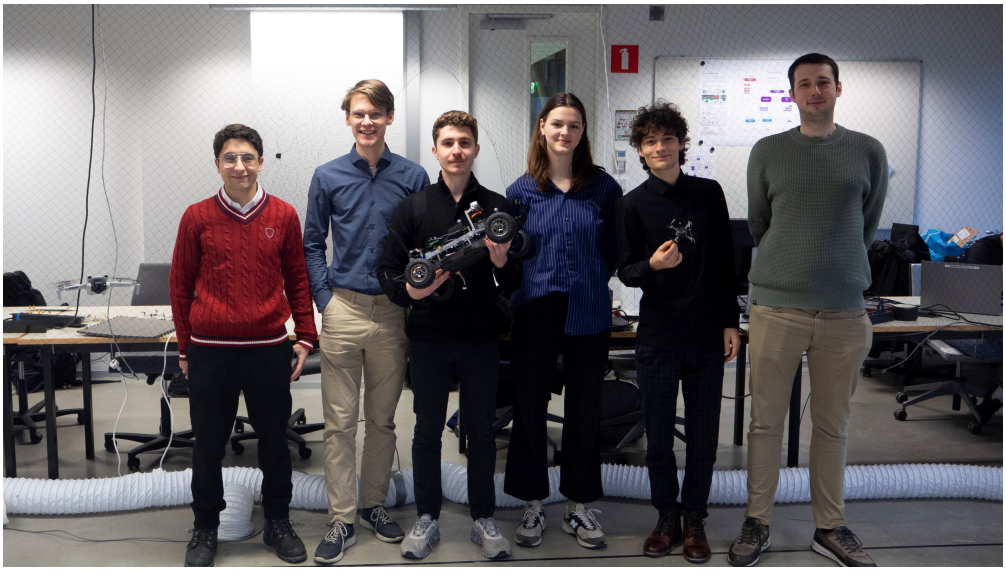
High Tech Systems Honors

MTMS Honors Report

31-09-2024 to 25-05-2025

Full Name	Student ID	Study
Nora Baljé	1839888	Computer Science
Milosz Janewski	1962736	Computer Science
Mathijs Smulders	1973169	Computer Science
Ismail Hassaballa	1569457	Electrical Engineering
Daniel Tyukov	1819283	Electrical Engineering

Coached by Alex Ceano



Eindhoven, June 24, 2025

Contents

1 Introduction

1.1 Problem statement

Rescue operators have to risk their lives every day by going into dangerous and unknown terrain. This uncertainty of not knowing what lies ahead comes with a lot of anxiety, leading to stagnation of important decisions. In a field such as rescue operations, time is a crucial factor, so any delay in decision-making could be detrimental. Decreasing the uncertainty that rescue operators face would, in response, increase the speed of decision making, leading to more successful rescues.

We wanted to create a product that could help rescue workers and first aid responders. With the main focus on decreasing **uncertainty**, and secondary goals of increasing aid speed, increasing terrain awareness, and decreasing needed supplies. We decided on building a Multi-Terrain Mobility System (MTMS), a system that can traverse any terrain, or more specifically: land, air and water. It would be the first tool sent into an unknown terrain and can be used to scout people in need. Once a person in need has been found, we can give the exact location, terrain, and route to a rescue operator. This way they have a clear vision of what they need to do, where to go, and what tools to bring to complete their mission.

This led us to our long-term technical question: "How do we effectively combine ground, water, and aerial movement in one vehicle?". This question will lead to a long-term project, so in this first report we focused on the question: "How do we effectively combine ground and aerial movement into one vehicle?". We set our end goal for this year to build a functional prototype of such a system that achieves both ground and aerial movement.

1.2 Literary Research

One of the things that initially sparked the idea for this project was the Caltech M4 robot [3].

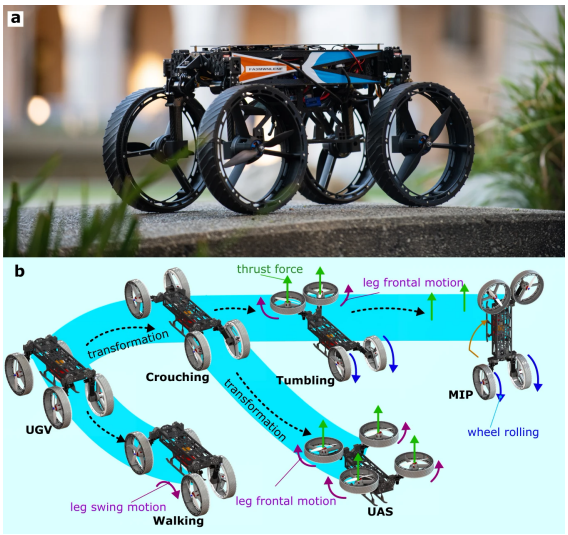


Figure 1:
a M4 in wheeled mode
b M4 transformations to other modes [3]

The M4, which stands for Multi-Modal Mobility Morphobot, can combine its 4 wheels and propellors to transform between 6 different modes, depicted in figure 1. The system was inspired by the movement of animals. For example, birds can walk up steeper slopes by generating an upward thrust with their wings, saving energy compared to flying. The body is made of carbon fiber, which was partially 3D printed. Each leg is equipped with two servos - located at the "shoulder" -, one motor for the wheel, and one motor for the propellor. The propellors are embedded into the wheels, which in turn are actuated via a gear mesh on the rim of the wheel.

Once our problem statement was established we spend some time exploring different solutions to multi-terrain robotic movement. We noticed that the main-stream solutions to this problem are dividable in two categories.

The first type of solution we came across were projects that "statically" combined ground and aerial movement. Some did it by simply adding wheels on an off-the-shelf drone. [6], [9]. Others build a product with the propellors inside the wheels of an RC-like car, where no morphing is required switching modes [17].

The second type of solution we came across were projects that enable morphing to swap between ground and aerial movement. Morphing robots can take many forms, and a lot of research has already been done:

- **Morphing drones in the air** [13], [4]: creating aerial robots that can change shape while in flight, allowing them to fit in small spaces without landing first.
- **Soft body morphing** [2]: materials that can change shape, hold that shape and return to the original configuration, allowing for morphing without the use of motors.
- **Soft morphing joints** [14]: reconfiguring robots through shape morphing joints, allowing for multiple trajectories without changing the mechanical design.
- **Morphing wheels** [19]: creating a transformable wheel, allowing for terrain adaptability while keeping a compact structure.

Looking at these existing solutions we saw that many of these very advanced morphing techniques were still in early stages of research and development. This intrigued us, as morphing robots are much more promising in the future then 'static' robots. We decided that we wanted to contribute to the research on morphing robots, and thus looked into methods of morphing we could use.

We found inspiration for our own project in the FSTAR2 [5]
This is a second iteration of sprawling robots called STAR [18]. Like other robots in the STAR family, the FSTAR2 can extend its arms. In this case, they hinge on the body and move underneath the body when 'retracted'. This is considered its driving mode, while with its arms fully extended it is in flying mode. FSTART2 uses a single motor to control each pair of propellor + wheel by using gear boxes, and a single servo is responsible for extending and retracing the arms. When in driving mode, the mechanism around the servo effectively locks the arms in place, so the servo does not have to continue to provide power to keep shape.

Having seen the M4 and FSTAR2, and knowing that more advanced morphing techniques could be very risky to execute, we decided to stay close to the M4 and FSTAR robot when designing our MTMS.

2 Process

2.1 Team coordination and Meetings

At the start of the academic year, we planned weekly team meetings, where we started by discussing progress and any upcoming weekly plans. As a lot of the initial work required team consensus, we often continued with discussing ideas for the project. The rest of the meeting was then spend on working these ideas out together.

When we got more concrete ideas about what was required to complete our project, we started with some internal workshops. Here, one person would focus on achieving a required skill, and then they would host a workshop to transfer all the relevant knowledge to the rest of the team. More details are discussed in Section 2.2.

Later in the year, when the components had arrived, we realized that weekly team meetings no longer worked. Our schedules did not align and everyone was working on different tasks, making it easier to work separately than together. This is why we switched to work sessions, here a few people would together on a specific topic. We would still update each other on our progress through Whatsapp, coach meetings, or when we worked on a topic together.

Throughout the whole year, we tried to organize a coach meeting once every two weeks. Here we updated our coach on the state of the project and got valuable feedback.
For each of these meetings, we randomly assigned the following roles to a team member:

- **Chairperson**: In addition to being responsible for booking a date, time and room for the meeting, this person would also make an agenda to make sure the most important things were discussed in the meeting.
- **Scribe**: They would be responsible for taking the minutes of the meeting.
- **Timekeeper**: This person would make sure we did not go over time on agenda points, so that we would have time to discuss everything on the agenda. In practice, this would mostly be redirecting the team from irrelevant tangents back to the important topics on the agenda.

The random distribution of these roles meant that everyone was able to experience all of these roles and that the workload was divided equally.

Throughout the year, we realized that we were too last-minute with planning meetings. Sometimes, this led us to have to postpone our meeting by a week. That is why we altered the responsibility of the chairperson, they would set the date of the next meeting at the end of every meeting. After some feedback from our coach, we started using placeholders to block general time slots up front. We would specify the exact time and distribution of meetings later, but this ensured that everybody would stay available.

2.2 Workshops

As part of self-development, knowledge transfer, and gaining the required skills, we started adding workshops to our meetings. During these workshop meetings, one of the team members would prepare a small workshop to educate the team on a specific topic. This allowed us to achieve the skills required to turn the MTMS system into reality. In total we had 6 workshops, an overview of these workshops can be found in Table 1.

Topic	Description	Responsible
ROS2	Basics of nodes & topics	Milosz
ROS2	Basics of services	Milosz
ROS2	Basics of actions	Milosz
Micro-controller	Basics of MCU and Protocols	Ismail & Daniel
Fusion360	Basics of CAD design & 3D printing	Ismail
Git	Basics of Version Control	Daniel

Table 1: Workshops

These workshops would start with explaining the necessary background information on the topic, after which we would have an interactive exercise to practice with the topic. We used slides to give this structure. Slides of the first of ROS2 workshop are attached as an example in Appendix D. Some workshops would include homework, such as a programming exercise or designing something using Fusion360 software to be printed.

This workshop approach sped up our process a lot. Instead of everyone having to individually research each topic, one person did the research and transferred only the necessary information.

2.3 Planning

During the year, we used Gantt charts to plan our progress. Our first version can be seen in Figure 2.

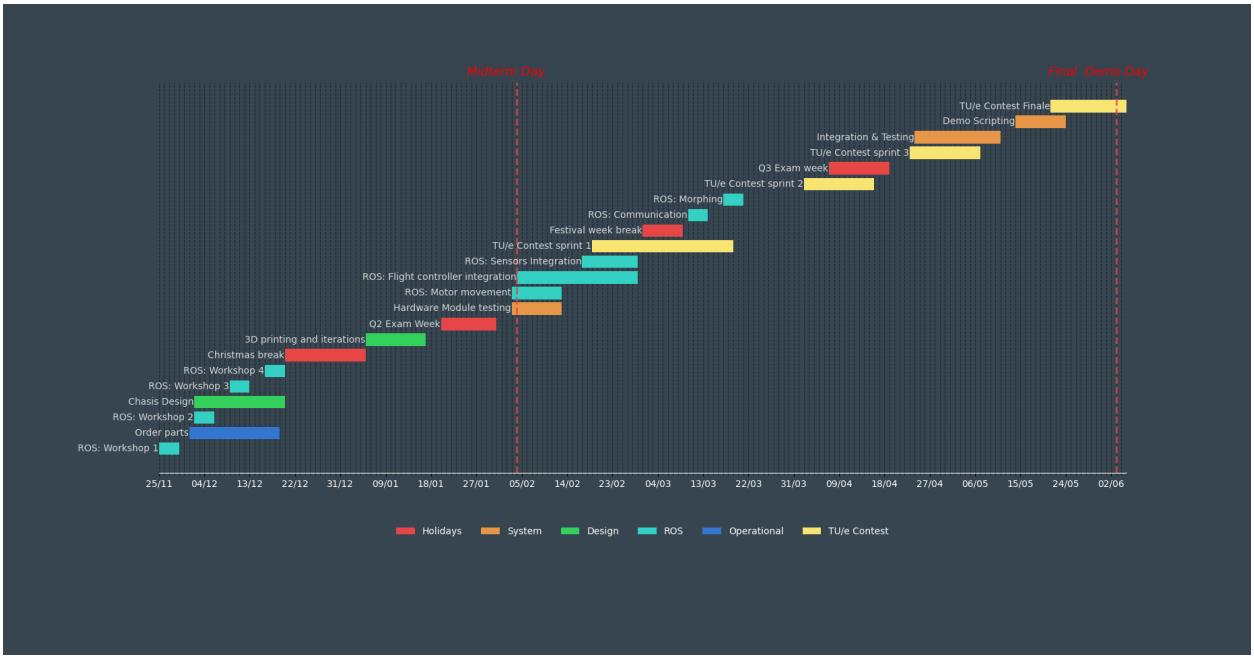


Figure 2: Gantt planning Version 1

Along the way we had a few setbacks, but the most notable was the delay on the budget, and with that the ordering of parts. We had made a draft budget for the project proposal, which we hoped to refine quickly. We wanted to get the components as fast as possible, so we could start working on them. However, this

refining of the budget, would turn out to take a lot of time. More details about this are discussed in section 2.4

After finalizing the budget, numerous delays occurred in the process of ordering parts. Due to these delays in the delivery of the hardware components, we revised our planning to the one seen in Figure 3. We focused on tasks unrelated to the specific hardware components, such as TU/e contest, research, MicroROS and 3D design.

