WE-01 – Tactical Drone

Unmanned Aerial System (UAS)

Architectural Design Document

WECORP INDUSTRIES

Executive Summary

Wecorp Industries Ltd, a subsidiary of Wecorp Ltd, has been commissioned by the jHub accelerator, which is part of Joint Forces Command, to Research, Develop and Manufacture the WE-01 Tactical Unmanned Aerial System (UAS). The WE-01 represents a novel approach to dominate the Urban Warfare environment. The platform is optimised for indoor operation in GPS (Global Navigation System) denied environment and it is a technological solution to the problem of the "First Man" or a substitute for the 'Attack Dog.'

This architectural design document is the main source of documentation to keep track of the project status and to provide evidence of the Standards met during the design.

Version and Amendments

Document Version	Date	Author	Comment
0.1	April 2019	Trigger Board - Michael Strasser Effector – Michael Strasser	Initial version
0.9	December 2019	Ready for production redesign – R&D	Nearly complete version. Next update upon production completion

Table of Contents

Abbreviations and Definitions

ABBREVIATION	DEFINITION	COMMENTS
UK	United Kingdom	…
R&D	Research and Development	…
TRL	Technology Readiness Level	…

Chapter 1 Introduction

The WE-01 has been designed with a high interaction with the customer as well as the user group since the beginning of the project (January 2019). The initial requirements were vague and flexible, such that the project could be tailored and moulded on the needs of the users as well as in the direction of the technological development. After a first design iteration where most of the basic requirements where met in April 2019, clearer requirements were set. These constituted the external constrains of the design and can be summarized as follows:

1. Dimensions: the platform needs to be able to fit through the majority of doors and window. The average door width in UK was then selected as a reference dimension for the design of the platform.
2. Flight time: 15 mins.
3. Effector range: lethality at 12 m.
4. Effector precision: grouping within 30 cm diameter from 12 m distance.
5. Operability (flight): the platform needs to reduce the burden (workload) from the operator and be as easy to fly as commercially available drones. The system needs to have assistance from onboard sensors that allow a non-experienced pilot to be able to operate the system with minimal training.
6. Operability (firing): the effector system can be operated in two modes. Manual mode of aiming and firing, in order to give full flexibility to the user; AI assisted aiming mode. The latter is meant to remove, when possible, any overload from the pilot: even if very stable in space, the drone is a system with 6 dof (degrees of freedom) in space and manually aim precisely at a target might be impossible in certain situations. A combination of camera and Lidar will ensure instead that adjustments to the point of aim are made at 15-40 Hz (15 to 40 times every second).
7. Operability in low light environment (not affecting stability)
8. Light emission control (visible and IR).

The result of this multidisciplinary project is a very multidimensional design, i.e. a design where the variables belong to different disciplines and they are interconnected or dependent on each other.
For this reason, in this document a deductive approach (from general to particular) is adopted to describe the system, even though chronologically often an inductive approach has been used (from particular to general). Hence the structure of this document does not show the logic with which the design has developed, but more the result to which the solution to smaller individual problems and followed by their integration has led.

The architectural design document for the WE-01 is then organised as follows: Chapter 1 gives an introduction to the project and the design starting from the main requirements; Chapter 2 shows the macroscopic level of architecture of the system; more details on the general architecture of each system are given in Chapter 3, i.e. schematics belonging to the structural, electrical and control field as well as the effector and triggering system; Chapter 4 shows the deep level of details in each sub-system; 0 deals with the security architecture; Chapter 5 describes instead the flight envelope testing procedure and summarizes the results of the testing campaign.

Chapter 2 High Level Architecture

The WE-01 Tactical Drone Platform consists of an Unmanned Aerial System (UAS) remotely controlled by an operator via a Radio Transmitter. The simplified diagram below denotes the two systems (Platform and User Interface).

The Platform System is divided into five functional groups, which are identifiable with the dark blue boxes (top raw) in Figure 2.1; these are further split into sub-systems denoted by the turquois boxes and sensor modules displayed in white.

The User Interface in the first iteration of the WE-01 Tactical Drone consists of a Radio Transmitter with Effector Trigger and a Video Downlink. The Encryption interaction between the Platform and User interface is simplified in the diagram.

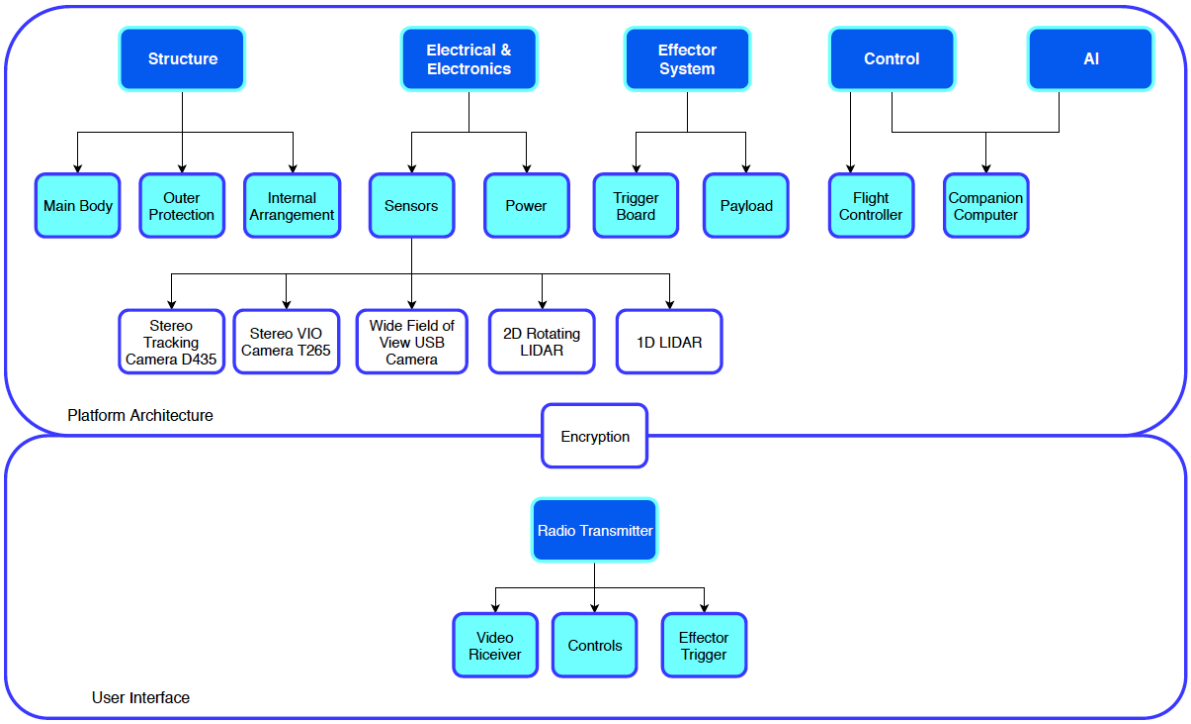


Figure 2.1 Macroscopic System Architecture Diagram

The next chapter will show diagrams and schematics for each system shown in Figure 2.1.

Chapter 3 Platform Architecture

Section 3.1 Component Layout

The WE-01 is a hexacopter UAS whose main body is a carbon fibre composite structure with two main components: a core monocoque body, with the double function of structural component as well as housing (shell) for the electronic components and some sensors, and an outer structure, designed to host the main cameras and to protect the inner core from external actions and impacts.

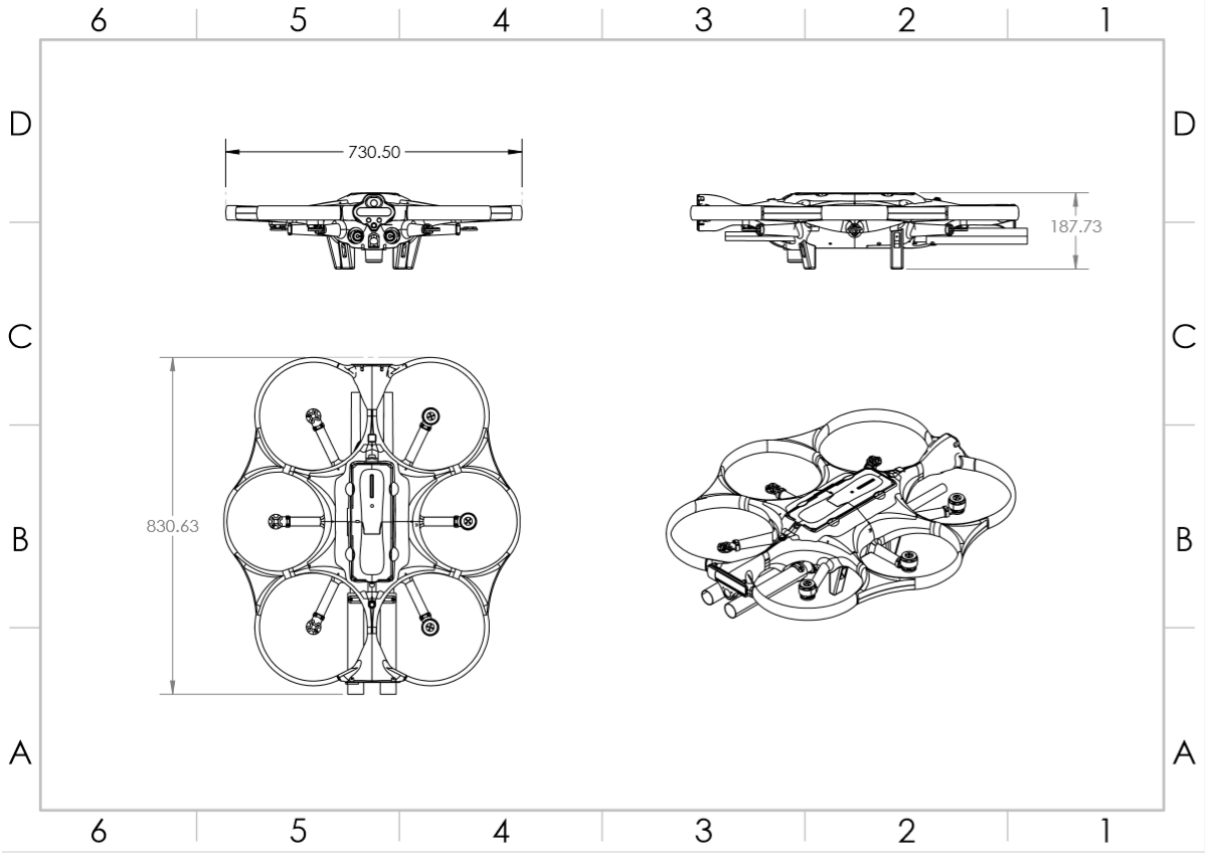


Figure 3.1 Major dimensions of WE-01

Within the inner core the arrangement of internal hardware is separated into three major sections. The central section is occupied by the Power Distribution Board, vertical and rotating LiDARs and the payload in the bottom part; a carbon fibre H frame attached to the body via aluminium brackets has the double function of attachment point for the payload and it works as support for the battery which is hosted in the top half of the same central section.
The right section (Flight Controller section) includes the Flight Controller (FC) and Trigger Board. The left section instead (TX2 section) includes the on-board companion computer (Nvidia Jatson TX2), the Light Board and VTX. A top view of these internal components is shown Figure 3.2.

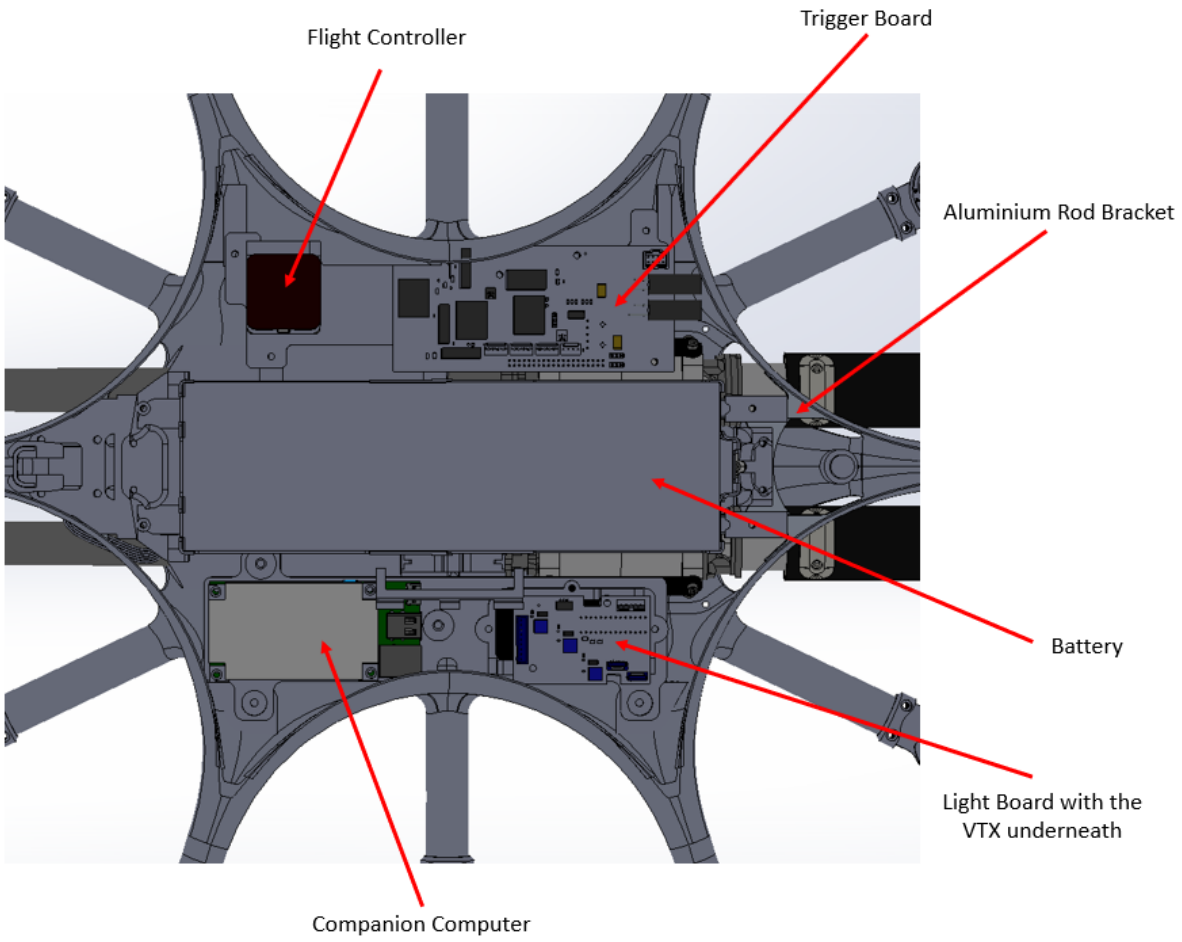


Figure 3.2 Internal structure arrangement - Top View

Other hardware in the monocoque are the ESCs, which are fitted inside the protrusions leading to the Prop Guard Arms, the GPS and the Key Switch; these last two are located on the top of the body along the centreline, symmetrically in the front and the back portion of the core body.
The hardware in both the FC and TX2 sections are mounted on stiff carbon fibre sandwich plates linked to the body and to the central support rods (H frame). The FC section plate is hard mounted to its supports whilst the TX2 section plate is mounted on rubber dampers to protect the Companion Computer against shock loads.

The effector is mounted to the structure using an aluminium clamp. The combination of the effector and the clamp composes the Payload. The clamp is a damped rail system constrained and supported, as mentioned earlier, to the cross-rod of the H frame and to the rear aluminium bracket. The damping system within the clamp is meant to alleviate the shock loads experienced upon firing. Details about these components are shown win their respective locations in Figure 3.3.

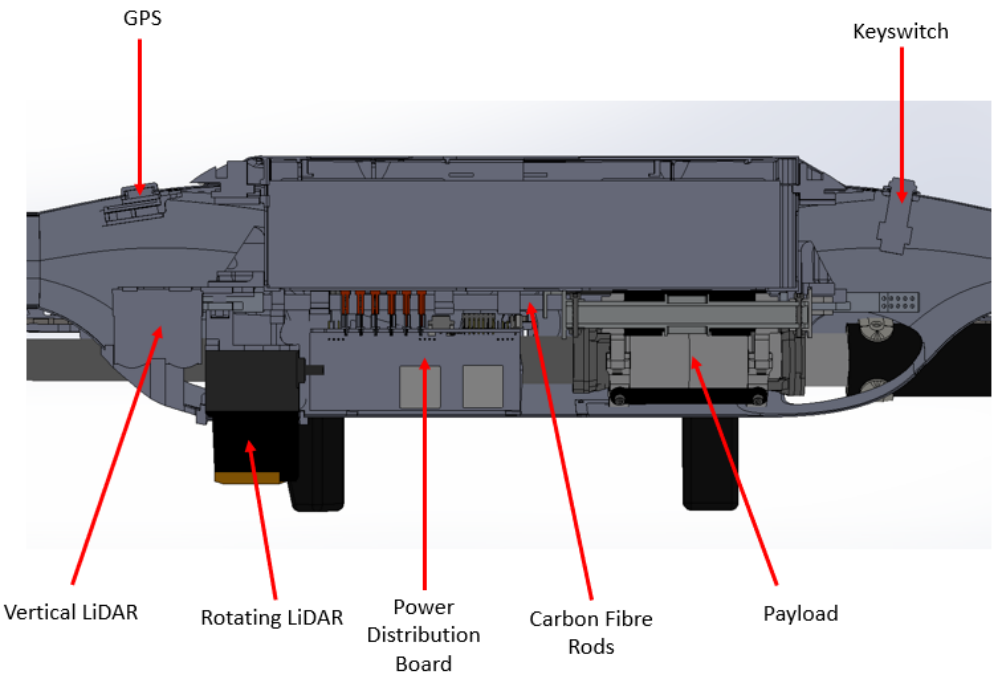


Figure 3.3 Internal arrangement - Side view

The barrels run inside the body through two protective tubes which physically separate the payload from the electronics.
The battery is protected by a plastic case which includes openings to accommodate latches to secure it during flight. It is further secured in the lateral directions by stoppers placed on the front and rear of the craft.
The LiDARS are mounted on nylon supports to the CF rods and supported around the openings arranged for them to reduce the likelihood of impact damage.

The front camera module houses two cameras respectively for navigation and aiming. The FPV camera used for navigation is placed at the above the Intel Realsense D435, which is instead the depth camera used for aiming. Two lights, one visible and one infrared, are located underneath. Finally, the aiming laser and its collimation mechanism are placed at the lowest point. The systems are held by a plastic mount screwed to the carbon fibre module (see Figure 3.4).

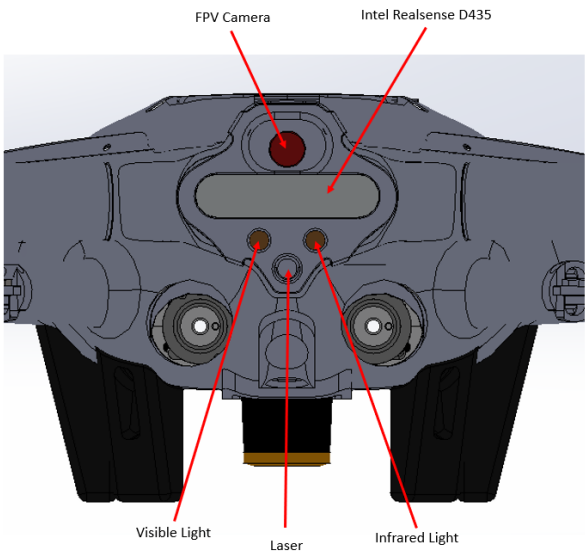


Figure 3.4 Front Camera Module

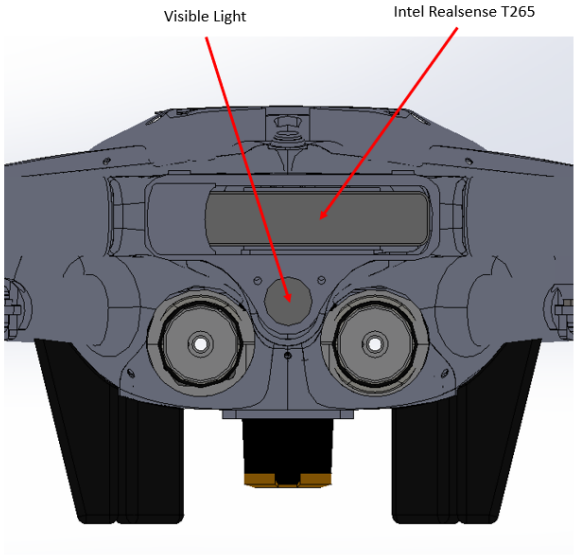


Figure 3.5 Rear Camera Module

The rear module houses the Intel Realsense T265 and a visible light which is used to enhance the visibility in low light condition (see Figure 3.5). The camera is use to acquire and process Visual Inertial Odometry data and it is fitted with internal Inertial Measurement Unit (IMU), hence it is mounted on a damping system. Plastic mounts fixed with screws secure the sensors to the camera module.

More structural details will be described in Section 4.1.

Section 3.2 Power and Data

In this section the architecture of the electronic components is presented. The schematics of these components can be classified for simplicity in Power and Data. The former diagram is shown in Figure 3.6; here the power distribution board (PDB) acts as a hub that provides and regulates the power coming from the main source of energy, which is the battery. As shown in the diagram there are four levels of voltage: battery voltage (25.5V) for ESCs and trigger board, 12V for Rotating Lidar and Companion Computer, 7V for the light board and 5V for the rest of the components.

Figure 3.7 shows instead the equivalent diagram for the Data Lines. In this case the hub is represented by the combination of Flight Controller and Computer.

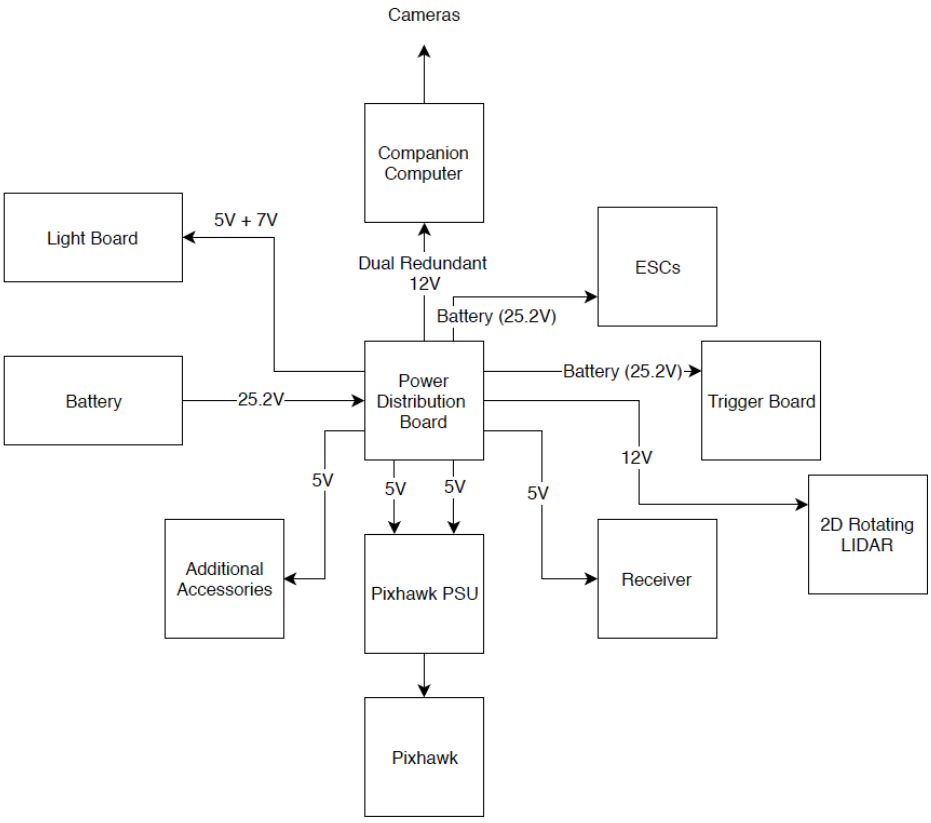


Figure 3.6 Power Diagram

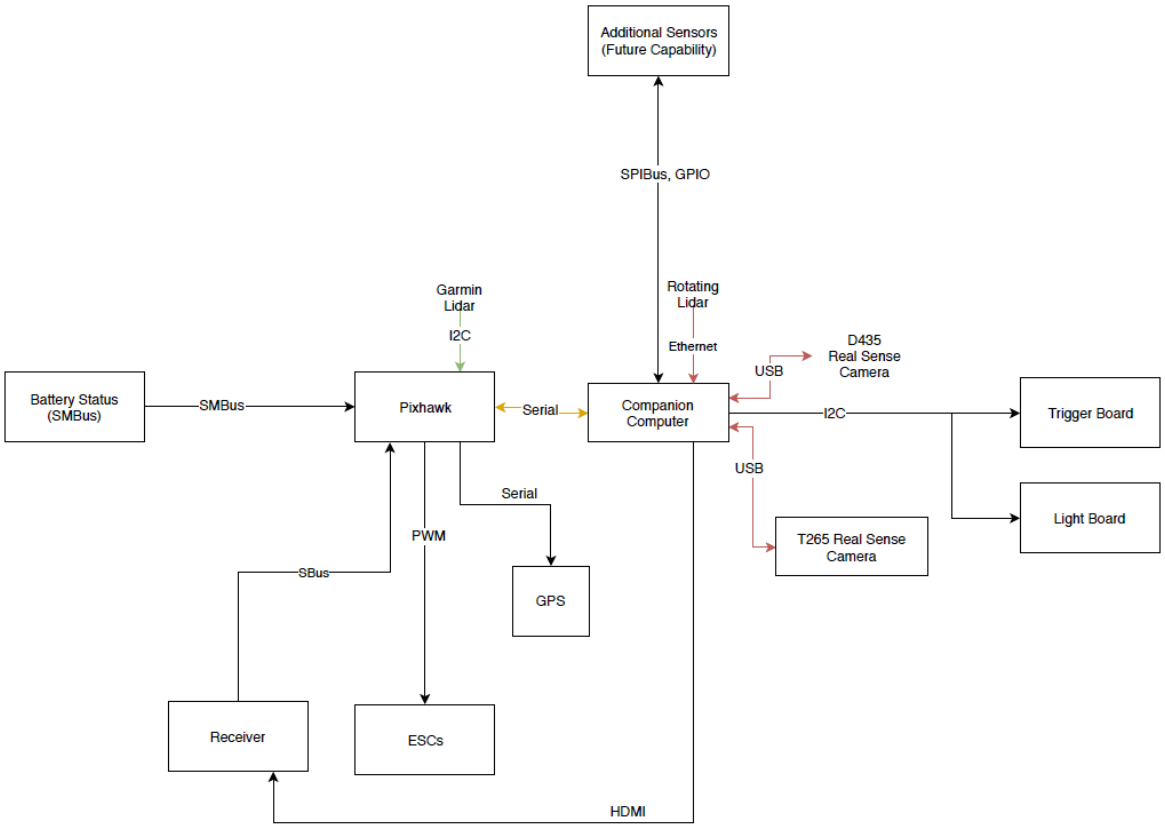


Figure 3.7 Data Diagram

Section 3.3 Control and Aiming

In this section the architecture of the general control strategy is shown, with particular attention to the interconnection among the different sensors and algorithms. Figure 3.8 represents the complete architecture of the system from the control point of view, i.e. platform and transmitter. Within the Platform box two main sub-section can be identified: Pixhawk (Flight Controller) and Companion Computer. These two communicate through MAVLink (Micro Air Vehicle Link), which is a protocol for communicating with small unmanned vehicle. The software running on the FC is called PX4 and this section of the platform is where the low-level operations are done with the ultimate goal to communicate to the motors how to adjust in order to achieve the setpoint (wanted action). In order to do so internal sensors (IMU) and an Estimator are present; the internal estimator's main function is to collect raw data coming from different sensors (IMU, 1D Lidar, GPS) and fuse these sources of information in order to get an output on which the final action is based. Another benefit of an estimator is to reduce the noise that each sensor data set will have, such that the output data can have a more "deterministic" and accurate estimation.
The second block concerns instead the Companion Computer; Figure 3.8 shows an "aiming" section on the left, where the logic used to achieve the targeting is explained, and a main area which deals instead with the main functionalities of the drone during normal flying operations (e.g. stabilization via external sensors, position holding, etc.). It is worth noting that still within the Platform box, but external to either of the previous subsections there are ESCs and Motors, which constitute the downstream components of the propulsion system and the Trigger Board. This is a piece of hardware that takes care of operating the activation of the payload (as it will be explained in the following sections) and it embeds the logic for its functioning and safety, thanks to a set of microcontrollers. The trigger board, in a form of a PCB, communicates with the main computing resource of the platform (Companion Computer), where additional logic for the payload activation is embedded with additional safety features.

