UNIVERSITY OF TURKU

A Method for Semi-Autonomous Map Generation and Path Finding In UAV-UGV Cooperation for Search and Rescue Mission in Dynamic and Unmapped Environment

Computer Science/Faculty of Technology

Master's thesis

Author:
Olli Kiviniemi

2.5.2026

Turku

Master's thesis

Subject: Data Analytics & Robotics

Author: Olli Kiviniemi

Title: A Method for Semi-Autonomous Map Generation and Path Finding In UAV-UGV Cooperation for Search and Rescue Mission in Dynamic and Unmapped Environment

Supervisors: Professor Jukka Heikkonen, Professor Tomi Westerlund

Number of pages: 77 pages

Date: 2.5.2026

Catastrophes, accidents and wars can create circumstances where rescue missions need to be conducted without delay. These rescue missions can be as dangerous for rescuers as they are for the ones waiting to be rescued. Landscape may be deformed due to the events, or environment can have dynamic events occurring and therefore prior mapping of the area is not valid. Therefore, there is need for solutions that allow operations to be performed without delay and without endangering rescuers. To tackle this issue this thesis introduces a concept for search and rescue missions where unmanned aerial vehicle (UAV) and unmanned ground vehicle (UGV) agents are used in collaboration to map the area and to find a path for UGV to reach the target destination. Concept is possible to implement with fully or partially autonomous agents. Concept allows scaling up and down operations depending on resources. In addition to this high-level concept, this thesis introduces methods for map and path generation for UGV using depth images taken from aerial perspective of the mission area by the UAV. The algorithm for the map generation filters away false obstacles that are present in raw images from above, but do not actually form obstacles for the UGV on the ground. These false obstacles can be for example foliage of trees or powerlines. As in some situations it is important to be able to remain undetected, this mapping and path generation method is developed to support passive sensors. Path generating part of the method is modified version of A* search algorithm. This method allows adjusting of the route optimization according to traversability of the UGV used in the mission.

Keywords: algorithms, autonomous vehicles, unmanned aerial vehicles, unmanned vehicles, mapping, passive sensors.

Table of contents

1 Introduction

Throughout history humanity has faced catastrophes. Some are result of human activities like wars and some are occurring independent of humanity like earthquakes. Both examples, war and earthquakes do cause ruins, collapsed buildings and generally chaotic changes to the landscape when occurring in residential area. In these occasions there are usually also human casualties, dead and wounded. Getting help for the wounded is a race against time. However, in a warzone as well as in earthquake area, rescue operation is dangerous as there is risk for new strike or aftershock. Robots offer an alternative and aid for human rescuers for the high-risk areas. Due the cataclysm at hand, the landscape may have dramatically changed from what it was in the latest map or satellite image taken of it. Therefore, up-to-date mapping of the operation area is required if unmanned ground vehicles are to be sent to navigate to the location of wounded person in need for rescuing.

The goal for this thesis is to create a novel approach for mapping and path generation for UGV using UAV-UGV cooperation and depth sensors from above. Scope for this thesis is set within a scenario where search and rescue style operation is needed. General idea is that there is a scenario where UGV needs to navigate to target location to perform a rescue operation. Scenario is defined to be either completely without a prior mapping of the operating area or with drastic changes to prior map if one exists, and hence, lack of valid mapping per se. Therefore, a map needs to be generated. For this purpose, UAV is to be used to survey area from above to build a map for the UGV on the ground to use for navigation. To mimic real-world, it is likely that there are unexpected events occurring during the mission in the operating area that is to be mapped and thus it is to be considered as a dynamic environment. Hence there can occur changes for the environment itself or there may be objects that move in the area during the mission. Thus, initial mapping may become outdated several times during the operation. To keep the scope of the study fit for master's thesis, this study is conducted in a simulator environment. Therefore, some elements of the real-world scenarios like effects of weather are thus out of the scope of the study part of this thesis, and hence, discussed only on speculative level.

To apply robotic agents for cooperative search and rescue mission, there are plethora of aspects to consider. This thesis offers an approach for some of the critical areas of this mission while taking into consideration many of the challenges in the real-world scenarios. The approach includes a broader framework as a solution for collaborative operating model for one or more UAVs and UGVs for search and rescue missions. While the framework is comprehensive but high-level concept, this thesis also provides detailed modular method for map and path generation within the framework.

In this thesis a method to build map and path for UGV with help of UAV is proposed along with the concept of collaborative operating model framework for one or more UAVs and UGVs for search and

rescue missions. This approach solves how to map quickly an area that is not mapped or has changed drastically since it has been mapped and generate map and path on it that allows UGV to navigate through the area to target destination. Method introduced in this thesis can also be implemented for area that is dynamic and constantly changing and allows rapid update loop for the UGV path to avoid collisions with moving objects and constantly changing environment. This thesis focuses on generating map for UGV from the depth map formed from sensor data of UAV. Sensors are expected to be capable of sense depth. The method of this thesis is not bind to any specific depth sensing technique. However, different depth sensing techniques are reviewed and discussed.

In addition to map generating, the modular process introduced in this thesis includes a path generating module. This method can generate path that not only avoids obstacles but also plans path that avoids too steep slopes for the optimal traversability for the UGV in use.

To sum up the scope and the goal of this thesis goes as follows: Aim is to develop a solution to generate map for UGV with help of one or more UAVs. Solution should be applicable for areas without valid prior map. Map generation should be possible without revealing sensor use by emitting any form of electromagnetic radiation like emitting wavelengths of visible light or infra-red. Solution should also be applicable for poor lighting conditions. Optimizing path search algorithm for the novel approach is out of scope of this thesis and thus left for future studies. Localization of UGV on the map as well as localization and mapping for UAV are out of the scope for this study and thus developed algorithms do not cover solutions for these areas. Therefore, positioning UGV on the map is taken as an assumption. Goal is to produce outcome that can be applied to search and rescue missions using UAV and UGV agents though it is likely to be applicable for other mission types as well as there are similarities in fundamental requirements for most missions covering UAV-UGV collaboration.
The rest of the thesis is organized as follows:

Chapter 2. is about theoretical background and literature review for the subject aiming to give reader understanding and overview of relevant concept. It defines concepts and component that are relevant for the study and thus it is preferable to have aligned description of those for the reader. These concepts cover entities like UAV and UGV, sensors and more abstract concepts like UAV and UGV cooperation. Review for related methods is in this chapter. It also brings up the gaps of current solutions and emphasis the need for novel approach.

Chapter 3. covers the study. It introduces the concept of UAV and UGV cooperation for rescue operations in catastrophe areas with no map available that is up to date. Chapter goes into details of the map and path generation algorithm developed for this thesis and shows results from the method.

Chapter 4. covers discussion over the method analysing challenges and shortcomings of the developed method. It also weights relevance of the approach in real life environments. Discussion chapter also covers potential subjects for future studies around this topic.

Chapter 5. covers conclusion and contribution of this study.

Final part of this thesis consists of list of references used for this thesis.

2 Theoretical background and literature review

This chapter defines basic concepts and reviews earlier research literature that is relevant to the topic of this thesis. Chapter is organized to sections as follows:

- 2.1 Conceptual definitions, containing subsections:
  - 2.1.1 Unmanned aerial vehicle,
  - 2.1.2 Unmanned ground vehicle,
  - 2.1.3 UAV and UGV cooperation,
  - 2.1.4 Communication and control,
  - 2.1.5 Sensors,
  - 2.1.6 Map generation and path planning,
  - 2.1.7 A* search algorithm,
  - 2.1.8 Other relevant algorithms,
- 2.2 Review on related work,
- 2.3 Chapter summary.

2.1 Conceptual definitions

This section defines concept for key elements that are present in this thesis. With these definitions reader is given opportunity to orientate to the topic with aligned understanding of entities and their relation to each other and the research question in hand. Definitions of each concept is as comprehensive as subject matter requires. Thus, some concepts may lack areas that are not seen relevant for reader to understand the research introduced in this thesis.

2.1.1 Unmanned aerial vehicle

In this thesis an unmanned aerial vehicle refers to a machine that has capability to fly and perform airborne operations with either autonomously according to its programming or by being controlled by a human operator or by another system remotely. Thus, UAV can be considered as an aerial agent.

UAV is fit for many kinds of tasks that can be beneficial to perform from above [1], [2], [3], [4]. Movement is less limited due obstacles, though not completely free when operating in

low altitudes. This allow shortcuts and reaching locations that are unreachable for ground-based vehicles. Bird eye perspective allows surveillance of vast areas signals can be send and received with less obstacles to interrupt communication than on ground.

UAVs can be categorised in different categories according to their fundamental structure [1], [2], [3], [4]. These categories include:

- Multi-copter
- Single rotor
- Fixed wing hybrid
- Fixed wing

List gives principal description of typical UAV structure categories. In figure 1 is visualization of these categories including example of variations of multi-copter category. However, there are other UAV structure types available like blimps, glides and flapping-wing types to mention few [5]. These are not listed here as those are rather uncommon and most of the research papers published in recent years focus on variety of multi-copters. Different structural types have qualities that fit for different mission types [4]. Fixed wing UAVs need a runway to lift off and land. Wings produce lift when flying in the air with high enough speed. Therefore, hovering is not possible as the lift relies on the pressure difference over and under the wing caused fast moving air. Fixed wing UAVs are generally fit for missions where large areas need to be covered or speed and endurance is required. Drawback for fixed wing type UAV are launching and landing and generally this type is costly. Area survey is one typical mission type for fixed wing UAVs [1], [3], [6]. Unlike fixed wing types, copters can hover, take off and land vertically. Single rotor copter (helicopter) is capable to carry large payload but is typically high priced. Advantages of multi rotor copter are their availability and low price in comparison to other UAV types. However, multi-copter payload carrying capability is less than that of helicopter. Another drawback is short flight time. Although flight time is relatively short, electricity consumption per flight is also modest [4].

Rotary winged UAVs, including helicopters and multi-copters, have low energy efficiency in comparison to fixed wing and fixed wing hybrid types [3]. Rotary winged UAVs also typically rely only on battery as energy source while both winged types may have fuel and combustion engine as alternative source of energy. Therefore, flight endurance of rotary

winged UAVs tends to be less than those with fixed wings. In general battery life is one of the key challenges of UAVs and to tackle this issue solutions like charging stations and wireless power transfer have been developed [2], [3]. The latter includes photovoltaic cell-based charging using sunlight to charge batteries and laser power transfer that allows batteries to be charged with laser beam targeted to UAV's photovoltaic cells. Although some UAVs have payload capacity of several hundred kilograms, for UAVs with capability for vertical liftoff, landing and hovering like rotary winged UAVs, payload capacity is typically limited to tens of kilograms at max [3]. This is another bottleneck challenge for UAVs. Although there are less obstacles in air than on ground, battery capacity bottleneck forces communication signal strength to be limited and thus for example buildings and random Wi-Fi signals may cause interference for the communication [6].

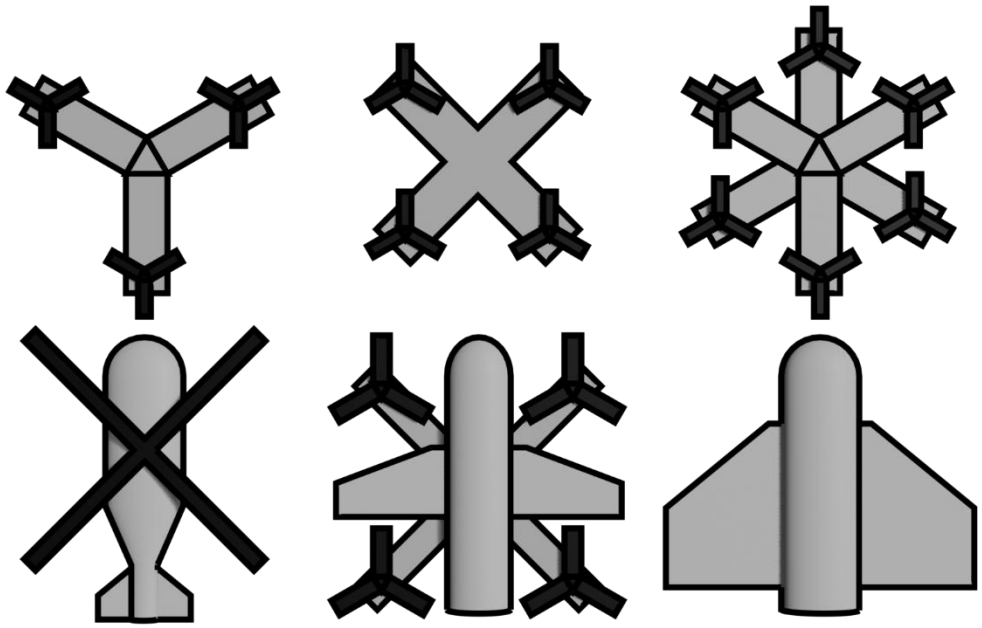

Figure 1. UAV categories based on the structure. Multi-copters: (1. Tricopter, 2. Quadcopter, 3. Hexacopter), 4. Single rotor, 5. Fixed wing hybrid, 6. Fixed wing

2.1.2 Unmanned ground vehicle

Definition for unmanned ground vehicle for this thesis is a machine that has capability to manoeuvre on ground independently or remotely controlled by human or machine operator. Therefore, similarly to UAV, UGV can also be considered as an agent operating on ground. For structured environments there is little challenges for locomotion design of UGV as simple wheeled assembly is typically apt for such a scenario. However, unstructured environment

easily causes conditions that require more sophisticated solutions for movement [7]. Therefore, UGV types can be categorized based on movement method as [2], [6], [7]:

- Wheeled
- Tracked (crawler)
- Legged

Visualization of these types can be found in figure 2. In addition to these three arch types there are hybrid models between each two and combination of all three types [7]. Tracked type is very capable to manoeuvre in wide range of environments [6]. Tracked UGV can rotate and have some level of climbing capability. However, tracked UGVs are typically slower than wheeled type UGVs. Although wheeled UGVs are faster than tracked, they are not operative in all terrains as they are more prone to sink in soil and wetlands. However, for challenging terrain legged UGVs can prove to be superior when it comes to traversability [8].

When considering and comparing UGV capabilities, following features are relevant to take in account [7]:

- Maximum speed – Speed on easy surface
- Obstacle crossing capability – How well UGV can cross random shaped obstacles like rocks
- Step climbing capability – Ability to climb stairs in structured environments
- Slope climbing capability – Ability to climb inclined surfaces
- Walking capability on soft terrain – Soft terrain like sand
- Walking capability on uneven terrain – For example on rocky terrain or grass field
- Energy efficiency – Energy efficiency on variety of terrains

In addition to listed features, other features that are to be considered when considering UGV design are mechanical complexity of the system, complexity of control and technological readiness referring to the maturity of the solutions used for the design of the system.

As UGVs travel on ground, they can be built to carry heavy loads [6]. Therefore, batteries and other equipment can also be heavier than for UAVs allowing UGV to endure longer and complete variety of tasks on the ground. Payload capacity and large energy supply also

enables UGV to be equipped with significant amount of computational power. Although UGVs can be equipped with variety of tools and sensor due load carrying capacity, it may have limited sight due terrain and buildings. Environment can also limit UGV's movement radically.

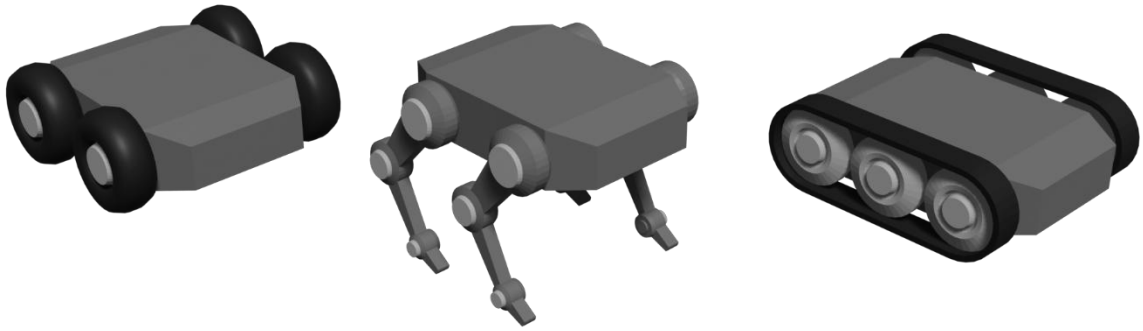

Figure 2. UGV categories based on the structure. 1. Wheeled, 2. Legged, 3. Tracked.

2.1.3 UAV and UGV cooperation

Unmanned vehicles capable of completing tasks, agents, can operate in cooperation with each other forming groups or swarms [9]. These cooperation models allow agents to carry out tasks that would be impossible for individual agents alone. Cooperation can take place between similar agents like in swarms of drones (UAVs) or within groups of heterogenous agents like between UAV and UGV collaboration. Although, in this thesis only UAV and UGV concepts are defined and described it to be mentioned that there are also agents build for other environments like unmanned surface vehicle (USV) to operate on surface of water and autonomous underwater vehicle (AUV) to operate below the water surface [9]. However, as this thesis is about UAV-UGV cooperation, remaining of this subsection focuses on this specific form of heterogenous agent collaboration.

Both UAV and UGV have strengths that can complement operational capability of other [8], [10]. Therefore, cooperation of UAV and UGV can provide solution for some of the shortcomings which both systems face when operated independently. UAV is apt for moving

quickly around with only a modest need for consideration of physical obstacles or terrain [2], [8]. It is great for surveillance and mapping areas from above. UAV is not best suited for carrying heavy loads or executing tasks that require manipulation of the environment. Power consumption along with modest payload capacity limits battery mass that UAV can carry and thus limits endurance causing a bottleneck for the use of UAVs. However, as UGV can be loaded with larger burden, in UAV-UGV collaboration UGV can be equipped with large batteries and charging station for UAV [11], [12]. This is one way to prolong operational endurance of UAV.

Typical mission types for UAV-UGV collaboration include [1], [12]:

- Search and rescue
- Environmental monitoring
- Precision agriculture
- Infrastructure inspection
- Surveillance and reconnaissance

All these mission types have specific features. For search and rescue missions UAVs offer solution for fast scanning of the area after disaster as they can pass many obstacles that limit traversability for UGV. With aerial surveillance, UGVs can be led to target locations efficiently and efficaciously [2]. Thus, aid and supplies will reach destination faster carried by UGVs with mitigated risk from debris and hazards. UAVs and UGVs can be equipped with special sensors like lidar and sensor for thermal imaging that help searching for targets. This thesis is about map and path planning for UAV-UGV collaboration in search and rescue missions and thus the focus is on features concerning this mission type and therefore this is given more attention. However, to understand key features of the collaboration in general, it is good to have brief walkthrough for other mission types as well.

All mission types have in common to take advantage of UAV's aerial ability that allow fast movement and ability to monitor vast areas from above and UGV's ability to operate with specialized equipment or to carry more load. This is also the case in environmental monitoring. From air, UAV can monitor large area and point areas of interest. UGV can then move to analyse these areas more in depth with special gear that it has been equipped with. For precision agriculture top level concept is similar. UAV searches points of interest as it can rabidly monitor large areas. In this case point of interest can be area of crop with pests, area in

need of fertilizers or area in need of irrigation. Next UGV is sent to deal with the issue in hand taking advantage of UGV's carrying capacity and ability to be equipped with special gear. This concept allows narrowing down areas of crops that neep special attentions and therefore, limits need for pesticides, fertilizers irrigation, human labour and thus it limits costs. UAV-UGV collaboration for infrastructure inspection follows same basic principles that other mission concepts do. UAV inspects infrastructure from above making monitoring of condition infrastructure like power grids, dams and other relevant building faster and easier than it has been in the past. Once some issue has been detected, UGV can be sent to solve the problem in hand. [2]

Last mission type listed above does differ a bit though on high level principle remain close to those of the other collaboration mission types. For formation control in military and defence missions, UAV is used for getting aerial perspective of the situation [2], [13]. However, instead of simply tracking points of interest like in previously introduced mission types, in formation control UAVs and UGVs spatial distribution is optimized in dynamic situations for each unit to be able to perform its task and to support other. This is achieved with modern algorithms. However, using formation control algorithms for heterogenous system like UAV-UGV is more challenging than it is for homogenous system like UAV swarm as in heterogenous system motion models of agents differ from each other [9].

