Mälardalen University
School of Innovation Design and Engineering
Västerås, Sweden

Thesis for the Degree of Master of Science in Engineering -
Dependable Systems 30.0 credits

APPLYING UAVS TO SUPPORT THE SAFETY IN AUTONOMOUS OPERATED OPEN SURFACE MINES

Rasmus Hamren
rhn15001@student.mdh.se

Course: FLA500

Examiner: Håkan Forsberg
Mälardalen University, Västerås, Sweden

Supervisors: Baran Çürüklü
Mälardalen University, Västerås, Sweden

Company supervisor: Stephan Baumgart,
Volvo Autonomous Solutions, Eskilstuna, Sweden

February 3, 2021

Abstract

Unmanned aerial vehicle (UAV) is an expanding interest in numerous industries for various applications. Increasing development of UAVs is happening worldwide, where various sensor attachments and functions are being added. The multi-function UAV can be used within areas where they have not been managed before. Because of their accessibility, cheap purchase, and easy-to-use, they replace expensive systems such as helicopters- and airplane-surveillance. UAV are also being applied into surveillance, combing object detection to video-surveillance and mobility to finding an object from the air without interfering with vehicles or humans ground. In this thesis, we solve the problem of using UAV on autonomous sites, finding an object and critical situation, support autonomous site operators with an extra safety layer from UAVs camera. After finding an object on such a site, uses GPS-coordinates from the UAV to see and place the detected object on the site onto a gridmap, leaving a coordinate-map to the operator to see where the objects are and see if the critical situation can occur. Directly under the object detection, reporting critical situations can be done because of safety-distance-circle leaving warnings if objects come to close to each other. However, the system itself only supports the operator with extra safety and warnings, leaving the operator with the choice of pressing emergency stop or not. Object detection uses You only look once (YOLO) as main object detection Neural Network (NN), mixed with edge-detection for gaining accuracy during bird-eye-views and motion-detection for supporting finding all object that is moving on-site, even if UAV cannot find all the objects on site. Result proofs that the UAV-surveillance on autonomous site is an excellent way to add extra safety on-site if the operator is out of focus or finding objects on-site before startup since the operator can fly the UAV around the site, leaving an extra-safety-layer of finding humans on-site before startup. Also, moving the UAV to a specific position, where extra safety is needed, informing the operator to limit autonomous vehicles speed around that area because of humans operation on site. The use of single object detection limits the effects but gathered object detection methods lead to a promising result while printing those objects onto a global positions system (GPS) map has proposed a new field to study. It leaves the operator with a viewable interface outside of object detection libraries.

Keywords— Object Detection; You only look once (YOLO); Unmanned aerial vehicle (UAV); Situational Awareness; Critical situations; Autonomous vehicle; Critical Global Positioning System (GPS) mapping

Sammandrag

Drönare blir allt mer använda i många branscher för de möjligheterna till olika användningsområden. Mer och mer utveckling av drönare görs där olika sensorer och funktioner läggs till, där multifunktionella drönare kan användas inom områden där de inte har flugit tidigare. På grund av deras tillgänglighet, låga kostnad och flygvärdighet ersätter de dem dyra system som helikopterövervakning och flygövervakning. Drönare används också för övervakning då de lätt kan kamma av områden, ge information till föremålsavkänning för videoövervakning och drönarnas rörlighet gör det lätt att hitta ett föremål från luften utan att störa fordon eller människors som är på marken. I denna avhandling löser vi problem genom att använda drönare på autonoma områden, hittar objekt genom föremålsavkänning och motverkar kritiska situationer. Genom att stödja webbplatsoperatörer med ett extra säkerhets lager, tack vare drönarens kamera, kan vi höja säkerheten inom dessa områden. Efter att ha hittat ett objekt på en sådant område används GPS-koordinater från drönaren för att se vart objektet befinner sig och sedan placera det detekterade objektet på en 2D-karta. Koordinater beräknas för de detekterade objekten, och kartan visas för operatören, som kan se vart objekten befinner sig, och se om kritiska situationen kan uppstå. Direkt under objektdetekteringen kan kritiska situationer rapporteras på grund av att säkerhetsavstånden bryts, som ger varningar om objekt skulle komma för nära varandra. Systemet stöder dock endast operatören med extra säkerhet och varningar, vilket lämnar operatören till att göra valet om att trycka på nödstopp eller inte, vid en förmodan kritisk situation. Objektdetektering använder föremålsavkänningsnätverket YOLO som huvudobjektdetektering, som är ett neuralt nätverk, blandat med kantdetektering för att öka noggrannheten under fågelperspektiv-filmning, och även stöd av rörelsedetektering för att hitta alla objekt som rör sig på plats, även om YOLO kan inte hitta alla objekt på området. Resultatet visar att UAV-övervakning på autonoma platser är ett utmärkt sätt att höja säkerheten inom ett område om operatören exempelvis skulle vara ofokuserad, eller att hitta objekt på området före start. Operatören kan då flyga med drönaren runt på området och skapa en extra säkerhetsåtgärd för att hitta människor på plats före start. Möjligheten att också flytta drönaren till en specifik plats, där extra säkerhet behövs, skapar högre säkerhet för operatören genom att se och få information om området, och sedan kunna ta ett beslut om exempelvis begränsa autonoma fordonens hastighet runt det området på grund av människors verksamhet på just den platsen. Användningen av objektdetektering med enstaka metoder begränsar effekterna men flera kombinerade objektdetekteringsmetoder leder till ett lovande resultat. Objekt markeras på en GPS-karta, som inte har gjort tidigare, vilket skapar ett nytt fält att studera. Det lämnar operatören med ett bra gränssnitt utöver objektdetekteringsbiblioteken.

Acknowledgement

This thesis would not have been possible without Volvo CE, and I can not express my gratitude enough for the opportunity my supervisors Stephan Baumgart and Baran Çürüklü have been given me. I would not have accomplished this thesis without my supervisor, Volvo CE as company or software platform manager Andreas Deck. Without all of you, I would not have achieved my result or been able to try the opportunity to research within object detection category and safety layer assessments.

I also want to thank Nordic Electronic partner (NEP) for material such as FLIR-camera, and by this given me more inputs to prove my research. I also want to thank NEP for help, such as report-startup, office at MDH, and other support.

List of Figures

Glossary

CCTV Closed Circuit Television

AOZ Autonomous Operating Zone

UAV unmanned aerial vehicle

VAS Volvo autonomous solutions

YOLO You only look once

CNN Convolutional neural network

R-CNN Region-Based Convolutional Neural Network, Object detection Neural Network

OS Operative system

ISO International Organization for Standardization

MSAR Marine search and rescue

IR Infrared

FPS Frames per second

NN Neural Network

GPU Graphical Process Unit

CPU Central Process Unit

GPS Global Positioning System

WSL Windows Subsystem Linux

FLCC Flight Control Computer

Table of Contents

1. Introduction

Unmanned aerial vehicle (UAV) are growing importance in many industries for various applications. In the past, unmanned aircraft circled the sky mostly to support with photographically images [1] for military purposes. While balloons and airplanes have been expensive and not affordable for private users, small drones' development made it possible to own drones with cameras and sensors privately. From balloons to airplanes, from paper planes to own private drones, development from the basics to more advance UAVs than ever before, with cameras and sensors to support tasks and other more complex usages, such as agriculture usage and human saving operations.

UAVs are replacing expensive systems, such as helicopters to record videos, movies, or other replacement where complexity and cost have been limiting factors. UAVs fly autonomously and deliver, for example, package [2] or collect agricultural data [3]. Another application scenario is using object detection in images to sustain a safe environment for humans.

Many of today's commercially available drones are equipped with cameras as the primary sensor. This requires object detection algorithms to identify and track predefined and relevant objects. The advantage of UAVs is that they are flexible in their areal application. In Maritime Safe and Rescue (*Marine search and rescue* (MSAR)) operations, where humans at risk need to be located quickly in large maritime areas, a fleet of UAVs can support the rescue teams with scanning for the humans to rescue [4]. It is necessary to reliably identify objects and provide feedback to determine which area the rescue operation should focus on. Additional sensors like *Infrared* (IR) or enhanced algorithms can improve the object localization and tracking results.

Apart from autonomously operating UAVs, automating systems are considered in many domains. While automated cars operate as a single unit, transport solutions for goods require connected automated systems. In the earth moving machinery domain, autonomous vehicles for transporting material in open surface mines are developed, such as in the Electric Site research project by Volvo Construction Equipment [5]. In this project, a fleet of up to eight autonomous haulers has been used to transport rocks in a Skanska quarry site. While human operated machines do not require advanced monitoring, a fleet of autonomous machines must be monitored to catch potential critical situations in time. This monitoring of many moving vehicles from a control room utilizing, among others, the video feed from static cameras located at the quarry site is challenging and tiring for humans. There is a risk that critical situations are overseen, and operators react too late. It is interesting to investigate how a drone can support the human operator to identify critical situations in this context.

This thesis investigates how to implement additional monitoring features to improve overall safety on the autonomous site, where finding humans before startup is one of the tasks. During autonomous operation of the site, UAVs can support identifying unauthorized humans and report this to the site operator since this situation is potentially critical. One entry point is to examine how a UAV can be applied to support other monitoring features and support site operators with identifying critical situations. To provide additional monitoring features of using a UAV on site, appropriate data extraction algorithms need to be identified and evaluated. Robustness and reliability are essential characteristics such an algorithm shall provide to enable the application in safety critical use cases. Since autonomous vehicles, humans, and other vehicles are likely to move; the object detection must be capable of detecting and tracking movable objects, the camera feed of the UAV. This shall be reported to a 2D grid map to mark critical areas for the site operator. Providing proof of concept will be the main focus during the implementation part. Using videos from the UAV and support object detection, data extrusion, GPS coordinates, and other essential data, will be the main inputs to prove using this system as an additional safety layer for ensuring a safe operation of the fleet of autonomous vehicles.

Different *Convolutional neural network* (CNN), such as R-CNN, Fast R-CNN, Mesh R-CNN, RFCN, SSD, and different versions of *You only look once* (YOLO), are candidates for object detection, but only one is used to sustain the system with accurate neural network object detection, and proving the system with object detection neural network. Proof of using multiple object detection methods will be realized and overwhelm a higher accuracy than using only one method. You only look once (YOLO) is the used neural network within this thesis, with other object detection methods aside such as; moving detection and edge detection, encourage higher accuracy of finding an object on site. IR is discussed and tested on site but never implemented into the developed system.

The thesis is structured as follows. In section 2., the background of this thesis is described, including details on safety critical systems. The industrial case we study in this work is explained in section 3. In section 4., we discuss the related work relevant to this research. The research questions and applied research methods used throughout this thesis are described in section 5. In section 6., we provide details about the implementation and technical solutions. The application of our concept in the industrial case and the outcomes are presented in section 7. In section 8., we discuss our results and limitations of our solution. We conclude this thesis in section 9. and provide information for the future directions.

2. Background

This section presents the background for this thesis, which contains an introduction to safety-critical systems, object detection methods, and image detection algorithms.

2.1. Unmanned Areal Vehicles (UAV)

An UAV is an unmanned flying vehicle, where the main focus is to have non-humans on board, doing task either autonomously or radio-controlled. UAVs is being adopted more and more to the military since its support of doing a task without interfering physically on site. UAVs also gets used more and more since its usage in the private industries were taking pictures and flying has been a main area of usage. Races are being held between users, and hobby-photographers use UAV to take helicopter-like pictures. UAV is being called drone in many contexts outside of the area och technical industries, and engineering [6]. [7].

UAV can be controlled using radio control or can be run autonomously [8], where the ground role of a UAV is that is shall always be unmanned. UAVs are equipment that can have a different purpose and can be used in the military [9], or task such as object detection [10]. UAVs can also be used in assignments were making it easier for humans, such as spraying field to protect people from malaria [11]. This proves the multipurpose of UAVs. The information gathered is that the object detection system may be used on Volvo autonomous site since the multipurpose in different application areas.

2.2. Critical situations

Critical situations state a problem to solve, where a situation is accruing between vehicle - vehicle, human - vehicle, or animal - vehicle. A critical situation is defined as an accident that is about to happen, including an on-site object. Three levels of identifying different critical situations are further discussed in this thesis, such as

- Catastrophic: Loss of life, complete equipment loss
- Critical, Hazardous: Accident level injury and equipment damage
- Marginal, Major: Incident to minor accident damage
- Negligible, Minor: Damage probably less than accident or incident levels

All the defined scenarios of situations will not define words such as; Catastrophic, Critical, Hazardous, Moderate, Major, Negligible, or Minor. However, the situations will need the reader to understand these levels of situations in thinking about safety [12].

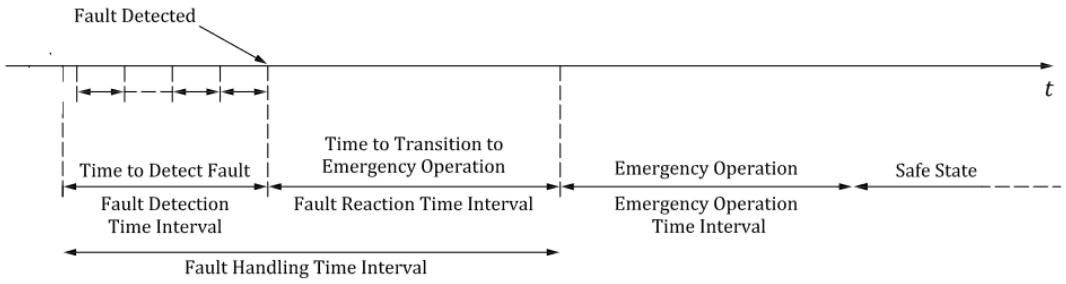


Figure 1: Fault reaction time figure gives the operator time to see a potentially critical situation, to press the emergency stop.

One localized critical situation probability is the human factor of the operator. The limit of having a human operator guarding all autonomous vehicles at once creates a risk of critical situations. In Figure 1, downloaded from [13], fault detection is being pointed, showing the time to detect a fault and the time to transition to emergency operation. “Fault handling time interval”, in Figure 1, defining the operator’s mental distance to find a critical situation and the reaction time to stop it. What if the operator is not focused on the *Closed Circuit Television* (CCTV)-screens, or just tired of being there all day? The detection of fault time will raise and a lack of physical reaction time to handle the site’s emergency operation. This human factor can lead to a critical situation based on missing out on information or the lack of focus, creating a risky environment for humans and machinery. Emergency operation and safe state will always be defined times, since after pressing the emergency button, the system shall stop immediately, leaving a safe state on site.

2.3. Machine Learning

When computers were developed, the goal was to make them think like humans, where machine learning is a substitute for that [14]. When creating machine learning, the purpose is to make computers learn by themselves without interfering [14]. An example of this is learning by doing, and where supervised learning is one way, and unsupervised learning is another, this will gain different time for the result [15]. Example supervised learning is a group and interpret data based only on the input data where a classification or regression is made. In contrast, unsupervised learning develops a predictive model based on both input and output data where clustering is the outcome [15]. Some other development in machine learning with learning by doing is; localization in space and mapping, sensor fusion and scene comprehension, navigation and movement planning and evaluation of a driver's state, and recognition of a driver's behavior patterns within the self-driving machine learning for cars [15]. Machine learning use the sampled data, the trained data, making a prediction or decision without being programmed to do such a thing. An example is; object detection, where the machine learning guesses multiple objects on a picture, only to print one of those because one label has a higher percent of predicament than the others; therefore, it gets printed.