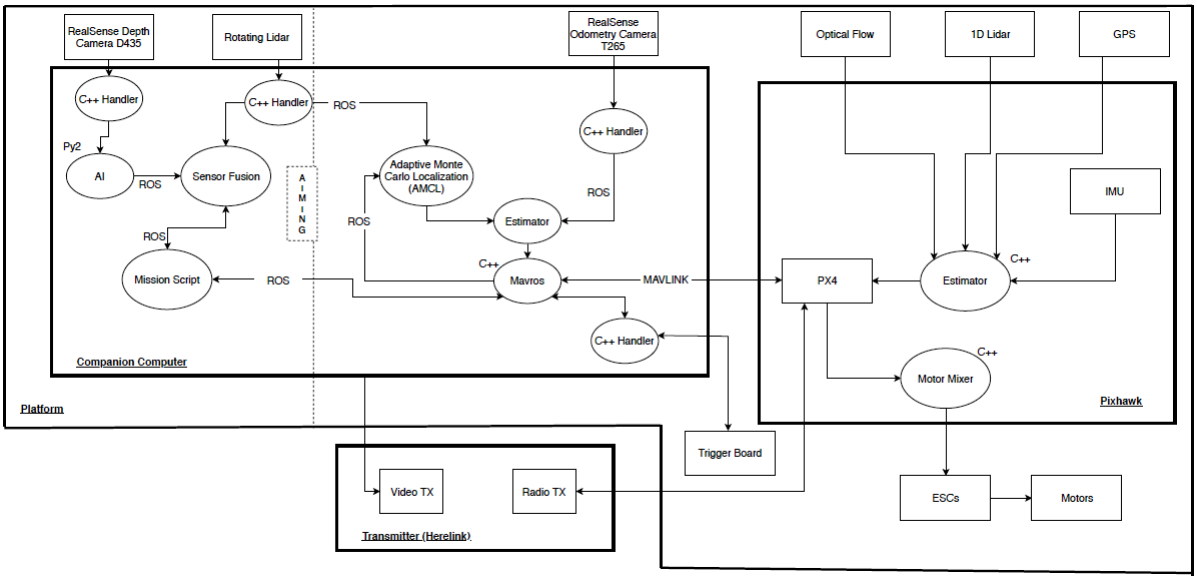


Figure 3.8 General Control Strategy

Finally, the Transmitter/Receiver completes the architecture. The commercially available Herelink is used, where the video signal is transmitted from the Companion Computer and the controls input are fed into the Flight Controller. This component also provide encryption for control and video communication from and to the air platform.

The following *Figure 3.9* Figure 3.11 show details respectively of the AI, Sensor Fusion and Estimator flow charts. These are blocks that can be found in Figure 3.8.

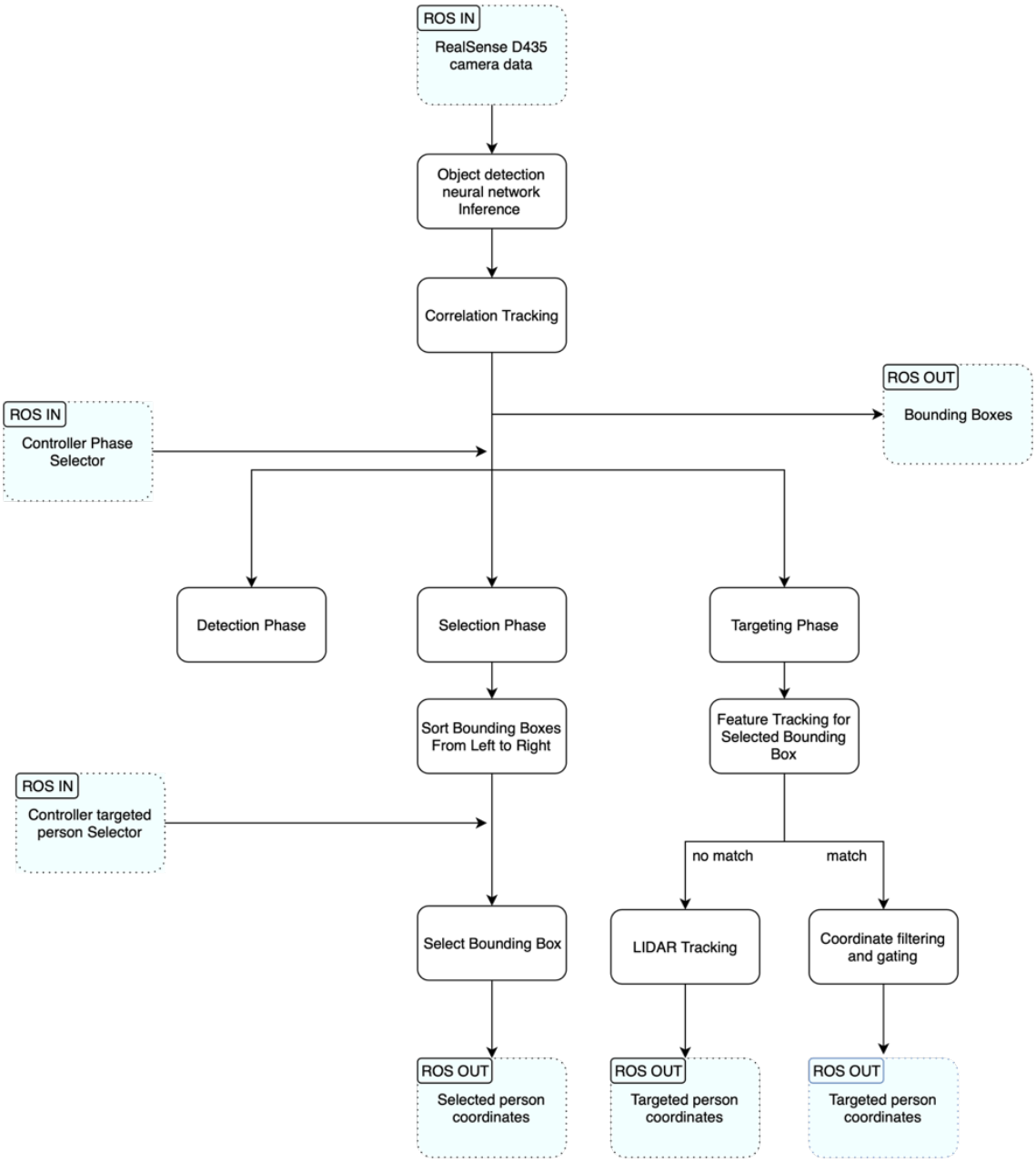


Figure 3.9 AI Block Diagram

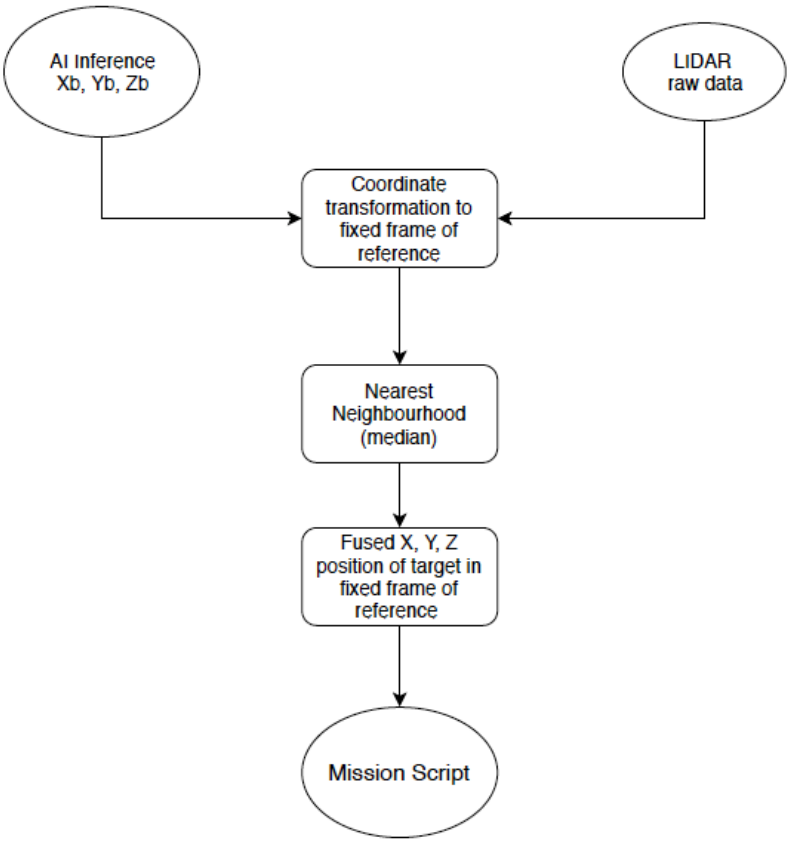


Figure 3.10 Sensor Fusion Diagram

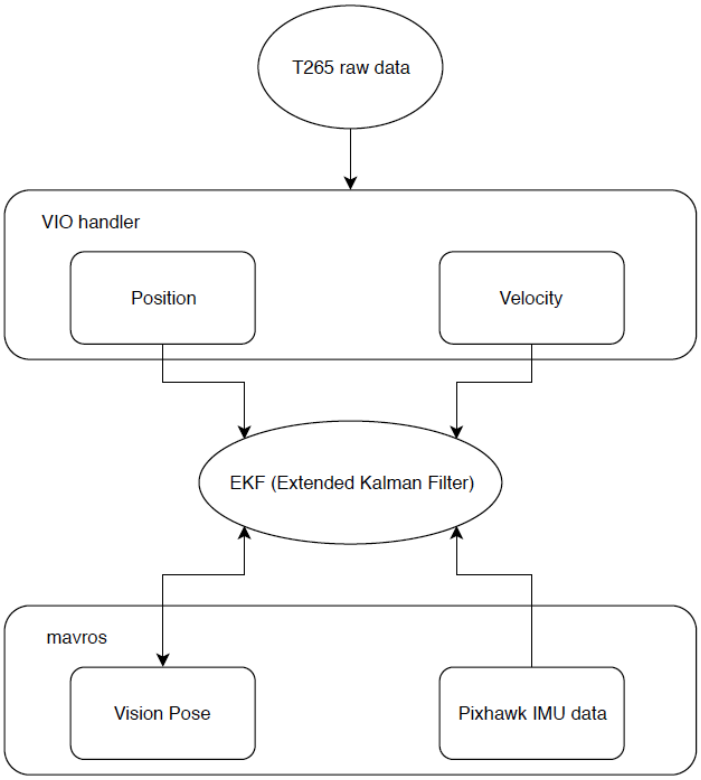


Figure 3.11 Estimator

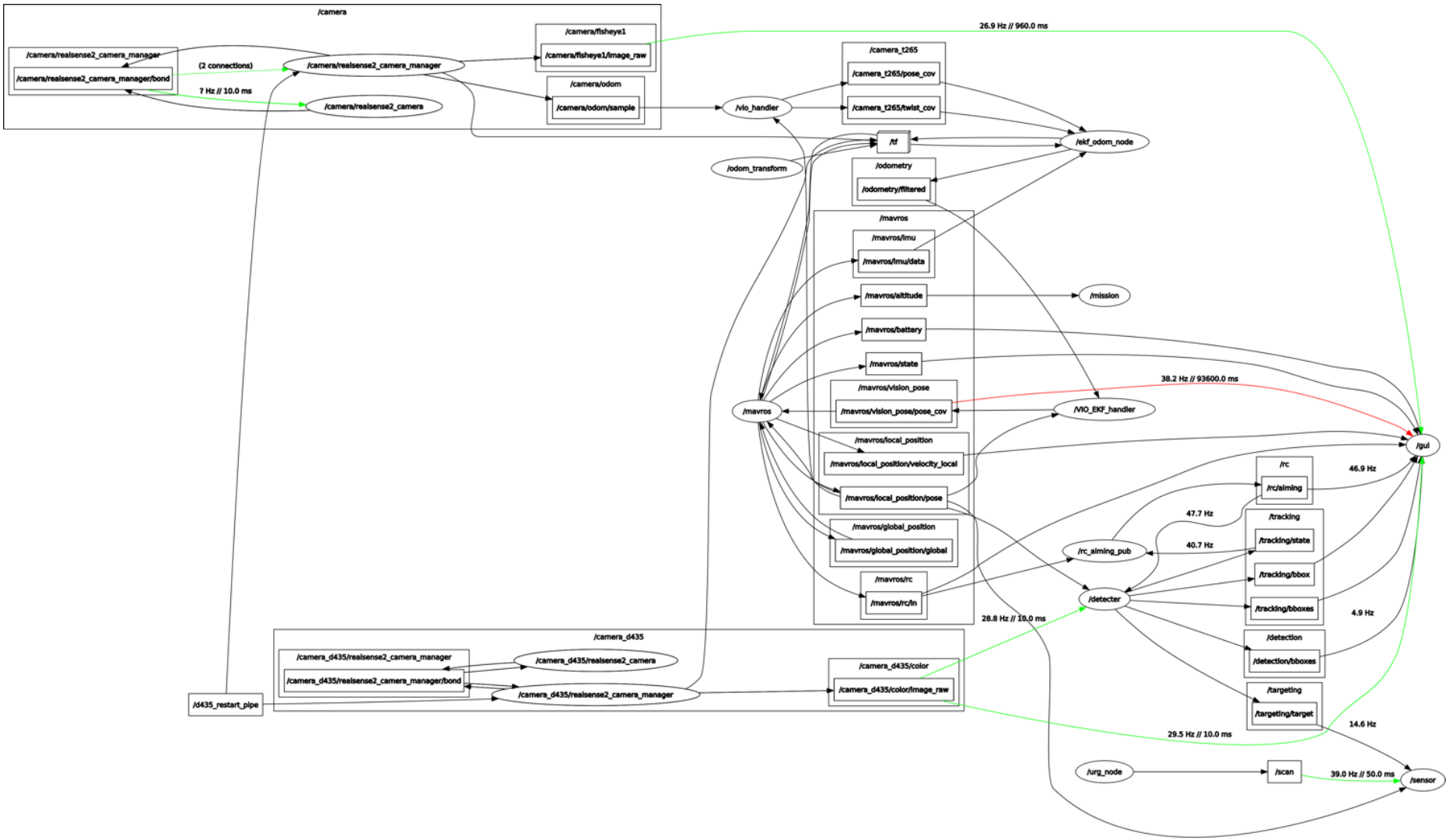


Figure 3.12 ROS Architecture diagram.

Figure 3.12 shows the ROS architecture internally created by the software, displayed here for reference.

Section 3.4 Effector and Trigger Board

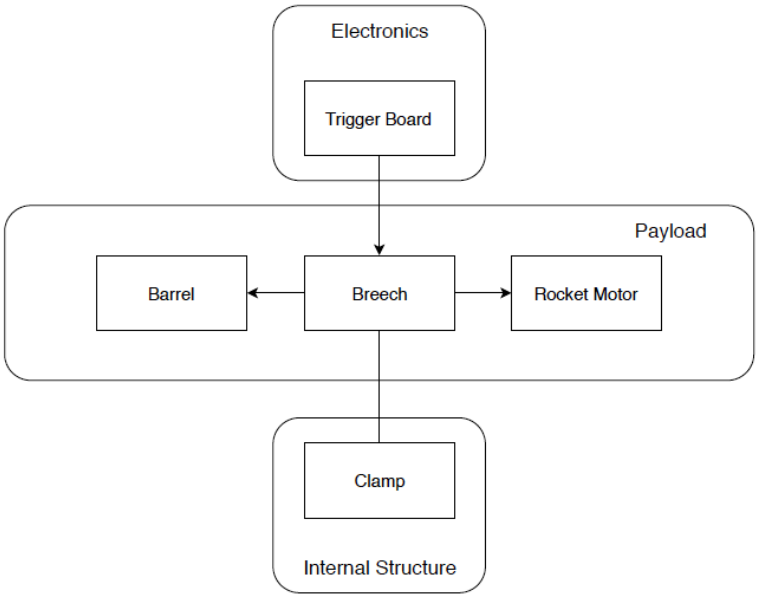


Figure 3.13 Effector Diagram

In this section the conceptual diagrams of the elements composing the payload are shown.
Error! Reference source not found. shows, very simplistically, the relationships among the different sections of the effector: barrel and rocket motor are physically attached to the common breach block, which is physically connected to the UAV body through the clamp. This is constituted by an aluminium shaft rigidly attached to the H frame of the internal structure (as mentioned in Section 3.1), and clamping mechanism made in two halves to host the breach block. The top half of the clamping mechanism is meant to slide on the rigidly mounted shaft. Hence, the damping system mentioned in Section 3.1 controls the relative speed with which these two components slide on each other and reduces the force transmitted to the rest of the body, due to the remaining net recoil force (Figure 3.14).

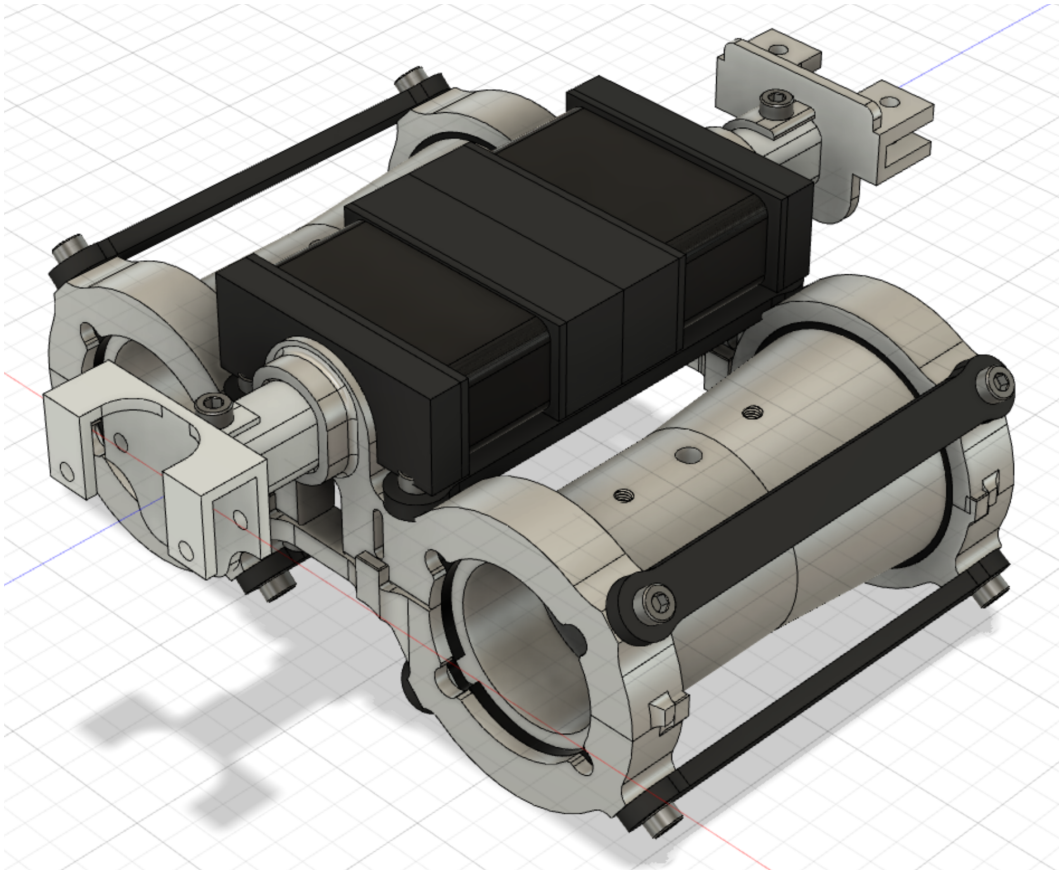


Figure 3.14 Breech block assembly with damping system

Since the ignition of the effector's cartridges is electrical (thanks to the use of EEDs), the breech block is electrically connected to the Trigger Board. Figure 3.15 shows the conceptual block diagram describing the Trigger Board.

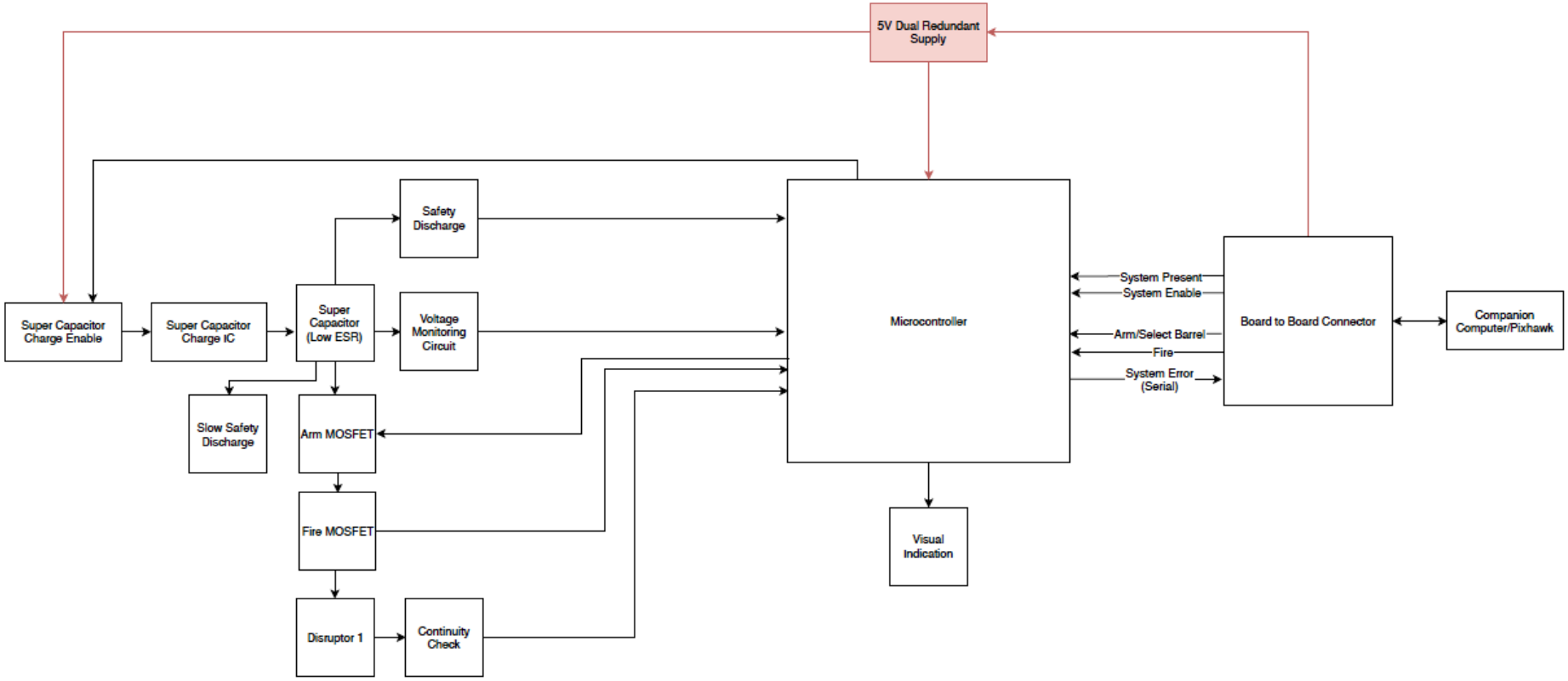


Figure 3.15 Trigger Board Diagram

Chapter 4 Platform Subsystems

In this chapter a deeper explanation of the single components is given, highlighting the critical aspects of the design either in terms of functionality or safety.

Section 4.1 Structures

The main layout of the external structure can be found in Figure 4.1. The Prop Guards are linked to the Main Body through the Prop Guard Arms. The Prop Guards and Prop Guard Arms are designed to obey to a fail-safe design philosophy, hence in case of impact, they would absorb the shock and in the eventuality of structural failure (damage nucleation or critical failure) the integrity of the Main Body would be preserved. The elements are joined through overlapping sections secured by fasteners.

