Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Robotics - Project Course - DVA490, 30 ECTS
Project in advanced embedded systems - DVA474, 30 ECTS

SAFE TRUSTWORTHY AUTONOMOUS DELIVERY SYSTEM

Kasra Ekrad
ked22001@student.mdu.se

Andreas Johansson
ajn19017@student.mdu.se

Walter Lagerhäll
wll19001@student.mdu.se

Sebastian Leclerc
sic19001@student.mdu.se

Elon Pettersson
epn19005@student.mdu.se

Erik Rågberger
err19002@student.mdu.se

Sharifeh Yaghoobi
syi22001@student.mdu.se

Examiner: Mikael Ekström
Mälardalen University, Västerås

Supervisor: Martin Ekström
Mälardalen University, Västerås

Supervisor: Emil Persson
Mälardalen University, Västerås

2024-01-11

Abstract

This report summarises the foundation built for an ongoing research project at Mälardalen University investigating autonomous delivery systems in dynamic environments. Autonomous vehicles face many challenges with localisation, path planning, obstacle avoidance, line of sight, etc. This project aims to build an Unmanned Aerial Vehicle (UAV) and Unmanned Ground Vehicle (UGV) system connected by a tether. A tethered solution for UAV's provides benefits such as not requiring an onboard battery and more robust communications. However, issues with variable shifting weight arise and that issue will be explored in this project. The other part of the system, the UGV needs to be designed to carry a payload together with the UAV, while still maintaining clear vision around it. The goal of the UAV is to be the eye in the sky and to give the UGV information about the surroundings where the UGV lacks a line of sight. For the UGV, a loading area was constructed with a landing pad on top. Light Detection And Rangings and cameras were mounted around the base of the loading area enabling vision in all directions. The UAV was designed and constructed from the ground up enabling good sensor placement, customisability, and sufficient lifting power for the tether. The UAV achieved manual flight with the tether attached. A pose estimation method was established utilising object detection of two landing pad marks that could estimate rotation and offset to the UGV. This method can be incorporated in the future when experimenting with autonomous flights.

Acknowledgements

We, in the Safe Trustworthy Autonomous Delivery System (STAD) project, would like to express our gratitude to the following individuals and organisations for aiding and guiding us in this project.

Firstly, we would like to thank the supervisors for the STAD project, Martin Ekström and Emil Persson at Mälardalen University (MDU), for their continuous support and guidance throughout the project. They have helped us move the project in the right direction by providing insightful feedback, thereby shaping the project.

We would also like to thank all the other teachers and members working at MDU for listening and providing mentorship whenever contacted. Specifically, we extend our deepest appreciation to Bengt-Erik Gustafsson for aiding us not only in designing the UAV frame but also in 3D-printing it in several iterations. At MDU, we finally want to thank the student Carl-Johan Höglind for sharing his expert knowledge on UAV applications. With Carl-Johan's help, we gained confidence in how to approach the design of the UAV.

Finally, we extend our sincere appreciation to all the companies that have been involved and helped in this project. Alstom, for lending the Nvidia Drive PX2. Volvo CE, for being involved in the UAV in the early stages and lending us the OAK-D Pro Depth camera. Imero Stål & Metall, for giving us aluminium to build a landing pad and loading area. Makers of Västerås, for helping us build said landing pad and loading area. SICK, for gifting us cables and connectors for their LiDAR.

Thank you all for your involvement and support.

Sincerely,

The members of the STAD project

Contents

List of Figures

List of Tables

1. Introduction

In a world where robotics and artificial intelligence play an increasingly significant role, the concept of autonomous vehicles is no longer an impossibility; rather it is the pulsating reality reshaping the way we envision transportation. The possibilities for the future are boundless and the notion of autonomous cities is increasingly emerging as a plausible concept in the foreseeable future.

Finnslätten is an industrial area in Västerås, Sweden with ambitious aspirations. Its vision is to become a technology center fostering collaboration and inspiration among various entities, including companies and universities. An objective for this area is to transform it into an autonomous city aiming to redefine traditional urban landscapes. In this futuristic urban setting, autonomous vehicles facilitate the transportation of people and goods.

The aim of this report is not to develop an autonomous city in the Finnslätten area. However, the report aims to serve as a foundation for the development of an autonomous delivery system within the Finnslätten area. The report is a part of the STAD project which is a collaboration between MDU, Volvo CE and Alstom. As the name of the project implies the objective of the project extends beyond the development of an autonomous delivery system, emphasising the parallel goals of safety and trustworthiness.

Autonomous delivery systems have successfully been developed by several companies[1, 2, 3, 4] consisting of a UGV. Some of the key components to the development of such platforms are advancements in Artificial Intelligence (AI) and the integration of high-quality sensors. This report focuses on providing a UGV with the required components such as computers and sensors to enable the autonomous concept for future iterations of the project. We present a system equipped with several sensors including Light Detection And Ranging (LiDAR)'s, cameras, Inertial Measurement Unit (IMU) and Global Positioning System (GPS), decided based on a thorough literature review [5, 6, 7, 8, 9, 10, 11, 12]. Considerations such as usability, adaptability to varying weather conditions, and future scalability were taken into account during the sensor selection process. The literature review also presents methods for localisation of the UGV and surrounding objects.

Moreover, apart from the UGV system this paper also aims to go beyond the development of a UGV and also incorporates a tethered drone solution, a solution proposed by several authors [13, 14, 15, 16]. The idea of this tethered UAV is to facilitate an elevated top-down view of the UGV's surroundings, equipping it with a 360 camera. The tethered solution offers numerous benefits compared to a regular UAV including wired power transfer and the possibility for reliable wired communication between the UGV and UAV. The report addresses challenges related to flight stability and a tracking algorithm for the UAV to autonomously follow the UGV using computer vision.

Flight stability is mainly investigated for regular UAVs [17, 18] and can typically easily be achieved through flight controller calibration. However, some works also exist related to flight stability for tethered UAV solutions [14, 15]. This project aims to construct a lightweight tether providing power and communication capabilities without compromising the flight dynamics of the UAV which neither of the mentioned works addresses clearly. The UAV showed quite promising results while being controlled manually in terms of flight stability, no autonomous flights were performed. However, this paper concludes that to improve the stability of the tethered UAV the shifting center of gravity depending on height has to be considered.

To track an external target in terms of a UGV with computer vision is a challenge that has been addressed before [19]. However, this paper proposes a new algorithm where two targets are detected on the UGV. Knowing the angle between these targets ensures alignment with the UGV which simplifies the tracking and ensures no twist of the tether. The algorithm showed promising results for detecting the two targets however, it was never tested during autonomous operation. In addition to the addressed challenges, the report also presents the construction process of the self-made drone, including design, hardware components and software configuration.

This paper hopes to inspire future research and provide a foundation for an autonomous delivery system. The system is intended to be used in future iterations of the STAD project for further development and testing of the autonomous concept.

2. Background

In the following section, a brief background is provided for the fundamental building blocks of an integrated UAV-UGV system. The main components of the system which are the UAV and the UGV are presented individually in Section 2.1 and 2.2. The relevant concepts such as autonomous flight, powering the system, and the vision algorithms for the system localisation are presented throughout the subsections. Finally, the related methods to integrate the subsystems are described in Section 2.3.

2.1 UAV

A UAV, or drone, is a small aircraft without a human pilot onboard [20]. They are commonly operated from the ground via a radio link or programmed to perform different missions. They typically consist of a frame housing the various connected electronic components such as sensors and controllers calculating how to actuate the motors, a power source and distribution system, and communication hardware such as radio and telemetry. UAV's come in many different shapes and forms, such as copters with varying numbers of motors, planes, blimps, etc. Many vendors sell UAV components, whether it is complete vehicles or separate customisable parts. Originally UAV's were developed for the military domain where they were used in war in the early seventies [21]. Today, however, such vehicles can be seen in multiple industries, amongst competing enthusiasts and hobbyists, and within several research fields [22].

2.1.1 Key UAV components, concepts, and flight

The main device controlling the UAV is the Flight Controller (FC), also known as the autopilot hardware [20]. One brand of FC's is the Pixhawk series which may run different autopilot software, known as flight stacks, acting as the main control loop [23]. For the Pixhawk FC's, the vendor-recommended flight stack is called PX4[1], an open-source software developed and supported by a community of academics and professionals. Another open-source flight stack option is called ArduPilot, also community-driven and backed by commercial partners [24]. To set up, configure, receive inflight telemetry, and plan missions for the UAV, some Ground Control Station (GCS) software is required such as QGroundControl [20]. The software is installed on a computer connected to the FC to flash and install a flight stack software, calibrate various settings, and set parameters as needed from the Graphical User Interface (GUI).

The UAV's various states are determined by the sensors and how the accompanying Proportional, Integral, Derivative (PID) controllers respond to sensor values [25]. In short, a PID controller is a feedback control system that has a proportional, integral and derivative term that tries to match an error to a desired output [26]. Furthermore, the PID controller can be tuned to achieve smooth transitions between states. The proportional term responds proportional to the current error. The integral term accounts for the error term over time and can help in reducing a Steady-State error. The derivative response considers the rate of change in the error. It can help with reducing overshoots and oscillations.

The Pixhawk 6X FC's, consist of redundant accelerometers, gyroscopes, and magnetometers which form the redundant IMU's [23]. Accelerometers are used on the UAV to determine changes in velocity while gyroscopes can aid in determining the UAV orientation. Magnetometers can further aid in determining the heading of the UAV by measuring the strength and direction of the magnetic field. Additionally, barometer sensors are also available for altitude control. It is important to note that sensors such as gyroscopes and accelerometers can experience interference and drift over time and may become unreliable the longer a mission is operating without re-calibration. The sensors pass input to the layered PID controllers which attempt to set response values based on the setpoints [27, 28]. These setpoints are set by the pilot controlling the UAV from a handheld Radio Controller (RC), a peripheral computer, or from a predefined mission set in GCS software. A well-tuned vehicle should match the response values as closely as possible to the setpoints and avoid overshoots. The goal is to configure the UAV for smooth flight dynamics regarding stability, performance, and control when performing various manoeuvres. Specifically, it is common to perform PID tuning for rate and attitude controllers in UAV's. Rate controllers focus on controlling angular rates, while attitude controllers focus on controlling the UAV's overall orientation.

Some of the most important flight dynamic parameters are the velocity from the throttle and the angles of rotation from the Center of Gravity (COG) [29, 30]. The three angles are called yaw, roll and pitch,

[1] https://github.com/PX4/PX4-Autopilot

collectively known as attitude. See Figure 1 for an illustration of the different orientations. Due to these flight dynamics, it is also crucial to correctly configure the motor's position in the X, Y, and Z-axis relative to the COG. Each time the distribution of weight is changed on the UAV, these parameters need to be updated for the UAV to understand its frame geometry. When performing some inflight manoeuvre, such as rolling the vehicle to the right, the vehicle should optimally stabilise in as few oscillations as possible. For many UAV's, a system log can be saved for each flight, i.e., when the vehicle is armed and fully powered [31]. These logs can then be reviewed for understanding the current flight behaviour related to how the PID controller is tuned, flight dynamics setpoints and responses when performing different manoeuvres.

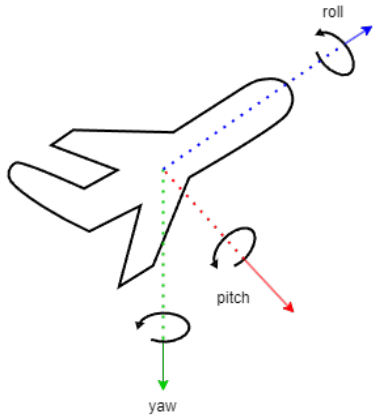


Figure 1: The three angles of rotation, known as attitude; roll, pitch, and yaw.

The output of the FC controls the various peripherals such as the Electronic Speed Controller (ESC) that drive the motors [30, 32]. Different ESC's hardware supports different protocols regulating, e.g., the motor speed and motor spin direction. Examples of such protocols which have different signal standards are Pulse-Width Modulation (PWM), OneShot, and DShot. The ESC's are configured in the GCS software to, e.g., set the PWM duty cycle, to avoid overheating and possible hardware damage. The aim is for all motors to have equal Revolutions Per Minute (RPM) given the same control input when the vehicle is level and sufficient power for take-off and continuous flight.

Various flight modes can be configured to define how the overall vehicle operates or how the vehicle responds to input from the RC [33]. Flight modes can be fully manually controlled, semi-autonomous, or fully autonomous. Consequently, some flight modes are easier to operate as a beginner. Depending on the FC model and flight stack, for semi-autonomous or fully autonomous operations, the UAV's must possess accurate knowledge of their position. This positional awareness can be derived from sources such as GPS or alternative positioning systems. With a GPS, a UAV can easily leverage a global position estimate to derive its current location. To use different flight modes, the RCs various switches can be configured with different modes or it may be possible to switch modes via custom software. Some important flight modes which PX4 uses are:

- **Manual:** The UAV can be controlled via a remote controller. Needs no sensor input.
- **Altitude:** Remote controller mode where the UAV maintains the current altitude. The UAV can still drift horizontally. Needs sensor input for height estimation.
- **Offboard:** The UAV follows commands provided by a companion computer. The UAV needs sensor input from positioning systems.

Another key component for an initial test flight is the RC, known as a transmitter, which communicates with the receiver located on the UAV [34, 35]. These devices communicate wirelessly, e.g., on the 2.4 GHz frequency band, using different firmware and protocol standards depending on the region where the UAV is operated. The communication may either be unidirectional, or bidirectional. Depending on the hardware and communication protocol, up to 16 channels are available to configure. Each channel can be configured with some manoeuvre, typically, the attitude controls, velocity, a vehicle arming switch, and some flight modes are minimally required. A popular protocol to achieve RC communication to a UAV

is the proprietary FrSky protocol developed by FrSky Electronic Co., Ltd[2]. The protocol supports basic control signals and in addition, it enables bidirectional communication, providing the capability to receive telemetry data such as Received Signal Strength Indicator (RSSI), battery voltage and more from the aircraft back to the transmitter.

The different flight stacks have several security measurements integrated into the software, e.g., to help avoid collision and damage [36]. The UAV will perform a pre-flight checklist to verify that the system is ready for take-off or warn if something is misconfigured. For all EU member states, some regulations exist for operating UAV's [37]. In Sweden, they are stipulated by Transportstyrelsen but do not cover UAV's operating in an indoor environment. For outdoor use, regulations exist regarding UAV identification, pilot licensing, whether the vehicle is over 25 kilos, if the vehicle is flown at an altitude above 120 meters, etc.

2.1.2 Power a UAV from a UGV

The operational endurance of a UAV is typically constrained by onboard battery limitations. Frequent charging compromises mission efficiency, prompting innovative solutions such as powering the UAV from a ground vehicle. This approach, utilising a power cable for a continuous power supply enables extended endurance and performance of longer missions.

In the power transfer mechanism, the power cable plays an important role [38]. Its length and thickness highly affect the power transmission efficiency. The following formulas describe how the length and thickness of power cables affect power loss:

$$R = \frac{\rho L}{A} \tag{1}$$

Equation 1 can be used to calculate the resistance of the cable [38]. Where ρ is the resistivity of the cable material, L is the cable length and A is the cross-sectional area.

$$V_d = IR \tag{2}$$

Equation 2 is used to calculate the voltage drop V_d [38]. Where I is the current flowing through the cable.

$$P_{loss} = I^2 R \tag{3}$$

Equation 3 calculates the power loss indicating the power dissipated as heat in the cable [38]. It is therefore desired to have as low power loss and voltage drop as possible in the cables. However, the decision of power cable thickness is a trade-off between weight and power loss/voltage drop. Heavy cables will also increase the power loss of the cables due to increased current draw for the UAV. Therefore careful consideration is required when selecting power cable thickness.

A possible solution to decrease the power loss in the cables is the usage of high voltages since it enables high power transfer with low currents [38]. However, it is important to remember that the UAV motors are made for a specific voltage range. To have a high voltage in the cables a DC-DC converter solution is required to step down the voltage in the UAV [39].

2.1.3 Autonomous Flight

Achieving autonomous flight in the context of UAV's is a complex task. A major part of achieving autonomous flight is the trajectory generation which allows the vehicle to move on its own [40]. The vehicle in question can be divided into several reference frames, and the operations that transform the vehicle between these frames is the trajectory generation algorithm. Both local and global frames are used, and the dynamics of a UAV system can be expressed with matrices using an inertial world frame, a fixed body frame,

[2]https://www.frsky-rc.com/about/

and an intermediate frame after yaw rotation. The utilisation of matrices has the benefit of allowing linear transformations to convert between these frames.

Other flight mechanics such as hovering around a point in space or keeping a stable altitude can be achieved using internal sensors such as IMU's. Altitude mode, which is one of the flight modes mentioned in section 2.1.1, utilises this for keeping a consistent height. The accelerometer within the IMU only provides the acceleration [41]. To know the position and velocity, the acceleration has to be integrated over time. Accelerometers suffer from drift, and it gets worse depending on the number of indirections that are used within the system's calculations. Indirections such as velocity and position have a quadratic and cubic relationship respectively to the acceleration, generating greater errors over time. External sensors will be needed for trajectory generation, but greater control is achieved through the use of IMU's.

Generating trajectories and additional flight-related tasks are mainly handled by the GCS to which the UAV is connected [42]. There are certain expectations of GCSs and they must contain the following components. First, maintaining control of the UAV is essential, hence, the communication links must be present for human intervention at all times. The degree to which the GCS is influencing the UAV does however depend on the autonomy level of the vehicle. Mission planning is the main method of setting up autonomous trajectories. This feature must be able to take the limitations of the UAV into account and generate an optimal path suitable specifically for that vehicle. Analytical tools for data collection during flights as well as post-flight analysis are available in GCSs. In common software today such as PX4-Autopilot or Ardupilot, much development has gone into autonomous flight and they are capable of being used in conjunction with most available hobby GCSs[3] available today.

Mission planning requires external sensor data that has knowledge of the world around the vehicle, and in most applications available today, the standard is to use GPS as the world frame reference point [43]. PX4-Autopilot offers integration of several world frame reference points such as GPS, Motion Capture, Visual Odometry (VO) for navigation, and even a barometer for height estimation, each with its benefits and drawbacks.

For global position estimation, GPS is the most used technology available today, providing coordinates in all three dimensions and being accessible anywhere on Earth [41]. As such, it is suitable for high-altitude flight over open environments as that allows for a reliable connection to satellites [43]. Naturally, the drawbacks of using GPS present themselves when flying indoors, in narrow environments such as between buildings, or when great precision is desired. When indoors, the signal-to-noise ratio decreases to unacceptable levels which provides unreliable readings. The Root Mean Square (RMS) error can vary up to 5 or 10 meters depending on the material of the building when inside [44].

Motion capture is useful in the case of indoor flights as it involves computer vision to locate the UAV within its field of view [43]. This can be an attractive solution when the area of interest for all navigation is in a very limited space. Additionally, it provides high accuracy which is desirable in closed environments.

In VO, the vehicle's position and orientation are estimated using images taken with a camera [45]. Consecutive images are analysed to find the correlation from one frame to another, and in this process, motion can be estimated as well. Feature extraction techniques are used to find these correlations, and it is an ongoing research field where different methods are proposed, each with their advantages and drawbacks. Optical flow is one of many techniques used for exploiting the nature of light and images [46]. Pixels that are located close together most likely represent the same surface. Additionally, these pixels are inclined to move the same amount from frame to frame, when the camera is moving. It is entirely up to the application which sensor localisation technique should be used. Sensor fusion technology also exists so that multiple navigation sources can be used simultaneously [47].

2.1.4 Companion Computer

While the FC often comes with autonomous flight abilities, a communication link with a GCS is required [48]. This limitation can be circumvented by utilising a companion computer, which in the context of UAV's, is an external device capable of communicating with the FC. Messages can then be sent and received between these devices, providing functionalities similar to GCS. Depending on such messages, the companion computer can be programmed to make decisions accordingly. Thus, allowing the UAV behavior

[3] https://ardupilot.org/copter/docs/common-choosing-a-ground-station.html

to be tailored specifically for the requirements of a given application, such as implementing custom features that are not available in GCSs.

Many relevant UAV software applications available today, use Micro Air Vehicle Communication Protocol (MAVLink)[4] for messaging [48]. It is a custom communication protocol built specifically for UAV's. MAVLink uses a subscription and publishing pattern, and point-to-point communication [49]. Different MAVLink systems support different messages, referred to as dialects. While many dialects exist, providing tailored functionality, there is also a common dialect that most GCSs support.

For development, MAVLink Software Development Kit (MAVSDK)[5] can be used instead of directly using MAVLink. MAVSDK is a C++ library, including many other language bindings, that uses MAVLink internally. MAVLink exposes much to the developer, and in MAVSDK most of the information is abstracted away. This results in a library that is easier to use, less error-prone, and allows for faster development [50]. MAVROS[6] is the third alternative which essentially combines ROS and MAVLink by providing a link between them. It is, however, in the process of obsoletion as it only supports Robot Operating System (ROS) 1.

2.1.5 Computer Vision

Computer vision is a vast research field where the desire for vision-based applications grows rapidly [48]. The aim of these applications goes beyond achieving vision to also making computers comprehend what is perceived, such as detecting objects. Several methods exist and have been explored through the years, but Convolutional Neural Networks (CNN's) are the most popular algorithms as they require the least pre-processing and are the most efficient to deploy. The CNN is a type of artificial neural network in the Machine Learning (ML) field that utilises convolutions on some of its layers. It is a mathematical operation well suited for images because pixels close together generally belong to the same object. These neural networks are trained using deep learning, which means that the data is raw, as opposed to labels. In the context of images, this means raw pixel data, hence, minimal to no pre-processing is required which is desirable for speed.

While many computer vision algorithms are built on CNN's, they differ from each other and excel in specific areas [51]. It often comes down to trading off accuracy against speed. Fast Region-based Convolutional Neural Network (R-CNN) which replaced the regular R-CNN is a two-step object detection technique with relatively high training and testing speed due to being able to share the computational results between its internal layers when data is passed. Two-step in this context mean that two passes through the network are required for detection.

You Only Look Once (YOLO) as opposed to Fast R-CNN is a single-pass algorithm that is capable of achieving class prediction and object detection faster and more reliably [52]. The structure is relatively simple which allows any current-era computer with a Graphics Processing Unit (GPU) to achieve real-time image classification. This algorithm has been reported to achieve detection speeds of up to 150 frames per second. The reason why such performance can be achieved is because the object detection problem is rethought as a regression problem [53].

Visual computing applications in the context of UAV's have been growing in recent years [47]. Optical flow and VO which were discussed in Section 2.1.3 affect the vehicle directly, as they serve as the reference point within the environment. While this provides knowledge about where in space the vehicle is, many applications instead utilise vision to detect and localise objects surrounding the vehicle. This is also of interest to many researchers as collision avoidance is equally important as localisation.

2.2 UGV

A UGV is a type of unmanned mobile robot that can be equipped with various sensors to obtain intellectual capabilities to operate autonomously. The term *autonomous* means that the robot can navigate and plan its path, as well as detect obstacles and avoid them by itself [54]. An autonomous UGV is designed to operate on the ground and carry out various tasks assigned to humans. The tasks may be too rigid, dirty, slow, or

[4] https://github.com/mavlink/mavlink
[5] https://github.com/mavlink/MAVSDK
[6] https://github.com/mavlink/mavros

dangerous for human beings [54]. A Husky A200 from Clearpath Robotics is one example of an unmanned mobile robot with a large payload and an appropriate platform for research applications. The Husky is compatible with ROS and has a companion computer called Mini ITX, however, it requires expansion to be able to operate autonomously[55]. While this project does not specifically address navigation and path planning, there are some key features to establish the foundation of an autonomous system that aligns with the goals of this project.