2.4. Object detection using camera picture with UAV or CCTV

Using camera equipment such as CCTV or the applied camera on the UAV have a significant difference of implementation where CCTV-cameras are attached with a cable. In contrast, UAVs camera is sending data trough WiFi, or it is needed to manually extruding data from the SD card. This limit is air-time, where the UAV also runs on battery but is movable, and CCTV-camera runs in a fixed position and is limited cable and fixed position. Though the difference is minimal during the object detection phase because object detection only uses video inputs, it can be extruded from both UAV or CCTV. When the UAVs is in fixed-position in the air, hovering, the information has almost zero difference because the camera is still and fixed. Therefore, the object-detection can be applied on the Video from the CCTV-camera or the drone the same way. By extruding videos to use in the object-detection system. The camera is typically 720p to 1080p resolutions, where the resolution may differ depending on camera and usage, and in this thesis, the limitation is 720p24fps to decrease the usage of storage. UAVs equipped with a camera use gimbals to reduce vibration and can be movable. A work by N.Shijith, Prabaharan Poornachandran, V.G.Sujadevi, and Meher Madhu Dharmana, [16], gives us the difference between a CCTV and UAV, where they use CCTV to find flying UAVs, using CNN for detecting the presence of the UAVs.

Also, the main difference to use UAV in this thesis, and not CCTV even though it is possible, the UAV can be moved where to give much higher safety and security on a specific position on site.

2.5. Neural Network

A neural network (*Neural Network* (NN)) is based on inputs, sending into hidden layers to predict a result's outcome. Datasets are being used as a symbol of default interaction. Think about this as a" template" where the system uses the trained files and compare this to inputs. The inputs get analyzed in the network, and from there gives us an output. [17].

Region-Based Convolutional Neural Networks (*Region-Based Convolutional Neural Network, Object detection Neural Network* (R-CNN)) have an original goal of taking input from an image and produce a set of bounding boxes as output. This also shown the category the boxes contain, where a verification check has to be done on the different categories to find the correct object.[18]

2.5.1 Object detection

Finding an object has been a way to allow machines to see pictures and make assumptions about them. Object detection is a way of finding objects in a picture, where multiple methods and ways to find an object, such as NN object detection's, motion detection, edge-detection or just abnormally postulate based on pictures. Object detection is a division from computer vision, where the base of object detection is to find an interesting object for the defined case [19]. Placing boxes around objects to prove the finding is a way to confirm for the supervisor that the object detection works and easier to verify with a human-view. This will be a verification method for detecting object [20]. As in the article Understanding object detection in deep learning [20], they confirm that "Like other computer vision tasks, deep learning is the state of the art method to perform object detection", also confirmed in the report [21] by J.Redmon and A.Farhadi, where they state that YOLO is a state of the art method to perform object detection, and are a project in the deep learning area. Some of the most used representative models of deep learning are CNN [22]. A typical CNN has a hierarchical structure and is composed of several layers to learn representations of data with multiple levels of abstraction [22].

Before deep learning was introduced, slower layer abstraction was done, but since the algorithms have evolved and computer power has risen, deep learning is the fastest way to handle learning in object detection [23].

2.5.2 Datasets

Datasets can be used in multiple implementation methods, mostly used in a neural network to train and limit the neural network to a specific task. As mention before, think about this as a template, where an object detection framework has to do one assignment and not detect multiple objects outside of the scope. Kilic et al. are doing a Traffic Sign Detection with TensorFlow, using a traffic sign dataset where photographs in different traffic and weather conditions to create a traffic sign object detection, workable on all environment that the system shall be in use [24].

2.6. Masking

Masking can be applied as a variant of the filtering method. Information and accuracy can be gained to decrease unnecessary information without using other algorithms. Due to such heavy computational power need, a method to filter the information to gain higher accuracy and decrease the computer power-need is needed, such as Zhongmin et al. limiting inputs for YOLO by using mean shift [25]. Masking also supports using pictures and videos as an input, where filtering methods will be extruding unnecessary data for better results. Right now, an pretrained network finds objects that do not need to be found, such as toilets and boats. These classes are on the correct vehicles, such as TA15 is mostly defined as a toilet, boat, or truck, which can associate the correct information. The masking method can be used, especially when using IR-camera, because running in 2D view, persons will be seen in the IR, and masking can be done.

2.7. Programming language

The whole implementation is programmed in a specific language choice since the need for multiple languages is non-existent. This because of the choice of an available language with multiple packages to help the proof of concept implementation.

2.7.1 Python

The reason to choose Python was its multipurpose and integration with the system more quickly and effectively than other languages such as C, C++, or C-sharp. This, because of the imports of libraries and the more comfortable interface of mapping and graphical interface. It is also easier to install OpenCV, CUDA, and TensorFlow in Python, where the graphic card can accelerate object detection. Python is an open source language with integration that can support multiple libraries to import, that support the implementation with a proof of concept. [26] The Python version that is used is 3.8.5 because included in Ubuntu 20.04 *Operative system* (OS) [27].

2.7.2 Operative system (OS)

The purpose of using Ubuntu 20.04 as OS is based on making it easier to develop the system and include libraries. This also causes Python was included [27]. When tested and developed, *Windows Subsystem Linux* (WSL) was used to make it easier to use the system on Windows PCs [28].

2.8. Standards

Standards guide practitioners on developing safety-critical products under consideration of potential accidents or other mishaps that shall be avoided. ISO standards, such as ISO12100, are applied to the autonomous site and fulfill the customer's settle requirements, and the industrial standard of risk assessment and risk reduction [29, 30]. In this thesis, the standard will be discussed but never considered in the implementation of proof of concept. The implementation system is just a concept and never meant to be applied to a fully worked system.

3. Industrial Case

In the earth-moving machinery domain, there is a paradigm shift from single-human operated machines towards the usage of autonomous machines. In recent years, Volvo Construction Equipment has been working on autonomous haulers for transporting material in open-surface mines. The autonomous machines (called HX or TA15) have been applied and studied in real environment conditions at a quarry site [5].

A site operator monitors the HX fleet's activities and status from a control room. Static cameras located at different positions at the site provide visual feedback for the site operator.

Such a site can be complex with many vehicles in motion. Humans might not control the HX routes and check the video feed of all static cameras at the site.

- Critical traffic scenarios can be identified too late or not at all before an accident has happened.
- Humans might enter the (*Autonomous Operating Zone* (AOZ)) misinterpreting the capabilities of the autonomous vehicles and are not identified by the site operator due to camera blind spot or loss of focus from the operator.

There is a need for additional monitoring features to support the site operator with information for decision making or even directly interact with the system to solve critical situations.

3.1. TA15 - Autonomous machine

The TA15 [31] is an autonomous hauler designed and developed by Volvo Construction Equipment and Volvo Autonomous Solutions(*Volvo autonomous solutions* (VAS)). The machine is used to transport material in off-road environments such as open surface mines (Figure 2).

(a) TA15 side viewed

(b) TA15 as seen from birds eye view

Figure 2: The object detection of Volvo autonomous solutions machinery, TA15, using NN pretrained YOLOv4-weight file.

The TA15 is fitted with various sensors to, for example, detect objects in the path ahead. In Figure 3 the opportunities to control a TA15 (HX) is shown. The machines can receive their missions from the fleet control server by the operator, or the alternative is to control a single machine that can be directly controlled through a radio-hand-controller.

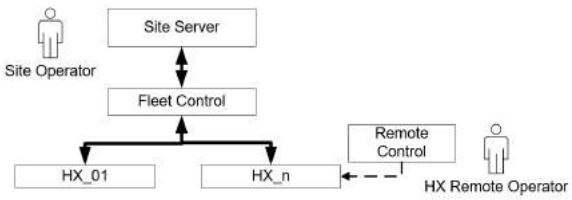


Figure 3: Control Structure of TA15 (HXn), giving the information that a TA15 is both autonomous within the fleet control or the site server and radio-controlled from the remote operator.

3.2. Application Scenario

Figure 4, a typical application scenario of a fleet of autonomous TA15 (HX). The quarry sites are being worked as following; Blasting rocks to receive the material, pre-crushing to limit big rocks' error rates from the first blasting, and pile building by wheel loader. This later gets loaded on to the wheel loader, transporting it to the secondary crusher, getting pre-crushed again to fulfill the rocks' requirements settled by the customer. [30]

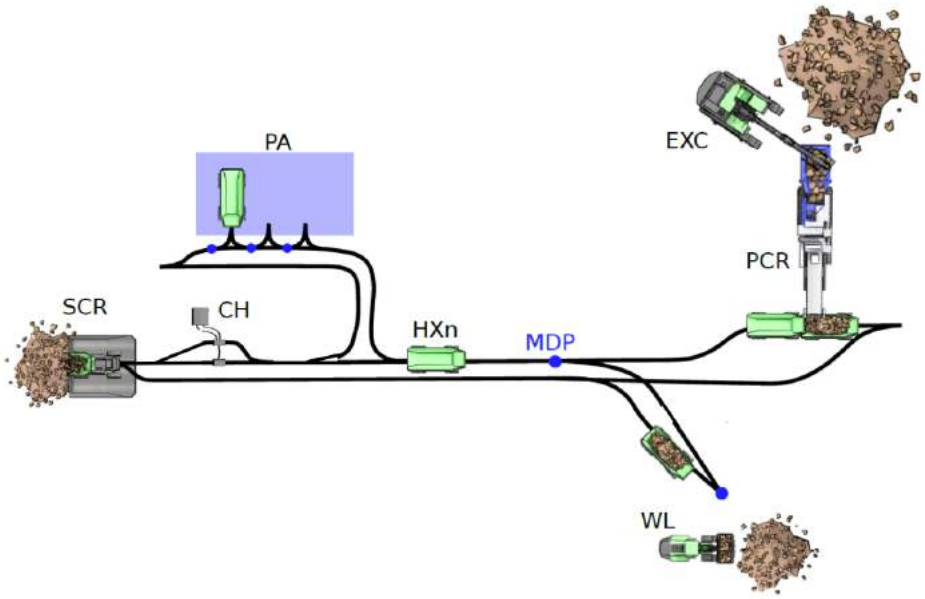


Figure 4: Quarry Site Automation - Electric Site Example, giving the information about how an autonomous site and machinery work in fleets, and charging abilities to sustain an autonomous site always running with assignments and operations.

In Figure 4, borrowed from [5], five HX-vehicles, defined as HXn (TA15), one Wheel loader (WL), and one excavator (EXC). On the left, SCR is placed (secondary crusher), and on the right, PCR (pre-crusher). On top, there is a Parking (PA) spot for HX-vehicles and charging (CH). How does this work then? The loading onto the HX-vehicles is being done by humans, driving the different vehicles, WL and EXC. The HXn drives around, transporting the material from A to B, charging on their way one at a time, depending on the need. The site operator decides how many HX is on the site, where the rest stands on the PA. PCR is loading the material into the HX itself, while the EXC loads into the PCR. On the WL-side, the WL has to load the material onto the HX itself.

Each mine may look different, where the automation system and the safety activities need to be adapted based on the needs. Autonomous sites need to operate without any human interacting on the site, so when doing morning startup, the TA15 gets driven autonomously into the sites, where they start the autonomous operations when the site is clear of objects. To know if the site is safe to start, the site operator needs to know whether the site is clear of objects or not. The site operator also can go around on-site to see if it is clear or not, though this is not sufficient.

Normal on-site operations mean that the autonomous vehicles are assigned missions executed without interruption by either humans, such as getting loaded with a rock by a human-driven wheel loader and transporting this to another position. During an unexpected event, such as an unauthorized human on site, the site operator can press a shutdown button and start a shutdown operation. This is starting an event of a shutdown and immediately stops the site, where all TA15 immobilizes, leaving a safe state and reducing the probability of a hazardous or catastrophic event.

3.3. Camera Surveillance

Static cameras positioned in suitable locations shall enable the monitoring fleet's activities from the operator's control room. The video feed from each camera is put on a separate screen in the control room, requiring the operator to watch the screens for possible critical situations continually. Although the operator supervises the fleet and events on-site, the general problem is to keep track of all moving machines and situations at once, as shown in an example in Figure 5.

Figure 5: Example of a CCTV operator, where the operator sits and watches all the CCTV screens at once to sustain the safety on site, using two different methods: overview map of the autonomous vehicle and camera surveillance on site.

The operator is required to identify a critical situation when watching the screens and, if necessary, to stop the fleet of autonomous vehicles by pressing an emergency button.

The site operator is required to react in time to ensure that the safe state, i.e., stops the autonomous vehicles, within a safe period of time. Figure 1, the fault reaction time of a safety-critical embedded system is shown as described in ISO 26262 [13]. Once the fault is detected, the system shall react in a specific time interval, i.e., the fault reaction time interval, to set the system into a safe state. The same mindset is applied here, where the site operator shall react at a specific time to set the system into a safe state.

3.4. Identifying hazards

To identify critical situations, hazard analysis is conducted. Each situation and function is analyzed to find hazardous events. Typically, a table is used to collect and document all hazards like shown in Table 1

ID	Event	Humans at Risk	Effect
H1	Start of Autonomous Operations and humans at site	Unauthorized Human	Fatal Injury
H2	Normal operation and humans at site	Unauthorized Human	Fatal Injury

Table 1: List of possible critical risks at the site, pre-defined at Volvo CE. The list includes finding humans on start-up and finding people while operating on site. The list is not complete and can be updated due to multiple risks. These risks will be simulated and discussed.

Table 1 exemplifies and simplifies how possible critical events can be identified. Hazard analysis methods such as Failure Mode and Effect Analysis (FMEA) [32] or the Preliminary Hazard Analysis (PHA) [33] can be applied to identify hazards in a structured way.

4. Related Work

In this section, we go through the related work of the thesis. Related work is based on similar work where methods are alike with those used in the thesis, and from that, proofs of the correct concept of the method. Research findings from other research groups doing similar works with other assumptions, where the focus of finding related works, prove that the methods of choice are competent to solve the research question and create a verifiable system. In the related work, we mention those other research and point out why it is not working in our context or point out their limitations. NN, object detection methods are in full development, where the question stands, what type of object has to be found and how high does the accuracy need to be to verify the system? How have other projects solved similar problems?

4.1. Object detection

Similar work, stated below, does not use the image detection system to define and provide operators with a dangerous situation and prevent critical situations. Works, such as Siyi et al. only looking for new ways of tracking objects, [10], never intended to apply the system into safety-critical-system, just giving new object tracking where other research has to implement to prove the use-of-concept. Other related object detection work and future updates can be real-time object detection, tested and optimized within different state-of-the-art surveys and implementations. Jaiswal et al. are proving the use of object detection in UAV real-time object detection, using *Graphical Process Unit* (GPU) power to gain speed and sustain accuracy [34]. In Jaiswal's paper, they try different algorithm methods such as feature detection, feature matching, image transformation, frame difference, morphological processing, and connected component labeling to support high accuracy of finding objects and detect their movement. The limitation of these two works is the lack of data usage after their run, where they provide information about the speed and accuracy of the system, not any implementation of real-life uses.