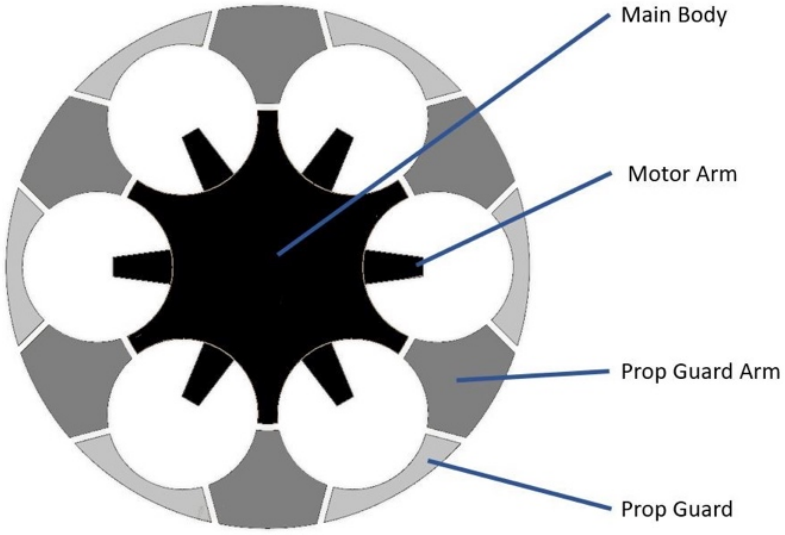


Figure 4.1 Layout of the external structure

The Main Body, differently from the outer protective structure, has been designed to obey to a damage tolerant design philosophy. It is a structural shell (monocoque) that serves as the main structural component as well as protective shell from external actions (loads), perpendicular to the body itself. This approach makes sure the load is distributed (spread) across the whole structure such that no single point of failure is present in the structure.

Composite material is used for the manufacturing of the body, so that the material is actively part of the whole design. Composition, material orientation and thickness of the shell are tailored to the purpose of the components.
Regarding the material composition, carbon fibre reinforcement with thermo-set epoxy matrix is used. Thickness varies throughout the body accordingly to the stress level of the region under consideration: the thickness of material varies from 1mm in the motor arms and the sections leading to the Prop Guard Arms to 0.5mm in non-load bearing sections such as the lower "belly". For the sake of completeness *Figure 4.2* shows the lay-up map for the different sections of the main body.

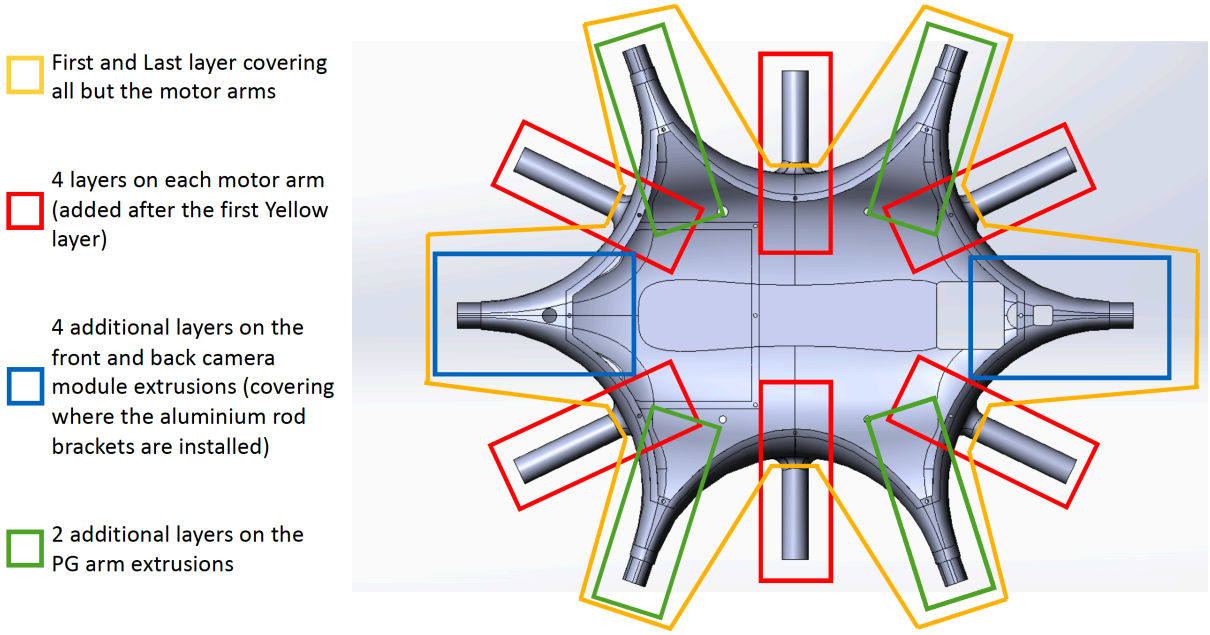


Figure 4.2 Body lay-up map

The Main Body is moulded in two halves which are then joined using a carbon fibre strip.
Two maintenance lids are built into the body to allow for access to the internal components. The top lid gives complete access to the electronics whilst the lower opening is used to install the payload clamp. The lids are secured during normal operation using M3 screws.

The same material of the body is used for the Prop Guard Arms (including the front and back camera modules) with thicknesses varying from 0.5 to 1mm. Like the body, they are fabricated in two parts which are then joined using an incorporated lip, which ensures some overlapping of the two halves and hence enhanced strength.

The Prop Guards are a combination of aramid/epoxy and carbon fibre/epoxy composite for improved impact performance. Rohacell foam ribs are added during manufacturing for the double purpose of increasing stiffness and aiding the manufacturing process. The graphical representation of the composition is shown in Figure 4.3.

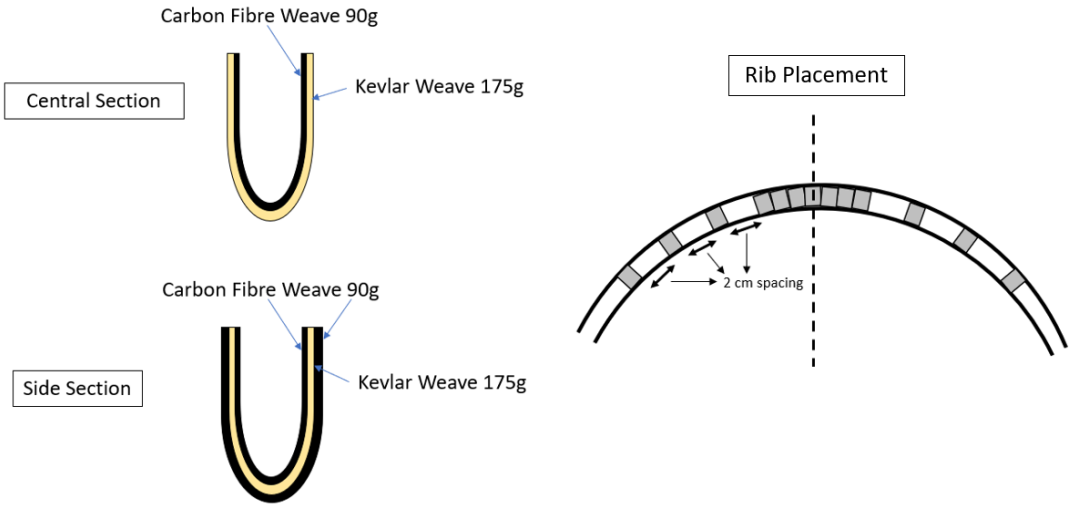


Figure 4.3 Prop Guard Composition

It is worth noting that the current design of the propeller guards has been achieved thanks to an analytical approach followed by validation through a bespoke numerical FEA code and experimental tests carried out at Wecorp. The flowchart of the design and manufacturing approach is shown in Figure 4.4. This result has been achieved during a Master Final Year Project by Ms Alix Berlizot (Ecole Centrale Nantes) during an internship at Wecorp: the title of the project was *"Design and manufacturing of shock and impact resistant outer structure for drone operating in confined enclosed conditions"*.

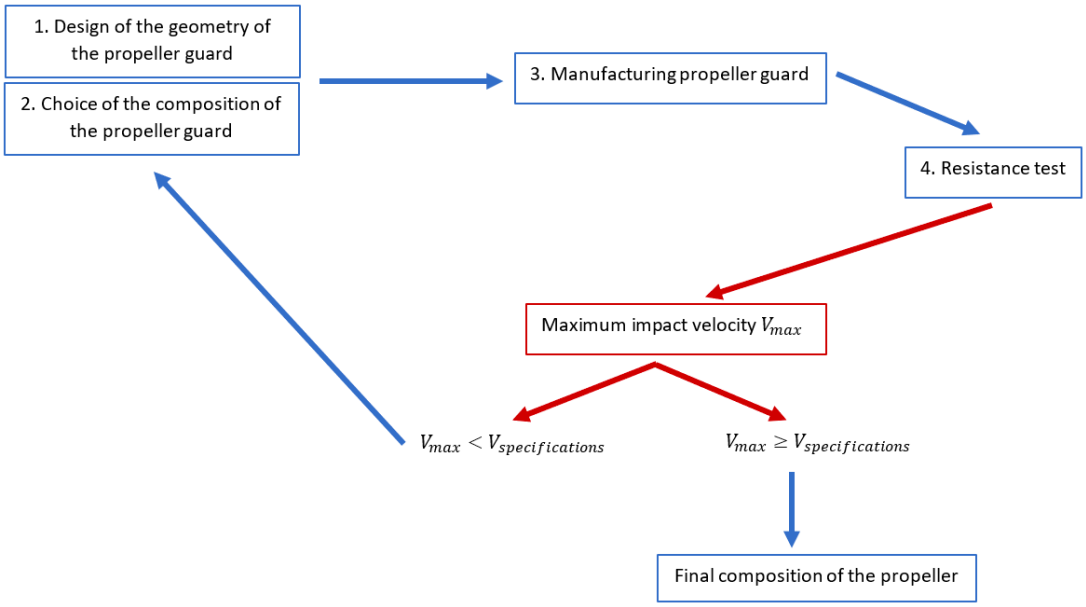


Figure 4.4 Propeller Guards design and manufacturing process flowchart

The Prop Guards have been studied as curved circular beams where, contrarily to the normal Euler-Bernoulli beam formulation, coupling between axial and bending actions is guaranteed. The formulation takes inspiration from non-linear or buckling analysis of beams. The aim of the analytical formulation was to establish a relationship between the geometrical properties of the prop guard as well as the cross-section parameters (drastically affecting the second moment of inertia) and the stiffness of the structure. The numerical FEA, based on the aforementioned analytical relationships, allowed to obtain quick comparisons among different geometrical parameters. Finally, the experimental testing validated the resistance to different impact speed; since the model was built considering steady-state loading condition, a pendulum test has been devised where the potential energy at different heights is considered to be transformed entirely into kinetic energy at the impact point. The following relationship has then been considered (see *Figure 4.5*):

$$h = \frac{v^2}{2g}$$

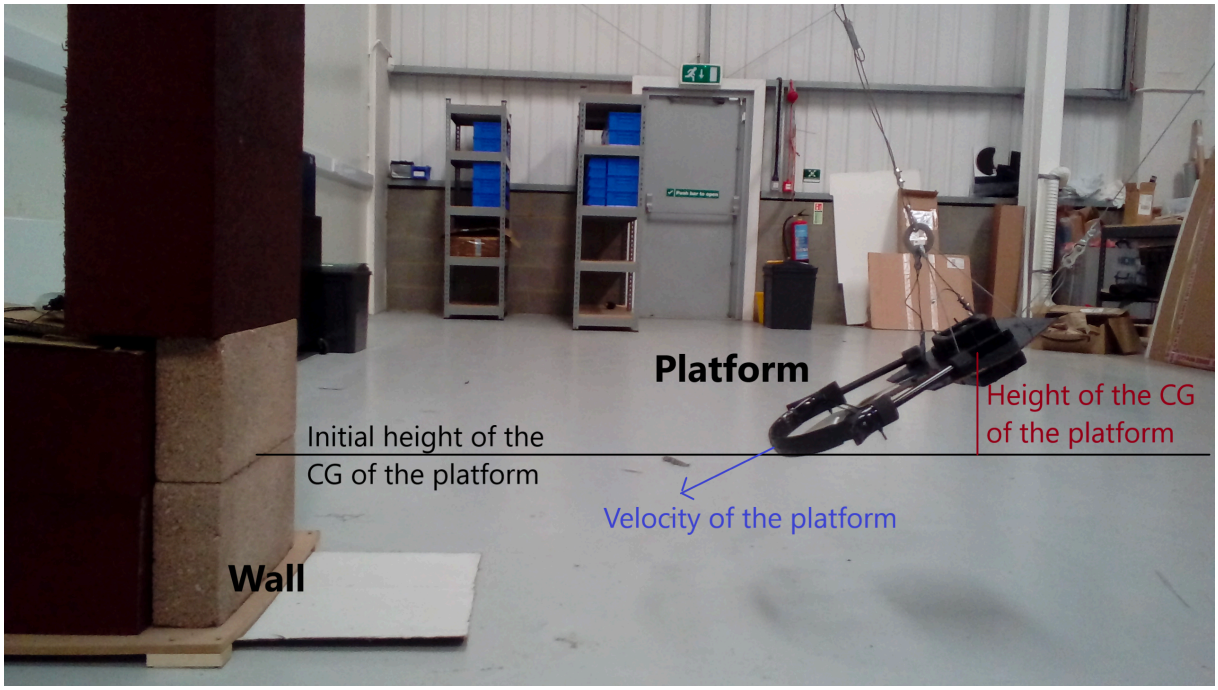


Figure 4.5 Test rig for impact resistance of prop guards

The kinetic energy upon impact has been considered to be injected into the system and distinguished into three main components: 1) energy absorbed through the work done by the prop guards' deformation; 2) energy elastically transmitted to the rest of the structure; 3) energy dissipated through the formation of new crack surfaces (in case of fracture), expressed in terms of energy release rate. In order to achieve a successful design, the relative difference among these three energies must guarantee the following goals: 1) the shock in not entirely transmitted to the rest of the body and hence to the electronics; 2) the propeller guards do not deform inwards more than the clearance between their internal surface and the tip of the propeller (6mm); 3) the propeller guards do not dissipate the impact energy entirely via critical failure.

Table 4-1 Resistance test results for the elliptical section

Propguard denomination	weight	finish	maximum impact velocity at break
E1235-10-CFK1	28g	g, ns	2.5m/s
E1235-10-CFK2	26g	b, ns	2.5m/s
E1235-10-CFK3	23g	g, s	2m/s
E1235-10-CA	24g	g, s	2.5m/s

Table 4-1 shows the summary of the test results for the elliptical section, where the *g* indicates good surface finish for production, *b* stands for bad, i.e. not acceptable for production and *s* and *ns* respectively shiny and non-shiny. The chosen composition is the last in the table and the nomenclature *E1235-10-CA* stands instead for Elliptical section, 12mm width, 35mm height, 10'' diameter, Carbon Aramid. This shows the highest impact velocity at break, it does not critically fail at first 2.5m/s impact and it is the second lightest composition, hence a good design point for resistance and lightness.

The four landing gears are fabricated from 3D printed TPU to provide suspension during landing. The gear is fixed directly onto the main body through a combination of screws and epoxy.

The six motors of the UAS are fastened to 4mm thick carbon fibre mounts and damped with a TPU plate. The plate is attached with removable screws to a nylon adapter fixed with epoxy to the Main Body. These mounts make sure that the thrust of each motor is transmitted to the main body as a steady-state (no vibration) perpendicular force to a cantilever beam, where the "clamped" section of the beam is considered to be the junction between the arm and the body. The arm has then been designed mainly to resist to bending moment. The arm has a variable cross section from root to tip: this means that the arm is tapered from elliptical (with the major axis being the vertical) to circular, such that the stress distribution along the arm is as uniform as possible and hence the weight is minimized. The reasons behind the choice of elliptical and circular hollow cross-sections are two: 1) since the force is applied due to high speed rotation of a propeller, torsion transmitted to the arm cannot be neglected and hence a closed section has to be chosen; 2) a closed hollow section allows to protect electronic components (ESCs) and wires that connect the motors to the main body.

Section 4.2 Propulsion

The propulsion system consists of four main elements:

- **Propellers:** the current system uses a combination of four 10'' two-blades and two 9'' tri-blades propellers produced and sold by Master Airscrew.
- **Motors**: the motors are T-motor U3 700 KV and they are the same for 9'' and 10''.
- **ESCs:** T-motor Air 40 Amp are used for the six motors.
- **Battery:** the battery pack is assembled by Wecorp who buys the individual cells from the manufacturer Kokam Co., Ltd. These are ultra-high-density energy (248 Wh/kg) lithium polymer (LiPo) battery cells that allow to have a high energy stored into the system (nominal 96.2Wh per cell) with a significant weight saving compared to common LiPo batteries (180 Wh/kg).

The consequence of this very high efficiency of the battery is the maximum discharge rate that the battery can bear (2C nominal, 4C peak) which is very low compared to the typical 20C of a common LiPo for UAV applications. However the WE-01 has a very unique field of application and thanks to the bespoke design of the propulsion system (flowchart shown in **Error! Reference source not found.** Figure 4.7) the system is within the current draw: 52 Amp in nominal conditions and 104 Amp for peak, i.e. within 2C and 4C for a 26000mAh battery.

The battery pack has a 7S configuration, i.e. seven cells in series; this would in theory provide to the system a minimum of 21V and a maximum of 29.4V (25.9V nominal voltage). In reality, by monitoring the voltage level during flight it has been noticed that the loaded system shows a significant drop of voltage, almost equivalent to one cell (≈4V) upon take off; this leaves the system under and effective voltage of 22.2V, i.e. a 6S configuration.
The benefit of this approach consist in increasing the voltage level the system is subjected to such that the same power requirement is achieved with less current draw, having then a more efficient system (less energy dissipater into heat) and ensuring that the design meets the battery requirements (2-4C).

This unique approach to the battery pack design could in theory be applied to other configurations (8S, 9S, etc) if the requirements could accommodate for the extra weight in the system. However, weight per se is not the only limitation because the system shows already a saturation behaviour

with the 7S pack. In other words, beside the weight increase that an 8S would cause, the system would actually become less efficient.

Finally, the achievement of the current design, as shown in the flowchart (**Error! Reference source not found.**), required the creation of a database with different combination of the element composing the propulsion system (propellers, motors, ESCs and batteries). The performances of each system are then analysed in a bespoke Matlab code where constrains are applied in order to achieve a number of combinations meeting design criteria. The basic form of the code optimises for hover conditions, while the portion of the flowchart shown in Figure 4.7 shows the possibility to input the parameters of a specific mission (in terms of horizontal and vertical velocities and the durations of the respective flight conditions). New optimized solutions among the existing combination in the Wecorp's database will be then given as output and each with the estimation of the respective flight time.

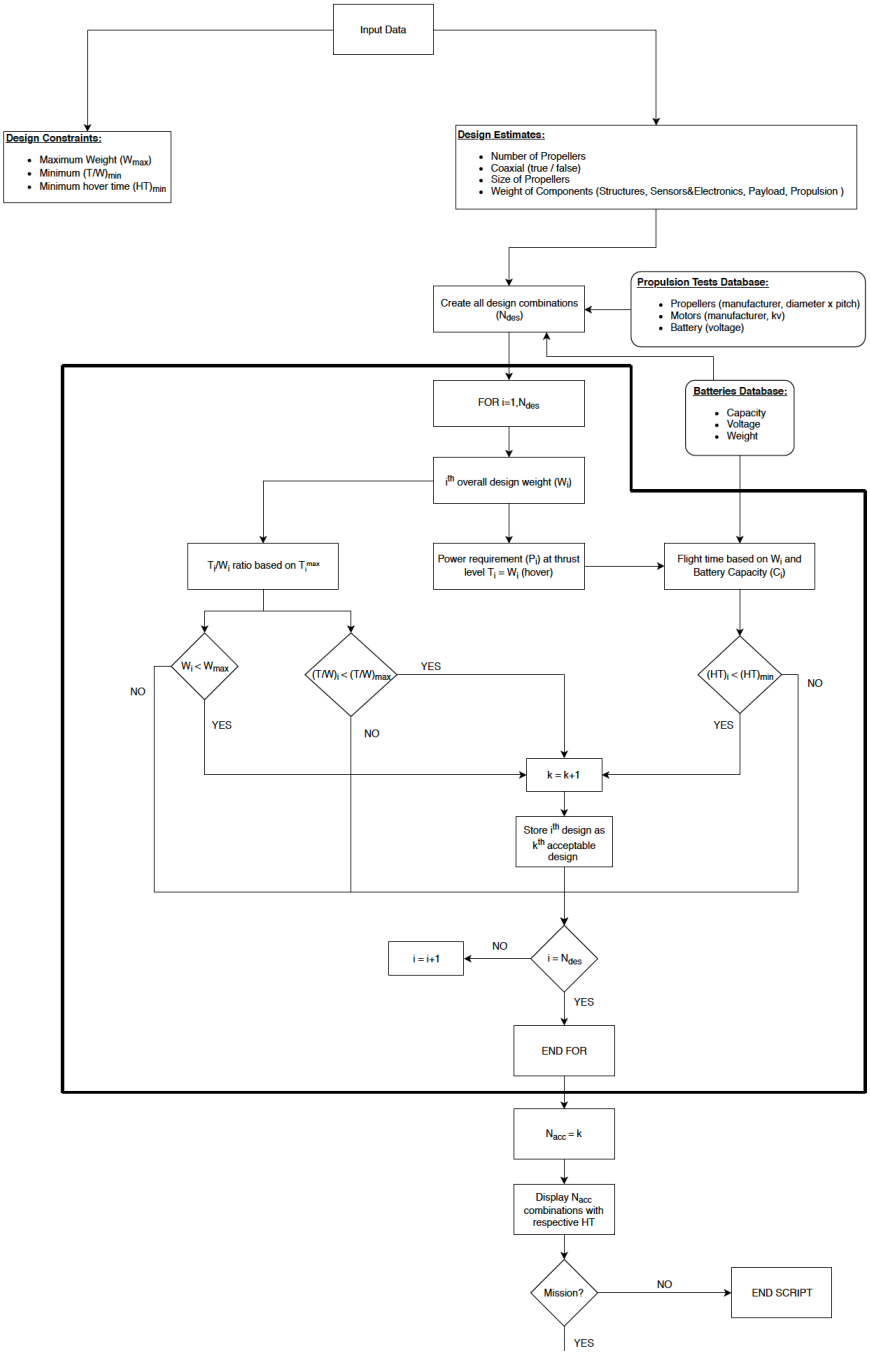


Figure 4.6 Flowchart of propulsion system design

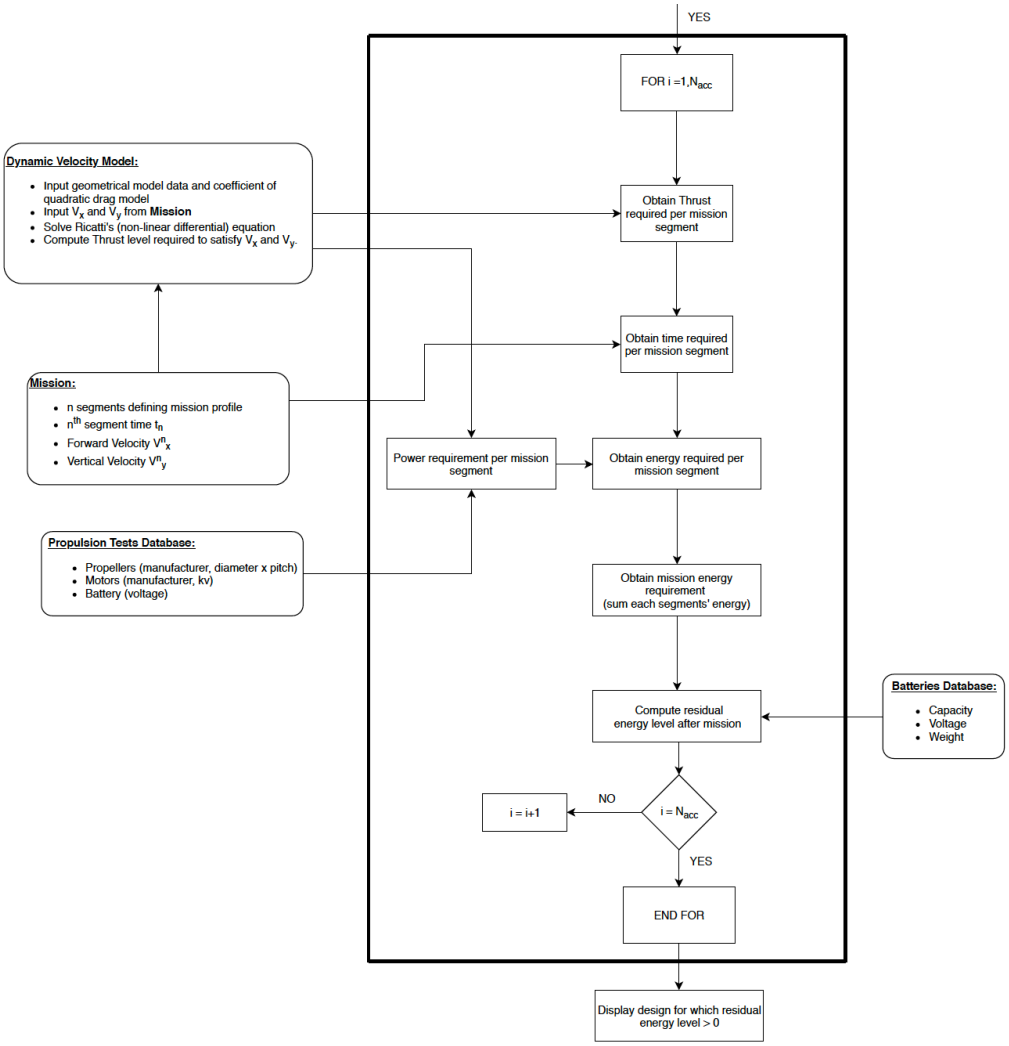


Figure 4.7 Flowchart of propulsion system design – continued

Section 4.3 Electronics

The on-board electronics consist of several printed circuit board (PCB) electronic circuits (PECs) that interconnect the control system of the platform to the various sensors and the propulsion system. The key electronic components are:

- Cube flight controller/autopilot
- Nvidia Jetson TX2 embedded computer
- Hokuyo Rotating Lidar
- Intel D435 RealSense tracking camera
- Intel T265 RealSense tracking camera
- Garmin LIDAR Lite v3
- Trigger PEC
- Light Control Circuit

These components are explained in detail in the following sections.

Flight Controller/Autopilot

The motors on the platform are directly controlled by commands from the Cube autopilot. The Cube is a flight controller and autopilot that is running the custom PX4 flight stack on the NuttX Operating System.

The Cube includes a triple redundant Inertial Measurement Unit (IMU) consisting of:

- 3x triple axis accelerometers
- 3x triple axis gyroscope
- 3x triple axis magnetometers
- 2x barometers

The IMU is dampened to improve stability. The cube also features redundant power supply inputs and automatic failover.

Companion Computer

The platform consists of an NVIDIA Jetson TX2i embedded computer mounted on an Auvidea J120 carrier board. The carrier board provides interfaces into the Jetson computer.

The following interfaces are used.