Like in all above missions, also in surveillance and reconnaissance missions UAVs are used for their wide view from aerial perspective to cover large areas. However, for this mission type UGV is used to complement observations from UAVs. UGV has different perspective in comparison to UAV and combining these aspects allows more accurate and reliable monitoring and recognition of persons and tracking of the objects. [2]

One corner stone for UAV-UGV collaboration is map generation for UGV. Using aerial view UAV can capture large area of ground below at once. This combined with fast movement speed of aerial agent, allows superior premises for map generation in comparison to UGV alone. Although premises are great, real-world cast challenges for mapping. Weather conditions may be challenging for visibility and thus be obstacle for capturing the ground image. In addition to visibility methods for image processing can be computationally challenging for the hardware in use. [14]

Though cooperation between UAV and UGV has potential for many advantages in comparison to homogenous agent collaborations and solo agents, there are also limitations

and challenges. Cooperation is vulnerable to communication and coordination issues [9], [14]. These can be caused by challenges in communication like problems in quality and poor response speed. These can affect to coordination of the cooperation between agents. However, coordination can be challenging also due heterogenous movement capabilities of the agents which is more complex to handle than in homogeneous collaborative systems. Although ensuring UAV-UGV collaboration is functional can be complex, potential applications do emphasize importance of developing solutions and methods this form of collaboration between agents.

2.1.4 Communication and control

Communication in general is vital for any kind of collaboration. This applies also for collaboration of unmanned agents. If wireless communication breaks down or suffers from delays, control and cooperation between agents is affected [14]. In missions like search and rescue, where speed of achieving results is essential, it is critical to have working communication between agents and operation command [15]. Collaboration of agents can be arranged in centralized or decentralized manner [6], [9]. In centralized approach all agents are communicating and organized by command centre. Although command centre can be specialized entity, it is not necessary as this role can also be set for one of the agents in the operative group. Thus, all agents are communicating with this specific agent. Idea of decentralized orchestration is that each agent communicates directly with others as equals. Figures 3 and 4 demonstrate different orchestration of communication in centralized and decentralized collaborations.

Having centrally orchestrated agent collaboration allows command centre to have an overview of the operation and optimize formation of the agent swarm. This has been common approach in the past. However, there are several weaknesses when control of collaboration is centralized. First, optimizing entire swarm formation is computationally heavy burden which prone to prolong delay, and thus casts challenge to operate in real-time. Secondly, spatial scalability is limited as distance to command centre and outermost agent is limited to communication capability between these units. Thirdly, centralized collaboration system is vulnerable to communication interference whether those are intentionally caused or occur for example due barriers caused by landscape. Loosing command centre is fatal for centralized

system. However, decentralized orchestration is stronger in the weak areas of centralized system. As each member of the swarm communicates with each other, it is enough to reach some other member of the group to operate. Therefore, this approach is more resilient for communication interference and losing any individual agent is not necessarily devastating blow for the operation. [9]

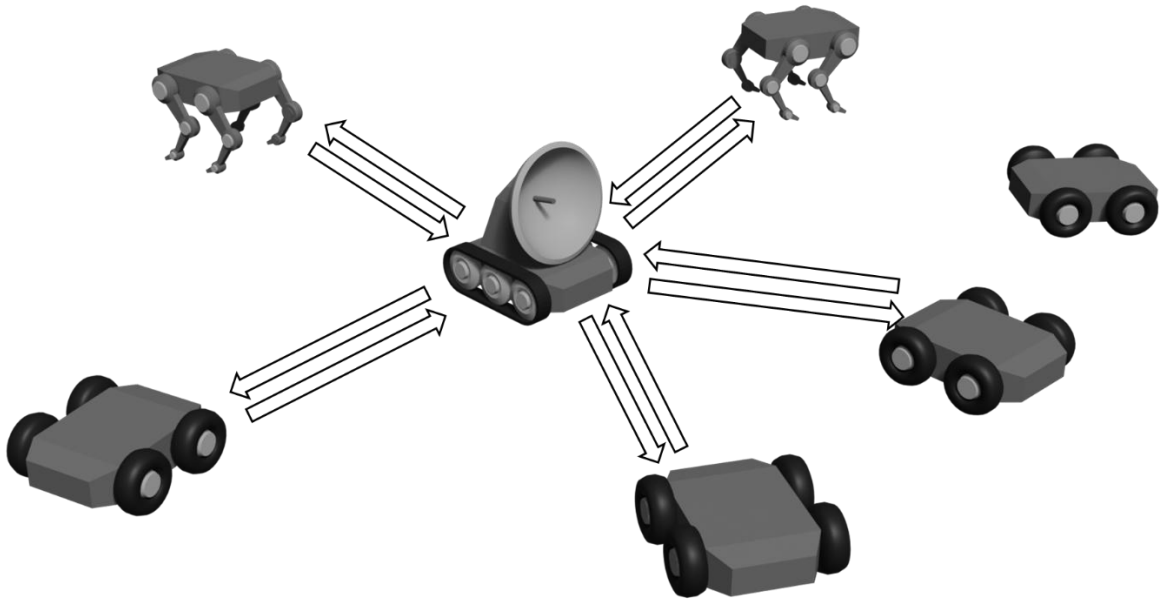

Figure 3. In centrally orchestrated agent collaboration agents communicate with command centre that is responsible for coordinating the collaboration.

When communication channel between agents is operational and data transfer capability is efficient, it can used also for other purposes besides coordinating collaboration [8]. It enables sharing computational resources between agents. Therefore, UAV with sensor capabilities does not need to carry powerful computation unit as a payload needed for further processing of the sensor data as this processing can be executed by UGV with powerful computing hardware. However, capacity of wireless data transfer falls as distance grows [15]. Therefore, it is to be considered case by case whether it is more proficient for example to transfer raw images from UAV to be processed in other agent or to do part of the processing like down scaling before data transfer to optimize communication speed and reliability.

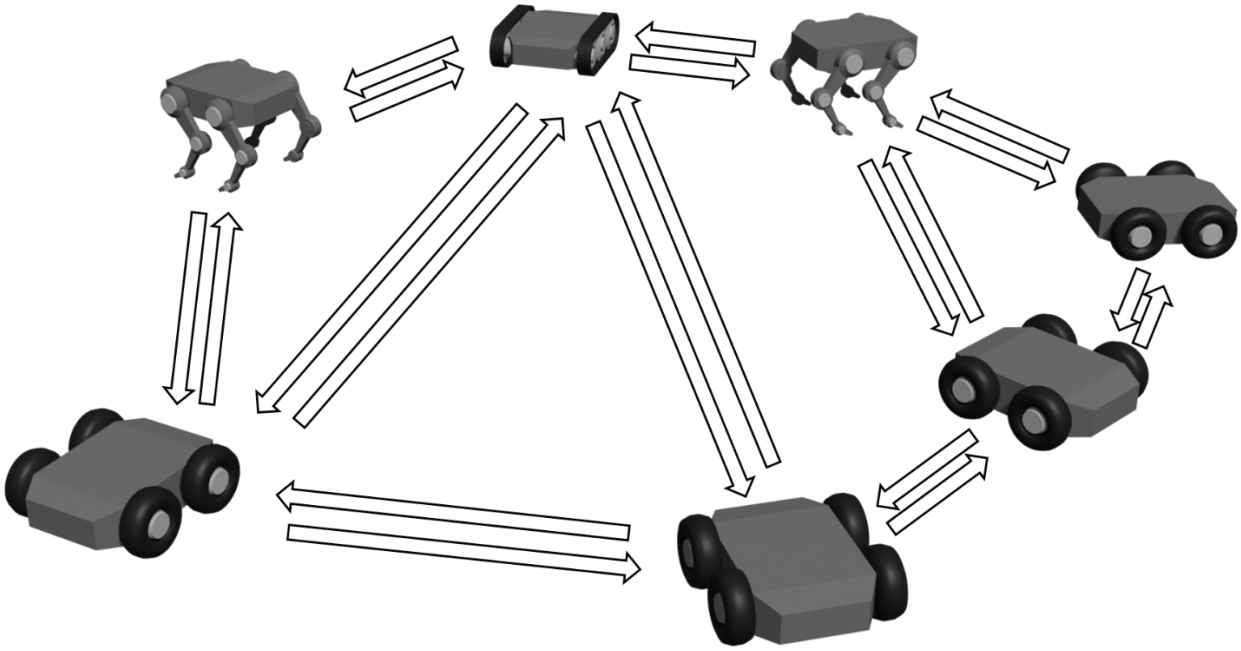

Figure 4. In decentralized orchestration agents communicate with peers and collaboration coordination takes place locally with nearby agents.

2.1.5 Sensors

Common sensors used in robotic systems cover a wide range of different technologies [4]. With sensors, robotic systems can be fed with information about the surrounding world. This information can then be refined and used as an input on which a robot can plan an action or react on, according to its programming. For the scope of this thesis, sensors that provide information of location of objects and form of surfaces are the most relevant ones. These sensors include:

- Visible light sensors like red, green and blue (RGB) cameras
- Infrared (IR) cameras
- Depth cameras
- Lidars
- Radars

Although, it can be argued that there are other sensor types that are used for surface detection like sonars, only the listed sensor categories are included to the scope of this thesis and narrowed down even further later for the developed method.

In past orbital satellites were best in class for remote sensing purposes from above [4]. However, satellites are expensive systems to build and get in to orbit. Upgrading satellite sensors is not possible in most cases. Therefore, modern UAVs offer more cost-effective option for many remote sensing tasks. In addition to significantly lower costs, UAVs are also more flexible in operating and can be used to avoid some pitfalls of satellite capabilities. Satellites operate from vast distance above. This causes images to show relatively low resolution for object on ground even though the camera sensor is ultra-high resolution. Therefore, UAV closer to target, can get more detailed images with camera with lower resolution. Satellite view is prone to be blocked by clouds, though synthetic aperture radar also known as SAR can see through clouds [16], [17]. However, UAV though not entirely invariant to weather conditions, is able to operate under clouds and thus capable to observe target area in more robust manner than satellite.

For UAV's common sensors include visual sensors, spatial sensors, radars and gas sensors [4]. For this thesis the first two categories are most relevant though radars could also be used for environment sensing in some degree. Visual sensors include RGB cameras, multi-spectral cameras and hyper-spectral cameras. Common RBG camera types used in UAV are high resolution integrated camera and single lens reflex (SLR) camera. Former can capture high resolution images with high frame rate. These cameras are typically light weighted and thus apt for UAVs. Although this type of camera offers good solution for aerial imaging it only has one lens option. Therefore, SLR camera which can have different lenses for different occasions is solution when for example tele lens or wide-angle lens is needed. However, SLR is often heavier than integrated high-resolution camera. Hence, optimal RGB camera type depends on the mission in hand.

Multi-spectral and hyper-spectral cameras are visual sensors for special purposes like cartography of vegetation and analysing material compositions [4]. Thus, these sensors are used in fields like agriculture, geology and archaeology. Both camera types can capture image data from several wave lengths of light spectrum including light not visible to human eye like infrared. From these two types of cameras hyper-spectral camera is capable to capture wider range of narrow wavelength bands than multi-spectral camera. It also captures these in contiguous manner while multi-spectral camera does this in discrete form. However, hyper-spectral camera is often more expensive than multi-spectral camera.

Infrared sensors are visual sensor that capture electromagnetic radiation that has wavelength in infrared zone [18]. Sensor can be divided to near IR and far IR sensors. Former sensors capture light that is close to spectrum that is visible to human eye while later is further away. Practical applications for IR sensor include low light vision as near IR can enhance vision when light is low. Far IR can be used to detect specific heat trails of target objects. IR sensing is less sensitive to lighting and weather conditions in comparison to other visual sensing methods. IR sensors are also low cost. However, IR sensors tend to have low resolution and those lack colour, texture and depth information. Like other visual sensors also infrared sensors are passive sensors. This means that sensor is not actively emitting energy, for example directed pulses of electromagnetic radiation to be able to receive information of the surrounding world [19]. Therefore, passive sensor relies on information provided by externally emitted energy, for example sunlight bouncing from objects and is then received by sensor. This information is then used to form visualisation of the world.

While UAVs have bird eye view and can use sensor for large area survey, UGVs are ground bound and therefore, they use sensors to perceive their local environment [18], [20]. Requirement for sensors relates to purpose of the use for UGV. For example, self-driving cars need sensors that are applicable for detecting other vehicles and other potentially moving objects nearby and sensor for comprehending build environment like roads and buildings. However, offroad environments are unstructured and terrain is typically uneven. Also, vegetation is blocking visibility and variety of obstacles is abundant. Therefore, UGVs that equipped for offroad mission like for military purposes can have different emphasis for optimal sensors in comparison to the ones that are built for urban scenarios.

For UGVs list of typical sensors does include many sensor types that are common also for UAVs like visual sensors, spatial sensors and radars [18]. However, there are still difference in use case with many sensors as point of view is different. For example, monocular cameras with RGB sensors are used for sematic segmentation and detection of typical objects within the detection range. It is good for detecting details and textures and it is cheap and light sensor to install to UGV as it is to UAV. However, it does not give any spatial information alone and is prone to weather conditions and thus it is typical for UGV to have other sensors in addition.

Both, UAVs and UGVs can be equipped with spatial sensors [4]. These sensors can sense spatial dimensions of the world and therefore, these sensors are used for 3D mapping of the environment and objects. With these sensors, distance between target and observer like UAV or UGV can be defined accurately. One of the spatial sensor types is laser imaging, detection, and ranging sensor commonly known as lidar. Lidar operates by sending laser beam and measuring time it takes for the beam to bounce back from the target surface. Hence a 3D map can be formed by aggregating data from the measured points to form a point cloud.

Lidars used are typically either mechanical scanning lidar or more modern solid-state lidar [4], [21]. In the former prism or mirror rotates causing laser beam to scan different direction as it rotates. Latter has a laser or an array of multiple lasers directed to the target. Solid-state lidars are smaller and lighter in comparison to mechanical scanning lidars. This makes solid state lidars better fit for UAVs. Lidar is active sensor type as it casts laser beam and therefore, it can be detected and located which is to be taken into consideration for military solutions [18].

Point cloud formed by lidar is dense and thus it can form detailed 3D scan of the environment [18], [20]. Lidar is invariant for lighting conditions as it relies on the light bouncing back from the laser beam it has cast. However, lidar is sensitive to whether conditions and fog and rain can limit its performance. Use of lidar is also affected by vegetation as laser is bouncing off from foliage. Radar is based on similar principle as lidar. However, it relies on radio waves instead of laser. Radio waves penetrate more material depending on the wavelength than laser and therefore with radar, it is possible to prolong vision range of the agent. This is useful especially for UGV as its vision range is often limited due nearby obstacles. Therefore, using radar can be complimentary for lidar. As radar emits radio waves it acts as active sensor like lidar and thus it can also be detected and located [18].

Typically, lidar has more dense point cloud than radar in general. However, for radars point cloud density is dependent on the wavelength of the radio waves. With short wavelengths it is possible to achieve higher resolution. However, these wavelengths are more prone for noise. Long wavelengths on the other hand have less resolution, but are capable of deeper penetration, thus allowing to see through obstacles. Unlike lidar, radar is not only robust for lighting differences but also for weather conditions, Therefore, sensor fusion of radar with

other sensor types including lidar is a prominent field for studies regarding sensors for UGVs and other agents. [20]

Alternative approach to lidar and radar for spatial imaging is multi-view stereoscopic imaging (MVS) or stereo vision. MVS is fundamental problem in computer vision, and it is common method used for robotic navigation [22]. Advantages for this method are lower cost and simpler data processing in comparison to lidar system [4]. Method is based on images taken from multiple angles and used to generate detailed 3D mapping of the target area and objects. Instead of using stereoscopic camera specifically build for the purpose, MVS system can be built by using several regular RGB cameras attached to UAV or UGV [18]. However, processing images require algorithm that can generate spatial information from the combination of the images. MVS algorithms triangulate 3D positions of the objects in the view based on the basic assumption that same spatial point in the target scene is observed from slightly different angle and location by different cameras [22]. MVS methods can be categorized in traditional methods and methods that are based on deep learning. Traditional methods can be categorized to four categories according to what each approach is based on. These categories are as follows:

- Volumetric
- Mesh
- Point-cloud
- Depth map

First three are representations that visualize the scene directly. However, depth map representation is a fusion of multiple depth estimates that addresses 3D problem into 2D depth map [22], [23]. Therefore, depth map methods are robust and can give good results. However, traditional MVS methods are not optimal and fall short with surfaces that lack texture and that are non-Lambertian. Therefore, other approaches are needed to tackle these challenges. Deep learning based MVS methods can solve these problems. However, deep learning model for MVS are constrained by relatively large memory requirements.

Other relevant spatial imaging techniques that are applicable for mobile agents in addition to the ones described above include time-of-flight (TOF) and structured light approaches (SL). TOF is based on emitted light being reflected back from the target surface. As speed of light

in medium is known constant, distance of target surfaces can be calculated according to the duration between emitting light and it returning to the sensor. Emitting can be single pulses or continuous wave modulation. TOF has several advances. It is faster than lidar as it captures the entire view at once instead of structuring it from point cloud. It is computationally lighter than MVS and does not require textured surfaces. As it emits light it is robust for poor lighting conditions. [24]

Like TOF also structured light technique is based on emitted light [24]. However, in SL method range measurement is not based on time, but for triangulation like MVS. Whereas MVS relies on external radiation like visible light or IR, SL has projector and camera combination that forms active stereo pair. Projector emits light with patterns like stripes on target. As camera sees the scene from lightly different angle pattern are shown to be distorted according to the spatial shapes of the objects that they are cast upon to. Like lidar and radar, also TOF and SL technologies are active as they emit radiation. Therefore, MVS is the only passive range sensing technology reviewed here.

In addition to sensors described above, other sensors used for UGV and UAV include omni-direction camera, event camera and ultrasonic. Omni-direction camera is used to capture surrounding view at once. Processing of produced data is computationally demanding which limits its applicability. Event camera is specialized camera for very dynamic scenarios. Ultrasound emits soundwaves that can be used for obstacle detection to avoid collision. However, operating range is very low. [18]

2.1.6 Map generation and path planning

For robotic system to move autonomously from initial position to target position, it requires directions how to activate its mobility system [25], [26]. These directions can be based on information generated by a path planning process. For UGV systems, path planning can be split into global path planning and local path planning. Global path planning is typically based on assumption of static overview of the target area [26]. However, real world is rarely static and thus local path planning is adaptive approach that complements path planning process by providing more real-time information of dynamic environment around UGV. Another thing to be taken into consideration while planning path is traversability [27]. Therefore, planning a path in urban area differs from planning a path to off-road environment.

To be able to follow the path autonomously robotic agents need be able to position themselves on map [28]. However, map is not always available, nor it is up to date when available. To effectively generate a map, a swarm of UAVs can be utilized instead of just individual agent [29]. Mapping and localization are interconnected problems, as localization is not possible without map and map is useless if location is not known [28]. Thus, both issues are to be solved for complete system to be operational. One profound concept to solve localization and lack of map is SLAM, Simultaneous Localization and Mapping [28], [30]. As name suggest, this concept is aiming to tackle both localization and mapping issues. This concept has been foundation for numerous studies in field of robotics during past decades. SLAM concept can be applied to 2D and 3D mapping. Although SLAM concept is for agent to orient to its position in the world, it can also be adopted for multi agent collaboration systems. However, using SLAM in multi agent systems does make it more complex and computationally heavy.

In recent years SLAM systems have been introduced for UAV-UGV collaboration that are able to autonomous localization and mapping even in GPS-denied environments [31]. Modern SLAM research uses sophisticated deep learning solutions for various purposes including removing dynamic objects when generating 3D map of static environments [30], [32].

SLAM methods can be divided into single sensor-based methods and multi sensor methods [30]. Former can be further divided into visual based and lidar based approaches. Multi sensor-based approaches are based on sensor fusion. During recent years, number of studies conducted on subject relating to visual based SLAM or VSLAM has been increasing [33]. This is likely due to the low cost of sensors and the fact that visual sensors do capture abundant information of textures of the environment. Therefore, visual sensor data offers also fruitful source for sensor fusion.

Having map of the environment and position of the agent on the map is first step in navigating towards target. However, to be able to move to the target a path must be planned. Path planning methods can be divided into three main categories [34]:

- Classic
- Heuristic
- Artificial Intelligence (AI) based

All of these high-level categories cover plenty of methods. Approaches categorized as Classic methods are:

- Artificial Potential Field (APF)
- Sampling Based Approaches (SBA)
- Graph Search Based Approaches (GSBA)

In APF method the agent is navigated towards the target with by a potential function. While the target location is pulling agent closer, obstacles are repulsing agent further away. Therefore, agent is approaching goal and simultaneously smoothly avoiding obstacles. However, vanilla version of APF suffers from local minima issue and agent can end up being trapped in local minima and thus not be able to reach destination. Although APF is computationally light, it is not optimal for areas of large scale.