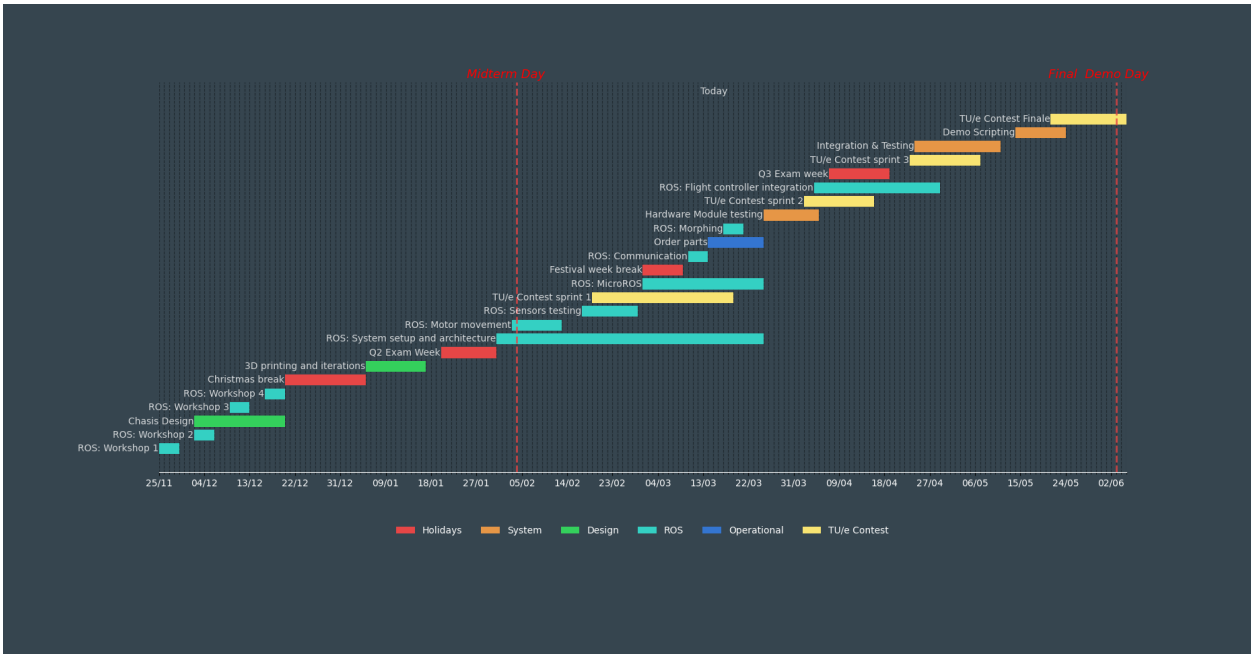


Figure 3: Gantt planning Version 2

Once our components arrived, we used a Github project to manage, assign, and document our tasks. Due to our extensive preparation while waiting for components, this stage was not meticulously planned. Instead, we prioritized the tasks through a list in Github, keeping in mind blocking tasks and the minimum viable product we envisioned for the final demo. We were all assigned to one or more of these tasks, and when a task had been finished we would move to another high-priority task of the list.

2.4 Budget

As mentioned in section 2.3, we made a first version of the budget for our project proposal. A summary of this budget can be seen in Table 2. The full version can be found in Appendix C.1.

Group	Cost
Computing	1070
Electronics	629
Actuators	310
Flight	620
Chassis	90
Labor	11250
Total	**13970**

Table 2: First version of our budget

After our project idea was approved by our track coordinator, we refined this budget.

To do this, we added a more detailed description, calculated shipping costs, and estimated delivery times. We also spend some time investigating the weight of each component to make sure that the propellors would

be able to carry the vehicle. We noticed that this was initially not the case and adjusted the components accordingly. Once we were confident in our budget, we asked for the help of some mechanical engineers to give us some feedback on our component list.

In the process of refining the budget, we managed to cut costs by a thousand euros. A summary of the refined version can be found in Table 3. The full version can be found in Appendix C.2.

Group	**Cost**
Computing	300
Electronics	437
Actuators	367
Flight	525
Chassis	30
Labor	11250
Total	**12909**

Table 3: Second version of our budget

This money save is majorly due to downgrading our main computing module for a cheaper option. We originally wanted to use the UP Xtreme i14, which is a very powerful computer. For long-term development, this might be the better option, but we decided to cut costs and focus on components that fit our 1 year needs. This, as you are never sure what issues you run into during development, starting on a raspberry pi allows us to be more adaptable to unexpected changes and costs.

3 Results

3.1 Prototype

The structure is mainly based on general drone designs, as they have more strict requirements compared to ground vehicles. In the end, we developed a concept featuring a main body with four connected legs, which are equipped with wheels and propellers.
Figure 4 shows the an overview of the CAD model of our design.

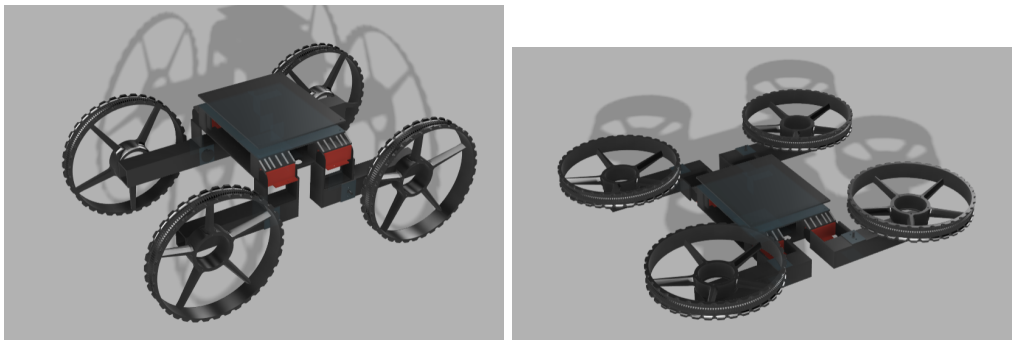


Figure 4: Prototype CAD model overview

The extent to which this prototype aligns with our goals can be seen in Table 4, showing a comparison between our goals and our results.

Functionality	**Goal**	**Result**
Driving	Operate different movement commands from a keyboard, which enables it to move forward, backward and turn left or right.	Thanks to the ROS architecture (§ 3.7) and overall code, we are able to control separate wheels, which allows us to drive forward, backward and turn to different directions by moving some wheels forward and some backwards.
Flight	Operate different movement commands from controller, which enables it to move in any direction and allows to fly at least 5 meters.	Thanks to the flight control and other drone components (§ 3.6), we were able to essentially imitate a drone with our robot and fulfill the flight requirements.
Morphing	Be able to change from drive mode to fly mode and vice versa.	Thanks to the servo motors and mechanical structure (§ 3.5), we are able to precisely move legs for different modes in a safe manner.

Table 4: Comparison goals and results

3.2 Hardware

This section will give a more in depth description of most important hardware component found on our budget as discussed in Section 2.4 and Appendix C.

3.2.1 Main Computer

The main computer is responsible for running the ROS2 stack and coordinating any operations between subsystems. In the selection process, we considered two main options: the **UP Squared** board and the **Raspberry Pi 5**.

The UP Squared board was attractive due to its dedicated Neural Processing Units (NPUs) and higher computing performance, which would make it a strong candidate for future integration of autonomous

features such as onboard AI-driven navigation or computer vision. However, the board came at a significantly higher cost and was not necessary for the goals set for this year's prototype.

Given our focus on establishing a reliable ROS2-based system and our budget constraints, we chose the Raspberry Pi 5. It offers sufficient performance for running the ROS2 environment and handling basic data processing tasks. Its availability, cost-effectiveness, and community support made it a practical choice for this phase of the project.

3.2.2 Morphing Actuators

For the morphing mechanism, we used **servo motors** to rotate the arms between driving and flying. Servo motors do not move continuously for a longer period of time, like DC motors. Instead, they can rotate at a specific angle - which is limited to 180° - with high precision.

We selected the specific servo based on the required torque, which we calculated using the expected thrust of the motors and the length of the arms. In addition to the servo, we sourced a pair of **linear actuators** as a fallback option, in case the torque requirements later exceeded what the servos could reliably handle under load. However, the servo-based design remained the preferred solution due to its simplicity.

3.2.3 Ground movement Actuators

For ground movement, we selected **DC motors** equipped with encoders and integrated gearboxes. These motors are responsible for driving the wheels of the robot and were chosen to provide sufficient torque to move the full system during driving mode.

The inclusion of encoders allows for future implementation of speed control and position feedback if needed. The integrated gearboxes reduce the output speed while increasing torque, which is essential given the expected weight of the robot and the terrain it may need to traverse.

3.2.4 Flight Computer

The flight stack is based on **Pixhawk 6C** flight controller bundled with an **M10 GNSS** GPS and the **PM07** power-distribution module. We chose this set for the following reasons:

1. **Hardware redundancy:** The 6C employs dual STM32H7/F4 MCUs, allowing the IO processor to maintain control should the main FMU reboot. This is essential for a heavy complex system.
2. **Open ecosystem:** Pixhawk is natively supported by ArduPilot and PX4 both expose MAVLink.
3. **Total Mass Considerations:** The whole flight controller package would weigh around 34.6 g which is important for a weight efficient system.

3.2.5 Flight Actuators

Actuation for the quad-X configuration is provided by:

- **Motors** - four EMAX Pro 2812 motors rated at 1100 RPM per volt. The 1100 kV point gives a good compromise: high enough Kv for 6S operation without exceeding the ESC's 35 A continuous rating, yet low enough to swing our 8 inch tri-blade props efficiently. Each motor weighs 71.6 g, total 286 g.
- **Propellers** - Master Airscrew 3MR 8×4.1 three-blade. Static bench pulls show 8.7 networns at 16 A per motor, which equates to a $1.2\times$ thrust-to-weight ratio for the current 3 kg system.
- **ESCs** – EMAX *Bullet 35 A* BLHeli-S speed-controllers. They tolerate 6S and weigh only 25.2 g each.

We postponed the motor-prop selection until the rest of the structure (legs, batteries, avionics) have arrived. With the final mass estimate in hand (3 kg) we targeted a hover current below 60 % throttle to leave headroom for gusts.

3.3 On board Sensors

The Pixhawk IMU set (ICM-20689 + BMI055) supplies the primary attitude solution with triple-redundant gyros and accelerometers mounted on vibration-isolation foam inside the flight controller.

The Holybro M10 GNSS module delivers GPS positioning, integrates a compass and safety-switch, and provides the vehicle's absolute position and heading reference.

We included a LiDAR sensor to enable basic mapping and environment awareness. The LiDAR is intended to support future autonomy features by providing distance measurements and obstacle detection. While not yet fully integrated into the navigation system, it was chosen to ensure the robot could eventually perform SLAM (Simultaneous Localization and Mapping) or basic terrain scanning.

3.4 Power Distribution

Our system is powered by a 6-cell LiPo battery (22.2V nominal, 25.2V fully charged), which supplies all components through a custom-designed modular power distribution setup. A splitter cable converts the EC5 connector on the battery into two XT60 outputs: one dedicated to the flight controller stack and the other feeding an off-the-shelf power distribution PCB (Fig. 5) with multiple XT30 outputs. Usage of these connectors has been summarized in Table 6.

Each major component or group of components is powered by a dedicated buck converter, selected based on voltage and current requirements, specified in Table 5. All converters are connected to the PCB via XT30 plugs to support safe, modular, and swappable connections.

Component(s)	**Voltage**	**Max Current**	**Step down Buck Usage**
Raspberry Pi 5	5.1V	5A	One dedicated buck
Each Servo (4 total)	8.4V	3.4A per servo	Four bucks (One buck per servo)
All 4 DC Motors	12V	0.3A per motor (1.2A total)	One shared buck
Flight Stack (Pixhawk + ESCs)	22.2V (direct)	Varies	No converter needed (powered directly from battery)

Table 5: Power requirements and buck converter usage

Connection Type	**Used For**
EC5	Main battery output
XT60	Battery splitter output, input to power distribution PCB, flight controller power
XT30	Output from power distribution PCB to individual buck converters
Custom Splitter Cable	Converts EC5 to dual XT60

Table 6: Connector types and their usage in the system

The power distribution PCB, seen in Figure 5, accepts XT60 input (1) and offers multiple XT30 outputs (2), enabling individual power lines for each buck converter (3). This approach ensures easy maintenance and scalability, and avoids overloading any single converter.

All wiring was hand-soldered using wire gauges appropriate to each load (ranging from 10 AWG to 14 AWG), ensuring safe operation even under stall conditions. The modular design allows components and converters

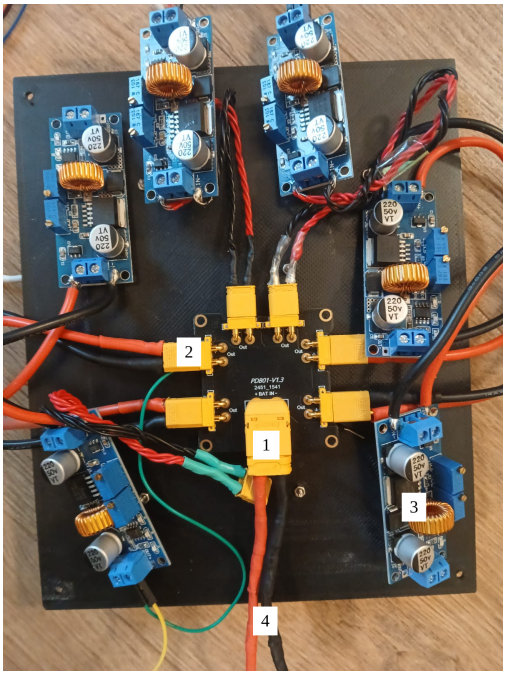


Figure 5: Power Distribution Board
1: XT60, 2: XT30, 3: Buck Converter, 4: Power cable

to be swapped quickly during testing or upgrades.

3.5 Mechanical

Our robot contains a lot of specific hardware components, which require highly customizable design. Because of this, we decided to 3D print the structure using PLA plastic. PLA stands for Polylactic acid, and is widely used for 3D printing because of its low melting point, relatively high strength, low thermal expansion, and good layer adhesion [15]. Stronger 3D printing materials exist, such as carbon-fiber re-enforced filaments. These materials are more expensive and difficult to use, however, and since we are still in the prototyping phase of this project we did not use these stronger filaments. We designed all the parts in the Fusion360 software.

While refining the design and mechanics, we encountered different problems and options for actions. We will elaborate on these in sections 3.5.1 through 3.5.6, going part by part.

3.5.1 Main body design

Starting with the core of our project, we had two options for the structure.

1. One big enclosed box
2. Multiple layered plates

An enclosed box would make the process of 3D designing a lot simpler and there would be no need to know the exact components, only their rough sizes. However, it overlooks that some components require special conditions, such as radio receiver requires not being blocked or micro-controllers require that there is no electronic noise. Moreover, it would be impossible to do quick repairs, since we would have to take out all of the components to exactly identify what is wrong. With that in mind, an enclosed box is a bad idea in the long run due to the risk of damaging components.

Multiple layered plates fix those problems by having special spots for components, allowing for safe storage

and easy maintenance of components. The only downside of such solution is that its significantly harder to create such a design, we would need to know all of the components in advance, the wires might be too short, and modifying such a design may be tricky.

We have decided to go with the second option, due to accessibility and our concern for safety of the components, which are costly compared to the 3D printed chassis.

This multi-layered approach does introduce the balancing act of making sure that every component can fit but also not make the plates too big, as that could put too much stress on them. Our goal was to make enough space for all of the components, while making it as small as possible. In the end, the sum of all components, that are put on the plates, is around 64 000 mm^2. To accommodate that, we will use 3 plates with size 164 x 200mm^2. Additional reasoning for the exact chosen size and number can be found in Section 3.5.5 discussing the leg design.