The different object detection system, such as R-CNN and Fast R-CNN, was used in the report from Siyi et al., and Jaiswal et al. Those object detection methods have been proven fast and accurate, but some object detection methods were faster. YOLO is a state-of-the-art, real-time object detection system [21]. Where different version of implementation exist, such as Tiny-YOLOv3, YOLOv3, YOLOv4, Tiny-YOLOv4, YOLOv5 and YOLO-PP (PaddlePaddle, Parallel Distributed Deep Learning) [35, 36]. The decided NN will be based on how easy it is to implement and how sustainable it is to prove the concept. Most of the researched image detection areas focused on UAV, YOLO-weight was general representations of fast objects detection method, with high accuracy. YOLO is proven extremely fast, more than 1000x faster than R-CNN, and 100x faster than Fast R-CNN, stated by its creators Redmon et al., giving proofs of its result [21]. Zhu et al. [6] point out that YOLOv3 combined with LZZECO, another way to find objects, got the highest score in their enormous dataset of VisDrone. VisDrone was created as the biggest dataset of UAVs, trained with the most used neural network of object detection by Zhu et al. They compared different object detection NN, proving multiple algorithms combined to create the dataset's best result. YOLO-run itself was average accurate comparing to other algorithms when comparing results of the algorithms on the VisDrone-DET dataset [6].

Supeshala, [36], compares the difference between all the versions of YOLO because new backbones layers have been updated, and stacks have been improved. More efficient YOLO-version, such as PP-YOLO are now used, aside from YOLOv4 and v5. A lot has happened since Redmon, [21], developed YOLO. From YOLOv3, the last update with Redmon as the developer to the newer versions; YOLOV4, V5, and -PP. Multiple developers have developed the later YOLO-version since Redmon left the YOLO project because the area of usage YOLO has been taken, where he was not expecting the outcome [36].

The focused research about design methods, where all different works supported improved object detection accuracy, was used to detect object detection and other object detection frameworks. Zhongmin et al. state that the proposed combined algorithm, including YOLO and Meanshift, another method of motion detection, raise accuracy and improves tracking precision [25]. Their proposed algorithm "suits the rapid movement, which has strong robustness and tracking efficiency", where that states that the support of multiple object detection methods raises accuracy and precision. YOLO never resulted in the best accuracy. The work of Zhu et al. had always result of YOLO being fast and accurate since its speed, but never the best [6]. Same result transpired for Siyi et al., [10], and Jaiswal et al. [34].

Multiple ways of using object detection in multiple areas are some main differences between all the works. Ryan et al. explain in their research [37] how they are applying an UAV for UAV-assisted search and rescue where the applicability from a UAV can be both fast, easy to control, and supportable. They also use UAV alongside with a helicopter "UAVs can assist in U.S. Coast Guard maritime search and rescue missions by flying in formation with a manned helicopter while using IR cameras to search the water for targets." This prove the statements of using a UAV in the multipurpose area and implementing applications. Nur et al. [4] have later proven the use of UAV in search and rescue applied with Determining

Position of Target Subjects combining with different ways to find an object (see Figure 2. in MSAR search patterns, ref.[4]).

YOLO was easy to find with pretrained datasets, and since its back-compatibility from YOLOv4 to YOLOv2, old YOLO-versions could be tested. V5 and -PP used other types of backbone that were not plug-and-play compatible with YOLOv2 and YOLOv4 [36].

4.2. Object of Interest

Wu et al. [38] are using YOLO comparing to Fast R-CNN to identify objects such as cars, buses, trucks, and pedestrians. Gathering UAV pictures, such as bird-eye view and angled view videos, has been a big interfering of using the right algorithm to the right network. The way the trained networks are learned has a prominent data placement of the different results from this thesis's output.

Vasavi et al. [39] highlights that counting vehicles like cars, truck and vans is useful for intelligent transportation. They are using the YOLO architecture combined with Faster R-CNN to count the number of vehicles with different spatial locations. The authors are using two pretrained datasets; Cars Overhead with Context (COWC) and Vehicle Detection in Aerial Imagery (VEDAI). The result gave them the proposed system with raised accuracy by 9 percent to detect smaller objects than existing works. COWC is a dataset trained to identify cars in bird-eye-views when VEDAI is a trained dataset for detecting boats, camping cars, cars, pickup, plane, tractor, truck, van, and others. [40, 41]

Rabe et al. use their approach of 6D vision, where they estimate the location and motion of pixels simultaneously, which enables the detection of moving objects on a pixel level. This method, combined with a Kalman filter [42], results from 2000 image points in 40-80ms. With the analyzed vectors, the result proves where the object is heading and solve the critical situation. The result was done with an old 3.2 GHz Pentium4 *Central Process Unit* (CPU), wherein this thesis we mostly used a core I7 5400U 2.1GHz, proving the lack of computer power and GPU may not limit the result. The analytical methods of the critical solution from Rabe et al. can be compared to motion detection in this thesis's implementation system. The motion detection applicant is an excellent way to find moving objects without knowing the object's type, but they lack the implementation and usability of their motion detection. The developed motion detection, used in the thesis, has a base of just using OpenCV library [43], making the system fast and supporting threshold changes for filtering out unnecessary information.[44]

4.3. Neural network frameworks

Redmon is mention that Darknet is an open source neural network framework developed in C and CUDA. The purpose of developing a CUDA-system is because the use of CUDA-cores from GPU's such as Nvidia makes object detection run faster. Darknet is stated as fast, easy to install, and supports both CPU and GPU computation. Darknet gives the support of using YOLO and also gives the opportunity of an open source neural network framework. [45]

The limitation with Darknet is the backbone of the new YOLO-version, where the new YOLO-version has been updated from Darknet to other backbones [36]. Darknet is nowadays not used within the new YOLO-versions. Darknet was used as a backbone phase, just as ResNet, DenseNet, and VGG. These backbones produce different levels, and the networks use different assessments of layers. Long et al. wrote in their report about the new YOLO-PP, using new backbone. The new blackbone is ResNet50, creating YOLO-PP non-back-compatible with the darknet [46].

4.4. OpenCV

OpenCV is a tool for the YOLO and other image-processing libraries used in the thesis. Multiple works use the OpenCV library [43], for example, Brdjanin et al. wrote in their paper, using OpenCV as a single Object Trackers. OTB-100 dataset using OPE and SRE combined with Precision and Success Plot succeed to find single object trackers [47]. With OpenCV support, Brdjanin et al. created image processing and libraries for their video and picture data calculations [47].

OpenCV is an Open Source Computer Vision Library with more than 2500 optimized algorithms and state-of-the-art computer vision with machine learning algorithms. It delivers easy access to all video extruded and algorithm support for object detection this thesis [43].

5. Method

The objective of this thesis is to investigate how to implement additional monitoring features to improve overall safety. One entry point is to investigate how a UAV can be applied to support additional monitoring features, and from there supporting the site operator about critical situations that are about to happen. The site operator will use this system as extra support for notifying critical situations and making decisions. Problem formulating leaving two research question, 5.1.2.

5.1. Proposed Structure

The research process shall include proof and solutions to the problem formulation. A literature study shows the appropriate ways to identify objects for this thesis, where the prestudy will determine the best way to go. Experiments and tests will be applied to extruded information from the codes and implementation method, providing solutions for the research questions.

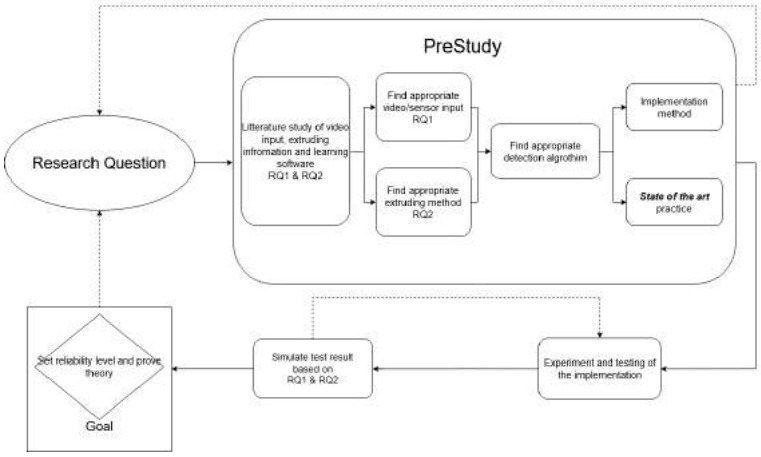


Figure 6: Flow of method from Research question and problem to functional implementation.

The simulations will prove the states of the system and reliability. Focused on the implementational, identifying if it can be applied to another system in the future will prove its value and support for the stakeholder. Feedback proves if the reliability needs more testing and more development or is safe to use in any case and implemented in other systems.

After simulation and implementation, the reliability level will be given, and the theory will be proven for both RQ1 and RQ2. Real-time inputs will be tested to prove the variability and multiple inputs to the system, providing a legit result. The already settled system from Volvo CE, for their autonomous site, will be in mind providing the operator correct and informative information without interfering with other vehicles disturbing the autonomous site.

5.1.1 Research Problem

Giving the site operator the highest command, where the system shall only support higher safety and not take over the safety on site entirely, the site operator's information needs to be informative, usable, and never misleading. The information shall give the operator information if a critical situation is about to happen and raise the safety on site. The system shall detect humans and vehicles on-site in multiples ways, including object detection, edge-detection, and motion detection within the vast area of usage.

From our study of the industrial case, we can identify the following open issues we aim to study in this thesis.

1. It is challenging to keep track of all vehicles through the camera surveillance system and watching all screens with the camera feeds.
2. The site operator is required to react in time to set the system into a safe state if required.

This thesis aims to support the site operator to identify critical scenarios using an independently operated UAV.

5.1.2 Research Questions

- Research Question 1 (RQ1): How can a UAV be used to detect humans and vehicles at such a site?
- Research Question 2 (RQ2): How to support the site operator in monitoring safety critical situations in an open surface mine with autonomous vehicles?

5.1.3 Research Methods

Problem statement Problem statement is not the same as the research question. Problem statements shape research questions, where statements are found through the autonomous site of Volvo CE. To solve these issues, mention before; *It is challenging to keep track of all vehicles through the camera surveillance system and watching all screens with the camera feeds.* and *The site operator is required to react in time to set the system into a safe state if required.* reasonable solutions need to be assembled to solve the issues and problems.

Literature Study Studying the fields of practice, literature, and identifying of already available methods of solving the object detection problem. Many studies use methods to find objects and combine different NN to gain accuracy and speed, but it comes to choose those works where the field is closest to the settled field of practice. Understanding object detection, we studied literature to find solutions to the open issue. We used the following databases for our search of relevant literature: IEEE, Springer Link, PapersWithCode, arXiv, and books. Even other sources were considered where additional information was provided.

Design Research We applied a design research method for developing the object detection solution presented as a result of our work. The design research method requires is a type of research method, where a product is iterative developed with continuous validation and feedback loops. The process and tools developed throughout this research where continuously improved and refined based on real application data from the industrial site. Specifically, we applied Python as programming language to enable a fast prototyping and refinement, since the purpose of this research was a proof of concept. Based on our literature study, many researchers utilize Python as well. Other works use other programming languages like C, which requires a more stringent development process and more development efforts, which was out of scope for this research.

Validation As we pointed out before, the results have been continuously validated in real-life applications at the Volvo CE test track. In comparison to applying simulations, the application of the UAV in real environments enabled us to validate and refine our results continuously. An important question on how the system interferes in real-life environments was part of the validation activities. In the shortness of time, we were not able to test different weather conditions, which might have an impact on the results.

5.2. Evaluation

With the information gathered from the prestudy and other similar implementations in previous projects, filtering and optimization will be applied, not focusing on speed but accuracy. Proofs will be provided from using a UAV as an extra safety layer on an autonomous site while object detection, edge detection, and motion detection combined will be a fast and stable method for image detection, determine the research-questions and problems.

Additional monitoring features of a site appropriate data extraction need to be identified and evaluated with collected information from the UAV. Reliability is an essential characteristic that such a network shall provide to enable the application in safety-critical use cases. To provide a structured evaluation of the potential network, a need to identify objects and identify critical situations from the data, provided from the UAV and the code. Comparing the different outputs from the code, such as gridmap and object detection, real-life viewpoint on Volvo's autonomous site will be necessary. Since no focus on introducing the implementation into Volvo's autonomous system, the Evaluation needs to be done by hand and proving that the system is accurate enough to use as an extra safety layer for site operators.

5.3. Limitations

The limitations are based on the research questions, where the main focus will be to uncover the most reliable algorithm for this specific task and implement it. Also, leaving behind a thesis available for future updates and implementation.

All the different object detection methods and applications cannot be described since there are so many and no need to implement comparison. The choice of object detection application will be due to its support for easy implementation and development. For this thesis, speed combined with accuracy and easy-to-implement is the main area to looking into when choosing an algorithm. YOLO has been shown as a fast algorithm, with high accuracy and easy-to-install. Therefore the choice of algorithm to implement was YOLO.

RQ2 will be based on an pretrained network to save time and create a workable and provable implementation. RQ2 will be based on the data extruded by the settled UAV (DJI Mavic Air). If there are other essential data needed to sustain accuracy and higher dependability, it will be researched and discussed but not implemented. The focus will be to prove the proof of concept and not create a fully working implementation.

This thesis's primary goal is to investigate how a UAV can support the Site Operator by monitoring the site. This work uses existing tools like the OpenCV library, [43], to identify humans and machines in motion at any site of choice. The target is to identify critical stations and support the site operator in decision making. The task is to evaluate how a UAV could support the Site Operator to ensure higher safety at the site. This focus will lead to other on-site-notifications such as humans on-site, other vehicles on-site, or unplanned interaction, such as "unintended" human interaction on site.

6. Implementation

This section describes our implementation's technical details, where we have collected information, configure essential software, and applied a neural network to fulfill the thesis's requirements. To thoroughly test the implementation, creating a simulation was necessary, such as OpenCV image viewer where a video configured with YOLO can prove that the problem formulation is solved, verified by the operator and user. By this, verification is done, and where to see if necessary changes must be implemented in the system, or critical states findings need to be adjusted.

6.1. Process of system, basic overview

The dedicated UAV to the project is a DJI Mavic Air, even though it is closed source code on the DJI's product. There was a consideration between DJI Mavic Air and an open source drone called Open Drone, a project from Malardalens University. We compared pros and cons, and a decision to use the DJI drone was made [48, 49]. The decision to use a UAV from DJI instead led to a faster workflow for the data collection where proof of concept was in focus, considered is was closed source. The Open Drone project was an open source project developed at Malardalens University, using flight control computer from open source developer pixhawk [50]. Open Drone would have been an excellent UAV to use in this thesis, except it did not have any camera included, and no support was implemented for streaming or stability of cameras, and this would take unnecessary time to implement for this thesis [48].

The implementation can be used in defined task and assignment, where the YOLO-weight file is trained on those specific object that needs to be detected. For example, in this thesis, the dataset used is an pretrained network, with everyday objects such as vehicles, groceries, humans, and more, where the focus will not be to create or train datasets. The usage of an pretrained dataset will limit the result to the pretrained and selected dataset. All tested datasets are standard without any editing except filtering out unnecessary data, such as detecting groceries. On-site, trucks need to be detected, wheel-loaders, humans, TA15, and maybe animals. Proof of using a fast and accurate algorithm will be necessary because computer power usage will rise when more objects get included.

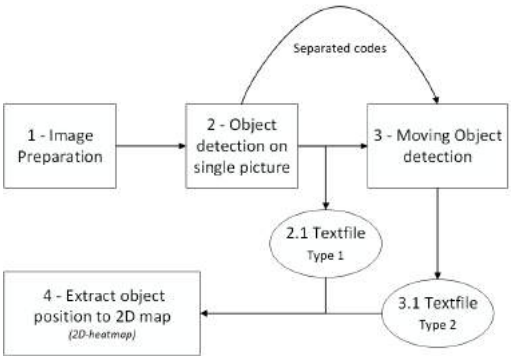


Figure 7: Basic structure of the implementation.

