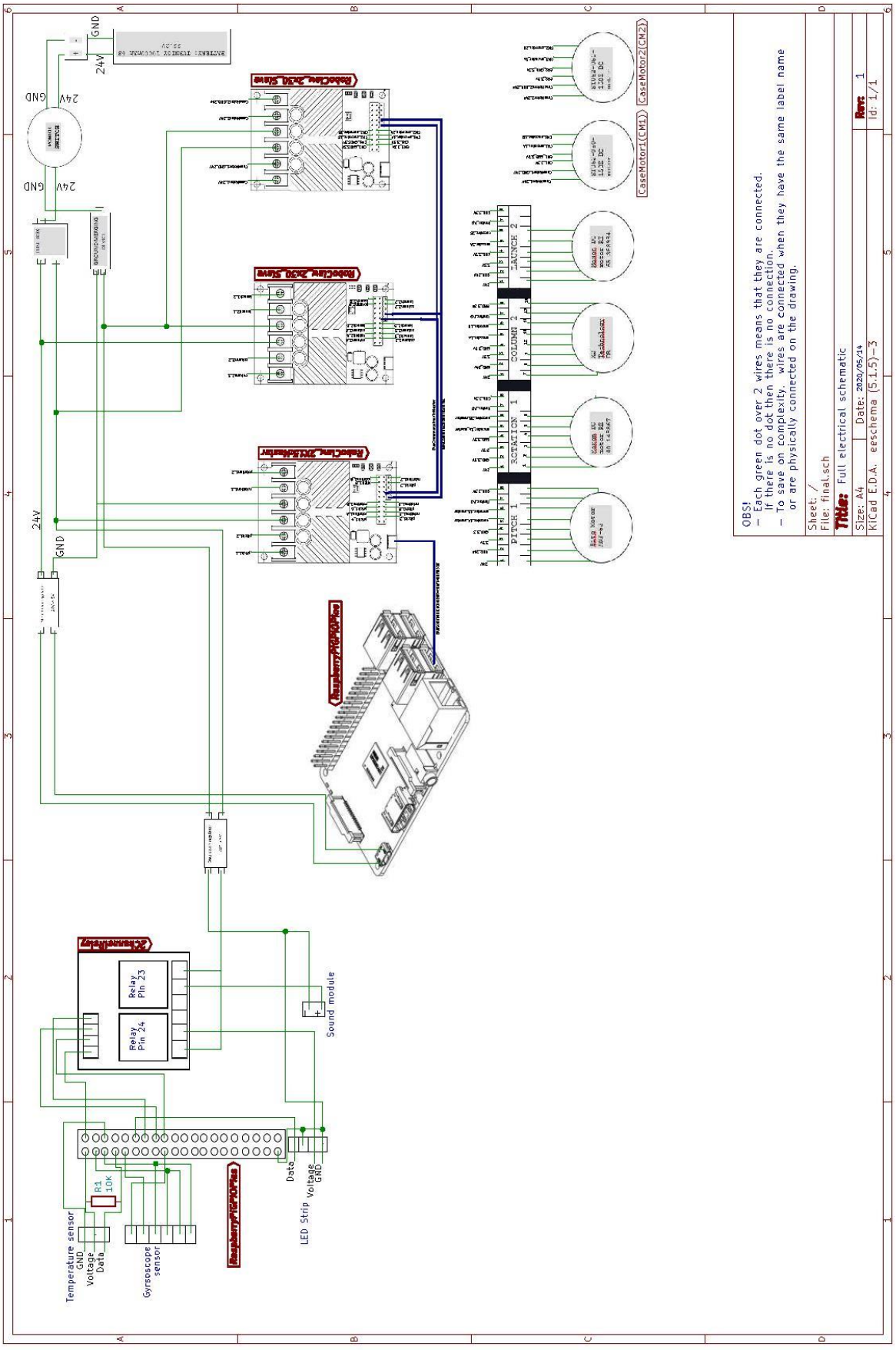

Raspberry Pi operations:
Server to active side using the buttons necessary to fully run the manual pitch options.
START POWER ON
Wait 10 sec -> Ngrok activate tunnel
Thread Temperature
Wait 5 sec -> python file activate
GUI accessible @ dronelauncher.nrgok.io
Button Continuously pressed
Flask route #1 pitch.up()
Activate
Thread Lights sound
JavaScript Thread Data collector (every second)
Pitch
Up
Down
Go
STOP ALL
Button Continuously pressed
Flask route #2 pitch.down()
Button released
Flask route #3 pitch.position()
Numerical entry + GO pressed
Flask route #4 pitch.stop()
Deactivate
Button Simply pressed
Flask route #5 stop.all()
...
Sensors
Flask route #33 Measurements
ORDER
HTML + JavaScript + Data Speed
Wireless transfer
Python code
GUI is always set to debugger mode.
Roboclaw #1
Roboclaw #1
Roboclaw #1