3.5.2 Component placement

For this section, the three major constraints are horizontal weight distribution, specific components, and distance between holes. Important terms are center of gravity, meaning the midpoint of weights combined, and center of thrust, meaning the meeting point of the propellors.

Starting with specific components, we have to make sure that all of the little requirements of our hardware are met. Those include:

- **Battery:** If the battery is placed too low or too high from the center of thrust, our drone would become unstable and oscillate rapidly. For that reason, battery compartment is put exactly under the plate with servo motors.
- **Microcontrollers:** Servo motors introduce electrical noise due to their design. This may create issues with our microcontrollers, which is why they are placed in separate sections.
- **Drone components:** If the radio receiver is blocked by some material, the connection might deteriorate. Thus, drone parts are put on the top plate.

We have constructed the following distribution in Figure 6. In the lowest part, we will have the battery compartment. It will be connected to the bottom plate, which will contain the power distribution components and servo motors for legs. Then, we will have the middle plate, which will have both microcontrollers - Raspberry pi and Arduino. Finally, at the top plate, we will have all the components required for drone connection.

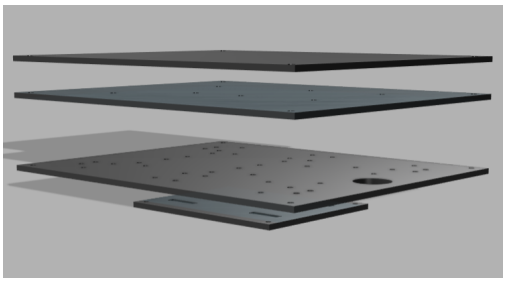


Figure 6: Central plates CAD model

The next constraint, horizontal weight distribution, in simpler terms means that we have to make sure that the center of gravity is at the same horizontal point as the center of thrust. It is necessary for correct movement of the drone, as otherwise our robot would be skewed to one side which makes it impossible to fly. To this end, we will categorize hardware into heavy parts and light parts. Additionally, some light components

will be grouped and regarded as one heavy part.
To solve this issue, all heavy parts will be put symmetrically to the center so that the weight vector is in the middle of plates / center of thrust. For the smaller parts, we will try to place them as non-impactful as possible. It is important to notice that we can do as such, thanks to the drone module software that can mitigate small weight errors. Additionally, we will test the drone mode to see if the error margin is small enough.

Based on the previous description, the horizontal positions of heavy components are arranged as shown in Figure 7.

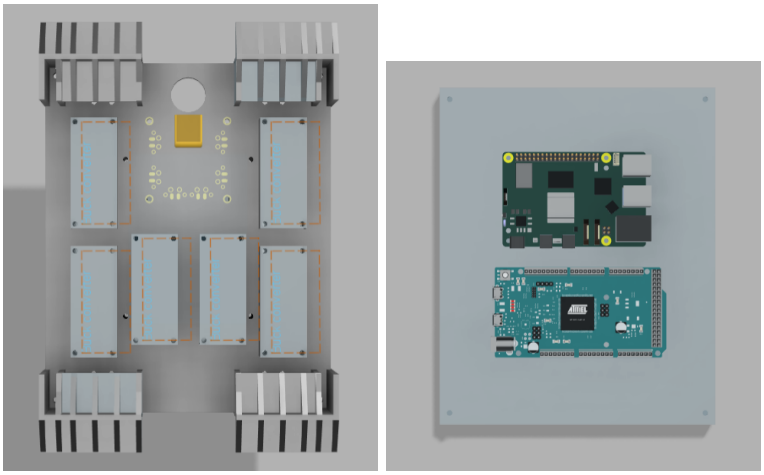


Figure 7: Bottom, Middle - Plate overview

The plate for the battery is put symmetrically in the middle as it does not interfere with any other component. On the bottom plate, we have symmetrically screwed 4 servo motors and 4 bucks. With this, we have been able to achieve the symmetry of legs and wheels, which were the main contributors to the weight of our design. The rest of the components in this section - 2 bucks and the power distribution board - were balanced to have minimal impact. On the middle and top plate, we balanced Raspberry 5, Arduino, and drone components to also have minimal impact.

Lastly, due to the integrity of the plastic plates and holes for the components, we have ruled that all holes should be at least 3 mm apart. We concluded this from testing 3D prints, where there was a significant increase in the stability of the structure when the 3 mm rule was respected. In the end, we used this constraint as a good rule of thumb when placing components, however, there was never any issue, where we had to change major placements due to this rule.

3.5.3 Wheel

The wheel needs to satisfy three roles: **Ground locomotion** with low rolling resistance. **Housing of an 8-inch propeller and its EMAX brushless** motor for flight mode. **Mechanical isolation** of the propeller shaft from wheel rotation so that the motor can remain stationary during driving.

Structural concept

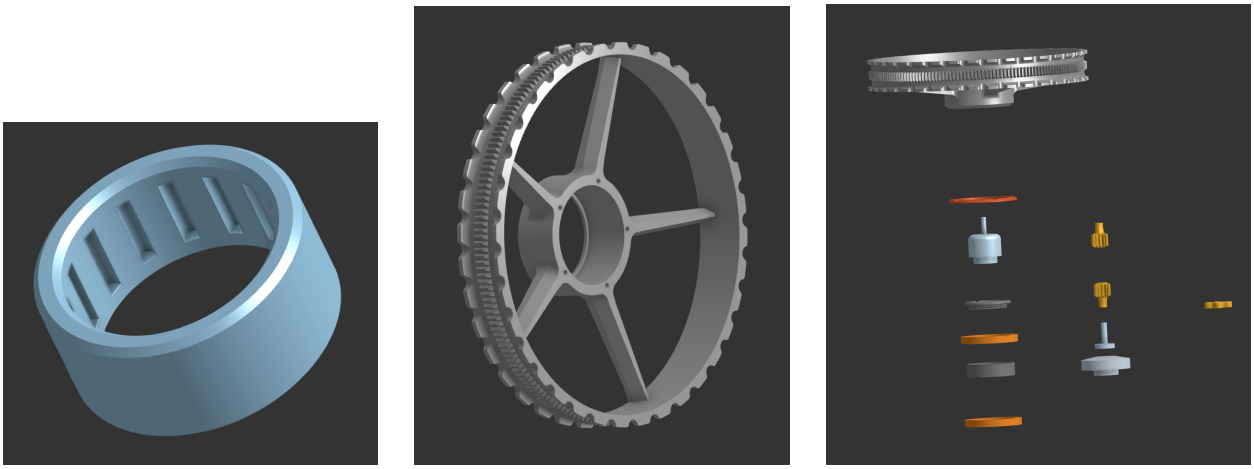
- The rim is offset from the hub (Figure 8b) so the propeller disk clears the ground when the leg is in drive configuration but sits on the line once the wheel is rotated to the flight position.
- Five radial spokes tie the hub to the rim while leaving cut-outs for airflow.
- A tooth profile is printed directly into the outer circumference. A geared bulge on the leg engages these teeth, allowing a compact DC motor to spin the wheel from the outside.

Custom lightweight bearing Commercial bearings at the required ∼58 mm outer diameter would add several hundred grams. Instead, an in-house **rod-bearing cartridge** (Figure 8a) was designed:

- Sixteen 17 mm steel rods act as needle rollers.
- The rods are retained by a 3D-printed cage and grease.

This solution keeps the bearing mass at low weight while still supporting radial force.

Wheel actuation During driving the wheel is powered through the rim gear. For flight the leg servo rotates the entire wheel-propeller assembly by 90° (Figure 8c), aligning the motor shaft vertically and locking the rim so the brushless motor can spin up without spinning the tyre.



(a) Needle-bearing insert (b) Rim and hub overview (c) Exploded drivetrain

Figure 8: Wheel components and integration sequence.

3.5.4 Morphing

The goal of the morphing is to transform the wheel from vertical position to horizontal position and vice versa. As a team, we had several different ideas, however the most simple and reliable one was to have a servo motor connected to a leg which is connected to the wheel and propellers, and then move the servo 90°. Figure 4 presents the movement of the servo motors. This will be done in small steps, to smoothly lower the robot to the ground.

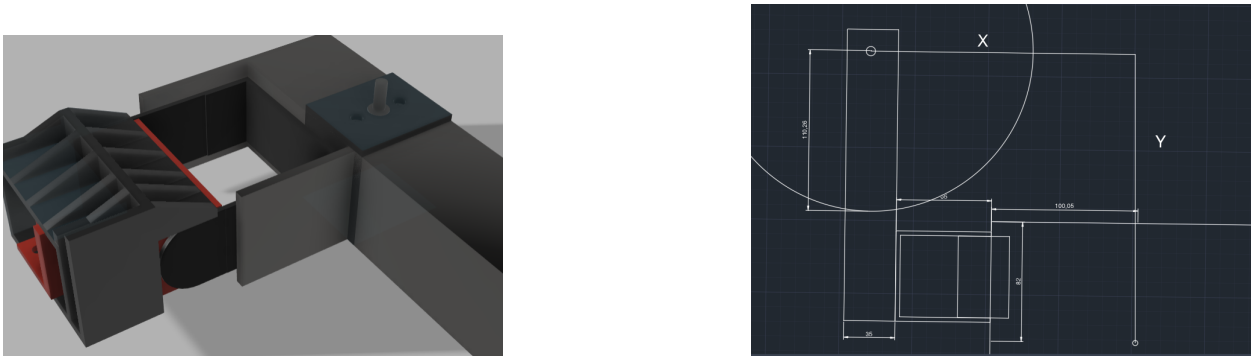
Some issues arise with this solution, such as having to push the wheels to the ground while switching from fly to drive mode, or the force generated by the propellers might induce strain to servo motors. To answer those issues we have created a strong and stable connection between servo-leg-wheel, created a mechanical servo lock design, which will secure the servo from moving more than 90°. Further explanation is provided in Sections 3.5.5 and 3.5.6, discussing the leg and the servo holder.

3.5.5 Leg design

The leg design mostly depends on the wheel design and main body sections.
The leg structure is very simplistic. It is a cuboid with height and width 37 x 37 mm^2 and 3 mm walls. These dimensions are optimized for holding the DC motor and connection to other components. Additionally, it has extra walls around metal bars in servo motors for extra stability, as shown in Figure 9a.

The length of the leg requires more in-depth calculations. The major part in those is to create a “square” from propellers. For that, the x and y distance have to be the same in the Figure 9b. Next, we will analyze our decision making in this matter.
Since the wheel design is fixed, we adapted the leg design based on that. The radius of our wheel is 110 mm, which means that the distance between the propeller and the DC motor has to be around 112 mm for it to spin the wheel. That gives us a lower bound on the length of the leg, however we have to remember that the DC motor cannot go in the way of any screw connections with the servo motor, so the leg has to be

slightly longer. Taking in regard the previous requirement, we can extend the calculations. Since we already have a lower bound on the leg and we do not want to make it any longer since that will impact the structure stability, we optimized the plate sizes to fill the requirements - minimized length, maximized width, while keeping plates stable.



(a) Leg extension

(b) Size calculations

Figure 9: Leg components and integration.

3.5.6 Servo holder

In this section, we will discuss the extra lock design for servo motors. As mentioned before, the thrust of the propeller motors will generate force that will try to move their position from the perfect 90°. Normally, this force would be negated by the respective servo motor keeping the leg in 90°. However, there are multiple factors, where this scenario might fail, such as taking too much energy, frying components or servos failing. To stop it from happening, we implemented physical locks that will forbid the servo from moving more than $0 - 90°$ angle (Figure 10).

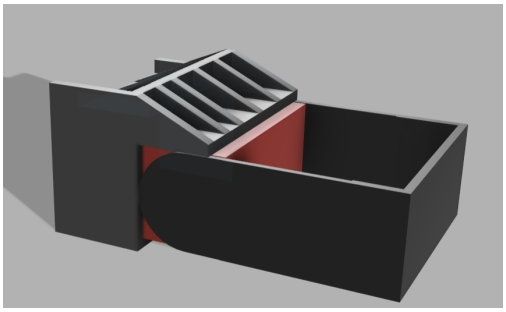


Figure 10: Mechanical lock for servo

3.6 Flight

3.6.1 System overview

For the aerial mode we adopted a conventional quad-X layout driven by the **Pixhawk 6C** flight-controller bundle (aluminium enclosure, PM07 power-module and M10 GNSS). The controller runs a stock ArduPilot build. ArduPilot was chosen for its ready-made multirotor support, extensive logging facilities and industry standard, while still allowing further parameter adjustemnt required for later autonomy experiments.

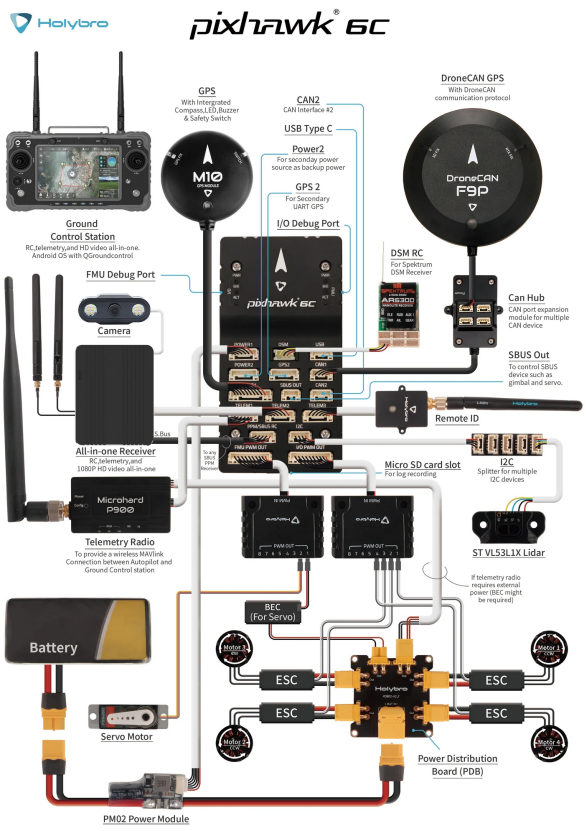


Figure 11: Reference wiring used for the flight stack (adapted from the Holybro Pixhawk 6C datasheet)

3.6.2 Flight hardware

Component	Function in the MTMS prototype
Pixhawk 6C (+PM07 +M10 GNSS)	State estimation, control loops, battery monitoring, arm/disarm logic
4 x EMAX *Bullet 35 A* BL-Heli_S ESC	Translating PWM to three-phase motor drive
4 x EMAX Pro 2812-1100 kV	High-torque brushless motors sized for 8 inch tri-blade props
4 x Master Airscrew 3MR 8 × 4.1	Optimised for static thrust at the expected weight
6S 4000 mA h Li-ion polymer battery pack	Primary energy storage
Microhard P900 telemetry radio	Long-range MAVLink bridge to the ground station
PWM breakout harness	allows for connection between the flight controller FCU and ESCs

Table 7: Key flight components

3.6.3 Electrical integration

Figure 11 served as the reference for our wiring:

- **Power path:** The 6S pack feeds the **PM07**. The board steps down to 5.2 V/2 A for the Pixhawk and exports voltage/current telemetry on the `POWER1` port. Its four leads deliver raw battery voltage directly to the ESCs, while a separate 5 V, 3 A.
- **Signal path:** PWM OUT 1-4 carry 400 Hz PWM to the ESCs via the breakout. `TELEM1` hosts the P900 radio for bidirectional MAVLink, whereas GPS, safety-switch and magnetometer share the `GPS` UART/I^2C port.