2.2.1 Localisation

One of the key fundamental components to enable an autonomous system is localisation. Localisation in the context of mobile robotics refers to the ability of a mobile platform to determine its position and orientation relative to its environment [56]. The localisation process heavily relies on sensor data which enables the platform to perceive its surroundings. Sensors used for localisation of an autonomous platform can be divided into proprioceptive and exteroceptive sensors. Proprioceptive sensors are sensors measuring internal values to the system and are typically in the context of mobile robotics which includes IMU and wheel odometry. Exteroceptive sensors are sensors that acquire measurements from the surroundings of the robot. LiDAR and cameras are two examples of sensors that are exteroceptive. Each sensor type (both proprioceptive and exteroceptive) has its strengths and limitations and its data requires fusion to ensure accurate localisation of the mobile platform. Extended Kalman Filters and Particle Filters are two common techniques for sensor fusion and estimating a platform's pose accurately. Usually, a combination of proprioceptive sensors and exteroceptive sensors are used when performing sensor fusion to estimate the pose.

2.2.2 Computer vision approaches for autonomous platforms

In the context of mobile robotics computer vision has become pivotal for machines to understand their surroundings and perform decisions based on the sensed information [57]. Within the field of computer vision sensor data are most often acquired from sensors such as cameras or LiDARs. The data acquired from these sensors can be used for a wide range of purposes. Cameras play a multi-functional role and can for example be used for object detection, semantic segmentation or depth perception. These features facilitate recognition of objects such as persons, other vehicles and traffic signs. They also enable road following and terrain analysis, etc. Furthermore, the depth perception using both cameras and LiDARs can be used to detect obstacles and hence also to avoid collisions. Utilising computer vision on a mobile platform unlocks new dimensions of autonomy and provides the robot with a more profound understanding of the complex environments these robots navigate.

2.2.3 Nvidia Drive PX2

The NVIDIA Drive PX2 is a cutting-edge computing platform specifically designed for autonomous driving applications [58]. The platform is built in a modular fashion which enables it to be integrated into various vehicle configurations. The platform boasts a large number of sensor interfaces with 12 Gigabit Multimedia Serial Link (GMSL) ports, 6 Controller Area Network (CAN) ports and many more, enabling it to have a large variety of sensors equipped.

The drive platform is accompanied by an Operating System (OS) tailor-made for autonomous vehicles [59]. The OS includes software necessary for enabling autonomous driving such as CUDA and tensorRT. These libraries enable faster deep learning and ensure real-time decision-making capabilities. The key component on the platform is DriveWorks a library that contains features such as custom-trained networks for object detection specifically trained to detect people, road signs and vehicles. The library also contains functions for interfacing sensors and seamlessly sending the data between functions. The goal of these features is to help reduce the time consumption when building and programming autonomous vehicles.

2.3 Integration of UGV and UAV

Integration of sub-systems is typically achieved by connecting different system components through communication schemes. As Thulasiraman et al. mentioned in [60], this can be done via any communication link, whether wireless or wired. However, each communication scheme comes with its own set of benefits

and drawbacks. Furthermore, the quantity of resources needed to set up the software infrastructure for any new robots is one of the challenges in the software and robotics realm. Thousands of lines of code have to be written to interface and operate with the hardware in the system under development. As a result, for each new system or component, new software should be developed. Thus, utilising frameworks and libraries is crucial to reduce the development time of robot systems. In this section, methods on how to connect devices will be discussed, as well as tethering a UGV to a UAV.

2.3.1 Sockets

Sockets are a common way to quickly establish communication channels over a network via an Application Programming Interface (API) to allow different devices to communicate [61]. A socket can be seen as an endpoint to facilitate the sending and receiving of data and is typically used in a server-client fashion. There are three types of sockets, TCP, UDP, and Raw [62]. A TCP socket, also known as a stream socket, provides reliable, connection-oriented communication where the order of data is guaranteed. UDP sockets, on the other hand, are connectionless and unreliable. Furthermore, there are no guarantees for packet deliveries, and no acknowledgements are made, as opposed to TCP. Therefore, these types of connections may provide lower response times in data delivery. Raw sockets allow direct access to lower-layer communication protocols such as ICMP or IP, bypassing the standard TCP/IP stack. Although network sockets are commonly used in the TCP/IP stack, socket programming exists for other communication protocols such as Bluetooth, one example being the BlueZ library[7].

Upon creation of a socket, a socket descriptor is returned, which is the handle that is used for any ensuing communication [62]. Subsequently, a server typically binds a socket to a specific address and port which in turn will allow remote connections on that socket. The server needs to be in a listening state to permit connections on the previously bound socket. A client may then connect, and depending on whether TCP is used the server may accept to establish a connection. For UDP, each datagram is sent independently without establishing a connection. Send and receive functions are used in both the server and the client to exchange data. When the communication is over, the sockets need to be closed to release any resources that are used. See Figure 2 for an illustration of how sockets are created, maintained, and closed.

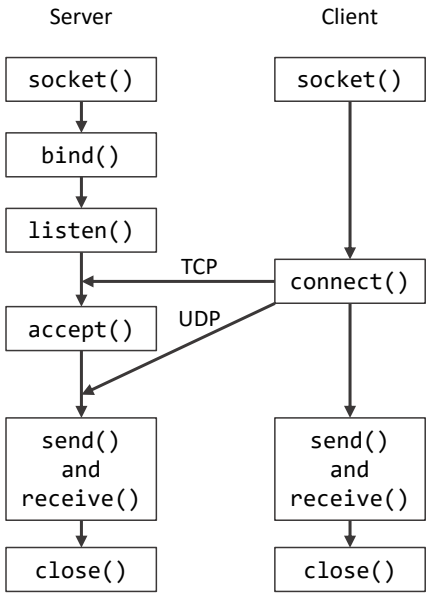


Figure 2: Socket connection flow.

[7] https://www.bluez.org/about/

2.3.2 ROS

As mentioned in Section 2.3, to reduce the development time, utilizing the libraries and frameworks is vital in nowadays projects which also applies to ROS. This resource disposal can be reduced by using ROS as a centralized software infrastructure. ROS, developed and maintained by Open Robotics, is a fundamental framework that controls the software infrastructure in the robotics domain [63]. ROS serves as a toolbox, providing sets of tools and libraries that have been designed to simplify the complexity of creating a new robotic unit.

The ROS was first launched in 2007 under the ROS 1 name [64]. It became the officially recognized norm for many robotic applications. A communication middleware known as *rospy* for Python and *roscpp* for C++ is the main component of ROS 1. The communication between several nodes is facilitated by a centralized controller with a publish-subscribe method. Despite its popularity, ROS 1 has many drawbacks, including problems with scalability, security, and real-time performance. The creation of ROS 2 was driven by these constraints. Although ROS 1 established the groundwork for the general application of ROS in robotics, ROS 2 is a great advancement with improved features and functionalities. The purpose of ROS 2, is to fulfill the changing demands of the robotics field and to solve the weaknesses of ROS 1. The first release of ROS 2 occurred in 2017. Increased real-time capabilities, stronger security features, more flexibility, and improved communication middleware using Data Distribution Service (DDS) are just a few of the major enhancements brought forth by ROS 2. The ROS 2 middleware has not undergone significant changes, yet it still offers many DDS implementations.

3. Related Work

This section is divided into three separate parts. Initially, in Section 3.1, works focused on the control of UAVs are explored. Subsequently, in Section 3.2, literature related to the construction of UGVs is presented. Finally, in Section 3.3, an overview of works that integrate these kinds of sub-systems via data communication and a physical tether is given.

3.1 UAV

In this section, related works to UAV control strategies are discussed in Section 3.1.1 and UAV localisation techniques are discussed in Section 3.1.2.

3.1.1 UAV stability and control strategies

In [17] Etts et al. have researched how to develop an autonomous UAV that can be utilised to collect data for environmental science purposes. They argue the benefits of their system being customisable, reusable, and less expensive as opposed to systems based on balloons which might not be recoverable. One of their main focuses related to the STAD project was to stabilise the vehicle's flight. In contrast to the STAD project, they used an ArduPilot-based hexacopter, i.e., a UAV with six motors and propellers. When experimenting with the UAV, the vehicle experienced drifting in different directions, a common occurrence when operating UAV's. The author's main theory behind this drift is due to the FC experiences excessive vibrations. Their solution was a 3D-printed vibration-dampening FC platform which lowered vibration measurements for the accelerometer from 8 m/s^2 to 1 m/s^2, however, more testing needs to be performed.

In [18] Jiang et al. developed a framework integrated with ROS for estimating and controlling FC states such as velocity and position with external sensors for better flight stability. The UAV was a DJI-based quadcopter which was evaluated with the framework in both simulations and experiments to follow a challenging trajectory. Similarly, in the STAD project, it is of interest to investigate alternative solutions in GPS-denied environments, such as indoors. The results of the experiment are measured via an external motion capture system. They show less than 0.075 meter positional mean error and standard deviation in the setpoint compared to the response values of the trajectory based on their flight log.

When comparing the stability of tethered UAV's, Yang et al. have in [14] implemented a control strategy for such systems which does not require GPS. The strategy is based on Lyapunov's second method for stability, which is used in dynamic systems and control theory. Two external controllers are implemented, one onboard their UAV and one for their winch system, based on IMU data and camera module data. Concerning the STAD project, the aim and requirements are similar. However, the STAD project did not have access to a winch system, nor was the initial focus to fly with a taut tether. The simulation results from Yang et al. show that the tether is kept taut and relatively stable, where variables such as velocity measurement, noise, delay, and some levels of wind are considered. Experimental results argue that when using their control strategy, the vehicle could land safely with a steady velocity, although the description of how the experiment was performed is vague.

3.1.2 UAV localisation techniques

In the article [65], an investigation was made where sensor fusion of UAV inertial data and optical flow was explored. The fusion was made using an Extended Kalman Filter (EKF) and showed promising results. They found that the fused data was much smoother and generally closer to the ground truth, proving that this is a viable solution in indoor environments, where GPS is not an option. The circumstances were however ideal without noise consideration, indicating that more research needs to be done.

Wei Yan et al. [66] investigated monocular VO for velocity estimation on their UAV. Specifically, they used the Oriented FAST and Rotated Brief (ORB) algorithm for extracting image features that later were processed using the Lucas-Kanade Optical flow method. Comparing their observed results with a GPS as ground truth, showed very similar accuracy. The best results were achieved when the velocity was around four meters per second. A limitation of the system is that the vehicle must operate at a constant altitude and keep a gimbal pitch of zero degrees. The significance of their work is that GPS like performance could be achieved using a monocular camera which is comparatively worse than stereo vision VO.

Both of these papers discuss the utilisation of vision to know the ground truth position of the UAV. Tian et al. [19] instead use vision to track an external target, a UGV. They use the YOLO algorithm to maximise detection performance while also achieving adequate accuracy. With recent advancements in this technology, they had access to YOLOv3-tiny which performs exceptionally well. Additionally, the Kernelized Correlation Filter (KCF) algorithm was used as it is significantly faster to compute. KCF is capable of keeping track of an object after it has been detected but suffers from drift and might lose the target, which it is unable to recover from. The solution they found was to use the algorithms together. After YOLO detects the target, KCF keeps track of it until it is lost, and then YOLO finds it again. To deal with uncertainties a Kalman filter was applied. The results showed that the UGV could be tracked in an effective and stabilised way. These findings are meaningful for the STAD project as the UGV is to be tracked by the UAV. Performance is of great concern for which YOLO is an attractive algorithm to explore.

In STAD, the UAV tests are performed indoors, to prevent snow and rainfall from damaging the equipment. Vision appears to be a valuable solution as success has been achieved in both [65] and [66].

3.2 UGV

Related works of the localisation techniques in UGV are discussed in 3.2.1 and related work for sensor efficiency in diverse conditions is discussed in 3.2.2.

3.2.1 UGV localisation techniques

A.Demetriou. [5] surveyed diverse sensors and techniques regarding UGV's localisation. According to the author, the use of a single technique e.g. computer vision is not commonly employed for addressing navigation problems. Instead, there is a growing interest among mobile robot developers in combining computer vision with LiDARs or ultrasonic sensors for accurate distance estimation. This approach is employed for localisation as well which is the fundamental of navigation and path planning. This study demonstrates that GPS as an absolute localisation technique, is one standard solution for addressing localisation among outdoor applications UGVs. However, it is utilised in combination with other relative localisation techniques such as wheel odometry and/or inertial navigation. Wheel odometry uses encoders, while inertial navigation employs gyroscopes and accelerometers which are fused and preprocessed in an IMU.

Alkendi et al. [12] surveyed state-of-the-art vision-based localisation techniques to enable the autonomous navigation of unmanned vehicles. According to the authors, fusing multiple sensors and sensor types for localisation increases the system's reliability, robustness as well as fail tolerance. The drawback of sensor fusion, however, is high computational complexity. The result of this survey proves that the visual inertial odometry (VIO) method in combination with filtering/ optimisation such as Kalman filtering methods has good accuracy in a GNSS-denied/ challenging environment. Vision in VIO refers to different visual sensors including camera, LiDAR, and radar which are evaluated and discussed in this survey. While inertial refers to the data acquired by an IMU. According to this survey, camera/ LiDAR inertial odometry is a widely used method to address unmanned vehicles' localisation in urban environments.

Liu et al. [6] "A Review of Collaborative Air-Ground Robots Research" surveyed collaborative UAV/ UGV from different aspects such as application, function, sensors, et cetera. The study showed that a set of sensors including wheel odometry, IMU, camera, LiDAR, and GPS can be used to address localisation and collision avoidance mechanisms for a ground robot exploring a complex and dynamic outdoor environment.

Gao et al. [7] proposed employing LiDARs for implementation of both localisation and mapping for UGV. According to this study, LiDARs offer various advantages, they have small size, high overall precision, ability to measure and estimate the distance, location, and shape of the obstacles. Moreover, LiDARs, have better performance in different light conditions. However, in this study, they utilised cameras as well, to acquire more accurate information for depth [7]. To address localisation, this study employed two GPS antennas, and an IMU. The autonomous navigation of this study has been tested for a dynamic outdoor environment and the results of the study, have revealed how the sensors combination can effectively address the problems above.

Kumar et.al [8] utilised a set of sensors including a stereo camera in combination with LiDARs, GPS, and heading sensors for real-time path planning and collision avoidance mechanism. According to Kumar et

al., LiDARs have good precision and are easier to work in real-time, however, to prevent misinterpretation of objects, stereo cameras are employed as well. The obstacle avoidance mechanism implemented in this study is designed for working in real-time and in a dynamic outdoor environment which is aligned with the vision in the STAD project. According to the authors in [8], utilising the camera as the only object detection source is not an effective approach due to the long time needed for data acquisition and processing, making it difficult to work in real-time. Hence, data from cameras and LiDAR are fused for the implementation of real-time path planning and collision avoidance.

Spector et al. [11] proposed a framework for simulating an autonomous Husky robot in a military environment. The robot was equipped with both cameras and LiDARs where ROS was utilised for controlling the robot and accessing data from the sensors. They proposed using semantic segmentation on their image data to classify the different types of terrains in the environment and also to classify different objects in the environment. The results of the semantic segmentation are used to find the general position of the different objects. To find the more exact position of different objects object detection is also performed with the image data as well as the LiDAR data. Their work proves how cameras and LiDARs can be used to localise different objects around a UGV.

3.2.2 Sensors efficiency in various conditions

Sezgin et al. [10] performed a sensor evaluation in adverse weather conditions for autonomous driving. Their research evaluated LiDAR, radar, and cameras in dry, foggy, and rainy weather conditions. A comparison was also made in how well the sensors performed between day and night conditions. The research concluded that radars are in general robust in challenging environments. However, LiDARs and cameras are poorly affected by harsh weather conditions, even in light rain. The worst degradation of these sensors is in dense fog conditions. Their research also concluded that cameras are highly affected by poor lighting conditions.

A survey was performed by Giuffrida et al. in [9] investigating the difference between radars and LiDARs for autonomous driving systems. The survey concluded that LiDAR is more appropriate for short-range and high-resolution scenarios, for example, a city scenario. While radar is more appropriate for detecting long-range, fast-moving objects, for example, a highway scenario. The survey also highlights challenges for the utilisation of radars on an autonomous platform since several antennas are required to enable the same angular resolution as for a LiDAR. On the contrary, employing LiDAR is associated with high costs, and its effective range is constrained due to eye safety regulations.

3.2.3 NVIDIA Drive PX2 Applications

In article [67] researchers utilised an NVIDIA Drive PX2 platform to construct an integrated framework by combining the lane detection, object detection and free space detection frameworks on the PX2. The integrated framework combined the outputs of the three deep neural networks to give a complete representation of what could be detected in the image. In their testing they mounted a GMSL camera behind the windshield of a car, the camera captured video with a resolution of 1920x1208 at a frame rate of 30 Frames Per Second (FPS). They found that with their setup they could run the PX2 at 19 Hz when they captured the video and rendered the video with all detected objects, lanes and free space. This shows that the NVIDIA Drive PX2 is a viable option for utilising in autonomous vehicles.

In another article [68] researchers investigate strategies for reducing time used in CNN computations on the PX2. The time reduction is of utmost importance when applying it to real-time systems. In their work they looked at distributing workload between the Central Processing Unit (CPU)s and GPUs and could with that increase throughput twofold with no loss in accuracy. Another method they investigated was to compare the performance of the CNNs when looking at 4 images separately or a 2x2 composition with all the images. They found that with the composition of images, throughput increased fourfold but at the cost of a 10% accuracy loss.

3.3 Integration

A substantial body of work has been done on the topic of integrating multi-robot systems such as UGVs and UAVs to cooperate and perform various tasks. The works of [69, 70, 71, 72, 73] are particularly

relevant in this context, as they all have worked on UGV-UAV systems with some form of ROS-integration, similar to this project. The works in [69] and [70] will be covered in further detail in Sections 3.3.1 and 3.3.2, respectively. Other works, more related to tethered UGV-UAV solutions will be covered in Section 3.3.3.

3.3.1 UGV and UAV cooperation in GNSS-challenged environments

Xu et al. [69] present a UGV-UAV solution that is equipped with vision sensors and IMUs while communicating throughout the multi-robot system with ROS. The goal of their work is to produce a framework that can jointly build a three-dimensional point map to determine the relative position of the UGV and UAV to each other. Determining the position in this way is done with the motivation that in some places, a Global Navigation Satellite System (GNSS) signal may not be available, thus other measures need to be taken when having a multi-robot system that has to synchronize their movements. However, both systems are equipped with GNSS receivers, which can further aid the accuracy of determining relative (as well as global) positions whenever the systems can receive a GNSS signal. Equipping the system with sensors such as vision, IMU, and GPS, is similar to how the system is equipped in this project. However, the UAV in this project is solely responsible for determining its relative position to the UGV via vision tracking. The solution in this project requires less complexity but may result in lower accuracy because fewer sensors are used to determine a relative position.

To enable communication between the two systems in [69], ROS is used over a wireless network. Although the authors do not specify the network type, it is assumed that Wi-Fi is employed because the systems are not tethered, and ROS natively supports Ethernet or Wi-Fi. In comparison to the work in this project which uses Bluetooth between the UGV and UAV instead of Wi-Fi, Bluetooth may offer more robust communication due to its use of a frequency hopping scheme [74, 75].

The main conclusion presented in [69] is that in a virtual simulation, the system managed to maintain an adequate positioning accuracy over hundreds of meters without too much-introduced drift. However, an intermittent GNSS signal was still needed in the end to realign the positions to counteract position estimation drift. In comparison to a single-platform independent positioning, their cooperating system reduced the error of co-location by 15.5% and 19.7% respectively.

3.3.2 Remote Planning and Operation of a UGV Through ROS

There have been attempts to create frameworks for wireless robots since ROS became a generally recognized standard for robotics. In [70], ROS 2 is employed as the communication platform to control the UAV-UGV disaster robot compound. They use Internet of Robotic Things (IoRT) architecture which utilizes ROS 2 over 4G and 5G Ethernet networks to enhance robot-human wireless interaction autonomously. ROS 2 nodes and topics were used to transfer sensory data and images between UAV, UGV, and a ground station that monitors the entire system as well as sends control commands to the system. In this paper, all the communications among two cell towers and between the UAV-UGV system are done through ROS over Ethernet which utilises ROSLink. Furthermore, there are two LAN networks in their cloud which assess the possibility of using ROS and its communication capability over different networks. They incorporated ROS with Ubuntu 18.04 installed on an Intel NUC8i7BEH minicomputer which receives GPS sensory data over a Universal Serial Bus (USB) port and is connected to the network via a 5G Ethernet connection.

The primary parallels lie in the utilization of ROS 2 over Ethernet, a similarity shared with this project. However, it's noteworthy that in this project an Ethernet link is employed instead of wireless communication. While they opted for ROSLink, the STAD project utilized MAVLink which is developed under DDS. Both projects involve controlling the UAV-UGV from a central command centre through ROS commands, in other words, teleoperation from a remote location. Additionally, both studies involve the use of ROS on Ubuntu 18 and the collection of GPS data on a companion computer, highlighting a complete resemblance between the two endeavours.

The [70] study has shown that the proposed architecture utilising 4G and 5G ethernet network may be used to teleoperation a UGV-UAV, either as a backup plan or in place of a nearby control centre.

3.3.3 Tether-based UGV and UAV systems

Several works exist that contain exploration of tethered configurations between a UAV and UGV, which present unique challenges and opportunities. Borgese et al. [15] focuses on the relative localisation of a UGV and a multi-copter connected by a tether in both indoor and outdoor environments, leveraging the catenary configuration of the tether. The approach is based on the idea that the tether forms a catenary curve while the aerial vehicle hovers. The tether angles are then sensed at both ends to determine the relative locations of the two vehicles. The authors suggest that this type of solution is computationally light and low-cost. In their work, an accuracy of 22 cm was obtained. Martinez-Rozas et al. [16] introduce a novel trajectory planning methodology for a marsupial robotic system, comprising a UAV linked to a UGV via a non-taut tether, addressing synchronisation, collision avoidance, and dynamic constraints in both simulated and real-world scenarios. In the study, the authors emphasise that the tether itself is seldom taken into consideration in collision avoidance algorithms. This oversight is one of the primary focuses of the authors. Martinez-Rosas et al. propose a path-planning method based on Rapidly Exploring Random Trees (RRT) that incorporates all three components, UAV, UGV, and tether when determining movement strategies. Their results show that despite being a complex problem of planning paths for a variable length tether it is feasible to provide solutions in a matter of seconds with a 100% success rate. Both Borgese et al. and Martinez-Rosas et al. motivate a tethered system by citing the limited operational time of UAVs. However, neither work addresses the challenges associated with designing a lightweight tether. The key challenge in the STAD project is to create a tether that does not compromise the flight dynamics of the UAV while still providing power and, potentially, communication capabilities.

4. Problem Formulation

This project aims to build an initial research platform for an autonomous delivery system that enables students and researchers at MDU, in cooperation with various companies, to learn and hopefully get new insights into the field of robotics and embedded systems. A part of the future vision of Finnslätten, an industrial area in Västerås, is to become a dense technology centre fostering inspiration and innovation among companies, researchers, and students. Such a transformation, however, will restrict access to public roads. Consequently, innovative solutions must be made to facilitate the transportation of goods between different buildings. One idea is to utilise autonomous robots. Such autonomous robots require complex systems to navigate dynamic environments shared with humans. Therefore, the autonomous robot must employ advanced technology using a combination of sensors, actuators, software, and hardware to be safe and trustworthy.