In Figure 7 shows the basic overview of the system, and the different steps define the different chases.

Step 1 Image preparation is being done, where extruding a picture from a frame of the video.

Step 2 The picture is interested in the object detection, where the object detection is using YOLO and edge-detection to find an object on the picture.

Step 3 The alternative of using motion detection, attached in a different code, is shown. Motion detection is an extra redundancy interfering with the support of gaining accuracy for object detection from step 2. From both step 2 and step 3, a textfile is created where the files are stored as a buffer for the next step. There is one textfile for each object detection and motion detection.

Step 4 The last step of the implementation where the created textfiles, discussed in step 3, are used to create a grid-map, creating coordinates and a map with *Global Positioning System* (GPS)-interface, pointing where the detected object is from the perspective of the UAV. Creating coordinates is possible since the saved data from previous steps store distance information, relative to the UAV and uses the UAV-coordinates to calculate the coordinates for the founded object.

Detail information off all the steps is being shown in Figure 10, 13, 14 and 15.

To prove the implementation's easy to install and use, basic process-visibility needs to be hand over where the user, or future developer, could see how the implementation behaves and how to install the procedure. Every box in Figure 7 will be discussed further below and go into every subject's depth, showing and defining every implementation process's usage. *Textfile type 1* is the output from the object detection version, where *Textfile type 2* is from the moving detection. Type 2 needs data from Type 1 to solve distance calculation, such as camera angle and height of UAV. All steps use library OpenCV to handle video inputs, frames extrusion, and picture creation. YOLO is also applicable to the OpenCV library, making it easier to handle inputs such as videos [43].

6.2. Calculation basis for gridmap

By going over objects, such as people and vehicles, a definition of the distance between the two objects was made. The coordinates from the UAV was used, and the object position was made. This limits the UAV's use of bird-eye-view and finds the object straight under in the center of the camera, extruding the coordinates. For this to work on multiple vehicles, the UAV then have to fly to the next object to verify its position. This takes extra time comparing to the Pythagorean theorem-method, 9. More errors can occur over the other method since more objects need to be detected, and higher accuracy can not be granted. However, this method is more distance-accurate since the drone could use its coordinates over the vehicle and other vehicle coordinates in future updates. This also raises the accuracy of positioning the vehicles when combing all methods, including the Pythagorean theorem-method and the future support of triangulating with the site vehicles, if that future update would be fact.

Coordinates from the UAV are being extruded from flight-data-recorder in DJI GO 4 application (IOS usage) as a CSV-file, reading coordinates every 10HZ. Different events such as "staying still for 5s" sets a timestamp in the CSV-file, and from there, data and coordinates can be extruded.

Figure 8: Gridmap where blue square is the UAV and the reds squares are the detected objects. Straight upward is the North, while left is west, right east and down south. The compass angle from the UAV will be used to referee where the object are referring to the UAV. This is being static in the gridmap code, appendix C

Using image detection to find objects, where accuracy can differ depending on different weight- files, the founded objects' usage gets analyzed with a coordinate system on the image viewer from the script. Therefore, the data will be extruded where the vehicles and humans are placed on the picture, and from there, calculate the distance between the objects. By defining the length of the pixel in the camera distance between object, offset position has been found, and by using height, angle of the camera, and GPS-coordinates data, new information is being calculated from the available information.

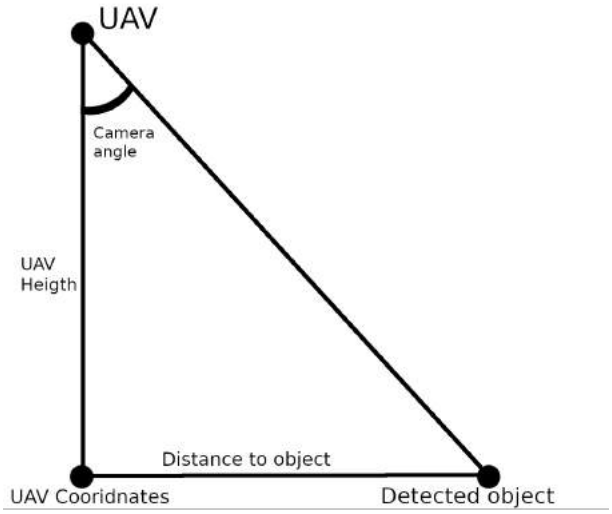


Figure 9: Usage of pixel and height, giving information about how to use the camera angle, UAV height, UAV coordinates and the detected object. The only missing variables then is the distance to object, being calculable.

Using the Pythagorean theorem, pixels, and height, we can define an object's length and calculate the distance between the object and UAV. This gives objects distance in the object detection output and could define if a situation could be dangerous or not. It is easier to find objects during an angled camera due to its more significant RGB inputs from the object, and since most of the training in the YOLO-datasets is trained on objects standing aside. The accuracy is better with an angled camera. Therefore that is the way to record videos for tests. From there, multiple objects can be found, such as humans and vehicles, print them on the heat map, and from there, see to define if it is a critical situation or not.

6.3. Preparation of image

Using the film from the drone and OpenCV library, a simple one-frame from the video was extruded and create a calculable picture to run in YOLO and Edge detection.

Extruding the data from the drone, such as camera angle and CSV-file from the DJI go application [51], where that information is needed later to do the calculation. However, the extrusion of video, camera angle, and CSV-file is all needed for the implementation.

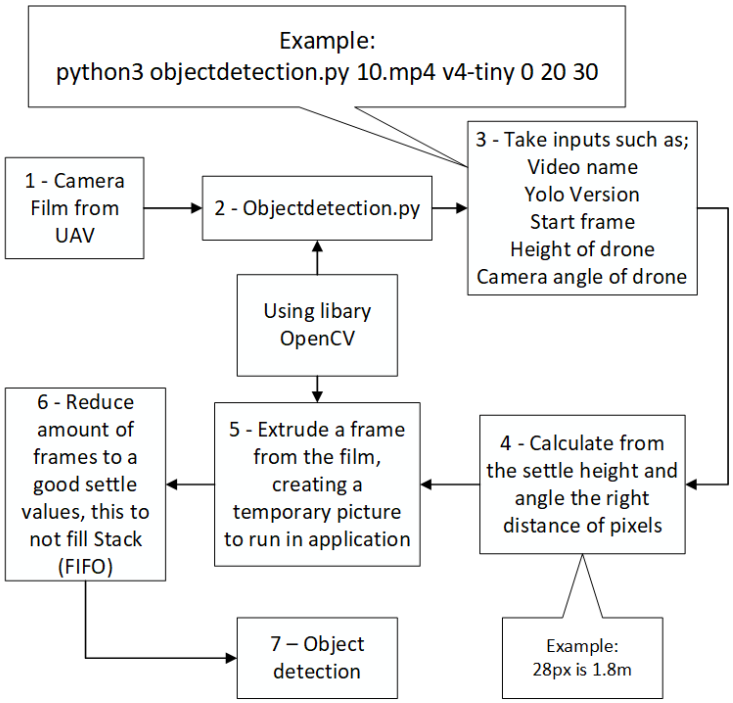


Figure 10: Overview of *objectdetection.py-code, appendix A* with preparation-of-image focus.

In figure 10,

Step 1 Manual user implementation, such as defying what videos are being extruded of its frames, creating a picture to use in a run. The video is being defined in the command; see example in Figure 10.

Step 2 User define what frame to start on, what version of object detection dataset such as YOLOv2-tiny to YOLOv4 [52]. The user gives the system the input of the UAV height, seeing on the DJI-go application, and what angle the camera is in, also viewed in the DJI-go application. All this shall include in the command when running the application.

Step 3 Just showing the data needed to run the application, and when the command has been run, and all data is included, the application starts. If not all data is defined, it warns that information is missing and do not start.

Step 4 Estimates distance using the height of the drone, and a point in the middle of the picture, estimating how many pixels is a specified distance, such as 28px 1.8m.

Step 5 The extruded picture is being applied in the loop, where every new run, a new picture is replacing the old in the loop process. Pictures are being extruded from the frame from the user-decided video.

Step 6 Extrudes a picture from a specified frame, once every ten frames. Step 6 can be edited by the operator or user, depending on the computer's compute-power. Having a more powerful computer, or GPU with CUDA-support [53], the system can be run in higher *Frames per second* (FPS) and use a higher amount of frames raising the accuracy and the probability of finding objects.

Step 7 Gives the showcase of suing the next step of the implementation, object detection.

6.4. Data inputs

This section gives what inputs need to be achieved by the user, and what data the system requires to operate, also, how the mathematical aspect of the calculations has been created.

6.4.1 Camera angle

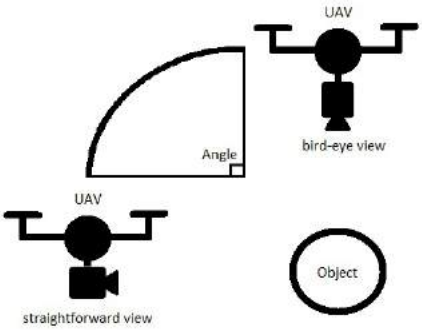


Figure 11: A visual figure about how the camera angle works for the UAV. bird-eye view is zero degree angle, while straightforward view is going to 90 degrees angle.

The camera's defined angle, inserted by the user like the example in 10, for the object detection to find where the object is on the map, see Figure 15 The needed information is applied to be calculated through a equation of distance and coordinates, equation 1 & equation 2 The information gets calibrated in the object detection file and stored in a new text file created by the object detection. The previous data that was introduced, angle and height can be used to determent where the object is from the perspective of the UAV position .

6.4.2 Equations

Equation in map.py, appendix C, gridmap equation for calculation distance to object

$$
\begin{gathered}
d = distanceToObject \\
R = radiusOfEarth = 6378.1km \\
\\
(AngleOfUavToNorth + angleOfObjectFromOffset) * \pi/180 = Bearing \\
\\
arcsin(sin(UAVLat) * cos(d/r) + cos(UAVLat) * sin(d/r) * cos(bearing) = Latitude \\
\\
UAVLong + arctan(\\
sin(bearing) * sin(d/R) * cos(UAVLat), \\
cos(d/R) - sin(UAVLat) * sin(Latitude)) = Longitude
\end{gathered}
\tag{1}
$$

Equation 1 is inspired by Veness function to calculate longitude and latitude coordinates using multiple inputs, [54]. The function variables are changed to fit the system, and the variables are defined in appendix C. The angles that are multiplied by pi divided by 180 is to make degree into radians.

The equation is applied in the map-plot script, of *map.py, appendix C*, Longitud later converted to degrees to create decimal-degree-coordinates. Distance to object is calculated as in equation 2. CameraAngle is defined as angle zero degrees is right under the UAV was ninety degrees goes to a high distance measurement that is excluded by the system since the value is not accurate to use. *Recommended usage is zero to seventy degrees.* To verify the equation as functional, real life measurement was compared to the equations.

Equation in objectdetection.py, appendix A, variation depending on the angle of the camera

$$PreEstimatedDistane * sin(CameraAngle * \pi/180) = EstimatedDistance \quad (2)$$

Equation 2 is made to assure the accuracy of the distance to the object from the position of the UAV, only using simple inputs. This is done since the pre-estimated distance may vary, deepening the camera's position and angle. The pre-estimated-distance is only to make the system pre-define a distance, where later is using the object-detection to precise the distance. This is being made since the pre-estimation is using pixels of the frame, for example, 1920x1080, were then using the angle and the height of the UAV, the center in the frame should be in at a specific distance from the UAV, excluding the field of vision for the camera lens. Object detection detects an object, where the object is a human. Humans are around heights of 1.8m, giving or taking 20cm, leaving a way to increase accuracy. This is a way to make more accurate information and remove error-rates. That calibration is using a recursive loop.

6.4.3 Angle to object

The angle to object is needed to define the distance to the object and the angle it is estimated to be at from the perspective of the UAV. This, combined with the bearing, is needed to make position on the 2D-map.

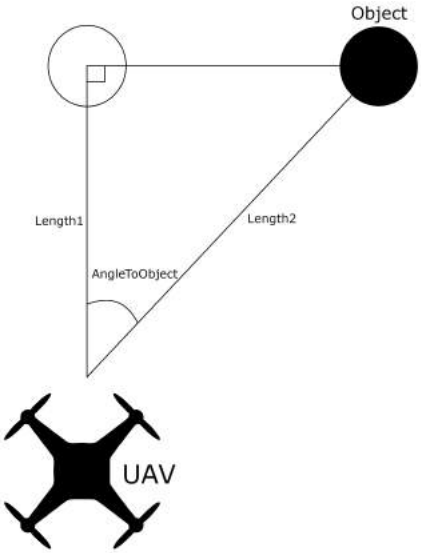


Figure 12: 2D-view how to measurement the angle to the object, UAV perspective.

A distance is being measured, were finding where the object is and what angle the object is related to the position of the UAV, detailed pictured in Figure 12. The white circle is the virtual object, defined in the code, where giving the object a distance if it was in the right in front of the UAV. Later, by using the angle to object, creating a 2D view of where the object refers to the UAV. When the distance from UAV has been estimated, information is used in 2, and stored in a textfile. The last step is being used in 1, where the last calculation generates coordinates from the stored information.

6.5. Object detection

The python-script of object detection, using user-decided YOLO dataset [52], working in the way of applying the drone's values such as; video-name, YOLO-weight file version, start frame, the height of the UAV, and camera angle to settle where the object is comparing to the drone. The run is not just using YOLO object detection. The object detection is combined with edge detection for the bird-eye view object detection, using the OpenCV image library [43]. Using YOLO on the picture extruded from the decided video, the image detection uses pretrained datasets and can determent different classes to sustain the system with information. From the first YOLO-run, values have been gained, such as position on the object on the frame and from there, calculate where the object is. Length1 and Length2 are being estimated, 12, and from that use, AngleToObject defines where the object is, more precisely.

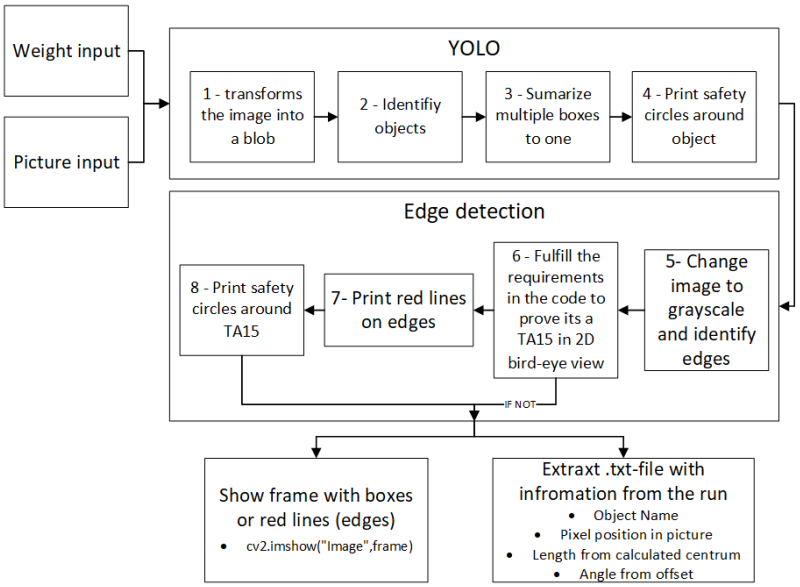


Figure 13: Overview of object detection with YOLO and Edge detection, and how the object detection frameworks proceed with the image extruded from videos or streams.