3.6.4 Firmware configuration

After flashing, the following ArduPilot parameters were changed from their defaults:

- `FRAME_CLASS = 1` (quad-X) and `MOT_PWM_TYPE = 4` to match the BLHeli_S 48 kHz input.
- `BATT_FS_CRT_VOLT = 19.8` V, corresponding to 3.3 V/cell under load.
- `SERVO_BLH_AUTO = 1` for bidirectional DShot ESC telemetry.

3.6.5 Planned bench and flight testing

Static thrust trials. Arms locked in flight position: 8.7 N per motor at 16 A—about 1.2× the hover requirement, giving a theoretical ∼5 min endurance at a 3 kg take-off mass.

3.7 Software

3.7.1 Frameworks

ROS (Robot Operating System) is a framework often used to build robot software. It provides existing libraries for tasks like sensor integration, movement, perception, and communication. These existing libraries simplify the development and testing of complex robotic systems. Due to these reasons we decided to use ROS for the development of our system.

There are currently 2 active versions of ROS in the market, ROS1 and ROS2. We struggled deciding on which version to use but eventually settled on ROS2. ROS1 is the original version and has been on the market longer, which means it has more supported libraries and more precise documentation. These could be really useful during development, as it would simplify the process. However, when it comes to long term support ROS2 is the only way to go, as ROS1 is slowly being phased out and eventually will no longer be a supported version. Considering the longevity of our project we decided that ROS2 was the more sustainable option, even if it would complicate the development process.

As mentioned in Section 3.2.1, we are using a Raspberry Pi 5 as a main computer, and here we have installed ROS2. We are also using an Arduino Due for easy control of the motors, here we are running micro-ROS, a version of ROS2 designed to run on small micro-controllers. Micro-ROS allows embedded devices to communicate with larger ROS systems, simplifying the integration of sensors and motors. To facilitate this communication we implemented a micro-ROS agent. This is an application that can function as the translator between the main computer and the micro-controller, it allows the micro-controller to function without running the full ROS framework.

In ROS you work with **nodes**, these are processes that perform a specific task. Each node can be run independently, but multiple nodes together form a full system. Nodes can communicate with each other through 3 different communication methods:

- **Topic:** a one-way communication channel, where one node functions as the publisher, and one functions as the subscriber.
- **Service:** a two-way request-response channel, where one node requests a service and the other node responds.
- **Action:** like services, but it allows for feedback and cancellation during execution

3.7.2 Architecture

For our architectural approach, the main concern was the adaptability of the code for future development. As this is the first iteration of this project, and the future developments are unclear, an easily understandable and modifiable code was considered crucial. To achieve this we tried to implement the following code standards:

- **Interfacing:** creating clear ways for different parts of the system to communicate with each other.
- **Separation of concerns:** ensuring that each part of the code has a single, clear job.
- **Abstraction:** hiding complex details behind simpler code.
- **Modularity:** building the system in independent parts such that one part can be adapted without interfering with the rest.

To achieve these code standards we constructed a node for each subset of tasks and introducing a hierarchical structure in the connection between nodes. The hierarchical structure allows us to abstract from the deep level code. Using these principles, we designed the our architecture, which can be seen in Figure 12. In this image the ovals denote the different ROS2 nodes, and the arrows denote the connections between nodes. The command listed on the arrows is used to signal what type of command is send, but this is not the exact command, we will explain these later. In this first year of the project only the selected area has been implemented into ROS2 code for our prototype. The drone can be controlled, but this is with an outside controller, not with ROS code.

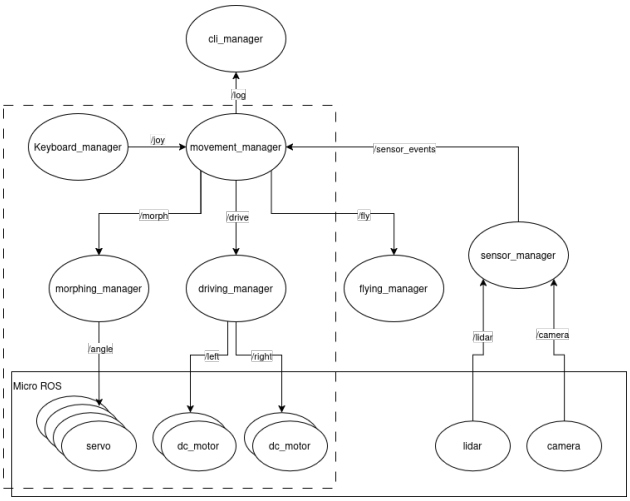


Figure 12: ROS2 architecture diagram

3.7.3 Communication

For communication we mainly utilized topics, this as most our communication is one-sided. But also as topics are more effective for communicating (sensor) data at high frequencies. The only two connections where we did not use a topic were for /morph and /angle. For /morph we opted for an action as morphing is a slow gradual process, where pausing or canceling needs to be possible. For /angle we used a service as we want to enable feedback from the servo motors, this way we can track if we were actually able to morph into the position we desire.

3.7.4 Connections

The /joy, /drive and /fly topics use the Joy message type from the predefined ROS2 sensor_msgs package. This message includes information about the axis measurement and button presses. For buttons we sue WASD for forwards, backwards, or sidewards movement, and we use Q for morphing.

The /left and /right topics use the predefined Int32 message type from the predefined ROS2 std_msgs package. This message is just a singular 32-bit integer that denotes the speed the motors should move in a range of [-255, 255]. Here we use a minus sign to differentiate forward from backwards, and the magnitude denotes the speed, from no movent (0) to maximum speed (255). Message types do not allow for ranges, so this needs to be enforced on publishing side. If implemented on the subscription side, the publisher will not be able to receive errors without using an extra topic.

The /morph action uses a self defined action type MorphAction. An action has to be defined as a request, feedback and response. The request consists of a 32-bit integer that denotes the mode. This mapping from integers to the mode is done by means of an enumeration defined in the movement_manager node. The feedback also consists of a 32-bit integer that denotes the completion percentage. The response consists of a boolean denoting success, and a string to supply a message that can be used for error handling.

Finally, the /angle service uses a self defined service type SetAngle. A service has to be defined as a request and a response. The request consists of a 32-bit integer that denotes the requested servo angle. This angle should be in range [0, 180]. Again, services do not provide range requirements, so this should be enforced on either the server or client side. We do this on the server side. The response is the same as the response in the MorphAction action type.

3.7.5 Nodes

The movement_manager node is our central node. This node is responsible for forwarding all the sensor inputs to the appropriate nodes. It keeps track of in what mode the robot is, so it can use the correct /drive of /fly topic for movement. This is syncronised with the morph_manager through the requests and responses of the /morph action.

The keyboard_manager node is actually two nodes: one for registering the keyboard inputs and one for translating keypresses into the Joy message type. We got the code for these nodes from Github [1].
We register directional keys as WASD. Additionally we register the Q key as the first button and SPACE as the second. Because of the modularity the movement_manager decides what the first and second button do. There also is a configuration file in which we can what button is activated from which key very easily.

The driving_manager node receives an axis measurement and translates this into left and right motor outputs to accommodate differential steering using the following formula:

$$[L, R] = 255 \cdot \begin{bmatrix} \frac{(x+abs(x)\cdot y)+(x+y)}{2} \\ \frac{(x-abs(x)\cdot y)+(x-y)}{2} \end{bmatrix}$$

Where [x, y] denote the input axis measurements in the range [-1, 1] for x as left/right and y as forward/backwards measurements. [11]

The morphing_manager node only receives a request to morph to a specific mode (defined through an enumeration for easy addition of other modes). Depending on what mode the robot is in at that moment it

calls a morphing function. This function sends a sequence of degrees to the servos, such that the robot is lowered to the ground in a smooth motion.

3.7.6 MicroROS

All functionality of microROS has been implemented one node. This means we can use a singular **executor** and do not have to worry about multi-threading (executing tasks concurrently) yet.

An executor in ROS2 (and MicroROS) is the central component responsible for handling timers and incoming data. This comes down to listening for events - like a new service request or a timer triggering - and calling the appropriate callback function to handle each one.
More specifically, we are using the standard **rclc** executor, which is optimized for micro-controllers with resource and time constraints.

Since we use a single-threaded executor on one node, all callbacks are executed one at a time, in the order that they are ready. While it avoids the need of multi-threading and synchronization, we must ensure the executor can keep up with the incoming events. In practice, this means trying to minimize callback overhead, such that the executor 'wastes' as little time as possible on each callback.

We reduced this overhead by only having three callback functions that all control multiple motors as opposed to one callback for each motor separately. This reduces the overhead in the sense that message data extraction and processing have to be done only once per callback.

- Left DC motors: This callback gets executed when new messages come in on the /drive_left topic. We need two values for controlling the DC motors: speed and direction. Incoming messages are in range [-255, 255], and we translate these to the two motors inputs These commands will be sent to two motors.
- Right DC motors: Similar to the left DC motors, but listening to the /drive_right topic and addressing other motors.
- Servos: This callback gets executed when a new request appears on the /servo_angle service. It will write the exact angle request to all the servos, after checking if the requested angle is in the allowed range (as discussed above). Depending on if the angle is allowed or not it sends back a response.

We can control the DC motors with a pwm signal for speed and a high or low signal for direction. Our motors also have an interrupt wire. We initialize this input wire in the code but have not yet implemented handling of the signals this sends.

For controlling the Servo's we make use of the Arduino Servo package. Using this package we can assign a control output pin to a Servo using Servo.attach(pin), and call Servo.write(angle) to move the servo to the specified angle.

4 TU/e contest

4.1 Activities

At the start of the year, the honors academy instructed us to choose a development path for our project. From the moment we decided on our topic, we knew that it would be able to make a positive impact in many different exploration-related fields. However, none of us really knew how to harness and communicate this potential. As we did not only want to use our honors project to learn technical skills but also build the soft skills required to harness the potential of our product, we decided that we wanted to go with the Greenhouse path and join the TU/e contest. We thought it would be helpful to give ourselves a critical stance towards our own product, which would encourage rethinking design and feature requirements.

The contest events included multiple feedback sessions and workshops on different topics. We tried to have a minimum of two people pitching at each feedback session and at least one person per workshop. We would use the workshops and feedback to create and improve our slide deck for the next feedback session. In the end, we had a full slide deck for the finalist selection, attached at Appendix B.1.

At the start of the contest, making these slides and rethinking our ideas and design choices was an integral part of our work. We had long discussions and spent a lot of time on the TU/e contest during team meetings. Later, when prototype work became our priority, these discussions would be more inconsistent and individual between the designing, assembling, and programming of our prototype.

4.2 Results

As a result of our participation in the TU/e Contest, we have a slide deck and a poster that we used to support our pitch. These can be found in Appendix B.

At the 21st of May we attended the finalist selection event, where all the TU/e contest teams had a stand to pitch their ideas to the other participants and the Contest partners. Everyone could vote for teams to enter the contest finals, and some awards were awarded. Unfortunately, we were not selected for the finals, nor did we win any awards, but the important results are the experience gained, the connections made and most importantly all the feedback we received. In the following sections, we will go over all aspects of our pitch, and highlight some of the feedback we got during the feedback sessions and the finalist selection event.

4.2.1 Problem & Solution

In 2018, a football team was trapped in a cave in Thailand [16]. Heavy rainfall had blocked the way back. A rescue operation of 18 days followed, taking the life of one of the divers. All exploration to find the boys had to be done by human divers, which was slow and dangerous, because the very diverse terrain of the partially flooded cave could not be tackled with a robot.

This highlights the problem of having to explore hazardous, diverse, or unknown terrain while putting the lives of rescue operators on the line. All because of the lack of technological or autonomous solutions that can explore both air, land, and water at the same time.

The current alternative is to use multiple systems together, but this defeats the purpose of exploring the entire system. You would have to explore section by section, moving all the equipment over each hurdle. This takes time and, therefore, costs lives not only of victims, but also first responders themselves.

That is why we are building a fully autonomous system that incorporates these three modes of transport in a single robot. Using this, any terrain can be explored in one go, giving the necessary information to make a full rescue plan before even putting anyone in danger. This saves time and lives, but also reduces the anxiety of rescue workers and puts them at less risk. Using AI, we could even let the robot decide on a rescue plan itself, saving even more time.

4.2.2 User profile

Our main target user is rescue operations. In most cases these organizations are nationwide or even larger, this means a large budget. The worldwide Search and Rescue market size was 2.23 billion USD in 2023, and estimated to be 3.46 billion in 2034 [12].

Another huge potential market is the military. We spent a lot of time discussing how to handle them as a user. Although the military could greatly benefit from our product during rescue missions, none of us were interested in creating a potential weapon. We decided to restrict use for the military by offering it as a service, rather than as a product. In this way, we would have full control over the implementation of the system, preventing misuse.

In addition, there is effectively no real competition in this area. As discussed above, there are no solutions available that combine three modes of transport in one system.

The adjacent markets our product could find interest is in other areas that require scouting and exploration of versatile terrain. This could include monitoring of environmental and cultural heritage, archaeology, and agriculture. As our product is built on ROS2, it could also easily be used for research, and even filmmaking could be a market!

4.2.3 Feedback

We talked to many people at the contest events throughout the year. Feedback sessions with coaches, meet-and-greets with partners, and networking with other teams during dinner, to mention a few. During all these conversations, we collected a lot of feedback on our product, we will discuss the most notable feedback:

- **Contacting potential users**: while we did contact companies such as Avular, we did not contact potential end users to ask for their opinion on a product like ours. The majority of our audience expressed this as a big concern, we had no data on how useful our product would be, or how often our product would be used.
- **Price**: when discussing our price we named the production price of our prototype around 3000 euros. Many people asked us about our selling price and what the profit margin would be, which is something we had not thought off yet.
- **Future**: Many people asked us how we viewed the future of this team. Would we become a company? Or a student team?