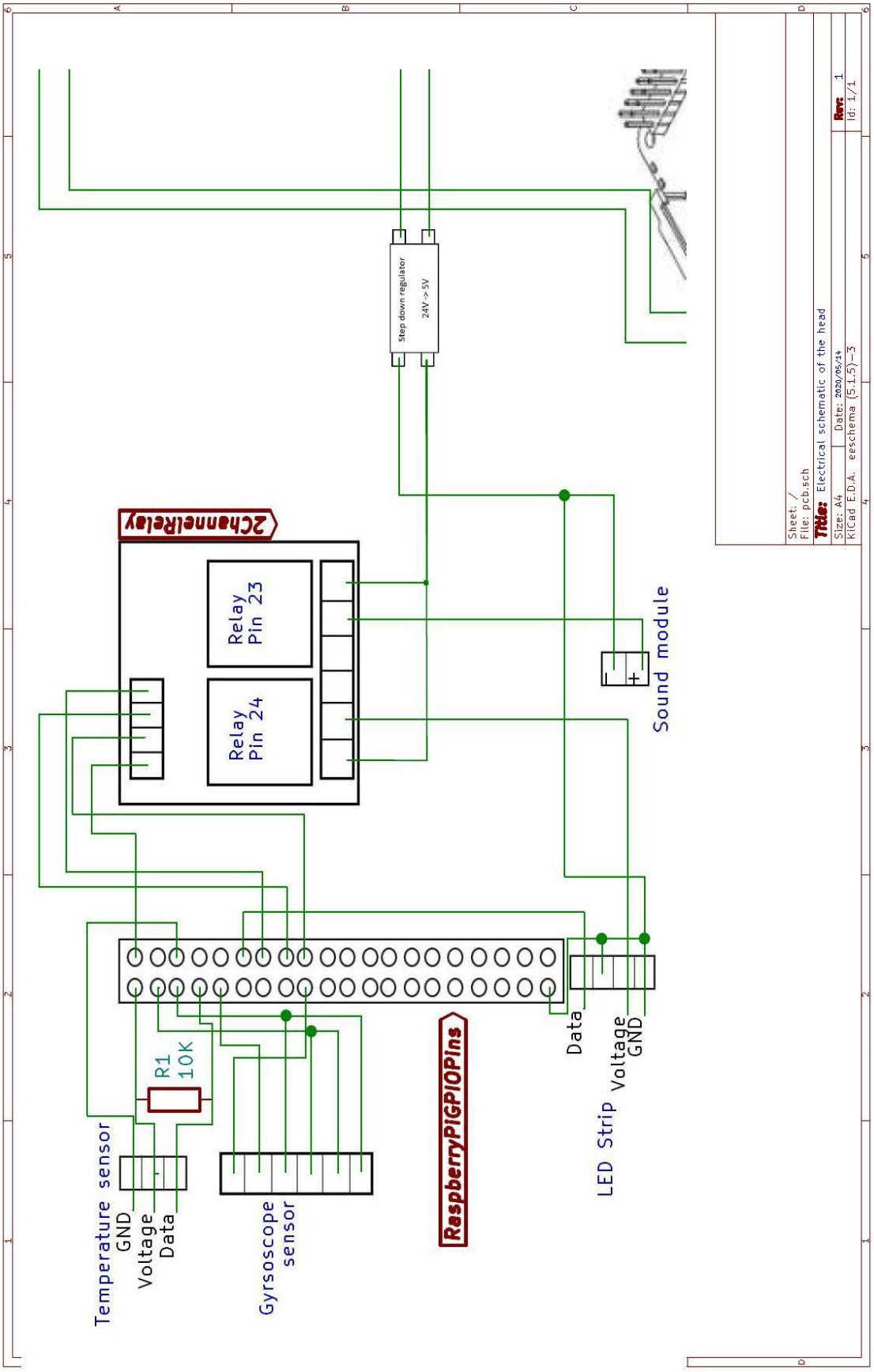
FLOWCHART OF AN ORDER USING THE MANUAL PITCH UP FUNCTION

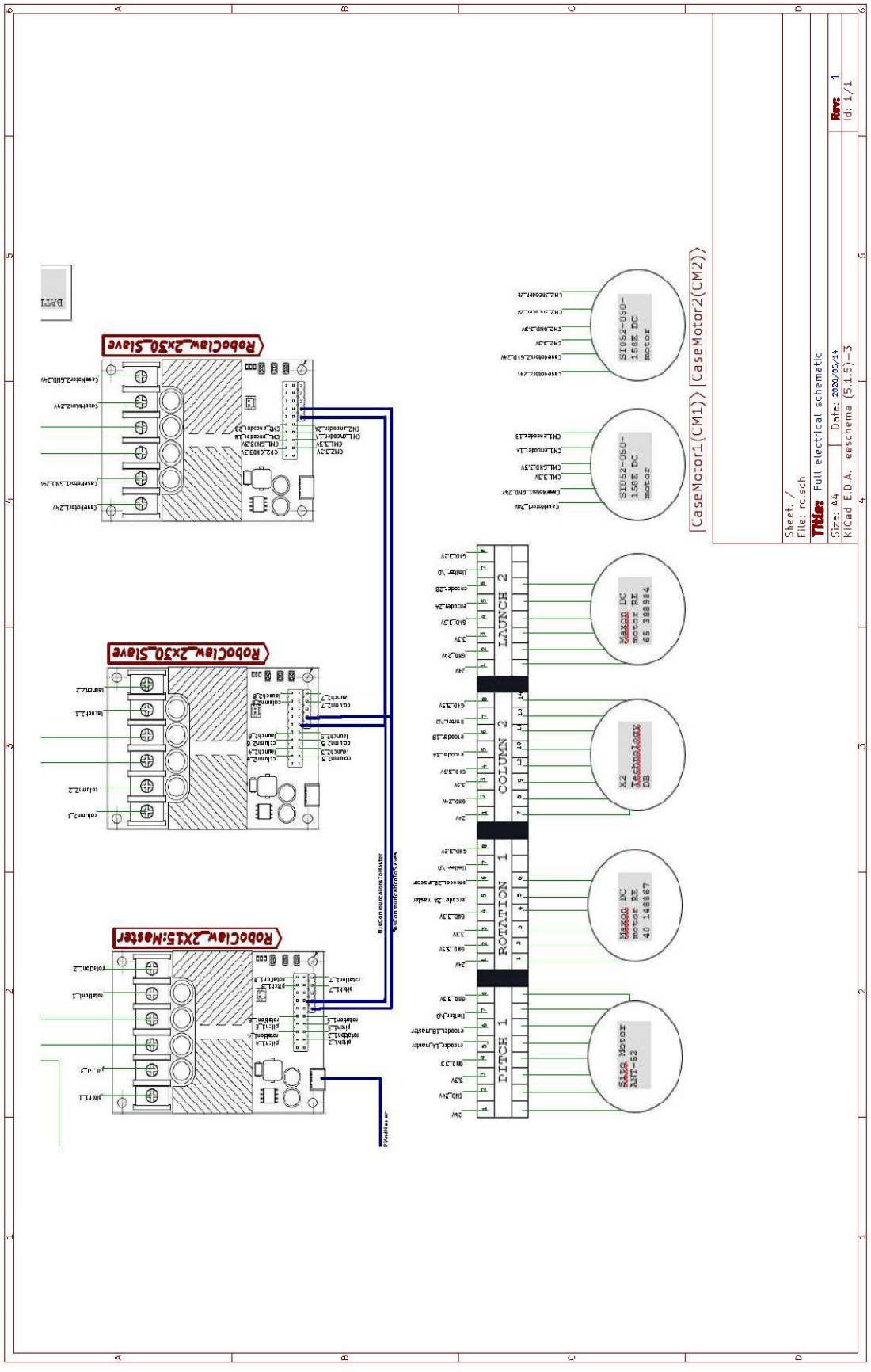
This function lifts the launching platform from 0° to 90°

* The pitch up function was specifically picked as the pitch motor is wired backwards meaning that Backwards to it is Forward. This occurence does not repeat with any of the other motors.

Electrical drawings


OBS!
– Each green dot over 2 wires means that they are connected.
– If there is no dot then there is no connection.
– To save on complexity, wires are connected when they have the same label name or are physically connected on the drawing.
Sheet: /
File: final.sch
Title: Full electrical schematic
Size: A4
Date: 2020/05/14
Rev: 1
KiCad E.D.A. eeschema (5.1.5)-3
Id: 1/1
CaseMotor1(CM1)
CaseMotor2(CM2)
COLOUND 2
LAUNCH
ROTATION 1
PITCH 1
24V
GND