Sampling Based Approaches cover Rapidly Exploring Random Trees (RRT) and Probabilistic Roadmap (PRM) methods. Advances for RRT are that it can be used for multidimensional and complex spaces efficiently. However, RRT can be computationally expensive to use for small adjustments to the path as it requires recalculating entire path always when an obstacle occurs to already planned path. As RRT also PRM is suitable for multidimensional scenarios. However, PRM suffers from high computational requirements for the initial path planning and is not optimal for dynamic environments. [34]

Graph Search Based Approaches cover Breadth First Search (BFS), Depth First Search (DFS) and Best First Search methods. Best First Search covers Dijkstra and A* search algorithms. Strength of BFS that it does find optimal solution while being simple to implement. However, though BFS can find optimal path, it is very costly. Memory requirements are very high, and computational cost increases rapidly as scenario size grows. On the other hand, DFS has low memory consumption, but may not find optimal path. Djikstra can find optimal solution, and it is applicable to variety of scenarios. However, as BFS also Djikstra is strained by memory consumption and computational complexity. Thus, it is not optimal for dynamic environments. [34]

The second approach in the high-level categorization is about heuristic methods. This category includes [34]:

- Tabu Search (TS)

- Bio-inspired Approaches (BIA)
- Clustering Algorithm (CA)

Tabu search is flexible and can be used for complex problems. It utilizes memory to be able to avoid getting stuck in local minima [35]. However, it is complex to implement and it and sensitive to parameter set up. It is also prone to be computationally costly to use. Bio-inspired approaches cover methods like Particle Swarm algorithm (PSA), Ant Colony algorithm (ACA) and Genetic algorithm GA [36]. PSA converges quickly and is applicable for distributed systems and easy to implement [34]. However, like TS, also PSA is very sensitive to parameters. In some cases, PSA also converges too quickly and thus jams in local minima. ACA can be effectively used for optimization, and it is applicable method for distributed systems [34]. However, like other heuristic methods also ACA is sensitive to parameter selection. It is not optimal algorithm for large scenarios, nor does it converge with haste. GA is inspired by inheritance of genes. Chromosome presents a search path and evolution process includes mutation and crossover processes. Unlike ACA, GA can be used for large scenarios. It can also avoid local minima problem better than PSA and is applicable to dynamic scenarios. However, along with other mentioned heuristic method parameter sensitivity is characteristic for this method too. GA converges slowly while requiring high computational resources. In addition to these bio-inspired algorithms there is Aptenodytes Forsteri Optimization Algorithm (AFO) which has shown to be potential in engineering problems [37]. Algorithm is inspired by emperor penguin behaviour to tackle winter conditions. Last of the heuristic approaches introduced here is Clustering algorithm [34]. This algorithm is simple to implement and reduces the complexity of the problem in hand. Therefore, it can be used for large scale scenarios. However, ill-defined clusters may result in non-optimal path. Another challenge with CA is that the algorithm is not optimal for dynamic scenarios.

The third and the final part of the pathfinding approach introduction is covering methods based on AI solutions. These solutions cover following types:

- Fuzzy Logic (FL)
- Neural Networks (NN)
- Reinforcement Learning (RL)

Fuzzy logic algorithm offers a robust solution for path planning with smooth and natural path as an outcome. However, it suffers from complexity in rule planning and can endure challenges if robot count increases. Well trained neural networks can be adaptable, scalable and able to solve complex path finding problems. However, training requires large amounts of data that is applicable to path finding problem at hand. Training and applying of model can be computationally costly. Reinforcement learning allows adaptability for the environment and thus optimizing on-the-ground. However, initial training can require numerous iterations. Although adaptability to environment at hand is positive quality in general, challenge is to balance between learning new and applying already learned policies. Reinforcement learning can also be computationally costly solution. [34]

All the high-level path finding concepts briefly introduced above, are applicable as stand-alone methods. However, there are numerous papers covering hybrid algorithms that combine methods in search of a improved applications. [34]

2.1.7 A* search algorithm

This work utilizes modified A* search algorithm as a part of the developed method. Therefore, wider and more in-depth description of A* algorithm is relevant to achieve aligned conceptual meaning between author and reader.

A* search algorithm is one of the most referred algorithm in studies concerning path planning algorithms [34]. A* is based on Djikstra algorithm [25], [38], [39]. It categorises nodes to three categories, discovered, pending discovery and not discovered. Therefore, practical implementation for A* algorithm has open set for nodes that have not yet been check but are possible next steps on the path from start node to goal, closed set for nodes that have already been checked and a map in which cost of access from one node to another has been recorded.

Although A* is categorized in this work to classical approaches, the algorithm does apply heuristics, and therefore, it is different from Djikstra. A* algorithm calculates

$$\boldsymbol{f(n) = g(n) + h(n)}$$

where ***f(n)*** expresses estimated cost for the path from the start location to the target location when travelling through node ***n***. In this function ***g(n)*** expresses the cumulative cost from the start to current node ***n*** and ***h(n)*** estimated cost from node ***n*** to the target location. In the

formula, ***h*** means heuristic formula which is set to evaluate the cost to reach the target destination. [25], [40], [41]

A*

```
FUNCTION reconstruct_path(came_from, current)
        total_path = {current}
        WHILE current in came_from:
                current = came_from[current]
                total_path.prepend(current)
        return total_path

FUNCTION A_Star(
        start,                          # Start location for the path
        goal,                           # Target location for the path
        heurestic_distance):            # Function to estimate the shortest path between start and goal

        # Initialize A* search
        open_set = priority_queue()
        open_set.push(start, heuristic_distance(start, goal))

        came_from = an empty dictionary
        g_score[start] = 0
        f_score[start] = heuristic_distance(start, goal)
        WHILE open_set is not empty:
                current = open_set.pop_lowest_cost_node()

                IF current == goal:
                        RETURN reconstruct_path(came_from, current)

                # Check neighbours
                FOR each neighbour in get_neighbours(current):
                        new_cost = g_score[current] + cost_between(current, neighbour)

                        # If better path found, update
                        IF new_cost < g_score[neighbour]:
                                came_from[neighbour] = current
                                g_score[neighbour] = new_cost
                                f_score[neighbour] = new_cost + heuristic_distance(neighbour, goal)
                                IF neighbour IS NOT in open_set:
                                        open_set.push(neighbour, f_score)

                END FOR

        END WHILE

        RETURN None                     # No path found
END FUNCTION
```

Figure 5. Pseudocode for A* search algorithm [38].

In A* algorithm, for each iteration of the main loop, the node with the lowest ***f(n)*** value in the open set is removed from the open set and added to the closed set [38]. If the node is the goal node the path is found and it can be reconstructed with data recorded in the cost map. If it is not the goal node, algorithm will implement following for all of its neighbours:

- Skip neighbour node if it is on closed list
- Calculate cost of moving through this neighbour

  $$\boldsymbol{g(n) + cost(from\ n\ to\ m)}$$

  where ***n*** is current node and ***m*** is neighbouring node
- Compare which neighbour node has lowest cost and keep that one
- Add neighbour to the open set if it is not yet there

If open set is empty before goal node is found, the path search has failed. Thus, either there is no path available or it could not be found with given parameters. Although, closed set is considered to be part of the typical data structures of A* implementation, it does not need to be declared in the code [38], [39]. If code does not revisit nodes once those are removed from the open set it acts as moving visited nodes to closed set. This applies to the example pseudocode in Wikipedia which is shown in figure 5.

Selecting optimal heuristic function for A* search depends on the use case. If path search takes place within a grid, Manhattan distance function is apt option as it calculates distance between current and goal nodes by summing up x and y axis steps that are required to connect these two nodes in optimal case. Formula for Manhattan distance is:

$$\boldsymbol{h(n) = |x_1 - x_2| + |y_1 - y_2|}$$

where $\boldsymbol{x_1}$ and $\boldsymbol{y_1}$ present location of the current node and $\boldsymbol{x_2}$ and $\boldsymbol{y_2}$ location of the target node. Manhattan distance works well in grids where moving options are limited to up, down, left, and right. However, if moving is not limited to grid along x and y axis, Euclidean distance function offers approach that always gives the direct and thus shortest distance between current and target node. Calculating Euclidean distance goes as follows:

$$\boldsymbol{h(n) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.}$$

Like with Manhattan distance formula, also here x_1 and y_1 present the start location while x_2 and y_2 present the target location. Manhattan distance and Euclidean distance are visualized in figure 6. [41]

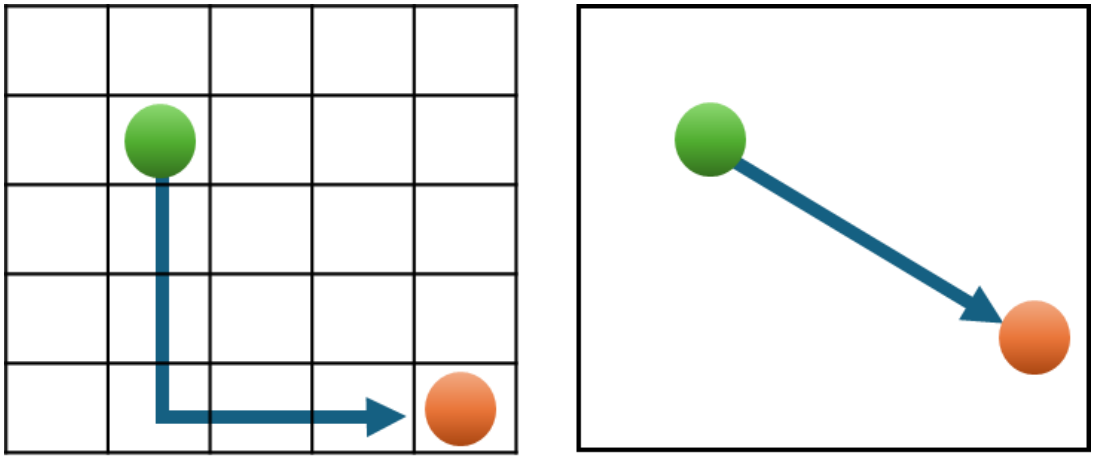

Figure 6. Left picture is an example of Manhattan distance in grid map and right picture is an example of Euclidean distance in scenario where moving directions are free.

In addition to Manhattan and Euclidean distance other heuristics that can be applied for A* search include Chebyshev and octile distance. While Manhattan distance counts cost by combining distance on x and on y axis it causes A* algorithm to approach target node on grid map by moving either on x or on y axis. Thus, next node is one of four neighbours of the current node and therefore, this forms staircase styled pattern to the path when going diagonal direction. Following such path can cause jerky motion for UGV as it will face sharp turns repeatedly. To mitigate these sharp turns alternative heuristics like Chebyshev and octile distance can be used to augment possible directions from 4 to 8 neighbours as these approaches allow diagonal distance valuation. Chebyshev gives equal cost for each neighbour orthogonal and diagonal to current node. Although this flattens sharpest edges on path, valuating orthogonal and diagonal neighbours similarly causes anisotropy. Octile distance is otherwise similar to Chebyshev distance except for the cost of diagonal movement like demonstrated in figure 7. Instead of having cost of 1 for each 8 directions, octile distance valuates diagonal neighbours with cost of $\sqrt{2}$. Therefore it maintains isotropy. [41], [42]

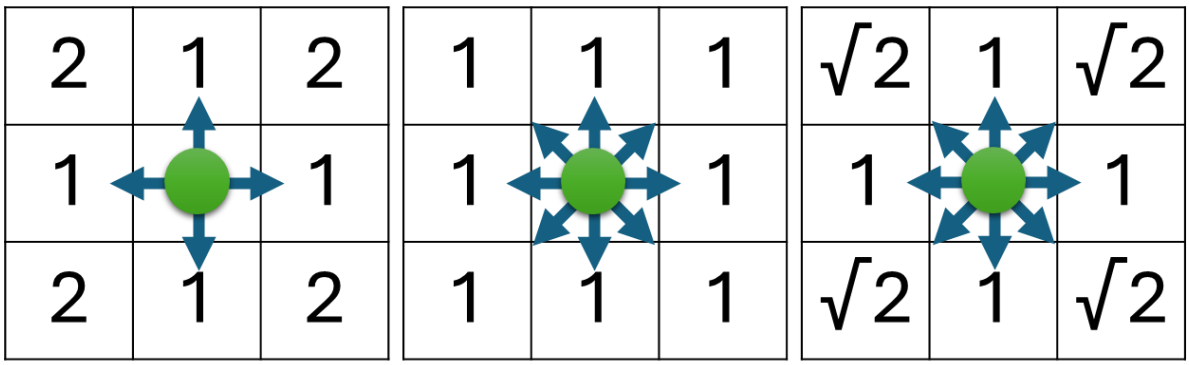

Figure 7. From left to right Manhattan, Chebyshev and octile distances and respecting available moving directions for each approach.

2.1.8 Other relevant algorithms

The method introduced in this work has several algorithms embedded in. Therefore, this subsection is dedicated to give reader understanding of these algorithms.

Inpaint refers to techniques used to restore images in general, and it has been used for physical paintings for centuries. In digital era, inpaint can be used to restore digital images that are corrupted or have unwanted elements that need to be removed. This is done by replacing unwanted areas with information from the surrounding areas. The inpaint algorithm used in this work is from OpenCV library. OpenCV has two algorithm options for inpaint: TELEA and NS. Former is based on the paper "An Image Inpainting Technique Based on the Fast Marching Method" by Alexandru Telea and latter algorithm is developed from paper "Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting". [43], [44]

TELEA algorithm starts inpainting process from the boundaries of the region that is to be inpainted and gradually moves in. It is based on a method called Fast Marching Method (FMM), a paradigm that starts building solution from known information and then goes further. Therefore, TELEA uses known pixels at the border of the unknown, masked area that is to be inpainted, to get best available information to use to fill this unknown area. For each pixel in the area that is to be inpainted, TELEA takes normalized weighted sum of values of pixels in neighbouring area and uses this to replace the target pixel. Weights are emphasized on pixels that are close to the target pixel, pixels that are close to the normal of the masked

area border and pixels that are on border contours. This process is then iterated for each masked pixel. [43], [44], [45]

NS algorithm is a heuristic algorithm that uses partial differential equations and fluid dynamics to determine pixel values for the masked area to fill in inpaint process. This algorithm aims to ensure continuous flow of brightness patterns, isophotes, from the know areas of image into the masked area. Algorithm proceeds along border of the masked area of an image. While it advances, it is using information from the know side of the border to fill the unknown side of the border. Algorithm maintains flow of isophotes inside this unknown region of the image aligned to the brightness patterns of the known area of the image. [43], [44], [46]

In paper "Comparative Study of OpenCV Inpainting Algorithms" TELEA, NS and two versions of Rapid Frequency Selective Reconstruction method are compared for quality of the outcome and speed of the implementation [44]. According to the study, when processing time is preferred to be short, TELEA offers best quality. Algorithm in this study is developed for search and rescue missions where speed is priority. Therefore, TELEA is chosen to be the inpaint algorithm for this study.

Edge detection algorithms are used to extract object edges and textures in digital images [47], [48], [49]. Edges are regions in image where colour or intensity rapidly change. Therefore, algorithms that identify these changes are considered to be edge detectors. Two edge detection algorithms tested in this work are Sobel and Canny.

Process for edge detection for both edge detection methods, Sobel and Canny, starts by converting input image to grey scale [47], [48], [49]. This simplifies process ass only intensity of a pixel is used for search of discontinuities instead of using both, colour variety and intensity. Both methods, Sobel and Canny, use kernels to process image by amplifying or by attenuating neighbouring pixel intensities. Sobel uses following square matrixes also known as kernels for convolution operation [48], [49], [50]:

$$\begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}, \qquad \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

First one of these kernels is used for horizontal convolution and second one for vertical. Although vertical kernel here has positive values in top row and negative values in bottom row, originally vertical filter top and bottom rows were other way around [50]. This was due

assumption that y axis values are read from bottom to top order. However, computer graphics solutions typically read from top to down and thus kernel presented here is in order suitable for this approach. Given an input image P, G_x and G_y represent images resulting from convolutional operation with kernels above conducted along x and y axis respectively [48], [49], [50], [51]. Therefore:

$$G_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} * P, \qquad G_y = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * P$$

where symbol * is convolution operator. In resulting images G_x and G_y every pixel represents an approximation of the horizontal and vertical derivatives of the P respectively. From these two images magnitude of the gradient for each pixel can be calculated with following formula:

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

and the angle of orientation of the edges can be calculated as follows:

$$\theta = arctan\left(\frac{G_y}{G_x}\right)$$

contributing to detection of edges. Figure 8 shows results of Sobel algorithm used on aerial image.

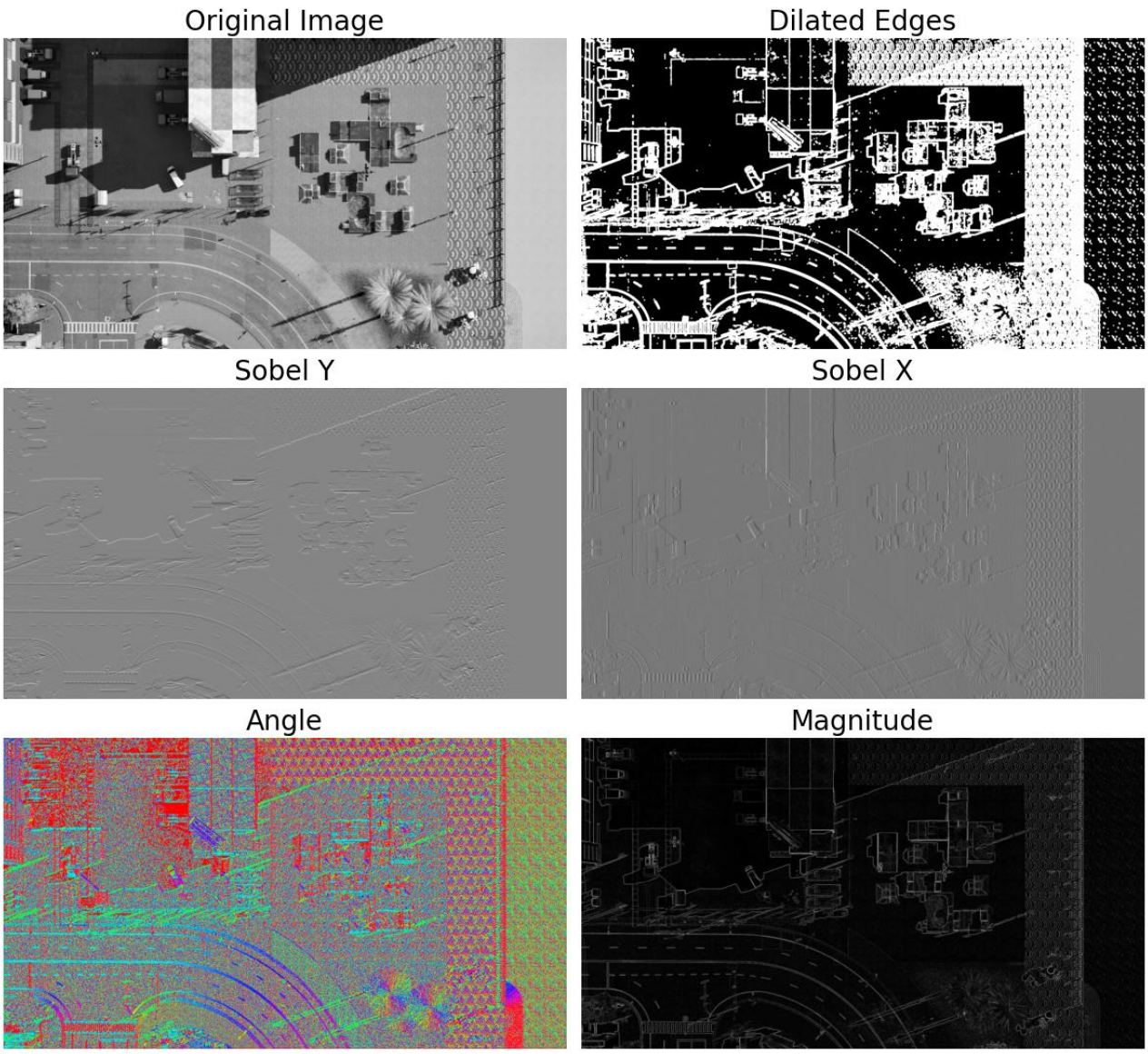

Figure 8. On top row are original grayscale image on the left and edges detected with Sobel algorithm on the right side. Dilated edges image contains magnitude image information that is emphasised with dilatate algorithm. On the middle row are results from the convolution operation with x and y kernels. On the bottom row there are visualizations of angle of orientation of edges and magnitude of edges.