4.2.4 Conclusion

Overall, the TU/e contest gave us a unique view of our product. As a group of solely technical people, we had never thought about the market aspect of our product, so looking at our solution through this new lens brought us a lot of useful lessons. Especially if any of us wishes to start a company in the future.

5 Reflection on this years project

5.1 Solution fit

The question we originally posed in Section 1.1 was "How do we effectively combine ground and aerial movement into one vehicle?". We created a well-defined problem statement and did not stray from this question at any point during the past year. We decided to approach this problem with a mechanical morphing vehicle. Like many of the solutions researched in Section 1.2, we thought that a morphing vehicle, while more difficult to create, had more potential. Currently, we appear to have been successful in combining ground and aerial movement. However, due to governmental regulations [8] in combination with a time constraint, we were unable to test the flying ability of our MTMS. Because of this, we have not yet compared the efficiency of our solution with other solutions. This is definitely a question we want to answer next year, as the development of MTMS only makes sense if it is effective in its assigned task.

Although we stayed close to the problem statement, there are a few discrepancies between our original idea and the final prototype. As visible in our budget C, we ordered a LiDAR sensor and a camera. We were planning to implement these to allow for long distance control but due to delays in receiving the components, we no longer had the time to implement these sensors. Camera's and sensors are a vital requirement for long term development, so these will be added first thing next year. We are also not controlling the propellors with ROS2 yet. This is currently an outside controller, which means that you need 2 controllers to control the system. This is not ideal and therefore the flying also needs to be implemented in ROS2. Finally, we need to look into the battery life of our system. While in peek demand the system can operator for about 15 minutes before draining the battery, this is fine for development, but for a long term solution we would need to find a way to elongate the life span.

However, since we created our solution from the ground up, it is very adaptable for future development. Any and all issues that we come across can be changed very quickly. This is especially nice during a development phase where the general approach is trial and error.

Despite these discrepancies, we achieved our goals of combining ground and aerial movement into one system, which can morph between the two modes, as can be seen from Table 4 in Section 3.1.

Our biggest edge on other multi-terrain solutions is our price, we were able to develop a low budget RC vehicle that combines ground and aerial movement. When we look at base cost of developing 1 system we are at 1.7K currently, a lot less then any of the researched alternatives.

5.2 Conclusion

We were able to make a low budget RC vehicle that combines ground and aerial movement, but we are unsure about its effectiveness. We are happy with what we were able to achieve in this first year of the project, but to make this a long-term solution a lot of work still needs to be done.

6 Future

As discussed in Section 5, the first steps in the continuation of this project are the following:

- LiDAR sensor and camera integration
- ROS2 flight integration: singular controller
- Request flight permissions for our MTMS, to enable testing

However, these should be reasonably easy tasks, as we already have all the necessary supplies.

Because we will stay in the development phase of this project, battery life is not a priority for next year. However, it should not be forgotten in future iterations.

In the following sections, we will discuss future plans for software, increased mobility, and autonomy.

6.1 Software

As seen in Figure 12 in Section 3.7 the software architecture has not been finished yet. To expand the functionality of our system, we plan to implement this full architecture. Additionally, we would like to utilize the full power of the action /morph. Actions have a feedback channel that allows requests to be canceled mid-way through. Using this, we could implement the canceling of morph requests, to make the recovery of an accidental morph faster.

To improve ROS2 performance, we wish to use Node Composition to communicate more efficiently between nodes. Node composition effectively places multiple nodes in a single process. This enables for process-local optimizations leading to a reduction of network related delays. Specific implementations would be zero-copy communication via IPC or loaned message optimizations. This saves resources and therefore improves performance, especially in our more recourse constraint Raspberry pi environment. Macenski et al., therefore recommends this node composition as a best practice [7].

Next, we can enhance the MicroROS performance, using multi-threading to run concurrent nodes more efficiently.

6.2 Water movement

Our big challenge for the nearby future is to expand our two modes of transport to three, implementing a mode that allows water movement. This addition would give our MTMS the signature innovative characteristic of being the first robot to combine these three modes into one robot.

We need to do more thorough research on how this can be implemented most effectively, but we suspect that the addition will require the following steps:

- Waterproof the body and electronics. This probably includes choosing a better material than PLA, as PLA is not waterproof [10].
- Add extra propellors optimized for water mobility. They could utilize the DC motors already in place, but this would require a redesign with a type of gearbox to accommodate the different torque requirements.
- Expand the ROS2 architecture with a water manager node, and add morph sequences to and from water mode.

This water movement will be the primary focus for next year. We hope to have a fully remote controlled MTMS by the end of next year.

6.3 Autonomy

In the far future, our ambition is to make MTMS autonomous. Considering our desired use case of scouting terrain for rescue operations, this is a vital feature. As in remote, difficult-to-access locations, there tends to be difficulties with internet connections. Thus, autonomous navigation is high on our priority list. This would mean that the system should find the shortest route to its goal while optimizing its available modes of transport and minimizing energy use.

In this case, the system would capture 360 video and LiDAR measurements to be analyzed by experts when the system returns. In this way rescue operators can make a plan for the fastest and most effective rescue.

This opens up another opportunity to speed up the rescue process: Make the system itself analyze the sensor data to determine (part of) a rescue plan! This could be implemented in many different ways, but also begs the question about ethics and safety of such a system. When starting with this functionality, we should conduct extensive research about the shape this will take.

7 Use of AI tools

We used AI tools in varying ways throughout the year.

7.1 Writing

None of us are strong writers, so we have used AI tools like ChatGPT to improve our writing. In doing this, we would first write a full version ourselves, and then ask ChatGPT to improve upon this text. Possibly with extra context like "Put more emphasis on x", but majority just as a punctuation and grammar checker. We would then use the generated text to rewrite our original text. This does not include copying complete paragraphs from the generated text, but can be related to spelling or story and sentence structure.

7.2 Programming

When programming, we used the inline completion feature of GitHub Copilot. This tool helped us write code faster, but we did not use it to generate code from scratch. Copilot suggests code snippets based on all code in the file. Since a lot of our code follows the standard ROS2 format as suggested on the ROS2 wiki, copilot's suggestions were mostly what we already planned to write. In that case, it was faster to accept such a suggestion than to finish writing it ourselves. Very rarely did Copilot suggest non-trivial ideas that did what we intended, and even then we had to modify these suggestions to ensure correct functionality.

We also used ChatGPT for debugging a few times. Most notably, during the setup of the microROS environment. The documentation for microROS can be quite sparse for specific topics. If we found that the microROS wiki and GitHub issues could not help us, we would use ChatGPT to try and discover the problem. More often than not, this would lead to nowhere. However, in some cases, it was able to pinpoint the problem from where on we would solve the problem ourselves.

References

[1] Pedro Alcantara. ros2-keyboard: Keyboard teleoperation for ros 2. https://github.com/pxalcantara/ros2-keyboard, 2024. Accessed: May 2025.

[2] Michael Bartlett. New soft robot morphs from a ground to air vehicle using liquid metal. https://news.vt.edu/articles/2022/02/eng-bartlett-morphing-drone.html, 2022. Accessed: 2025-05-22.

[3] Sihite, E., Kalantari, A., Nemovi, A., et al. Multi-modal mobility morphobot (m4) with appendage repurposing for locomotion plasticity enhancement. *Nat Commun*, 14(3323), 2023.

[4] Amedeo Fabris, Steffen Kirchgeorg, and Stefano Mintchev. A soft drone with multi-modal mobility for the exploration of confined spaces. In *2021 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 48–54, 2021.

[5] Eran Gefen and David Zarrouk. Flying star2, a hybrid flying driving robot with a clutch mechanism and energy optimization algorithm. *IEEE Access*, PP:1–1, 01 2022.

[6] Arash Kalantari and Matthew Spenko. Modeling and performance assessment of the hytaq, a hybrid terrestrial/aerial quadrotor. *IEEE Transactions on Robotics*, 30(5):1278–1285, 2014.

[7] Steve Macenski, Alberto Soragna, Michael Carroll, and Zhenpeng Ge. Impact of ros 2 node composition in robotic systems, 2023.

[8] RVO Netherlands Enterprise Agency. Rules for flying drones. https://business.gov.nl/regulation/drones/?gad_source=1&gad_campaignid=2066589066&gbraid=0AAAADAUIc8jCv7AkrkTN4k-vJ3KmUmHO&gclid=Cj0KCQjw_8rBBhCFARIsAJrc9yDirUlFbFvgA1sKdpo9BNlZSAicdAkWw19isFGhZnAbqvcv34ENB5gaAvkEEALw_wcB. Accessed: May 2025.

[9] Jared R. Page and Paul E. I. Pounds. The quadroller: Modeling of a uav/ugv hybrid quadrotor. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4834–4841, 2014.

[10] Plasticz. Is pla bestand tegen water? https://www.plasticz.nl/nl/blogs/plasticz-blogs/is-pla-bestand-tegen-water/, 2023. Accessed: May 2025.

[11] Evan W. Pratten. Joystick to voltage mapping. https://ewpratten.com/blog/joystick-to-voltage, 2020. Accessed: May 2025.

[12] Precedence Research. Search and rescue (sar) equipment market size, report by 2034. https://www.precedenceresearch.com/search-and-rescue-equipment-market, 2024. Accessed: April 2025.

[13] Valentin Riviere, Augustin Manecy, and Stéphane Viollet. Agile robotic fliers: A morphing-based approach. *Soft Robotics*, 5(5):541–553, 2018. PMID: 29846133.

[14] Jiefeng Sun and Jianguo Zhao. An adaptive walking robot with reconfigurable mechanisms using shape morphing joints. *IEEE Robotics and Automation Letters*, 4(2):724–731, 2019.

[15] Wikipedia. Polylactic acid — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Polylactic_acid, 2025. Accessed: May 2025.

[16] Wikipedia. Tham Luang cave rescue — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Tham_Luang_cave_rescue, 2025. Accessed: April 2025.

[17] Xerall. Xerall. https://xerall.com/. Accessed: October 2024.

[18] David Zarrouk and Liran Yehezkel. Rising star: A highly reconfigurable sprawl tuned robot. *IEEE Robotics and Automation Letters*, 3(3):1888–1895, 2018.

[19] Xiang Zhang, Faliang Zhou, Xiaojun Xu, Teng'an Zou, and Hu Chen. Configuration design and analysis of a multimodal wheel with deformable rim. In *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 772–777, 2019.

A Individual contributions

A.1 Daniel

Role in the team

At the start of the project I proposed building the MTMS and kept the team focused on the single requirement that the robot must both drive and fly. Together with Ismail I covered the electrical side of the work, while also joining Milosz in CAD to draw the wheel and first transformer arm models that fixed the overall size and structure of the prototype. Throughout the year I was the person teammates came to when a choice affected flight performance whether that was battery mass, servo torque or propeller clearance and I explained the trade offs so that everyone, including the first-year members, could follow the reasoning. Socially I was very engaging in every conversation within the team trying to turn every discussion into an action point that we as a team would work towards as well as questioned the specifics for raising personal understanding as well as the understanding of my teammates.

Contribution to process

I opened most hardware-related discussions during weekly meetings, steered the talk toward flight and power implications, and shared short how-to sessions on ESC flashing, Pixhawk setup and basic ROS to flight controller communication. When deliveries slipped, I helped split the task board into work that could proceed and work that had to wait, so mechanical printing and wiring continued while parts were still in transit. I also kept the component spreadsheet up to date and checked new ideas against weight and current budgets before we committed to them. I have also been actively participating in the TU/e contest for the team including the preparation, idea formation and presenting.

Before we have started to do the full CAD design I was presenting the team with several ideas of how we can get started with basic prototypes that allowed us to showcase the progress during the Honors intermediate presentations, ranging from CAD designs of a simple morphing mechanism, to iterations of the wheel design, to doing some side time independent work on getting ROS2 working on a small drone with a different flight controller to showcase that the concept on the bigger scale would also work. With this I was generally always kept the ball rolling providing sporadic ideas on how we can move forward while still waiting for the main primary components to arrive.

Contribution to deliverables

My main tangible output is the transformable wheel: a 200 mm rim that carries an internal 8 inch propeller on a custom lightweight bearing and links to the leg so it can rotate from driving to drone position, the design and its constraints are described in the wheel section of the report. I selected and ordered the flight hardware EMAX motors, 35 A ESCs, Pixhawk 6C, power-distribution board and 6S battery—documenting every item in the component list and wiring them on the bench. I soldered the PDB, made the XT-60 and telemetry looms, verified the 5 V / 12 V rails under a 10 A load, and tuned the first hover test in Mission Planner while logging data for the ESCs. I have provided several tutorials some more practical such as the github workshop to the whole honors academy and some internal which were on the inner setup and workings of the flight controller, the general physics of a drone and electronics to my teammates who were getting starting in that field.

As mentioned in the contribution to the process I engaged in multiple small iteration CAD designs and further served as advisor on a couple of key CAD designs. I was also able to use the gained ROS2 knowledge to apply it onto the small Crazyflie drone to be able to simulate it lifting off the ground and landing and then through the ROS nodes and just a single button click executing this in the real world, and conveying this research and ability to my peers.

A.2 Ismail

Role in the team

My role in the team was mostly on the technical side. I focused on making sure all the submodules of the system worked well together and that the overall system was technically sound. This meant thinking ahead about how different parts like the motors, servos and controllers would interact with each other so that we do not run into problems down the line. I also played a key role in initiating and supporting team skill development by proposing the idea of technical workshops early in the year. These sessions were aimed at building a foundation in areas like ROS2, microcontrollers and embedded systems, and CAD, which helped the whole team work more effectively on the technical challenges we later faced.

Contribution to process

At the start of the project, I was actively involved in selecting components and shaping the overall system design. I helped compare available hardware options, focusing on the main computer and many of the non-flight components, and worked with the team to allocate the budget across different subsystems. This included weighing trade-offs between cost, performance, and availability to ensure our final list matched both our technical needs and our constraints. I also contributed to creating the initial Gantt chart and made sure that the project timeline planning was feasible given the expected lead times and dependencies.

Later in the process, I focused more on designing and validating the power distribution system. I made sure our LiPo battery setup could safely and reliably power all components, including motors, servos, controllers, and converters. This involved testing motor stall currents, selecting appropriate wire gauges and buck converters, and checking that voltage and current levels would stay within safe margins during operation. I also researched best practices for power safety and worked closely with other members to integrate and test the full system.

In addition, when some of our parts were delayed, I helped gather alternative components from university stock so we could begin testing the ROS architecture. Using these, I built a working live demo with sensors that we used during our midterm presentation. I also looked into the trade-offs between ROS1, ROS2, and microcontroller-based setups, which helped guide our initial direction in software development. Throughout the year, I supported integration across subteams and participated in several TU/e Contest workshops and pitch sessions, representing the team during sprint events.