In Figure 13, we describe the object detection part of the python code *Objectdetection.py, appendix A* where two blocks divide the neural network and edge detection. Firstly, user-decided video is being introduced to prepare the picture to be used in the loop, see Figure 10. A YOLO-weight file is also being introduced by the user, depending on what type of pretrained YOLO-weight file that wants to be used in the system [52].

Step 1 In the YOLO part of the code, transformation of an image to a blob is being done [55].

Step 2 Identify an object, the pictures are being divided into multiple boxes, where identification of an object is being made with the weight file's support. If the weight file is big and trained in a higher amount of datasets, the system will take a longer time to identify an object but have better accuracy on the identified object, while smaller weight files such as YOLO4-tiny, is faster but less accurate. The boxes mentioned before dividing the picture into another with multiple objects detected.

Step 3 The system summarizes the boxes into one where only printing the object with the highest confidence. For example, it is unnecessary to print an identified *broccoli* with an accuracy of 0.05, 5 percent, while the system has identified a person with 0.95 accuracies, 95 percent? No, of course not.

Step 4 Safety circle is printed around the squares to limit the area around an object and identify if a person or object comes too close, creating a critical situation.

All these steps can be referring to Holzer YOLO object detection installation, and description [55].

6.6. Motion detection

Other implementation will be necessary to gain accuracy and support algorithms, such as moving detection when the UAV is in a fixed position to support YOLO algorithm. This method will not need much computing power to prove if a vehicle or person is moving or not [56]. The result will show if an object is moving, not what type of object. The decided object detection algorithm will show what object is moving, while the moving detection will store in a buffer from the algorithm, YOLO object detection in the leading network of this thesis, and YOLO is the only object detection in this thesis that can provide information on what type of object is moving.

Motion detection is an another in-house developed code, based on Rosebrock "Basic motion detection and tracking with Python and OpenCV" [56], aside from the previous object detection and edge-detection script. The application is as an extra layer of safety to sustain the entire object detection system, redundant. If following an object, such as a TA15, motion detection can see and stating that a "moving object within the same area of previous coordinates from YOLO is found". Therefore, the object has to be the same for some detection in the frames to determine that the object has to be that specific object detected object, specified from the YOLO and edge detection script. However, an increase of the system's accuracy is applied with motion detection, and non-defined objects can be found even if the datasets are not trained to other entities. If YOLO object detection have not found any object within the dataset, the motion detection will show a moving object where the operator can decide. If the UAV is moving too much, motion detection will find the frame moving and leave warnings on everything.

Motion detection creates its text-file since motion detection has no calibration-tools of its own to estimate distance accurately. Motion detection only supports finding a moving object and where they seem to be, compared to the UAV and not more. Data is being printed into the gridmap as a simple 2D-viewpoint.

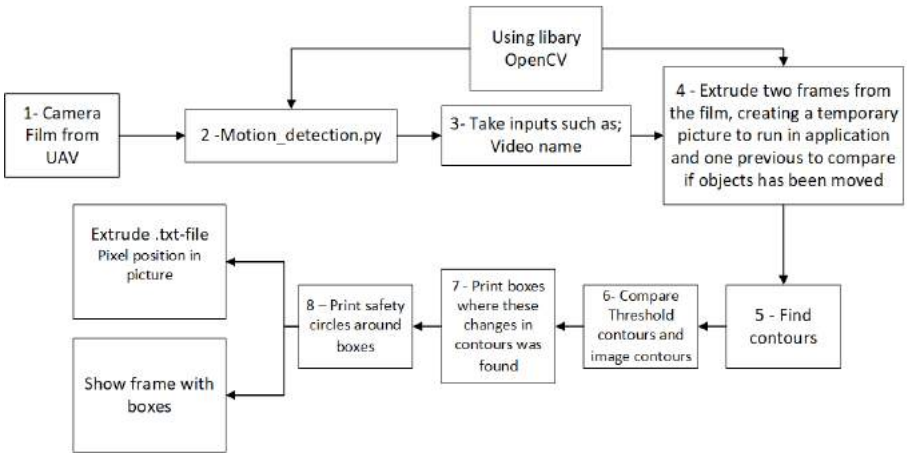


Figure 14: Basic overview of motion detection.

In Figure 14 we have multiple steps such as previous object detection, Figure 13.

Step 1 - 3 In step 1, 2, and 3, video is being decided by the user in a command, example; *motiondetection.py -v videoExample.mp4*, appendix C, where running the motion detection python code, openCV library [43] takes two captions of frames in the video, in different states. Frame 1 is being the default, while frame 2 is the frame to compare to frame 1. Frame 2 is the frame after frame 1, therefore seeing if something has changed or not within those two frames.

Step 4 Provides information by having two frames to compare to. After the code loops, frame one is compared to the last frame, frame 2. Therefore we always see the changes without losing the previous detection or rerun the code for every video frame.

Step 5 Contours are being found, creating a gray-scale of the frame, where using OpenCV library and findCountours function

Step 6 FindCountours is comparing the contours on both frames, and the threshold can be changed and filtering out unnecessary contours or pixel changes in the frames. If directly using the RBG-frames, original frames comparison is made on everything, and that would be leaving with everything moving and create worthless motion detection since we want to see objects, not threes or ground.

Step 7 - 8 Prints boxes on the pixels-coordinates where the changes have to be localized printing squares, and print safety circles in step 8 around the squares where our object was localized. This is being shown in a frame-show, were giving the operator or user visibility of the system and see where the object is moving. A textfile is being extruded, and from there could be used in the gridmap system, see Figure 15.

The pros and cons of motion detection are that; there is no way to see what object is moving or have an accuracy of knowing the object's size. Motion is just a supporting application, giving the operator information where the objects are moving and information on that there is something moving. Using information from the previous object detection run, estimating what type of object is moving can be applied in the gridmap system, giving the operator more understating of the situation. Motion detection is a fast way to find an object on site, even though it lacks information. Object detection from the previous code must be used to support the gridmap system's full use since no measurements such as distance to object are being done in the motion detection code also.

6.7. Object positioning and position-mapping

The grid-map uses the data from the textfiles to calculate where the objects are. A confirmation can be made proving where objects are and determine where to place them on the grid map. The grid map applies to the rest of the system, and from the grid map, critical situations can be proven and align with future implementation in map-attachment for path-planning. The grid map supports a GPS-coordinate system, so any map or system with GPS-coordinates can use the same data.

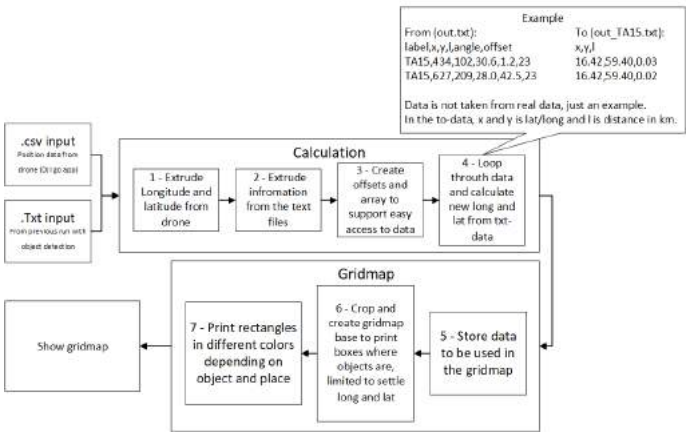


Figure 15: Basic overview of how the gridmap system works.

In the Figure of gridmap, 15, two inputs are the main inputs for the gridmap-system. The drone's .csv input, extruded from the DJI-go application, where all the flight data logs are being stored and used by the gridmap, and the .txt-file extruded from object detection or motion detection.

Step 1 The system extruded the longitude and latitude from the UAV .csv-file where that information is being stored to be used to do calculations, see equation in 1

Step 2 Extrudes information from the textfiles and stores them to later be used to see where the objects are compared to the drone's coordinates.

Step 3 Leaves the previous step data in a buffer, were not to hurt the primary (original) data and used the data in another format since edit.

Step 4 Data has already been stored in files, dividing the different objects into different text files. The code loop through the .csv- and .txt-data, reading and buffer all the data, *see example in 15* After dividing, data is being stored.

Step 5 The data that is divided into different textfiles, such as *person.txt* or *TA15.txt* leaves the gridmap to create different arrays and differ them with a color-schedule depending on the object when printing on the 2D-gridmap.

Step 6 Crop and creating the gridmap layer is being done, based on the coordinates defined as an offset in the code. A map lies in the background, and different longitude and latitude coordinates can be defined as the x- and y-axis.

Step 7 Print rectangles in a different color depending on the object, and place them where they have been estimated to be by the equation of distance estimation, 2 The last step is printing the gridmap to a graphical interface for the operator to see, using pandas and matplotlib [57, 58].

6.8. Application

How is the application working? The detailed version of every step, such as creating data and sustain site operator with information as described above and how data is used. Another way to go is by applying the application to the developer, user, or locality. How to use the system and the interfering purpose of the application for the user or operator.

6.8.1 Preparation of Image

Preparation of image, 10, is being done as described above, but the application is being as the user, or the site operator gives the code inputs on what video to use, what frame to start on, what is the height of the UAV in the video, and what is the angle of the camera attached to the UAV. The system itself only used libraries and installations files downloadable for Ubuntu OS, where multiple video formats are being applied to the application. As earlier described, the application uses multiple python-codes to sustain, run, and give the operator a graphical interface to interfere with.

Since the image's preparation part extrudes a picture from a video frame, video-stream can also be proved possible, pointing to support of real-time-implementation. This thought limits the real-time GPS-coordinate data from the UAV through its bounded support and restricted data. DJI software development toolkit existed, though this was no time to study during this thesis. Videos have multiple frames per second, around 24 to be exact, during this thesis, leaving an enormous amount of frames in one video. One frame can be extruded as a picture, freezing the frame and extruding a picture to run the object detection. Of course, that loses the direct consideration in the gridmap system with object detection support, considering the video needs to be run all at once. However, as a warning procedure, the operator can see and understand if a critical situation occurred or not directly in the object detection and the gridmap system. A real-time test concluded a delay with ten seconds in such a real-time-system, with DJI Mavic Air and Nginx-server [59].

Using the function of cv2.dnn.blobFromImage, were taking a blob from an image and using mean subtraction to help the YOLOCNN not to illuminate changes in the input image. Mean subtraction is a technique to aid the CNN. Computing the average pixel across the image, and in the training set, looking for colors such as blue, green, and red is needed the compare it to the datasets and trained networks. The comparison works as follows; the detected objects from the dataset compare if the system is confident to tell if the object is a dog or a horse. Maybe the comparison has an accuracy of 0.4, 40 percent for a dog and 0.3, 30 percent, for a horse. The object will then be printed as a dog since that is more accurate than the car. The CNN is the main base of detection objects using datasets and images, where there are five types of R-CNN: R-CNN, Fast R-CNN, Faster R-CNN, Mask R-CNN, and Mesh R-CNN. The first three CNN versions of R-CNN focused on object detection, and the other focus either on instance segmentation or add a 3D mesh to a 2D picture. [18, 60, 61]

6.8.2 Object Detection

The object detection will support the UAV while in fly-mode and applicable if it is in a fixed position or moving around. The fixed position also uses moving detection to gain accuracy, but the system will apply to any drone video and live video stream. Live-stream has been tested, though due to its delay, it was put asides since the thesis will not focus on the live-stream application, it is possible. The application shall apply to any surveillance video from a UAV and support easy testing and providing grid-map application.

6.8.3 Situation

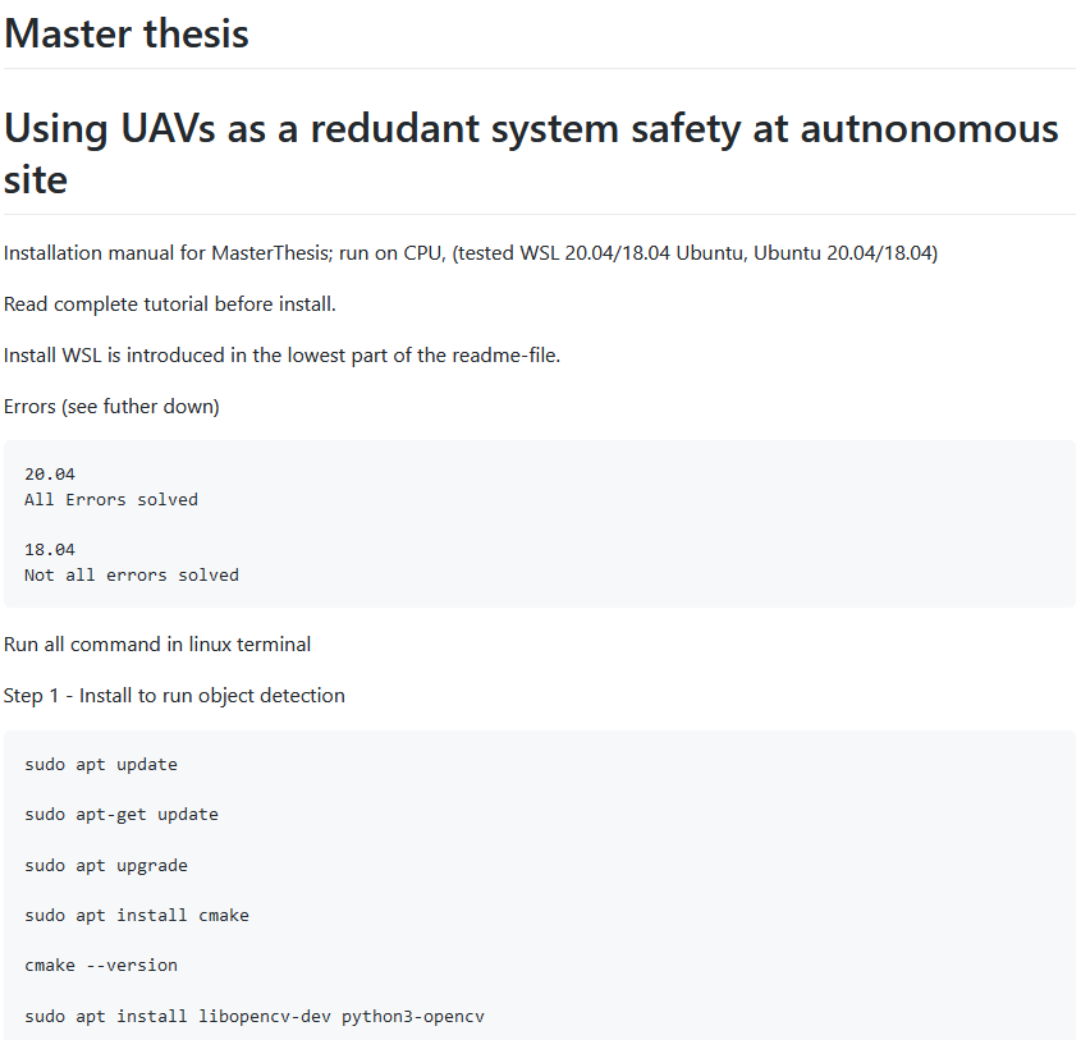
Situations being developed such as circles and squares colliding creates critical situations, warning the operator. The situations are also printed onto the gridmap, where the operator can see if a situation occurs or not. All objects on site can be seen, and the operator can decide if a situation is about to occur. The codes themselves are not trained in any specific task, but can, if an object comes to a close, warn about a possible collision. The implementation extrudes distance, action, and pin-points position for the operator to see.