Table 4-2 Interfaces used on the Nvidia Jetson TX2 computer

Interface	To	Function
USB 3.0	Intel D435 RealSense tracking camera Intel T265 RealSense tracking camera Wide Angle Camera	Depth Tracking Visual Odometry Main Flight Camera
UART	Pixhawk Cube 2.1	MavLink Telemetry and Control
I2C	Trigger PEC Light Control Circuit	Control Log errors
Ethernet	Hokuyo Rotating Lidar	2D mapping
HDMI	Video Transmitter Unit	GUI

The NVIDIA Jetson TX2i is a low power computer often used for robotics applications where the onboard GPU can be leveraged for certain computing applications. The 'I' designator shows that this it is a device designed to handle more shock, vibration and extreme temperatures compared to their standard models, making this device more appropriate for the use case.

Figure 4.8 Nvidia Jetson TX2i

Auvidea J120 is the smallest carrier board usable while retaining the bare minimum IO required. However, it will be succeeded by the Connecttech Quasar Carrier Board. It benefits of a much larger operating temperature range of -40 to +85 degrees Celsius, matching the capabilities of the TX2i. Lastly it is more tolerant to different voltage levels, ranging 9 to 19 Volts compared to the fixed 12 Volts of the J120.

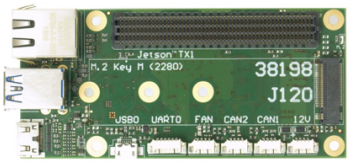


Figure 4.9 Connecttech Quasar.

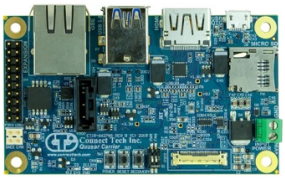


Figure 4.10 Auvidea J120

Sensors

The Platform hosts a multitude of sensors, which have been perfectly selected tailored to the application

- **Hokuyo UST-20LX:** This unit is a 270-degree FOV Lidar developed by Japanese company Hokuyo. It generates 2-dimensional point cloud data that can be used in mapping, localization and object/environment modelling.

The sensor rotates at 2400 rpm, with a scan speed of 25 ms, and can achieve an angular resolution of 0.25 degrees. The device has a range of 20m under normal conditions. It has been designed to be very robust, since all the mechanisms are enclosed within its protective shell, allowing for the device to experience 20g in all directions ten times before failure occurs.

The sensor consists of a class 1 laser designed and tested to IEC-60825.

- **Intel RealSense Tracking Camera T265**: The RealSense Camera T265 is a Simultaneous Localization and Mapping (SLAM) device that can build a map of an environment by keep tracking its own location. The T265 includes two 170 degrees fov fisheye lens sensors, an Inertial Measurement Units (IMU) and an Intel Movidius Myriad 2 VPU. The camera outputs an estimate of its position and orientation in 3D space with an accuracy of < 1% under optimal conditions, at a publishing rate of 200Hz with sub 6ms latency.

- Intel RealSense Depth Camera D435: The RealSense Camera D435 is a camera that provides a field of view of 87 degrees and global shutter sensors, which allow applications to navigate in low-light environment. The D435 includes Left& Right Imagers with 1080p image sensors (FOV- H:91.2/V:65.5/D:100.6), Infrared Project with class 1 laser compliant (FOV-H:100.4/V:69/D:110.4), and colour camera with 1080p RGB image sensor (FOV-H:69.4/V:42.5/D:77). It has a range of around 10m allowing it to perform well at object tracking.

- USB Camera ELP-USB3MP01H-L170: The ELP camera is a camera with WDR (Wide Dynamic Range) function that can be used in any light condition. The camera contains H.264/MJPEG/YUY2 High profile compression, wide angle 170-degree lens, and can provides the images with maximum resolution of 1920(H)x1080(V) and maximum frame rate of 30FPS. This gives the operator a large FOV allowing greater spatial awareness. Due to the global shutter it will eliminate Shutter Roll caused by rapid panning of the camera.

- Garmin Lidar Lite V3: A Compact Optical Range finder used to take altitude measurements for the platform. With a range of 0 to 40m and accuracy of 2.5cm. This Time Of Flight Laser Range finder updates at frequencies as high as 500hz.

The sensor consists of a class 1 laser designed and tested to IEC-60825.

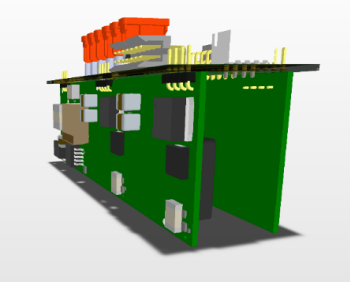
Power

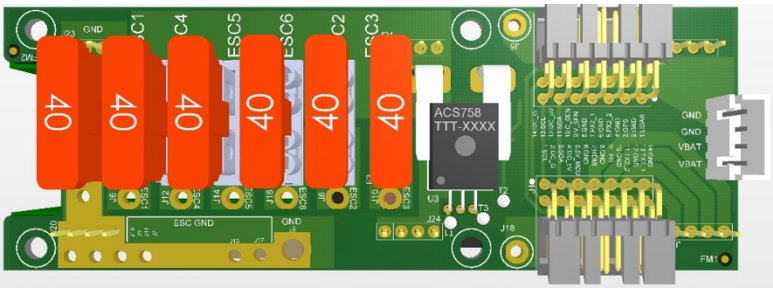
Below are the power requirements of the electronic components of the platform.

Table 4-3 Power requirements of electronic components

Component	Voltage (Volts)	Current (Amps)	Power (Watts)
Cube	5	2.75 (max)	13.75
Nvidia Jetson TX2i	12		20
Hokuyo UST-20LX	12	0.45	5.4
Intel D435 RealSense tracking camera	5.25	0.7	~4
Intel T265 RealSense tracking camera			
Garmin LIDAR Lite v3	5	0.135	>1
Trigger PEC	Battery		
Light Control Board	5 + 7		
Wide Angle Camera	5		
Motor (for overall consider x6)	Battery	37	958.3

Power is delivered to the components from a Power Distribution Board (PDB) which contains buck controllers that step down the battery voltage to the required level. The buck controllers are based on the Texas Instruments LM2596 Integrated Circuit (IC). The PDB also houses the primary current sensor, capable of measuring a peak current of 178 Amps. It is also fitted with automotive fuses to protect the ESC against a current surge often caused by stalled motors, saving both the ESC and motor in such a situation.





Controller

The Controller for the platform is currently the herelink Controller by Profi CNC. It is a digital controller that is capable of digital video transmission at HD resolutions up 12km distances. It consists of an air unit and ground unit. The ground unit consists of a mobile device with RF hardware and control interface running a custom version of android.

This system runs at 2.4Ghz with a bandwidth of 20/10 Mhz. It is capable of sending Mavlink telemetry packets from the Cube Flight Controller, allowing for versatile control and configuration of the Platform. The Herelink system has a total measure latency of 180ms using the standard setup and firmware, which is considered very workable. The unit runs RSA256 Encryption, it is a standard that cannot be disabled on the unit. This system meets all the CE standards, thus the power level will be within the legal requirements.

Since the system is android based, it will give many opportunities in the future for software side development, especially leveraging the onboard touch screen for software buttons.

Light Control Circuit

The Light Control Circuit is a dedicated PCB for controlling the Lights and Lasers onboard the platform. It is controlled over the 2 wire I2C Bus connected to the main companion computer. The board powers the following devices:

- Front Visual LED Light
- Front IR LED light
- Rear LED Visual Light
- Front Visual Laser
- Front IR Laser

The LEDs are powered by a 660ma Constant current supply based on the AL8805 LED buck driver, allowing for cool and efficient control of an LEDs. The lasers are powered by LM317 based constant current supplies biased to the ideal current for each diode. The respective LEDs bring the light levels up to the minimum required levels for the visual sensors to function in pitch black.

Trigger PEC

The Trigger PEC applies current to a pair of electric match heads in the effector, such as the "1W/1A"[1], which acts as a trigger. The current is delivered by two 2.2 Farad capacitors connected in parallel.

The following chapter provides details in the circuit design, the microcontroller configuration, both available communication channels (PWM signal – Receiver; I2C bus – Companion Computer) and finally the logic and fail-safe mechanism implemented.

The main functions of the Trigger Board are:

- Check if any effector is loaded with a live round

[1] Purchased from https://www.schaffler.org

- Process the input signals (either via I2C from the companion computer or PWM from a Transmitter)
- Arm any effector while charging the capacitors
- Fire the previously armed effector with the current from the charged capacitor

Circuit Design

The trigger PEC is composed of the following circuits:

- Capacitor charger/discharge
- Arming/firing
- Dual redundant power supply
- Microcontroller

Arming/firing

The effector is armed and fired using two MOSFET relays in series that are switched from a signal from the microcontroller. The microcontroller switches each relay on in response to a command from the operator to arm and fire the effector respectively. A circuit also checks for continuity in the effector to ensure it is loaded. If the continuity is open-circuit, the fire signal is not sent from the microcontroller.
When the fire signal is sent, two 2.2 Farad capacitors in series are discharged across the electric matches in the effector.

Capacitor Charging Circuit

The electric match heads in the effector require at least 3.5 Amps to trigger. This current is provided by two 2.2 Farad capacitors in series. These capacitors are charged using a super capacitor charging circuit based on the Texas Instruments TPS 25940 IC. The circuit charges the 2.2 Farad capacitors to 5 Volts in 2 seconds. The capacitor will only charge following a command from the operator. The capacitor will also be safely discharged through a 2.2 Ohm, 15 Watt, resistor following a command from the operator.

Dual Redundant Power Supply

A dual redundant power supply gives power to the circuits on the trigger board. The power supply is based on a Texas Instruments LM25119 IC. The power supply is built into the PEC so that the supply is standalone from the rest of the platform and only requires battery voltage to function.

Microprocessor

In the following figure the utilised microcontroller (ATmega328) is pictured including all pins (input/output) that are required to realise the Trigger Board functionalities.

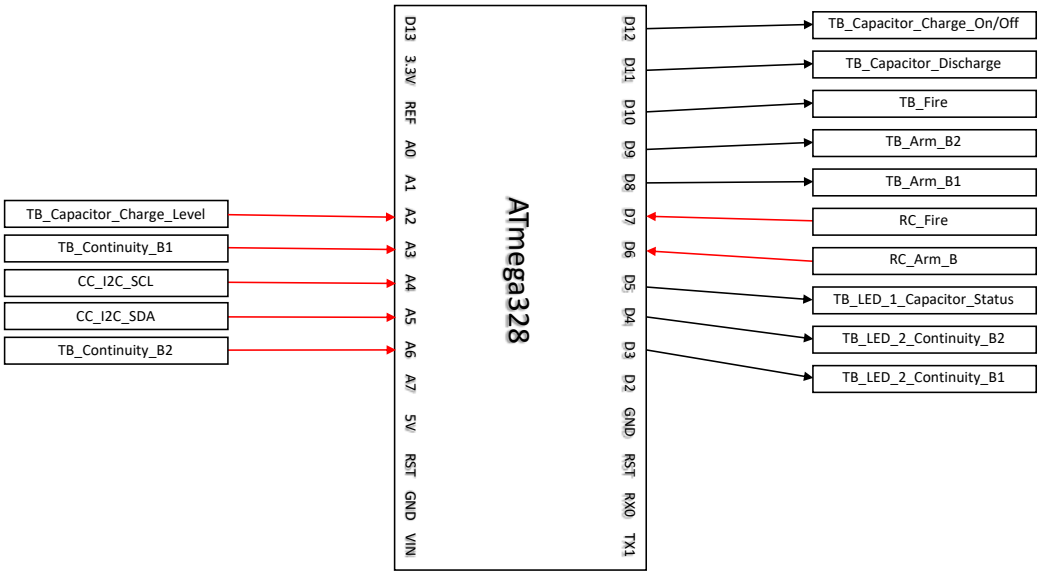


Figure 4.11: Microprocessor (ATmega328) connected to Trigger Board electronics, Receiver and Companion Computer

Each pin is either utilised as input or output whereas the type of signal depends on the functionality. The Trigger Board requires for different types of data:

- Voltage – e.g. to check if the capacitor is charged
- I2C bus – used for the communication with the Companion Computer
- Binary – used for most pins, acting as a switch (HIGH = 5V, LOW = 0V)
- PWM Signal – RC receiver transmit PWM signals which are further processed inside the microcontroller

A higher level of detail is provided in the next table.

Table 4-4: Microprocessor Trigger Board Pin Configuration

Pin	Channel	Definition	Signal Type	Input/output	Unit	Values	Comment
A2	TB_Capacitor_Charge_Level	Trigger Board Capacitor Charge Level	Analog	Input	Voltage	0 – 5 V (1023 steps)	Reads the voltage of capacitor
A3	TB_Continuity_B1	Trigger Board Continuity Barrel 1	Analog	Input	Voltage	0 – 5 V (1023 steps)	Checks if Barrel 1 is loaded (<4.5V == loaded)
A4	CC_I2C_SCL	Companion Computer I2C Bus Serial Clock	Analog	Input/output	I2C bus	-	See **Error! Reference source not found.**
A5	CC_I2C_SDA	Companion Computer I2C Bus Serial Data	Analog	Input/output	I2C bus	-	See **Error! Reference source not found.**

D12	TB_Capacitor_Charge_On/Off	Trigger Board Capacitor Charge On/Off	Digital	Output	Binary	High Low	Turns on/off capacitor charging circuit
D11	TB_Capacitor_Discharge	Trigger Board Capacitor Discharge	Digital	Output	Binary	High Low	(De-)Activates capacitor discharge circuit
D10	TB_Fire	Trigger Board Fire	Digital	Output	Binary	High Low	Fires Barrel 1 or 2, depending on the arming state
D9	TB_Arm_B2	Trigger Board Arm Barrel 2	Digital	Output	Binary	High Low	(Dis-)Arm Barrel 2
D8	TB_Arm_B1	Trigger Board Arm Barrel 1	Digital	Output	Binary	High Low	(Dis-)Arm Barrel 1
D7	RC_Fire	Receiver Fire	Digital	Input	PWM	0 – 2000	Receives Transmitter input as PWM signal (2-stage switch)
D6	RC_Arm_B	Receiver Arm Barrel 1 or 2	Digital	Input	PWM	0 – 2000	Receives Transmitter input as PWM signal (3-stage switch)
D5	TB_LED_1_Capacitor_Status	Trigger Board LED 1 Capacitor Status	Digital	Output	Binary	High Low	LED lights up when capacitor is charged
D4	TB_LED_2_Continuity_B2	Trigger Board LED 2 Continuity Barrel 2	Digital	Output	Binary	High Low	LED lights up when Barrel 2 is loaded with a live round
D3	TB_LED_3_Continuity_B1	Trigger Board LED 3 Continuity Barrel 1	Digital	Output	Binary	High Low	LED lights up when Barrel 1 is loaded with a live round

I2C Communication Protocol

On an I2C bus either one master is communicating with multiple slaves or multiple master are communicating with multiple slaves. An I2C bus requires two analog pins on the ATmega328 (A4, A5), which are defined as SCL (Serial Clock) and SDA (Serial Data) lines. In this project the Trigger Board is setup as the slave and each board is given a unique address (e.g. 0x01), that is required for communicating with the Companion Computer.

The unique address acts as a Fail Safe due to the limiting the access for any external device to communicate with the ATmega328 in case the address code is unknown. In each model, i.e. each drone will have a unique address.

Table 4-5: I2C Communication Protocol between Trigger Board and Companion Computer

Receives Information	Variable	Send Information on Request	Variable

Integer [1,2,3] (Dis-)Arm Barrel 1 or 2	== 1 -> no Barrel is armed == 2 -> Barrel 1 is armed == 3 -> Barrel 2 is armed
Integer [1,2] Fire	== 1 -> no fire == 2 -> fire

Integer [1,2,3,4] Continuity	== 1 -> No Barrel is loaded == 2 -> Barrel 1 loaded == 3 -> Barrel 2 loaded == 4 -> Barrel 1 & 2 loaded
Integer [1,2,3] (Dis-)Arm Barrel 1 or 2	== 1 -> no Barrel is armed == 2 -> Barrel 1 is armed == 3 -> Barrel 2 is armed
Integer [1,2] Fire	== 1 -> no fire == 2 -> fire
Integer [1,2] Capacitor Charged	== 1 -> not charged == 2 -> fully charged

The two-way communication between Trigger Board and Companion Computer enables to implement Fail Safe mechanism and troubleshooting on both sides. Therefore, a simple mechanism is utilised, the Trigger Board sends back the input as received in case everything is working according to plan. In case of any malfunction, the output signal changes into default state, which prevents the Trigger Board from acting at the same time. The next table provides examples of input/output and error messages in all variants.

Table 4-6: Trigger Board States by Input (Master) and Feedback (Slave), inlcuding Errors

ID	Setup (external)	Input Master	Feedback Slave	Result	Action/Error
001	Barrel 1 not loaded Barrel 2 not loaded	Arm == (1 ll 2 ll 3) Fire == (1 ll 2)	Continuity == 1 Arm == 1 Fire == 1 Capacitor == 0	Barrel 1 & 2 not loaded	Load Barrel If (Capacitor == 1) Error – faulty trigger board (capacitor charged)
002	Barrel 1 loaded Barrel 2 not loaded	Arm == (1) Fire == (any)	Continuity == 2 Arm == 1 Fire == 1 Capacitor == 0	Barrel 1 loaded	Ready to arm barrel 1 If (Capacitor == 1) Error – disarm and reboot
003	Barrel 1 loaded Barrel 2 not loaded	Arm == (2) Fire == (1)	Continuity == 2 Arm == 2 Fire == 1 Capacitor == 1	Barrel 1 loaded Barrel 1 armed Capacitor charged	Ready to fire barrel 1, Delay 1s If (Capacitor =/ 1) Error – Capacitor does not charge
004	Barrel 1 loaded Barrel 2 not loaded	Arm == (2) Fire == (2)	Continuity == 2 Arm == 2 Fire == 2	Barrel 1 loaded Barrel 1 armed Capacitor charged Barrel 1 fire	Delay 1s; If (Continuity =/ 1) Error – faulty round

			Capacitor == 1		
005	Barrel 1 loaded Barrel 2 not loaded	Arm == (3) Fire == (1 ll 2)	Continuity == 2 Arm == 1 Fire == 1 Capacitor == 0	Barrel 1 loaded	Error – wrong barrel armed / barrel 2 not loaded
005	Barrel 1 loaded Barrel 2 not loaded	Arm == (2) Fire == (<1 ll >2)	Continuity == 1 Arm == 1 Fire == 1 Capacitor == 0	Program remains in default state	Error – wrong input command
006	Barrel 1 loaded Barrel 2 not loaded	Arm == (<1 ll >3) Fire == (any)	Continuity == 1 Arm == 1 Fire == 1 Capacitor == 0	Program remains in default state	Error – wrong input command
007	Barrel 1 not loaded Barrel 2 loaded	Arm == (1) Fire == (any)	Continuity == 3 Arm == 1 Fire == 1 Capacitor == 0	Barrel 2 loaded	Ready to arm barrel 2 If (Capacitor == 1) Error – faulty trigger board (capacitor charged)
008	Barrel 1 not loaded Barrel 2 loaded	Arm == (3) Fire == (1)	Continuity == 3 Arm == 3 Fire == 1 Capacitor == 1	Barrel 2 loaded Barrel 2 armed Capacitor charged	Ready to fire barrel 2, Delay 1s If (Capacitor =/ 1) Error – Capacitor does not charge
009	Barrel 1 not loaded Barrel 2 loaded	Arm == (3) Fire == (2)	Continuity == 3 Arm == 3 Fire == 2 Capacitor == 1	Barrel 2 loaded Barrel 2 armed Capacitor charged Barrel 2 fire	Delay 1s; If (Continuity =/ 1) Error – faulty round
010	Barrel 1 not loaded Barrel 2 loaded	Arm == (2) Fire == (1 ll 2)	Continuity == 3 Arm == 1 Fire == 1 Capacitor == 0	Program remains in default state	Error – wrong barrel armed / barrel 1 not loaded
011	Barrel 1 not loaded	Arm == (3) Fire == (<1 ll 2>)	Continuity == 1 Arm == 1	Program remains in default state	Error – wrong input command

	Barrel 2 loaded		Fire == 1 Capacitor == 0		
012	Barrel 1 not loaded Barrel 2 loaded	Arm == (<1 || 2 || >3) Fire == (any)	Continuity == 1 Arm == 1 Fire == 1 Capacitor == 0	Program remains in default state	Error – wrong input command
013	Barrel 1 loaded Barrel 2 loaded	Arm == (1) Fire == (any)	Continuity == 4 Arm == 1 Fire == 1 Capacitor == 0	Barrel 2 loaded	Ready to arm barrel 2 If (Capacitor == 1) Error – faulty trigger board (capacitor charged)
014	Barrel 1 loaded Barrel 2 loaded	Arm == (2) Fire == (1)	Continuity == 4 Arm == 2 Fire == 1 Capacitor == 1	Barrel 1 & 2 loaded Barrel 1 armed Capacitor charged	Ready to fire barrel 1, Delay 1s If (Capacitor =/ 1) Error – Capacitor does not charge
015	Barrel 1 loaded Barrel 2 loaded	Arm == (2) Fire == (2)	Continuity == 4 Arm == 2 Fire == 2 Capacitor == 1	Barrel 1 & 2 loaded Barrel 1 armed Capacitor charged Barrel 1 fire	Delay 1s; If (Continuity =/ 1) Error – faulty round
016	Barrel 1 loaded Barrel 2 loaded	Arm == (3) Fire == (1)	Continuity == 4 Arm == 3 Fire == 1 Capacitor == 1	Barrel 1 & 2 loaded Barrel 2 armed Capacitor charged	Ready to fire barrel 2, Delay 1s If (Capacitor =/ 1) Error – Capacitor does not charge
017	Barrel 1 loaded Barrel 2 loaded		Continuity == 4 Arm == 3 Fire == 2 Capacitor == 1	Barrel 1 & 2 loaded Barrel 2 armed Capacitor charged Barrel 2 fire	Delay 1s; If (Continuity =/ 1) Error – faulty round
018	Barrel 1 loaded Barrel 2 loaded	Arm == (2 || 3) Fire == (<1 || >2)	Continuity == 1 Arm == 1 Fire == 1 Capacitor == 0	Program remains in default state	Error – wrong input command
019	Barrel 1 loaded	Arm == (<1 || >3)	Continuity == 1	Program remains in default state	Error – wrong input command

	Barrel 2 loaded	Fire == (any)	Arm == 1 Fire == 1 Capacitor == 0		

Logic

The high-level structure consists of global definitions, setup, main loop and two functions for I2C communication. While receiveEvent () reads information send by the Master (Companion Computer), requestEvent () provides information to the Master as detailed in **Error! Reference source not found.**.

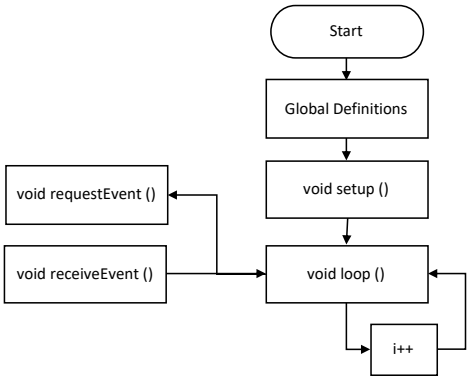


Figure 4.12: Trigger Board high level code structure

Several fail-safe mechanisms implemented ensure a safe operation.

Fail-Safe Mechanism

1. Global Definitions; The communication between Companion Computer (I2C) or any Transmitter (PWM) is only possible by matching the unique address code (I2C), selecting the way of communication correctly in the code when uploading it on the microcontroller and providing PWM signals in the correct range (e.g. 1000 – 2000) and correct denomination (2- or 3-stage switch).
2. Void Setup (); All relevant pins (arming weapon, firing weapon, charging capacitor) are set to LOW during the setup process
3. Void Loop (); When booting the microcontroller an initial condition pre-sets that the first 20 cycles are skipped in case any faulty input signal is read. These input signal may appear from static charge in the circuit during downtime. However, best practice has proven this method being simple and effective.
4. Void Loop (); The Trigger Board is connected to the power supply of the Companion Computer, in case the Companion Computer is not powered, the Trigger Board switches into safe mode, disabling any input or execution.
5. Void Loop (); Input signals are processed in three Level (1-3), which work in a waterfall structure, e.g. arming a barrel (Level 2) requires that the respective barrel is loaded with a live round (Level 1). And fire that barrel (Level 3) requires that the barrel was armed previously (Level 2).

6. Void requestEvent (); The trigger board provides all relevant information about its current status on request. In case master input and slave output mismatch one can deduct the error according to **Error! Reference source not found.**.
7. Void Loop (); In case of non-classified input signals the Trigger Board switches into safe mode, disabling any execution.

Global Definitions contains the following parameter

- All constant parameter, such as int IN_Arm_B = 6; // digital pin number 6 is used for reading signal input for firing command
- Libraries, such as #include <Wire.h> // required for I2C communication
- Define unique Trigger Board slave address, such as #define slave_address 0x01 // this Trigger Board (example) is only accessible via I2C by calling the here defined slave address
- Initial setup, such as int PWM_I2C = 1; // by changing the integer value to 1 or 2, one enable either communication via I2C or PWM signals
- Global variables, such int counter = 0; // utilised for processing information, value may change during void loop (i) or void loop (i++)

```
////////////////////////////////////////////////////
//
// Global Variables and Libraries
//
////////////////////////////////////////////////////

/*
 * Switch between PWM Signal (Pixhawk) and I2C Signal (Companion Computer) Connection
 * PWM_I2C == 1 - takes PWM Signal
 * PWM_I2C == 2 - takes I2C Signal
 */

 int PWM_I2C = 1; // input required as stated above

// ---------------- Libraries and Global Definitions
#include <Wire.h>   // used for I2C communication
#define slave_address 0x01 // I2C address of this specific trigger board

// ---------------- Set Pins ----------------
// Input from the companion computer
int IN_Arm_B = 6; // (digital)(input) arm barrel 1 & 2
int IN_Fire = 7;   // (digital)(input) fire, counts for both weapons
int IN_System_Present = A0; // (analog)(input) provides feedback if the companion computer is powered
int IN_System_Enabler = A1; // (ananlog)(input) enables the trigger board in case the correct code is
transmitted

// ---------------- Variables ----------------
int counter = 0; // counter used to skip the processor booting process and error signals
```

Void Setup () initial setup of ATmega328

- Setup of communication channel, such as Wire.begin (slave_address); // defines that this Trigger Board is only accessible by calling the previously defined slave address (0x01)
- Define pins either as INPUT or OUTPUT pins, such as pinMode(IN_Arm_B, INPUT); // defines this pin (D6) as INPUT
- Initialising pin setup, such as digitalWrite(OU_Arm_B1,LOW); // sets the pin to LOW for safety reason

```
////////////////////////////////////////////////
//
// Setup to Define Pins
//
////////////////////////////////////////////////
void setup()
{
  Serial.begin(9600);

  // join i2c bus as a slave
  Wire.begin (slave_address);   // sets slave address
  Wire.onRequest(requestEvent); // function send out data
  Wire.onReceive(receiveEvent); // function receive date

  // Input from the companion computer
  pinMode(IN_Arm_B, INPUT);
  pinMode(IN_Fire, INPUT);
  pinMode(IN_System_Present,INPUT);
  pinMode(IN_System_Enabler,INPUT);

// initialising pin settings for safety reason and indicating booting process
  for (int i = 0; i <= 5; i++)
  {
  digitalWrite(OU_Arm_B1,LOW);
  digitalWrite(OU_Arm_B2,LOW);
  digitalWrite(OU_Fire,LOW);
  digitalWrite(OU_Safety_Discharge, HIGH);
  digitalWrite(OU_Cap_Charge_Switch, LOW);
  // visual feedback
  digitalWrite(OU_Error_Check_Connection_B1,HIGH);
  digitalWrite(OU_Error_Check_Connection_B2,HIGH);
  digitalWrite(OU_Visual_Feedback_Weapon_Armed,HIGH);
  delay(200);
  digitalWrite(OU_Error_Check_Connection_B1,LOW);
  digitalWrite(OU_Error_Check_Connection_B2,LOW);
  digitalWrite(OU_Visual_Feedback_Weapon_Armed,LOW);
  Serial.println("booting");
  delay(200);
  }
```

Void Loop () main loop

Due to the complexity of the void loop () **Error! Reference source not found.** provides details on the main parts, however, each is separately explained in the next table.