Additionally, building on the earlier initiative of introducing the technical workshops, I prepared and presented two technical workshops for my team: one on CAD design in Fusion 360 and 3D printing, and another on microcontroller basics and communication protocols. These sessions helped us speed up early-stage development and ensured that more team members could contribute effectively to the tasks involving mechanical design and embedded system integration.

Contribution to deliverables

My main contribution to the deliverables was the power distribution system. I designed and implemented a setup that allowed the LiPo battery to safely power all components in the system, including servos, DC motors, the main computer, the flight controller, and converters. This involved measuring stall currents, selecting and wiring appropriate buck converters, and verifying that current and voltage levels stayed within safe limits during testing. I also documented the power system and supported its integration into the full prototype.

In addition to the power system, I was involved in selecting many of the core components, particularly those not related to the flight subsystem. This includes the main computer, microcontrollers, converters, and sensors. I helped define the requirements for these components, verified their compatibility, and contributed to the final component list and budget allocation, as detailed in the budget appendix.

A.3 Matthijs

Role in the team

My role in the team in the first part of the year was more of an organizational role. I tried to keep the team organized with document structure for example. I tried to help remind everyone about deadlines and responsibilities. This did not work all the time, as we had some disagreements about documentation in the team, for example. Fortunately, I'm always open to discussion and we talked it out, though my stance did delay us a bit in the end.
Since I was still learning a lot of things like ROS and Fusion, I did not have a hard-skill responsibility or role yet, but it helped me prepare to take these responsibilities in the future.

Later in the year I continued to fulfill this organizational role by taking charge of the ordering of the components, but I also became one of the team members that was responsible for the software side of the project.

Throughout the year I was assigned to be the chair or scribe for some coach meetings. When I was, I made sure to plan the meeting in time and have an agenda prepared if I was chair, and made extensive minutes with clear sections for action points if I was scribe. I also tried to be at every team, coach and contest meeting in person.

Contribution to process

At the start of the year I did most of the exploratory research on the existing solutions. This gave us a basis and some background in the area of multi-modal robots and robot morphing, allowing us to define our challenge more clearly.
I also communicated with the university on the topic of the ordering of components. I made sure that we moved fast with ordering these ourselves, after half of our requested components were not ordered by the university.

Because Milosz and I will (hopefully) continue this project next year, without the other third years team members, we found it important to transfer the knowledge these team members had. We did this by making sure one of the two of us would always help one of the older team members in their task. In the last few weeks before the final demo day this approach was sometimes neglected a little, due to the increase in workload. However, we update all the team members on our progress and decisions during this time.

Contribution to deliverables

Together with Nora we are responsible for all the code that has been implemented in our robot. This is not limited to only writing the code, but also includes configuring the ROS and MicroROS environments, designing the architecture, and making sure the code runs solely by powering the robot on.

Mostly in light of knowledge transfer, I have also helped others in lots of different areas: Brainstorming mechanical designs, electrical wiring, and flight control, for example.

At the end of the year I wrote a significant part of the draft report, which later got refined by the other team members. I also did most of the formatting of figures, appendices, and references.

A.4 Milosz

Role in the team

My role in the team was an implementer and team worker role. Throughout the year, I actively joined different events for our team and offered a helping hand to team members with their tasks and with learning. With the help of others I also worked on implementing the knowledge for the design of our project. I was always open for discussion and any feedback on my skills and approach to problems.

Additionally, as a first year honors student, I also had the role of knowledge transfer for my team. To this end, I have actively participated in Daniel's and Ismail's work and learned basic electrical and mechanical concepts, so that we pass this knowledge to the next year.

Contribution to process

I helped my team with different design choices by researching and comparing different solutions. Some of those choices include: choosing software framework, drone design (four propellers, weight distribution), and project use case.

Additionally, I actively collaborated with my peers when it came to our tasks to streamline the progress. For example I collaborated with Daniel so that we could integrate his wheel design into our design and I helped Ismail with connecting and testing electrical components, which also made me learn a lot about electronics. I have also communicated to Nora and Matthijs so that they understand the mechanical design of our robot and can create meaningful code for it.

Contribution to deliverables

In the first half of the year, I was responsible for learning ROS2 software and passing on my knowledge to my teammates. For that reason, I have took a course on ROS2 and created workshops with presentations for my teammates, which were meant to teach new topics and help with thinking about how to program in ROS2. Later, those presentations served for a good background and reminders when it came to coding our robot.

In the second half of the year, I became mainly responsible for the mechanical part of our design and chassis design. To this end, I have visualized specific parts of our design in Fusion software and carefully tested if they fulfill our requirements. A detailed description of such is in the Mechanical section of this report. However, I am not responsible for the wheel design, which was Daniels's work. I only gave feedback on it, tested and integrated it with our structure. In the end, having the 3D prints, I worked on assembling the parts and testing the overall effectiveness of our structure.

Throughout the year, we have also participated in TU/e contest with our project. I actively participated in the events of this contest and brainstormed solutions / gave feedback regarding our project as a solution to a problem.

A.5 Nora

Role in the team

My role in the team was a leadership role and a software developer role. Throughout the year, I managed our deadlines and task distributions. During team meetings, I took on the chairman role. Making sure that we were productive during our team meetings and that we did not stray too far from the schedule. The TU/e contest was also one of my responsibilities. I contributed here by creating slides, the final poster, and pitching multiple times. When we got further into the year and got to work on more concrete details, Matthijs and I took charge of creating the ROS architecture and code. We ensured that our robot would function as desired before the end of the year.

Contribution to process

As I took on a leadership role during this project, my contributions to the process throughout the year were quite diverse. In the beginning of the year I made sure the new students were included in our meetings and that they would be transferred all our knowledge. This helped them to be included in our project, leading to better teamwork throughout the year

Throughout the year I made sure that everyone on the team was aware of deadlines, whether internal or external. These could be large deadlines, such as the final report deadline, but also small deadlines, such as booking a meeting with our coach. This greatly helped us speed up our process and ensured that nobody would miss anything.

I made sure we had consistent and efficient meetings, and if someone mentioned a roadblock during these meetings, I would be proactive and find a way to solve the problem.

Contribution to deliverables

Together with Matthijs I was responsible for the software created for our robot. We designed the ROS architecture and each greatly contributed to the programming of MTMS. We would have separate work sessions and update each other on the progress and errors we were facing. Personally, I am responsible creating the microROS and micro-ros-agent set up. This allowed us to run code on the arduino Due, and communicate with the nodes on the arduino from the raspberry Pi. I also created the basis of the servo and the DC motor code, which Matthijs expanded upon.

As I was also the primary person responsible for the Tu/e contest, I created most of the slides and I created the final poster. I pitched in almost all pitch sessions, and attended multiple workshops. Additionally, I am in charge of creating the poster for the demo day, and I wrote and proofread a significant part of the final report. I took care of making sure our report was uniform, and had similar language all throughout.

B TU/e Contest

B.1 Slides


MTMS

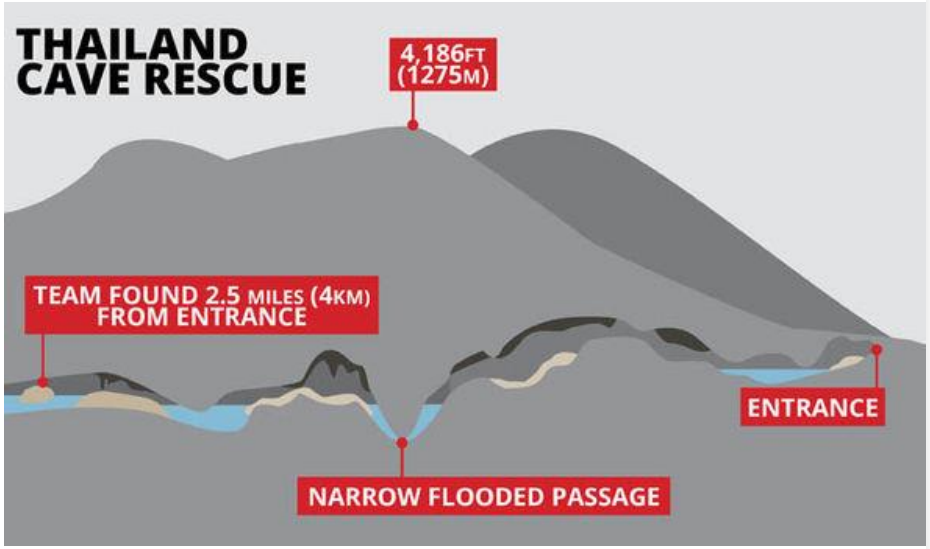
Problem Statement

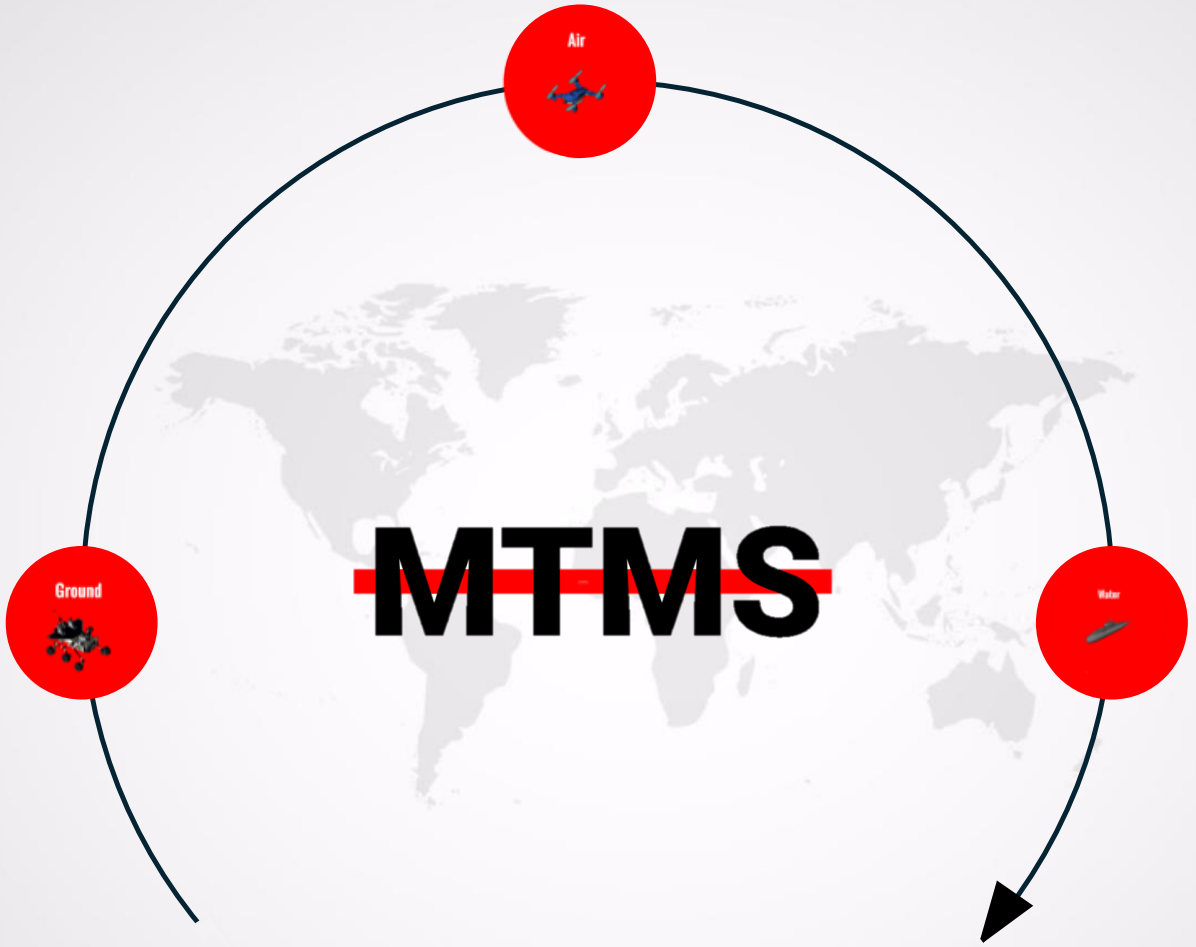
Rescue missions are slowed down by the need to navigate difficult/unknown terrain, delaying victim location and emergency response

Quicker, easier and more terrain exploration

Lives saved

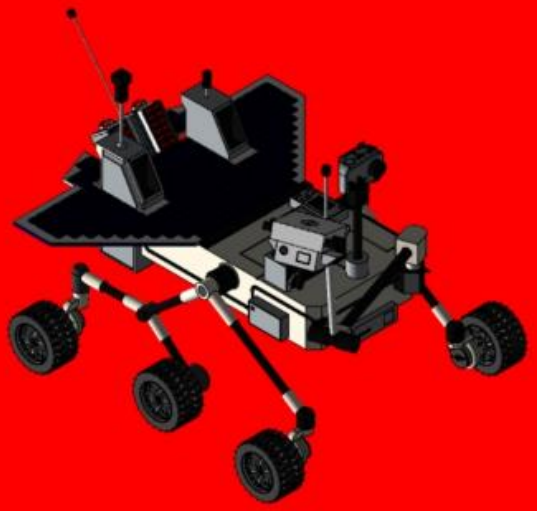
Alternative: using the three options separately



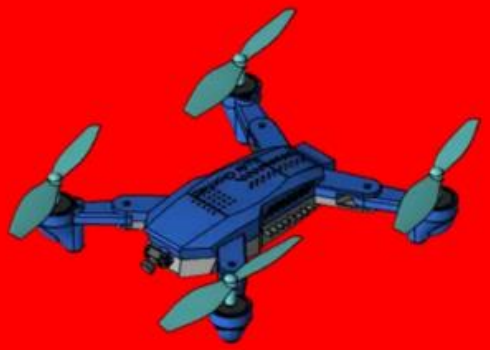


Solution

Ground



Air



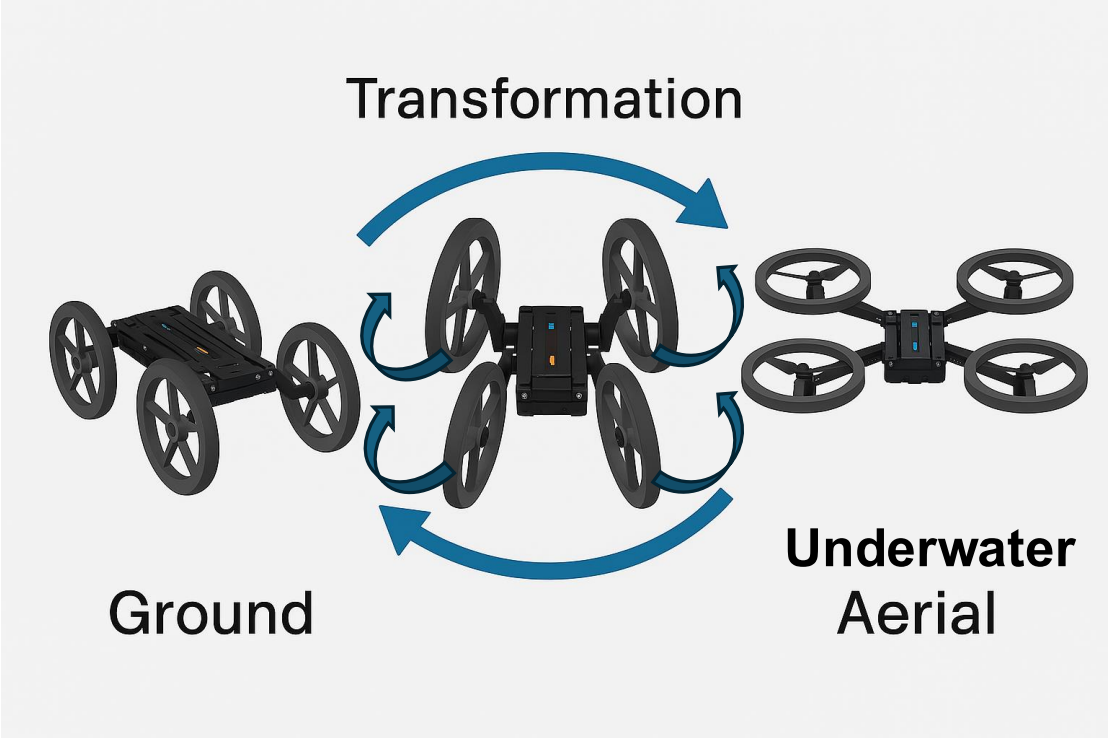
Water

Multi terrain Mobility System

Combining aerial movement, underwater and ground movement!