Figure 16: Person and TA15 *(Truck)*, predicted situation, person in blind-spot, Weight:YOLOv4.

Figure nr 16 shows a situation that will be analyzed further-on during the thesis, where the person could be unseen by the TA15. The person could come from the blind spots, where the TA15 could not predict a situation until the person is being shown on the machine sensors.

6.8.4 Installation

The installation manual[1] is tested and verified to work on PC with Ubuntu (18.04/20.04) and Windows 10-computers with support of WSL, tested on six different computers. Therefore, the system can be run on multiple systems, with different inputs from video to live-streams (Real-time-implementation). However, the real-time-implementation is functional but never used because the DJI Mavic UAV delayed the stream for ten seconds.



Master thesis

Using UAVs as a redudant system safety at autnonomous site

Installation manual for MasterThesis; run on CPU, (tested WSL 20.04/18.04 Ubuntu, Ubuntu 20.04/18.04)

Read complete tutorial before install.

Install WSL is introduced in the lowest part of the readme-file.

Errors (see futher down)

```
20.04
All Errors solved

18.04
Not all errors solved
```

Run all command in linux terminal

Step 1 - Install to run object detection

```
sudo apt update

sudo apt-get update

sudo apt upgrade

sudo apt install cmake

cmake --version

sudo apt install libopencv-dev python3-opencv
```

Figure 17: Installation manual published GitHub

[1] GitHub installation manual with implementation https://github.com/Theralley/MasterThesis/releases.

7. Results

The result section will connect how the result was achieved and where it has been taken the research. With accuracy and system design, this thesis result has given solutions and significant availability of a UAV as extra safety on sites, or air surveillance in general. The result provides proofs and multiple methods of solving the problem statements. Critical situations that have been observed have been exposed and solved, marked on the site operators interface, making it easy to follow and non-misleading. With this result, the operator can sustain higher safety on site, supporting the operator with information and accessible object detection methods. The research surrounding this subject has raised using object detection and placement of the founded object. Giving object placements on maps has not been done before, giving a new field to study. The result may not reach a hundred percent accuracy, but the given result provides solutions to the problem statement, achieving this thesis's goal.

7.1. Design review

The design provides solutions to the research problem by combining the grid-map-system and multi-usage of object detection. By comparing grid-map to reality, the needed proves that its design fulfills the established research question and prove that there is a possibility to use a UAV as extras-safety-layer and the availability to be redundant-system for the site operator. Grid-map also gives the operator critical areas of the site. To inform the site operator about the critical situation now has to be evaluated, such as being on-site. Known, from testing, a small difference is existing between the result and real-life. The object detection is directly viewed in a graphical image, with warnings' availability, where the operator sees what is to happen on site. In the 2D grid-map-system, one can notice some error-rates, as seen in the appendix C, compared to safety circles figure 20. This error rate was decreased since the big black outline square in the grid-map system makes the smaller squares, the object, big enough to cover the area. The result only applies if the operator or user inserts all the inputs correctly. The application itself can warn directly, but applying the system to the grid-map gives the site operator easy-to-view methods of seeing if a critical situation is about to occur and where a situation can occur. The critical situation can then be limited to those places because the operator can either reschedule the vehicles' route or limit vehicle speed.

7.2. First version session

The system's direct implementation showcase was better than expected due to its easy-implementation of YOLO, edge-detection for 2D view, and moving detection. Later on, to prove the research question of the thesis, the grid-map was created. This ended in a new way of thinking, combing the object detection to the UAVs information. From there, prove the statements of using UAV as a critical situation finder. The UAV can be attached to the system and give data to the object detection from the videos.

The system is tested on PCs, where a video, CSV-files from the UAV and text-files from the framework for data usage. The system could be used for anything where there is enough information, stated in the work description, section 6.. Information used is; GPS-coordinates from the UAV, the height of the UAV, Videos extruded from the UAV, object detection algorithm YOLO with other object detection supports, and OpenCV libraries to give the operator a way to verify the system.

7.3. Critical situations and redundancy

By using the UAV implementation and verifiable surveillance, integration provides redundancy and safety. People and developers at Volvo settled the critical situation and verified the system is working. The Critical situation is now based on algorithm proving object detection and, from there, adding simple, verifiable information to be verified by the Volvo employees and site operators. The state of a critical situation is landing in an area of the very definable situation created to support the system, even when the system needs multiple usages for a situation. The system can support the site-operator with enough information to state this as an extra safety layer and along the site operator to find critical situations before they happen. It also creates less accident where the site operator could have missed and not stopped.

7.4. Showcase of application

In this section, an essential showcase of the application is being published, giving the reader a basic understanding of how the system interferes with the site operator, user, or developer. Two different videos are being used, and these videos can be downloaded, and are attached to GitHub[2].

(a) Normal motion detection.

(b) Error since UAV moved, see Right square on frame.

Figure 18: Motion detection running showing off error that can be implicitly in motion detection object detection.

Motion detection, see Figure 18, detect moving object at such grace, it even notifies small movements coming from the UAV. When the wind or movement from the user, the motion detection sees this as a moving object and registers this as an object, this is filtering out, where such as small movement can still be excluded. However, if the UAV moves too much, errors occur.

(a) Tiny-YOLOv4.

(b) YOLOv4.

(c) Edge-detection in bird-eye-view.

Figure 19: Comparing between object detection where tiny YOLO found more object but no humans and had lower accuracy comparing to v4. Also viewing edge-detection in bird-eye-view.

Three different runs in Figure 19 gives the operator information that one object detection is faster but less accurate, another is more accurate but slower, while the third is edge-detection in bird-eye-view. Edge-detection does not change its accuracy or speed when changing weight-file, but since Tiny-YOLO support more frames for a specific time, edge-detection is more accurate because it has more picture to make estimations. YOLOv4 is a lot slower but has a higher accuracy when finding an object, while tiny-YOLOv4 can not find the human on-site but can find the TA15 in more frames than YOLOv4. Though this, tiny-YOLOv4 extrudes data as it is less accurate than YOLOv4, but sees the TA15 in more frames. This can also be viewed in Figure 21.

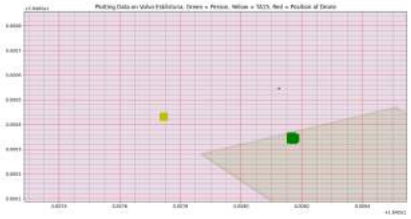
[2]GitHub installation manual with implementation https://github.com/Theralley/MasterThesis/releases.

(a) detected objects.

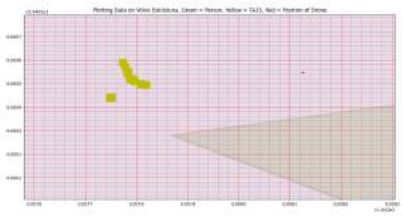
(b) Buffer of safety circles even if no object is detected.

Figure 20: Viewing the safety circles, buffering when object has not been detected.

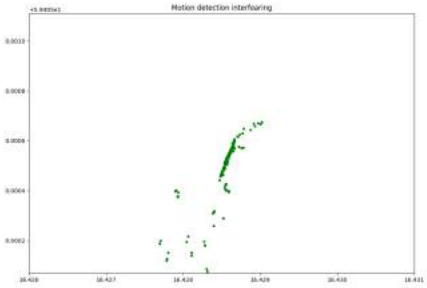
Safety circles see Figure 20, is used to notify the operator if something is happening. Safety circles are buffering after detecting an object, expanding for every frame until reaching a specific pixel-size. This buffering is because the object is not flying away from its position. The object detection may not just see the object on all the frames. The circles expand as an extra-safety layer told the operator that something inside of this circle was probably earlier detected.



(a) YOLOv4 pointed on gridmap.



(b) Tiny-YOLOv4 pointed on gridmap.



(c) Motion detection pointed on gridmap, error since UAV moved because of wind.

Figure 21: Comparing all runs pointed on gridmap, same video input.

A different version of grid maps referring to Figure 21 are being pointed into the matplotlib and pandas interface. The purpose is to show the different object detection results using a more prominent framework, smaller framework, and motion detection. By looking at motion detection, an error is being printed on the interface, while the difference between the two object detection is the amount of detection. We can see that there is more detection on tiny-YOLOv4 because of its easy-to-find object but less accuracy than YOLOv4. Thought, YOLOv4 found humans on-site, that tiny-YOLOv4 could not see, and since humans are in higher safety criteria than vehicles, YOLOv4 should be used to find humans on site.

8. Discussion

Other approaches during the thesis were considered, and the giving solution could have been utilizing other methods to identify and resolve each problem. How to the verifiable system working right now is no other methods than going to the site and comparing the result from the UAV and the outputs from the implementation. No given proofs, or data storage, from the TA15, was done during the runs. More accessibility to verify the system can be done remotely and whiting the already used autonomous system and site server. Even this, though, the system is verifiable as working when all inputs match the real-life inputs, leaving the critical part to sustain the system with accurate data from the operator and user.

From the beginning, running object detection had a speed of less than 0.005FPS, using YOLOv4 and simple direct-input from the video frames. Why is it like this? Well, the way of using object detection is the calculation on each frame, each picture. When just using your CPU, there are limitations because of processing pictures and running OpenCV library with YOLO. The easiest way to make the system faster would be to use a GPU, not just because of its processing power, but also since the way of calculating the pictures with OpenCV and YOLO. GPU releases values in the frame-stack, previously discussed, removing data from there, faster runs could have been done, and also accuracy would have been raised since there would be more pictures and frames to run into YOLO within the same amount of time, as we are conferrable have right now.

So the question stands, why not just implement this? Well, it is not that easy. First of all, using NN on a Windows PC is pretty complex to install. Using OpenCV is easier, but to combine the OpenCV and NN frameworks such as YOLO into a functional computer took some time but was possible. Even this, when developing an install guide, drivers and CUDA-support felt impossible. Every installation was going in the wrong directions, creating an object detection framework not usable and sometimes even forced to reinstall the OS. Why was it like this? Some human factors, maybe such as simple as not reading guides correctly or it was object detection install method, where some are still new and don't have any real good install guides. Even install the system on a PC has to take precautions and use Ubuntu. While using the WSL, we still used Ubuntu within the windows OS.

While laying the GPU support aside, the CPU method was doing fine. Jumping over ten frames per run empties the stack, leaving no frames left to run and didn't spent an unnecessary amount of time, while rending an interface for ten seconds in one second, leaving a good framework with provided solutions. Leaving the work, just knowing someone can install GPU-Cuda support, raising the frame-rates is still leaving possible way to improve just by implementing something easy, if they already know how to handle it.

Still, parts went well despite restrictions from the companies during pandemic Covid-19, and the result was going in the right direction. Volvo CE is feeling proud of the work, and myself of course, leaving new research areas and gaining experience in the subject.

Validity is proven as possible for all the research questions to verify the test result and applicable scenarios from the Videos recorded, though this has to be done manually and leaving a possibility for human errors. The system's verification is done by viewing the data and comparing it with different scenarios and real-life comparisons. The need for knowledge from the autonomous site and Volvo CE application of autonomous unis is high to verify that the system holds the goal as functional and applicable. The functional proven statement has been made with Volvo CE, and the information gathered from the autonomous site. To prove that the system is working, placements knowledge and human interaction confirm the system as working. Sadly working along with experienced site operators, and a fully functional system aside where there was no time to include the implementation into Volvo autonomous system or even accessibility to it, is still leaving a sad mark as lower accuracy and lower safety are a reality this. The test engineers at Volvo CE autonomous site though about this implementation will give not just the area of raising safety UAV, but also normal CCTV surveillance.

How about using this system in other areas? By using the pretrained NN YOLO, limitations were being held, limiting ourself to the trained dataset of those objects in that specific coco-set. Future development may not just train their network but also use the network to fit the implementation into other usage areas. One early test was running the DJI Mavic Air in the apartment (NOT RECOMMEND), while real-time-streaming was sending data into the framework, doing calculations on a real-time basis. Pros with this were that the implementing support areas were not focused and applicable for developing within other areas outside of autonomous open surface mines. Cons, the delay and, of course, flying inside, leaving states of madness and object rolling down on the floor, but all tests are good tests.

8.1. Ethical and Societal Considerations

The ethical part of this will be to define the line between the site operator and the system of systems. In this case, the UAV warn the operator if a critical situation occurs. If the operator does miss a situation, the UAV can support the operator with a reminder or warning about the situations that are to occur. The implemented solution is not meant to replace the operator at this level. However, it supports more rescue in the area.

For this implementation and level of making decisions, the site operator will still be the overhead controller and take responsibility if a critical situation occurs, ending in a hazardous or catastrophic situation. The site operator shall not see the UAV and implementation as a solution, and it shall be seen as a tool. The implementation's safety level supports the operator and sustains the operator with the necessary information and warnings, where the implementation shall not give any misleading information or given decision for the operator to make.

The given information for the operator shall always stand as a tool, where some states have been localized for the parts of the ethical issue at the moment:

- Focus: The operator can lose focus on the CCTV screens
- Decision making: Operator shall always be the one to press the emergency button if a critical situation is about to happen or not.

In the later development of the work, system of systems could be implemented, given the implementing rank of making its own decision. The society of using the implementation as a safety system, not a tool, moves the reliability and accuracy to maintain the site in safe states. How will the system handle situation where a given solution of saving multiple humans lives or forcing the vehicle to go into an unsafe environment? Shall we leave the decision making to the system instead of operators?

The implementation, such as the version today, human life's shall always be prioritized. Forcing vehicles into emergency brakes, saving human lives is the main priority. Sometimes, vehicles can not stop within time due to their speed and velocity, which may be forcing them to turn dangerously into the ditch without interfering with the human. This is just speculation, but finding critical or possible critical situations is a solution as the system requirements. The possible critical situation can be limit or even neutralized, given the grid map system's values. The grid map system gives the operator information about vehicles in specific areas and even human interaction, leaving the information for the vessels to slow down in those certain areas.

8.2. Reliance on support functions

This thesis provides a proof-of-concept on how to support the site operator to identify safety-critical situations. Such features provide information to a human, but the human must double-check the provided information. As both reliable and usable, using the system as a support for the site operator raises a question regarding the system's safety. There is a risk that the site operator relies on the data provided from the UAV and the data extracted. By giving the operator a functional system, such as this, the operator needs to have in mind the system's usability. Right now, the operator shall always be at guard, focused, and never wholly assuming the system identifying the critical situations at all states and times.

If this system would serve as a base for monitoring the site's safe operation, a thorough hazard analysis must be conducted to identify possible hazards from, for example, a failing UAV, object detection system, and mapping system. Based on those failures' criticality, stringent development and verification processes described in functional safety standards like IEC 61508 [62] or ISO 13849 [63] must be followed to develop this system.

This has not been the scope for this study, and instead, the site operator is responsible for monitoring the site's safe operation.

In the future, a direct interaction between the UAV and the autonomous vehicles is possible to, for example, command reduced speed in certain areas when critical situations are identified. It must be understood what failures of the system would lead to critical accidents.

9. Conclusion

Object detection library YOLO is implemented with other methods to gain accuracy and retain speed, such as motion detection and Edge-detection. The research study focuses on reaching the best speed and accuracy within the chosen NN matching the complexity. The usage of the most complex and fastest algorithm may credit to the best result, though other algorithms can prove that the implemented methods are safe enough. Therefore other methods are required to be compared, and situations need to be proved with the decided NN. Critical statement tests proved that the methods are safe enough to use in situations and proven their autonomous site usage in multiple test scenarios. More testing to prove the usage of multiple sites is needed, that the system is to test on more autonomous open surface mines.