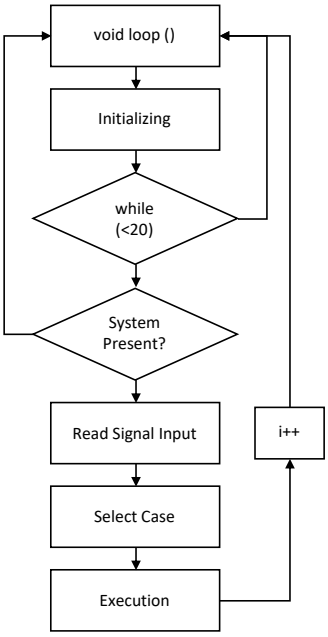


Figure 4.13: Main loop structure.

Process Step	Code Example	Details

Initializing	// ---------- skipping the first iterations
	// first loop required to skip the first 20 processor loops in order to eliminate error signals
	// occuring during the processor booting process
	if (counter <= 20){
	counter = counter+1;
	delay(10);
	}
		Booting the ATmega328 causes issues by reading the first values, hence, a simple loop was implemented to skip the first 20 loop cycles before reading any values

System Present	System_Present = analogRead(IN_System_Present); // (ananlog)(input) A0
	float Voltage_System_Present = System_Present * (5.0 / 1023.0);
	
	if (Voltage_System_Present >= 1.0)
	{
	
	(placeholder for Read Signal Input, Select Case and Execution part)
	
	}
		Analog pin (IN_System_Present) is connected to the power of the Companion Computer, hence, when the Companion Computer is not powered, the

	else // initial conditions "Voltage_System_Present" and "System Enable" not fullfilled
	{
	digitalWrite(OU_Error_Check_Connection_B1,LOW); // de-activates LED 1 for Barrel 1
	digitalWrite(OU_Error_Check_Connection_B2,LOW); // de-activates LED 2 for Barrel 2
	digitalWrite(OU_Arm_B1,LOW); // set output pin Arm_B1 to LOW
	digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW
	digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
	digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
	digitalWrite(OU_Fire,LOW);
	// digitalWrite(OU_Visual_Feedback_Weapon_Armed,LOW);
		voltage drops to 0V. In that case all critical pins (arming, firing, capacitor charging) are set to LOW, disabling any function. Furthermore, input signals are processed.

Read Input Signals	// ---- receive date from maser
	void receiveEvent()
	{
	// condition required to activate I2C communication
	if (PWM_I2C == 2)
	{
	Arm_B_Master = Wire.read();
	/*
	* LEVEL 2 - Master Input
	* case # 1 - not armed (Arm_B_Master == 1)
	* case # 2 - barrel 1 is armed (Arm_B_Master == 2)
	* case # 3 - barrel 2 is armed (Arm_B_Master == 3)
	*/
	
	Fire_Master = Wire.read();
	/* LEVEL 3 - Master Input
	* case # 1 - not firing (Fire_Master == 1)
	* case # 2 - firing (Fire_Master == 2)
	*/
	}
	}
		Input signals are either received via I2C or PWM. In case of I2C the function reveiveEvent () defines the number and type of input.

Select Case	// ---------- Conditions and Cases ---------- /* * LEVEL 1 * case # 1 - Continuity_Check_B1 && Continuity_Check_B2 = FAIL (>=4.5V) * case # 2 - Continuity_Check_B1 = OK (<=4.5V) && Continuity_Check_B2 = FAIL (>=4.5V) * case # 3 - Continuity_Check_B1 = FAIL (>=4.5V) && Continuity_Check_B2 = OK (<=4.5V) * case # 4 - Continuity_Check_B1 = OK (<=4.5V) && Continuity_Check_B2 = OK (<=4.5V) */	This part is devided in three levels Level 1 – checks if there is continuity on the barrels, i.e. if the barrels are loaded with live rounds Level 2 – process the input from either Companion

	if (Voltage_Continuity_Check_B1 >= 4.5 && Voltage_Continuity_Check_B2 >= 4.5){
	cases_level_1 = 1;
	}
	if (Voltage_Continuity_Check_B1 <= 4.5 && Voltage_Continuity_Check_B2 >= 4.5){
	cases_level_1 = 2;
	}
	if (Voltage_Continuity_Check_B1 >= 4.5 && Voltage_Continuity_Check_B2 <= 4.5){
	cases_level_1 = 3;
	}
	if (Voltage_Continuity_Check_B1 <= 4.5 && Voltage_Continuity_Check_B2 <= 4.5){
	cases_level_1 = 4;
	}
	
	/*
	* LEVEL 2 - Master Input
	* case # 1 - not armed (Arm_B_Master == 1)
	* case # 2 - barrel 1 is armed (Arm_B_Master == 2)
	* case # 3 - barrel 2 is armed (Arm_B_Master == 3)
	*/
	if (Arm_B_Master == 1){
	cases_level_2 = 1;
	}
	else if (Arm_B_Master == 2){
	cases_level_2 = 2;
	}
	else if (Arm_B_Master == 3){
	cases_level_2 = 3;
	}
	else
	{
	cases_level_1 = 1; // program remains in default state due to faulty input
	}
		Computer (I2C) or any transmitter (PWM). On Level 2 the operator can arm one of the barrels or disarm the effector.
		
		Level 3 – same as Level 2 besides the command, which allows the operator to fire any previously armed barrel.
		
		The level structure works in a waterfall principle, i.e. Level 1 conditions enable or disable Level 2 conditions. Same counts for the relation between Level 2 and 3.

Execution	////////////////// LEVEL 1 - Case 1 //////////////////
	switch(cases_level_1){
	case 1:
	cases_level_1_slave = 1; // confirms status for master feedback
	cases_level_2_slave = 1; // confirms status for master feedback
	cases_level_3_slave = 1; // confirms status for master feedback
	//
	//
	// Level 1 - case 1 Continuity Barrel 1 & 2 FAIL
	//
	// *Continuity Barrel 1 == LOW
		Depending on the previously defined cases on each Level, the Execution part sets the respective pins either HIGH or LOW.

```
            //  *Continuity Barrel 2  == LOW
            //  *Arm Barrel 1         == LOW
            //  *Arm Barrel 2         == LOW
            //  *Capacitor Charge     == LOW
            //  *Safety Discharge     == HIGH
            //  *Fire                 == LOW
            //
            ////////////////////////////////////////////////
            digitalWrite(OU_Error_Check_Connection_B1,LOW); // de-
activates LED 1 for Barrel 1
            digitalWrite(OU_Error_Check_Connection_B2,LOW); // de-
activates LED 2 for Barrel 2
            digitalWrite(OU_Arm_B1,LOW); // set output pin Arm_B1 to LOW
            digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW
            digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity
charging circuit
            digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity
discharge circuit
            digitalWrite(OU_Fire,LOW);
            break;
```

Section 4.4 Control

Estimator

Overview

The estimator uses an Extended Kalman Filter (EKF) to fuse position and velocity data from the T265 camera and accelerometer data from the Pixhawk's accelerometer in order to output a position and orientation estimate of the craft to feed to the PX4's own internal EKF.

Pipeline

- T265 handler: boots and initialises cameras, publishes raw data and covariances to ROS
- VIO_handler : subscribes to ROS cameras pose and velocity topics. Perform frame of reference translation to obtain the position and velocity data of the drone's centre of mass.
- Ekf_odom_node: sets up EKF, with associated variances and process noises. Subscribes to the IMU data from the Pixhawk, and the VIO pose and velocity from the velocity data. Outputs filtered odometry pose data to mavros vision topic.
- MAVROS vision topic: forwards estimated drone position data to PX4
- PX4 firmware: fuses VIO data into its own estimator.

Future work

This estimator was built to be vastly configurable and flexible. Thus it is possible to fuse position data obtained from the D435 camera, the rotating lidar or the GPS unit. Development is in progress to include those new sensors in order to improve the precision and stability of the estimator.

Graphical User Interface (GUI)

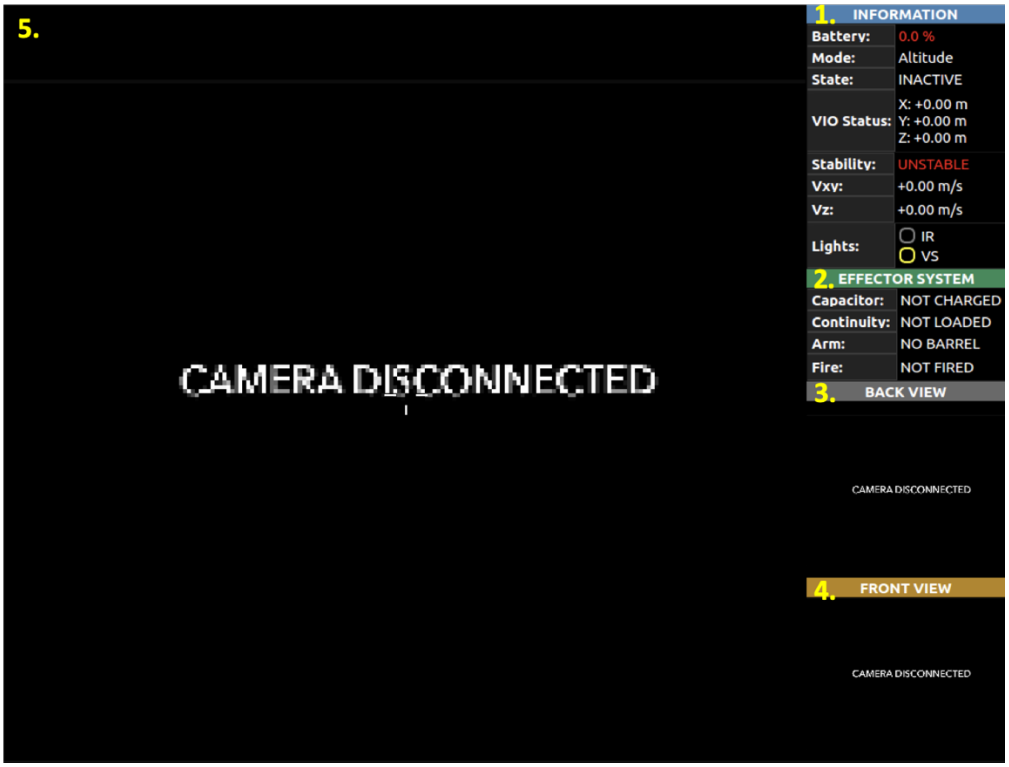


Figure 4.5: Graphical User Interface, GUI.

The GUI can be divided into five parts, described as below.

1. Basic Information
- Battery: the current battery level of drone. If the remaining battery level is larger than 25 percentage, it would indicate the value with white colour; If the remaining battery level is between 10 percentage and 25 percentage, it would indicate the value with orange colour; If the remaining battery level is smaller than 10 percentage, it would indicate the value with red colour
- Mode: the current flight mode, such as Altitude mode or Position mode.
- State: the current status of drone's motor. If the status is active, then the propellers of drone is spinning, and it would display "ACTIVE" with red colour. Otherwise, it displays "INACTIVE" with white colour.
- VIO Status: Presents the X, Y and Z drone position coordinates from the EKF VIO estimator.
- Stability: the stability of drone. If the flying speed (both in XY direction and Z direction) are smaller than specific values for one second, then it means that the drone is stable. Otherwise, the drone is in unstable condition. The specific checking values are set in a launch file (/i2c_ros/launch/triggerboard.launch). In the launch file, a parameter of "check_sqrt_velocity_x_y" is the checking value of drones' speed in X and Y direction. In addition, a parameter of "check_velocity_z" is the checking value of drones' speed in Z direction. If the drone is unstable, it would display "UNSTABLE" with red colour in GUI. Otherwise, it displays "STABLE" with white colour.
- Vxy: the flying speed in x and y direction
- Vz: the flying speed in z direction
- Lights: the current status of VR (visual light) and IR (infrared light). If the light is turned on, the check box is filled with yellow colour for VR light and grey colour for IR light.

2. Effector System

- Capacitor: it indicates whether the capacitor is charged or not, presented as "NOT CHARGED" or "CHARGED"
- Continuity: it indicates whether the barrel is loaded or not, presented as "NO BARREL", "BARREL 1", "BARREL 2" or "BARREL 1 and 2"
- Arm: it indicates whether the barrel is armed or not, presented as "NO BARREL", "BARREL 1" or "BARREL 2"
- Fire: it indicates whether the system is fired or not, presented as "NOT FIRED" or "FIRED"

3. Back View

- Display the image acquired from T265 camera
- If T265 camera is disconnected or missing frames, " CAMERA DISCONNECTED" is showed in the centre of window.

4. Front View

- Display the image acquired from D435 camera or USB camera, depending on the parameter set in a launch file or the command from radio controller.
- In the launch file, a parameter of "main_window_mode" decides the image displayed in Main Window of GUI. If the parameter is set as "d435", then the window of Front View would acquire the image form USB camera. In the same way, if the parameter is set as "usbcam", then the window of Front View would acquire the image form D435 camera.
- Only if the above parameter is set as "usbcam", then the radio controller contains a function of image's switching between D435 camera and USB camera. If user switch the controller to off-board mode, then the window of Front View only shows the image acquired from USB camera. In addition, during target selection, the image displayed in the window of Front View is switched from D435 camera to USB camera. Then, after 20 seconds, the image would be switched back.
- If AI code is launched, the information of detection is showed on the image of D435 with green colour's box.
- If the camera is disconnected or missing frames, " CAMERA DISCONNECTED" is shown in the centre of window.

5. Main Window

- Display the image acquired from D435 camera or USB camera, depending on the parameter setting in a launch file or the command from radio controller.
- In the launch file, a parameter of "main_window_mode" would decide the image display in Main Window of GUI. If the parameter is set as "usbcam", then the Main Window would acquire the image from USB camera. In the same way, if the parameter is set as "d435", then the Main Window would acquire the image form D435 camera.
- Only if the above parameter is set as "usbcam", then the radio controller contains a function of image's switching between D435 camera and USB camera. If user switch the controller to off-board mode, then the Main Window only shows the image acquired from D435 camera. In addition, during target selection, the image displayed in the Main Window is switched from USB camera to D435 camera. Then, after 20 seconds, the image would be switched back.

- If AI code is launched, the information of detection is showed on the image of D435 with green colour's box.
- If AI code is launched and the radio controller is switched to target selection, the information of tracking is showed on the image of D435 with green colour's box. In addition, the selected target is overlaid with orange colour's box.
- If AI code is launched and the radio controller is switched to off-board mode, the information of targeting is showed on the image of D435 with red colour's box.
- If the camera is disconnected or missing frames, " CAMERA DISCONNECTED" is showed in the centre of window.
- The crosshair is displayed on the Main Window. The location of crosshair is decided by the parameters in the launch file, such as dim/main_frame/width, dim/main_frame/height, dim/crosshair/x_offset and dim/crosshair/y_offset.

Section 4.5 Effector

Figure 4.14: The effector assembly

Weapons of all calibres go through a rearward motion (recoil force) when discharge due to pressure acting on the breech block until the projectile departs and exits the muzzle. So-called recoilless weapons eliminate the recoil force by accelerating either a counter mass or burned propellant gas in the opposite direction of the projectile, subsequently achieving momentum conservation.

The current system relates to a lightweight recoilless weapon system, in particular developed for small UAV applications. It features the capability of engaging targets with various types of ammunition. Hence, the UAV can be utilised in a wide range of operations. By attaching multiple of these systems to a small UAV, the operator is able to engage a target with different ammunition at the same time.

The design consists of an enforcement part (12ga shotgun) attached to a rocket-based (nozzle) recoil elimination system, but physically separated by a solid breech block in between the two. Hence, the system requires triggering of two rounds simultaneously. Since the enforcement part of the weapon system is physically separated from the nozzle part of the weapon system by the solid breech block, the two rounds are independent. Hereby, the system is capable of engaging targets with various types

of ammunition (e.g. lethal, non-lethal or disruptive), while using tailored rounds for the nozzle and ensuring complete recoil compensation for each type of ammunition used.

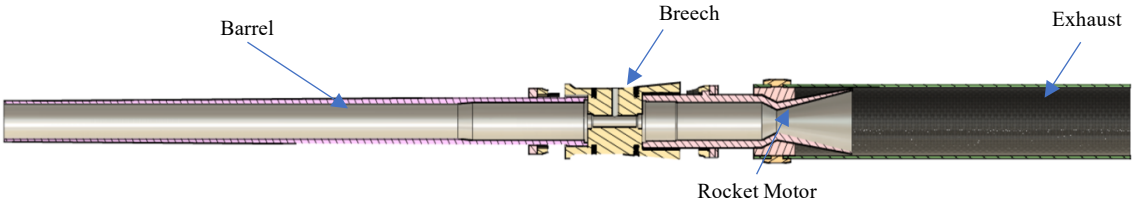


Figure 4.15: Cut away view of the effector system assembly

Components

In total the system comprises the following components:

- Effector – smooth bore 12ga shotgun barrel with a length of 30 cm, in combination with a chamber for bespoken effector cartridges. Barrel to Breech connection is realised by M28x1 interrupted thread. A location fit (H7/h6) allows the cartridge to slide inside the chamber and expanding up to 0.26 mm during firing.
- Breech – Two contact pins are housed inside the breech. These two pins are, however, insulated from said breech and connected with cables which exits the breech centre to connect to the trigger board. To achieve two fully parallel circuits, two further contact pins ensure reliable connection between the cartridges and the breech body. The breech acts as a common ground to the two circuits and a ground wire, connected to the breech, ensures connection with the trigger board.
- Nozzle – Burned propellant in the nozzle-chamber is accelerated through the convergent divergent part of the nozzle, generating a counter momentum to the effector side. Due to high longitudinal stress on the connection between nozzle and breech, a M30x1 interrupted thread has been chosen. A location fit (H7/h6) allows the cartridge to slide inside the chamber and expanding up to 0.26 mm during firing.
- Effector Cartridge – While the internal cartridge dimensions are fit for slugs, bird- or bug shot, the external diameter varies due to the electrical triggering system. Hence, only bespoken ammunition is fit for purpose, standardised rounds cannot be fired in this system.
- Nozzle Cartridge – Compared to the effector cartridge, the nozzle cartridge does not shoot any projectile. However, it contains a high amount of propellant. Once ignited, the combustion gases are accelerated up to Mach 3 to 4. Due to the high amount of propellant, the pressure is significantly higher than on the effector side. The round (basically a blank) is triggered in an equivalent way to the effector cartridge.

Material

All the effector parts are made of aluminium 7075-T6 which is a high-quality aluminium alloy. This alloy combines the low mass of aluminium with much increased strength when compared to most aluminium alloys. The resulting high strength-to-weight ratio makes it an appealing candidate for applications where low weight is critical. Hence, aluminium 7075 T6 is typically used for aircraft components as well as military applications such as rifle receivers.

Effector Barrel

The effector barrel comprises both the chamber that houses the effector cartridges and the barrel that allows expansion of the combustion gases and hence acceleration of the projectile. The effector barrel is designed to sustain inner pressure of 150 MPa (21,750 psi) with an applied safety factor of 1.5 bringing the maximum ultimate inner pressure to 225 MPa (32,630 psi). The barrel is a smooth bore barrel for 12 gauge shotgun application and measures 0.3m from end of chock to muzzle.

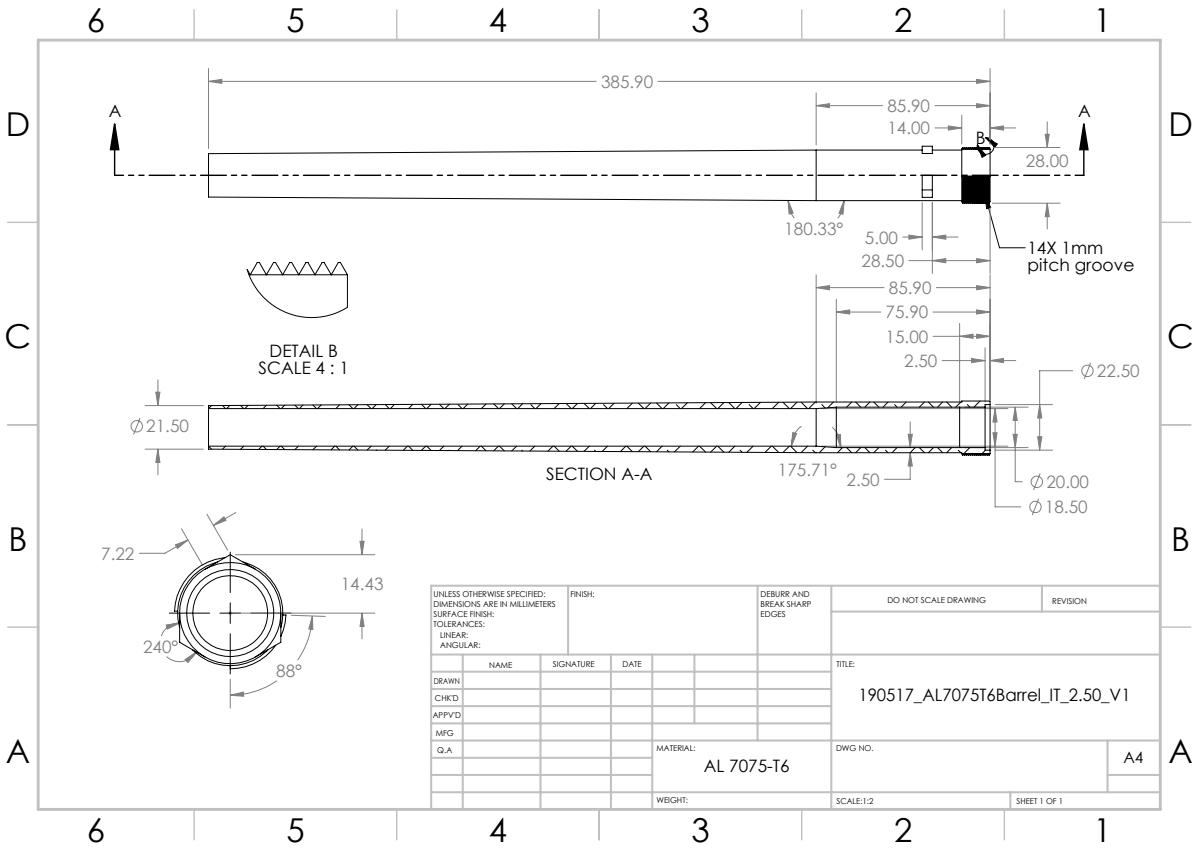


Figure 4.16: Dimensions of effector barrel and chamber.

Breech

The breech fulfils three main functions in the effector assembly:

1. Holding the three parts into one single system.
2. Separating the gases from the effector chamber and from the nozzle chamber.
3. Enabling access of the electric circuits to the primers in order to ensure a functioning firing system.

The breech block is connected to the effector barrel through a M28 x 1 thread on the front side and to the nozzle through a M 30 x 1 thread on the rear side. Both threads are interrupted, with only 50% of the circumference being threaded. This allows quick and reliable locking of the barrel/nozzle inside the breech in a single 90 degree rotation. The fine thread ensures high load bearing capability.

The breech block also contains two channels which house the wiring connecting the primers to the rest of the electric system. The main channel runs from one threaded connection to the other in the longitudinal direction of the breech while the secondary channel connects the main channel to the outside of the breech radially. Insulators are positioned at both ends of the main channel in order to

separate the electric wires from the breech and to accurately place the pins which extend from those wires in the threaded connections of the breech. A ground wire is connected to one of the external holes on the breech. This acts as common ground for both the effector ammunition and the nozzle ammunition.

The breech is designed to sustain inner pressure of 200 MPa (psi) with an applied safety factor of 1.5 bringing the maximum ultimate inner pressure to 300 MPa (psi).