Relay Pin 23
Relay Pin 24
Sound module
R1 10K
Temperature sensor
GND
Voltage
Data
Gyrsoscope sensor
LED Strip
Data
Voltage
GND


Step down regulator
24V -> 5V
2ChannelRelay
Relay Pin 23
Relay Pin 24
Sound module
R1 10K
Temperature sensor
GND
Voltage
Data
Gyrsoscope sensor
RaspberryPIGPIOPins
LED Strip
Data
Voltage
GND
Sheet: /
File: pcb.sch
Title: Electrical schematic of the head
Size: A4
Date: 2020/05/14
KiCad E.D.A. eeschema (5.1.5)-3
Rev: 1
Id: 1/1


RoboClaw_2x30_Slave
RoboClaw_2x30_Slave
RoboClaw_2x15:Master
CaseMotor1(CM1)
CaseMotor2(CM2)
SI1064-060-
150E DC
motor
SI1064-060-
150E DC
motor
Maxon DC
motor RE
65 388984
X2
Technology
DB
Maxon DC
motor RE
40 148867
SAGA Motor
ANT-62
LAUNCH 2
COLUMN 2
ROTATION 1
PITCH 1
Sheet: /
File: rc.sch
Title: Full electrical schematic
Size: A4
Date: 2020/05/24
Rev: 1
KiCad E.D.A. eeschema (5.1.5)-3
Id: 1/1