9.1. Research Question

- **Research Question 1: How can a UAV be used to detect humans and vehicles at such a site?**
  First, it is necessary to understand the critical situations at a site where autonomous machines operate. This task is vital since the following tasks build upon this knowledge, and what is the chance of finding and not finding the humans on site. This is a study phase, where different algorithms are discussed, where the reason is because of the wide case-test area of the of such object detection. Different NN-algorithms can be better in different cases, and such as the algorithms, cases can variate. The main focus are to stop critical situations from happening and detect this critical situation, where cases are leaded to giving the operator and the reader more real-life events and real-case-experience.
- **Research Question 2: How to support the site operator in monitoring safety critical situations in an open surface mine with autonomous vehicles?**
  In this thesis, we present the information that is needed for the operator. This is presented as an interface leaving a 2D-map to show where objects are and a framework for the videos, with safety circles during the video framework and a definition of distance during 2D-map-view. The frameworks point out if a possible critical situation occurs or not, sustaining the operator with information when detecting objects. In contrast, the 2D-map points out the position of the object, also giving the operator the choice of defining a critical situation or not. The presented information needed for the operator has to be hard to misunderstand to reduce human errors and be reliable to sustain safety. Additionally, the data's accuracy will need to be assured and how to secure the data in and reliable manner, but right now, human error is a risk. Enough information is retrieved from the UAV to maintain reliable results that include GPS coordinates and Video-feed data. Basic non-real-time implementation of sensor data such as; coordinates, video, and height; is attached to a functional algorithm to show the proof of concept design, but these inputs need to be inserted by human hand. Therefore, human error can be a risk factor to have in mind. If all inputs are correct, the 2D map interface gives the operator easy ways to see if objects are placed on the site and where they are. The animated 2D map used right now is an excellent way to present the information to the operator. Therefore the operator shall see this interface as easy to follow and hard to misunderstand. Also, the views on the object detection run, where seeing boxes and safety circles, also give the operator multiple ways to have the information presented to sustain safety. This research question will then be verified as solved since a reasonable solution is implemented.

9.2. Implementations levels

The implementation level is the settled implementation levels that this thesis needs to accrue to prove the RQs and verify that it is functional and usable in proof of concept criteria.

Level 1
Video information and GPS coordinate are extruded from the UAV. The UAV are sustaining the user with information and coordinates to make it possible to do an action based on the UAV's information. The UAV can start, land, and store information, where the information can be extruded through a memory card or the *Flight Control Computer* (FLCC), and by this, prove if it is possible to find objects and their position on-site.

Status:Succeed
Comment:Python scripts are extruding information through pictures and videos. The videos are worked on YOLO-library since accuracy compared to the speed is adjustable to the computing power. The coordinates are being extruded by a script from the picture, called GeoTag. The video and pictures are being extruded through SD-Card and worked on a separate PC.

Level 2
Critical situations and information about objects on-site could be predicted. The system are compared to the defined object on site, and if every object is found, also to prove UAVs' support to find objects and find if the situation is either critical or not.

Status:Succeed
Comment:There are multiple ways to find a critical situation or not. To solve if a person's action could interfere with a vehicle such as a TA15 could easily be implemented as a vector-direction, where a vector is exported from the previous XY-coordinates to the new XY- coordinates. The vector gives a direction and could easily be extended. If vector meets, a critical situation is found, but also there is a circle around the object, where the defined circle shows safety distance from a person or vehicle. Suppose the circle interferes with one and another. In that case, the safety layer attaches and give the operator a warning or prove the vehicle to run at a slower speed to lower the chance of a critical situation.

Level 3
The UAV could support, by this point, the system with video and coordinates for the base table and visual algorithm. The data should contain enough information to make it possible to predict paths through the video and coordinates based on the collected data and prove that there is a possibility to find a critical situation before it has happened.

Status: Half-succeed
Comment: The implementation system can use different data inputs such as different media exports, for example, .mp4 .AVI, and real-time-stream. The data is enough to inform the operator about critical situations, but predicting the path is not solved. Vectors-integration is implemented in edge detection in birds-eye-view and from that, see where the vehicle is heading. Path-planning and vector-integration are not implemented in side-view, where YOLO is working best.

9.3. Future work

Future implementation should be combined with machine learning to make the system attachable to path-planning and track humans and machines with direction and speed. It is necessary to detect machines on camera pictures/films made by the UAV. However, as a second step, the movement direction needs to be calculated based on pictures/videos and GPS coordinates. This could be integrated into a separate computer calculation through vector-field and specification from the vehicles and humans. Some are implemented, such as a warning circle where when humans a vehicle comes to close, we get a warning, though some better path-planning algorithm needs to be done to make the system more reliable where the object is going.

9.3.1 System of systems

System-of-systemd (SOS) provides higher security and accuracy and can also support the operator with instant information between the UAV and vehicles, without interfering. The main issue is the accuracy of the UAV, where the need for testing and providing a low-error-rate, and that the system is applicable in any situation. Applying the UAV and implementing the thesis into SOS can result in an error from other vehicles or false-accusations when dealing with real-time implementing. A good thing about the SOS is using the information to gain and raise the thesis-implementation accuracy. Padmaja et al. referring IoT (internet of things) to gain accuracy and better road-handling such as GPS-rescheduling and development of driving with autonomous cars [64]. The thesis implementation can use the same way. Using the coordinates from the autonomous vehicles, an estimation can be done and proving where the object is not just by object detection but also by their coordinates. There is also a possibility to see if something is drifting away or not, like a TA15 is not following its path.

Even though the system is working, easy-to-use, and simple to install, the proof of stating this system as verifiable and validated guarantees accuracy to be used in industrial and fulfill the ISO-standards. Future development could focus on this, combing the ISO-standards from the autonomous site to the implementing of the UAV.

9.3.2 Path planning

Losing its purpose, the thesis focused on handling path-planning into the gridmap system and showing the next move of vehicles. The plan was to identify the vehicles' vectors, and from there, see the speed and velocity. If vectors were colliding, the object would meet after a specific time, where the operator shall be warned that the object has their path planning to collide with each other. This work was laid aside, but since the gridmap system uses coordinates and interaction with maps, path-planning can be done where the previous thought can be as a "predicament" on where objects are moving. The purpose lost its chance since there was no time for one student to create multiple object detection, gridmap, and path-planning to develop higher safety.

9.3.3 Open source UAV

The limitation was needed to fulfill the thesis within the schedule. Using a closed source UAV was necessary to prove the concept of using UAV on site. Video recording was in focus, and nothing else, but when the UAV needed to extract information for calculations and other tasks, the closed source was not the right way to go. BY having an open source, such as pixhawk [50], where extruded all data directly could simplify the inputs into the codes and support the system in a faster and more accurate way. The open drone, [48], used a pixhawk system but had no camera included. This thesis's future work needs to be done, apply an open source flight controller, creating easier access to data, and easier development of real-time-system. The Pixhawk system is just an example, but the need for easier access to data and essentially the longitude and latitude of the drone. The system needs some more attachment of development, where the implementation of an open source UAV could support better data support and better accuracy, decreasing human errors.

There are multiple ways to remove and add more UAV to an autonomous site, where the availability is enormous due to its multipurpose usage. The need of a Wi-Fi camera would be less since the UAV can stay in the air, but what about the battery time? Well, there is some research about the drone being in the air for hours, due to its substantial carriage weights and easy to combine the system with higher altitudes or moving the camera for better surveillance [65].

9.3.4 Masking

IR-Cameras and a Raspberry PI have been provided by company NEP since using IR and Mask technology could potentially masse away non-needed objects and, therefore, filter away unnecessary errors [66]. The system applies as a separate system, where a single Raspberry PI is attached to the drone's bottom and the IR-camera watching down. Data saved as a video and could potentially be masking material. An appropriate way to find humans, animals, and machinery even during night-time and Winter. The limits to this application are that the camera is separated from the DJI-drone and is limited to bird-eye-view, but in later works, the system could be more developed, giving better masking methods and filtering of data.

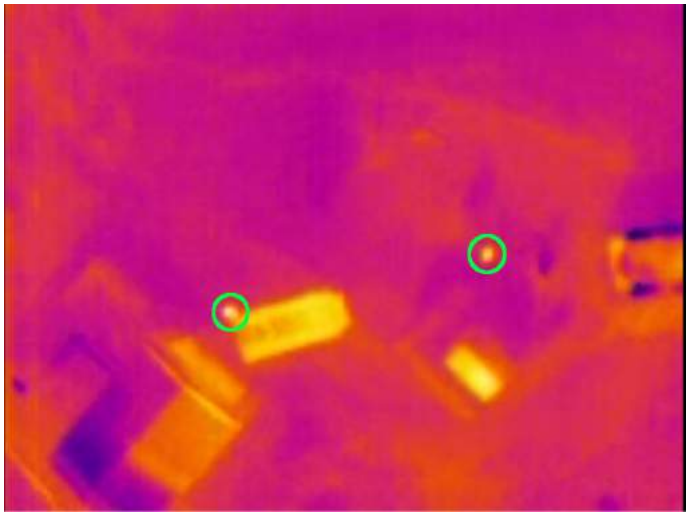


Figure 22: IR heat detection, circled around humans.

References

[1] C. Schultz, "This picture of boston, circa 1860, is the world's oldest surviving aerial photo," Apr 2013, accessed: 2021-01-04. [Online]. Available: https://www.smithsonianmag.com/smart-news/this-picture-of-boston-circa-1860-is-the-worlds-oldest-surviving-aerial-photo-14756301/

[2] "Amazon wins faa approval for prime air drone delivery fleet," accessed: 2021-01-04. [Online]. Available: https://cnb.cx/3lz65T7

[3] L. Hassan-Esfahani, A. Torres-Rua, A. M. Ticlavilca, A. Jensen, and M. McKee, "Topsoil moisture estimation for precision agriculture using unmmaned aerial vehicle multispectral imagery," in *2014 IEEE Geoscience and Remote Sensing Symposium*, 2014, pp. 3263–3266.

[4] S. N. A. M. Ghazali, H. A. Anuar, S. N. A. S. Zakaria, and Z. Yusoff, "Determining position of target subjects in maritime search and rescue (MSAR) operations using rotary wing unmanned aerial vehicles (UAVs)," pp. 1–4. [Online]. Available: http://ieeexplore.ieee.org/document/7890765/

[5] Volvo Construction Equipment, "Electric Site Project," accessed: 2021-01-04. [Online]. Available: https://www.volvoce.com/global/en/news-and-events/news-and-press-releases/2018/carbon-emissions-reduced-by-98-at-volvo-construction-equipment-and-skanskas-electric-site/

[6] P. Zhu, L. Wen, D. Du, X. Bian, Q. Hu, and H. Ling, "Vision meets drones: Past, present and future," accessed: 2021-01-04. [Online]. Available: http://arxiv.org/abs/2001.06303

[7] P. Fahlstrom and T. Gleason, *Introduction to uav systems.* Wiley, 2013.

[8] *Unmanned aircraft systems: UAS*, ser. ICAO circular. International Civil Aviation Organization, no. 328, OCLC: 729908130.

[9] J. Rogers, "drone warfare: the death of precision," 2020, accessed: 2021-01-04. [Online]. Available: https://thebulletin.org/2017/05/drone-warfare-the-death-of-precision/

[10] S. Li and D.-Y. Yeung, "Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models," p. 7.

[11] M. Suk, Julian, C. Tuazon, and J. Woetzel, "Can drones defeat malaria? - dji drones for good," Mar 2020, accessed: 2020-11-20. [Online]. Available: https://content.dji.com/fighting-malaria-with-drones/

[12] F. A. A. (FAA), *Airplane Flying Handbook.* Aviation Supplies Academics, Inc., 2017.

[13] International Organization for Standardization, "ISO 26262:2018 - Road vehicles – Functional safety," 2018.

[14] A. McAfee and E. Brynjolfsson, *Machine, platform, crowd.* Norton Company, 2017.

[15] "How machine learning algorithms make self-driving cars a reality — intellias blog," accessed: 2021-01-04. [Online]. Available: https://www.intellias.com/how-machine-learning-algorithms-make-self-driving-cars-a-reality/

[16] N. Shijith, P. Poornachandran, V. G. Sujadevi, and M. M. Dharmana, "Breach detection and mitigation of uavs using deep neural network," in *2017 Recent Developments in Control, Automation Power Engineering (RDCAPE)*, 2017, pp. 360–365.

[17] BOLO, "A basic introduction to neural networks," accessed: 2021-01-04. [Online]. Available: http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html

[18] L. Weng, "Object detection for dummies part 3: R-cnn family," Dec 2017, accessed: 2021-01-04. [Online]. Available: https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

[19] X. Long, X. Long, J. Reyes, A. Bolen, and M. Grau, "Understanding object detection in deep learning," 2020, accessed: 2021-01-04. [Online]. Available: https://blogs.sas.com/content/subconsciousmusings/2018/11/19/understanding-object-detection-in-deep-learning/

[20] S. Insights, "Computer vision: What it is and why it matters," 2020, accessed: 2021-01-04. [Online]. Available: https://www.sas.com/en_us/insights/analytics/computer-vision.html

[21] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018, cite arxiv: 1804.02767, Comment: Tech Report. [Online]. Available: http://arxiv.org/abs/1804.02767

[22] L. Liu, W. Ouyang, X. Wang, P. W. Fieguth, J. Chen, X. Liu, and M. Pietikäinen, "Deep learning for generic object detection: A survey," *CoRR*, vol. abs/1809.02165, 2018. [Online]. Available: http://arxiv.org/abs/1809.02165

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[24] I. Kilic and G. Aydin, "Traffic sign detection and recognition using tensorflow' s object detection api with a new benchmark dataset," in *2020 International Conference on Electrical Engineering (ICEE)*, 2020, pp. 1–5.

[25] F. L. WANG Zhongmin1£¬2, DUAN Na1, "Object tracking algorithm fused with yolo detection and meanshift," *Computer Engineering and Applications*, vol. 55, no. 10, p. 186, 2019. [Online]. Available: http://cea.ceaj.org/EN/abstract/article_37761.shtml

[26] "Python3.8.x documentation," 2020, accessed: 2021-01-04. [Online]. Available: https://docs.python.org/3.8/

[27] "ubuntu docs 20.04 (ubuntu serverguides for previous lts versions: 18.04 and 16.04)," 2020, accessed: 2021-01-04. [Online]. Available: https://ubuntu.com/server/docs

[28] "install windows subsystem for linux (wsl) on windows 10," 2020, accessed: 2021-01-04. [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/install-win10

[29] "Iso standards are internationally agreed by experts," 2020, accessed: 2021-01-04. [Online]. Available: https://www.iso.org/standards.html

[30] S. Baumgart, J. Fröberg, and S. Punnekkat, "Can stpa be used for a system-of-systems? experiences from an automated quarry site," 2020.

[31] "Volvo autonomous hauler wins red dot award 2020," accessed: 2021-01-04. [Online]. Available: https://www.volvoce.com/global/en/news-and-events/press-releases/2020/volvo-autonomous-hauler-wins-red-dot-award/

[32] United States Department of Defense, *MIL-STD 1629A - Procedures for Performing a Failure Mode, Effect and Criticality Analysis*, 1980, accessed: 2021-01-04. [Online]. Available: http://www.fmea-fmeca.com/milstd1629.pdf

[33] C. Ericson, *Hazard analysis techniques for system safety*. Wileys, 2016.