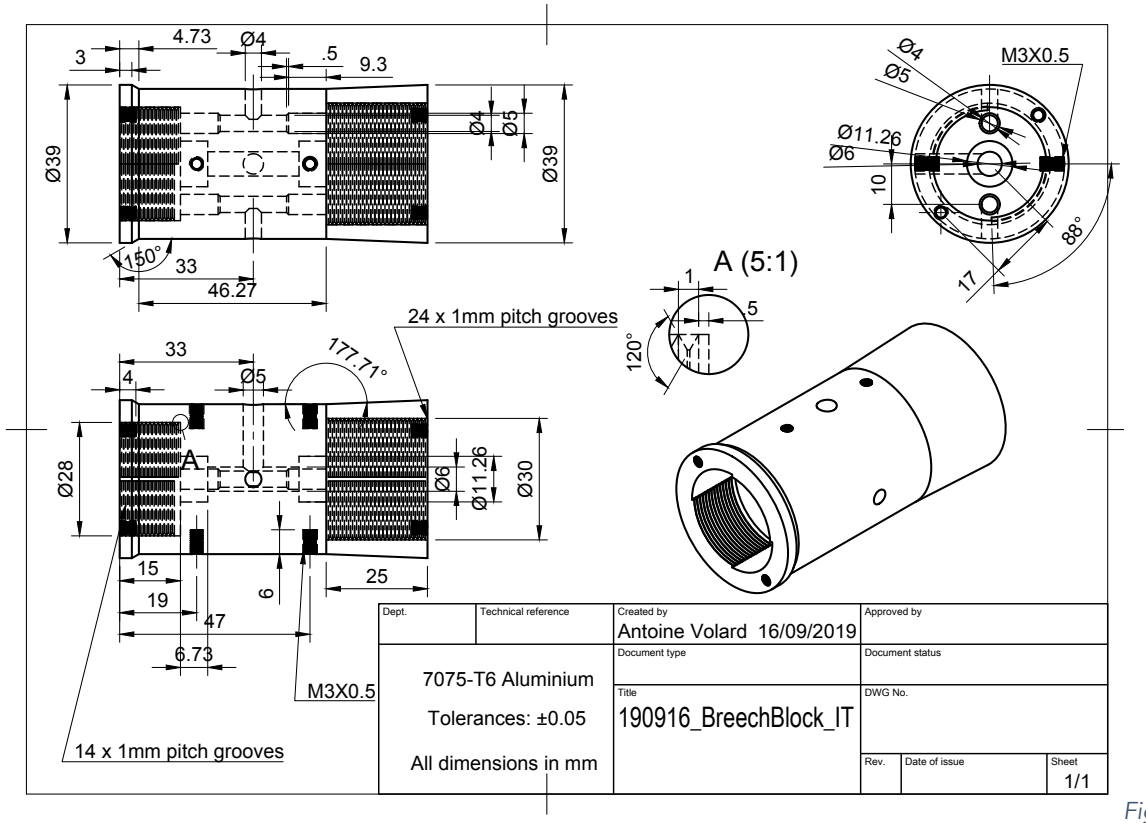


Figure 4.17: Dimensions of breech

Rocket motor

The rocket motor accelerates hot combustion gases to extremely high speed to create the thrust which compensates the recoil force created by the effector on firing. The rocket motor comprises a chamber which houses the rocket motor ammunition. This chamber includes a step which serves two purposes: first of all, it allows for easier cartridge extraction after firing ; furthermore, it ensures that the effector ammunition cannot be inserted inside the rocket motor as the internal diameter past the step is smaller than that of the effector cartridge.

The rocket motor works like a conventional convergent-divergent nozzle. Upon ignition of the ammunition, combustion gases exit the rocket motor ammunition to fill the chamber. These slow-moving gases are then accelerated in the convergent part of the rocket motor. At the point of minimum diameter, called the throat, the gases reach a local velocity of Mach 1 – that is exactly the local speed

of sound. To further accelerate this flow, a divergent section is added downstream of the throat. At exit, velocities of Mach 3 or Mach 4 can be achieved, which result in high forward force being produced.

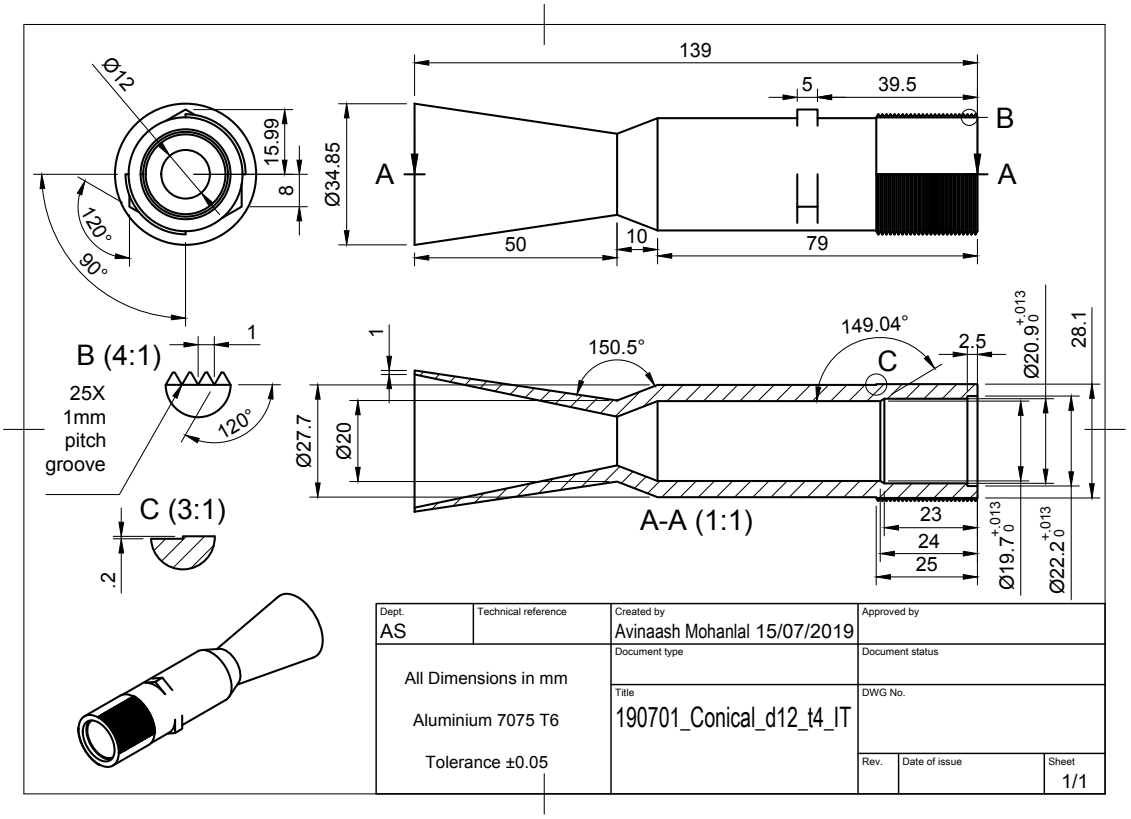


Figure 4.18: Dimensions of rocket motor - chamber and nozzle

Ammunition

Primer

The electrical primer is realised by a combination of two components – glass-to-metal seal and electric match head. Whereas glass-to-metal seals are most commonly used for high pressure electrical connections, an electric match head is mainly for fireworks purposes. Soldering material is added to connect the contact pins with the match head. A driving fit (H7/s6)[2] connects glass-to-metal seal and cartridge, preventing pressure leaking up to 150 MPa.

[2] Medium interference which can be assembled with hot pressing or cold pressing with large forces - e.g. permanent mounting of gears, shafts, bushes (tightest possible with cast iron)

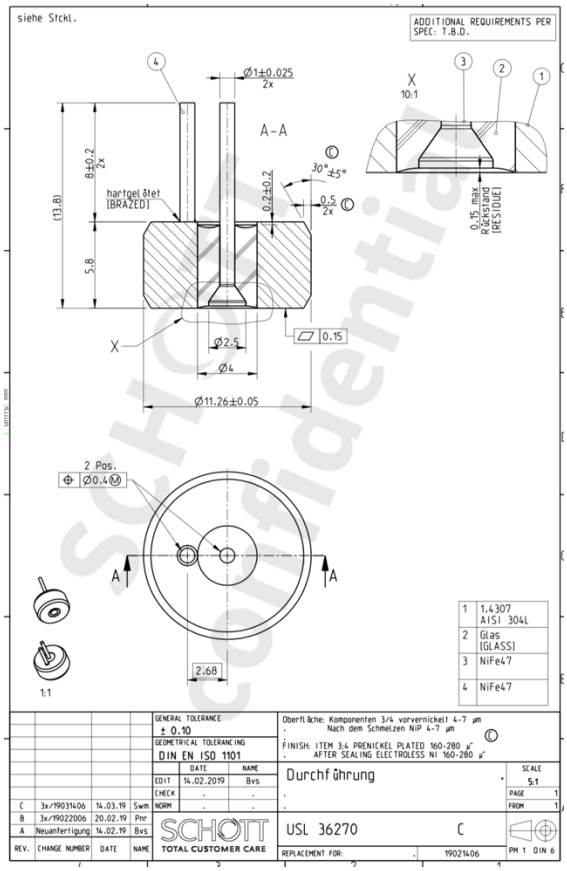


Figure 4.19: Dimensions of the glass-to-metal seal.

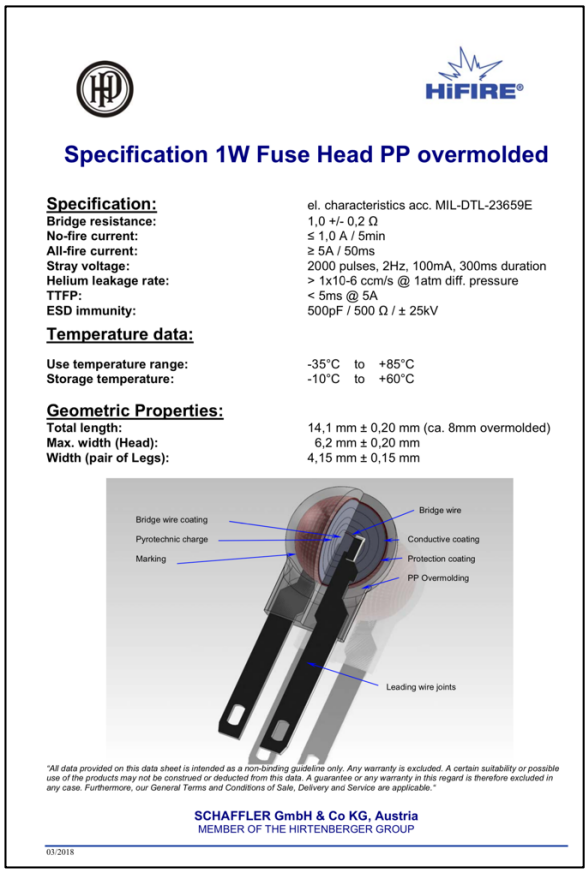


Figure 4.20: Technical details of the electric match head.

Glass-to-metal seal

High pressure seal with one contact pin in the middle, that is isolated by glass from the wall and the second pin. The component allows to pass high current through to ignite the match head inside the cartridge while the cartridge base remains sealed up to high pressure.

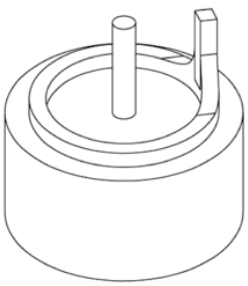


Figure 4.21 Glass to metal seal - graphic representation

Effector Ammunition

Cartridge

The cartridge is manufactured out of Aluminium 7075-T6, with the following physical properties[3]:

- Hardness, Vickers 175
- Ultimate Tensile Strength 572 MPa
- Tensile Yield Strength 503 MPa
- Modulus Elasticity 71.7GPa

From bottom to top the cartridge contains the components as listed below:

- Primer assembly
- Paper Disc
- Propellant
- Projectile
- Plastic Disc

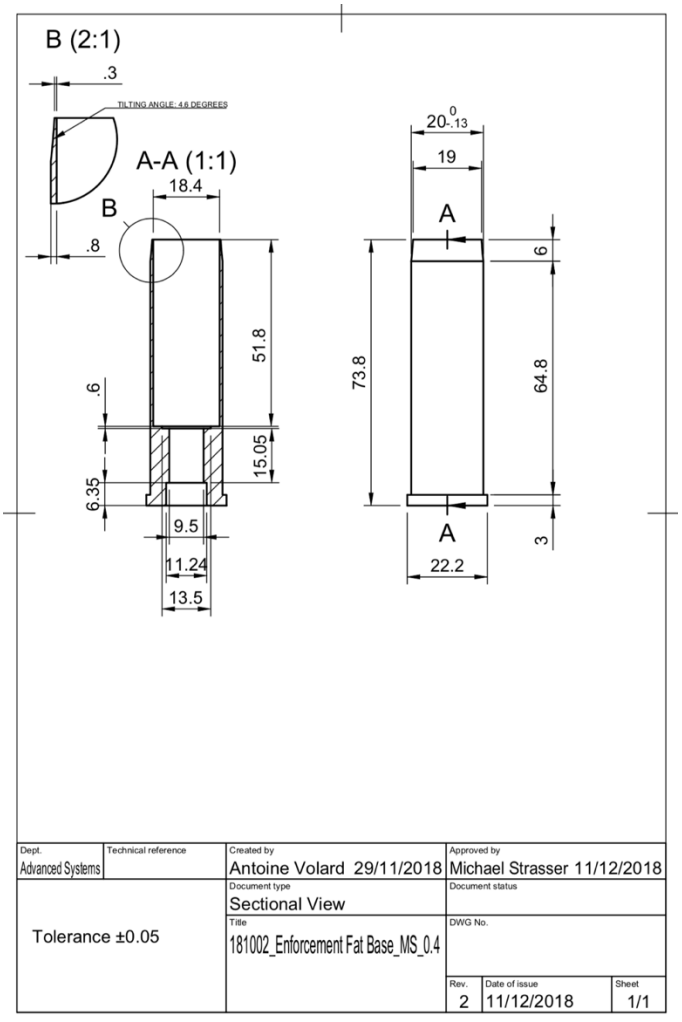


Figure 4.22: Design of effector cartridges.

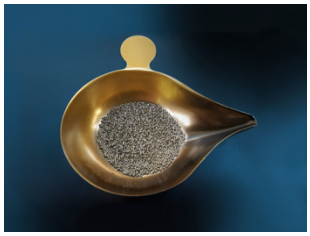
Paper Disc

A ⌀ 14 mm paper disc (17 g/m²) separates match head and propellant from each other, preventing chemical reaction during storage and increasing efficiency of ignition.

Propellant

VihtaVuori N310 is an accurate, extremely clean and fast burning handgun powder. It is a single-base, tubular powder type with grain dimensions of 0,7 mm length and 0,6 mm diameter. Beside handgun applications, it is used for loading shotshells and blank ammunition.[4] According to the safety data sheet released by VihtaVuori it is classified as 1.3C hazard (UN number 0161).[5] The effector cartridge is filled with 9.7gn of propellant.

- Appearance — Cylinder grains
- Odour — Odourless
- Relative density — 350-1000 g/cm³
- Auto-ignition temperature — >175°C



Projectile

Three types of projectiles have been developed for use with the current effector system:

- Slug
- 00 Buckshot
- Less-than-lethal rubber round

1 – Slug

According to the manufacturer the utilised projectile - thug slug - presents the following features.[6]

- Double-reinforced gas seal
- Quad-buckle self-adjusting cushion system
- Rifled ribbing engages rifling and reduces the friction of bore-contact
- Flat-nose design increases relative shock power
- Hollow-point design for maximum impact and rapid energy transfer

In a 30 cm smooth bore barrel the projectile achieves a lethality of:

- Velocity — 300 m/s
- Projectile mass — 28 g
- Penetration 20% Ballistic Gel — 24 cm

[3] http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA7075T6 (31.05.2019)

[4] https://www.vihtavuori.com/powder/n310-handgun-powder/ (31.05.2019)

[5] http://ns.hodgdon.com/PDF/MSDS%20Files/Smokeless/VihtaVuori/VihtaVuori_N100-N300.pdf (31.05.2019)

[6] https://www.ballisticproducts.com/Thug-Slug-12ga-1oz-25_pk/productinfo/1261228/ (31.05.2019)

Figure 4.23: Thug Slug as utilised in the effector.

2 – Buckshot

A round of 00-buckshot has been developed with a projectile mass of 24g. The shots are contained in a standard 12ga plastic wad.

3 – Less-than-lethal

A less-than-lethal round has been developed which uses a .68 rubber ball as projectile. This has a mass of approximately 3g. The rubber ball is contained in a standard 12ga plastic wad.

Plastic Disc

The pressure is contained by a 12ga frangible plastic overshot card. Designed for large shot sizes and Roll Turnover closures, these disc sits directly on top of the shot and fragment when leaving the barrel.[7]

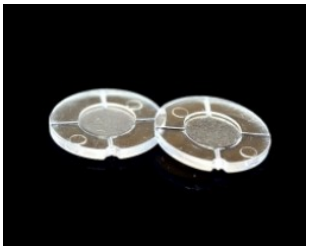


Figure 4.24 Plastic disc

Rocket motor ammunition

Cartridge

The cartridge is manufactured out of Aluminium 7075-T6, with the following physical properties[8]:

- Hardness, Vickers 175
- Ultimate Tensile Strength 572 MPa
- Tensile Yield Strength 503 MPa
- Modulus Elasticity 71.7GPa

Contrarily to the effector cartridge whose external diameter is smooth, the rocket motor cartridge has an external step in diameter. This design provides two functionalities:

[7] https://www.claygame.co.uk/12ga-frangible-discs-pd130 (31.05.2019)

[8] http://asm.matweb.com/search/SpecificMaterial.asp?bassnum=MA7075T6 (31.05.2019)

1- It makes it impossible to load the cartridge in the wrong chamber: the bottom step of the rocket motor cartridge does not allow insertion of the rocket motor cartridge inside the effector chamber ; the internal diameter of the rocket motor chamber is too small for insertion of the effector cartridge inside the rocket motor.
2- This external feature is easily identifiable by the user, even in the dark, which renders manipulation of the ammunition and reloading of the system easier and safer in low visibility conditions.

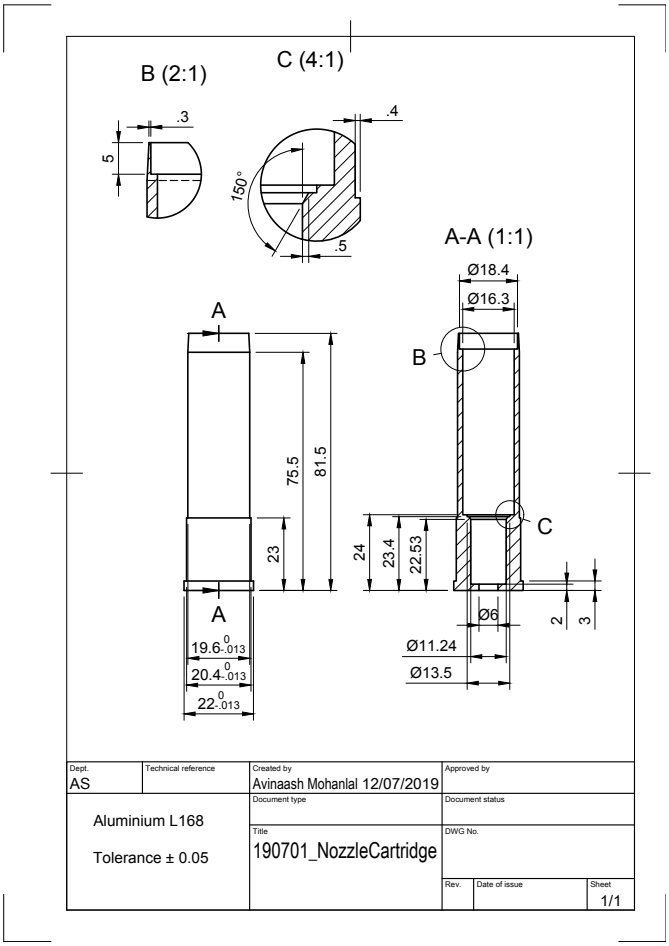


Figure 4.25: Design of rocket motor cartridges

From bottom to top the cartridge contains the components as listed below:

- Primer assembly
- Paper Disc
- Propellant
- Fillers
- Plastic Disc

Propellant

The cartridge is filled with 60gn of propellant.

Fillers

4 12ga felt cushion fillers are used in the rocket motor ammunition.

Exhaust

The exhaust guides the high-velocity gases exiting the nozzle of the rocket motor. Those gases are highly energetic and can damage the components they encounter. More importantly, the shock wave that would result from the interaction of a components with these gases can lead to considerable structural damage to the platform.

The exhaust itself is made of prepreg carbon fibre containing unidirectional layers in both longitudinal and circumferential directions of the tube to ensure integrity of the exhaust when exposed to both longitudinal and hoop stresses.

Section 4.6 TES Systems Equipment

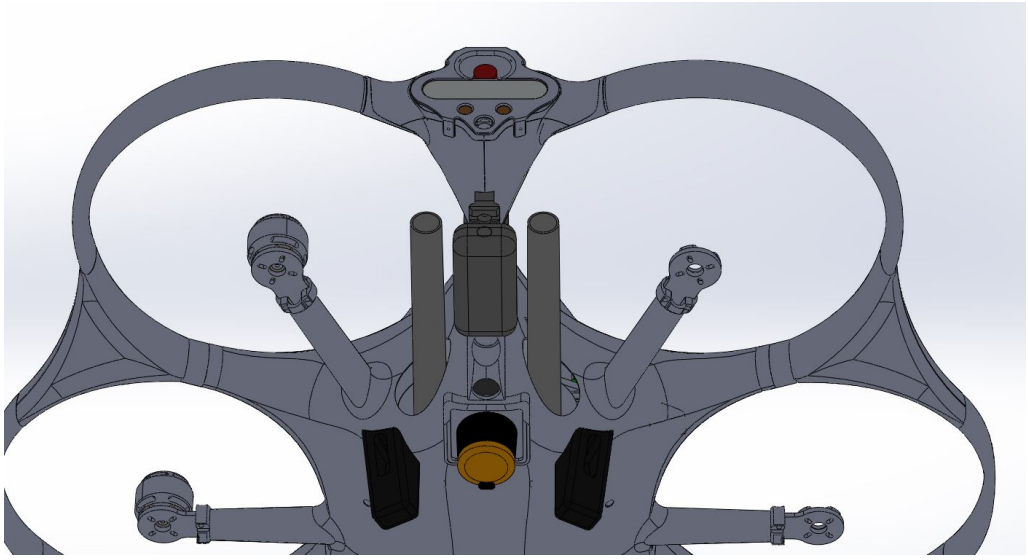
The platform and effector will be supplied a first in a training version which is compatible with Saab's Tactical Engagement Simulation (TES) system. The TES system is broadly used by the MoD and various militaries around the world to practice real battle conditions using laser transmitters and receivers.

Safety

In order to render the practice exercise safe while using the TES system, the effector system is rendered inert by removing all electric circuitry from the breech block. Furthermore, a training software intended for use with the TES system is used on the trigger board instead of the software used for regular ammunition engagement. Dummy barrels and nozzle will also be used, making sure however that the weight distribution of the aircraft will not be altered.

Attachment

The main piece of equipment of the TES system is the Small Arms Transmitter (SAT) which can be mounted onto firearms and triggers a laser to simulate weapon firing. The SAT comes with a bracket that is usually fitted onto the barrel of the weapon. Because the WE-01 comprises 2 effector systems, the SAT has been placed between the two barrels and is attached to the front camera module using the supplied bracket, as can be seen on Figure 4-26 .



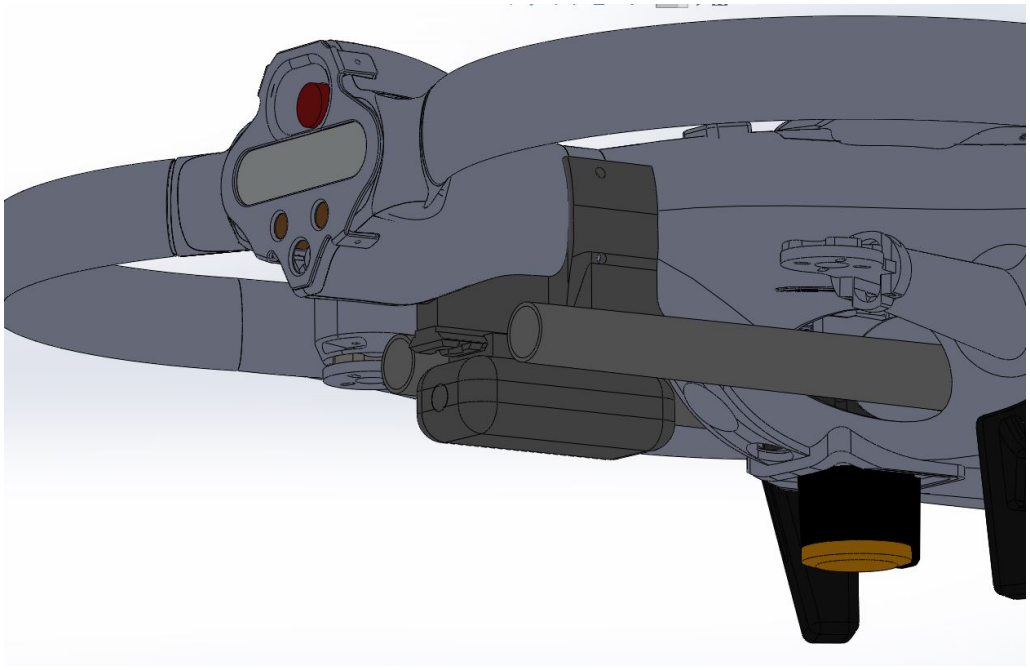


Figure 4-26: SAAB Small Arm Transmitter for TES system mounted on WE-01

Triggering

The SAT is triggered through the trigger board installed on the drone as part of the effector system. However, the software used in this training mode is specific to the triggering the SAT and prevents any triggering of real ammunition in case the effector was to be inadvertently loaded.

Integration

WECORP INDUSTRIES has currently completed all upstream work for integration of Saab's TES Small Arms Transmitter onto its platform and is now waiting for Saab to supply the SAT units for final integration.

Section 4.7 Artificial Intelligence

The Artificial Intelligence (AI) subsystem is functioning as a component package within ROS (Robotics Operating System) and is written solely in python2. The main function of the AI is to assist with the aiming by estimating the position of the target in 3D space.

The process starts with the AI component receiving data from the RealSense D435 stereo camera through ROS. The image is passed through a Deep Learning Artificial Neural Network (ANN) trained for object detection. The ANN outputs a bounding box for each object. Tracking methods are then run alongside detection to create a robust full system.

Finally, based upon the bounding boxes, an estimation of the objects' positions in 3D space is obtained, which is then passed on and combined with data acquired from the Lidar in a process named Sensor Fusion, which results in a more accurate position of the target.

The logic within the AI subsystem is dependent upon the 'phase' in which the user activates. The desired action of the user coming from the radio transmitter user interface is decoded in ROS and passed on to the AI subsystem. The action is broken up into the following phases:

- Detection Phase (default):
  Object detection from the ANN and correlation tracking are run to track multiple objects on the RealSense D435 camera feed. Only the bounding boxes for the objects are published with ROS.
- Selection Phase:
  The same methods are run as in the detection phase, but the user may now select a target from the available objects detected. The detected bounding boxes are sorted from left to right to enable the user to cycle through them. The bounding box and the 3D coordinates of the selected object are published with ROS.
- Targeting Phase:
  Once a target is selected this final stage is initialised. The selected target is now the only object tracked using deep-feature tracking and assisted by LIDAR-only tracking. The bounding box and the 3D coordinates of the targeted object are published with ROS. The targeting phase is activated when the drone is put in to 'off-board'. The period in which the phase is active is according to the target's distance.