Canny is a multi-stage edge detector algorithm. Although literature has examples of different count for stages, Canny algorithm can be summarized in four stages as follows:

1. Smoothing
2. Gradient magnitude
3. False edge mitigation
4. Hysteresis

In smoothing stage image is blurred by processing it with gaussian filter. This will remove unwanted noise from image and ensure detection of strong edge signals. After blurring, gradient magnitude and angle of orientation of the edges is calculated. Sobel edge operators are typically used to calculate G_x and G_y for the magnitude and angle. However other edge detectors like Prewitt or Roberts can be used as alternative options. After angle of gradient of each pixel calculated, it is adjusted to the closest 45 decree angle. In the third stage algorithm cleans away pixels that are not in the centre of the edge. Algorithm checks magnitude of pixels along the gradient. If pixel has larger magnitude than two following pixels along the gradient, it will be counted as an edge. If the magnitude is smaller, it will be counted as background. This will result in thinner and more accurate edges. Final stage, hysteresis, algorithm uses a threshold value for pixels that were marked as edges in previous stage to divide them in strong and weak edges accordingly. Pixels with magnitude above the threshold value are considered to be strong edges. However, pixels with magnitude less than the threshold and are therefore considered as weak edges, will either be labelled as edges or as background depending on their location in relation to strong edges. If a weak edge pixel is positioned close to strong edge it will be labelled as strong edge and thus affects its neighbouring weak edge pixels as strong edge pixel does. However, a weak edge pixel further away from strong edge pixels will be labelled as background. Sobel and Canny algorithm results are compared in figure 9. [47], [52], [53], [54]

Dilate function in OpenCV library is a function that morphs image by growing size of bright areas in the image. Dilation is conducted as sliding-window operation where a kernel slides over input image. For each step it checks maximum value in the kernel area and sets that value for the anchor point of the kernel. Anchor point is typically the centre pixel. [55]

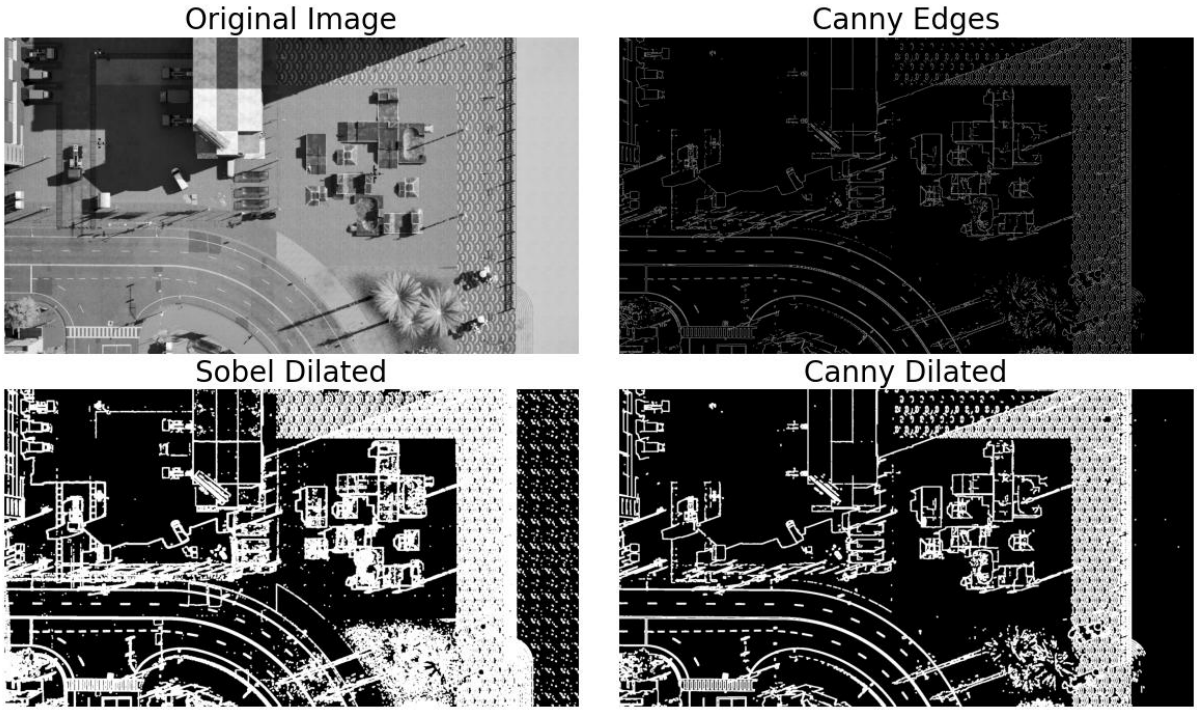

Figure 9. On top row is comparison between original grayscale image on the left and edges detected with Canny algorithm on the right. On the bottom row are dilated result of Sobel and Canny methods with same parameters for dilate operation.

Object detection is a critical capability of autonomous vehicles. Objects in real world environment have countless shapes and sizes which casts a challenge for object detection algorithm. Challenges are not limited to only features of the object's dimensions and textures as weather and lighting conditions add more variety for appearance of objects and therefore, more complexity for detecting and recognizing objects. [56]

You Only Look Once or YOLO is a modern deep learning algorithm capable for effective and efficient object detection with real time or near real time detection speed which is required by many applications for object detection [56]. YOLO has been actively developed over the years and the latest version released by Ultralytics is YOLO26 which has been developed for edge and low-power devices [57].

YOLO algorithm has undergone significant changes between different versions. However, core components have always included convolutional neural networks (CNN). Latest versions are built on architecture that can be divided into 3 main modules: Backbone, Neck and Head.

Input data is fed to the Backbone which acts as a feature extractor. Next the Neck module conducts feature fusion and finally the Head module contains detection layers. [56]

2.2 Review on related work

Review on scientific literature exposes examples of methods that are developed for map generation for UGV in UAV-UGV cooperation for search and rescue missions and methods that fit for this purpose as such or partially even though researcher do not specify this purpose in the paper. This section discusses of state-of-the-art approaches of this subject in more detail. Aim is to give reader overview of the matter and how previous studies have been conducted for this subject and what are conclusion of different approaches.

In this thesis the primary purpose for search and rescue (SARe) mission is to find target and either get help for the target or evacuate it. Therefore, the first step is to search and locate targets. However, this thesis is not focusing on this step of the SARe mission. Thus, it is to be shortly mentioned that in literature there are approaches for UAVs to effectively search for casualties from disaster areas using cameras and pinning their location using Global Navigation Satellite System (GNSS) coordinates [58]. Instead of visual detection, people can be located by UAVs also indirectly by using signals from their mobile devices, as today people commonly carry smart phones with them [59].

After detecting and locating point of interest UGV needs to be navigated to this target. UGV can be equipped with sensors that can capture information of the surroundings with many methods. However, as sensors are attached to the UGV perception is limited to relatively close perimeter. Therefore, using UAV as a sensor in UAV-UGV collaboration is typical in this form of agent cooperation. There are different approaches in literature of how this is conducted. System of systems architecture considers UAV and UGV as one system [10]. This means that UAV is not thought to be a separate system but an extension of the same system that includes UGV. Therefore, as UGV tries to navigate to target, UAV enhances UGV's navigation capability by moving ahead and informing about obstacles and hazard on the route and helping to reroute when needed. This architecture is thus to be considered extension of path planning capabilities for UGV as it allows to foresee need for route adjusting that is not possible from UGV point of view.

In urban area accident can take place on roads with heavy traffic. To get help to the location where accident occurred requires navigation between cars and preferably on the road. For UGV this navigation is based on a map that need to update frequently to keep up with dynamic environment that is road with live traffic. For such case using UAV with just RGB camera as a mapper is an option when using object detection algorithm like YOLO [60]. However, YOLO must be finetuned with dataset containing car images taken from above as the basic training for YOLO models does not contain images from above and thus, the basic model does not recognize car from bird eye view. Path finding for this case was conducted with modified version of rapidly exploring random tree (RRT) algorithm with embedded Bézier curves to smoothen the turns typical to RRT. Such approach for UAV-UGV map generation for search and rescue cooperation seems applicable for missions occurring on roads. However, method relies on the lane markings of the roads to keep the UGV path on the road. Therefore, approach may be challenging to apply on other environments beside roads. Also, method relies only on visible light and RBG camera and hence approach is not robust for changing weather and lighting conditions and thus it is not applicable as for example during nighttime. Yet another short coming for this approach is that it lacks solution for other obstacles like fissures, cracks, rocks, fences, bonds and other random form objects that can occur on road nor it has solution for worn off lane markings or snow- and ice-covered roads. Therefore, though suggested method has potential in specific scenarios, there are many aspects left unsolved for real world applications.

Deep learning models can be used to more general traversability recognition than in previous example. Using CNN model for sematic segmentation of the view from above offers solution to categorize and thus valuate areas of the image turning image to simpler map [61]. This offers base for UGV path generation. However, though given enough high-quality annotated training data can allow model to become capable to segmentate wide range of different terrains, model as such will still stumble with foliage and similar false obstacles.

Instead of relying on shape recognition from monocular RBG images, depth recognition gives great framework to perceive obstacles that could affect traversability of UGV. For this purpose, in previous literature can be found methods using depth imaging with multi view stereoscopy (MVS). MVS can be used by multi-UAV collaboration to aid UGV to navigate even in GPS denied scenarios [62]. This approach enables using monocular cameras instead of more expensive depth cameras. Method also works effectively from higher altitudes than

optimal range for most depth cameras is. Scaling of the UAV altitude for this technique depends on camera resolution.

One resent approach for UAV-UGV collaboration approaches mapping problem by fusing 2D laser scan result from above to 3D laser scan result from UGV [27]. This approach solves issue of false obstacles like foliage that does look like obstacle from above, but not from perspective of the UGV. Method does map hight differences of the ground and thus path finding algorithm gets inclination data as input and hence path can be planned to avoid risk of too steep slopes along other obstacles. Method seems comprehensive for generating map that allows generating applicable path for UGV with mitigated risk. However, as method relies on active sensors, it is not optimal in scenarios where it is preferable to remain undetected.

2.3 Chapter summary

In this section is a brief collection of key elements needed to be taken into consideration while developing a model for UAV-UGV collaboration in search and rescue missions. These are based on findings from review on relevant prior literature that was surveyed in previous sections of this chapter.

UAV and UGV agent collaboration is capable combination of swift aerial surveillance and slower but strong ground platform that is capable of performing various tasks. UAV enhances remote sensing capabilities of UGV and allows better navigation. UGV can act as a charging station for UAV and thus prolong range and operation duration for UAV.

Communication is key for the collaboration and thus it must work. Control of the cooperation is based on working communication. There are plenty of path search algorithms with very different paradigms. However, how well they fit for the problem on table, depends on the problem qualities and availability of computing resources.

Effective environment mapping requires remote sensing capabilities that allow modelling the target area. As UAV can fly above ground, it is optimal tool to use remote sensing tools for scanning large area of the operating zone of UGV. Optimal sensor depends on the purpose mission. Although basic RGB cameras can be affordable and powerful for many purposes in UAV and UGV cooperation, mapping may require sensors that developed for spatial sensing. These sensors can be divided into active and in passive types. Former does emit

electromagnetic radiation which it then senses as it bounces back from targeted surfaces and estimates distance according to information extracted from the returning radiation. This information can be for example time that it took for return or triangulation calculation between emitter, receiver and the surface that the radiation bounced back. However, passive sensors do not emit anything and thus they rely on external radiation source like sun to receive information. Although active sensors are more capable than in passive in conditions with poor visibility, it is to be taken into consideration that in some situations like on battlefield conditions, it is preferable to avoid active sensors as those can be detected and located by enemy operators.

3 A method for semi-autonomous map generation and path finding in UAV-UGV cooperation for search and rescue mission in dynamic and unmapped environment

This chapter starts with analysis over the key elements found from prior literature that are to be taken into consideration while developing new model for UAV-UGV collaboration for search and rescue missions. Review covering relevant prior literature was conducted in chapter 2. Goal is to find gaps of existing approaches and try to propose solutions to fill these gaps. This chapter then continues by describing the conducted study and by explaining the developed concept and methods. Chapter is organized to sections as follows:

- 3.1 Foundations for the study,
- 3.2 The Concept, containing subsections:
    - 3.2.1 Description of the concept,
    - 3.2.2 Target locating,
    - 3.2.3 Depth scanning,
    - 3.2.4 Map generating,
    - 3.2.5 Map stitching,
    - 3.2.6 Path generating,
    - 3.2.7 Hardware setup overview
- 3.3 The method for semi-autonomous map generation and path finding, containing subsections:
    - 3.3.1 Starting point for the research,
    - 3.3.2 Finding a solution, 3.3.3 MapGenerator,
    - 3.3.4 MapStitcher and chapter ends with
    - 3.3.5 PathGenerator.

3.1 Foundations for the study

Real-world casts challenges in form of possibility for constant change of physical obstacle location and pose. However, challenges are not limited to objects as real-world also is subject to changing lighting and weather conditions. This sets challenges for sensors that are reliant on visible light spectrum. Although map generating in clear weather conditions and daylight conditions could be done with RGB sensors, it is not possible in darkness like during night or

in poor weather conditions. On the other hand, lidar and radar are invariant to lighting conditions and are therefore applicable also in dark. Lidar and radar are also less sensitive to fog and rain than RGB sensors and are great for mapping range and thus 3D structure of the environment. However, both lidar and radar are active sensors which can cause these sensors to be detected and located which may not be preferred during search and rescue missions on battlefield conditions. Therefore, there is a need for alternative approach where this study aims to give an answer.

Depth cameras that are not emitting radiation like ones based on stereo vision and are therefore passive, offer an interesting starting point for developing new method for mapping, as they are not detectable as active sensors, like radar, lidar and depth cameras using structured light or light pulse like TOF cameras. However, as most commercial passive depth cameras so are based on RGB sensor solutions, it is relevant to consider how well can these solutions cope in dim lighting conditions and thus whether these offer robust enough solution for the mapping problem in real-world environment. However, there exists commercial solutions like MultiSense® ST25 MEGA from Carnegie Robotics that is passive thermal stereo sensor. This opens room for a new approach for map generating model that is both, based on passive sensors and is also applicable to poor lighting conditions.

This study is conducted in a simulator environment. Using state-of-the-art simulator allows simulating lighting effects, but not weather conditions like wind, fog, rain, snow and dust clouds. In addition to remote sensing, these conditions do affect to operation feasibility for UAV and UGV also in other ways. However, weather effects are out of the scope of this study, and therefore, deeper analysis on this matter was not conducted.

Dynamic environments require frequent update for the map and hence the path re-planned for the UGV. Therefore, requirement for algorithm is that it is computationally light and hence update frequency can be kept high. To achieve this design of algorithm starts with initial assumption to avoid time complexity of ***O(n2)*** and beyond if possible.

For path planning algorithms there are plenty of options to choose from. However, the goal of this thesis is to create a novel approach for mapping and path generation for UGV using UAV-UGV cooperation and depth sensors from above. Thus, finding optimal algorithm for the path generation section of this new approach is left for future studies. Here A* is chosen to be the base for the path search algorithm and it is modified to fit the needs of the novel approach introduced in this thesis. A* is applied as it is well known and fit for the purpose of

path generation. Like most path search algorithms, A* can become computationally demanding when scaling up the scenario. Therefore, this study aims to form a map of the environment that has information of obstacles which affect to UGV movement but is not computationally demanding for path search algorithm.

This study introduces approach to generate map and path to target destination for an unmanned ground vehicle with one or more unmanned aerial vehicles in semi-autonomous manner. The method offers solution for real-time mapping of dynamic and unmapped environments to ensure obstacle free pass adaptation for UGV.

The method was built upon research on a simulator environment to test idea and prove the viability of the concept. Carla 0.10.0 is based on UnrealEngine 5.5 which offer realistic environment as it uses realistic lighting conditions that mimic real world with detailed and dynamic shadows based on tray tracing method. The simulator also allows detailed surface textures and variety of heterogeneous shapes of buildings, trees and other objects that would cause challenges in real world environments. Carla simulator offers variety of sensors and therefore, it was apt for this research as the method relays on depth camera data.

Developed method is one key element to build a semi-autonomous search and rescue operating model. Therefore, this thesis also introduces broader concept for the model of operating in search and rescue mission, in addition to the map generation and path finding part of it, to provide reader more thoroughly understanding of how this method works as part of this concept and why the method is operating as it is. Concept is comprehensive and it is structured in a manner that it can be made to run autonomously as a whole or partly, depending on the resources and capabilities dedicated for each subprocess.

Mapping method is built in a manner that allows flexibility for hardware resources. Input for the mapping can be provided by one or more UAVs and map and path generating steps can be computed in chosen computing unit depending on the availability computing power and considered data transferring capabilities between equipment. Input data of an area can be used to build optimal maps and paths for multiple UGVs. Thus, both inputs, general scanning for targets with RGB or infrared, and mapping with depth cameras can be processed according to each UGV unit’s initial position in respect to selected target.

Following sections of this chapter are dedicated to introducing the concept for search and rescue missions and to break down the novel method in parts. Method is split into three modules named MapGenerator, MapStitcher and PathGenerator.

3.2 The concept

This section introduces a concept for using UAVs and UGVs for search and rescue missions. The concept is structured in a manner that allows partial or throughout autonomous operation. However, as this study focuses on the map and path generation part of the process and does not present autonomous solutions for other parts. The first subsection lists steps of the concept with a short description of each. Following subsections dissect the concept steps further in more details.

3.2.1 Description of the concept

As the result of the research the following concept was created:

1. An UAV or set of UAVs is scanning for target area for wounded persons with RGB or infrared camera and object detection method like YOLO. Scanning can be done as high as hardware of UAV's allows flying and sensors are able to reliably detect persons.
2. Once person or persons are located, coordinates are stored.
3. An UAV or set of UAVs with depth cameras pointing downwards are sent to map area between target (person) and UGV. UAVs fly optimally at 20m hight from the ground floor level using UGV position as a calibration point.
4. For each frame, depth image data is processed with MapGenerator of the method. As an output, MapGenerator returns compressed grayscale map per frame where false obstacles above the UGV's operating area are removed.
5. Partial maps are stitched by MapStitcher to form larger map to allow searching viable route for UGV from initial location to the target destination

6. Modified A* search algorithm in PathGenerator is used to generate path for UGV on the stitched area map.
7. UGV follows the path.
8. Update map by returning to step 3.

In figure 10 these steps are arranged in process chart within three phases in which agents operate. Figure 11 demonstrates operating phases of the agents. The following subsections are dedicated to explaining the steps of the concept and the methods in more detail.

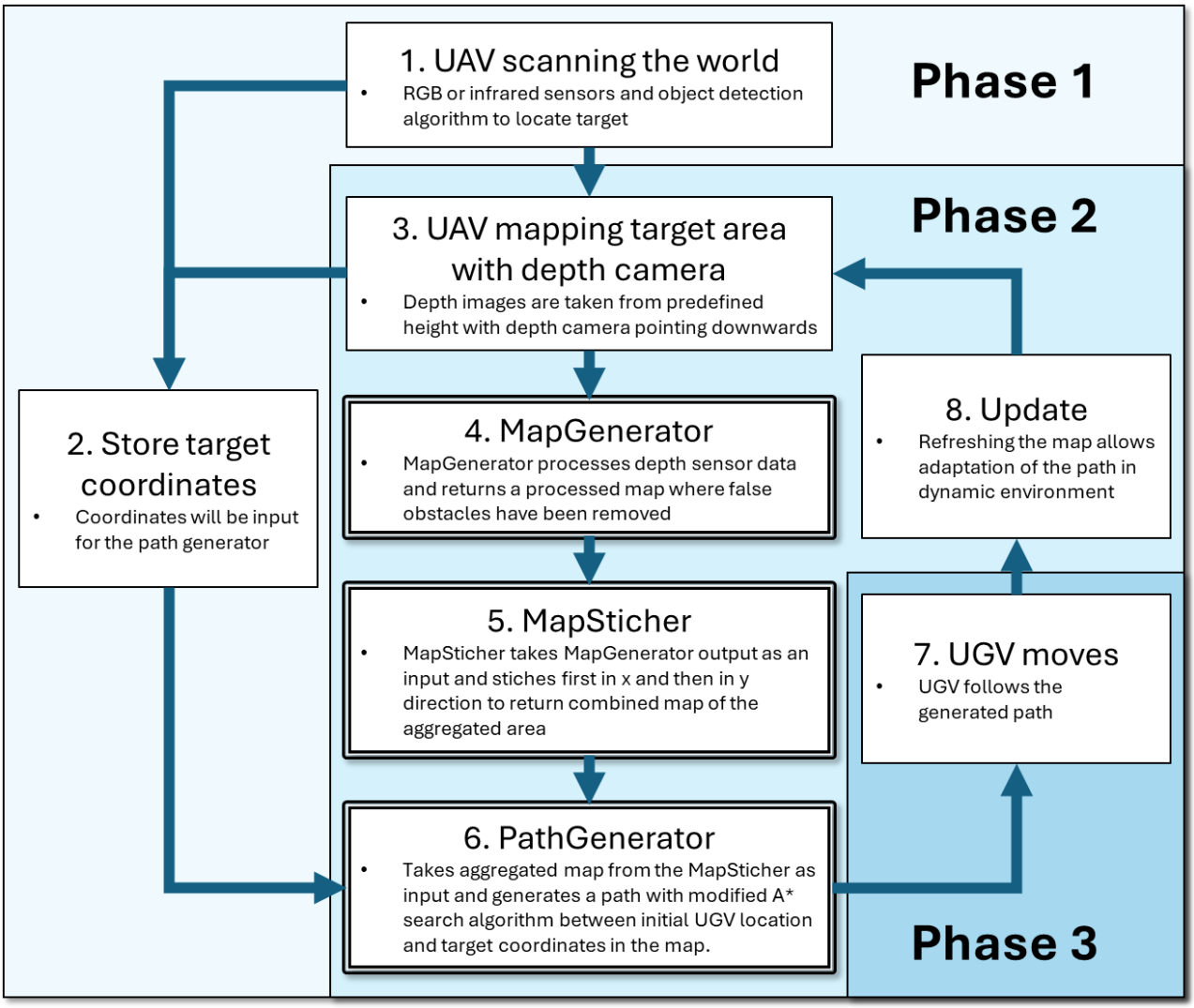

Figure 10. Process chart of the concept. Phases 4, 5 and 6 are highlighted as algorithms for these phases were developed in this study and are described in detail later in this chapter.