The core hypothesis is that a tethered UGV platform with a tracking UAV can be developed to facilitate future research in autonomous vehicles with path planning, navigation, and collision avoidance.

This student project marks the initial phase in a series of iterations aimed at advancing the UGV and UAV platforms. At this stage, the objective is to build the foundation of the research platform with ongoing development and enhancements planned for the coming years. A complete autonomous system is therefore out of scope during this project. This includes more advanced topics such as path planning, dynamic collision avoidance, autonomous UAV takeoff and landing, etc. It is also assumed that the Clearpath Husky UGV and the Nvidia Drive PX2 computation node which are given to the project are in functional order and may have some development restrictions. Consequently, the following Research Questions (RQ's) are considered for the platform's initial design within the project time scope.

- RQ1: What measures can be taken to design a system that supports various sensor types and facilitates future scalability?
- RQ2: How does a tether affect the overall flight performance of the UAV?
- RQ3: Can a novel visual image processing method, utilising a depth camera for autonomous tracking and alignment UGV be implemented in the UAV?

Regarding RQ1, different sensors that are required for autonomous navigation will be explored. The underlying platform requirement, which facilitates the sensors, needs to be taken into consideration. This involves careful consideration of factors such as sensor interfaces, communication, power requirements, processing power, and optimal sensor placement. As an example, it will be explored how ROS can be utilised in an environment where different firmware versions and OS's are distributed in different nodes in the system. This question aims to understand how to build a scalable multi-sensor platform that can be used for autonomous robots.

Typically, a UAV is equipped with an onboard battery and is operated freely in the air. However, for reasons such as longer UAV operational time and UAV legislation, a tethered UAV solution has been chosen for this project. This begs the question of how this tethered solution affects the UAV flight performance. The purpose of RQ2 is to understand how the tether can be implemented, and how it affects variables that make the UAV fly in a stable manner, which consequentially affects autonomous flight.

RQ3 aims to investigate how the tethered UAV can be autonomously controlled using computer vision. To enable safe and reliable control of the UAV, it should accurately and within a timely fashion position itself to always follow the UGV. To achieve this control, the flight controller which consists of IMU's that controls the flight behaviour, requires translated input from the computer vision algorithm to adjust variables such as horizontal position, rotation, altitude, velocity, etc. Additional data from the camera such as depth, or external sensor data from a GPS may also be useful to explore.

5. Method

This section outlines the methodological framework used, providing insight into the methods, procedures, and the utilisation of hardware and software tools respectively. It provides a cohesive approach to address the RQ's given in Section 4..

5.1 Methodology

In addressing RQ1, a combination of research methods has been applied. To design a system accommodating various sensor types and ensuring scalability, a focused literature review was conducted. This review aimed to understand the integration of the Clearpath Robotics Husky system with the Nvidia Drive PX2 and explore methods for equipping the UGV with diverse, suitable sensors. The literature review enabled methods for equipping the UGV with diverse, suitable sensors. Subsequently, the implementation of these sensors into the system was undertaken for testing, to evaluate their functionality and compatibility.

To tackle RQ2, a subset of tests was conducted involving UAV flights with and without a tether to collect data in flight logs. Comparisons were made through the analysis of these logs. While simulation could be an alternative method, creating a simulation comparable to the real-world test was deemed impractical. To the team's knowledge, there are no software tools that facilitate quick and efficient testing of flight dynamics of a tethered UAV.

To address RQ3, a software implementation was developed and tested for accuracy in detecting the desired object. The software autonomously determines alignment and calculates the off-centre position within the camera frame. Simulations were also conducted to facilitate the development of software capable of autonomously flying the UAV. Given the potential risks, performing these tests in a real-world scenario without extensive prior testing was deemed unsafe.

5.2 Procedures

The STAD project primarily aimed to integrate a pre-built UAV with a UGV, addressing challenges such as path planning, image processing, object detection, and collision avoidance. However, as UAV systems were non-functional and the UGV system had some development restrictions, both scope and goals had to be altered. The aim was changed to provide a functional research platform for the upcoming iterations of the project. As a result, most of the tasks became focused on manufacturing the UAV and equipping the UGV. Early in the project, the focus was on performing a thorough evaluation of the equipment that had been provided. The evaluation required careful analysis to determine the purpose of each provided physical component. The objective was to determine and sort out the useful equipment that fit the project goals. Furthermore, steps had to be taken to ensure whether the equipment fit with the project requirements or had to be updated. Additionally, pre-installed software components were also examined to ensure compatibility with the chosen hardware and project requirements.

The UGV system was equipped with a wide range of sensors and devices to ensure that it could be used as a fully featured research platform in the future. The sensors comprised eight cameras, two LiDAR, and a full-featured IMU with an external GPS module. Other necessary sensors also pre-existed on the UGV, including two motor encoders. This combination of sensors assists the UGV with a strong and versatile sensory framework that improves its capabilities for a range of use cases. Two LiDARs were employed to achieve a comprehensive view, covering both the front and rear sides of the UGV. This configuration is supposed to enhance the reliability of the object detection mechanism. Additionally, eight cameras were integrated to ensure a complete 360° coverage of the UGV's surroundings. Although nearby objects could not be detected clearly with cameras. This multi-camera setup was used to facilitate the identification of moving objects, considering the limited view angle of each camera. The IMU is utilised to establish the relational positioning between the UAV and UGV.

The UGV was originally shipped with an internal computation node which is a Mini ITX computer. This device helps to interpret ROS commands and sends them to the UGV inner control unit. The Mini ITX acts as the main communication distribution node. For this project, the UGV was also custom-equipped with an extra powerful computational unit, the Nvidia Drive PX2. The PX2 is a GPU system which enables further equipping of sensors, it also comes with an array of interfaces such as FlexRay, CAN, Local Interconnect

Network (LIN), and Universal asynchronous receiver-transmitter (UART). Furthermore, the PX2 can aid the UGV platform in the computation of demanding algorithms such as those required to process vision input from multiple cameras. The decision to refrain from utilising the PX2 for the tasks which the Mini ITX performed, was rooted in compatibility issues between the PX2 device and ROS versions. The UGV has also received a custom-built loading area, enabling it to securely load cargo. Additionally, the UGV was fitted with a landing pad for the UAV. The landing pad was placed at a height of 1.5 m above the ground, to not interfere with any payload placed on the UGV.

The UAV unit was completely designed, built and equipped from the ground up. The reason for opting to construct a custom-made UAV rather than purchasing a pre-assembled one was to facilitate any customisation needed for the project, such as peripheral devices, strong motors for more lifting capacity, etc. To enable autonomous flight coupled with tracking, and alignment to the UGV, the UAV was equipped with an Oak-D Pro depth camera. The depth camera was placed below the UAV, with its Point of View (POV) facing the ground. The goal of the depth camera was to detect the UGV via an algorithm and facilitate data extraction for alignment, depth, and horizontal relations between the systems. The extracted data could then be interpreted and used to move the UAV accordingly. To facilitate the UAV with autonomous flight capabilities it was fitted with a PixHawk 6X as the FC and a Raspberry Pi 4 acting as a companion computer. The companion computer was responsible for communicating with the UGV and FC, as well as processing vision input from the depth camera. Additionally, the UAV was equipped with a Ricoh Theta X 360° camera that was directly connected to the UGV platform, further providing the UGV with vision input.

The component that physically connects the UAV to the UGV is a tether system. The tether system consists of a rope for tethering the UAV to the UGV, power cables to power the UAV from the UGV, and a USB cable to provide the 360° camera live stream to the UGV. The tether rope ensures that the movement of the UAV is restricted. The power cables are connected to a battery on the UGV to feed the UAV and enable longer mission duration in contrast to having an onboard battery. These power cables are connected to the Power Distribution Board (PDB) on the UAV to distribute the electricity to all the residing components on the UAV. The Universal Serial Bus (USB) cable was connected from the Mini ITX computer on the UGV to the companion computer on the UAV to enable the 360° camera video live stream. In the end, all the cables were patched together by a strong and lightweight cable shield to protect them from tension and deterioration.

To enable a communication link between UAV and UGV, a custom-made communication message delivery software was developed. This communication software was developed using Bluetooth to send and receive commands between UAV and UGV. Bluetooth was chosen to decrease the amount of cabling between the two systems, thus reducing the weight the UAV has to carry. Furthermore, Bluetooth is more energy-efficient than alternatives such as Wi-Fi, and provides robust communication with the use of adaptive frequency hopping [75]. The software was also developed to communicate via Ethernet sockets between the PX2 and Mini ITX. The socket program message structure was designed in a generic way so it can handle different types of messages that are required in the system. The communication between nodes on the UAV and UGV was done by ROS to achieve synchronization of the messages on both systems. Due to compatibility issues in the Linux versions of the Mini ITX and PX2, ROS was not utilised for this purpose during the project. On the UAV, ROS 2 is employed to transmit commands between the FC and companion computer to achieve autonomous flight. On the UGV side, ROS 1 is utilised between the Mini ITX computer and the UGV control unit to move the UGV.

5.3 Tools

In this section, the tools used during the project are outlined.

5.3.1 Ultimaker 2+

Plastic parts for the UAV were printed using Ultimaker 2+[8]. The build volume of this printer is 223 x 220 x 205 mm and is compatible with filament with a diameter of 2.85 mm. This printer has a high precision when printing thin layers and is widely used by the industry.

[8] https://ultimaker.com/3d-printers/s-series/ultimaker-2-connect/

5.3.2 Original Prusa i3 MK3S

Plastic parts for the UGV were printed using Original Prusa i3 MK3S [9]. The printer has a build volume of 250 x 210 x 210 mm and is designed to use filament with a diameter of 1.75 mm. It can be used with a wide range of thermoplastics including Polyactic Acid (PLA), Polyethylene terephthalate glycol (PETG) and Acrylonitrile butadiene styrene (ABS).

5.3.3 Ultimaker Cura

Ultimaker Cura is an open-source, easy-to-use 3D printing software for slicing and printing 3D parts. The software tool offers several settings to optimize the printing process for the user's needs. Ultimaker Cura has been used in this project to export g-code for the Ultimaker 2+ printer.

5.3.4 PrusaSlicer

PrusaSlicer[10] is an open-source tool for exporting g-code to 3D printers and is specially tested for Original Prusa printers. In this project, it has been used to export g-code to the Original Prusa i3 MK3S printer. The tool has a clear and simple GUI and supports both print time and feature analysis.

5.3.5 Visual Studio Code

Visual Studio Code (VS Code)[11] is a modular, free, and open-source Integrated Development Environment (IDE) which allows users to install extensions from its marketplace. Furthermore, it is platform-independent and supports hundreds of programming languages, contributing to its widespread popularity. This tool was used mainly to write different software components. Via the Secure Shell Protocol (SSH) extension in VS Code, the team could also directly program and build applications on the project's devices.

5.3.6 SolidWorks

SolidWorks is a leading computer-aided design (CAD) software developed by Dassault Systèmes. It specialises in creating precise 3D models of mechanical parts and assemblies. The application supports parametric modelling and simulation capabilities. SolidWorks is widely used in engineering for designing and testing products before physical prototyping. Solidworks played a crucial role during this project in designing all parts, both for the UGV and UAV.

5.3.7 Oracle VM VirtualBox

Oracle VM VirtualBox[12] is a free and open-source virtualisation tool for type 2 hypervisors. Type 2 hypervisors typically negotiate with the OS to borrow hardware resources to virtualise other operating systems. In this project, a great deal of virtualisation was required to set up development platforms with the correct Linux distributions. Virtualisation was done to ensure compatibility and functionality according to requirements from different hardware components.

5.3.8 QGroundControl

QGroundControl[13] is an open-source software used for different unmanned robot vehicles such as UAV's. It is compatible with Windows, OS X, Linux, Android, and iOS platforms. It provides flight control, mission planning, real-time telemetry, vehicle setup configuration and calibration. QGroundControl is designed to use the PX4 flight stack but can also use other software such as ArduPilot. QGroundControl was used in this project to configure and calibrate the UAV.

[9] https://www.prusa3d.com/product/original-prusa-i3-mk3s-3d-printer-3/
[10] https://www.prusa3d.com/page/prusaslicer_424/
[11] https://code.visualstudio.com/
[12] https://www.virtualbox.org/
[13] http://qgroundcontrol.com/

5.3.9 Gazebo

Gazebo[14] is an open-source 3D simulator used to simulate robotic applications. Gazebo can simulate sensors, physics, and different robot models. Furthermore, Gazebo can be integrated with ROS, which makes it a highly useful tool for this project. Gazebo was used to simulate autonomous control of the UAV before attempting any type of control in a real-world scenario.

[14] https://gazebosim.org/about

6. Ethical and Societal Considerations

There are many ethical considerations for an autonomous delivery system, one apparent issue would be privacy where the UGV and UAV both will be capturing data in the form of video, etc. Data should not be recorded without consent of individuals in the vicinity of the UGV. The exception is if a data set is needed for simulation and training, then it should be created using people who have given consent to be recorded and make sure others are not included.

Safety is a crucial aspect of the project. The system needs to enable several safety features before entering public roads. For example, the UAV should be equipped with protocols that address power loss or communication breakdown scenarios. In such situations, the UAV must be capable of executing a safe landing to prevent potential harm or collisions that could endanger individuals or property. Additionally, the entire system is required to stop on demand, therefore it should have an emergency stop button that can be used if it is spiralling out of control. The system should also incorporate collision detection and perform a stop before the collision. Since the project in its current iteration is a prototype, all testing should be done in controlled environments until the prototype is classified as safe.

Since the aim for the system is to be autonomous in the future, security will be of utmost importance, communications should be encrypted and it should not be possible to send commands to the vehicle from unauthorised individuals. If this is not fulfilled, the person breaking into the system could come across sensitive information or even send commands that would make the vehicle dangerous.

Additionally, the UGV and UAV systems must ensure they follow the laws and regulations where the system operates. Currently, the regulations in Sweden state that autonomous vehicles are only allowed in trial applications. To conduct a trial, an application needs to be approved by Transportstyrelsen [76]. With the technology moving forward, the laws and regulations will change so the project needs to stay up to date with the advancements. For UAVs in Sweden, specific regulations need to be accounted for. In the testing stages where the UAV will be piloted manually, the pilot should have a drone license unless it is tethered or tested indoors [37].

7. Implementation

This section outlines the implementation of the system. Section 7.1 presents the design and construction of the UAV together with its software implementation. Section 7.2 describes the design and construction of additional equipment to the UGV such as such as the landing pad, loading area and sensors. Furthermore, the section describes how the sensors were implemented in the software. Section 7.3 focuses on the data communication throughout the system and presents the implementation of the different data communication protocols. Finally, in Section 7.4, an overview is given of how the sub-systems were connected via a bundled tether package.

7.1 UAV construction

This section initially describes how the UAV's hardware and software were set up and configured in Sections 7.1.1 and 7.1.2. Note that all the configuration of the UAV software, for initial flight via RC, is performed through the QGroundControls GUI as described by the documentation of PX4[15] and QGroundControl[16]. Additional in-depth guides can be found in the GitHub repository[17] regarding the build, configuration, and how to operate the vehicle. Due to the existing documentation, the first two sections highlight the configuration and any specific choices made for the UAV built during the project. Additionally, some of the software configuration steps might need to be repeated continuously, e.g., sensor re-calibration due to external interference.

After describing the initial build of the UAV, the following sections are related to making the UAV autonomous. Namely, how the landing pad targets were designed in Section 7.1.3. Sections 7.1.4, 7.1.5, 7.1.6, and 7.1.7 describe how the computer vision was implemented to control the UAV. More precisely, how the object detection for said targets was implemented, the position and orientation between the UAV and the targets, the software implementation in the companion computer, and a GUI for UAV command and control. Finally, Section 7.1.8 describes the 360° camera located on the UAV.

7.1.1 UAV hardware and design

Table 1 displays the hardware components obtained from various sources for the construction of the UAV. To understand how the electronic components were connected, see Figure 3. The UAV's frame was custom-designed specifically for the project using SolidWorks by a third party[18]. The design was continuously updated in iterations, to test different solutions and ideas, based on the observations gathered whilst building and operating the UAV. Once all the necessary components were collected, their dimensions measured, individual weights recorded, and the UAV requirements specified, a final design was 3D printed, as depicted in Figures 4 and 5.

[15] https://docs.px4.io/main/en/
[16] https://docs.qgroundcontrol.com/master/en/qgc-user-guide/index.html
[17] https://github.com/MDU-C2/Shuttle/tree/main/documentation/guides/UAV
[18] Bengt Gustafsson, Lecturer in IDT, Division of Product Realisation

Component	Model
FC	Pixhawk 6X
PDB	PM03D Power Module
Companion Computer	Raspberry Pi 4 Model B, 4 GB RAM
GPS module	M9N GPS Module
Motors	T-Motor MS2820-7 830 KV
Propellers	APC 12x4.5 Multirotor (both directions)
Radio receiver	RadioMaster R81
RC	Radiomaster TX12 Mark II
Battery	LiPo Battery 4 S 14.8 V 11000 mA, Gens Ace
Frame	Custom design and 3D print
Camera 1	OAK-D Pro
Camera 2	Ricoh Theta X
ESC	EMAX BLHeli 20 A 2-4 S OneShot125
	Aikon AK32 35 A 2-4 S BlHeli32
	X500 V2-BLHeli S 20 A
Other	4 pin JST-GH to USB-C cable
	6 pin CLIK-Mate to 6 Pin JST-GH
	4 pin JST-GH to RJ45 Ethernet cable
	5 m USB-C to USB-C cable
	USB-C to USB-A cables and adapter
	EC5 male/female connectors
	5 m 10 AWG power cable
	Cable sock
	Rope
	Extra XT30, XT60 and JST connectors
	Various soldering hardware, cables, tubes, etc.
	Various screws, nuts, bolts, cable ties, etc.

Table 1: The UAV hardware components.

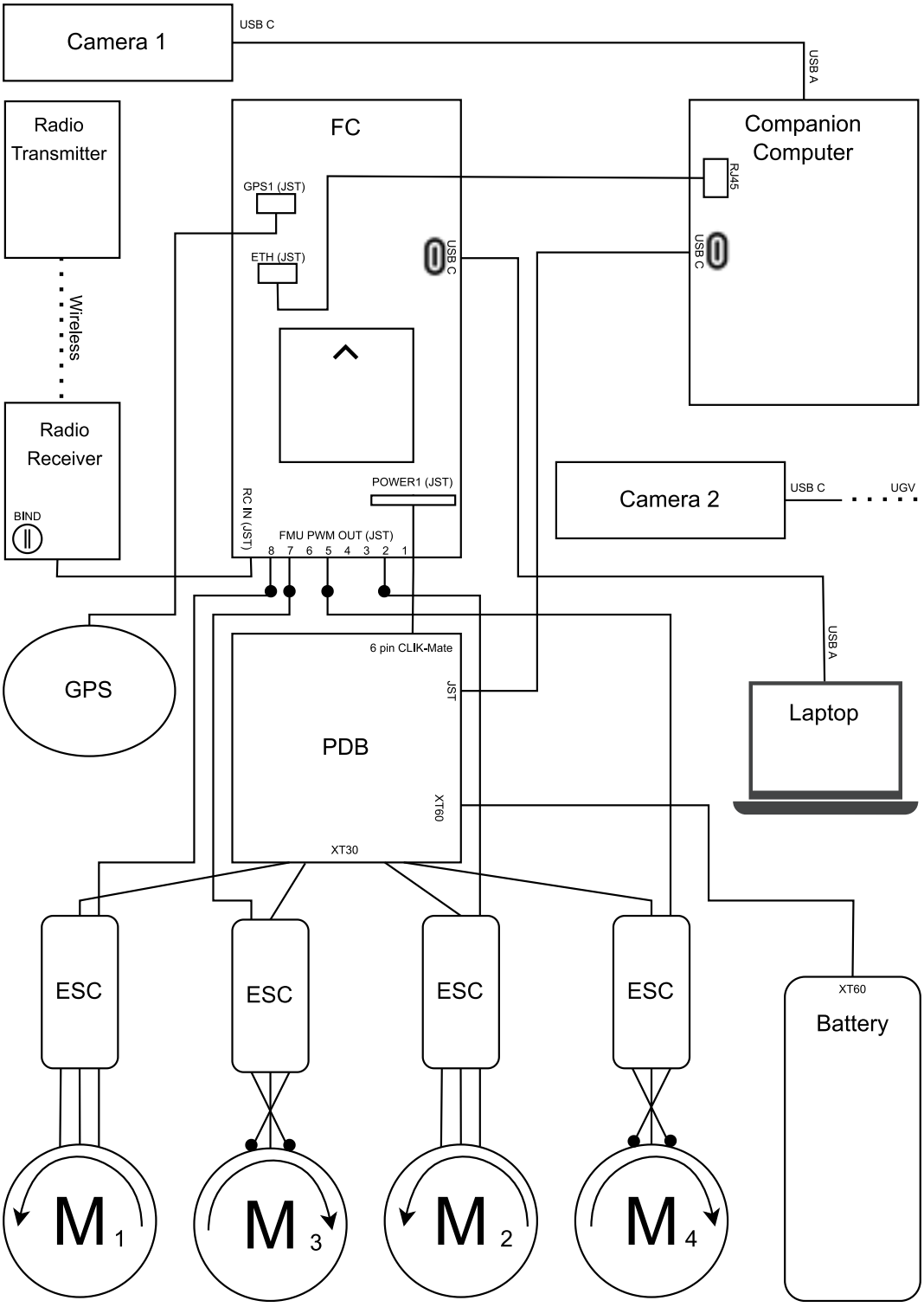
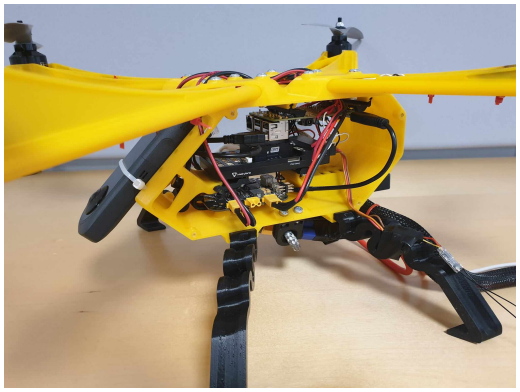


Figure 3: The schematics of the UAV, representing how each component was connected and the type of connector. The black dots indicate that cables were soldered. Note that Camera 2 was housed in the UAV frame but connected to the UGV and not to any UAV component.

Figure 4: The design of the drone as seen in SolidWorks.



(a) UAV with the connected components housed in the frame.

(b) UAV with the custom walls attached.

Figure 5: The 3D printed UAV design.

7.1.2 UAV software and calibration for initial flight

After connecting all the hardware components, the latest version 4.3.0 of QGroundControl was installed on a computer connected via USB cable to the FC to access and configure the system as illustrated in Figure 6. The first UAV configuration step was to flash the FC and install the latest stable version 1.14.0 of the PX4 firmware. Due to the design being custom, the airframe was chosen as a generic quadcopter, which most closely resembles the frame geometry and number of motors.