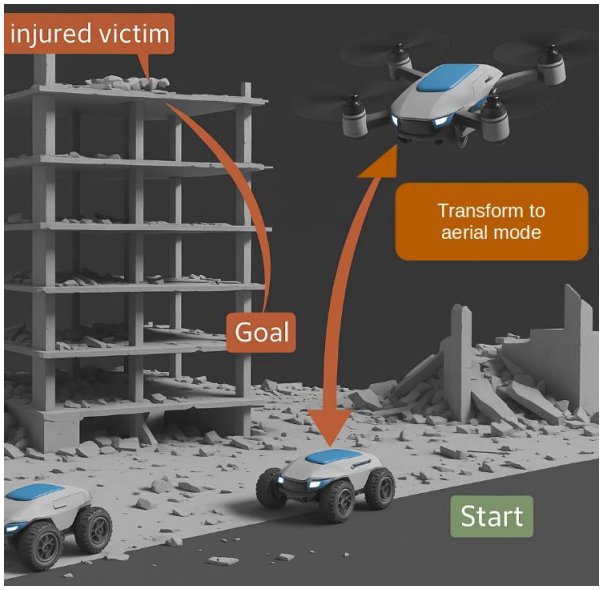
Technology

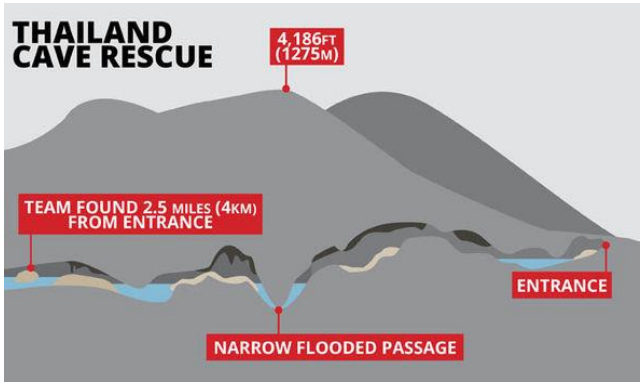
- **Transformable Mobility System**
- **Can easily switch between air, ground and underwater mode**



Product

- Ground, aerial & submersible mobility robot with morphing capabilities
- Manual control via ROS2 interface
- Foldable arms with wheels designed to host propellers
- 3D-printed components for flexible design





Competition

- No existing solution combines all three terrains (ground, air, water)
- More affordable than research alternatives like CalTech's M4
- ROS2 platform enables future autonomous capabilities
- Modular design allows for easier adaptability
- Smaller scale makes it accessible for varied applications

Markets

Main market -> search and rescue

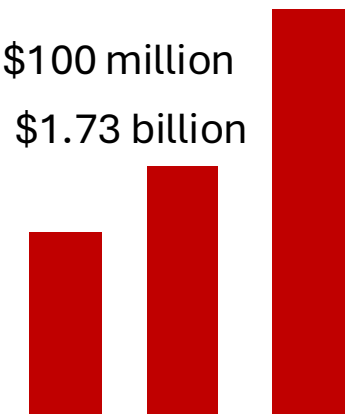
NGOs
red cross,
KNRM etc

Government
military, coast guard,
law enforcement

Turkey-Syria earthquake: $100 million

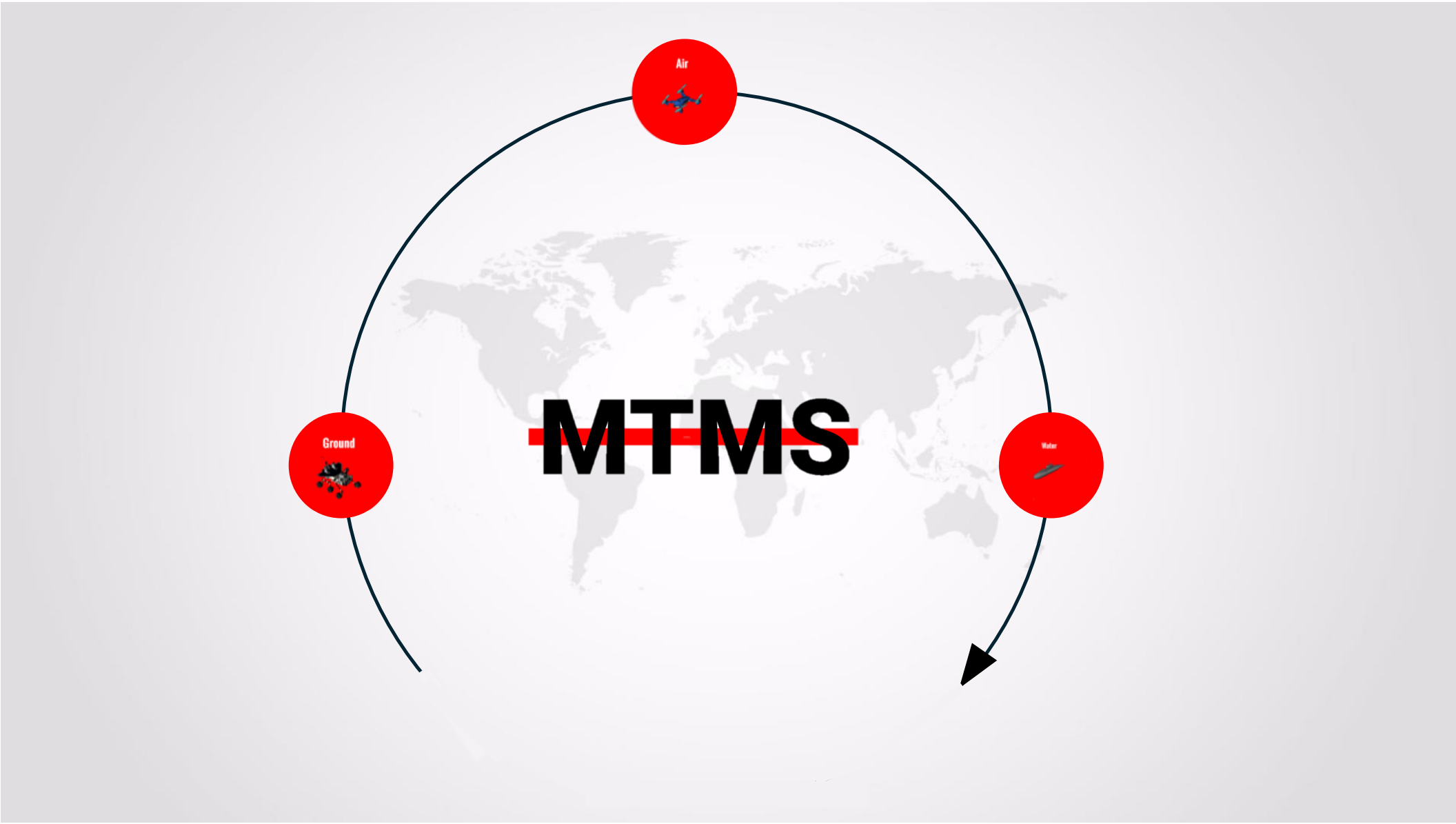
U.S. Firefighters: $1.73 billion

Deaths per year: ~ 50 000

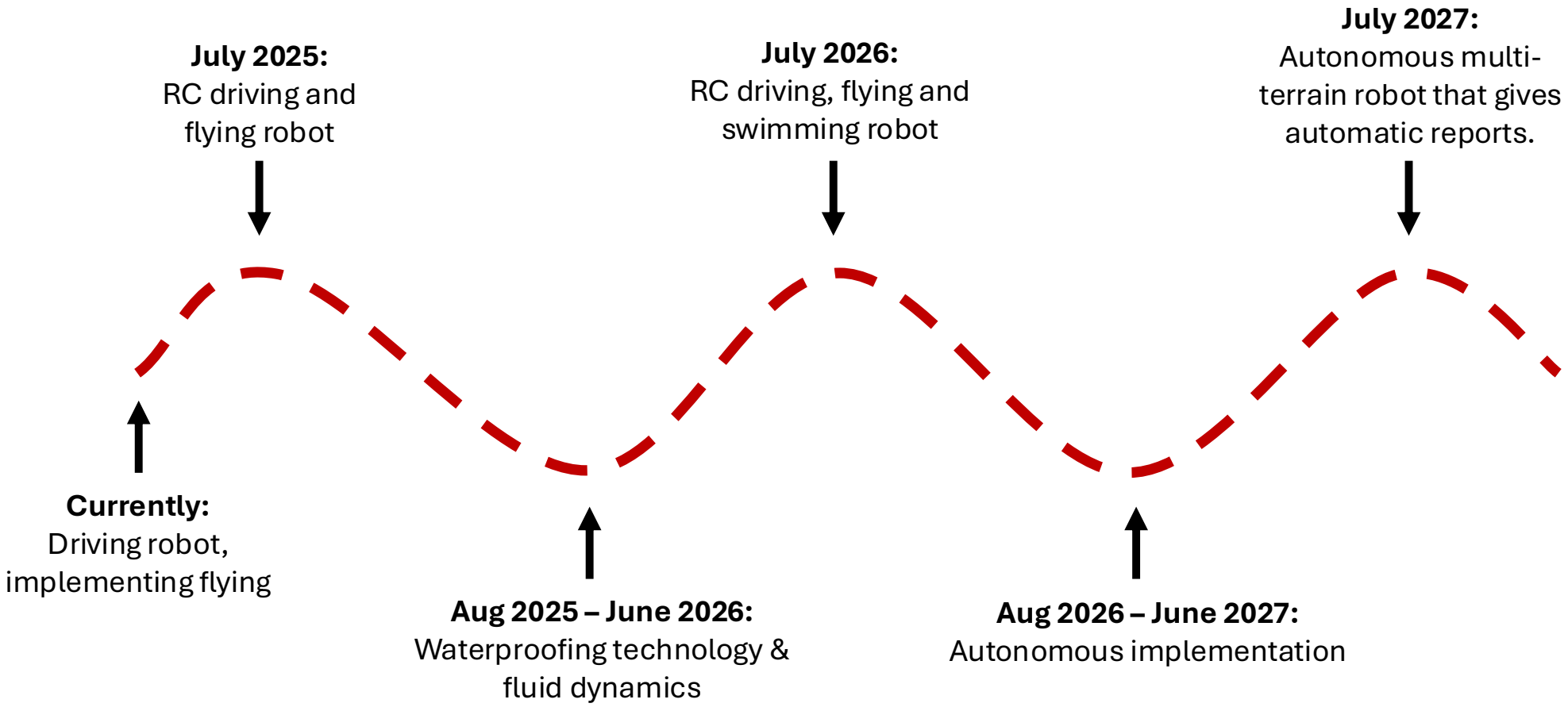


Adjacent markets:

- Environmental monitoring
- Archaeology
- Research
- Geopolitical
- Filmmaking


Air
Ground
Water
MTMS

Roadmap


July 2025:
RC driving and flying robot
July 2026:
RC driving, flying and swimming robot
July 2027:
Autonomous multi-terrain robot that gives automatic reports.
Currently:
Driving robot, implementing flying
Aug 2025 – June 2026:
Waterproofing technology & fluid dynamics
Aug 2026 – June 2027:
Autonomous implementation

Team

B.2 Poster

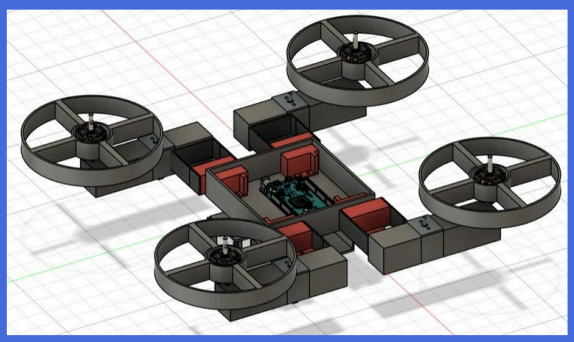
MTMS

One machine, every terrain

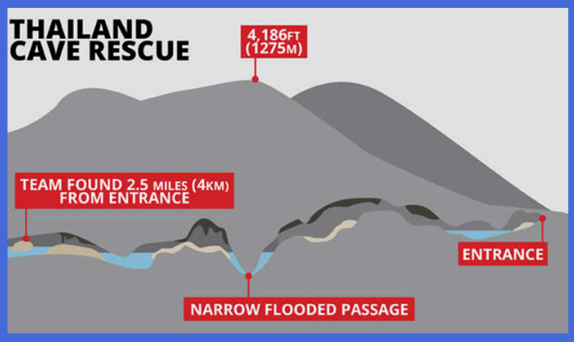
Rescue operators are **risking their lives** in unknown terrain, while limited by time and resources.

HOW CAN WE HELP?

Our MTMS **scouts**, locating victims without risking rescue operator's lives, supplying them with the exact route and resources needed.