3.2.2 Target locating

Concept of search and rescue mission with UAV-UGV cooperation starts by locating target where UGV navigates to. Example target is a wounded or unconscious person laying on a ground still or with limited ability to move around. Therefore, it can be expected that after locating person there is time for UGV to reach person before target location has changed significantly.

When there is risk of casualties, there is need for swift action to locate persons in need of help and get the required aid to those in peril. Therefore, scanning area of interest is done with one or more UAVs depending on available resources and size of area. Methods for locating target and defining position are out of scope of this study and therefore, exact methods are not defined here. However, the principle of scanning phase of the concept is defined next.

UAV or swarm is equipped with thermal imaging sensor or RGB camera along with pretrained object detector algorithms. Object detection algorithm like YOLO must be fine-tuned to detect humans from above. A short test shows that YOLO11 nano pretrained with COCO dataset can detect a human figure laying in ground if it is positioned in image vertically oriented head up and feed down. However, if sensor does capture lying human in rotated angle YOLO does not detect it. Thus, finetuning YOLO with dataset containing rotated human figures and humans lying in various position is required for reliable detector.

UAVs are set to fly and scan area where there are potential casualties. Flight and scanning can be done autonomously or by drone pilots depending on the available resources. For scanning phase higher flight altitude allows larger area to be scanned in shorter time. However, UAV flight altitude is limited due limitations of hardware and detection capabilities of thermal and RGB sensors in combination with object detection algorithms. Weather and visibility conditions can also limit flight altitude.

Once potential target or targets are detected, limits for area to be mapped are defined and location of the targets within this locally defined area stored and set to path finder module of the algorithm. Directions of area to be mapped are set for the UAV or swarm that is equipped with depth cameras pointing downwards. UAVs used in this first phase can now either return

to origin, continue scanning for more targets, or change role to mapper UAV if those are equipped with depth camera and battery allows.

3.2.3 Depth scanning

When targets are located, a zone is defined that covers both, initial UGV position and the location of the target for which the UGV is to be sent. This zone is the area to be mapped with depth cameras for generating map and path for UGV to move along to reach targets. For mapping task one or more UAVs which are equipped with depth cameras pointing downwards, are sent to systematically capture depth images of each part of this area. These images are to be used as an input for the MapGenerator.

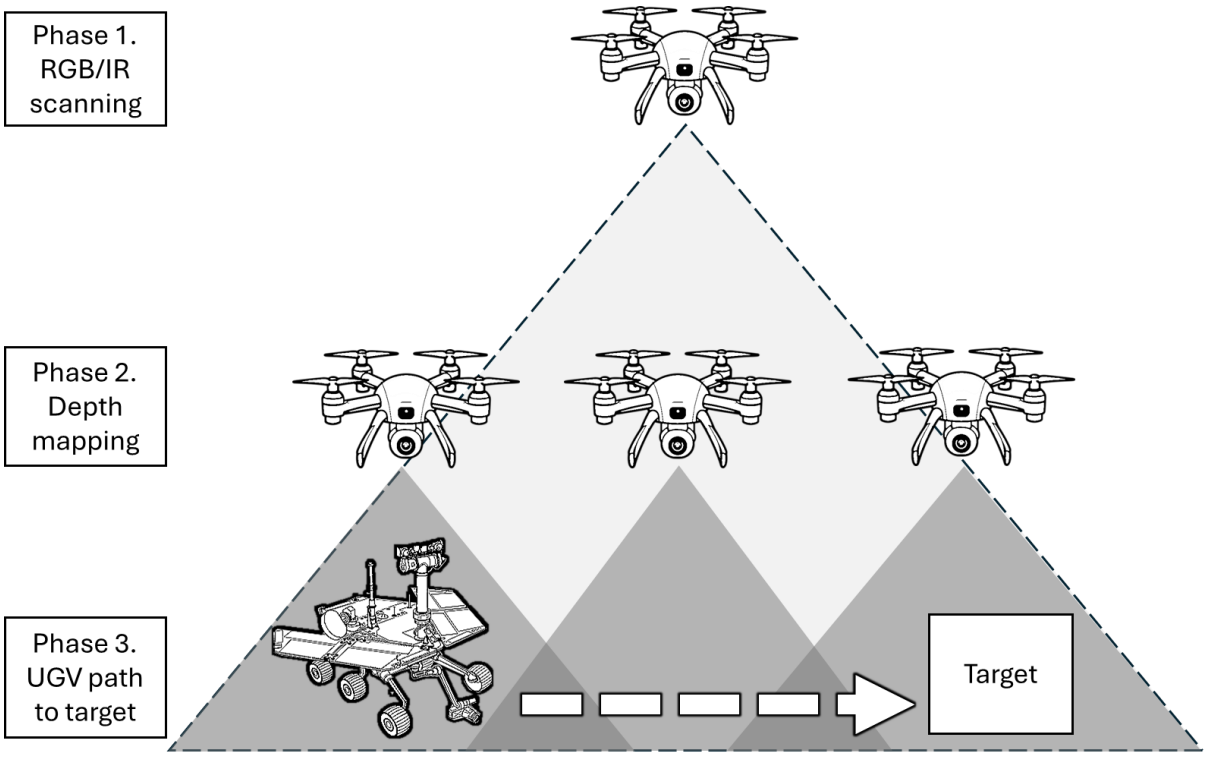

Figure 11. Unmanned vehicle action phases. In 1. Phase UAV scans area for target with RGB or infrared sensor. In phase 2. UAV with depth sensor maps area where both, target and UGV are positioned. Depth images overlap to improve aggregated map quality in stitching step of the process. In phase 3. UGV travels along the planned path to the target.

Depth cameras do have limited range of operating. Survey on technical specifications of commercial depth cameras shows that most long-range cameras have maximum operating range of 18 to 20 meters though there are some exceptions that can operate to 35 and one even

to 50-meter range. This maximum operating range of the camera sets altitude limit for the depth mapper UAVs. Camera qualities also set framework for depth image capture frequency. MapSticher takes a set of mini maps, processed by MapGenerator, as an input. This set is built from depth images that are taken of the area according to x and y axis with steps of which frequency, distance of each other's centre point, is calculated based on the UAV altitude which is calibrated according to the ground level under the UGV's initial location.

Depth images are named according to local x, y and z coordinates. As depth images are taken from air to ground, x and y present width and height dimensions of the images and values are in meters, while z marks the altitude of UAV in meters according to calibration. Images are fed to MapGenerator which returns mini maps that ready to be stitched to aggregated map of the defined area. Count of the images depends on the size of the area, field of view of the depth camera and the altitude of the UAV capturing depth images.

3.2.4 Map generating

Maps are generated with MapGenerator. MapGenerator takes raw depth images as an input and returns mini maps that are named with x, y and z coordinates like the input file. This systematic naming allows MapStitcher to aggregate mini maps in correct order to form complete map for the PathGenerator.

Map generating is semi-autonomous process as it may require some parameter fine tuning from the operator, though the entire process from area scanning to UGV movement and to update loop is otherwise possible to set to operate autonomously. MapGenerator takes raw depth image and turns it in to a map that does only contain relevant information for path planning. MapGenerator also removes a layer above the UGV that may otherwise cause false obstacles to path planning phase. This layer can contain for example tree branches, power lines, eaves and other objects located above the ground level. These do not cause true obstacle for UGV as these are above the height that UGV needs to follow. However, as image data is captured from above, these are embedded to the data unless these are filtered away before path generator. Therefore, operator may need to fine tune the layer or zone height to be filtered so that it is optimal for removing as much of the unnecessary objects as possible while still allowing enough height for the layer that is fit for the UGV up and down manoeuvrability capabilities.

UGV capability to climb up and down is to be considered when setting the parameters. If UGV in use can climb over steeper height differences with ease, MapGenerator parameters are to be set so that resulting map does include wider range of height differences available while UGV with modest manoeuvrability needs flatter map. The MapGenerator does set value of each pixel of the map either in various range of walkable scale or an obstacle value, depending on the settings based on UGV features.
In addition to absolute height limits for the walkable area, Sobel edge detection method is used to emphasize steepest edges in the map as vertical edge can cause more challenge for UGV with less elevation than another slope with higher overall elevation but with less steepness.

Output of MapGenerator is downsized significantly in comparison to original input to speed up following steps. Smaller maps do contain all relevant information for path generation and UGV navigation towards target. These mini maps are input for MapStitcher.

3.2.5 Map stitching

Once mini maps are produced, MapStitcher generates aggregated map of those. MapStitcher combines first mini maps along the local x axis, the width dimension of the images. Outcome of this step is width map strips of target area. Next step is to stitch these strips to final aggregated map of the area which was defined already in the first phase of concept after targets were located.

Images do show increasing level of perspective distortion the further the reviewed area is from the centre point of the image. This causes a challenge for map generation as optimal map is directly from above without distortions. As mini maps are generated directly from the raw depth images they do have distortion issue as the raw images. To mitigate this distortion effect for the final map, higher image capture altitude is one option. However, range of depth image sensors is often limited and another solution is required. As distortion increases further away from the centre point of the image, alternative solution for distortion mitigation is to use only narrow area of close to the centre of images for the aggregated map. In theory optimal result would be reached if images would be captured side by side with only one pixel row or column apart from previous one. From performance point of view this is not ideal as it would

require vast amount of image data to process and transfer between group of unmanned vehicles. Also, UAVs would need to move very slowly across the area while mapping it to get almost continuous flow of images. Therefore, in this study UAV depth images from altitude of 20 meters and raw depth images size of 1920 x 1200 (aspect ratio of 16 x 10) gave good quality for the aggregated map with MapStitcher algorithm when images were taken with steps of 10 meters on x axis and 5,5 meters on y axis.

MapStitcher returns aggregated map that is stitched from the mini maps. This final version map is stitched in a way that mitigates distortion due perspective and thus it has dimensions of objects closer to the actual dimension than what a single image from higher altitude would have. However, though MapStitcher uses only the near centre parts of the mini maps for the most parts of the aggregated map, the edge parts do contain the distorted areas from the outer edge of the mini map.

3.2.6 Path generating

To get UGV safely to the target it requires a path of which it can follow. This path will be generated by the PathGenerator. PathGenerator needs to have several inputs. First it needs to have aggregated map generated by MapGenerator and MapStitcher out of the depth images captured by UAVs. Secondly it needs to have initial position of the UGV on the aggregated map. Thirdly, PathGenerator requires target coordinates on the same map. With these inputs PathGenerator is generating a path for the UGV using modified A* algorithm.

PathGenerator return path on a map for the UGV to follow. Though the path takes in account the manoeuvrability of the UGV, assuming that parameters for MapGenerator have be set correctly, it is still expected in this concept that the local navigation kit of the UGV is able to adjust route locally if needed.

3.2.7 Hardware setup overview

Hardware setup for the search and rescue mission based on the concept introduced on this thesis is flexible and can be set according to resources available and the requirements of the mission ahead, given that few minimum conditions are met. The minimum requirement is to

have a single UAV with a depth sensor pointed directly downwards, accompanied with a UGV capable to navigate in the terrain of the mission area. This setup expects that target location is already known. Otherwise, an UAV with RGB or IR sensor and capability to run object detection algorithm either locally in UAV or on external equipment, is required for the first step to locate targets from the potential area. All these vehicles need to be able to communicate with each other either directly or via some additional equipment. Sufficient computing capability is required to handle data processing on each step of the concept, whether this is done in UAV, UGV or in additional equipment.

Concept allows scaling up count of UAVs and UGVs. This is a way to speed up searching, mapping and reaching targets. Therefore, scaling enables rescue operators to provide help quicker for larger area and for more casualties. UGV manoeuvrability affects to speed not by theoretical linear speed of UGV but also via path length. UGV capable to climb over modest hight differences like curbs may have shorter route to target than an UGV with higher linear speed but less manoeuvrability. For depth mapping UAV a depth sensor with longer operating range gives an advantage as it needs to provide less images of the area to form aggregated map and thus speed up the update loop for the UGV path.

The concept concentrates on reaching the targets with UGV. Therefore, equipment and capabilities of UGV deliver aid or to operate after target has been reached, is not considered further in this study. Though concept is developed for search and rescue purposes, it generalizes well to other missions that include target locating and path planning to target in unmapped area using UAVs and UGVs.

3.3 The method for semi-autonomous map generation and path finding

This section goes more in depth for the research and the method that was created as an outcome of it. First subsection is about the setup and the environment and challenges it set for the research problem. Second subsection is about developing the solution, and the remaining subsections are more in-depth introduction of the three modules of the method: MapGenerator, MapStitcher and PathGenrator.

3.3.1 Starting point for the research

The starting point for the research was to find a method to guide an UGV unit to a person in need of aid in an area where environment is dynamic like in a city with traffic or the environment is not mapped in advance. The latter may be due either lag of mapping in general or due drastic changes after the previous mapping. This could be caused by either cataclysmic natural reasons like earthquakes or actions of humankind like war that shape the landscape vastly in short period of time. These events do not only shape landscape with speed but also with chaotic manner, generating objects with plethora of shapes and sizes. This causes challenges for many object detection methods based on pretraining and thus using such methods is not straightforward in this scenario.

To build a method that would work in real life situations, options were either a real-world setup or a simulation that models real world phenomenon like shadow and light with such a quality that would cause challenges for the method like real world would. Real world test would have required significantly more resources, both material and time, than simulator. Therefore, simulator was chosen to be the test environment for this research. As simulator needed to be able to provide real world type challenges for the image data, Carla 0.10.0, using UnrealEngine 5.5. turned out to be an apt simulator. UnrealEngine 5.5 provides realistic light and shadow due its ray tracking technique. It allows high detailed surface textures and reflections, causing light to cast shadows and reflections in a complex manner like in real world environments. These qualities are prone to cause challenges for finding object shapes that could be used for mapping. As these are issues that also come up in real world situations, it was necessary to be able to face these issues already in the simulator. In addition to simulation quality, Carla also has variety of built in sensor types that allowed to try other approaches than those based on the RGB sensor to taggle the issue in hand.

3.3.2 Finding a solution

Light and shadow are constantly changing in outdoors scenarios of real world. When using RGB channel, this dynamic effect causes each pixel of image of the environment to constantly change value. This is also the case in grey scale image, though values only change in one channel instead of three as in RGB images. Thus, using visible light sensor data to map

generating is not a straightforward case. Therefore, first part of this study was to test other sensor options for mapping the environment from above.

Depth camera records distance to objects from the sensor. Though there are few different types of depth sensor technologies, the principal outcome is the same. Therefore, for this study it was not relevant define precise technology, but instead survey commercially available cameras and capabilities of these cameras. Survey indicated that common range for long range depth cameras is from 18 to 20 meters. However, there are some outliers that promise maximum range to 35 and even to 50-meter distance. As 18 to 20 meters was common maximum range like shown in figure 12, it was chosen to be the starting point the method development. Therefore, the testing was mostly done with UAV flying 18 to 20 meters above the zone to be mapped.

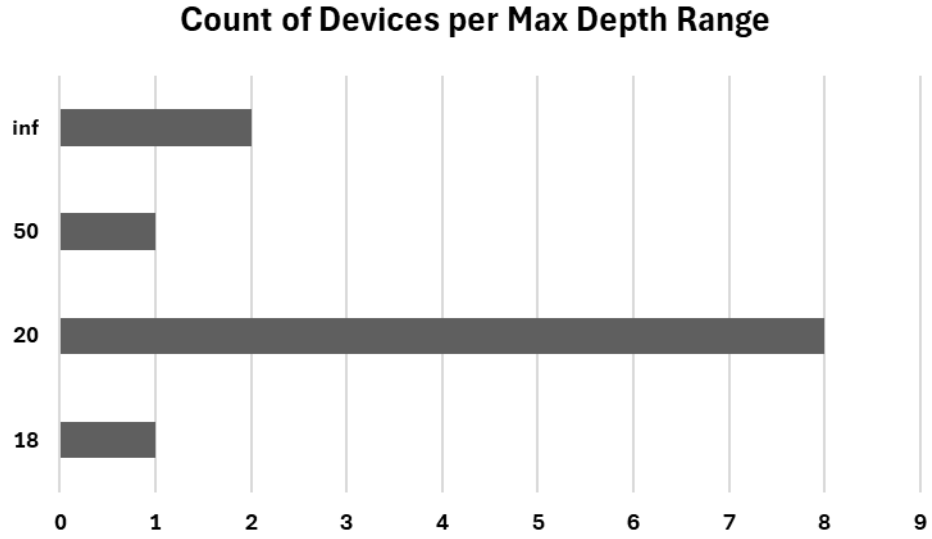

Figure 12. Survey on depth camera specifications show that long range depth cameras (range more than 15m) are based on stereo camera technology and the most common maximum operating range is 20m.

Field of view (FoV) of depth cameras according to survey was around 90 in width and around 55 in height and typical resolution was 1920 x 1200 and thus the aspect ratio of 16 x 10. Carla simulator allows freedom with sensor settings via python API. One limitation is that FoV can only be set for the width. Therefore, for the test following settings were used for depth camera:

- FoV 90 for x-axis (image width)
- Aspect ratio 16 x 10

- Resolution 1920 x 1200

3.3.3 MapGenerator

This subsection describes issues and solutions for generating maps from raw depth images. Combination of these solutions form MapGenerator module that takes raw depth images as input and generates mini maps that are input for the MapSticher module.

The first iterations of the map generating methods already showed promising results as depth camera approach was not sensitive to lighting conditions nor it was sensitive for surface textures and colours. Normalized depth camera image is grayscale image with pixel values within range 0 to 255. Value is greater for pixels that describe point further away from the sensor and value is smaller for the pixels describing close objects. Therefore, as sensor is attached to UAV and it is pointing downwards, pixel valued 255 would present a point of a surface at the maximum distance of sensor or further. As altitude of sensor attached to UAV in the test setup is 20 m from the floor level, value 255 describes all surfaces on that same level or below. For the first iterations, maps were generated based on idea that this floor level and marginal of 5 to 20 cm above and below were considered as manoeuvrable area and thus given a value of zero while all other pixels were converted to value 255. Thus, resulted map was two coloured and was easy to convert to a binary matrix where pixels 0 if walkable and 1 if an obstacle. This was a simple solution to generate map for path generation for UGV.

Binary map is fast to process and simply to apply. However, iterating tests on different areas of urban landscape in the Carla simulator showed that binary map has shortcomings that need to be tackled with more sophisticated methods. The first issue was that as binary image does only give path finder method information per pixel whether point of area is walkable or not. Therefore, preliminary settings, floor level and margin above and below it, determine threshold for height difference tolerance in the map. Thus, all points of area that are labelled walkable, are valued equally no matter the height difference and therefore, the effort required from UGV to pass such point. On the other hand, binary map also forces to set preliminary settings for height difference tolerance to be so narrow that it may transform perfectly walkable route to an obstacle if reaching that route is for example behind an inclined level which would be easy to climb for the UGV in use, but the route is located on a level that is out of the threshold boundaries.

The second issue with binary map came with objects well above the floor level. As mapping is done from above, depth camera captures all surfaces below the sensor. Therefore, for example, power lines, streetlights and trees do cover more surface in the depth image than those block traversable area for the UGV on floor level. For binary map this causes false obstacles that can block the route entirely. For example, in a case when power line goes across a street from one building to another it is above the walkable level threshold and thus it is labelled as an obstacle. Similarly, trees forming canopy do not block UGV route on ground level. However, using simple threshold to define walkable level results canopy to appear as an obstacle in the map.

To tackle the first issue, binary map is switched to a greyscale map. Greyscale allows to value walkable pixels heterogeneously according to their height level. Therefore, this approach allows overall threshold to be set more tolerant than in binary map preliminary settings as route planning can consider pixel feasibility for the path and not just simply consider it equal to all other walkable or obstacle pixel. This approach also enables using good route that is not on near the hight level of initial UGV position, if it is easily accessible for example via inclined ramp and is otherwise better than routes that would be closer to initial high level of UGV.