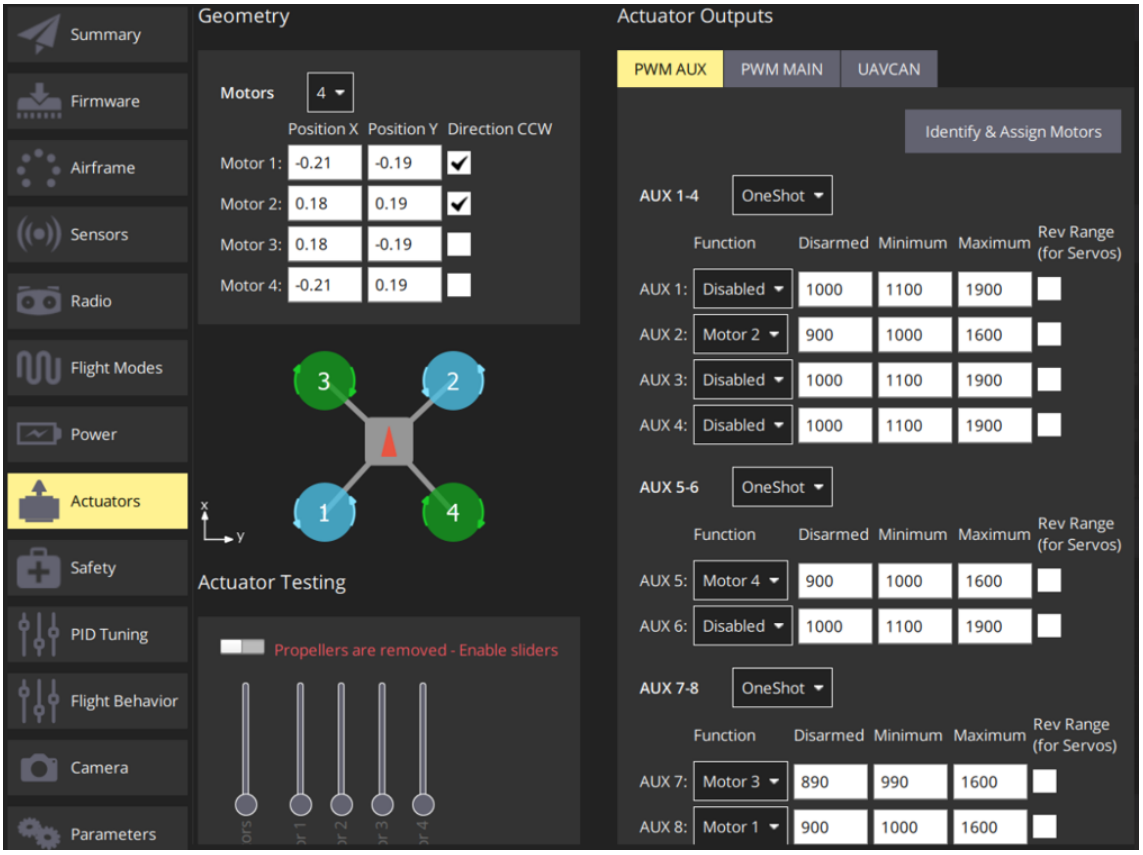


Figure 6: A partial snapshot of the GUI of QGroundControl in the Actuator section where the latest settings are seen.

The setup and calibration of the sensors; the magnetometer, gyroscope, and accelerometer were configured as instructed by the QGroundControl software, which involves rotating and holding the vehicle in different positions. For the UAV to have a correct understanding of its position related to the horizon the "Level Horizon" calibration was also performed. Since the FC was mounted upright facing the front, no adjustment was necessary for the FC orientation. Furthermore, the sensors are prone to interference and require recalibration if the UAV experiences abnormal behaviour such as drift and altitude jumps. QGroundControl will also fail its pre-flight check if a sensor requires recalibration before arming the vehicle. Examples of such interference include the magnetometer, which is affected by large metal objects in its vicinity. Another example is that the horizon might have to be levelled if the takeoff surface is not level.

To pair the RC transmitter and receiver, the antenna and batteries were first attached to the RC and the receiver was connected to the FC. For detailed guides on how to pair, refer to the hardware product guides *Transmitter Quick Start* and *Receiver Manual* in the project repository[19]. During this procedure, the chosen communication protocol was FrSky D8, due to the EU region and the receiver supporting a maximum of 8 channels. The result of the pairing was verified via a solid red Light-Emitting Diode (LED) on the receiver, indicating a successful pairing. A fail-safe was also set, to prevent an accidental receiver button press disconnecting the pairing and RSSI fine-tuning was also performed as described in the Receiver Manual. In QGroundControl, the "Mode" was set to 2, meaning that the left gimbal is used for throttle/yaw and the right for roll/pitch, which is standard for right-handed people. The radio calibration procedure was then performed as instructed by the GUI of QGroundControl, resulting in each movement on the handheld RC being tied to a certain channel. Finally, various tests were performed by adjusting the minimal and maximal output of the RC, e.g., limiting the maximal throttle or attitude controls. These limits were manually configured on the RC by testing various weights and offsets as described in the hardware product guide *Transmitter User Manual* under section 4.4.4 in the previously mentioned repository.

[19] https://github.com/MDU-C2/Shuttle/tree/main/documentation/guides/UAV

In the flight mode section of QGroundControl, different flight modes and behaviours were associated with an available communication channel tied to different RC switches. Due to the UAV not utilising a GPS, the only viable flight modes that were possible to use were Manual mode and Altitude mode. These are manually controlled flight modes, where the pilot operating the RC controls the vehicle as described in [33]. These flight modes were configured on channel eight, associated with a switch on the RC, allowing the vehicle to switch modes by flipping the said switch. Additionally, a vehicle arm switch was configured on channel six, an emergency kill switch on channel five, and an experimental offboard flight mode switch on channel seven.

Regarding the power setup section in QGroundControl, ESC calibration was performed, and parameters were set to reflect the LiPo battery in use. Each time the ESC's were changed, the ESC calibration had to be performed. This process involves disconnecting and reconnecting the battery and pressing the calibration button in the GUI in QGroundControl for an automated calibration process to take place. The battery parameters were set for the 4-cell battery, with a full voltage per cell of 4.2 and an empty voltage per cell of 3.2. The battery parameters are set so the system roughly understands the battery level, e.g., to prevent arming the vehicle if the level is critically low or causing some landing procedure.

In the actuator section of the QGroundControl GUI, it is possible to fine-tune the vehicle geometry in addition to calibrating and testing the motors. All the values that were set in this section can be seen in Figure 6. For the vehicle geometry, an estimation was taken of the X and Y positions of the vehicle COG when all components were mounted inside the UAV frame. Using the COG, a location marked on the roof of the vehicle, measurements were taken from the COG to the four motors to set their X and Y position in QGroundControl as described in [30]. The actuator output was configured based on how the cables were connected from the ESC to the serial AUX ports on the FC. In the settings, AUX2 was assigned to motor two, AUX5 to motor four, AUX7 to motor three, and AUX8 to motor one. The ESC protocol was set based on the ESC's used such as OneShot. Manual tuning was performed to find the proper duty cycle values for the disarmed, minimum, and maximum motor states. The chosen values were set so that the disarmed state caused no actuation, the minimum value so that the RPM was equal for all motors given the same current, and the maximum was set to the lowest possible value to avoid overheating. The RPM was roughly estimated by motor sound and visual inspection. The direction of the motor spin should be equal on the diagonal side of each motor arm. In case the motor spin direction was not correct, cutting the soldering cables connecting the ESC and the motor in a different order was required as illustrated in Figures 7 and 3.

Figure 7: Cables 1, 2, and 3 are connected so that the motor spins in a counterclockwise direction. To change the direction, cut cables 1 and 3 and switch their position on the motor side through soldering.

Finally, the PID tuning was manually configured in QGroundControl to stabilise the flight performance. The stability of the flight is checked whether the vehicle drifts in some direction or measured in how many oscillations the vehicle stabilizes after performing various manoeuvres such as directional rolls. Depending on the behaviour, the PID controllers were manually tuned experimentally. The final PID configuration from the tuning can be seen in Table 2. The Rate Controller parameters ending with K are controller gain, an overall multiplier scaling the P, I, and D values. Each time the vehicle was armed, a binary log file (.ulg) was created and saved on the vehicle's file system. These log files were extracted and analysed in online tools[20] to troubleshoot issues, such as comparing the roll setpoint to the response values. Due to the UAV not being equipped with any telemetry communication hardware, it was required to use a USB cable from the FC to a computer running QGroundControl to view the graphs of the PID controllers in real-time.

[20] https://review.px4.io/

Rate Controller

Angle	*Parameter*	*Value*
Roll	MC_ROLLRATE_K	0.55
	MC_ROLLRATE_I	0.0098
	MC_ROLLRATE_D	0.475
Pitch	MC_PITCHRATE_K	0.55
	MC_PITCHRATE_I	0.0084
	MC_PITCHRATE_D	0.5
Yaw	MC_YAWRATE_K	0.8
	MC_YAWRATE_I	0.0074
	MC_YAWRATE_D	0.38

Attitude Controller

Angle Parameter	*Value*
MC_ROLL_P	2.5
MC_PITCH_P	2
MC_YAW_P	1.7

Table 2: The settings of the Rate and Attitude Controller's various parameters.

7.1.3 Design of landing targets

After it was decided that the UAV would follow a target on top of the UGV landing pad, the next step was the design of the targets. The original idea was to have a circle with a cross in the middle. This can provide information such as the offset horizontally and vertically, and the depth camera can give the distance. The distance could help calculate the offset as it will change depending on height as a result of perspective. The problem, however, is that this single pattern provides no way of knowing the relative angle of the UAV to the landing target unless some sophisticated vision processing is utilised. This is a sub-ideal solution as a Raspberry Pi 4 was used as a companion computer, which limits the processing capabilities.

Instead, the chosen approach was to have two targets. The object detection model detects two targets and the angle between them can be calculated relative to the orientation of the camera. The landing pad targets were chosen to be as simple and clean as possible to maximise detection performance, but also as different as possible to minimize the risk of confusing them. The colours were selected as blue and red, one had a circle in the centre, and one had a cross. Both were circles as other shapes might give different bounding boxes depending on the viewing angle. Other shapes would also prevent the depth from being calculated, using the algorithm explained in Section 7.1.5, because the sizes would change depending on the viewing angle.

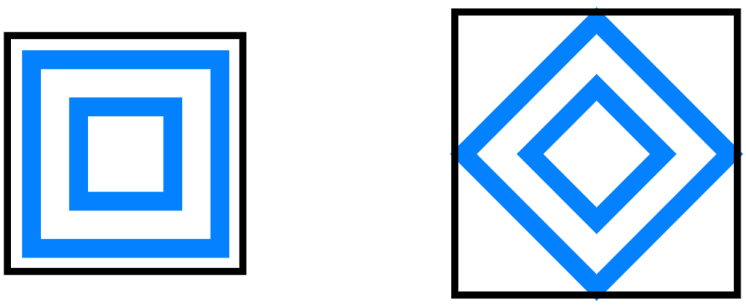


Figure 8: This figure illustrates how the size of the bounding box in black changes depending on the viewing angle if the object is not circular.

7.1.4 Object detection model

Roboflow was used for creating the ML model that detects the targets. The main benefit of Roboflow is the ease of use which saves time. The dataset was created by taking pictures of the landing targets with the OAK-D camera at different distances and viewing angles. These images were uploaded to Roboflow's website[21] and the targets were labeled red and blue accordingly. In Roboflow, there are tools available that simplify the process of labeling images. The bounding boxes were created by dragging a square around the targets within the pictures. When all images are labeled, the next step is to create a version. A version is a concept within Roboflow where additional processing is applied to the original dataset, which allows it to be used for training. This also results in a larger dataset because the original images are kept. The image augmentations used were brightness adjustments from -50 to +50%, blur up to 5.5 pixels, and noise on up to 5% of all pixels. The distortions were kept relatively low to prevent the data in the images from getting destroyed completely. The brightness adjustments were used because they mimicked different light settings well. In a version, the images can be rotated and skewed as well, but this tends to misplace the bounding boxes which is undesirable. The object detection model was trained on a version with the mentioned augmentations and the most important feature is that Roboflow supports Python and OAK-D Pro cameras, both of which were used in this project.

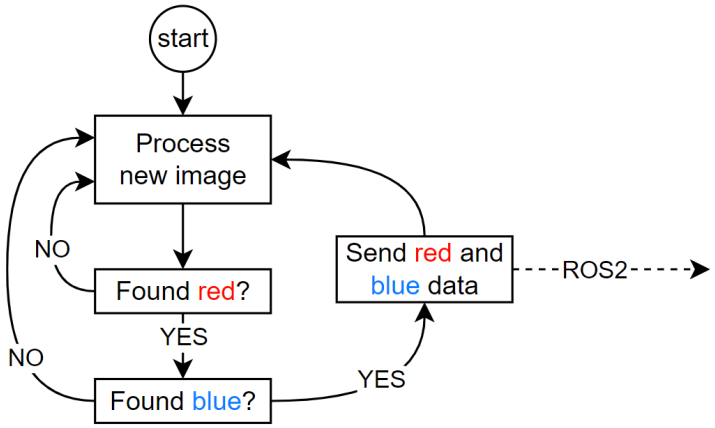


Figure 9: The object detection loop.

After the model had been trained it was imported into Python using the Roboflowoak[22] library. The model was imported using a confidence of 0.8, meaning that objects are considered found if the prediction confidence is above 80%. Figure 9 illustrates how the object detection loop is handled. Whenever a new image is seen, the Roboflow ML model processes it. The image is only valid if both labels (blue, and red) are visible, as one label is insufficient data. New images are processed repeatedly until both objects are detected. When they are, they get sent via ROS 2 to another node for processing. The data at this stage contains raw information, meaning x, y, $width$, and $height$ for the bounding boxes of both the red and blue targets.

There are several reasons why $width$ and $height$ are used instead of the depth value that can be directly obtained from the depth camera. First, the detection performance is significantly worse when depth data is captured as well. The time between consecutive detection frames is magnitudes longer, and an autonomous algorithm would be harder to develop as more interpolation is required between frames. Secondly, the data is highly distorted, approaching an unusable level. The UAV has to fly at an altitude of 5 meters steadily. At this distance from the ground, the depth values reported by the camera fluctuate by multiple meters, making fine-tuned movement adjustments difficult.

[21] https://roboflow.com/
[22] https://docs.roboflow.com/deploy/luxonis-oak

7.1.5 Pose estimation

The data at this point consists of arbitrary pixel positions and sizes of the targets. The pose estimation algorithm has three main purposes. The first is to extract the horizontal and vertical offsets from the targets in meters. The second purpose is to calculate the flight height of the UAV above the targets in meters. The final purpose is to calculate the angle between the heading directions of the UAV and the UGV. As mentioned, there are two targets but the UAV should follow one position and calculate one height value. Therefore, in the following equations, both targets are combined into one target. The following variables are used, where x is horisontal position, y is vertical position, w is the target width, and h is the target height.

$$x = \frac{(x_{red} - 320) + (x_{blue} - 320)}{2} \tag{4}$$

$$y = \frac{(y_{red} - 320) + (y_{blue} - 320)}{2} \tag{5}$$

$$size = max(\frac{w_{red} + w_{blue}}{2}, \frac{h_{red} + h_{blue}}{2}) \tag{6}$$

After image processing, the image resolution is 640 * 640, hence in Equation 4 and Equation 5, 320 is subtracted from each target to recenter origin around zero, and the averages are taken. Otherwise the coordinates $(0, 0)$ would be top left and $(640, 640)$ would be bottom right. In Equation 6 the average width and height are calculated, and the maximum is used as the size for the target.

Figure 10 shows that the target's width and height are identical when viewed from above in a). When viewed from a different angle as illustrated in b) and c), one axis gets distorted while the other is unchanged, hence the reason for using the maximum value. More sophisticated algorithms can be used for greater perspective compensation, but assuming the autonomous control system for the UAV is working as intended, these situations will never occur. Additionally, by just using rectangular bounding boxes, information is lacking for better perspective compensation. More detail would be required such as the lengths of the sides of the right bounding box in Figure 11. Additionally, in Figure 11, the left bounding box is what the object detector currently provides.

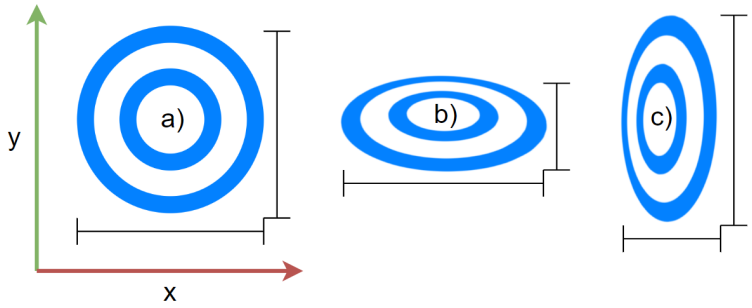


Figure 10: Landing pad target and the width/height ratio at different perspectives.

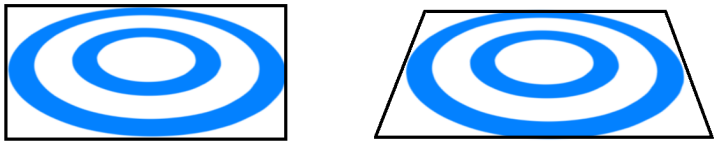


Figure 11: The difference between rectangular bounding boxes and true perspective.

After the target size is calculated, it is used for calculating the depth using weak perspective [77]. It is a common technique in computer graphics that is essentially stated in equations 7 and 8. They represent the conversion from world coordinates (subscripted with w), to screen space coordinates (subscripted with s). As illustrated, dividing by the world z coordinate i.e. the distance in world space, and multiplying by the focal length, the conversion is made. Naturally, this also applies to the size of an object which Equation 9 illustrates.

$$x_s = \frac{f x_w}{z_w} \tag{7}$$

$$y_s = \frac{f y_w}{z_w} \tag{8}$$

$$s_s = \frac{f s_w}{z_w} \tag{9}$$

After the image classification, the locations of the objects on the screen, are known. In this scenario, the world space coordinates are unknown, hence, inverting these equations becomes the solution. While the world x and y coordinates are unknown, the world target size is not. The distance z_w can therefore be calculated with Equation 10 where s_s is $size$ from Equation 6, s_w is 0.187, and f is 850. The real-world size of the targets used is 0.187 meters, and if new targets are printed in a different size, this value must be changed accordingly. As for focal length, f, it can normally be calculated using camera parameters, but they are unknown in this case. The current focal length was selected through testing, and utilising a value of 850 gives the correct answers to the equations. When z_w has been calculated, the x_w and y_w offsets are calculated using Equations 11 and 12, as demonstrated below.

$$z_w = \frac{f s_w}{s_s} \tag{10}$$

$$x_w = \frac{x_s z_w}{f} \tag{11}$$

$$y_w = \frac{y_s z_w}{f} \tag{12}$$

The angle of interest is the angular difference of the UAV's forward direction vector and the vector from the red target to the blue target. As a result of how the camera is mounted, the forward direction is positive x in screen coordinates shown in Figure 12.

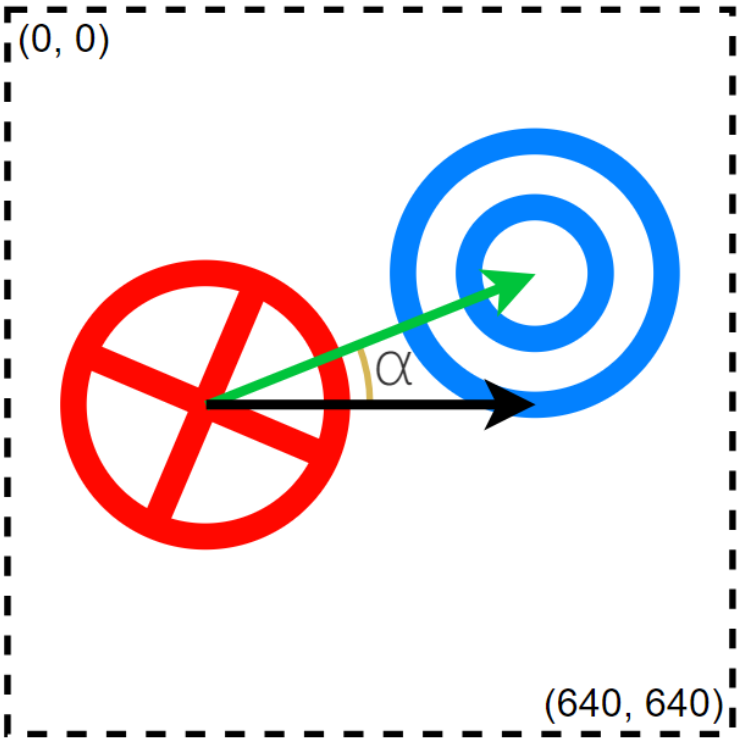


Figure 12: Angular relationship between the UAV and the landing pad targets. The green arrow is the UGV heading vector, the black arrow is the UAV heading vector, and α is the angular difference between them.

Utilising the angular relationship between vectors when performing the dot product, the angle α is calculated using the following equations where the vectors a and b are the forward vector and target vector respectively.

$$\begin{aligned} \vec{a} \cdot \vec{b} &= \|\vec{a}\|\|\vec{b}\| \cos(\alpha) \Rightarrow \\ \cos(\alpha) &= \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|} \Rightarrow \\ \alpha &= \arccos\left(\frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\|\|\vec{b}\|}\right) \end{aligned} \tag{13}$$

The value of α can never be negative due to the nature of $\arccos$. Whether α is positive or negative can be obtained by the cross product of $\vec{a}$ and $\vec{b}$ as they share the same sign. Equation 14 illustrates this logic. α_{old} corresponds to α obtained in Equation 13.

$$\alpha_{new} = \begin{cases} \alpha_{old} & \text{if } \vec{a} \times \vec{b} >= 0 \\ -\alpha_{old} & \text{if } \vec{a} \times \vec{b} < 0 \end{cases} \tag{14}$$

At this point, x_w contains the horizontal target offset in meters, y_w contains the vertical target offset in meters, z_w contains the distance to the target in meters, and α contains the angle difference between the target and the UAV in radians. This is the basic foundation, of which the autonomous flight algorithm is to be applied.

7.1.6 Companion computer software design

The entire companion computer system controlling the UAV, is divided into three ROS 2 nodes. They are named *vision_node*, *uav_main*, and *uav_gui* where the first two are essential, while *uav_gui* is optional and

only intended for development purposes as visualisation greatly helps.

These nodes exist within a ROS 2 workspace where vision_node is implemented in Python and the other two are implemented in C++. The system can be seen in Figure 13 below.

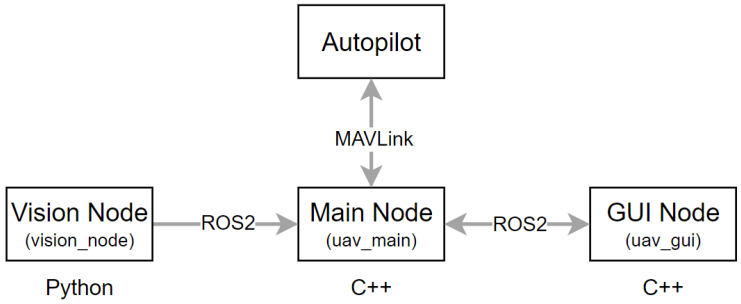


Figure 13: The ROS2 system. ROS2 is used for communication between nodes, while MAVLink is used for communicating with the autopilot.

The vision node contains the detection loop presented in Figure 9. This node was designed to be as light-weight as possible as Python is an inherently slow language, and everything UAV related is processed on a Raspberry Pi. Calculations are performed in C++ whenever possible, which is the reason why the vision node does not estimate the pose. It only detects the objects and sends their data forward to the main node.