In 'Off-board' mode the yaw of the drone is determined so that the drone points towards the target. Throughout the phase the drone tracks the target by yawing towards the coordinates outputted by the algorithms in the AI subsystem. The user always has the option to regain manual control of the yaw via the radio transmitter user interface.

The coordinates of the target are calculated, filtered and gated before being published through ROS and passed to the Sensor Fusion package to produce the final action commands. Just before the coordinates are sent, we implement a 'fail-safe' to ensure we are not sending erratic data. This is done through gating and a 'flag' system.

The flag system involves the AI subsystem communicating with control and ensuring that the movement of the drone within off-board mode is stable and robust. The flag either tells control to trust the 3D coordinates that are outputted or not to trust them. On top of this, during the targeting phase, whilst the drone is in off-board and is tracking the target, we record a small history of target positions. If a new position outputted by the ANN and feature tracking is such that is a significant leap from the recent history, then it is disregarded, and the drone remains steady.

Flag system pseudocode:

```
Flag:

0 - xyz from detection boxes is trusted
1 - No confidence in xyz.
2 - No confidence in xyz. Use LIDAR only tracking.

flag = 1

WHILE:
  IF signal_on_selecting == 1:
```

```
    Use xyz from detection boxes (flag = 0)

    IF selection timer > selection time:
        Reset detection (flag = 1)

    IF signal_off_board == 1:
        Run LIDAR-only tracking (flag = 2)
        sleep(0.4s)
        Use xyz from detection boxes (flag = 0)

IF signal_off_board == 1 AND features:
    IF detection box:
        IF features match:
            Use xyz from detection boxes (flag = 0)

        ELSE:
            Run LIDAR-only tracking (flag = 2)

    IF tracking timer > timeout time: (tracking duration currently based on target distance)
        Reset detection (flag = 1)
```

The AI component package comprises of 4 modules:

1. Mobile Net SSD V2

   The MobileNetSSDv2 deep learning model is open-source and trained on the COCO (Common Objects in Context) training set, which detects bounding boxes based on the SSD (Single Shot Multi Box Detector) algorithm designed for object detection in real-time. The model has been compressed in-house into a TensorRT* model to speed up the inference time of detection on each frame. Subsampling was also carried in order to limit object detections to people only.

2. Correlation tracking

   Tracks the position of an object as it moves from frame to frame in a video sequence. The correlation tracker used is from the python API for the Dlib library.

2. Deep tracking (Deep Feature Tracking)

   Generates features by running a Deep Learning model pretrained on the MARS (Motion Analysis and Re-identification) dataset. Those features are used for person re-identification and comparison of the visual appearance of bounding boxes using cosine similarity. If there is no match with previous features held in memory, then tracking using the LIDAR data is used for a short period of time. If, however, there is a match then, filtering and gating runs on coordinates extracted from matched bounding box.

4. Targeting

   Calculates and filters x, y and z coordinates of the target. The conversion from the 2D bounding box to the 3D coordinates in space uses the intrinsics of the RealSense D435 camera: focal length, the optical centre (principal point), and the skew coefficient.

Deprojection takes a 2D pixel location on a stream's images, as well as a depth, specified in meters, and maps it to a 3D point location within the stream's associated 3D coordinate space. The coordinates are then filtered using a Kalman Filter and subsequently gated to remove outliers.

5. Index Left-to-Right

   Simply sorts the detection bounding boxes from left to right for the user target selection process.

* TensorRT is a platform for high-performance deep learning inference. It includes a deep learning inference optimizer that delivers low latency and high throughput. We apply a lower floating-point precision (FP16) optimizations. Reduced precision inference significantly reduces application latency, which is a requirement for many real-time services, auto and embedded applications. These network optimizations do not change the underlying computation in the network, the final output remains the same. Instead, they look to restructure the network to perform the operations much faster and more efficiently.

A diagram of the AI subsystem code layout can be seen in the figure below. The boxes marked "ROS IN" are variables that the AI subsystem subscribes to from elsewhere. The boxes marked "ROS OUT" are variables that the AI subsystem publishes for other ROS packages to use.

Section 4.8 Pseudocode for the AI component main logic

```
INIT: detection model
INIT: tracking model
INIT: correlation tracker
INIT: Left-to-Right index
INIT: Target

Detecter:
    INIT: Ros Node
    INIT: cv bridge
    INIT: Subscribers and publishers
    INIT: Timeout Parameters
    INIT: Variables

    WHILE rospy is running:
        IF frame:
            GET: detection bounding boxes FROM: detection model on frame
            GET: tracking bounding boxes FROM: tracker, frame & detection bounding boxes
            PUBLISH: tracking bounding boxes

            IF tracking signal is ON
                tracking state = 1
                detection status = 0

                IF selection frame count > selection timeout frames:
                    detection status = 1
                    Go back to detection parameter settings
```

```
            GET: left-to-right bounding boxes FROM: tracking bounding boxes
            PUBLISH: left-to-right bounding boxes

            GET: target ID FROM: tracking bounding boxes & signal target ID
            GET: selected bounding box FROM: tracking bounding boxes & target ID
            PUBLISH: selected bounding box

            IF target ID is not 0:
               GET: target coordinates FROM: tracking bounding boxes & target ID
               IF off board signal = 1:
                  GET: selected bounding box FROM: tracking bounding boxes & target ID
                  GET: selected bounding box features FROM: frame & selected bounding box
                  PUBLISH: empty selected bounding box

                  detection status = 2
                  rospy sleep for set time
                  detection status = 0
                  Go back to detection parameter settings

         ELSE:
            tracking state = 0

         IF off board signal = 1 AND selected bounding box features != 0:
            GET: left-to-right bounding boxes FROM: tracking bounding boxes

            INIT: updated selected bounding box

            IF: distance to target > 8:
               feature tracking threshold = 0.2
            ELSE:
               feature tracking threshold = 0.4

            IF left-to-right is not empty:
               GET: features distance FROM: frame, selected bounding box features & left-to-right bounding
boxes
               GET: features tracking ID FROM: features distance & feature tracking threshold

               IF features tracking ID != 1:
                  RUN FEATURE TRACKING
                     detection status = 0
                     GET: updated selected bounding box FROM: left-to-right bounding boxes & features tracking
ID
                     GET: selected bounding box features FROM: frame & updated selected bounding box
                     GET: target coordinates from updated selected bounding box
               ELSE:
                  RUN LIDAR TRACKING
                     detection status = 2
```

```
            IF tracking frame count > tracking timeout frames:
                detection status = 1
                Reset Deep Features and parameters
            tracking frame count ++

        GET: distance to target FROM: target coordinates
        GET: tracking timeout frames
        IF: off board signal = 0:
            tracking frame count = 0
            tracking timeout frames = 0

        IF: tracking timeout frames != 0:
            percentage left to timeout = tracking frame count / tracking timeout frames
        ELSE:
            percentage left to timeout = 0

        Construct target message
    rate sleep
Close detection model
```

Chapter 5 Flight envelope testing

Section 5.1 Testing conditions

Table 5-1: Aircraft operation conditions

Model	June	**Pilot**	Maj Karl Eze
Date	06/08 & 08/08 2019	**Engineer**	Mr Samuel Garcin
Weight (kg)	7.79	**Engineer**	Mr Fernando Acero
		Data Analyst	Mr Samuel Garcin

Table 5-2: Environment conditions

Elevation (m a.s.l.)	93	**Weather**	Sunny
Temperature (C)	23	**Wind speed (m/s)**	1.5 – 4.5

Section 5.2 Results

Below are the flight envelope parameters as requested by the 700X NAS.

Table 5-3 Wecorp WE-01 Basic Specifications.

Expected Hovering Accuracy Vision Positioning Active	XY: 0.15m Z: 0.06m
Expected Hovering Accuracy Vision Positioning Disabled	XY: 2.5 m/s Z: 0.8 m/s (dependent windspeed)
Max Ascent Speed	9 m/s (8 m/s limited by software)
Max Descent Speed	11 m/s (0.6 m/s limited by software)
Max Service Ceiling (MSL)	3500m (Theoretically calculated)
Max Wind Speed Resistance	10 m/s
Max Weight	8.1 kg
Max Forward Speed	20 m/s (3 m/s limited by software)
Max Flight Time	20 minutes

Section 5.3 Stability and manoeuvrability

Position control flight – Vision

As observed in table 4, reliable performance was achieved during position hold flights using the T265 camera as a Visual Inertial Odometry source.

Upon flying, a consistent divergence from reality was observed of the VIO data. This resulted in the PX4 EKF to fail multiple times and revert to altitude hold. These are the divergences observed:

- The speed estimate using VIO is significantly below the GPS estimate. This is especially true at high speeds.
- Upon braking, the VIO estimates a change of speed direction, whereas in reality the speed is merely coming to 0.
- VIO data seems to become unreliable when looking at the sky (lack of discernible visual features) or when looking at the sun (video feed presented glare, over exposure, twinkling).
- Positive vertical speeds were measured at high speeds even when the aircraft was level with the ground.

Those issues may be solved by the implementation of the Robot Localization EKF, or by using the D435/D435i as an additional source of VIO so that no 2 cameras are looking at the sky or at the sun at the same time. This is to be confirmed with further tests.

Table 5-4 Position hold performance using VIO.

Position max variance XY (m)	Position max variance Z (m)	Velocity max variance XY (m/s)	Velocity max variance Z (m/s)
0.13	0.02	0.15	0.03

Position control flight – GPS

The GPS managed to connect to a high number of satellites (20) at the test site. The connection was reliable and steady. When using GPS only, the aircraft path started diverging into an expanding spiral, when no control inputs where applied. It was concluded that the current GPS accuracy is too low to be used as a unique source of positioning. It is recommended to upgrade the module, place it further away from the body or only use it as a supporting sensor. To use as a supporting sensor more work will be required in order to maintain the VIO and GPS axis aligned. It is recommended we move to a more reliable and recent GPS chip architecture and fuse it into our own EKF running on the TX2. For the time being it has been disabled.

Altitude control flight

Altitude control was enabled using the 1D Lidar. This sensor is by far the most reliable when used outdoors. Sunlight did not seem to affect it and reliable data was acquired at up to 30m height.

Max Ascent and Descent speeds

Ascent and Descent speeds were tested in stabilised mode to circumvent software speed limitations. Airmode was enabled to maintain roll and pitch authority at low and high thrust levels.

Measured ascent speed was up to 9.8 m/s however it was rounded down to 9.0 m/s to account for extra weight and measurement errors.

It is to be noted that descent speed can exceed 0.6 m/s when above a certain height by changing the PX4 parameters. However, upon landing the maximum descent speed is capped to 0.6 m/s.

Wind resistance

Given the maximum speed achieved by the craft during testing in headwind conditions (19.62 m/s [70.6 km/h] at 35 degrees pitch), it can be safely estimated that the aircraft will not suffer manoeuvrability issues due to a lack of available thrust when subjected to wind speeds of at least 10 to 15 m/s (36 to 54 km/h).

The limiting factor here would be the stability of the craft and its ability to hold position when subjected to gusts with large variation in direction and intensity. On June 6th we recorded gusts of up to 10 m/s, which the aircraft remained able to handle. This resulted in a higher position variance in position hold, however the craft was never in a situation of instability or pilot loss of control.

Due to lack of exposure to wind speeds higher than 10 m/s the decision was made to set this as the limiting operation capability of the drone for now. This limit may be revised after wind tunnel testing.

Section 5.4 Endurance and range

Endurance

When indoors and stationary, the aircraft stayed in the air 24 minutes before depleting its battery (note: confirm weight).

When flying outdoors at an average speed of 1.25 m/s, the craft stayed in the air 20:56 minutes, landing at 17% battery (recorded before landing).

The operating endurance was rounded down to account for flights at max weight, which is an extra 300g from the tested weight.

Range

A range analysis was performed by flying the craft in Altitude control mode at full pitch until terminal horizontal velocity was reached. The maximum pitch angle was limited in the Flight control software. The pitch angle tested ranged from 15 to 35 degrees, in step of 5 degrees. The manoeuvre was repeated in backwind and headwind and the measured velocity was averaged between the flights in order to eliminate the influence of wind.

The VIO velocity data proved to be unreliable and underestimated so the GPS velocity data was used instead. The amp draw and battery voltage was recorded once a constant speed was reached. Using the power law, the power consumed by the craft was calculated. The total energy in the battery was obtained from manufacturer's datasheet and a sagging coefficient of 0.7 was applied from past observation. It means that 70% of the battery total energy is available for consumption during flight. Lastly, the estimated flight time was obtained from the power draw and available energy.

The following assumptions were made to estimate the range:

- GPS is a reliable enough source of data for velocity.
- The craft is moving at a constant speed. Transient effects in acceleration and deceleration are ignored due to only lasting a few seconds for multicopters.
- The battery sagging coefficient was estimated from past data. Sagging increases proportionally with power draw; therefore, a conservative estimate was used to not overestimate the range.
- The craft remains at the same altitude during flight. This was confirmed with an analysis of the range sensor data, recording vertical velocity of a maximum of 0.6 m/s.
- The range was calculated for an altitude of 93 m a.s.l. and at a temperature of 23 C. Range characteristics depend on air density and therefore will vary for different conditions. It is a fair assumption that the conditions of the test are equivalent to standard sea level atmospheric conditions.
- Sidewind effects are negligible.
- Power required to take off and land are negligible / included in the sagging coefficient.

The results can be observed in *Figure* 5.1. A second order fit is applied due to the quadratic relation between speed and range caused by the drag force. A maximum range of 20 km is achievable when flying at 20 to 23 m/s. It is to be noted that the vertex was not reached during this testing. This means that a higher range could theoretically be reached by flying at higher speeds. It also means that the fit would be made better by obtaining data points at higher speeds.

Nonetheless, when comparing the real data obtained from the endurance test flight with the fitted model, it appears that the model predicts the aircraft's range well.

It is to be noted that, in real operating conditions, the range will also be limited by the operating range of RC and video communication between the operator and the aircraft.

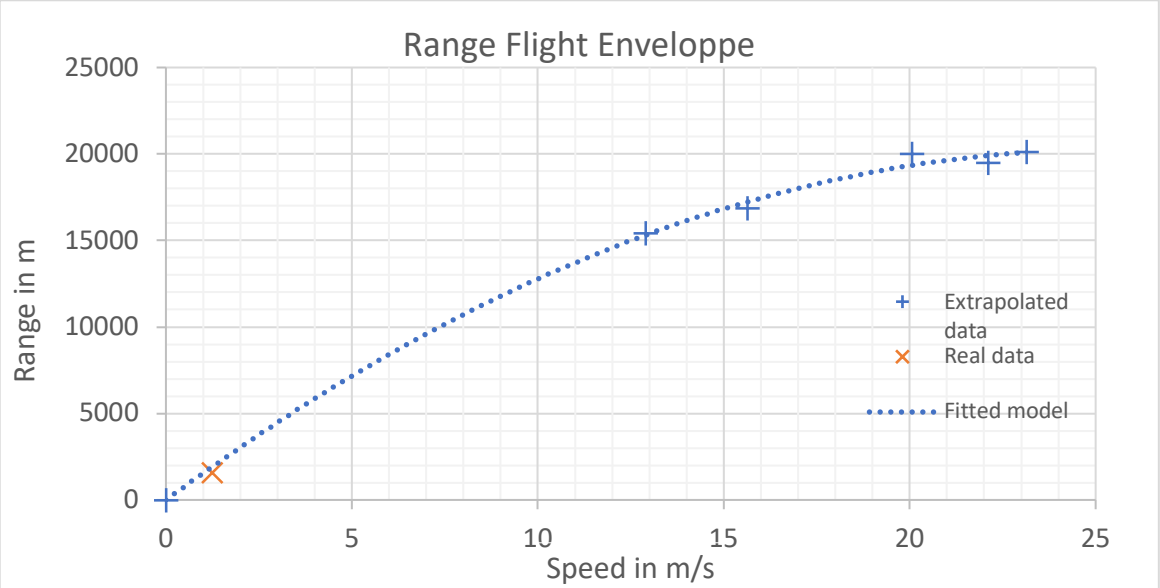


Figure 5.1 Range estimated from power draw measurements compared to real range achieved.

Section 5.5 Aerodynamic parameters

Service ceiling

The service ceiling was calculated to be the maximum altitude at which the aircraft maintains a thrust to weight ratio of at least 1.5. At sea level, the thrust to weight ratio of the craft is 2.1. Thrust is produced by lift and is therefore directly proportional to air density. Therefore, the ratio of air density at sea level ρ_{atm} to the air density at service ceiling ρ_{cei} must obey the relation

$$\frac{\rho_{atm}}{\rho_{cei}} = \frac{T/W_{atm}}{T/W_{cei}} = \frac{2.1}{1.5} = 0.714$$

Which corresponds to $\rho_{cei} = 0.875$ kg/m^3. Looking up figure 2, this corresponds to an operating service ceiling of 3500 m. As in the drone can safely be operated in elevations of up to 3500 m above sea level.

Aerodynamic forces

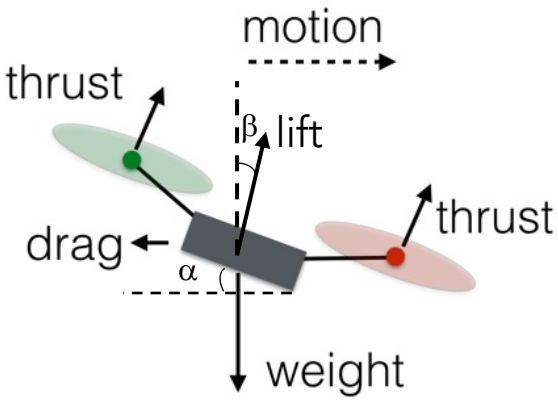


Figure 5.2 Free body diagram of aircraft in steady flight.

Figure 5.2 presents a free body diagram of the forces acting on the drone during steady flight. The nomenclature is as follow:

- α is the pitching angle of the drone, it is also the angle of the thrust vector with respect to the gravity vector.
- β is the angle of the velocity vector of the drone with respect to the ground. Additionally it defines the angle of the Lift and Drag forces with the vertical and horizontal coordinates, respectively.
- T is the thrust force
- W is the aircraft's weight
- L and D are the Lift and Drag forces, respectively.

Balancing the forces horizontally and vertically we obtain the equations

$$T\cos\alpha = W - L\cos\beta + D\sin\beta$$
$$T\sin\alpha = D\cos\beta - L\sin\beta$$

It is then possible to obtain the lift and drag forces.

Lift

The equilibrium equations have three unknowns (T, L and D) for 2 equations: they are under constrained. It is necessary to eliminate one of these unknowns. Given that the drone is flying on a nearly horizontal path (recorded maximum velocity of 0.6 m/s), the horizontal component of Lift can be ignored, and its vertical component can be incorporated into the Thrust. However, doing so makes it impossible to calculate the Lift force at this time.

Once we are able to measure the Thrust force, we can obtain the Lift. A way to measure the thrust force is to measure the PWM values of each motor and use thrust stand data to obtain how much thrust each motor is outputting. An efficiency factor can then be calculated by comparing the Thrust exerted by the aircraft at hover with the Thrust expected from the thrust stand data at hover PWM. Assuming that the efficiency factor does not vary with craft orientation or speed, the measured Thrust will then simply be the efficiency factor multiplied by the thrust stand measured compound force.

In the meantime, the lift force will be ignored.

Drag

Manipulating the aforementioned equations, we obtain the drag force as

$$D = \frac{W \tan\alpha + L(\sin\beta - \cos\beta)}{\cos\beta - \sin\beta \tan\alpha}$$

Ignoring the Lift we can solve for the Drag force. Next the Drag force can be expressed as

$$D = \frac{1}{2}\rho_\infty V_\infty^2 C_D S_{ref}$$

ρ_∞ can be obtained from the environmental conditions and V_∞ is calculated thanks to

$$V_\infty = \sqrt{(V_x - V_w)^2 + V_z^2}$$

Where V_x and V_z are the horizontal and vertical ground speeds measured by the drone's sensors and V_w is the velocity of the wind (headwind is positive).

The low measured values of V_z barely affect the magnitude of V_∞. It is to be noted that for these tests the value of β to be below 2 degrees at most and therefore it would be a reasonable assumption to ignore it in the model.

In order to estimate V_w an average velocity is taken between the headwind and backwind flights. In other words, the average speed between the two flights is taken as $(V_x - V_w)$.

The parameter S_{ref} is usually taken to be the surface of the wing planform for a conventional aircraft. Here it is not relevant, and it makes more sense to group C_D and S_{ref} together. The grouped drag coefficient is plotted against the pitch angle in Figure 5.3.

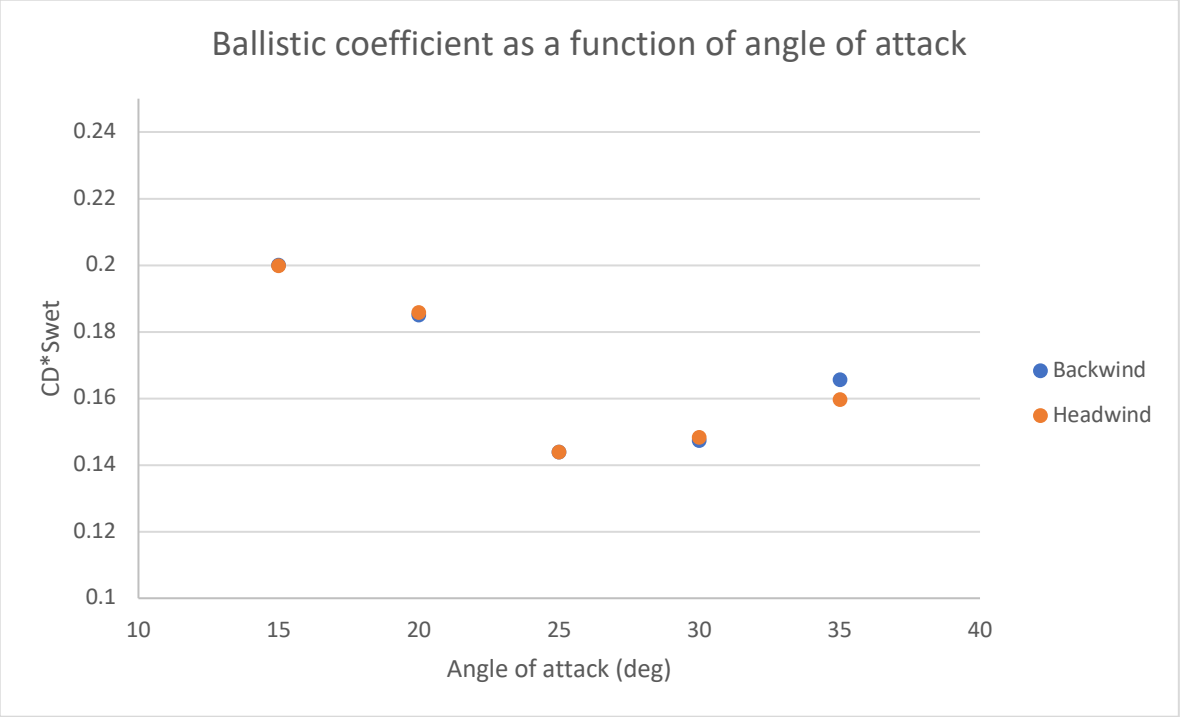


Figure 5.3 Grouped drag coefficient against pitch angle.

It can be observed that the grouped drag coefficient declines sharply between 15 and 25 degrees. It then slightly increases between 25 and 35 degrees. In any case, the grouped drag coefficient remains bounded between 0.14 and 0.20. The decline goes against the intuition that the exposed area becomes larger as the pitch angle increases and therefore that the grouped drag coefficient should increase. The two possible interpretations are that either the velocity measurements are inaccurate or that the lift force is playing a significant role and that the Thrust force is lower than expected. These assumptions will be tested during future wind tunnel tests.