The second issue is invariant to whether map is binary or grayscale as these false obstacles are on a level that is above the potential walkable area for UGV. Therefore, this issue requires tailored solution. Images taken from a single point or tight cluster of points do have perspective distortion that increases the further a pixel is from the centre of the image. Generally, this is not a favour for map generating task as for mapping ideal would be that each pixel would present a view perpendicularly downwards from a virtual plane at the level of the sensor and with size matching image presentation. However, to tackle the issue in hand this perspective distortion offers a solution. It is possible to see below object surfaces that have top surface high from the ground and if those objects are not attached directly to ground below them. Therefore, removing detected objects in a height zone between the UAV sensor height and defined UGV manoeuvre zone, does remove effectively parts of objects that are not relevant to path generation for UGV. Thus, this method will remove most false obstacles from this hight zone that is to be called false obstacle zone (FOZ). However, removing these false obstacles leads to holes in the map. Using Open CV’s Inpaint function to fill these caps

according to pixel values of surrounding area from the UGV zone works effectively. Thus, using inpaint algorithm allows to generate mapping of the UGV zone that is purified from inference caused by FOZ. Removal of FOZ and filling lost information with inpaint function forms FOZ-filter which produces purified view of the ground level where UGV operates. Figure 13 demonstrates FOZ concept.

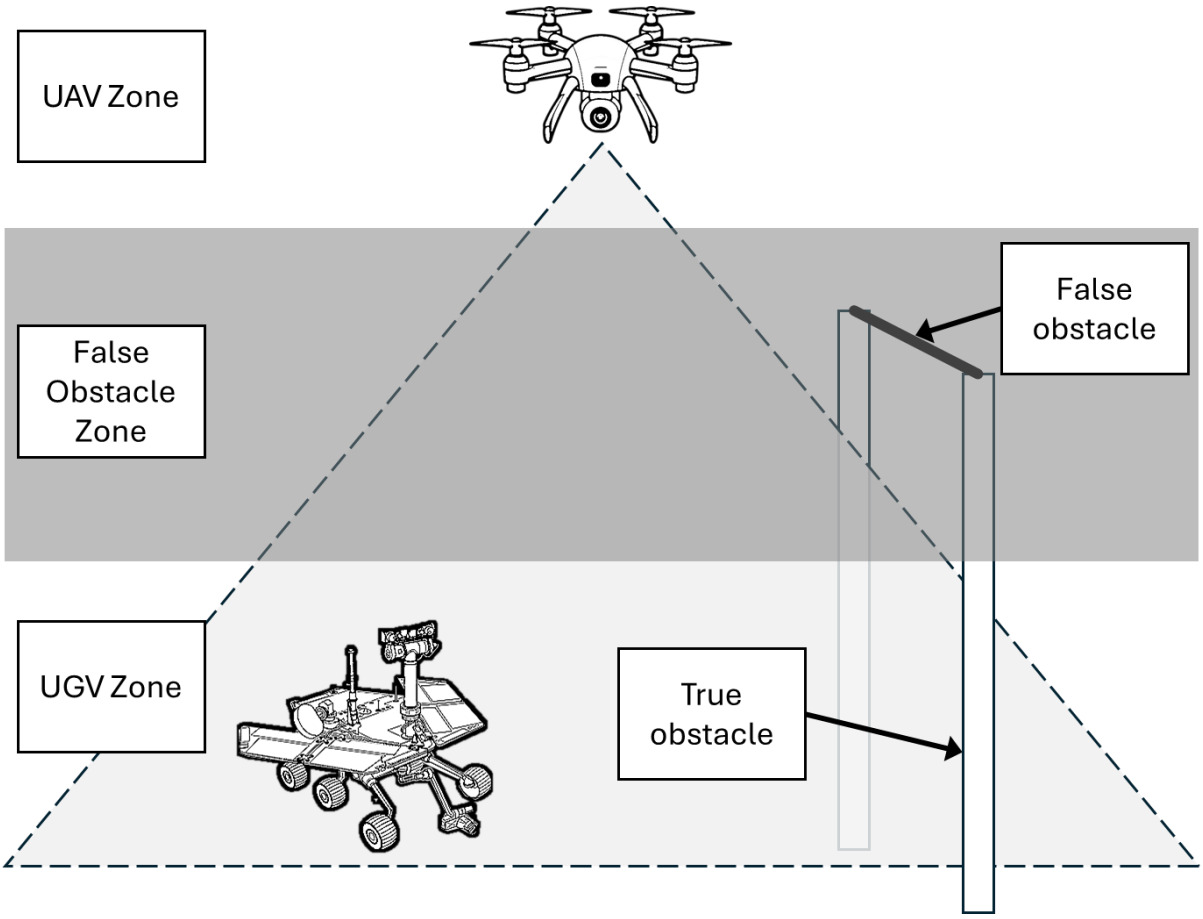

Figure 13. UAV, False obstacle zone and UGV zones defined for the process of generating map from depth images. False obstacle zone is removed from the map to avoid false obstacles of blocking potential UGV routes.

Greyscale map enables path finder to create path over range of limited height differences. This is useful capability as route over some height differences may be better that a flat one that needs to take a detour. If height difference is too large it may not be possible to traverse by UGV even though reaching the height level would be otherwise possible with less steep inclination. Therefore, a method to emphasis the steepest edges in the map was needed. At first canny edge detector was tested for this purpose. However, after iterating several tests, Canny method showed inconsistent quality for the edge emphasis. Setting Canny parameters optimal for one image edges did show poorly fitting edges for another image. Depending on

the parameter setting it showed solid edges on some parts of the image but had holes in other parts. Goal was to develop approach that would not require manual tuning of the parameters and therefore another approach was needed. Next method to test for the task was Sobel edge detection method. Sobel edge detection was embedded in the MapGenerator by first implementing convolution operations on X and Y axis and then calculating the magnitude from outcome of those operations accordingly. Although Sobel edge detection algorithm is more sensitive to noise and false edges than Canny, it is not a problem in this case as processed depth image that is given as an input for the edge detector is already cleaned from all textures and flattened to have high contrast for the edges that need to be emphasised. In comparison to Canny, Sobel edge detector is less biassed by the direction of the inclination of the edges. Therefore, Sobel edge detection method is embedded in MapGenerator. To ensure thickness of the highlighting and thus ensure that path will not be drawn through too steep slopes or actual obstacles, magnitude of edges detected by Sobel is further strengthened with dilate algorithm. Figure 14 has collection of intermediate outputs of different stages of the MapGenerator.

As raw depth image resolution is 1920 x 1200 pixels, it takes time and memory to process. To mitigate this resource demand MapGenerator resizes its output in the end of processing for chosen multitude of 16 x 10. This speeds up process in MapStitcher and PathGenerator modules of the method. However, resizing the outcome of MapGenerator does not speed up itself which is an issue as MapGenerator module is the bottle neck of the method. To tackle this issue, resizing operation is also implemented in early steps of MapGenerator, though this preprocessing step is less drastic to avoid information loss before processing steps.

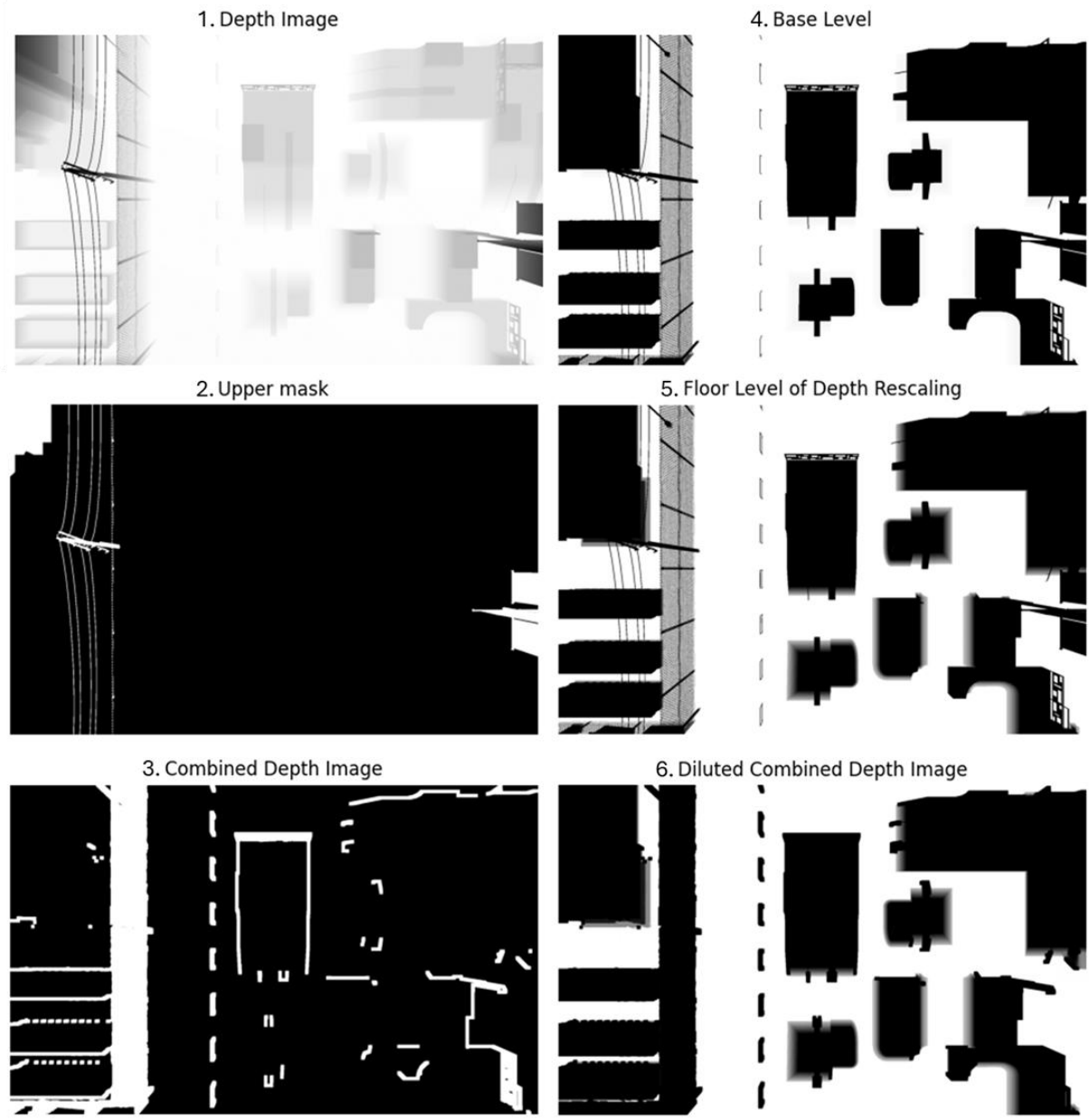

Figure 14. Different stages of MapGenerator. 1. Depth image input normalized to grayscale. 2. False obstacle zone extracted to mask. 3. Emphasised edge mask generated with canny or Sobel edge detector methods. 4. Base level based on initial UGV altitude with small threshold is coloured white and every pixel higher or lover is black. 5. Base/floor level is enriched with range that is possible to manoeuvre with UGV. This range is coloured with pixel values that are lower the further they are from the base level. 6. Edge mask is subtracted from image the fifth image. Inpaint function is used for areas marked by the upper mask to remove FOZ.

Resizing is done with resize function from OpenCV library. This function has different methods for resizing. All methods of resize function were tested, and results did show differences. Some of the methods did show better results in one part of the map and other in another part as can be seen in figure 15. Therefore, none of the methods could be declared as optimal. For instance, a method could be best in showing a cap between two obstacles that in

real world is wide enough for UGV to pass, while other methods would spill obstacle colouring on this cap. However, this same method may also be lacking in obstacle colouring of a thin fence that is an actual obstacle. Thus, a compromise was to prefer use of down scaling methods that are slightly too eager to spill obstacle pixels on walkable areas, then to use methods that do tend to leave obstacles partly unmarked.

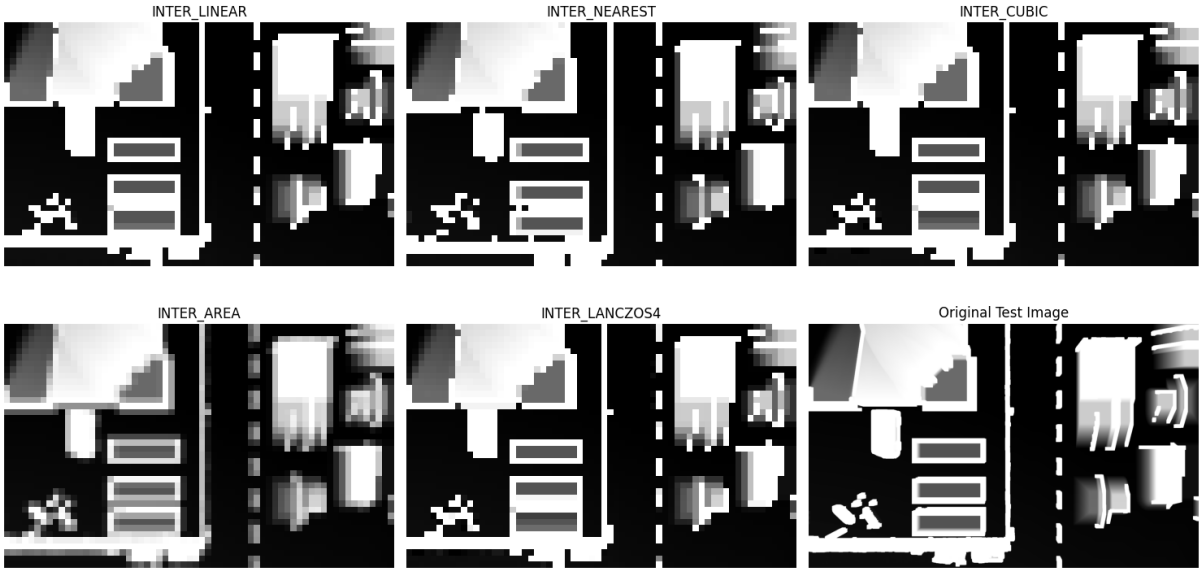

Figure 15. Rescaling with different methods. Bottom right image is a map before downscaling.

MapGenerator, which pseudocode is shown in figure 16, returns a map processed from the raw depth image. This map is part of the area where both, target and the UGV are located. These mini maps are then saved according to their location in the local coordinate system. Name includes x, y and z coordinates. Files are saved in folders according to their location on y-axis to ensure MapSticher operates faultlessly.

MapGenerator

```
FUNCTION MapGenerator(
        depth_image,                    # Depth image taken by UAV
        uav_altitude,                   # UAV altitude from ground level
        ugv_safety_margin,              # Height above UGV to be taken into account when moving
        foz_upper_limit,                # UAV zone bottom limit
        limit_hi,                       # Ground level top depth
        limit_lo,                       # Ground level bottom depth
        gradient_threshold,             # Max height level difference to be mapped walkable for UGV
        mini_x,                         # Mini map x dimension, should be multiple of 16
        mini_y):                        # Mini map x dimension, should be multiple of 10

        Convert values to meters and scale matrix size down to speed up following steps
        depth_in_meters = convert_to_meters(depth_image)
        depth_in_meters = resize(depth_in_meters, 640x400)

        Calculate UGV Zone height
        floor_level = uav_altitude - max(depth_in_meters)
        ugv_zone_height = floor_level + ugv_safety_margin

        Create FOZ (False Obstacle Zone) mask (area above UGV zone)
        foz_mask = create_mask(depth_in_meters, ugv_zone_height, foz_upper_limit)

        Create base level map with gradient around ground level
        base_level = create_base_level_map(depth_in_meters, limit_hi, limit_lo, gradient_threshold)

        Apply inpaint to remove FOZ objects from the map
        map_inpainted = inpaint(base_level, foz_mask)

        Detect edges using Sobel filter
        edges = sobel_edge_detection(map_inpainted)
        edges = dilate(edges)

        Emphasise edges and invert map colours
        map_final = invert_values_and_emphasis_edges(map_inpainted, edges)

        Scale down map dimensions
        mini_map = resize(map_final, mini_x, mini_y)

        RETURN mini_map
END FUNCTION
```

Figure 16. Pseudocode for MapGenerator.

3.3.4 MapStitcher

Depth sensor range is limited and therefore it limits the altitude of which UAVs can perform target area mapping. Thus, single depth image does only cover an area, which is likely to be less than what is needed to create a path from UGV's initial position to target position. For this purpose, is MapStitcher, a method to aggregate a comprehensive map of designated area

out of the maps generated from individual depth camera images by MapGenerator. This subsection introduces MapSticher module of the method.

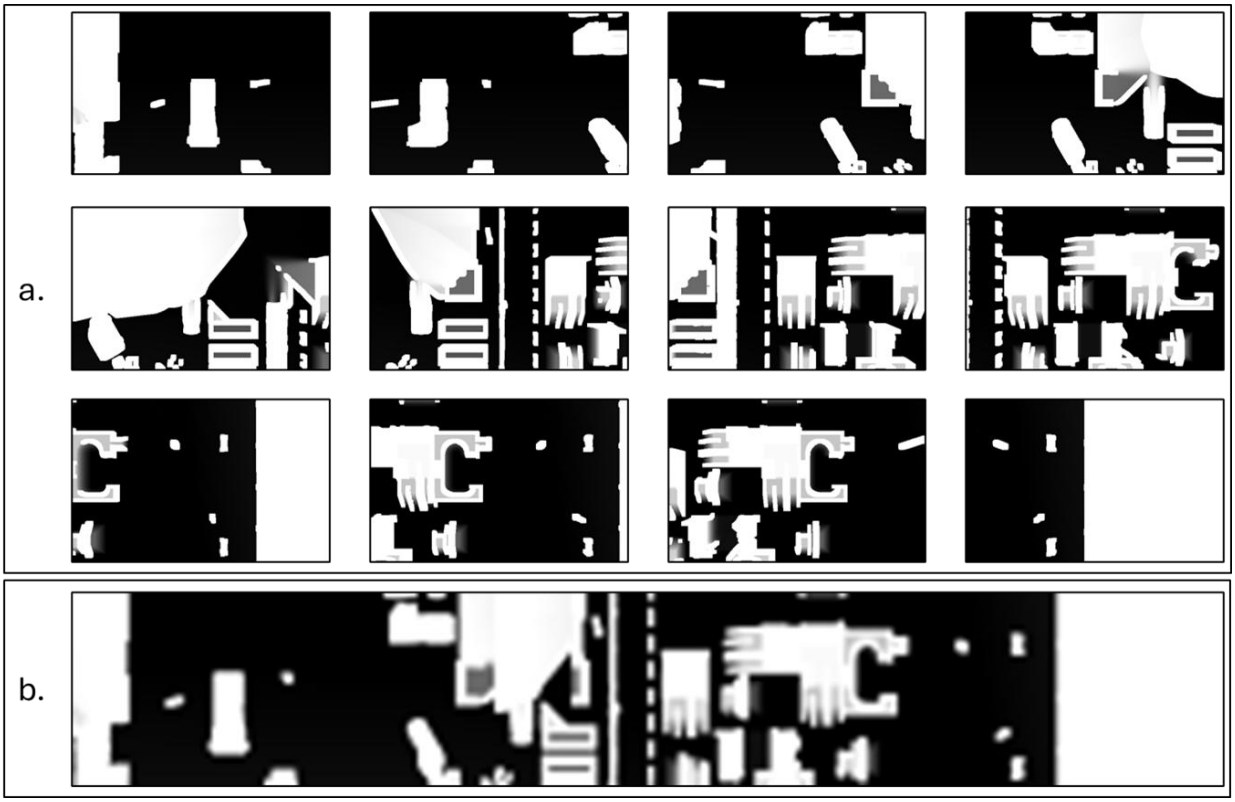

Figure 17. In frame a. is a sequence of partial maps each presenting a single depth image from UAV. In this example partial maps are not resized down to demonstrate how the process is a capable to preserve details even with radical down scaling. In frame b. is a stitched map stripe from the first phase of the MapStitcher. This map is built from MapGenerator mini maps that are scaled down to size of 64x40 pixels and thus this stitched map stripe is 240x40 pixels.

MapStitcher takes mini maps generated by MapGenerator as an input. It handles files according to coordinates included in the folder and file names. Therefore, it is necessary that naming system is intact throughout the mapping process. In the first phase MapStitcher concatenates images along the local x-axis to form wide map stripe out of each y-axis step. In figure 17 is example of phase 1. There is perspective distortion the further the pixel is from the centre point of the image. For the map this distortion is unwanted phenomena as it will warp objects to be different scale and twist them which might cause additional false obstacles or remove portion of true obstacles in the UGV walkable zone. In concatenating process MapStitcher uses only the middle rows of mini map image files given as input to build the aggregated map stripe from like demonstrated in figure 18. This is to mitigate this distortion effect as distortion is mildest close to the image centre. Although it is not preferrable to use the edge of the mini map image, for both edges of the aggregated map stripe, left and right,

MapStitcher needs to use that edge of the image despite the distortion to complete both sides as widely as there is information available. Thus, edges of the aggregated map do have distortion of the original input mini maps though distortions are filtered from the other areas.