The main node handles drone-related tasks as well as the pose estimation algorithm. For communication with the autopilot, MAVSDK was used which in turn uses MAVLink as its communication protocol. The main node starts by initialising ROS 2. The node is created and then all required publishers and subscribers are created. This involves one subscriber for receiving target information from the vision node, and both subscribers and publishers for communication with the GUI node. These messages are explained in depth, in section 7.1.7.

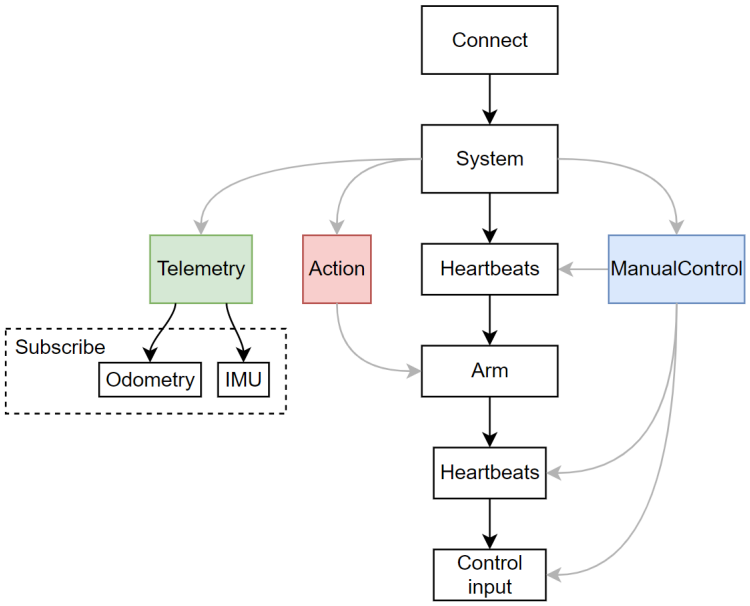


Figure 14: Initialization sequence for main_node.

When ROS 2 is initialised, the MAVSDK code is initialised. An overview of this process is shown in Figure 14. It starts by connecting to the autopilot on port 14550 with IP address 192.168.0.4 as this is

the configured settings for Ethernet. If the connection is successful, the UAV system "handle" is retrieved which is a datatype that gives access to all messages and commands that can be used to manipulate the autopilot. The relevant messages used so far are presented in the following list, and the grey arrows in Figure 14 illustrate what functionalities they are used for.

1. **Action**: Allows actions such as arm, disarm, land, and takeoff to be sent.
2. **Telemetry**: All the available information that the flight controller broadcasts. These contain the local and global coordinates, IMU, odometry, etc. Essentially, everything regarding the vehicle poses in space.
3. **ManualControl**: Gives access to the UAV control system allowing control commands to be sent. These contain position control, which allows coordinates or velocity to be specified. Greater control is given through altitude control, allowing the raw throttle, pitch, roll, and yaw values to be specified. This essentially overrides the radio controller.

To arm the vehicle from a companion computer, a heartbeat signal must have been sent repeatedly for at least one second. The autopilot treats a control input message as a heartbeat and expects at least two beats every second to consider the other device alive. Therefore, before arming, a manual control input command is sent repeatedly with all parameters set to zero, every 20 milliseconds for the duration of one second. At this stage only arming is desired, which is why zero-values are sent. The UAV is then armed by an action command, and the heart beating continues after for one second as well, to give a time margin to the arming command. If the arming succeeds, the vehicle startup is complete.

From telemetry, the main node is currently subscribed to the IMU and the Odometry topics. From IMU, the acceleration and angular velocity is obtained. For velocity, the odometry provides the local velocity obtained from internal sensors. Local position data is also included in the odometry topic. If the vehicle moves by 50 cm in some direction, the position data is updated to reflect this change. After the update, the data will slowly reset and drift towards zero as this data is kept relative to recent historical changes. This makes it difficult to utilise position in calculations.

This telemetry information was correctly subscribed when simulating in Gazebo. In practice, when connected to the real autopilot, no data is received on any telemetry channel. Instead, the same topics can be subscribed to via ROS 2 through the PX4 ROS 2 message interface, which retrieves the desired information.

The last node is the GUI node which communicates with the main node. The main node is a console application, which makes user interaction especially difficult to implement. When waiting for user input, the console is blocked, preventing any printing in the console. The GUI node was created to allow user input, but also keep it separate from the main node. As a result of running Ubuntu server 20.04 on the Raspberry Pi, no graphics are available, which would break the GUI code if it was written in the same application.

7.1.7 GUI node

The GUI node is implemented using Graphics Library Framework (GLFW)[23] and Immediate Mode Graphic User Interface (IMGUI)[24]. GLFW as the name suggests, is a framework for graphics that allows access to a window context and other related OS specific features such as keyboard and mouse input, rendering surfaces, and system time. Together with this framework, Open Graphics Library (OpenGL)[25] is used which is a specification for what graphics functionalities should exist within a GPU. OpenGL cannot render by itself but needs access to a window, which in this case is provided by GLFW.

IMGUI is a C++ library that makes GUI development simple. This library has support for many rendering backends, OpenGL being one of them. The result of this is that no programming was done using OpenGL directly, only through IMGUI which simplifies the process significantly. This is why a GUI was even considered. The GUI node is currently split up into three windows. The first is the controller window which is a virtual controller, allowing direct control of the drone. The layout is shown in Figure 15

[23] https://github.com/glfw/glfw
[24] https://github.com/ocornut/imgui
[25] https://www.opengl.org/

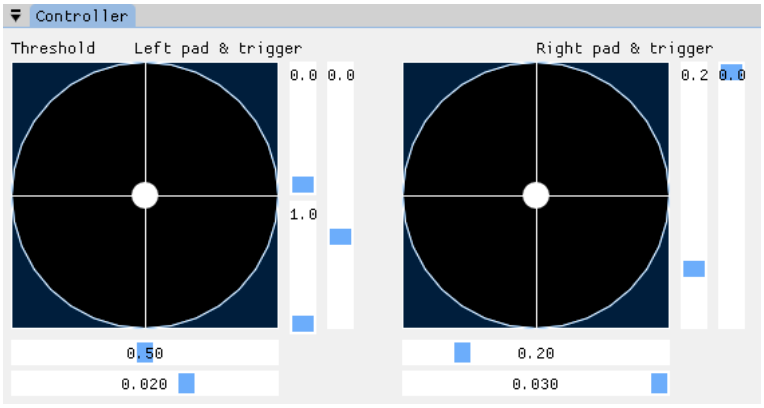


Figure 15: GUI controller window.

This virtual controller is modelled to resemble the RC and functions similarly. The left cross-hair is for yaw and throttle while the right cross-hair is for pitch and roll. These virtual joysticks are controlled with keyboard buttons. The throttle is increased or decreased with the W and S keys. Adjusting the yaw left or right is done through the A and D keys. The left and right arrow keys change roll and the up and down arrow keys change the pitch. When using a real controller, it is easy to maintain the same value by just holding the same position. The complex behavior that a real RC provides, is difficult to replicate with keys and trade-offs have to be made. For yaw, pitch, and roll, the value goes back to zero at a constant rate as soon as a key is released. For throttle, the value keeps its last value when a key is released, mimicking the real controller more closely.

The sliders in Figure 15 limit certain control aspects. The inner sliders closest to the cross-hairs, limit the possible positions of the dot in the centre. For throttle, two sliders are used limiting the minimum and maximum values separately as greater control is desired here. Depending on the weight of the UAV, the power of the motors, etc, different configurations might be of interest. The outer sliders adjust the speed at which the value changes when a key is held, yielding greater control over how the different controls behave.

Figure 16 shows the kinematics of the UAV, for both angular and linear quantities. The three columns display x, y, and z respectively for each of the quantities. When the main node receives this information from the autopilot, it directly sends it to the GUI node which displays it here.

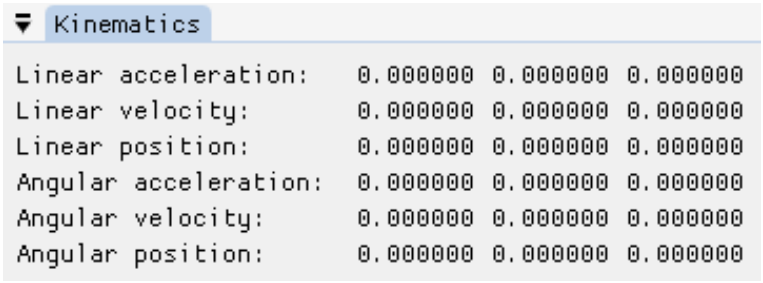


Figure 16: GUI kinematics window.

The last window is the command window shown in Figure 17. This is where commands can be sent by clicking buttons. As of now, the only commands are arm and disarm. It takes some time for arming to happen so until a response is received from the main node it will display that it is waiting as shown in the image. This is useful as commands sometimes fail. For example, arming will fail if the throttle is too high, and disarming will fail if the UAV is currently flying. The main node also receives a heartbeat from the autopilot every second which is sent to the GUI node and displayed accordingly.

Figure 17: GUI command window.

Below is a list of all currently implemented ROS 2 messages that are used in communication between the main node and the GUI node. All messages contain a timestamp parameter specifying when they were sent including additional parameters listed below.

1. **Arm**: Sent from the GUI node whenever the "Arm UAV" button is pressed in the command window.
2. **ArmAck**: Sent from the main node whenever an **Arm** message is received. It contains a boolean **success** value that is *true* if the arming succeeded, otherwise *false*.
3. **Disarm**: Sent from the GUI node whenever the "Disarm UAV" button is pressed in the command window.
4. **DisarmAck**: Sent from the main node whenever a **Disarm** message is received. It contains a boolean **success** value that is *true* if the disarming succeeded, otherwise *false*.
5. **ArmedHeartbeat**: Sent from the main node when it receives a heartbeat from the flight controller which happens once a second. The message contains a boolean **armed** value where *true* is armed and *false* is disarmed.
6. **ControlInput**: Sent from the GUI node with the virtual controller values in the controller window. It contains four 32-bit float values corresponding to throttle, yaw, pitch, and roll. This data is sent once every frame, i.e., 60 times per second.
7. **ControlMode**: Sent from the GUI node when the control mode is changed. As shown in Figures 15 and 17, this value currently cannot be specified. The message contains a 32-bit integer value where 1 is manual control and 2 is autonomous control. Currently, only manual control is used.
8. **Kinematics**: Sent from the main node whenever this data is received from the autopilot. This message contains six 32-bit float arrays with three values each. The indices 0, 1, and 2 correspond to x, y, and z respectively for each of the quantities mentioned.

7.1.8 360° camera stream

To provide a bird's-eye view, a 360° camera was placed on the UAV facing forward. When placed on the UAV, the camera can acquire a wide Field of View (FOV), covering most views except the one behind its body. Refer to Figure 18 for a representation of the camera placement and its FOV. Since this camera can only live stream in a 4K resolution, it requires a substantial amount of resources. When the live stream data rate was measured over USB, it required a 35 Mbit/s bandwidth. Without significant downscaling, this data stream could not be processed with computer vision algorithms on the UAV's companion computer. To alleviate the strain on the existing wireless connection and the companion computer while keeping the image resolution high, the stream was instead sent via the camera's native USB C-interface directly to the via a 5-meter USB cable supporting USB 2.0 which theoretically can transfer up to 480 Mbit/s [78] UGV.

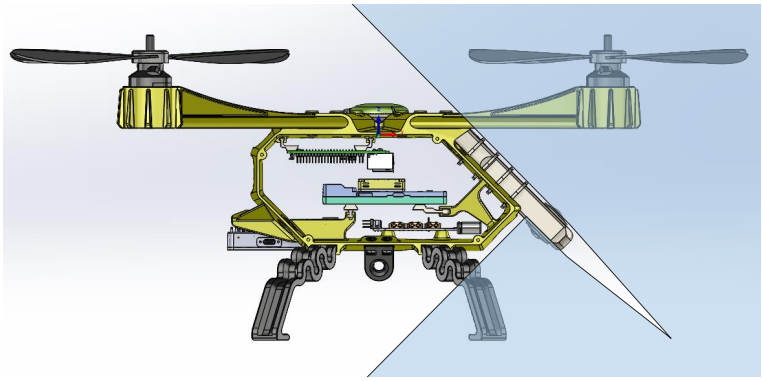


Figure 18: UAV 360 degree camera placement and FOV.

To facilitate live streaming to a Linux-based system, Ricoh offers a live streaming USB Video Class (UVC) library[26]. While the library is not guaranteed to work on all Linux distributions, it has been tested and verified to function reliably on Ubuntu 20.04 and higher. However, compatibility issues may arise, especially considering that the Nvidia Drive PX2 operates on Ubuntu 16.04. Due to this disparity, a decision was made to send the live stream to the Mini ITX computer running Ubuntu 20.04 mitigating any potential compatibility issues. In addition to the operating system considerations, the camera needs several GStreamer-related packages to operate effectively. Furthermore, after installing all the dependencies, modification of the Ricoh UVC "**thetauvc.c**" file was required. Specifically, a definition for the Theta X camera had to be established. For a comprehensive installation guide, refer to the project's Github repository[27].

To start the camera in USB live stream mode and send a stream to the UGV, the camera has to be connected to the Mini ITX computer via USB-C and toggled into live stream mode via the *mode* button on the camera. If the viewer program is built correctly and the computer has a GUI, a live stream can be observed simply by running the **/gst_viewer** command.

7.2 UGV construction

This section outlines how the UGV was built in terms of placing sensors as well as designing and constructing additional equipment. The section also describes how different sensors were implemented in the software.

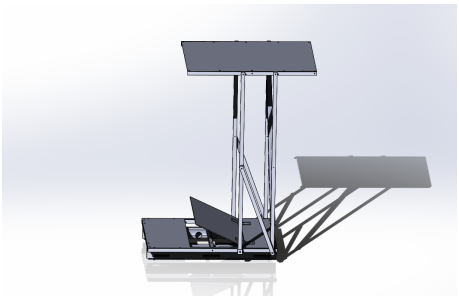
7.2.1 Design and Construction of Loading Area and Landing Pad

Since the UGV is aimed to be able to perform deliveries of goods a Loading area was designed and constructed throughout the project. In addition, a landing pad was designed and constructed to enable a place where the UAV can take off and land. The design and construction were made in collaboration with the association Makers of Västerås[28]. The loading area and landing pad were designed with the requirements of the project in mind. The design was made using SolidWorks and is presented in Figure 19a.

A requirement for the project was that the UGV would be able to carry a payload of 40 kg. Aluminium was decided as the material for the loading area due to its high strength-to-weight ratio. The loading area mainly consists of an aluminium plate 600 x 700 x 3 mm. Underneath the loading area, several support beams were placed to ensure stability and robustness.

Another requirement was that the UGV would be able to access data from all its sensors. This requirement needed to be taken into consideration so that all sensor cables could run between the sensor and its computer and that the constructed loading area or its supporting beams did not obstruct the cable routing. Therefore several holes were drilled through the supporting beams in which the cables could be routed. Usability was also taken into consideration when designing the loading area by placing a lid on top of the aluminium plate. By opening the lid all the hardware components inside the UGV can be accessed.

[26] https://github.com/ricohapi/libuvc-theta/tree/theta_uvc
[27] https://github.com/MDU-C2/Shuttle/tree/main/documentation/guides/UAV
[28] https://makersvasteras.se/



(a) Design of the Loading Area and Landing Pad

(b) The constructed Loading Area and Landing Pad

Figure 19: Design and construction of Loading Area and Landing Pad.

The landing pad was decided to be placed one meter above the loading area. The landing pad consists of an aluminium plate with dimensions of 600 x 700 x 2 mm and is designed similarly to a table with four supporting beams. Between the supporting beams, stays are also placed to ensure a fixed position for the whole landing pad construction.

On top of the landing pad, decals were placed which the UAV can use for the object detection model to follow the UGV. The constructed loading area and landing pad are presented in Figure 19b.

7.2.2 Sensor Selection

Sensors were selected to enable the UGV for autonomous navigation in a GNSS-denied environment. The sensors were chosen based on a comprehensive literature review including [5, 6, 9, 10, 12] as well as through examining compatible hardware with either the Mini ITX or PX2. Table 3 lists the selected sensors.

Component	Model	Number
Camera	AR0231, OnSemi	8
GPS Antenna	VN-200 Tallysman 2712, VectorNav	1
IMU	VN-200, VectorNav	1
LiDAR1	RPLiDAR A2, Slamtec	1
LiDAR2	LMS111, Sick	1

Table 3: The employed UGV sensors.

In a dynamic GNSS-denied/ challenging environment, IMU is a solution for addressing localisation of an UGV. However, the IMU data are noisy and unreliable over a long time due to drift. A solution to overcome this problem is utilising cameras in combination with IMU which is called visual inertial odometry technique. The disadvantage of this method, however, is the high computational complexity and poor performance of cameras in harsh weather and low light scenarios. LiDARs can be a good asset for adding redundancy to the system as well as working in low-light and adverse weather scenarios except for harsh weather conditions. A GPS is a standard solution for acquiring absolute localisation information of an UGV which works well in combination with IMU and encoders for addressing UGV's localisation in general outdoor applications.

7.2.3 Component Placement

Where to place different components was carefully considered in the early stage of the project. Component placement includes the placement of sensors, computers, batteries, and all other electronic components. The placement locations were discussed throughout several meetings within the group and with supervisors. The locations of each component are presented in the following table:

Component	Position
Cameras	See Figure 20
LiDARs	See Figure 20
IMU	User storage
Nvidia Drive PX2	User Storage
Battery Husky robot	Back storage
Battery Nvidia Drive PX2	User storage
Battery UAV	User storage

Table 4: Location of components.

The cameras and LiDARs were positioned to maximize their field of view for exteroceptive sensing. Note that near the vehicle the view is not 100% covered.

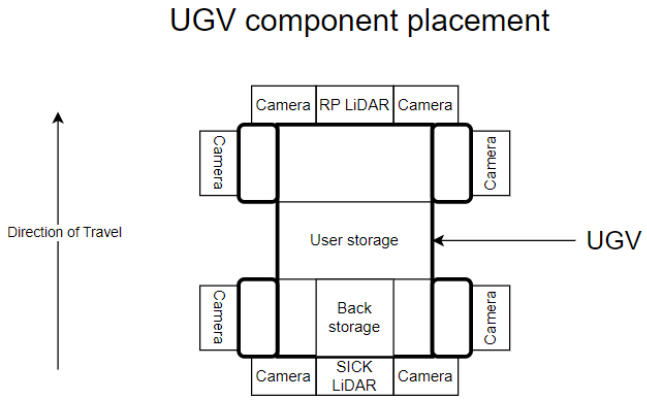
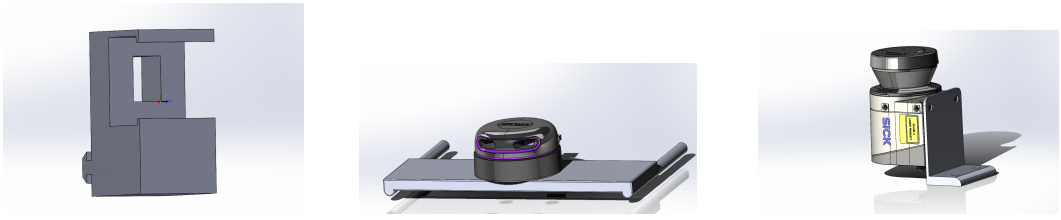


Figure 20: UGV component placement.

Once the locations for all components were decided the design of different component mounts was commenced. All component placements were designed using SolidWorks and made in PLA plastic. In Figure 21 the mounts for sensors are presented.



(a) Mount for camera (b) Mount for Slamtec LiDAR (c) Mount for SICK LiDAR

Figure 21: Mounts for sensors used in the project

7.2.4 Power Management

Due to differences in input voltage levels of the UGV and PX2, as well as the absence of an appropriate embedded peripheral socket on the UGV panel, a separate power system is utilised to power supply the PX2. Various hardware components used in the power circuit of the PX2 can be found in Table 5.

Component	Model
Battery	25Ah VRLA AGM 12V deep cycle, T15
Connector	Housing Receptacle, 10- Poles, 2- Rows, Molex
Control Panel	Custom design and 3D print
Diode	TVS 28.2V, 13.3A, P6KE33A
Fuse	Vehicle fuse 80 A 58V
Fuse Holder	MegaOTO Fuse Holder, iMaXX
Power button	Toggle switch 2-states, 12 V, 50 A
Terminal Block	Universal
Voltage level indicator	12 V DC
Wire Red and black	Cable 18AWG D1.1/2.4mm x 1m
Wire Red and black	Pre-Crimped Lead Mini-Fit Jr Female

Table 5: PX2 power system hardware components.

For more technical information and how the hardware components are connected in the circuit refer to the project repository[29]. By designing the power system, a custom control panel was designed and 3D printed to ease the access to the power button as well as monitor the charge of the battery assigned to the PX2 which is shown in Figure 22.

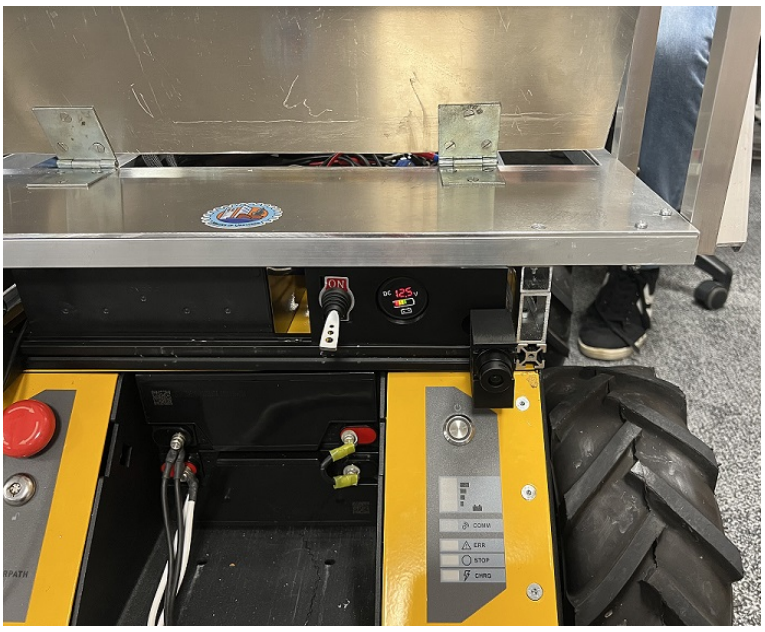


Figure 22: PX2 Power Supply Panel.

Other peripherals including Cameras, IMU, and RPLiDAR assigned to the UGV, will be power supplied by connecting the devices to the either Mini ITX computer or PX2. However, the Sick LiDAR, uses a separate cable for the power supply which is designed to be connected to the 24V, 5A embedded peripheral socket on the UGV. More technical information on the Sick LiDAR's power cable could be found in the project git repository [30].

[29] https://github.com/MDU-C2/Shuttle/tree/main/documentation/guides/UGV
[30] https://github.com/MDU-C2/Shuttle/tree/main/documentation/guides/UGV

7.2.5 Wireless Controller

The Clearpath Robotics Husky UGV comes with pre-installed software on the Mini ITX computer. This software is plug-and-play with the wired Logitech F310 controller provided in this project. The pre-installed ROS package enables the moving of the UGV. To interface and control the UGV via a wireless controller, a wireless PlayStation 4 (PS4) controller was used together with a Bluetooth dongle connected to the UGV. Additionally, the Ubuntu BlueZ package had to be installed on the Mini ITX computer to interface with the PS4 controller. For a more detailed guide, refer to the project repository.