Appendix A

A.1 Trigger Board Code

```
////////////////////////////////////////////////
//
// Global Variables and Libraries
//
////////////////////////////////////////////////

/*
 * Switch between PWM Signal (Pixhawk) and I2C Signal (Companion Computer) Connection
 * PWM_I2C == 1 - takes PWM Signal
 * PWM_I2C == 2 - takes I2C Signal
 */

 int PWM_I2C = 1; // input required as stated above




// ---------------- Libraries and Global Definitions
#include <Wire.h>   // used for I2C communication
#define slave_address 0x01 // I2C address of this specific trigger board

// ---------------- Set Pins ----------------
// Input from the companion computer
int IN_Arm_B = 6; // (digital)(input) arm barrel 1 & 2
int IN_Fire = 7;   // (digital)(input) fire, counts for both weapons
int IN_System_Present = A0; // (analog)(input) provides feedback if the companion computer is powered
int IN_System_Enabler = A1; // (ananlog)(input) enables the trigger board in case the correct code is transmitted

// Input from the trigger board
int IN_Continuity_Check_B1 = A3; // (anaglog)(input) continuity barrel 1
int IN_Continuity_Check_B2 = A6; // (analog)(input) continuity barrel 2
int IN_Cap_Charger_Level = A2;  // (analog)(input) read capacitor charge level

// Output to the trigger board
int OU_Arm_B1 = 8;  // (digital)(output) arm barrel 1
int OU_Arm_B2 = 9;  // (digital)(output) arm barrel 2
int OU_Fire = 10;   // (digital)(output) fire, counts for both weapons
int OU_Safety_Discharge = 11; // (digital)(output) safety discharge of capacitor
int OU_Cap_Charge_Switch = 12;  // (digital)(output) activates/deactivates capacitor charging
int OU_Error_Check_Connection_B1 = 3; // (digital)(output) lights up LED if continuity check succesfull
int OU_Error_Check_Connection_B2 = 4; // (digital)(output) lights up LED if continuity check succesfull
int OU_Visual_Feedback_Weapon_Armed = 5; // (digital)(output) lights up LED if weapon is armed

// ---------------- Variables ----------------
```

```
// input from the Companion Computer
int Arm_B_Master = 0;
int Fire_Master = 0;

// other varialbes used for internal processing
int System_Present = 0;
int System_Enabler = 0;
// Feedback variable - defined later
int Continuity_Check_B1 = 0;
int Continuity_Check_B2 = 0;
int Cap_Charger_Level = 0;
int Cap_Charger_Level_Feedback = 0;

int counter = 0; // counter used to skip the processor booting process and error signals

int cases_level_1 = 0; // used for switch-case function
int cases_level_2 = 0; // used for switch-case function
int cases_level_3 = 0; // used for switch-case function

int cases_level_1_slave = 0; // used for feedback
int cases_level_2_slave = 0; // used for feedback
int cases_level_3_slave= 0; // used for feedback

////////////////////////////////////////////////
//
// Setup to Define Pins
//
////////////////////////////////////////////////
void setup()
{
  Serial.begin(9600);

  // join i2c bus as a slave
  Wire.begin (slave_address);   // sets slave address
  Wire.onRequest(requestEvent); // function send out data
  Wire.onReceive(receiveEvent); // function receive date

  // Input from the companion computer
  pinMode(IN_Arm_B, INPUT);
  pinMode(IN_Fire, INPUT);
  pinMode(IN_System_Present,INPUT);
  pinMode(IN_System_Enabler,INPUT);

  // Input from the trigger board
  pinMode(IN_Continuity_Check_B1, INPUT);
  pinMode(IN_Continuity_Check_B2, INPUT);
  pinMode(IN_Cap_Charger_Level, INPUT);
```

```
  // Output to the trigger board
  pinMode(OU_Arm_B1, OUTPUT);
  pinMode(OU_Arm_B2, OUTPUT);
  pinMode(OU_Fire, OUTPUT);
  pinMode(OU_Safety_Discharge, OUTPUT);
  pinMode(OU_Cap_Charge_Switch, OUTPUT);
  pinMode(OU_Error_Check_Connection_B1, OUTPUT);
  pinMode(OU_Error_Check_Connection_B2, OUTPUT);
  pinMode(OU_Visual_Feedback_Weapon_Armed, OUTPUT);

  // initialising pin settings for safety reason and indicating booting process
  for (int i = 0; i <= 5; i++)
  {
  digitalWrite(OU_Arm_B1,LOW);
  digitalWrite(OU_Arm_B2,LOW);
  digitalWrite(OU_Fire,LOW);
  digitalWrite(OU_Safety_Discharge, HIGH);
  digitalWrite(OU_Cap_Charge_Switch, LOW);
  // visual feedback
  digitalWrite(OU_Error_Check_Connection_B1,HIGH);
  digitalWrite(OU_Error_Check_Connection_B2,HIGH);
  digitalWrite(OU_Visual_Feedback_Weapon_Armed,HIGH);
  delay(200);
  digitalWrite(OU_Error_Check_Connection_B1,LOW);
  digitalWrite(OU_Error_Check_Connection_B2,LOW);
  digitalWrite(OU_Visual_Feedback_Weapon_Armed,LOW);
  Serial.println("booting");
  delay(200);
  }
  Serial.println("all pins safe");
}

/////////////////////////////////////////////////
//
// Start the Main Loop
//
/////////////////////////////////////////////////
void loop()
{
  /*
  Serial.print("Master_Arm ");
  Serial.print(Arm_B_Master);
  Serial.print(" | Master Fire ");
  Serial.println(Fire_Master);
  */

  // ---------------------------------------------- skipping the first iterations
  // first loop required to skip the first 20 processor loops in order to eliminate error signals
  // occuring during the processor booting process
```

```
 if (counter <= 20){
   counter = counter+1;
   delay(10);
 }

 // ---------------------------------------------- starting the main loop
 else
 {

   /*  initial conditions
    *  Voltage_System_Present (<= 1.0) = Companion Computer not powered -> disable triggering
    *  Voltage_System_Present (>= 1.0) = Companion Computer powedere -> enable triggering
    *  System Enable --- not implemented yet (e.g. transmitted code that enables triggering
    */

   // both initial coniditions need to be fulfilled to run the main logic
   System_Present = analogRead(IN_System_Present); // (ananlog)(input) A0
   //System_Enabler = analogRead(IN_System_Enabler);
   float Voltage_System_Present = System_Present * (5.0 / 1023.0);
   //float Voltage_System_Enabler = System_Enabler * (5.0 / 1023.0);

   /*
   Serial.print(" System_Present ");
   Serial.println(Voltage_System_Present);
   */

////////////////////////////////////////////////
//
// CHANGE THRESHOLD IN FINAL CONFIGURATION (e.g. >= 1.0)
//
////////////////////////////////////////////////

   if (Voltage_System_Present >= 0.10)
   {

  // ---------------------------------------------- Main Logic


//////////////////////////////////////////////////////
//
// Read Input Signals
//
//////////////////////////////////////////////////////

   // PWM signal input
   if (PWM_I2C == 1)
   {
     // reads signal input from Pixhawk
     Arm_B_Master = pulseIn(IN_Arm_B,HIGH);
```

```
    /*
     * LEVEL 2 - Master Input
     * case # 1 - not armed (Arm_B_Master == 1)
     * case # 2 - barrel 1 is armed (Arm_B_Master == 2)
     * case # 3 - barrel 2 is armed (Arm_B_Master == 3)
     */
    // converts PWM signal input to the Level 2 states (1-3)
    if (Arm_B_Master <= 1200)
    {
      Arm_B_Master = 1;
    }
    else if (Arm_B_Master >= 1200 && Arm_B_Master <= 1700)
    {
      Arm_B_Master = 2;
    }
     else
    {
      Arm_B_Master = 3;
    }

    // reads signal input from Pixhawk
    Fire_Master = pulseIn(IN_Fire,HIGH);
    /* LEVEL 3 - Master Input
     *  case # 1 - not firing (Fire_Master == 1)
     *  case # 2 - firing (Fire_Master == 2)
     */
    if (Fire_Master <= 1200)
    {
      Fire_Master = 1;
    }
    else
    {
      Fire_Master = 2;
    }
  }

  // Analog voltage signal
  Continuity_Check_B1 = analogRead(IN_Continuity_Check_B1); // pin A3
  Continuity_Check_B2 = analogRead(IN_Continuity_Check_B2); // pin A4
  float Voltage_Continuity_Check_B1 = Continuity_Check_B1 * (5.0 / 1023.0); // convert the analog reading
(which goes from 0 - 1023) to a voltage (0 - 5V)
  float Voltage_Continuity_Check_B2 = Continuity_Check_B2 * (5.0 / 1023.0); // convert the analog reading
(which goes from 0 - 1023) to a voltage (0 - 5V)
  // Voltage_Continuity_Check_B1 == 5 V -> No continuity
  // Voltage_Continuity_Check_B1 == 0 V -> Continuity

   Cap_Charger_Level = analogRead(IN_Cap_Charger_Level);
   float Voltage_Cap_Charger_Level = Cap_Charger_Level * (5.0 / 1023.0); // convert the analog reading (which
goes from 0 - 1023) to a voltage (0 - 3.3V)
```

```
// Voltage_Cap_Charger_Level == 3.3 V (3.0 V) -> Charged
// Voltage_Cap_Charger_Level == 0 V -> Not Charged

Serial.print(" | Cap ");
Serial.println(Voltage_Cap_Charger_Level);

/*
Serial.print(Voltage_Continuity_Check_B1);
Serial.print("   ");
Serial.println(Voltage_Continuity_Check_B2);
*/

// Provides visual feedback on the capacity charge level
if (Voltage_Cap_Charger_Level >=2.0)
{
 digitalWrite(OU_Visual_Feedback_Weapon_Armed,HIGH); // LED turns on when capacitor is charged
 Cap_Charger_Level_Feedback = 1;
}
else
{
 digitalWrite(OU_Visual_Feedback_Weapon_Armed,LOW); // LED turns of when capitor is uncharged
 Cap_Charger_Level_Feedback = 0;
}

  // ---------------- Conditions and Cases ----------------
  /*
   * LEVEL 1
   * case # 1 - Continuity_Check_B1 && Continuity_Check_B2 = FAIL (>=4.5V)
   * case # 2 - Continuity_Check_B1 = OK (<=4.5V) && Continuity_Check_B2 = FAIL (>=4.5V)
   * case # 3 - Continuity_Check_B1 = FAIL (>=4.5V) && Continuity_Check_B2 = OK (<=4.5V)
   * case # 4 - Continuity_Check_B1 = OK (<=4.5V) && Continuity_Check_B2 = OK (<=4.5V)
   */
   if (Voltage_Continuity_Check_B1 >= 4.5 && Voltage_Continuity_Check_B2 >= 4.5){
     cases_level_1 = 1;
   }
   if (Voltage_Continuity_Check_B1 <= 4.5 && Voltage_Continuity_Check_B2 >= 4.5){
     cases_level_1 = 2;
   }
   if (Voltage_Continuity_Check_B1 >= 4.5 && Voltage_Continuity_Check_B2 <= 4.5){
     cases_level_1 = 3;
   }
   if (Voltage_Continuity_Check_B1 <= 4.5 && Voltage_Continuity_Check_B2 <= 4.5){
     cases_level_1 = 4;
   }


   /*
    * LEVEL 2 - Master Input
    * case # 1 - not armed (Arm_B_Master == 1)
```

```
       * case # 2 - barrel 1 is armed (Arm_B_Master == 2)
       * case # 3 - barrel 2 is armed (Arm_B_Master == 3)
      */
       if (Arm_B_Master == 1){
        cases_level_2 = 1;
       }
       else if (Arm_B_Master == 2){
        cases_level_2 = 2;
       }
       else if (Arm_B_Master == 3){
        cases_level_2 = 3;
       }
       else
       {
        cases_level_1 = 1;    // program remains in default state due to faulty input
       }

        /* LEVEL 3 - Master Input
         *  case # 1 - not firing (Fire_Master == 1)
         *  case # 2 - firing (Fire_Master == 2)
        */
        if (Fire_Master == 1){
          cases_level_3 = 1;
        }
        else if (Fire_Master == 2){
          cases_level_3 = 2;
        }
        else
        {
          cases_level_1 = 1;    // program remains in default state due to faulty input
        }

//////////////////////////////////////////////////
//
// Execution
//
//////////////////////////////////////////////////


////////////////// LEVEL 1 - Case 1 //////////////////
        switch(cases_level_1){
          case 1:
          cases_level_1_slave = 1; // confirms status for master feedback
          cases_level_2_slave = 1; // confirms status for master feedback
          cases_level_3_slave = 1; // confirms status for master feedback
          //////////////////////////////////////////////////
          //
          //  Level 1 - case 1      Continuity Barrel 1 & 2 FAIL
          //
```

```
        //  *Continuity Barrel 1  == LOW
        //  *Continuity Barrel 2  == LOW
        //  *Arm Barrel 1         == LOW
        //  *Arm Barrel 2         == LOW
        //  *Capacitor Charge     == LOW
        //  *Safety Discharge     == HIGH
        //  *Fire                 == LOW
        //
        ////////////////////////////////////////////////
        digitalWrite(OU_Error_Check_Connection_B1,LOW); // de-activates LED 1 for Barrel 1
        digitalWrite(OU_Error_Check_Connection_B2,LOW); // de-activates LED 2 for Barrel 2
        digitalWrite(OU_Arm_B1,LOW); // set output pin Arm_B1 to LOW
        digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW
        digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
        digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
        digitalWrite(OU_Fire,LOW);
        break;




////////////////// LEVEL 1 - Case 2 //////////////////
        case 2:
        cases_level_1_slave = 2; // confirms status for master feedback
        cases_level_2_slave = 1; // confirms status for master feedback
        cases_level_3_slave = 1; // confirms status for master feedback
        ////////////////////////////////////////////////
        //
        //  Level 1 - case 2      Continuity Barrel 1 OK, Continuity Barrel 2 FAIL
        //
        //  *Continuity Barrel 1  == HIGH
        //  *Continuity Barrel 2  == LOW
        //  *Arm Barrel 2         == LOW
        //
        ////////////////////////////////////////////////
        digitalWrite(OU_Error_Check_Connection_B1,HIGH); // activates LED 1 for Barrel 1
        digitalWrite(OU_Error_Check_Connection_B2,LOW); // de-activates LED 2 for Barrel 2
        digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW

////////////////// LEVEL 1 - Case 2 //////////////////
////////////////// LEVEL 2 - Case 1 //////////////////
            switch(cases_level_2){
              case 1:
              cases_level_2_slave = 1; // confirms status for master feedback
              cases_level_3_slave = 1; // confirms status for master feedback
              ////////////////////////////////////////////////
              //
              //  Level 1 - case 2      Continuity Barrel 1 OK, Continuity Barrel 2 FAIL
```

```
            //  Level 2 - case 1      Barrel 1 not armed
            //
            //  *Arm Barrel 1         == LOW
            //  *Capactor Charge      == LOW
            //  *Safetly Discharge    == HIGH
            //
            ////////////////////////////////////////////////
            digitalWrite(OU_Arm_B1,LOW); // set output pin Arm_B1 to LOW
            digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
            digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
            break;

////////////////// LEVEL 1 - Case 2 //////////////////
////////////////// LEVEL 2 - Case 2 //////////////////
            case 2:
            cases_level_2_slave = 2; // confirms status for master feedback
            ////////////////////////////////////////////////
            //
            //  Level 1 - case 2      Continuity Barrel 1 OK, Continuity Barrel 2 FAIL
            //  Level 2 - case 2      Barrel 1 armed
            //
            //  *Arm Barrel 1         == HIGH
            //  *Capactor Charge      == HIGH
            //  *Safetly Discharge    == LOW
            //
            ////////////////////////////////////////////////
            digitalWrite(OU_Arm_B1,HIGH); // set output pin Arm_B1 to HIGH
            digitalWrite(OU_Cap_Charge_Switch,HIGH); // enables capacity charging circuit
            digitalWrite(OU_Safety_Discharge,LOW); // disables capacity discharge circuit

////////////////// LEVEL 1 - Case 2 //////////////////
////////////////// LEVEL 2 - Case 2 //////////////////
////////////////// LEVEL 3 - Case 1 //////////////////
              switch(cases_level_3){
                case 1:
                cases_level_3_slave = 1; // confirms status for master feedback
                ////////////////////////////////////////////////
                //
                //  Level 1 - case 2      Continuity Barrel 1 OK, Continuity Barrel 2 FAIL
                //  Level 2 - case 2      Barrel 1 armed
                //  Level 3 - case 1      no fire
                //
                //  *Fire                 == LOW
                //
                ////////////////////////////////////////////////
                digitalWrite(OU_Fire,LOW);
                break;

////////////////// LEVEL 1 - Case 2 //////////////////
```

```
////////////////// LEVEL 2 - Case 2 //////////////////
////////////////// LEVEL 3 - Case 2 //////////////////
                case 2:
                cases_level_3_slave = 2; // confirms status for master feedback
                ////////////////////////////////////////////////
                //
                //  Level 1 - case 2     Continuity Barrel 1 OK, Continuity Barrel 2 FAIL
                //  Level 2 - case 2     Barrel 1 armed
                //  Level 3 - case 2     fire
                //
                //  *Fire               == HIGH
                //  *Fire               == LOW (1s delay)
                //
                ////////////////////////////////////////////////
                digitalWrite(OU_Fire,HIGH);
                delay(1000);
                digitalWrite(OU_Fire,LOW);
                break;
              }
            break;
          }
        break;
```

```
////////////////// LEVEL 1 - Case 3 //////////////////
        case 3:
        cases_level_1_slave = 3; // confirms status for master feedback
        cases_level_2_slave = 1; // confirms status for master feedback
        cases_level_3_slave = 1; // confirms status for master feedback
        ////////////////////////////////////////////////
        //
        //  Level 1 - case 3     Continuity Barrel 2 OK, Continuity Barrel 1 FAIL
        //
        //  *Continuity Barrel 1  == LOW
        //  *Continuity Barrel 2  == HIGH
        //  *Arm Barrel 1         == LOW
        //
        ////////////////////////////////////////////////
        digitalWrite(OU_Error_Check_Connection_B1,LOW); // de-activates LED 1 for Barrel 1
        digitalWrite(OU_Error_Check_Connection_B2,HIGH); // activates LED 2 for Barrel 2
        digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW
```

```
////////////////// LEVEL 1 - Case 3 //////////////////
////////////////// LEVEL 2 - Case 1 //////////////////
            switch(cases_level_2){
              case 1:
              cases_level_2_slave = 1; // confirms status for master feedback
              cases_level_3_slave = 1; // confirms status for master feedback
              ////////////////////////////////////////////////
              //
              //  Level 1 - case 3      Continuity Barrel 2 OK, Continuity Barrel 1 FAIL
              //  Level 2 - case 1      Barrel 2 not armed
              //
              //  *Arm Barrel 2         == LOW
              //  *Capactor Charge      == LOW
              //  *Safetly Discharge    == HIGH
              //
              ////////////////////////////////////////////////
              digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B1 to LOW
              digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
              digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
              break;

////////////////// LEVEL 1 - Case 3 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
              case 3:
              cases_level_2_slave = 3; // confirms status for master feedback
              ////////////////////////////////////////////////
              //
              //  Level 1 - case 3      Continuity Barrel 2 OK, Continuity Barrel 1 FAIL
              //  Level 2 - case 3      Barrel 2 armed
              //
              //  *Arm Barrel 2         == HIGH
              //  *Capactor Charge      == HIGH
              //  *Safetly Discharge    == LOW
              //
              ////////////////////////////////////////////////
              digitalWrite(OU_Arm_B2,HIGH); // set output pin Arm_B2 to HIGH
              digitalWrite(OU_Cap_Charge_Switch,HIGH); // enables capacity charging circuit
              digitalWrite(OU_Safety_Discharge,LOW); // disables capacity discharge circuit

////////////////// LEVEL 1 - Case 3 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
////////////////// LEVEL 3 - Case 1 //////////////////
                switch(cases_level_3){
                  case 1:
                  cases_level_3_slave = 1; // confirms status for master feedback
                  ////////////////////////////////////////////////
                  //
                  //  Level 1 - case 3      Continuity Barrel 2 OK, Continuity Barrel 1 FAIL
```

```
                //  Level 2 - case 3      Barrel 2 armed
                //  Level 3 - case 1      no fire
                //
                //  *Fire                 == LOW
                //
                ////////////////////////////////////////////////
                digitalWrite(OU_Fire,LOW);
                break;

////////////////// LEVEL 1 - Case 3 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
////////////////// LEVEL 3 - Case 2 //////////////////
                case 2:
                cases_level_3_slave = 2; // confirms status for master feedback
                ////////////////////////////////////////////////
                //
                //  Level 1 - case 3      Continuity Barrel 2 OK, Continuity Barrel 1 FAIL
                //  Level 2 - case 3      Barrel 2 armed
                //  Level 3 - case 2      fire
                //
                //  *Fire                 == HIGH
                //  *Fire                 == LOW (1s delay)
                //
                ////////////////////////////////////////////////
                digitalWrite(OU_Fire,HIGH);
                delay(1000);
                digitalWrite(OU_Fire,LOW);
                break;
              }
            break;
          }
        break;




////////////////// LEVEL 1 - Case 4 //////////////////
        case 4:
        cases_level_1_slave = 4; // confirms status for master feedback
        cases_level_2_slave = 1; // confirms status for master feedback
        cases_level_3_slave = 1; // confirms status for master feedback
        ////////////////////////////////////////////////
        //
        //  Level 1 - case 4      Continuity Barrel 1 & 2 OK
        //
        //  *Continuity Barrel 1  == HIGH
        //  *Continuity Barrel 2  == HIGH
        //
```

```
        ////////////////////////////////////////////////
        digitalWrite(OU_Error_Check_Connection_B1,HIGH); // activates LED 1 for Barrel 1
        digitalWrite(OU_Error_Check_Connection_B2,HIGH); // activates LED 2 for Barrel 2

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 1 //////////////////
          switch(cases_level_2){
            case 1:
            cases_level_2_slave = 1; // confirms status for master feedback
            cases_level_3_slave = 1; // confirms status for master feedback
            ////////////////////////////////////////////////
            //
            //  Level 1 - case 4     Continuity Barrel 1 & 2 OK
            //  Level 2 - case 1     Barrel 1 & 2 not armed
            //
            //  *Arm Barrel 1        == LOW
            //  *Arm Barrel 2        == LOW
            //  *Capactor Charge     == LOW
            //  *Safetly Discharge   == HIGH
            //
            ////////////////////////////////////////////////
            digitalWrite(OU_Arm_B1,LOW);
            digitalWrite(OU_Arm_B2,LOW);
            digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
            digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
            break;

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 2 //////////////////
            case 2:
            cases_level_2_slave = 2; // confirms status for master feedback
            ////////////////////////////////////////////////
            //
            //  Level 1 - case 4     Continuity Barrel 1 & 2 OK
            //  Level 2 - case 2     Barrel 1 armed
            //
            //  *Arm Barrel 1        == HIGH
            //  *Arm Barrel 2        == LOW
            //  *Capactor Charge     == HIGH
            //  *Safetly Discharge   == LOW
            //
            ////////////////////////////////////////////////
            digitalWrite(OU_Arm_B1,HIGH);
            digitalWrite(OU_Arm_B2,LOW);
            digitalWrite(OU_Cap_Charge_Switch,HIGH); // enables capacity charging circuit
            digitalWrite(OU_Safety_Discharge,LOW); // disables capacity discharge circuit

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 2 //////////////////
```

```
////////////////// LEVEL 3 - Case 1 //////////////////
                switch(cases_level_3){
                  case 1:
                  cases_level_3_slave = 1; // confirms status for master feedback
                  ////////////////////////////////////////////////
                  //
                  //  Level 1 - case 4      Continuity Barrel 1 & 2 OK
                  //  Level 2 - case 2      Barrel 1 armed
                  //  Level 3 - case 1      no fire
                  //
                  //  *Fire               == LOW
                  //
                  ////////////////////////////////////////////////
                  digitalWrite(OU_Fire,LOW);
                  break;

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 2 //////////////////
////////////////// LEVEL 3 - Case 2 //////////////////
                  case 2:
                  cases_level_3_slave = 2; // confirms status for master feedback
                  ////////////////////////////////////////////////
                  //
                  //  Level 1 - case 4      Continuity Barrel 1 & 2 OK
                  //  Level 2 - case 2      Barrel 1 armed
                  //  Level 3 - case 2      fire
                  //
                  //  *Fire               == HIGH
                  //  *Fire               == LOW (1s delay)
                  //
                  ////////////////////////////////////////////////
                  digitalWrite(OU_Fire,HIGH);
                  delay(1000);
                  Serial.println("Fire Barrel 1");
                  digitalWrite(OU_Fire,LOW);
                  break;
                 }
               break;

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
               case 3:
               cases_level_2_slave = 3; // confirms status for master feedback
               cases_level_3_slave = 1; // confirms status for master feedback
              ////////////////////////////////////////////////
              //
              //  Level 1 - case 4      Continuity Barrel 1 & 2 OK
              //  Level 2 - case 3      Barrel 2 armed
              //
```

```
            //  *Arm Barrel 1        == LOW
            //  *Arm Barrel 2        == HIGH
            //  *Capactor Charge     == HIGH
            //  *Safetly Discharge   == LOW
            //
            ////////////////////////////////////////////////
             digitalWrite(OU_Arm_B1,LOW);
             digitalWrite(OU_Arm_B2,HIGH);
             digitalWrite(OU_Cap_Charge_Switch,HIGH); // enables capacity charging circuit
             digitalWrite(OU_Safety_Discharge,LOW); // disables capacity discharge circuit

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
////////////////// LEVEL 3 - Case 1 //////////////////
                switch(cases_level_3){
                 case 1:
                 cases_level_3_slave = 1; // confirms status for master feedback
                 ////////////////////////////////////////////////
                 //
                 //  Level 1 - case 4     Continuity Barrel 1 & 2 OK
                 //  Level 2 - case 3     Barrel 2 armed
                 //  Level 3 - case 1     no fire
                 //
                 //  *Fire               == LOW
                 //
                 ////////////////////////////////////////////////
                  digitalWrite(OU_Fire,LOW);
                  break;

////////////////// LEVEL 1 - Case 4 //////////////////
////////////////// LEVEL 2 - Case 3 //////////////////
////////////////// LEVEL 3 - Case 2 //////////////////
                  case 2:
                  cases_level_3_slave = 2; // confirms status for master feedback
                 ////////////////////////////////////////////////
                 //
                 //  Level 1 - case 4     Continuity Barrel 1 & 2 OK
                 //  Level 2 - case 3     Barrel 2 armed
                 //  Level 3 - case 2     fire
                 //
                 //  *Fire               == HIGH
                 //  *Fire               == LOW (1s delay)
                 //
                 ////////////////////////////////////////////////
                  digitalWrite(OU_Fire,HIGH);
                  delay(1000);
                  digitalWrite(OU_Fire,LOW);
                  Serial.println("Fire Barrel 2");
                  break;
```

```
                }
              break;
            }
          break;
        }

// ---------------- Reset Conditions ----------------
//cases_level_1 = 0; // reset Level 1
//cases_level_2 = 0; // reset Level 2
//cases_level_3 = 0; // reset Level 3

delay(10);


} // closing "if (Voltage_System_Present >= 1.0)"

else // initial conditions "Voltage_System_Present" and "System Enable" not fullfilled
{
  digitalWrite(OU_Error_Check_Connection_B1,LOW); // de-activates LED 1 for Barrel 1
  digitalWrite(OU_Error_Check_Connection_B2,LOW); // de-activates LED 2 for Barrel 2
  digitalWrite(OU_Arm_B1,LOW); // set output pin Arm_B1 to LOW
  digitalWrite(OU_Arm_B2,LOW); // set output pin Arm_B2 to LOW
  digitalWrite(OU_Cap_Charge_Switch,LOW); // disables capacity charging circuit
  digitalWrite(OU_Safety_Discharge,HIGH); // enables capacity discharge circuit
  digitalWrite(OU_Fire,LOW);
 // digitalWrite(OU_Visual_Feedback_Weapon_Armed,LOW);


  // sets all level back to idle state in case System Not Present
  cases_level_1 = 1;
  cases_level_2 = 1;
  cases_level_3 = 1;

  delay(100);
}

} // closing "else"
} // closing "void loop();"

//////////////////////////////////////////////////
//
// I2C communication
//
//////////////////////////////////////////////////

// ---- send out data on request
void requestEvent()
{
```

```
  Wire.write(cases_level_1_slave);
  Wire.write(cases_level_2_slave);
  Wire.write(cases_level_3_slave);
  Wire.write(Cap_Charger_Level_Feedback);
}


// ---- receive date from maser
void receiveEvent()
{
  // condition required to activate I2C communication
  if (PWM_I2C == 2)
  {
    Arm_B_Master = Wire.read();
    /*
     * LEVEL 2 - Master Input
     * case # 1 - not armed (Arm_B_Master == 1)
     * case # 2 - barrel 1 is armed (Arm_B_Master == 2)
     * case # 3 - barrel 2 is armed (Arm_B_Master == 3)
     */

    Fire_Master = Wire.read();
    /* LEVEL 3 - Master Input
     *  case # 1 - not firing (Fire_Master == 1)
     *  case # 2 - firing (Fire_Master == 2)
     */
  }
}
```