Choosing the width of the area of the input map to use for the aggregated image depends on the input size. This is also tied to distance between each depth image captured UAV. These steps have different requirement for x and y-axis of local coordinate system if aspect ratio of a map image is not 1:1. Table 1. represents the relationship between different aspect rations, field of view, dimensions of captured area and dimensions that each pixel project from the real world. In this study image aspect ratio 16:10 was chosen as it was ratio used by several of commercial depth cameras. Using this ratio from UAV and thus sensor altitude of 20 meters above the target area, captured area width (x-axis) is 40 meters and length (y-axis) is approximately 21,38 meters. However, these dimensions apply as such only in ideal situation where area below UAV is equally flat and thus, simple triangulation applies. Although such completely flat area is special case, method is robust to slight variations of true dimensions captured. This robustness is result of the cherry-picking approach of the MapSticher method. As the method only uses the middle parts of each map, the resulted aggregated map is somewhat invariant for the accuracy of dimensions in the extreme ends of the input maps.

Table 1. Map dimensions

A chart to compare aspect ratio and resolution effect to map projection measurements. All figures in this chart expect flight altitude of UAV to be 20 meters. Image in m represents total size of area in x and y-axis that is captured by UAV from 20 meters above. Pixel in cm describes the dimensions in real world summarized in a pixel in the map projection.

Aspect ratio		Resolution		FOV		Image in m		Pixel in cm	
X	Y	X	Y	X	Y	X	Y	X	Y
16	9	1920	1080	90	50,625	40	18,92	2,1	1,8
4	3	800	600	90	67,5	40	26,73	5,0	4,5
16	10	1920	1200	90	56,25	40	21,38	2,1	1,8
16	10	1280	800	90	56,25	40	21,38	3,1	2,7
16	10	160	100	90	56,25	40	21,38	25,0	21,4
16	10	80	50	90	56,25	40	21,38	50,0	42,8
16	10	40	25	90	56,25	40	21,38	100,0	85,5
16	10	96	60	90	56,25	40	21,38	41,7	35,6
16	10	64	40	90	56,25	40	21,38	62,5	53,5
16	10	32	20	90	56,25	40	21,38	125,0	106,9

As MapSticher uses only centre area of each input map, these maps need to be overlapping to avoid caps. Therefore, UAV needs to move in steps or at least capture images in steps that

ensure overlapping. Distance of these steps from each other depend on the dimensions of the area that that each image represents. Although accuracy of dimensions is prone to have variety, step length should remain as accurate as possible in the local coordinate system of the target area to be mapped. Thus, relation of depth image centre points must be fixed and therefore, form a rigid grid. These, centre points are used for naming the images in the depth image phase and coordinate based names are used throughout the process all the way to MapSticher where maps are arranged according to these coordinates within the names.

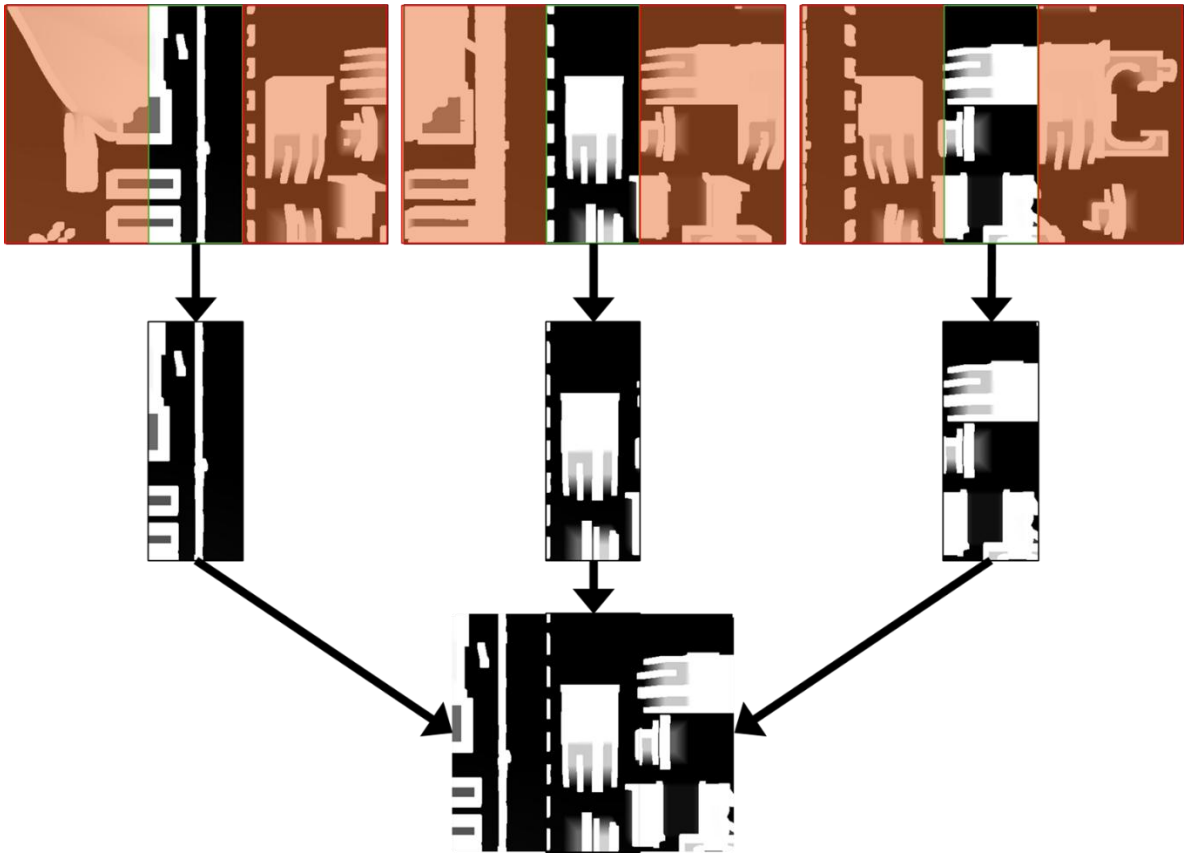

Figure 18. MapStitcher using only central columns of mini maps while generating aggregated map stripe along x-axis. Perspective distortion is mitigated the closer the pixel is of the centre point of the image and thus distortion is less in the aggregated map than in the original mini maps.

When optimal step length for UAV image capturing is defined, pixel count of the mini maps produced by MapGenerator, must be considered. Mini map size is destined to be very small for optimal performance. Therefore, limited pixel count affects to how mini map image matrix can be divided systematically into three parts: two edges and centre part. This division is to be done in a manner that is based on percent share of the width of the map in the x-axis concatenating phase and height in the y-axis aggregating phase. Although percentage share approach allows automatic method adaptation to different scales of mini maps, it sets few

limitations for the scale options as well as it dictates the UAV mapping step distances. For the map scale limitation is due to the need to have map split in discrete count of pixel columns when stitching on x-axis and pixel rows when stitching on y-axis. Thus, steps defined by percentage share of map image, must result in integer count of columns and rows. UAV mapping step length needs to be set according to MapStitcher image aggregation settings so that the centre areas that are concatenated, are not overlapping nor there is cap in between once the edge area of the image is removed. As a result of testing optimal results were achieved by using a portion of the image centre that was 25 percentage of the total image width while concatenating along x-axis and same share of image height while doing the same operation along y-axis. Therefore, when starting from left edge, the first mini map image is used all the way until 62,5 % of pixel columns are embedded to the aggregated map stripe. After the first image, from following images 37,5 % of pixel columns from both left and right edges are removed and only remaining 25 % of pixel columns in the middle are used for the aggregated map. However, for the right most mini map image, only the first 37,5 % is removed and the rest is used for the concatenation. Similar procedure applies when aggregated map strips are stitched together from top to down along y-axis.

Using one fourth of the image centre sets framework to define UAV movement steps in local coordinates. As stated earlier in Table 1, UAV (sensor) altitude 20m and aspect ratio of 16 x 10 equals projection of ground area of 40,00m x 21,38m to image. Therefore, moving UAV 10 meters along the x-axis, does equal 25% movement of the sensor and thus image centre. For y-axis 25 % movement would be approximately 5,345m step length. However, practical trials showed that despite theoretical ideal step would be 5,345m, better results were achieved by using 5,5m steps. Reason for this remains unknown for now.

MapSticher has two phases. First, it concatenates map stripe out of the mini maps in each folder named according to y-axis coordinates of UAV depth image capturing location in local coordinates. These stripes are concatenated in order according x-axis coordinates. Method then saves these stripes in new folder where each stripe is named according to y-axis coordinate of the stripe. In the second phase, MapStitcher concatenates map stripes using same process and principles that was used to generate map stripes from the mini maps. Outcome of this second phase is aggregated map that represents the whole area defined to be mapped to find path from the initial UGV position to the target location. Figures 19 and 20 show pseudocode for MapStitcher phases 1 and 2.

MapStitcher Phase 1

```
FUNCTION MapStitcher_phase_1:
        FOR each Y position in mapped area:

                ***Load all mini maps for this Y position and arrange according to X value***
                image_files = load_and_arrange_images(Y_position)
                IF no images found:
                        CONTINUE to next Y position

                ***Get dimensions of the mini maps***
                height, width = get_dimensions(first_map)

                ***Create empty canvas for stitched map stripe***
                stitched_map = create_empty_canvas(height, ((num_images-2) * 25% of width + 1,25% of width)

                ***Stitch images with 75% overlap (horizontally (37,5% for the edge images)***
                FOR each image in image_files:
                        IF first image:
                                place left 62,5% of image at the left end of the canvas
                        ELSE IF not last image:
                                place center 25% of image right side of previously filled columns
                        ELSE:
                                place right 62,5% of image at the right end of the canvas

                ***Save stitched result***
                save(stitched_map, output_path)

        END FOR
END FUNCTION
```

Figure 19. Pseudocode for phase 1 of MapStitcher.

MapStitcher Phase 2

```
FUNCTION MapStitcher_phase_2:

        Load all map stripes and arrange according to Y value
        image_files = load_and_arrange_images(map_stripes)

        Get dimensions of the mini maps
        height, width = get_dimensions(first_map)

        Create empty canvas for final map
        final_map = create_empty_canvas( ((num_images-2) * 25% of height + 1,25% of height, width)

        Stitch map stripes with 75% overlap (vertically) (37,5% for the edge maps)
        FOR i = 0 TO number_of_maps:
                map_strip = load_image(map_files_sorted[i])

                IF first map:
                        place top 62,5% of map stripe at top of the canvas
                ELSE IF not last map:
                        place middle 25% of map stripe below previously filled rows
                ELSE:
                        place bottom 62,5% of map stripe at bottom of the canvas

        END FOR

        RETURN final_map
END FUNCTION
```

Figure 20. Pseudocode for phase 2 of MapStitcher.

3.3.5 PathGenerator

To navigate the UGV to target, PathGenerator module generates path to follow in on the aggregated map from MapStitcher method. Although operator can adjust preliminary settings for MapGenerator to consider manoeuvrability of UGV, PathGenerator allows to fine tune sensitivity to height differences for optimal path according to UGV capabilities. To optimize speed of this method aggregated map size is to be kept small. This allows to have update frequency high and therefore, allowing path adjustments to changes in the environment. However, it is recommended for UGV to have local navigation capabilities that allow to react to unexpected obstacles and proceed with obstacle avoidance procedures.
PathGenerator is built around A* search algorithm using Manhattan distance as heuristic. Main adjustment for the vanilla version of the A* is distance cost function. Cost between neighbouring pixels can be set with Cost Multiplier to one of the following:

1. Fixed
2. Linear

3. Power of 2
4. X^X

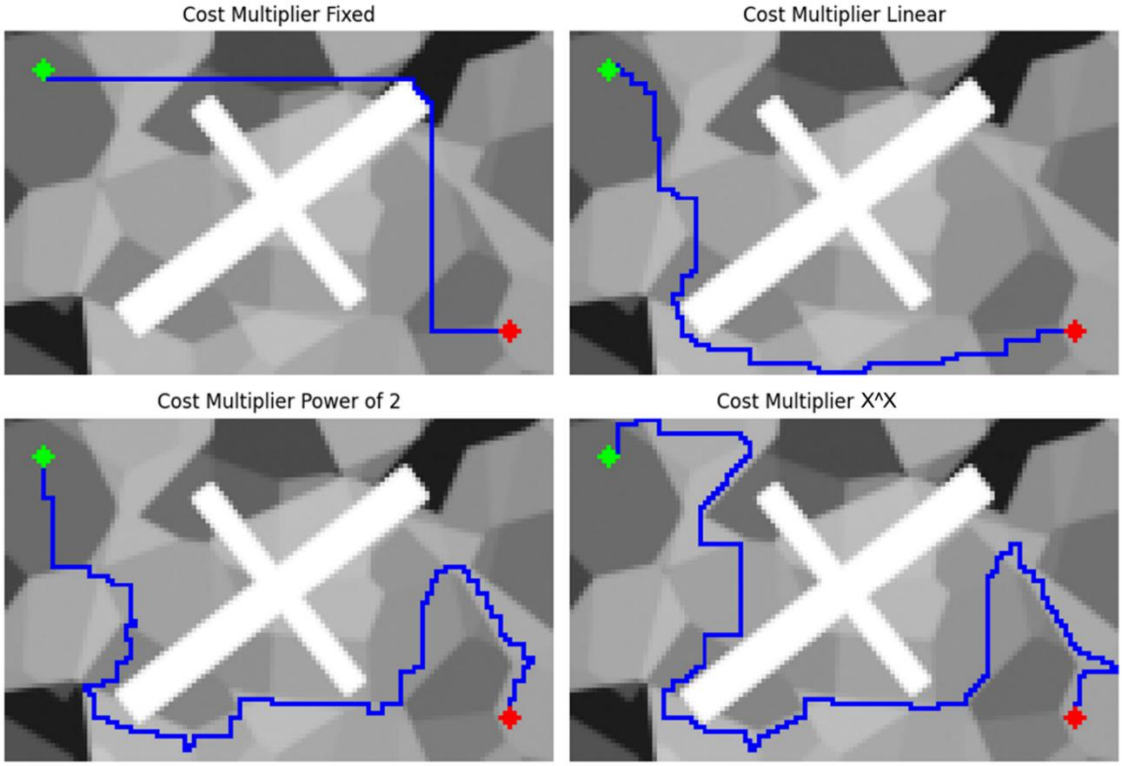

Figure 21. Path generated from fixed start and finish locations on a test map using different cost slopes for pixel value differences: Fixed, Linear (doubling), Power of 2 and X^X. As hue describes hight level of a pixel in comparison to other pixels, it is preferable to find a path which will avoid drastic hue differences between pixels and thus too steep slopes though it may be longer.

In Fixed mode, cost of moving to any neighbouring pixel is 1 and thus the value of a pixel representing the height level of the area mapped is invariant for A* search algorithm. Therefore, the result will be shortest route from UGV position to target position. However, in addition to Cost Multiplier, there is a threshold setting which allows to determine value limit that sets the defined value and all values higher to present obstacles. This threshold overdrives the cost function and thus, even path with Cost Multiplier set to Fixed, must go around pixels with value equal to threshold or higher. In Linear mode, cost is calculated by multiplying value difference of neighbouring pixels by 2. Therefore, cost increases linear to the value difference of the current pixel and neighbouring pixel, causing A* to avoid radical height differences when calculating the optimal path. Power of 2 mode does as name suggests and increases step cost with parabolic curve when value difference grows by raising the value difference of neighbours to the second power. This results in path, that is more sensitive to height differences than a path generated with Linear mode. The steepest curve comes with X^X mode where difference is raised to the power of itself i.e. x^x. The cost does increase even

faster than in Power of 2 mode making this mode the most sensitive of the available modes. Modes allows operator to balance path generation between shortest route and with smoothest route depending on UGV's manoeuvre capabilities to find optimal path for the given circumstances. Figure 21 demonstrates effect of Cost Multiplier to the planned path on synthetic map with variety of ground heights and pseudocode for PathGenerator is shown in figure 22.

PathGenerator based on A*algorithm

```
FUNCTION reconstruct_path(came_from, current)
        total_path = {current}
        WHILE current in came_from:
                current = came_from[current]
                total_path.prepend(current)
        return total_path

FUNCTION PathGenerator(
        map_image,                  # Final map generated by MapSticher
        walkable_limit,             # Threshold value for max height reachable for UGV path
        start,                      # Initial location pixel of UGV
        goal,                       # Target pixel for UGV
        heurestic_distance,         # Function to estimate the shortest path between start and goal
        cost_multiplier):           # Set cost curve steepness for movement between different heights

        Initialize A* search
        open_set = priority_queue()
        open_set.push(start, heuristic_distance(start, goal))

        came_from = an empty dictionary
        g_score[start] = 0
        f_score[start] = heuristic_distance(start, goal)
        WHILE open_set is not empty:
                current = open_set.pop_lowest_cost_node()

                IF current == goal:
                        RETURN reconstruct_path(came_from, current)

                Check all neighbours (up, down, left, right when using Manhattan distance as heuristic)
                FOR each neighbour in get_neighbours(current):
                        IF neighbour is walkable (pixel_value <= walkable_limit):
                                Calculate movement cost based on terrain difference
                                terrain_cost = calculate_cost(pixel_difference, cost_multiplier)
                                new_cost = g_score[current] + terrain_cost

                                If better path found, update
                                IF new_cost < g_score[neighbour]:
                                        came_from[neighbour] = current
                                        g_score[neighbour] = new_cost
                                        f_score[neighbour] = new_cost + heuristic_distance(neighbour, goal)
                                        IF neighbour IS NOT in open_set:
                                                open_set.push(neighbour, f_score)
                END FOR

        END WHILE

        RETURN None                 # No path found
END FUNCTION
```

Figure 22. Pseudocode for PathGenerator.

As aggregated map is to be downscaled to improve processing speed, area that pixel projects go up in comparison to original images. Table 1. shows example measures of the pixel projection to real world measurements. As an example, if final aggregated map is generated from mini maps sized 32 x 20 pixels, each pixel projects space in real world that is about 125 cm in x axis and 107 cm in y axis of the local coordinates. This method does not have safe margin around the path for the UGV width, and that is to be taken as a development task for future iterations. However, with scaled down maps, one pixel width path offers already width zone for UGV to travel in and correct trajectory with local navigation system based on vehicles own sensors.

Final outcome of the PathGenerator is a path added to the aggregated map. This map is to be used for navigation of the UGV. Figure 23 shows this final outcome in comparison to aerial RGB image of the same area. Although path may be optimal on the first iteration, looping the UAV mapping phase of the concept followed by the MapGenerator, MapStitcher and PathGenerator phases, does allow critical changes in environment to be taken into account for the generated path as UGV advances further. When resources like count of available UAVs are limited and there are sectors of the target area that are more dynamic than others, modularity of map and the method enable path to be updated constantly on these critical points only. Therefore, for example if there is a street with active traffic UAV can be set to constantly update that point in the local coordinate grid. Thus, only mini map of this particular sector is updated, and therefore refresh loop is rapid allowing close to real-time path adaptation.

Figure 23. Above is a cropped view from aerial 8k image taken from 250m above the target area. Green tag shows the initial location of the UGV and red tag the location of the target. Below is the path for UGV generated with the PathGenerator placed on the map generated by MapGenerator and MapStitcher.

4 Discussion

This thesis introduced a high-level concept for search and rescue missions where one or more UAVs are used in collaboration with single or multiple UGVs. In addition to this concept, this thesis defines modular method for map generation and path planning to be used in this concept. This method is semi-autonomous as it can be fine-tuned in initializing phase of the mission according to the hardware features by human supervisor or operator, but it can set to run autonomously during the mission.

Concept is meant to be guideline in broader scope of the mission. However, the scope of the concept is too wide for the scope of the master's thesis to be defined on technical level in all detail. Therefore, this offers opportunities for further studies around the subject while this thesis focuses on the mapping of the area of the interest and planning a path for UGV on that map.

The mapping and path planning method modularity allows improvements to be made for any of the three modules separately from other modules if input and output follow the same logic as now. Therefore, for example, switching path planning algorithm from A* search algorithm to some other algorithm does not cause issues for the pipeline if input and output of the module remain untouched. Thus, optimizing the method according to mission needs is possible.