The pairing of the PS4 controller was done in the following steps on the Mini ITX.

1. Open the Bluetooth control interface: **bluetoothctl**
2. Enable the agent: **agent on**
3. Start scanning for devices: **scan on**

Initiate pairing mode on the PS4 controller.

4. Stop scanning: **scan off**
5. Pair with the PS4 controller using its MAC Address: **pair <MAC Address>**
6. Trust the connected device: **trust <MAC Address>**
7. Establish a connection: **connect <MAC Address>**

7.2.6 LiDAR

LiDAR sensors provide accurate and real-time distance measurements of surrounding objects. This information is essential for detecting obstacles in the robot's path and also can be used to build a map for navigation. In this project, the integration of the RPLidar into the UGV system introduces an object detection functionality.

The development of specialised Lidar software optimises the utilisation of the RPLidar sensor that can access the transmitted data. This software was developed as a prototype to serve as a guide for data acquisition from the sensor. The software starts its operation by initialising essential variables, establishing the LIDAR connection, retrieving device information, and configuring the motor speed to ensure optimal sensor performance. Subsequently, the software transitions into a continuous loop, capturing scan data, processing it, and printing the information. The software is responsive to user input, actively monitoring the Ctrl+C signal to exit the loop when required. Upon the completion of the scanning loop, it halts the LIDAR operation, performs a pause to allow for the finalization of any ongoing processes, and reduces the motor speed to zero. Finally, the cleanup procedure executes to release resources. This program should include SLAMTEC's LIDAR SDK header files as sl_lidar.h and sl_lidar_driver.h to be able to interface with the sensor's functionalities.

7.2.7 IMU software

The UGV system was equipped with the VN200 sensor from VectorNAV. One purpose of employing IMU sensors in both UAV and UGV systems is to enhance navigation and control precision. The IMU can provide accurate information about the vehicle's position, velocity, and orientation. Both the IMU on the UGV and the IMU on the UAV play a pivotal role in establishing and maintaining relative positioning between the two systems. It facilitates coordination in the same direction for both systems. To achieve this, the IMU data from both systems should be processed and integrated on the same device.

However, since the UAV in this system follows the UGV, it is reasonable to perform the calculations on the UAV computational node, which serves as the companion computer. As discussed in Section 7.3, the ITX device within the UGV sends the IMU data. To read data from the VectorNAV VN200 sensor, the Software Development Kit (SDK) library, VectorNav Programming Library (VNPL), was utilized in the C++ language. The function form_imu_data is implemented to gather yaw, pitch, and roll parameters from the sensor. This algorithm allows the UAV to adjust its position and orientation dynamically relative to the UGV.

7.2.8 Cameras

To enable computer vision and object detection eight GMSL cameras with 60° FOV have been strategically placed around the base of the loading area. The placement of the cameras is demonstrated in figure 20, the strategic placement ensures vision in all directions around the UGV. This enables monitoring and detection of objects and obstacles in all directions. The cameras were connected to the NVIDIA Drive PX2 platform where the video stream is processed. The PX2 platform will in the future be utilised for computer vision with its robust object detection algorithms.

7.3 Communication

This section presents the flow of data communication, details on types of communication in various locations, and the implementation of functionalities.

Refer to Figure 23 for a high-level overview of data communication throughout the system. The figure illustrates a Bluetooth connection between the UGV and UAV. Each sub-system utilises two devices communicating via wired Ethernet. On the UGV, Ethernet communication is achieved through a customised socket messaging system, while on the UAV, it primarily involves MavLink messages. In the Mini ITX, ROS 1 facilitates inter-communication between applications. Similarly, on the Raspberry Pi 4B on the UAV, ROS 2 serves the same purpose.

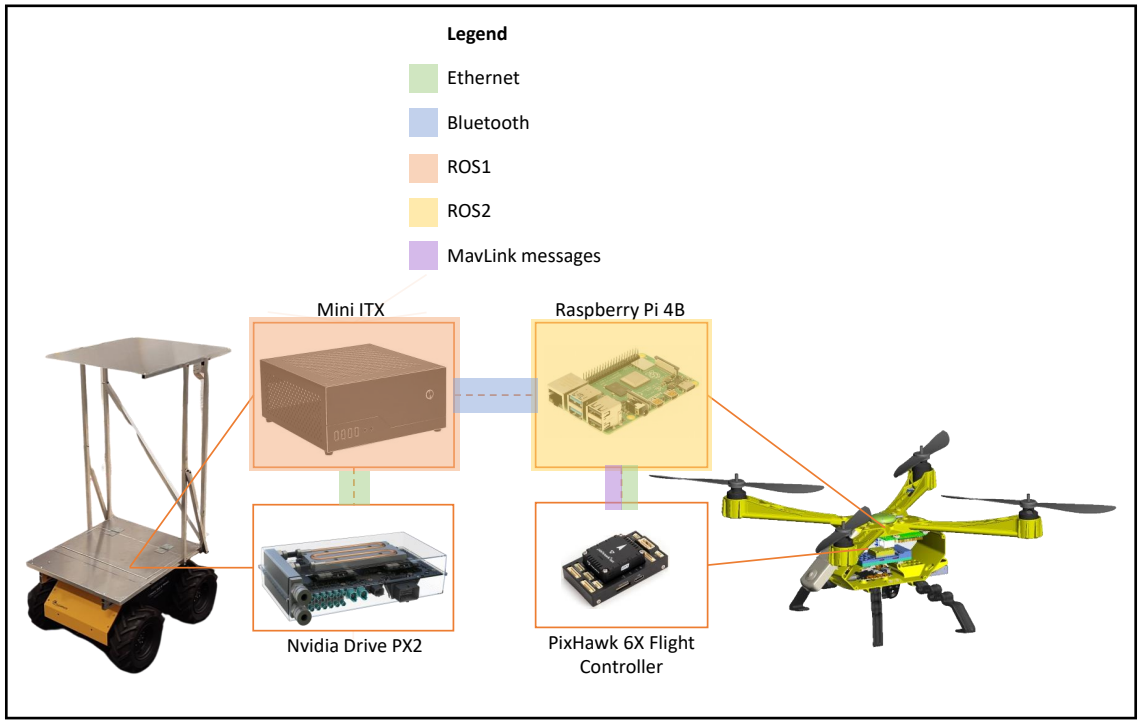


Figure 23: Communication scheme.

7.3.1 Socket communication

Since the different types of hardware employed throughout the system came with different recommended Linux distributions, it limited the choice of ROS distribution as both need to be compatible. Therefore, a ubiquitous ROS communication scheme within the system could not be implemented. The solution to this problem was to create a customised message system via Ethernet and Bluetooth socket programming in C++.

The socket communication program was designed to operate on both computers in the UGV and the companion computer on the UAV. This approach streamlines maintenance and code development, eliminating

the need for three separate programs, one for each device. The Mini ITX functions as the central node, communicating with the PX2 on the UGV via Ethernet and with the Raspberry Pi on the UAV via Bluetooth. Configurations are managed through the use of a *"config.cfg"* file. By adjusting the configuration depending on what device the program is running on, the centralised program will run and adapt its functionality accordingly.

Upon system startup, the designated server device listens for connections indefinitely until a connection is established, then transitions to message-exchanging operations. These operations take place independently in multiple threads, meaning both systems can send messages asynchronously. The UAV client will try to connect to the defined server for 15 seconds before exiting the program. If a *SOCK_STREAM* connection is established between the UGV and the UAV, the UGV initiates the transmission of VectorNAV IMU data to the UAV. This data can be used to enhance accuracy in determining the UAV's relative location to the UGV. The IMU messages, sent in a time-triggered fashion, serve the dual purpose of data exchange and acting as heartbeat messages.

Both systems are aware of connection status, with a timeout set to 1 second in the UAV and UGV. In case a timeout occurs, then either system has moved a maximum of approximately one meter before stopping since the maximum of the UGV is 1 m/s, adhering to project requirements. The UGV detects timeouts by utilising an implemented queue functionality for heartbeat messages (see Figure 24). The UGV continuously adds its sent messages to a queue, each with a corresponding message ID. If the queue threshold is passed, then the connection to the UAV is assumed to be lost. Upon receiving an acknowledgement from the UAV, the UGV pops the message with the corresponding ID from the queue along with all messages with a lower ID. Refer to Figure 25 for an illustration of how the UGV handles the queue in its receiver thread. The UAV on the other hand, employs a counter that continuously checks for received IMU data. Each time IMU data is received, the threshold counter resets to zero (see Figure 26). In case of a timeout in either system, autonomous behaviour should be implemented. Currently, functions to facilitate actions are in place. A customised message system currently can handle messages such as heartbeat, acknowledgement, stop, start, land, and stop. However, no actuation has been programmed yet for the move commands. Refer to the git repository for the code[31].

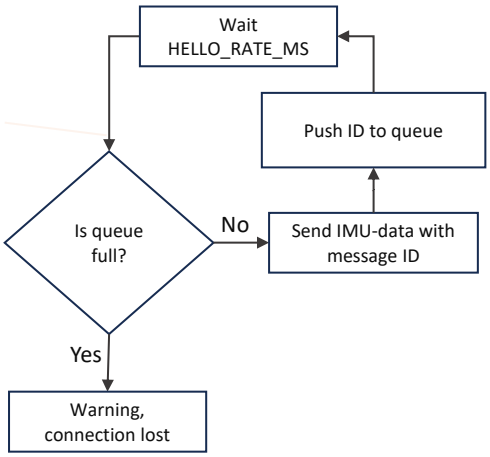


Figure 24: UGV queue threshold functionality.

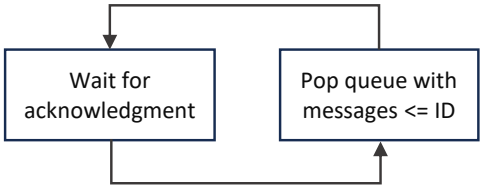


Figure 25: UGV message popping in receiver thread.

[31] https://github.com/MDU-C2/Shuttle/tree/main/communication

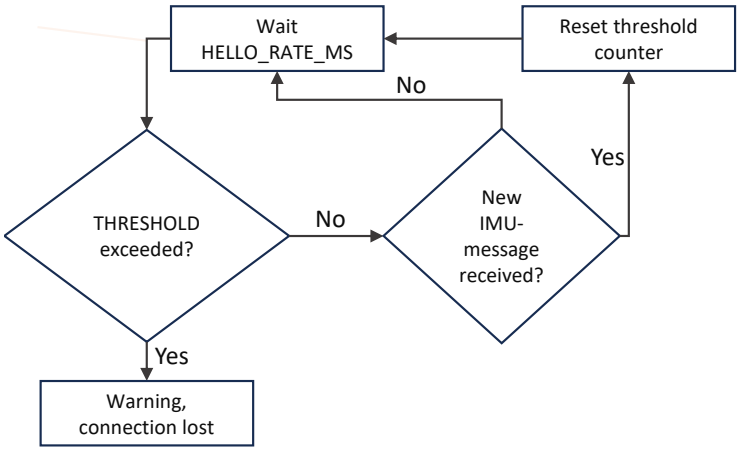


Figure 26: UAV heartbeat threshold functionality.

7.3.2 ROS nodes

ROS played a pivotal role in facilitating communication between UAV and UGV. The decision was made to maintain a clear boundary between the ROS environments of the UAV and UGV, avoiding direct communication between the two. The primary motivation for this segregation was compatibility issues in differing ROS versions and Linux distributions. The UGV's Mini ITX computer was configured to support Ubuntu 18.04, while the companion computer, a Raspberry Pi 4, was designed to accommodate Ubuntu Linux LTS versions 20 or 22. Furthermore, ROS Noetic could be installed on Ubuntu 18.04, and ROS Foxy was compatible with Ubuntu 20.04, which was the operating system deployed on the companion computer. Following a thorough investigation and an examination of the potential use of Docker to address compatibility issues, it was determined that the ideal approach was the utilisation of two distinct and isolated ROS environments for the UAV and UGV systems. This decision aimed at convenient development, ensuring that each vehicle's ROS environment could be tailored to meet its specific hardware and software prerequisites without compromise. Furthermore, this segregation strategy brought a modular and scalable architecture which enables future updates or modifications to be implemented independently for each vehicle type.

Within the context of UGV, the utilisation of ROS serves a dual purpose. Primarily, ROS is harnessed to establish comprehensive control over the vehicle in its initial phase of operation. This includes overseeing crucial aspects of UGV dynamics and ensuring precise and responsive manoeuvrability. Moreover, ROS functions as a sensory data retrieval system and monitoring of data positioned on the UGV, such as wheel encoders. In this specific configuration, ROS leverages predefined topics and nodes provided by Clearpath. These topics serve as communication channels through which information is exchanged between different nodes of the UGV system. Through these ROS topics, the controller gains access to the UGV manoeuvrability. In essence, ROS acts as the orchestrator, harmonising the communication between the UGV's sensors, controller, and the overall operational environment. Topics and nodes such as twist_mux, joy_teleop, husky_node, Husky_velocity_controller, and move_base were utilised to enable control over the UGV which is illustrated in Figure 27.

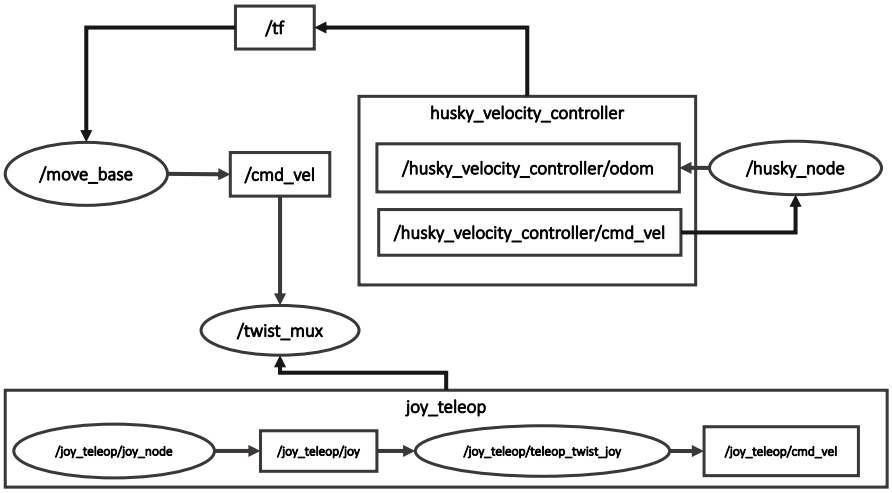


Figure 27: UGV ROS communication.

- Twist_mux is a node that serves as a twist multiplexer. It is used to combine multiple sources of velocity commands (twists) into a single output twist. This can be useful when different controllers or teleoperation nodes send velocity commands, this node can aid in merging them appropriately.
- Joy_teleop is a node responsible for teleoperation using a joystick. It subscribes to joystick input messages and converts them into velocity commands for the robot.
- Husky_node is the main node for the Husky robot. It interfaces with various hardware components on the robot and provides an interface for higher-level control nodes. It typically publishes sensor data and subscribes to velocity commands to drive the robot.
- The Husky_velocity_controller is responsible for controlling the velocity of the Husky robot based on the incoming velocity commands. It takes the desired linear and angular velocities and ends control commands to the robot's motors to achieve the specified motion.
- Move_base is a powerful node in ROS that provides a higher-level interface for controlling a robot's movement. It takes a goal in the form of a target pose and plans a trajectory to reach that goal, avoiding obstacles using information from various sensors. It integrates with local and global planners and the Husky_velocity_controller to execute the planned trajectory.

It is worth mentioning that a diagnostics node exists in this system that is used for monitoring and reporting the status of a robot's components. It helps in identifying and diagnosing issues related to sensors, hardware, and software components. As mentioned before in this section, ROS is also utilized for UAV node's intercommunication. For a detailed description of how the communication functions inside the UAV, refer to Section 7.1.6.

7.4 Tether

The tether system is a crucial component in this project, linking the UAV to the UGV and serving multiple functions to comply with regulations, provide power, and facilitate communication. The use of a tethered system for the UAV is driven by regulatory requirements in Sweden.

Tethering enhances safety and addresses concerns related to uncontrolled flight, ensuring that the drone stays within defined boundaries and adheres to airspace regulations. The tether not only serves as a physical link but also as a power supply for the UAV. This design eliminates the need for an onboard battery and significantly extends the drone's flight time by drawing power from a larger battery on the UGV prolonging aerial surveillance and monitoring. Furthermore, the tether facilitates a communication link, through a USB connection, between the UAV and the UGV. USB cable is connected to the 360° camera on the UAV. The other end of the USB cable is connected to the Mini ITX computer on the UGV. This connection is used to transmit high-rate data from a 360° camera to the UGV. A rope was also utilized in the system which

is the main part that holds the connection between UAV and UGV. Incorporating a regular rope into the tether system was a solution to protect the cables from tension and stretching. This is important in scenarios where the UAV and UGV may experience dynamic movements or encounter obstacles. The rope acts as a physical buffer, preventing damage to the cables during sudden movements or changes in terrain. The tether system also includes a cable shield. This protective element bundles and shields all the cables within the tether, preventing wear and damage. All the components of the employed tether system are illustrated in Figure 28. The length of the tether decided to be five meters in this interaction as a prototype. However, as the UAV should be able to detect upcoming obstacles, its operational height should be higher than five meters in the future. The weight of the tether is approximately 1000 grams which should be reduced by exchanging thick power cables to thin cables that convey high voltage instead of high current.

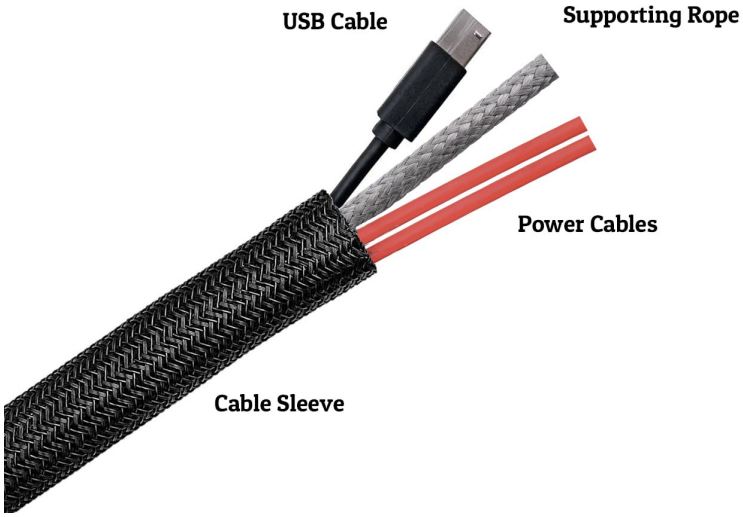


Figure 28: Tether system components.

The power cable thickness 6 mm^2 was decided based on Equations 1, 2 and 3 and to ensure a voltage drop less than 5%. Once equipped, a voltage drop test was performed to ensure that the theoretical calculations were correct.

8. Results

In this section, detailed efforts undertaken to address the research questions are presented. Tests and analyses were conducted to attain a scalable multisensor UGV platform, explore the effects of the tether in the system, and implement image processing techniques for pattern recognition on the UAV. The Sections 8.1, 8.2, and 8.3 sections provide an overview of the specific outcomes derived from the tests. This section includes key findings and relevant statistical metrics to answer the research questions.

8.1 Multisensor UGV platform

As a part of the method to answer Research Question 1, a comprehensive literature review was conducted to equip the UGV with the required sensor types for enabling autonomous driving in the future. The key role of each decided sensor type for autonomous driving is hereby summarised based on this literature review:

- **IMU:** An IMU consists of accelerometers and gyroscopes that measure the UGV's acceleration and angular rate. The sensor is typically used to address localisation [5, 7], typically fused with other sensors such as GPS, LiDAR or cameras.
- **GPS:** GPS is used to determine the UGV's global position. While GPS is valuable for providing overall location information, it may have limitations in urban environments with tall buildings. Therefore, it is often complemented with other sensors like IMU and odometry to enhance accuracy [5, 6, 7], especially in scenarios where precise positioning is critical.
- **LiDAR:** LiDAR measures distances by emitting laser pulses and measuring the time it takes for the light to return. This data is useful to detect surrounding objects, estimate their size and access the UGV position relative to the environment [5, 6, 7]. The LiDAR is appropriately used in city scenarios due to its ability to detect objects with high resolution in short ranges [9].
- **Cameras:** Cameras are crucial for visual perception in autonomous driving systems. The possible use cases for cameras in the context of autonomous vehicles are broad. They can for example be used to classify different objects and terrains [11] but also to perceive depth information [7].

Each of these sensor types namely IMU, GPS, LiDAR, and the camera has its benefits alone but greater efficiency can be achieved when combining sensor data through sensor fusion. One typical problem solved by sensor fusion is localisation. The performed literature review highlights papers that have used GPS in combination with IMU to address the localisation problem [5, 7]. The performed literature review also highlights the usage of VO due to its promising results in GNSS-denied/challenging environments [12]. Cameras in combination with LiDAR have also been showing promising results in localising different objects around a UGV [11].

In the literature review, the influence of weather and lighting conditions was also studied for exteroceptive sensors. LiDARs and cameras suffer greatly from harsh weather conditions [10]. Cameras are also greatly influenced by poor lighting conditions [7, 10].

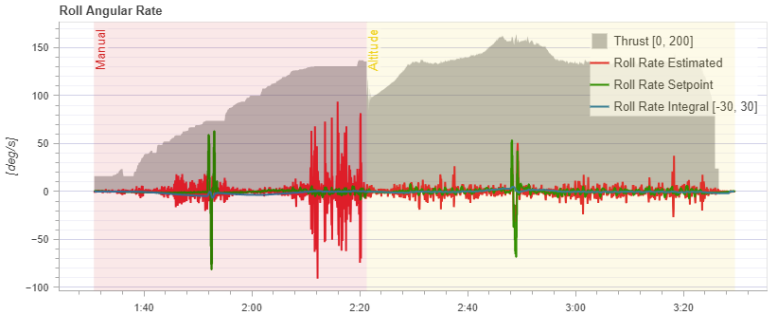
The review also aimed to understand the integration of the Robotics Husky Robot with the Nvidia Drive PX2 platform. The platform was successfully equipped inside the robot and powered by an external battery. The control of the robot was performed by the Mini ITX computer instead of the Nvidia Drive PX2.

As a next step to answer Research Question 1 the functionality and compatibility of the selected sensors were tested together with their corresponding computational device, to ensure each sensor's data is accessible. Sensor data from cameras, IMU and the SLAMTEC LiDAR were successfully accessed by either Nvidia Drive PX2 or the Mini ITX computer. Accessing data from the SICK LiDAR and the GPS antenna was not fulfilled during the project but they are equipped and connected in hardware.