Technology



Use case

Increase aid range
Increase aid speed
Decrease needed supplies
More lives saved! (~50000)

Our market is primarily **rescue organizations**, these tend to have a large budget, and we estimate our cost to **2000 Euro**'s

There are **no competitors**. Market alternatives combine 2/3 Research alternatives are more expensive.

Our focus for this year was on **ground** and **air** movement. We will expand upon this with water movement in the following year. For this we are **looking for experts in waterproofing and fluid-dynamics**

C Budgets

C.1 Budget V1

Component	Description	Price	Supplier	Delivery time
Main Computer	UP Xtreme i14	€967	Up squared/Mouser	Email inquiry
Flight controller	Pixhawk 6C	€220	Holybro	1 week
Secondary Computer	Raspberry Pi 5	€94.90	reichelt	3 days
Microcontroller	Raspberry Pi pico	€8.5	Tinytronics	3 days
Lidar sensor	For scanning area	€124	Generation robots	3 days
GPS Module	M10	€80	Holybro	1 week
Telemetry Radio	433MHz	€100	Holybro	1 week
3x Batteries	To power the system	€290.37	Racedayquads	3 days
4x wheel	To enable driving	€28	Amazon	3 days
4x 3MR Series - 3-Blade 8x4.1 Propeller	To enable flight	€20	masterairscrew	2 weeks
4x DC motor	To move the wheels	€74,04	Mouser Electronics	5 days
2x motor driver board	to control the motors	€33	Tiny tronics	1 day
4x EMAX Bullet Series 35A 3-6S ESC	To control and adjust power for the brushless motors	€80	Emax	1 week
4x Pro Series 2812 Brushless Motor	To move the propellors	€120	Emax	1 week
8x Servo (180°)	To enable the 2 joints per leg	€176	TinyTronics	1 day
2x Mini 12V 4"Stroke Electric Linear Actuator	To enable shifting between flight and driving	€60	Amazon	1 week
3x Camera's (120°)	To enable the person in control to see around vehicle	€91,28	Adafruit	4 days
2 kg PLA filament	For 3D prining custom parts for assembly and frame	€59,80	TinyTronics	1 day
Miscellaneous	Additional wiring, connectors and assembly parts	€90	TinyTronics, etc	Depends on parts
Total	-	**€2720.00**	-	-

Table 8: Component and Budget List for Multi-Terrain Robot

C.2 Budget V2

Component	Description	Price(Euro), incl taxes	Price(Euro), Excl taxes	Supplier	Price (Euro) per seller, incl shipping	Link	Delivery time	Depended on size/weight	Weight (grams)
LiPo batteries	To power the system	156.99		Amazon		https://www.amazo	3 days	No	645
2x Mini 12V 4"Stroke Electric Linear Actuator	To enable shifting between flight and driving	56.58		Amazon	259.79	https://www.amazo	1 week	Yes	160
4x wheel	To enable driving	40		Amazon		To be added later	3 days	Yes	
2x micro servo	for camera movement	6.22		Amazon		https://www.amazo	??	??	??
4x EMAX Bullet Series 35A 3-6S ESC	To control and adjust power for the brushless motors	74.35		Emax	220.73	https://emaxmodel	7 - 20 days	Yes	25.2
4x Pro Series 2812 Brushless Motor (1100KV)	To move the propellors	114.37		Emax		https://emaxmodel	7 - 20 days	Yes	286.4
Flight controller	Pixhawk 6C(Aluminium) + GPS(M10) + Power Module(PM07)	211.53		HolyBro	472	https://holybro.com	1 week	No	34.6
Microhard Telemetry Radio	P900 one radio	218.23		HolyBro		https://holybro.com	1 week	No	71
Power Distribution Board	To distribute power to ESCs	8.19		HolyBro		https://holybro.com	1 week	No	80
2x Linear Actuator Driver	For driving linear actuators	17.4		Kiwi electronics	21.35	https://www.kiwi-el	1 day	Yes	10
4x 3MR Series - 3-Blade 8x4.1 Propeller	To enable flight	20.55		Master Airscrew	43.85	https://www.maste	7 - 21 days	Yes	40.4
5x DC motor	To move the wheels	94.55		Mouser Electronics	100	https://eu.mouser.c	5 days	Yes	280
PWM Connector Break Out	(Electronic Speed Controllers) to a Flight Controller Board (FC) with	7.95		SpeedDrones	11.9	https://www.speedd	3 days	No	12
2x motor driver board	to control the motors	33		TinyTronics		https://www.tinytro	1 day	No	48
4x Servo (180°)	To enable the 2 joints per leg	88		Tinytronics		https://www.tinytro	1 day	Yes	320
2 kg PLA filament	For 3D prining custom parts for assembly and frame	36		Tinytronics	307	https://www.tinytro	1 day	No	600
Miscellaneous	Additional wiring, connectors and assembly parts	150		Tinytronics		Will be added later	Depends on parts	No	200
Lidar sensor	For scanning area	84.64		Antratek	88.39	https://www.antrate	1 day	No	145
Microcontroller	Arduino due	30		Reichelt		https://www.reichel	3 days	No	36
Main & test computer (so 2x)	Raspberry Pi 5 16GB	238		Reichelt		https://www.reichel	3 days	No	64
Cooling	Active cooler for rp 5	5.03		Reichelt		https://www.reichel	3 days	No	28
rpi power cable.	power the raspberry pi for testing	12.91		reichelt	435	https://www.reichel	3 days	No	-
Module 3 raspberry pi camera (120°)	To enable the person in control to see around vehicle	32.27		Reichelt		https://www.reichel	3 days	no	
Total					1960.01				3085.6

D Workshop Presentation example

ROS2 – part one

Setting everything up

- Any questions?

Colcon

- sudo apt install python3-colcon-common-extensions
- gedit ~/.bashrc
- source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash

Create a workspace for ROS2

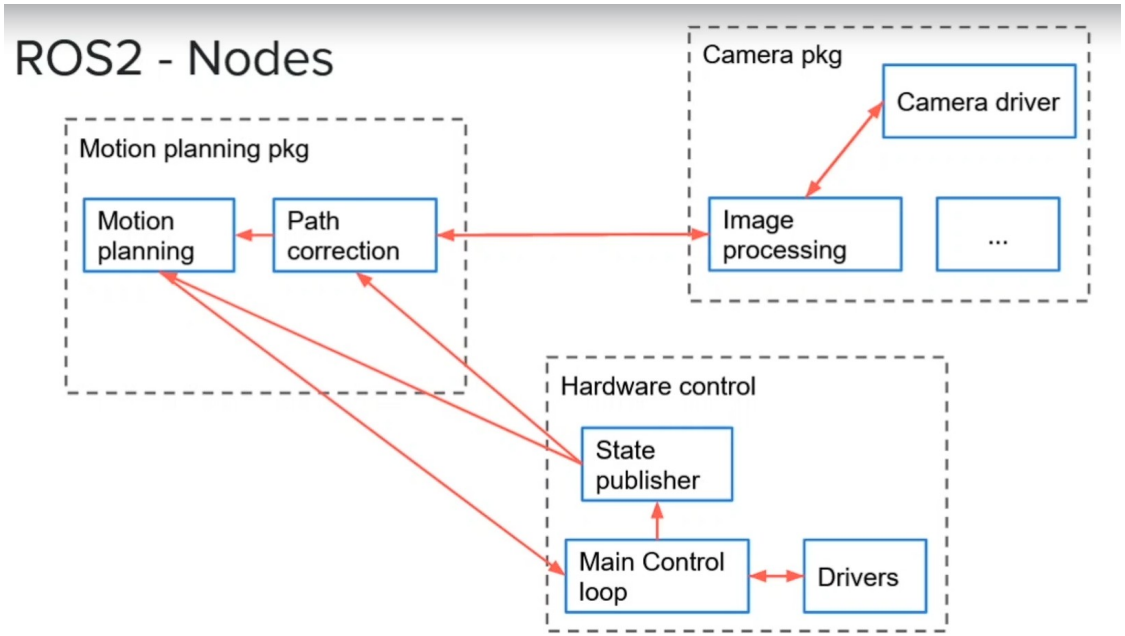
- mkdir ros2_ws
- cd ros2_ws/
- mkdir src
- colcon build
- gedit ~/.bashrc
- source ~/ros2_ws/install/setup.bash

```
18
19 source /opt/ros/foxy/setup.bash
20 source ~/ros2_ws/install/setup.bash
21
22 source /usr/share/colcon_argcomplete/hook/colcon-argcomplete.bash
23
```

Creating packages (python and c++)

- cd ros2_ws/src/
- ros2 pkg create my_py_pkg –build-type ament_python --dependencies rclpy
- cd ros2_ws/
- colcon build --packages-select my_py_pkg
- cd ros2_ws/src/
- ros2 pkg create my_cpp_pkg –build-type ament_cmake --dependencies rclcpp

ROS2 nodes

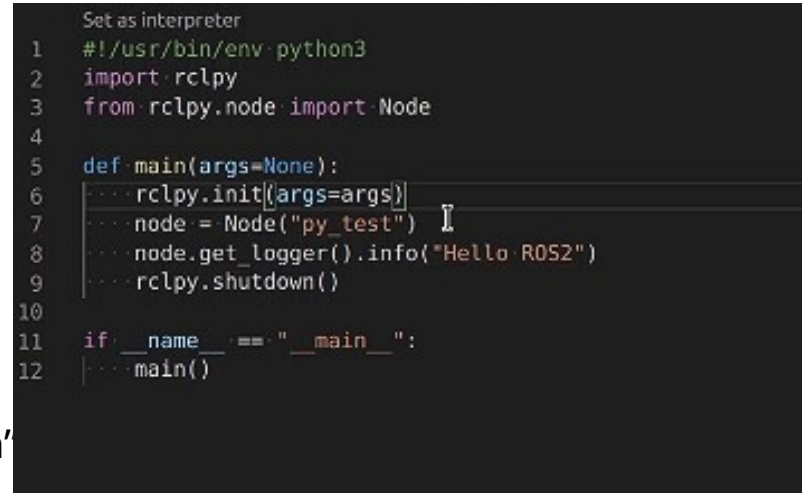

ROS2 - Nodes
Camera pkg
Camera driver
Motion planning pkg
Motion planning
Path correction
Image processing
...
Hardware control
State publisher
Main Control loop
Drivers

Python Node (simple)

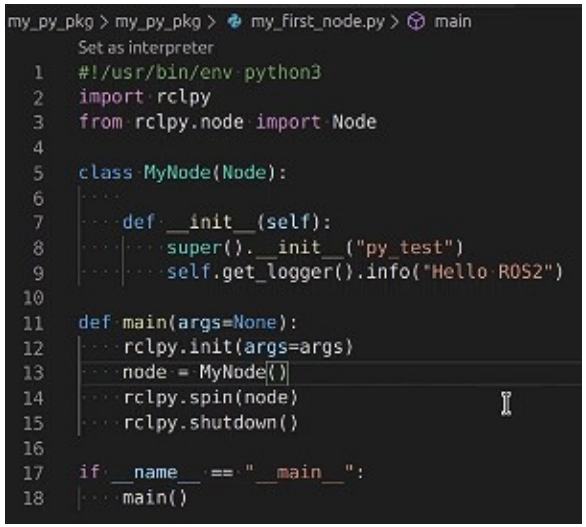
- cd ros2_ws/src/my_py_pkg/my_py_pkg
- touch my_first_node.py

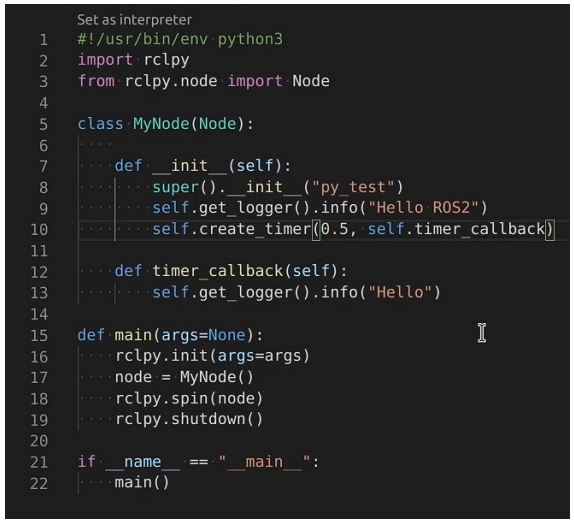
- chmod +x my_first_node.py
- ./my_first_node.py

- “py_node = my_py_pkg.my_first_node:main”
- cd ros2_ws/
- colcon build --packages-select my_py_pkg
- Source ~/.bashrc
- ros2 run my_py_pkg py_node



Python Node (OOP)





C++ Node (simple)

- cd ros2_ws/src/my_cpp_pkg/src
- touch my_first_node.cpp

- “/opt/ros/humble/include/**”

- colcon build --packages-select my_cpp_pkg
- Source ~/.bashrc
- ros2 run my_cpp_pkg cpp_node

```cpp
#include "rclcpp/rclcpp.hpp"

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<rclcpp::Node>("cpp_test");
    RCLCPP_INFO(node->get_logger(), "Hello Cpp Node");
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

```
add_executable(cpp_node src/my_first_node.cpp)
ament_target_dependencies(cpp_node rclcpp)

install(TARGETS
  cpp_node
  DESTINATION lib/${PROJECT_NAME}
)

ament_package()
```

C++ Node (OOP)

```cpp
#include "rclcpp/rclcpp.hpp"

class MyNode: public rclcpp::Node
{
public:
    MyNode(): Node("cpp_test")
    {
        RCLCPP_INFO(this->get_logger(), "Hello Cpp Node");
    }

private:

};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MyNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

```cpp
#include "rclcpp/rclcpp.hpp"

class MyNode : public rclcpp::Node
{
public:
    MyNode() : Node("cpp_test"), counter_(0)
    {
        RCLCPP_INFO(this->get_logger(), "Hello Cpp Node");

        timer_ = this->create_wall_timer(std::chrono::seconds(1),
                                         std::bind(&MyNode::timerCallback, this));
    }

private:
    void timerCallback()
    {
        counter_++;
        RCLCPP_INFO(this->get_logger(), "Hello %d", counter_);
    }

    rclcpp::TimerBase::SharedPtr timer_;
    int counter_;
};

int main(int argc, char **argv)
{
    rclcpp::init(argc, argv);
    auto node = std::make_shared<MyNode>();
    rclcpp::spin(node);
    rclcpp::shutdown();
    return 0;
}
```

Tools for ROS2

- ros2
- ros2 run –h
- ros2 node list
- Ros2 run my_py_pkg py_node --ros-args -remap __node:=abc
- Colcon build –packages-select my_py_pkg --symlink-install
- rqt
- rqt_graph

- ros2 run turtlesim turtlesim_node
- ros2 run turtlesim turtle_teleop_key