Developed method offers solutions for some of the shortcomings of methods introduced in previous studies. This method does offer robustness when it comes to lighting conditions, when using depth sensors that are not relying on external light source. Therefore, shadows are not shaping perception as those are when relying on algorithm using RGB camera input. Sophisticated deep learning algorithms for computer vision like YOLO can be trained to detect effectively all desired objects. However, real-world scenarios seldom are without irregular shaped and sized objects and features that have not been in training material and thus may be hard to be understood as obstacles by the algorithm. Therefore, using simpler computer vision approaches like edge and corner detectors offers more foundational and thus robust solution to detect potential obstacles. However, this does not apply for RGB view as world is full of textures that will cause simple edge detectors to be useless. Using edge detectors with depth camera image feed solves this problem by only taking in consideration edges according to spatial differences, not colouring. Also, sensitivity for edge detection can

be adjusted, like it is adjusted in the presented method, to define what kind of height differences are to be considered as relevant obstacles and what is irrelevant for the UGV movement.

Research was conducted using simulator. Although Carla 0.10.0 can be considered high quality simulation environment for this purpose, it is still a simulator and lacks real world features that do affect for the execution of the collaborative mission. Carla 0.10.0 is casting realistic lighting conditions and casting real looking shadow according to current location of sun and other light sources in the scenario. It allows simulation of dynamic environment as car and pedestrian traffic can be set to the scene while UAV updates images to the map and path generation pipeline. Urban area in the simulator is rich of shapes and objects of many types decorated with rich realistic looking textures and therefore, RGB processing algorithms would have plenty of challenges that would occur in real-world too. However, using depth camera simulation, depth sensor, may give too optimistic result in comparison to state-of-the-art real depth cameras. Therefore, a potential subject for future study is to test introduced algorithm in real-world environment with several depth cameras using different technologies.

In this thesis actual movement of UAVs is not in the scope. However, for the algorithm to work perfectly, method expects UAVs to be able to capture images from locations that are accurately specified in relation to each other. In simulator this is easy as exact location for UAV can be set. However, in real-world scenario precise locating of UAV can be hard even in optimal conditions. Therefore, in windy conditions and under interference on electronical positioning systems like in warzone setting UAV to the precise location it is destined for, may be impossible to achieve. Thus, there is need to improve the algorithm robustness for the map stitcher to be less sensitive for minor offsets in location where image has been captured.

Method introduced here is built to be adaptable. Modularity enables easy modification of each of the three modules: MapGenerator, MapStitcher and PathGenerator. Although, development work here was done with depth camera input to MapGenerator, method can be adjusted to take in point cloud data and thus for 2D laser scanner or lidar. These could increase the maximum altitude for UAVs and improve robustness to lighting and even weather conditions. However, active sensors are detectable and are therefore mostly applicable for scenarios where there is no hostile activity to be expected if detected.

As this work focused on map and path generating, an important aspect of implementation of localization for the methodology is left for future studies. This concerns both UAV and UGV

localization. Overall, scope of the study was around UGV navigation and thus neglecting entirely UAV navigation although that is vital for the method to work as the MapStitcher relies on images to be taken in specified distance from each other. To make this step more robust, adding some image matching method in to the stitcher module to be less sensitive to have mini maps from exact locations. However, this is also left for future studies.

One of the key features of this work is false obstacle layer removal with FOZ-filtering. As view from the above may blocked by objects that are irrelevant for the UGV on ground like foliage and powerlines, it is relevant to try to remove these from the UGV map. MapGenerator does do this effectively at least in simulator conditions. This is one the key features for future studies to test in real-world scenarios.

Another key feature is modifying A* to use slope steepness as cost factor. Therefore, PathGenerator parameters can be set according to UGV capabilities to either plan shortest path that allows some height difference between grid cells to on the route, or to plan longer but smoother path.

5 Conclusion

This thesis proposes a high-level architecture for search and rescue operation that is to be conducted by one or more UAVs and UGVs. In addition, detailed modular method for map and path generation based on depth image input from UAV is proposed. This map to path pipeline does not only provide an obstacle free path for UGV, but a route that is optimized to prioritize slopes with incline optimal to the specific UGV manoeuvrability. Although reviewing literature reveals approaches that have some similarities with some of the parts of the method that I propose, to the best of my knowledge, no prior work has combined depth map with false obstacle layer removal in this way relying only map provided by UAV like FOZ-filter does. Solution allows UAV to provide map for UGV even in offroad areas where foliage could block UAV mapping without false obstacle layer removal.

Key contributions of this thesis:

- A concept for operating model framework for search and rescue missions using UAV-UGV collaboration
- Modular map and path generating method based on depth map from above
- A mapping method that can be implemented without active sensors
- FOZ-filter, method that removes false obstacles including foliage and powerlines from the map layer that is above the level of which UGV moves
- PathGenerator that allows prioritizing slopes versus flat areas or vice versa on the route according to maneuverability of the UGV in use

References

[1] M. Lyu, Y. Zhao, C. Huang, and H. Huang, “Unmanned Aerial Vehicles for Search and Rescue: A Survey,” *Remote Sens.*, vol. 15, no. 13, p. 3266, Jun. 2023, doi: 10.3390/rs15133266.

[2] I. Munasinghe, A. Perera, and R. C. Deo, “A Comprehensive Review of UAV-UGV Collaboration: Advancements and Challenges,” *J. Sens. Actuator Netw.*, vol. 13, no. 6, p. 81, Nov. 2024, doi: 10.3390/jsan13060081.

[3] S. A. H. Mohsan, N. Q. H. Othman, Y. Li, M. H. Alsharif, and M. A. Khan, “Unmanned aerial vehicles (UAVs): practical aspects, applications, open challenges, security issues, and future trends,” *Intell. Serv. Robot.*, vol. 16, no. 1, pp. 109–137, Mar. 2023, doi: 10.1007/s11370-022-00452-4.

[4] Z. Zhang and L. Zhu, “A Review on Unmanned Aerial Vehicle Remote Sensing: Platforms, Sensors, Data Processing Methods, and Applications,” *Drones*, vol. 7, no. 6, p. 398, Jun. 2023, doi: 10.3390/drones7060398.

[5] F. Ahmed, J. C. Mohanta, A. Keshari, and P. S. Yadav, “Recent Advances in Unmanned Aerial Vehicles: A Review,” *Arab. J. Sci. Eng.*, vol. 47, no. 7, pp. 7963–7984, Jul. 2022, doi: 10.1007/s13369-022-06738-0.

[6] C. Liu, J. Zhao, and N. Sun, “A Review of Collaborative Air-Ground Robots Research,” *J. Intell. Robot. Syst.*, vol. 106, no. 3, p. 60, Nov. 2022, doi: 10.1007/s10846-022-01756-4.

[7] L. Bruzzone and G. Quaglia, “Review article: locomotion systems for ground mobile robots in unstructured environments,” *Mech. Sci.*, vol. 3, no. 2, pp. 49–62, Jul. 2012, doi: 10.5194/ms-3-49-2012.

[8] P. Fankhauser *et al.*, “Collaborative navigation for flying and walking robots,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Daejeon, South Korea: IEEE, Oct. 2016, pp. 2859–2866. doi: 10.1109/IROS.2016.7759443.

[9] Y. Liu, J. Liu, Z. He, Z. Li, Q. Zhang, and Z. Ding, “A Survey of Multi-Agent Systems on Distributed Formation Control,” *Unmanned Syst.*, vol. 12, no. 05, pp. 913–926, Sep. 2024, doi: 10.1142/S2301385024500274.

[10] P. Sherman and N. Bezzo, “A Heterogeneous System of Systems Framework for Proactive Path Planning of a UAV-assisted UGV in Uncertain Environments,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Abu Dhabi, United Arab Emirates: IEEE, Oct. 2024, pp. 13237–13244. doi: 10.1109/IROS58592.2024.10801791.

[11] S. Xu, Z. Zhou, H. Liu, X. Zhang, J. Li, and H. Gao, “A Path Planning Method for Collaborative Coverage Monitoring in Urban Scenarios,” *Remote Sens.*, vol. 16, no. 7, p. 1152, Mar. 2024, doi: 10.3390/rs16071152.

[12] N. Michael *et al.*, “Collaborative mapping of an earthquake-damaged building via ground and aerial robots,” *J. Field Robot.*, vol. 29, no. 5, pp. 832–841, Sep. 2012, doi: 10.1002/rob.21436.

[13] J. Zhang, X. Yue, H. Zhang, and T. Xiao, “Optimal Unmanned Ground Vehicle—Unmanned Aerial Vehicle Formation-Maintenance Control for Air-Ground Cooperation,” *Appl. Sci.*, vol. 12, no. 7, p. 3598, Apr. 2022, doi: 10.3390/app12073598.

[14] J. Li, G. Deng, C. Luo, Q. Lin, Q. Yan, and Z. Ming, “A Hybrid Path Planning Method in Unmanned Air/Ground Vehicle (UAV/UGV) Cooperative Systems,” *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9585–9596, Dec. 2016, doi: 10.1109/TVT.2016.2623666.

[15] E. Yanmaz, S. Yahyanejad, B. Rinner, H. Hellwagner, and C. Bettstetter, “Drone networks: Communications, coordination, and sensing,” *Ad Hoc Netw.*, vol. 68, pp. 1–15, Jan. 2018, doi: 10.1016/j.adhoc.2017.09.001.

[16] “ERS Radar Course 3 - Earth Online.” Accessed: May 01, 2026. [Online]. Available: https://earth.esa.int/eogateway/missions/ers/radar-courses/radar-course-3

[17] F. Bovenga, “Special Issue ‘Synthetic Aperture Radar (SAR) Techniques and Applications,’” *Sensors*, vol. 20, no. 7, p. 1851, Mar. 2020, doi: 10.3390/s20071851.

[18] Q. Liu, Z. Li, S. Yuan, Y. Zhu, and X. Li, “Review on Vehicle Detection Technology for Unmanned Ground Vehicles,” *Sensors*, vol. 21, no. 4, p. 1354, Feb. 2021, doi: 10.3390/s21041354.

[19] L. Zhu, J. Suomalainen, J. Liu, J. Hyyppä, H. Kaartinen, and H. Haggren, "A Review: Remote Sensing Sensors," in *Multi-purposeful Application of Geospatial Data*, R. B. Rustamov, S. Hasanova, and M. H. Zeynalova, Eds., InTech, 2018. doi: 10.5772/intechopen.71049.
[20] C. Ersü, E. Petlenkov, and K. Janson, "A Systematic Review of Cutting-Edge Radar Technologies: Applications for Unmanned Ground Vehicles (UGVs)," *Sensors*, vol. 24, no. 23, p. 7807, Dec. 2024, doi: 10.3390/s24237807.
[21] R. Roriz, J. Cabral, and T. Gomes, "Automotive LiDAR Technology: A Survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 6282–6297, Jul. 2022, doi: 10.1109/TITS.2021.3086804.
[22] S. Zhang, W. Xu, Z. Wei, L. Zhang, Y. Wang, and J. Liu, "ARAI-MVSNet: A multi-view stereo depth estimation network with adaptive depth range and depth interval," *Pattern Recognit.*, vol. 144, p. 109885, Dec. 2023, doi: 10.1016/j.patcog.2023.109885.
[23] J. L. Schönberger, E. Zheng, J.-M. Frahm, and M. Pollefeys, "Pixelwise View Selection for Unstructured Multi-View Stereo," in *Computer Vision – ECCV 2016*, vol. 9907, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., in Lecture Notes in Computer Science, vol. 9907. , Cham: Springer International Publishing, 2016, pp. 501–518. doi: 10.1007/978-3-319-46487-9_31.
[24] M. Aboali, N. A. Manap, A. M. Darsono, and Z. M. Yusof, "Review on Three-Dimensional (3-D) Acquisition and Range Imaging Techniques," vol. 12, no. 10, 2017.
[25] W. Xuan and Y. Ding, "Path Planning for Road Congestion Problems Based on the A* Algorithm," *J. Phys. Conf. Ser.*, vol. 2670, no. 1, p. 012011, Dec. 2023, doi: 10.1088/1742-6596/2670/1/012011.
[26] Abhijit Gadekar, "Recent developments in modular unmanned ground vehicles: A review," *Asia-Pac. J. Sci. Technol.*, vol. 29, p. 11, 2024, doi: 10.14456/APST.2024.1.
[27] R. Wang, K. Wang, W. Song, and M. Fu, "Aerial-Ground Collaborative Continuous Risk Mapping for Autonomous Driving of Unmanned Ground Vehicle in Off-Road Environments," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 59, no. 6, pp. 9026–9041, Dec. 2023, doi: 10.1109/TAES.2023.3312627.
[28] H. Taheri and Z. C. Xia, "SLAM; definition and evolution," *Eng. Appl. Artif. Intell.*, vol. 97, p. 104032, Jan. 2021, doi: 10.1016/j.engappai.2020.104032.
[29] T. I. Zohdi, "Multiple UAVs for Mapping: A Review of Basic Modeling, Simulation, and Applications," *Annu. Rev. Environ. Resour.*, vol. 43, no. 1, pp. 523–543, Oct. 2018, doi: 10.1146/annurev-environ-102017-025912.
[30] W. Chen *et al.*, "SLAM Overview: From Single Sensor to Heterogeneous Fusion," *Remote Sens.*, vol. 14, no. 23, p. 6033, Nov. 2022, doi: 10.3390/rs14236033.
[31] H. Qin *et al.*, "Autonomous Exploration and Mapping System Using Heterogeneous UAVs and UGVs in GPS-Denied Environments," *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1339–1350, Feb. 2019, doi: 10.1109/TVT.2018.2890416.
[32] Y. Fan, Q. Zhang, Y. Tang, S. Liu, and H. Han, "Blitz-SLAM: A semantic SLAM in dynamic environments," *Pattern Recognit.*, vol. 121, p. 108225, Jan. 2022, doi: 10.1016/j.patcog.2021.108225.
[33] W. Chen *et al.*, "An Overview on Visual SLAM: From Tradition to Semantic," *Remote Sens.*, vol. 14, no. 13, p. 3010, Jun. 2022, doi: 10.3390/rs14133010.
[34] S. Banik, S. C. Banik, and S. S. Mahmud, "Path Planning Approaches in Multi-robot System: A Review," *Eng. Rep.*, vol. 7, no. 1, p. e13035, Jan. 2025, doi: 10.1002/eng2.13035.
[35] V. A. Armentano, A. L. Shiguemoto, and A. Løkketangen, "Tabu search with path relinking for an integrated production–distribution problem," *Comput. Oper. Res.*, vol. 38, no. 8, pp. 1199–1209, Aug. 2011, doi: 10.1016/j.cor.2010.10.026.
[36] H. Guo, Z. Mao, W. Ding, and P. Liu, "Optimal search path planning for unmanned surface vehicle based on an improved genetic algorithm," *Comput. Electr. Eng.*, vol. 79, p. 106467, Oct. 2019, doi: 10.1016/j.compeleceng.2019.106467.
[37] J. Wang, K. Yang, B. Wu, and J. Wang, "Cooperative Path Planning for Persistent Surveillance in Large-Scale Environment with UAV - UGV System," *IEEJ Trans. Electr. Electron. Eng.*, vol. 19, no. 12, pp. 1987–2001, Dec. 2024, doi: 10.1002/tee.24157.
[38] "A* search algorithm," *Wikipedia*. Jan. 11, 2026. Accessed: Feb. 08, 2026. [Online]. Available: https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=1335609515

[39] C. Zhu, “Current Study on A* Algorithm in Autonomous Obstacle Avoidance,” *Highlights Sci. Eng. Technol.*, vol. 97, pp. 328–332, May 2024, doi: 10.54097/60sxg682.
[40] Y. Du, “Multi-UAV Search and Rescue with Enhanced A ∗ Algorithm Path Planning in 3D Environment,” *Int. J. Aerosp. Eng.*, vol. 2023, pp. 1–18, Feb. 2023, doi: 10.1155/2023/8614117.
[41] D. T. Xuan, N. T. Hung, and V. T. Thang, “A Comprehensive Review of Improved A* Path Planning Algorithms and Their Hybrid Integrations,” *Automation*, vol. 6, no. 4, p. 52, Oct. 2025, doi: 10.3390/automation6040052.
[42] “Heuristics.” Accessed: Feb. 22, 2026. [Online]. Available: https://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
[43] “OpenCV: Image Inpainting.” Accessed: Feb. 24, 2026. [Online]. Available: https://docs.opencv.org/3.4/df/d3d/tutorial_py_inpainting.html
[44] P. Chatterjee, S. Jana, and S. Ghosh, “Comparative Study of OpenCV Inpainting Algorithms,” 2021.
[45] A. Telea, “An Image Inpainting Technique Based on the Fast Marching Method,” *J. Graph. Tools*, vol. 9, no. 1, pp. 23–34, Jan. 2004, doi: 10.1080/10867651.2004.10487596.
[46] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, “Navier-stokes, fluid dynamics, and image and video inpainting,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA: IEEE Comput. Soc, 2001, p. I-355-I–362. doi: 10.1109/CVPR.2001.990497.
[47] moukthika, “Edge Detection Using OpenCV,” OpenCV. Accessed: Mar. 01, 2026. [Online]. Available: https://opencv.org/edge-detection-using-opencv/
[48] H. Khudov, R. Khudov, I. Khizhnyak, I. Hridasov, and P. Hlushchenko, “THE SMALL AERIAL OBJECTS SEGMENTATION METHOD ON OPTICAL-ELECTRONIC IMAGES BASED ON THE SOBEL EDGE DETECTOR,” *Adv. Inf. Syst.*, vol. 9, no. 2, pp. 5–10, Apr. 2025, doi: 10.20998/2522-9052.2025.2.01.
[49] N. D. Lynn, A. I. Sourav, and A. J. Santoso, “Implementation of Real-Time Edge Detection Using Canny and Sobel Algorithms,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 1096, no. 1, p. 012079, Mar. 2021, doi: 10.1088/1757-899X/1096/1/012079.
[50] “Sobel operator,” *Wikipedia*. Jan. 01, 2026. Accessed: Mar. 08, 2026. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sobel_operator&oldid=1330598968
[51] P. Sikka, A. R. Asati, and C. Shekhar, “HIGH-SPEED and AREA-EFFICIENT Sobel edge detector on field-programmable gate array for artificial intelligence and machine learning applications,” *Comput. Intell.*, vol. 37, no. 3, pp. 1056–1067, Aug. 2021, doi: 10.1111/coin.12334.
[52] “Canny edge detector,” *Wikipedia*. Mar. 02, 2026. Accessed: Mar. 15, 2026. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Canny_edge_detector&oldid=1341276744
[53] J. Canny, “A Computational Approach to Edge Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.
[54] L. Ding and A. Goshtasby, “On the Canny edge detector,” *Pattern Recognit.*, vol. 34, no. 3, pp. 721–725, Mar. 2001, doi: 10.1016/S0031-3203(00)00023-6.
[55] “OpenCV: Eroding and Dilating.” Accessed: Mar. 15, 2026. [Online]. Available: https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
[56] R. Sapkota *et al.*, “YOLO advances to its genesis: a decadal and comprehensive review of the You Only Look Once (YOLO) series,” *Artif. Intell. Rev.*, vol. 58, no. 9, p. 274, Jun. 2025, doi: 10.1007/s10462-025-11253-3.
[57] Ultralytics, “Models Supported by Ultralytics.” Accessed: Mar. 24, 2026. [Online]. Available: https://docs.ultralytics.com/models/
[58] J. Sun, B. Li, Y. Jiang, and C. Wen, “A Camera-Based Target Detection and Positioning UAV System for Search and Rescue (SAR) Purposes,” *Sensors*, vol. 16, no. 11, p. 1778, Oct. 2016, doi: 10.3390/s16111778.
[59] A. Albanese, V. Sciancalepore, and X. Costa-Perez, “SARDO: An Automated Search-and-Rescue Drone-Based Solution for Victims Localization,” *IEEE Trans. Mob. Comput.*, vol. 21, no. 9, pp. 3312–3325, Sep. 2022, doi: 10.1109/TMC.2021.3051273.
[60] X. Zhang, P. Wang, and Y. Han, “A Path Planning Method for Unmanned Rescue Collaboration System,” in *2024 IEEE 19th Conference on Industrial Electronics and Applications (ICIEA)*, Kristiansand, Norway: IEEE, Aug. 2024, pp. 1–6. doi: 10.1109/ICIEA61579.2024.10664964.

[61] G. Martinez-Soltero, A. Y. Alanis, N. Arana-Daniel, and C. Lopez-Franco, “Semantic Segmentation for Aerial Mapping,” *Mathematics*, vol. 8, no. 9, p. 1456, Aug. 2020, doi: 10.3390/math8091456.

[62] J. H. Kim, J. Kwon, and J. Seo, “Multi-UAV-based stereo vision system without GPS for ground obstacle mapping to assist path planning of UGV,” *Electron. Lett.*, vol. 50, no. 20, pp. 1431–1432, Sep. 2014, doi: 10.1049/el.2014.2227.