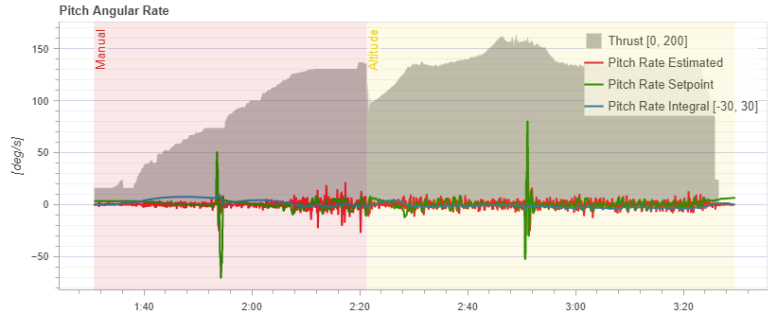
8.2 UAV flight performance

Regarding flight stability, the following section outlines some data from the flight logs before and after fine-tuning the UAV. The presented before and after data is taken approximately one month in between. Figure 29 illustrates the attitude logs from the first successful flight when the UAV was using a battery

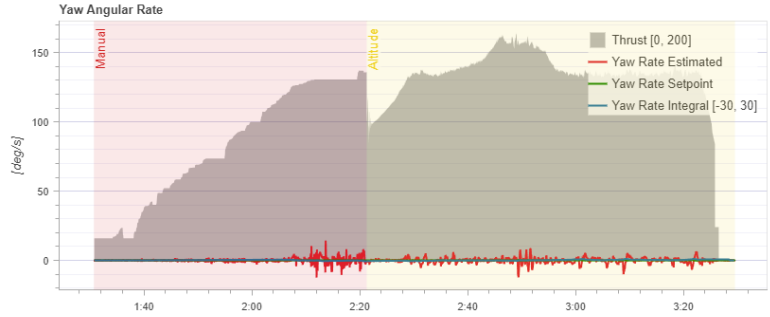
and no tether. The duration of the flight, in the X-axis, was approximately 2 minutes. During this flight, the flight mode was switched to altitude mode mid-flight around 2:20. The Y-axis shows the degrees per second the vehicle is set to turn by the pilot operating RC in green and the responses calculated by the controller in the FC in red. The flight starts at around 1:30, and at 1:50 both a roll and pitch manoeuvre are sent by the pilot in green, asking the vehicle to turn in those axes at about 50 degrees. The responses do not accurately follow the setpoint trajectory for the roll manoeuvres and overshoots, the graph is especially noisy before switching to altitude mode in Figure 29a. The yaw was not adjusted during the flight, hence no green setpoints. The pitch experienced fewer errors in comparison to the rolls, with some aggressive pitch manoeuvres around 1:50 and 2:50.



(a) Roll angular rate.



(b) Pitch angular rate.



(c) Yaw angular rate.

Figure 29: The performance of the PID controllers before tuning in roll, pitch, and yaw.

In Figure 30 the measured accelerometer vibrations spike at approximately 32 m/s^2 from the same first flight before switching flight modes as seen in the Y-axis. There could be many reasons for such high vibrations, which in turn may have caused the instability of the PID controller seen in Figure 29a. Some examples based on the experiences gained throughout the project are; a component not being fully secured in the frame, the pilot controlling the vehicle, poor vehicle calibration, insufficient power from the battery, or a combination of different reasons.

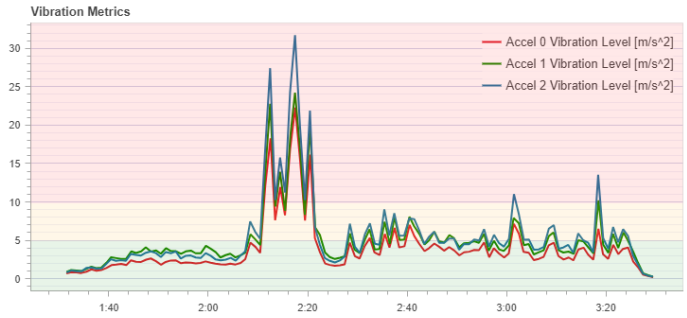
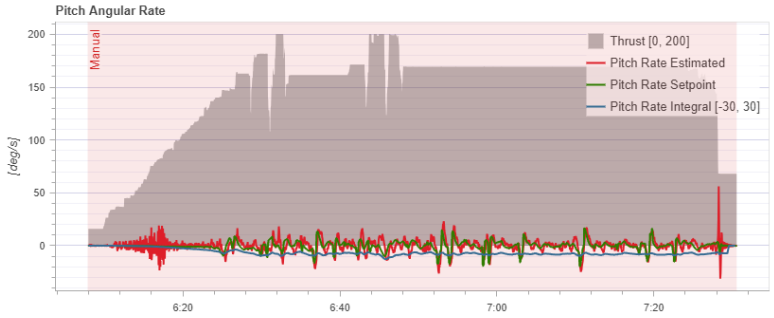


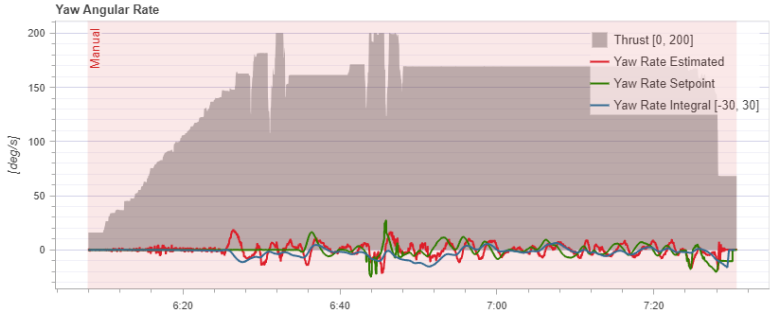
Figure 30: The accelerometer vibration metrics from the first flight.

In Figure 31, similar attitude logs are presented from one of the final flights with the UAV attached to the tether when the system was more fine-tuned at the end of the project. From this approximately 1-minute flight, the response values more closely follow the setpoints in the PID controllers. A certain noise level can be observed during take-off as seen at the 6:10 mark in Figure 31b. The setpoint and response trajectory show a greater deal of alignment after the 6:50 mark in Figures 31a and 31b, with some delay, indicating that the UAV is responsive, experience a stable flight, and following the pilot commands without noticeable overshoots.

(a) Roll angular rate.



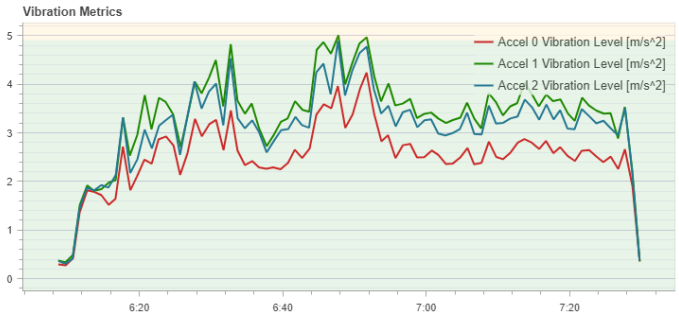
(b) Pitch angular rate.



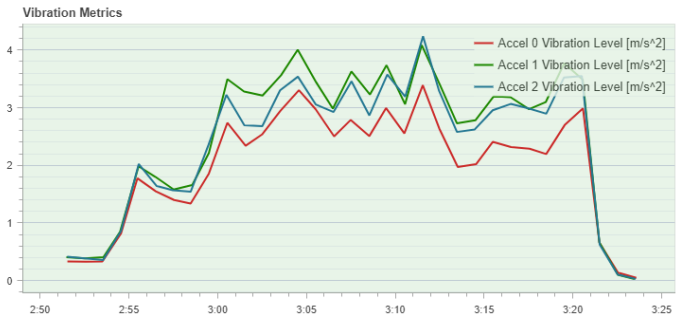
(c) Yaw angular rate.

Figure 31: The performance of the PID controllers after tuning.

In Figure 32, the measured accelerometer vibrations of approximately 4.5 m/s^2 can be seen, based on two separate flights after fine-tuning the vehicle. Comparing these results with the first flight in Figure 30, it can be observed that there is a significant decrease in vibrations.



(a) Vibration metrics of an approximately 1 min 40-second flight.



(b) Vibration metrics of an approximately 40-second flight.

Figure 32: The accelerometer vibration metrics from two separate flights after fine-tuning.

8.3 Vision detection

In Figure 33, the vision detection model was tested on 30 different images. In the figure, the X-axis represents the 30 different images while the Y-axis represents a scaled confidence score. The blue graph is the confidence of the blue target and the orange graph is the confidence of the red target. The value of image 30 is missing on the red graph because only the blue target was detected in that frame.

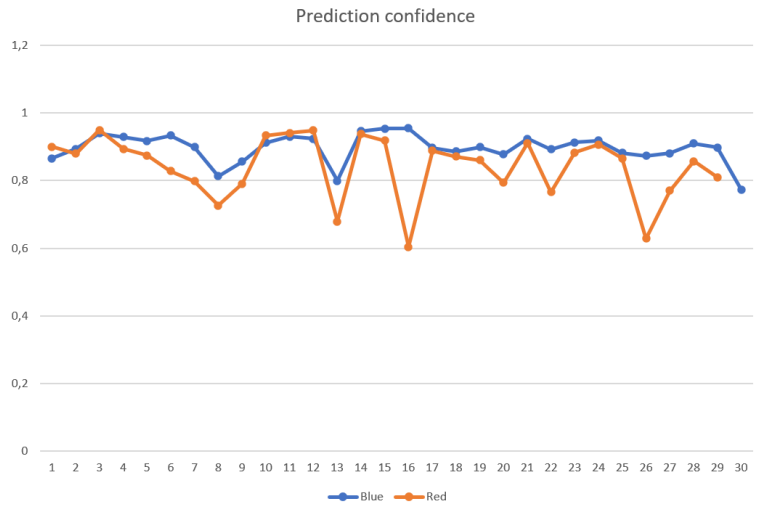
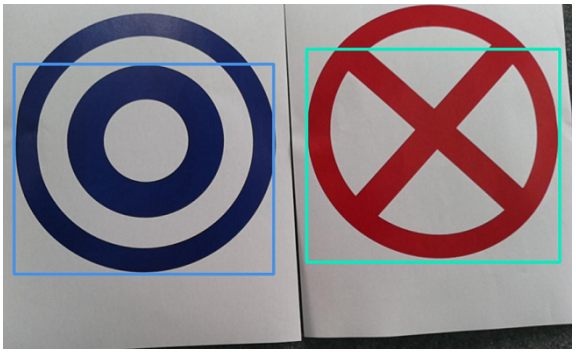


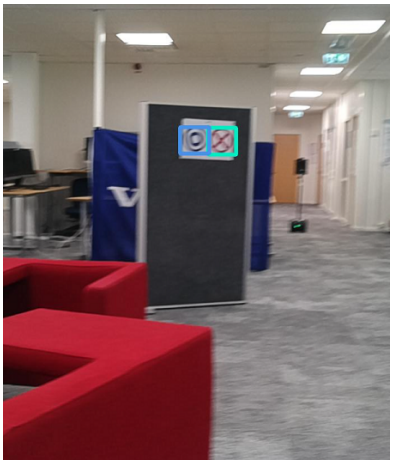
Figure 33: Confidence score of 30 different images.

Some images have significantly lower confidence scores than others as seen in Figure 33, such as image 16. Below, those images are presented in Figures 34a, 34b, 34c, 34d, and 34e for the targets together with their corresponding bounding boxes. These images effectively depict the dataset, showcasing variations in distance, angles, and blurriness induced by camera shake.

(a) Image 8 with bounding box prediction.

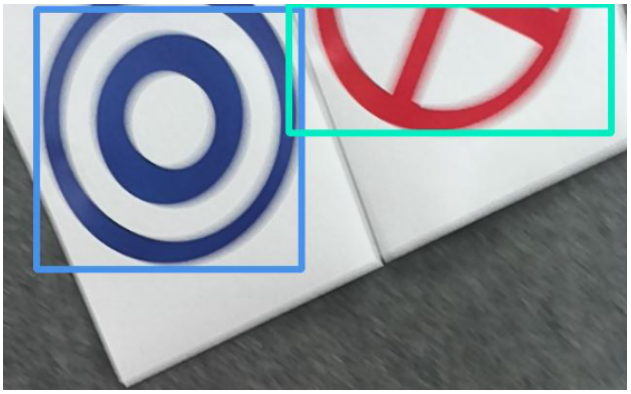


(b) Image 13 with bounding box prediction.



(c) Image 25 with bounding box prediction.

(d) Image 26 with bounding box prediction.



(e) Image 16 with bounding box prediction.

Figure 34: Various pictures with different depths of the landing targets and the corresponding bounding box the object detection model trained on.

9. Discussion

Building upon the presented results in Section 8., this section delves into the interpretation and implications of the findings. The achievement of a scalable multisensor UGV platform is discussed concerning its potential applications and contributions to the field in Section 9.1. The impact of the tether in the system is analysed, considering both its advantages and limitations in Section 9.2. Additionally, the implementation of image processing techniques for pattern recognition, pose estimation, and autonomous flight is discussed in Section 9.3. Throughout these sections, broader explanations of the results, unexpected outcomes, limitations, and future research suggestions are presented.

9.1 Multisensor UGV platform

The successful autonomous operation of a platform consisting of an UAV following an UGV for delivering goods in an urban environment relies on a carefully chosen and integrated set of sensors. This section discusses the measures taken to design a system supporting various sensor types, facilitating future scalability, and ensuring the adaptability of the UGV to operate autonomously. In this regard, a set of sensors combination including cameras, LiDARs, IMU, built-in encoders, a GPS antenna, and two onboard computers i.e. a Mini ITX and an NVIDIA Drive PX2, equips the UGV to navigate autonomously in various lighting and weather except for harsh conditions.

Diverse sensor integration including the deployment of eight cameras, strategically placed with a 60° field of view for each of them, ensures comprehensive visual coverage. However, cameras are affected by the light conditions of the environment, since in low-light scenarios they can not provide good results. Hence, two LiDAR sensors positioned on the front and back bumpers, enhance the UGV's perception capabilities. LiDARs with their technology for estimating accurate distance measurement and creating 2D maps of the surrounding environment, contribute to reliable navigation in different light and weather conditions except for harsh weather conditions. The IMU, provides information on acceleration and angular velocity that in conduction with the encoder's data, improves the UGV's localisation accuracy. These sensors are crucial for accurate and advanced navigation in GNSS-denied/ challenging environments. Moreover, the inclusion of a GPS antenna, further enhances the UGV's localisation capabilities. GPS provides absolute position information, supporting the UGV's navigation in outdoor application scenarios and offering redundancy and reliability in challenging urban environments. However, GPS alone can not be a good solution for accurate localisation in urban environments due to the risk of signal blockage by tall buildings, multipath, signal disappearing, and delayed rate for updating location.

The integration of multiple sensors adds redundancy to the system which is a critical feature for ensuring the system's robustness. In case one sensor defect or constraint due to environmental conditions other sensors can compensate, reducing the risk of failures that impact the overall performance of the system. However, to achieve this, comprehensive data fusion and potentially filtering/ optimising methods are required as well. Employing sensor fusion techniques integrates data from diverse sensors equipped on the platform. Filtering methods such as Kalman filtering which is a frequently used method, improve the integration of the sensory data, minimising noises, and errors. This comprehensive approach besides filtering/ optimising techniques enhances the reliability and accuracy of the system.

Dual onboard computers on the UGV add redundancy to the system and facilitate future scalability of the platform. The PX2 has significant capabilities in image processing, complex computations, and real-time decision making which are specially tailored for autonomous vehicles. The PX2 in cooperation with the Mini ITX computer can contribute to advanced autonomous navigation. Several reasons led to having dual onboard computers including compatibility issues of sensors with the PX2, and OS distributions. However, the use of dual onboard computers offers a flexible and scalable architecture by allowing the integration of a diverse range of sensors into the system. Currently, in the STAD project, the PX2 is dedicated to the computer vision task from the cameras where other sensors such as LiDARs and IMU data are assigned to the Mini ITX computer. In this way, the system is open to adding new sensors where the new sensors can be connected to either of the computers which enhances the adaptability of the platform in case of future expansion or technology advancement.

In this iteration of the STAD project, the UGV has been equipped with suitable sensors to enable autonomous operation in an urban environment as described. However, accessing data from some sensors, devel-

oping AI/ ML algorithms, and evaluating the performance and accuracy of autonomous operation with the current sensor combination have remained to the next iterations. One possible future work of the STAD project can be equipping the UGV with a radar to improve sensing the objects at longer ranges and assist cameras and LiDARs in harsh weather conditions since radars can perform better in such scenarios according to the literature study.

9.2 Tethered flight performance

Regarding RQ2, how the tether affects the flight performance of the UAV, one experiment with several scenarios were performed both before and after connecting the tether system. The vehicle was flown and the resulting flight data and performance were analysed through logs and experience based on observations.

There are many variables which will affect the flight performance of an UAV. Vibrations may cause damage and affect the COG if components are not secured, or the vibrations may compromise the performance of the system such as the PID calculations. In [17], their measured vibrations of 1 m/s^2 are slightly less in comparison to the STAD project which had 4.5 m/s^2. The main precaution taken to dampen the vibrations in the STAD's UAV was to use vibration-dampening pillows when mounting the FC, which was recommended by the documentation. It was also observed in the early flights that aggressive flight manoeuvres, or longer flights, coupled with the vibrations may compromise the vehicle. Therefore, more time needs to be spent on the UAV design process, ensuring that each component is tightly secured and designed to minimise and handle vibrations. Examples of such measures could be to explore mounting stands underneath each of the motor arms, or constructing the arms using another lightweight material other than plastic, since the major contributor to the vibrations are the motors.

In case manual modes are used, another major factor is the pilot operating the vehicle. They need to remain calm and be comfortable in flying the UAV. Aggressive movements or accidental switch flips may cause vehicle damage, prolonging testing until the damage is fixed. Due to the nature of such flights, it proved to be challenging to control all variables while still adhering to the standards of experimentation. For example, during different flights, the pilot would have to adjust the UAV based on the situation and location of the tether. Consequentially, it is difficult to make a fair comparison of such flights.

Considering flying the vehicle with a taut tether as in [14], some tests were performed in the STAD project. A benefit of flying with a taut tether is that altitude will not vary greatly, except during takeoff and landing, meaning that the COG with the tether system will be easier to calculate, subsequently leading to better flight performance. However, due to the UAV relying on the manual flight mode at the time and the indoor experimentation area being too small, it was decided that such tests have too high of a risk of damaging the UAV. More time needs to be spent exploring autonomous flight in a large and safe environment, or with a shorter tether, to succeed with flying with a taut tether in a non-dangerous way.

The sensitivity of the sensors was also an issue when wanting to fly the UAV. The sensors had to be re-calibrated several times and the horizon had to be levelled when the vehicle moved, to pass the PX4 pre-flight checks which allowed arming the vehicle. If the UAV is to be truly autonomous, meaning operating completely independently and making decisions without any external input, some time needs to be spent in for example automating this procedure.

Examining flight logs in other projects, the PID trajectory graphs seen in the PX4 documentation in [27] perform extremely well. For these graphs, it is described that the vehicle performed acrobatic flips in the air, and yet there are minimal overshoots and noise in contrast to the STAD UAV in Figure 31. Although it is unclear exactly how the flight was performed, or how the vehicle tuned and configured in this example. In [18], a similar tracking from the PID can be seen as in the STAD project. It is however clear that in Figure 31, there are still many errors and delays, especially during vehicle takeoff. This phenomenon was noticeable for the pilot operating the RC as well as he had to carefully manually adjust the vehicle roll/pitch to not flip the vehicle during takeoff. One reason was the long and heavy tether connected to the system and rolled up on the ground.

Several strategies have been employed to enhance the flight performance of both tethered and non-tethered UAV's. Typically, different sensors such as IMU, GPS, and vision are used as in [69], they may be combined with external controllers utilising different algorithms as in [14] or use some ML algorithm as in [16].

Typically, a mechanical or electric winch system is used to control the tether. Such systems can help prevent the ropes and cables from getting caught up and allow better control of the tether length. In this iteration of the STAD project, the main focus of the UAV was on the design, build, and initial flight. Future iterations of the project may explore winch systems, external controllers, or outdoor GPS which are required for testing the fully autonomous flight modes PX4 offers. For PX4-based UAV's, the GPS is also responsible for relaying the system date and time, which enables easier analysis of logs. In a future iteration of the project, it would also be beneficial to install some telemetry support on the UAV to save and analyse real-time data wirelessly.

9.3 Computer vision for UAV

In an attempt to answer RQ3, the depth camera was used to take images at various distances and angles from the targets. Thirty test images that were not part of the training data set, were chosen and presented in a graph. Some images of interest were also portrayed in the result and the reasoning behind them is discussed in this section. Additionally, other methods are discussed as there might be other paths of interest going forward, that differ from the work made in this iteration.

As the graph in Figure 33 illustrates, the confidence score is above 80% for most of the predictions. For many of them, the confidence is approximately 95%. The object detection model was however selected to have a confidence of 80% which means that objects detected with a confidence of 79% or lower will be discarded. The level of confidence could perhaps be too high, resulting in losing some of the images that could otherwise have been used for pose estimation. From observing the graph, perhaps a level near 60% would have been a better fit.

The model also seems to detect the blue target better than the red target. In practice, this is most likely not a significant difference because the red target is also more distorted on average, which is demonstrated in Figures 34a and 34c. Expanding on this, in Figure 34a the targets are viewed from an angle, which most likely caused the confidence score to go down. The red target is further from the camera, and it is smaller and more skewed which probably influenced the confidence negatively. In Figure 34b, it is clear that the bounding boxes do not cover the entirety of the targets. The confidence score is also relatively low, even though the targets are distinct in the image. It appears that the object detection model struggles when targets cover a too large area of the screen. The UAV is meant to fly at a height of five meters above the targets, which means the targets will never be seen this close except from takeoff or landing. Still, a worse confidence score at closer distances could worsen the performance of a takeoff or landing algorithm, making the reliance on other sensors such as IMU greater. Additionally, a byproduct of only detecting the targets partially is the fact the position is now estimated to be in a different location than straight below the vehicle.

Figure 34c, shows a scenario in which one of the targets is not fully visible. Still being able to detect it is very useful as this gives the UAV more leeway when flying, and it would not matter as much if one target goes slightly out of bounds. The images presented in Figures 34d and 34e are interesting because they appear to be very similar, but Figure 34e resulted in a significantly worse confidence score. The red target is slightly more distorted, but so is the blue target, which still was detected with great confidence.

The results demonstrate that a novel visual image processing method can be used to track targets, but probably not by utilising the depth from a depth camera. As mentioned in the implementation, the depth received from the camera was noisy and unusable which led to the decision to calculate depth using the sizes of the bounding boxes. When flying at the correct altitude, the bounding boxes fit the targets well like in Figure 34a.

To align the UAV to the targets and follow them, it would require the UAV to be autonomous, which it currently is not. The pose estimation algorithm presented provides good data about where in space, the UAV is. These values must then be used for moving the UAV accordingly, which is difficult to achieve, because without external sensors such as GPS, only raw input controls like throttle, yaw, pitch, and roll are available.

In the background (Section 2.1.3) and related work (Section 3.1.2), several external positioning sources were mentioned, such as GPS, motion capture, and VO. GPS could be useful in the future when the entire system is made autonomous. For precision flying, this method is out of the question. Preferably, centimetre

precision should be used which is something GPS is unable to provide. Motion capture is essentially already implemented in the pose estimation algorithm. Utilising a motion capture camera as the input source directly to the flight controller would grant access to position control because it is a supported sensor source. This is also true for VO. The accuracy of these methods must however be further investigated. When a monocular camera was used for VO in article [66], the results were GPS like performance. They mentioned that stereovision VO gives significantly better precision, which could be a reason for an investigation.

The counterargument to using position control arises with RQ2 and the noise of the flight logs. The UAV was observed to oscillate in different directions and had difficulties stabilising when changing altitude as a result of the heavy tether. Perhaps, the greater control of the throttle that is achieved through using raw inputs is essential for stability, as no other algorithm is made for a tether that constantly changes the center of mass of the vehicle. Further investigation should be done in the future before position control is decided upon.

10. Conclusions

The project has taken on the task of designing and implementing a UAV-UGV system to investigate autonomous delivery solutions in dynamic environments. To enable the UGV's autonomous operations, the Husky A200 has been equipped with LiDAR, cameras, IMU and GPS. The sensors have been mounted and connected to the computers, enabling future research to start exploring autonomous navigation utilising the system such as dynamic real-time object detection and tracking, path planning, mapping, and sensor fusion. A requirement for the system was that it should be integrated with a tether between the UAV and UGV, this complicated the flight of the UAV with the added weight which also constantly shifts the centre of mass of the UAV. With these challenges, the UAV had to be cautiously calibrated. In this project, the drone achieved a relatively stable flight when being piloted manually. No autonomous flights were performed during this iteration of the project. In future iterations of the project, the visual image processing method implemented in this project can be utilised for autonomous flights. The method achieved high confidence in detecting the landing pads and could be used to centre and orient the UAV over the UGV. The depth data was noisy and performed poorly, therefore the altitude is currently calculated using the sizes of the landing targets. To make the system safer and more robust, additional methods should be incorporated for monitoring the altitude of the UAV such as visual odometry or GPS.

References

[1] Kiwibot. Kiwibot autonomous delivery robots, revolutionizing the future of robotic delivery., 2023. URL https://www.kiwibot.com/. Accessed: 2023-12-20.

[2] Tele Retail. Logistics Automation AI | Logistics Automation AI, Autonomous Transport, 2023. URL https://teleretail.com/. Accessed: 2023-12-20.

[3] Starship Technologies. Starship technologies: Autonomous robot delivery, 2023. URL https://www.starship.xyz/. Accessed: 2023-12-20.

[4] Udelv. udelv | a driverless, multi-stop delivery EV for last-mile and middle-mile deliveries., 2022. URL https://udelv.com/. Accessed: 2023-12-20.

[5] Dr Georgios A Demetriou. A survey of sensors for localization of unmanned ground vehicles (UGVs). In *Proceedings of the 2006 International Conference on Artificial Intelligence*, 2006.

[6] Chang Liu, Jin Zhao, and Nianyi Sun. A review of collaborative air-ground robots research. *Journal of Intelligent & Robotic Systems*, 106(3):60, 2022. ISSN 1573-0409. doi: 10.1007/s10846-022-01756-4. URL https://doi.org/10.1007/s10846-022-01756-4.

[7] Haowen Gao, Sheng Cheng, Zhao Chen, Xiaofeng Song, Zhihong Xu, and Xiaohong Xu. Design and implementation of autonomous mapping system for UGV based on lidar. In *2022 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, pages 1–6, 2022. doi: 10.1109/ICNSC55942.2022.10004073.

[8] Saurav Kumar and Pallavi Awasthi. Navigation architecture for autonomous surveillance rover. *International Journal of Computer Theory and Engineering*, pages 231–235, 2009. doi: 10.7763/IJCTE.2009.V1.37.

[9] Luigi Giuffrida, Guido Masera, and Maurizio Martina. A Survey of Automotive Radar and Lidar Signal Processing and Architectures. *Chips*, 2(4):243–261, December 2023. ISSN 2674-0729. doi: 10.3390/chips2040015. URL https://www.mdpi.com/2674-0729/2/4/15. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[10] Fatih Sezgin, Daniel Vriesman, Dagmar Steinhauser, Robert Lugner, and Thomas Brandmeier. Safe autonomous driving in adverse weather: Sensor evaluation and performance monitoring. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–6, 2023. doi: 10.1109/IV55152.2023.10186596.

[11] Avi Spector, Wanying Zhu, Jumman Hossain, and Nirmalya Roy. Simulated Forest Environment and Robot Control Framework for Integration with Cover Detection Algorithms. In *2022 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, pages 277–283, December 2022. doi: 10.1109/BDCAT56447.2022.00046.

[12] Yusra Alkendi, Lakmal Seneviratne, and Yahya Zweiri. State of the art in vision-based localization techniques for autonomous navigation systems. *IEEE Access*, 9:76847–76874, 2021. ISSN 2169-3536. doi: 10.1109/ACCESS.2021.3082778. URL https://ieeexplore.ieee.org/document/9438708/.

[13] Seiga Kiribayashi, Kaede Yakushigawa, and Keiji Nagatani. Design and Development of Tether-Powered Multirotor Micro Unmanned Aerial Vehicle System for Remote-Controlled Construction Machine. In Marco Hutter and Roland Siegwart, editors, *Field and Service Robotics*, Springer Proceedings in Advanced Robotics, pages 637–648, Cham, 2018. Springer International Publishing. ISBN 978-3-319-67361-5. doi: 10.1007/978-3-319-67361-5_41.

[14] Lanjiang Yang, Guang-Xun Du, Yan Gao, and Quan Quan. Position control of tethered uav with onboard inertial sensors. In *2022 41st Chinese Control Conference (CCC)*, pages 2870–2875, 2022. doi: 10.23919/CCC55666.2022.9901867.

[15] Andrea Borgese, Dario C. Guastella, Giuseppe Sutera, and Giovanni Muscato. Tether-Based Localization for Cooperative Ground and Aerial Vehicles. *IEEE Robotics and Automation Letters*, 7 (3):8162–8169, July 2022. ISSN 2377-3766. doi: 10.1109/LRA.2022.3187504. URL https://ieeexplore.ieee.org/abstract/document/9811264. Conference Name: IEEE Robotics and Automation Letters.

[16] S. Martínez-Rozas, D. Alejo, F. Caballero, and L. Merino. Path and trajectory planning of a tethered UAV-UGV marsupial robotic system. *IEEE Robotics and Automation Letters*, 8(10):6475–6482, 2023. ISSN 2377-3766, 2377-3774. doi: 10.1109/LRA.2023.3301292. URL https://ieeexplore.ieee.org/document/10207830/.

[17] Douglas Etts, Mark Rossi, Roland Nzaou, Ruijie Zhu, Gregory C. Lewin, and Stephan F.J. de Wekker. Development of an autonomous multi-rotor copter for collecting atmospheric data near the ground. In *2015 Systems and Information Engineering Design Symposium*, pages 120–124, 2015. doi: 10.1109/SIEDS.2015.7116958.

[18] Jingqi Jiang, Xuetao Zhang, Jing Yuan, Kaitao Tang, and Xuebo Zhang. Extendable flight system for commercial uavs on ros. In *2018 37th Chinese Control Conference (CCC)*, pages 1–5, 2018. doi: 10.23919/ChiCC.2018.8483362.

[19] Runze Tian, Runze Ji, Chengchao Bai, and Jifeng Guo. A vision-based ground moving target tracking system for quadrotor uavs. In *2023 IEEE International Conference on Unmanned Systems (ICUS)*, pages 1750–1754, 2023. doi: 10.1109/ICUS58632.2023.10318456.

[20] Dronecode Project. Basic concepts | px4 user guide (main), Sep 2023. URL https://docs.px4.io/main/en/getting_started/px4_basic_concepts.html. Accessed: 2023-12-19.

[21] Airpower Journal. Unmanned aerial vehicles, 1991. URL https://web.archive.org/web/20090724015052/http://www.airpower.maxwell.af.mil/airchronicles/apj/apj91/spr91/4spr91.htm. Accessed: 2023-12-26.

[22] Droneii. Ed Alvarado. 237 ways drone applications revolutionize business, May 2021. URL https://droneii.com/237-ways-drone-applications-revolutionize-business. Accessed: 2023-12-26.

[23] Dronecode Project. Holybro pixhawk 6x | px4 user guide (main), Aug 2023. URL https://docs.px4.io/main/en/flight_controller/pixhawk6x.html. Accessed: 2023-12-19.

[24] ArduPilot Dev Team. Ardupilot documentation, Dec 2023. URL https://ardupilot.org/ardupilot/index.html. Accessed: 2023-12-20.

[25] Dronecode Project. Sensors | px4 user guide (main), Sep 2023. URL https://docs.px4.io/main/en/getting_started/sensor_selection.html. Accessed: 2023-12-19.

[26] National Instruments. The pid controller & theory explained, Mar 2023. URL https://www.ni.com/en/shop/labview/pid-theory-explained.html. Accessed: 2023-12-28.

[27] Dronecode Project. Multicopter pid tuning guide (manual/advanced) | px4 user guide (main), Nov 2023. URL https://docs.px4.io/main/en/config_mc/pid_tuning_guide_multicopter.html. Accessed: 2023-12-19.

[28] Dronecode Project. Auto-tuning | px4 user guide (main), Nov 2023. URL https://docs.px4.io/main/en/config/autotune.html. Accessed: 2023-12-19.

[29] Dronecode Project. Flight controller/sensor orientation | px4 user guide (main), Nov 2023. URL https://docs.px4.io/main/en/config/flight_controller_orientation.html. Accessed: 2023-12-19.

[30] Dronecode Project. Actuator configuration and testing | px4 user guide (main), Dec 2023. URL https://docs.px4.io/main/en/config/actuators.html. Accessed: 2023-12-19.

[31] Dronecode Project. Logging | px4 user guide (main), Nov 2023. URL https://docs.px4.io/main/en/dev_log/logging.html. Accessed: 2023-12-20.

[32] Dronecode Project. Esc calibration | px4 user guide (main), Nov 2023. URL https://docs.px4.io/main/en/advanced_config/esc_calibration.html. Accessed: 2023-12-19.

[33] Dronecode Project. Px4 flight modes overview | px4 user guide (main), Dec 2023. URL https://docs.px4.io/main/en/getting_started/flight_modes.html. Accessed: 2023-12-19.

[34] Dronecode Project. Radio control systems | px4 user guide (main), Sep 2023. URL https://docs.px4.io/main/en/getting_started/rc_transmitter_receiver.html. Accessed: 2023-12-20.

[35] RadioMaster. Tx12 mark ii radio controller - radiomaster rc, Dec 2023. URL https://www.radiomasterrc.com/products/tx12-mark-ii-radio-controller?variant=44128450576615. Accessed: 2023-12-20.

[36] Dronecode Project. Fly view | qgc guide (master), Dec 2023. URL https://docs.qgroundcontrol.com/master/en/qgc-user-guide/fly_view/fly_view.html. Accessed: 2023-12-20.

[37] Transportstyrelsen. Transportstyrelsen - drones, 2023. URL https://www.transportstyrelsen.se/dronare. Accessed: 2023-12-18.

[38] N Storey. *Electronics A Systems Approach*. Pearson Education Limited, UK, 6 edition, 2017.

[39] S Roberts. *DC-DC Book of Knowledge*. RECOM Engineering, Austria, 1 edition, 2016.

[40] Sarah Tang and Vijay Kumar. Autonomous flight. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):29–52, 2018. doi: 10.1146/annurev-control-060117-105149. URL https://doi.org/10.1146/annurev-control-060117-105149.

[41] Bruno Siciliano and Oussama Khatib. *Springer Handbook of Robotics*. Springer, 2008.

[42] G Natarajan. Ground control stations for unmanned air vehicles. *Defence Science Journal*, pages 229–237, 2001. doi: 10.14429/dsj.51.2234.

[43] A. Masiero, F. Fissore, R. Antonello, A. Cenedese, and A. Vettore. A comparison of uwb and motion capture uav indoor positioning. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13:1695–1699, 2019. doi: 10.5194/isprs-archives-XLII-2-W13-1695-2019. URL https://isprs-archives.copernicus.org/articles/XLII-2-W13/1695/2019/.

[44] Mikkel Baun Kjærgaard, Henrik Blunck, Torben Godsk, Thomas Toftkjær, Dan Lund Christensen, and Kaj Grønbæk. Indoor positioning using gps revisited. In Patrik Floréen, Antonio Krüger, and Mirjana Spasojevic, editors, *Pervasive Computing*, pages 38–56, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-12654-3.

[45] Vikrant More, Hitendra Kumar, Sarthak Kaingade, Pradeep Gaidhani, and Nitin Gupta. Visual odometry using optic flow for unmanned aerial vehicles. In *2015 International Conference on Cognitive Computing and Information Processing(CCIP)*, pages 1–6, 2015. doi: 10.1109/CCIP.2015.7100731.

[46] Deqing Sun, Stefan Roth, J. P. Lewis, and Michael J. Black. Learning optical flow. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 83–97, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-88690-7.

[47] Abdulla Al-Kaff, David Martín, Fernando García, Arturo de la Escalera, and José María Armingol. Survey of computer vision algorithms and applications for unmanned aerial vehicles. *Expert Systems with Applications*, 92:447–463, 2018. ISSN 0957-4174. doi: https://doi.org/10.1016/j.eswa.2017.09.033. URL https://www.sciencedirect.com/science/article/pii/S0957417417306395.

[48] L Delvaux and L Di Naro. Autopilot and companion computer for unmanned aerial vehicle: Survey, 2023.

[49] Dronecode Foundation. MAVLink. https://mavlink.io/en/, 2023. Accessed: 2023-12-20.

[50] Dronecode Foundation. MAVSDK. https://mavsdk.mavlink.io/main/en/index.html, 2023. Accessed: 2023-12-20.

[51] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[52] Fiza Joiya. Object detection: Yolo vs faster r-cnn. *Int. Res. J. Mod. Eng. Technol. Sci*, 4:1911–1915, 2022.

[53] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. URL http://arxiv.org/abs/1506.02640.

[54] Naveed Ur Rehman, Kundan Kumar, and Ghulam e Mustafa Abro. Implementation of an autonomous path planning & obstacle avoidance UGV using SLAM. In *2018 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–5, 2018. doi: 10.1109/ICEET1.2018.8338628.

[55] ClearpathHusky. https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/, 2023. Accessed: 2023-12-19.

[56] I Nourbakhsh R Siegwart. *Introduction to Autonomous Mobile Robots*. The MIT Press, Cambridge Massachussets, 2 edition, 2011.

[57] Xingshuai Dong and Massimiliano L. Cappuccio. Applications of Computer Vision in Autonomous Vehicles: Methods, Challenges and Future Directions, November 2023. URL http://arxiv.org/abs/2311.09093. arXiv:2311.09093 [cs].

[58] NVIDIA Drive PX2. https://developer.nvidia.com/drive/px2, 2023. Accessed: 2023-12-19.

[59] NVIDIA Corporation. NVIDIA Drive OS. https://developer.nvidia.com/drive/os, 2023. Accessed: 2023-12-20.

[60] P. Thulasiraman, Z. Chen, B. Allen, and B. Bingham. Evaluation of the robot operating system 2 in lossy unmanned networks. In *2020 IEEE International Systems Conference (SysCon)*, pages 1–8, 2020. doi: 10.1109/SysCon47679.2020.9275849. ISSN: 2472-9647.

[61] Definition of a socket. RFC 147, may 1971. URL https://www.rfc-editor.org/info/rfc147. Accessed: 2023-12-19.

[62] Oct 2023. URL https://www.ibm.com/docs/en/i/7.5?topic=programming-how-sockets-work. Accessed: 2023-12-18.

[63] Steven Macenski, Alberto Soragna, Michael Carroll, and Zhenpeng Ge. Impact of ros 2 node composition in robotic systems. *IEEE Robotics and Autonomous Letters (RA-L)*, 2023.

[64] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette, and William Woodall. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, 7(66):eabm6074, 2022. doi: 10.1126/scirobotics.abm6074. URL https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[65] Zhi Hou, Juntong Qi, and Mingming Wang. Fusing optical flow and inertial data for uav motion estimation in gps-denied environment. In *2019 Chinese Control Conference (CCC)*, pages 7791–7796, 2019. doi: 10.23919/ChiCC.2019.8866551.

[66] Neville Koh Wei Yan, Daniel Cheng Wang Tak, Matthew Ong Ying Quan, and Sutthiphong Srigrarom. Monocular visual odometry for small uavs. In *2023 Third International Symposium on Instrumentation, Control, Artificial Intelligence, and Robotics (ICA-SYMP)*, pages 9–14, 2023. doi: 10.1109/ICA-SYMP56348.2023.10044928.

[67] Narsimlu Kemsaram, Anweshan Das, and Gijs Dubbelman. An integrated framework for autonomous driving: Object detection, lane detection, and free space detection. In *2019 Third World Conference on Smart Trends in Systems Security and Sustainablity (WorldS4)*, pages 260–265, 2019. doi: 10.1109/WorldS4.2019.8904020.

[68] Ming Yang, Shige Wang, Joshua Bakita, Thanh Vu, F. Donelson Smith, James H. Anderson, and Jan-Michael Frahm. Re-thinking cnn frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 305–317, 2019. doi: 10.1109/RTAS.2019.00033.

[69] H. Xu, C. Wang, Y. Bo, C. Jiang, Y. Liu, S. Yang, and W. Lai. An Aerial and Ground Multi-Agent Cooperative Location Framework in GNSS-Challenged Environments. *Remote Sensing*, 14(19), 2022. ISSN 2072-4292. doi: 10.3390/rs14195055.

[70] Manuel Sánchez-Montero, Manuel Toscano-Moreno, Juan Bravo-Arrabal, Javier Serón Barba, Pablo Vera-Ortega, Ricardo Vázquez-Martín, Juan Jesús Fernandez-Lozano, Anthony Mandow, and Alfonso García-Cerezo. Remote Planning and Operation of a UGV Through ROS and Commercial Mobile Networks. In Danilo Tardioli, Vicente Matellán, Guillermo Heredia, Manuel F. Silva, and Lino Marques, editors, *ROBOT2022: Fifth Iberian Robotics Conference*, Lecture Notes in Networks and Systems, pages 271–282, Cham, 2023. Springer International Publishing. ISBN 978-3-031-21065-5. doi: 10.1007/978-3-031-21065-5_23.

[71] J. Bravo-Arrabal, M. Toscano-Moreno, J.J. Fernandez-Lozano, A. Mandow, J.A. Gomez-Ruiz, and A. García-Cerezo. The internet of cooperative agents architecture (X-ioca) for robots, hybrid sensor networks, and mec centers in complex environments: A search and rescue case study. *Sensors*, 21 (23), 2021. ISSN 1424-8220. doi: 10.3390/s21237843.

[72] J. Morales, I. Castelo, R. Serra, P.U. Lima, and M. Basiri. Vision-Based Autonomous Following of a Moving Platform and Landing for an Unmanned Aerial Vehicle. *Sensors*, 23(2), 2023. ISSN 1424-8220. doi: 10.3390/s23020829.

[73] M. Udochu Nnennaya, Etse-Oghena Akpaibor, Akash P. Borate, Aakash G. Deshpande, and Frank Lewis. Joint Behavioural Control of Autonomous Multi-Robot Systems for Lead-Follower Formation to Improve Human-Robot Interaction. In *Proceedings of the 9th ACM International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '16, pages 1–8, New York, NY, USA, June 2016. Association for Computing Machinery. ISBN 978-1-4503-4337-4. doi: 10.1145/2910674.2910676.

[74] Nov 2020. URL `https://www.bluetooth.com/blog/how-bluetooth-technology-uses-adaptive-frequency-hopping-to-overcome-packet-interference/`. Accessed: 2023-12-19.

[75] E. Ferro and F. Potorti. Bluetooth and wi-fi wireless protocols: a survey and a comparison. *IEEE Wireless Communications*, 12(1):12–26, 2005. doi: 10.1109/MWC.2005.1404569.

[76] Transportstyrelsen. Transportstyrelsen - self-driving vehicles, 2023. URL `https://www.transportstyrelsen.se/en/road/Vehicles/self-driving-vehicles/`. Accessed: 2023-12-18.

[77] The geometry of perspective projection, Mar 2003. URL `https://www.cse.unr.edu/~bebis/CS791E/Notes/PerspectiveProjection.pdf`. Accessed: 2023-12-22.

[78] Microchip. High speed usb, Nov 2023. URL `https://microchipdeveloper.com/xwiki/bin/view/applications/usb/speeds-specs/high-speed/`. Accessed: 2024-01-10.

A Individual contributions

Section	Title	Main contributor(s)
0.	Abstract	Erik
0.	Acknowledgements	Andreas
1.	Introduction	Walter
2.	Background	Kasra
2.1	UAV	Sebastian
2.1.1	Key UAV components, concepts, and flight	Sebastian & Andreas
2.1.2	Power a UAV from a UGV	Walter
2.1.3	Autonomous Flight	Elon
2.1.4	Companion Computer	Elon
2.1.5	Computer Vision	Elon
2.2	UGV	Sharifeh
2.2.1	Localisation	Walter
2.2.2	Computer vision approaches for autonomous platforms	Walter
2.2.3	Nvidia Drive PX2	Erik
2.3	Integration of UGV and UAV	Andreas & Kasra
2.3.1	Sockets	Andreas
2.3.2	ROS	Kasra
3.	Related Work	Andreas
3.1	UAV	Andreas
3.1.1	UAV stability and control strategies	Sebastian
3.1.2	UAV localisation techniques	Elon
3.2	UGV	Sharifeh
3.2.1	UGV localisation techniques	Sharifeh
3.2.2	Sensors efficiency in various conditions	Walter
3.2.3	NVIDIA Drive PX2 Applications	Erik
3.3	Integration	Andreas
3.3.1	UGV and UAV cooperation in GNSS-challenged environments	Andreas
3.3.2	Remote Planning and Operation of a UGV Through ROS	Kasra
3.3.3	Tether-based UGV and UAV systems	Andreas
4.	Problem Formulation	Sebastian & Andreas
5.	Method	Andreas
5.1	Methodology	Andreas
5.2	Procedures	Andreas & Kasra
5.3	Tools	-
5.3.1	Ultimaker 2+	Walter
5.3.2	Original Prusa i3 MK3S	Walter
5.3.3	Ultimaker Cura	Walter
5.3.4	PrusaSlicer	Walter
5.3.5	Visual Studio Code	Andreas
5.3.6	SolidWorks	Walter
5.3.7	Oracle VM VirtualBox	Andreas
5.3.8	QGroundControl	Andreas & Sebastian
5.3.9	Gazebo	Andreas
6.	Ethical and Societal Considerations	Erik

Section	Title	Main contributor(s)
7.	Implementation	Andreas
7.1	UAV construction	Sebastian
7.1.1	UAV hardware and design	Sebastian
7.1.2	UAV software and calibration for initial flight	Sebastian
7.1.3	Design of landing targets	Elon
7.1.4	Object detection model	Elon
7.1.5	Pose estimation	Elon
7.1.6	Companion computer software design	Elon
7.1.7	GUI node	Elon
7.1.8	360° camera stream	Andreas
7.2	UGV construction	Andreas
7.2.1	Design and Construction of Loading Area and Landing Pad	Walter
7.2.2	Sensor Selection	Sharifeh
7.2.3	Component Placement	Walter
7.2.4	Power Management	Sharifeh
7.2.5	Wireless Controller	Andreas
7.2.6	LiDAR	Kasra
7.2.7	IMU Software	Kasra
7.2.8	Cameras	Erik
7.3	Communication	Andreas
7.3.1	Socket communication	Andreas
7.3.2	ROS nodes	Kasra
7.4	Tether	Kasra
8.	Results	Kasra
8.1	Multisensor UGV platform	Walter
8.2	UAV flight performance	Sebastian
8.3	Vision detection	Elon
9.	Discussion	Kasra
9.1	Multisensor UGV platform	Sharifeh
9.2	Tethered flight performance	Sebastian
9.3	Computer vision for UAV	Elon
10.	Conclusions	Erik