[34] D. Jaiswal and P. Kumar, "Real-time implementation of moving object detection in UAV videos using GPUs," vol. 17, no. 5, pp. 1301–1317. [Online]. Available: http://link.springer.com/10.1007/s11554-019-00888-5

[35] P. Zhang, Y. Zhong, and X. Li, "SlimYOLOv3: Narrower, faster and better for real-time UAV applications," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, pp. 37–45.

[36] C. Supeshala, "Yolo v4 or yolo v5 or pp-yolo?" Aug 2020, accessed: 2021-01-04. [Online]. Available: https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109

[37] A. Ryan and J. Hedrick, "A mode-switching path planner for UAV-assisted search and rescue," pp. 1471–1476, 2005.

[38] Q. Wu and Y. Zhou, "Real-time object detection based on unmanned aerial vehicle," in *2019 IEEE 8th Data Driven Control and Learning Systems Conference (DDCLS)*. IEEE, pp. 574–579.

[39] S. Vasavi, N. K. Priyadarshini, and K. H. Vardhan, "Invariant feature based darknet architecture for moving object classification," pp. 1–9.

[40] S. Razakarivony and F. Jurie, "Vehicle detection in aerial imagery : A small target detection benchmark," vol. 34, pp. 187–203.

[41] T. N. Mundhenk, G. Konjevod, W. A. Sakla, and K. Boakye, "A large contextual dataset for classification, detection and counting of cars with deep learning," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Springer International Publishing, vol. 9907, pp. 785–800, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-46487-9_48

[42] G. Welch and G. Bishop, "An introduction to the kalman filter," USA, Tech. Rep., 1995.

[43] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[44] C. Rabe, U. Franke, and S. Gehrig, "Fast detection of moving objects in complex scenarios," in *2007 IEEE Intelligent Vehicles Symposium*, 2007, pp. 398–403.

[45] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016, accessed: 2021-01-04.

[46] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen, "Pp-yolo: An effective and efficient implementation of object detector," 2020.

[47] A. Brdjanin, N. Dardagan, D. Dzigal, and A. Akagic, "Single object trackers in opencv: A benchmark," in *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, 2020, pp. 1–6.

[48] E. Beckman, R. Hamrén, J. Harborn, L. Harenius, D. Nordvall, A. Rad, and D. Ramic, "Open drone - FLA400 - project in dependable system," p. 22.

[49] DJI, "Dji mavic air - foldable 4k drone - dji," accessed: 2020-12-10. [Online]. Available: https://www.dji.com/se/mavic-air

[50] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Auton. Robots*, vol. 33, no. 1–2, p. 21–39, Aug. 2012. [Online]. Available: https://doi.org/10.1007/s10514-012-9281-4

[51] "dji go 4 - dji download center," 2020, accessed: 2020-12-10. [Online]. Available: https://www.dji.com/se/downloads/djiapp/dji-go-4

[52] P. Adarsh, P. Rathi, and M. Kumar, "Yolo v3-tiny: Object detection and recognition using one stage improved model," in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020, pp. 687–694.

[53] H. Soebhakti, S. Prayoga, R. A. Fatekha, and M. B. Fashla, "The real-time object detection system on mobile soccer robot using yolo v3," in *2019 2nd International Conference on Applied Engineering (ICAE)*, 2019, pp. 1–6.

[54] C. Veness, "Calculate distance, bearing and more between latitude/longitude points," accessed: 2021-01-04. [Online]. Available: https://www.movable-type.co.uk/scripts/latlong.html

[55] R. H. Revision, "Yolo - object detection," accessed: 2021-01-04. [Online]. Available: https://opencv-tutorial.readthedocs.io/en/latest/yolo/yolo.html

[56] A. Rosebrock, "Basic motion detection and tracking with python and opencv," PyImageSearch, accessed: 2021-01-04. [Online]. Available: https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/

[57] T. pandas development team, "pandas-dev/pandas: Pandas," Feb. 2020, accessed: 2021-01-04. [Online]. Available: https://doi.org/10.5281/zenodo.3509134

[58] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[59] D. Sarkar, *Mastering Nginx*. Packt Publishing, 2013.

[60] G. Gkioxari, J. Malik, and J. Johnson, "Mesh r-CNN," accessed: 2020-12-10. [Online]. Available: http://arxiv.org/abs/1906.02739

[61] A. Rosebrock, "Deep learning: How opencv's blobfromimage works - pyimagesearch," 2020, accessed: 2020-12-10. [Online]. Available: https://www.pyimagesearch.com/2017/11/06/deep-learning-opencvs-blobfromimage-works/

[62] International Electrotechnical Comission, "IEC 61508:2010 Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," 2010.

[63] International Organization for Standardization, "ISO 13849:2015 Safety of machinery - Safety related parts of control systems," 2015.

[64] B. Padmaja, P. V. Narasimha Rao, M. Madhu Bala, and E. K. Rao Patro, "A novel design of autonomous cars using iot and visual features," in *2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2018 2nd International Conference on*, 2018, pp. 18–21.

[65] "Tethered mini uav for permanent surveillance," accessed: 2021-01-04. [Online]. Available: https://www.ecagroup.com/en/solutions/tethered-mini-uav-permanent-surveillance

[66] D. Bengtsson and W. Löw, "NON-CONTACT PCB FAULT DETECTION USING NEAR FIELD MEASUREMENTS AND THERMAL SIGNATURES," p. 53. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1446564/FULLTEXT01.pdf

[67] A. Rosebrock, "Target acquired: Finding targets in drone and quadcopter video streams using python and opencv," PyImageSearch, accessed: 2021-01-04. [Online]. Available: https://www.pyimagesearch.com/2015/05/04/target-acquired-finding-targets-in-drone-and-quadcopter-video-streams-using-python-and-opencv/

[68] Abdulkhadeer0200, "Real-time-object-detection-using-yolo," https://github.com/Abdulkhadeer0200/Real-time-object-detection-using-YOLO/blob/master/real_time_yolo.py, commit = 5d77ca5c23928df4f851471ad90ee413728f1c2b, 2019, accessed: 2021-01-04.

Appendices

A Object detection

The object detection code is divided into two different parts, the edge detection and the object detection suing YOLO neural network.

Edge detection is based on Rosebrock's target acquire, [67], where changing the edges to fit the TA15 was applied and changed. That created a fast and accurate method to detect the TA15 from bird-eye-view, as previously mentioned.

A developer at Github inspires the object detection part, user Abdulkhadeer0200, [68].

After the inspiration was gathered, the code was changed to fit the implementation, where functions and equations were added to create useful software for the research questions and make a usable application.

```
#Verison: 0.1.8
#Author: Rhn15001, Rasmus Hamren
#Applying UAVs to Support the Safety in Autonomous Operated Open Surface Mines

#Edge-detection is inspired by reference:
#A. Rosebrock," Targetacquired: Findingtargetsindroneandquad-
#coptervideostreamsusingpythonandopencv," PyImageSearch,
#accessed:2021-01-04.[Online].
#Available: https://tinyurl.com/y6kd3tsy

#Object detection, YOLO, is inspired by reference:
#Abdulkhadeer0200,
#"Real-time-object-detection-using-yolo,"
#https://tinyurl.com/y34qtepv,
#commit= 5d77ca5c23928df4f851471ad90ee413728f1c2b, 2019, accessed: 2021-01-04.

import cv2
import numpy as np
import time
import sys
import math
from math import sin
import imutils
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage.transform import (hough_line, hough_line_peaks)

imagename = sys.argv[1]
yoloV = sys.argv[2]
tt = sys.argv[3]
heightD = sys.argv[4]
camangle = sys.argv[5]

file = open("out.txt","w")
file.write("label,x,y,l,angle,offset\n");

md = (int(heightD)/(math.sin(math.radians(90)-math.radians(int(camangle)))))
md = int(md)

#Variables
c2 = 0
a = 1
xo = 0; xsave = 0; xsaveV = 0
yo = 0; ysave = 0; wsave = 0; ysaveV = 0; wsaveV = 0
diff = 2
yxdiff = 1
yydiff = 1
t = 0
line_thickness = 2
Multiplier = 0.5
OldCentrumX = 0
OldCentrumY = 0
add = 0
loop_stopper = loop_stopper2 = 0
```

```
#Load YOLO

# Original yolov4
if(yoloV == "v4"):
    net = cv2.dnn.readNet("yolov4.weights","yolov4.cfg")

# Original yolov3
if(yoloV == "v3-tiny"):
    net = cv2.dnn.readNet("yolov3-tiny.weights","yolov3-tiny.cfg")

# Tiny yolov4
if(yoloV == "v4-tiny"):
    net = cv2.dnn.readNet("yolov4-tiny.weights","yolov4-tiny.cfg")

# Tiny yolov3
if(yoloV != "v3-tiny" and yoloV != "v4" and yoloV != "v4-tiny"):
    print("Enter value after video name for Weight: v4, v3-tiny")

classes = []

with open("coco.names","r") as f:
    classes = [line.strip() for line in f.readlines()]

#print(classes)

#Load CV
im = mpimg.imread("ResultTemp.png")
cv2.imwrite("ResultDot.png",im)
im = mpimg.imread("ResultDot.png")

layer_names = net.getLayerNames()
outputlayers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers()]

colors = np.random.uniform(0,255,size=(len(classes),3))

#loading image
font = cv2.FONT_HERSHEY_PLAIN
frame_id = 0
cap = cv2.VideoCapture(imagename)

#FRAME TO START ON
for x in range(0, int(tt)):
    _,frame= cap.read()

while True:
    (grabbed, frame) = cap.read()

#JUMPOVER PICTURES#
    if(yoloV == "v4-tiny" or yoloV == "v3-tiny"):
        for x in range(0, 5):
            _,frame= cap.read()

    if not (yoloV == "v4-tiny" or yoloV == "v3-tiny"):
        for x in range(0, 10):
            _,frame= cap.read()

#Flip if want to
    #frame = cv2.flip(frame,-1) #FlipTest

    time.sleep(0.05)

    if not grabbed:
        break
        print("ERROR WHILE GRAB")

    ##CHANGE COLOR SPECTRUM

    b, g, r = cv2.split(frame)
    frame = cv2.merge((b, g, r))
```

```
#detecting objects
blob = cv2.dnn.blobFromImage(frame,0.00392,(416,416),
(0,0,0),True,crop=False) #reduce 416 to 320

net.setInput(blob)
outs = net.forward(outputlayers)

#Showing info on screen/ get confidence score of algorithm in
#detecting an object in blob
class_ids=[]
confidences=[]
boxes=[]

height,width,channels = frame.shape
#print(frame.shape[1])
#print(frame.shape[0])
if(frame.shape[1] > 0 and loop_stopper == 0):
    pl = po = (frame.shape[1] * 0.021875)
    loop_stopper = 1

#########EDGE FIXES#############
# convert the frame to grayscale, blur it, and detect edges
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
gray = frame
blurred = cv2.GaussianBlur(gray, (3, 3), 5)
edged = cv2.Canny(blurred, 200, 500)
#find contours in the edge map
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)

#cv2.imshow("Edged",edged)
cnts = imutils.grab_contours(cnts)

# loop over the contours
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.2:

            #onject detected
            center_x= int(detection[0]*width)
            center_y= int(detection[1]*height)
            w = int(detection[2]*width)
            h = int(detection[3]*height)

            #rectangle co-ordinaters
            x=int(center_x - w/2)
            y=int(center_y - h/2)

            boxes.append([x,y,w,h]) #put all rectangle areas

            #how confidence was that object detected and show that percentage
            confidences.append(float(confidence))

            class_ids.append(class_id) #name of the object tha was detected

indexes = cv2.dnn.NMSBoxes(boxes,confidences,0.4,0.6)

for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        confidence = confidences[i]
        color = colors[class_ids[i]]

        if(yoloV == "v4" or yoloV == "v3"):
            acc = 0.70
```

```
            if not (yoloV == "v4" or yoloV == "v3"):
                acc = 0.40

            if(round(confidence,2) >= acc):
                if(label == "person"):

                    p1 = [frame.shape[1]/2, frame.shape[0]]
                    p2 = [x, y]
                    distance = math.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2))

                    ans = md/(frame.shape[0]/2)
                    distance = ans * distance

                    pl = h
                    pl = pl*0.6+po*0.4
                    po = pl
                    lp = 2.3/pl

                    cv2.rectangle(frame,(x,y),(x+w,y+h),color,2)
                    cv2.putText(frame,label+" "+str(round(confidence,2)),
                    (x,y+30),font,1,(255,0,0),2)

                    c2 = math.sqrt((((int(frame.shape[1]/2)-x) *
                    (lp))*((int(frame.shape[1]/2)-x) * (lp)) + (md*md)))

                    c3 = (c2 + distance)/2

                    #Buffer
                    xsave = int(x+(w/2))
                    ysave = int(y+(h/2))
                    wsave = w

                    print("Person length from offset: ", c3)

                    im = cv2.circle(im, (xsaveV,ysaveV), radius=5,
                    color=(0, 0, 255), thickness=-1)
                    cv2.imwrite("ResultDot.png",im)
                    if(t == 100):
                        cv2.imwrite("ResultDot.png",im)
                        t = 0

                    #Angle from offset calculation (line to offset point,
                    #line to object == angle in degree)
                    angle2 = math.atan2(frame.shape[1]/2 - int(y+(h/2)),
                    int(x+(w/2)) - frame.shape[1]/2) * 180.0 / math.pi;

                    angle2 = 90 - angle2
                    print("Person angle from offset: ", angle2, "\n")

                    c3 = c3 * math.sin(math.radians(int(camangle)))

                    file.write(str(label)); file.write(",");
                    file.write(str(int(x+(w/2)))); file.write(",");
                    file.write(str(int(y+(h/2)))); file.write(",");
                    file.write(str(c3)); file.write(",");
                    file.write(str(angle2)); file.write(",");
                    file.write(str(md)); file.write("\n");

                    t = t + 1

                elif(label == "TA15" or label == "truck" or
                label == "motorbike" or label == "skateboard"
                or label == "car"):

                    if(label == "motorbike" or label == "skateboard"):
                        label = "TA15"

                    p1 = [frame.shape[1]/2, frame.shape[0]]
                    p2 = [x, y]
                    distance = math.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2))
```

```
                    ans = md/(frame.shape[0]/2)
                    distance = ans * distance

                    pl = h
                    pl = pl*0.6+po*0.4
                    po = pl
                    lp = 2.3/pl

                    cv2.rectangle(frame,(x,y),(x+w,y+h),color,2)
                    cv2.putText(frame,label+" "+str(round(confidence,2)),
                    (x,y+30),font,1,(255,0,0),2)

                    c2 = math.sqrt((((int(frame.shape[1]/2)-x) *
                    (lp))*((int(frame.shape[1]/2)-x) * (lp)) + (md*md)))

                    c3 = (c2 + distance)/2

                    #Buffer
                    xsaveV = int(x+(w/2))
                    ysaveV = int(y+(h/2))
                    wsaveV = w

                    print("Vechile length from offset: ", c3)

                    im = cv2.circle(im, (xsaveV,ysaveV), radius=5,
                    color=(0, 0, 255), thickness=-1)

                    cv2.imwrite("ResultDot.png",im)
                    if(t == 100):
                        cv2.imwrite("ResultDot.png",im)
                        t = 0

                    #Angle from offset calculation (line to offset point,
                    #line to object == angle in degree)
                    angle2 = math.atan2(frame.shape[1]/2 - int(y+(h/2)),
                    int(x+(w/2)) - frame.shape[1]/2) * 180.0 / math.pi;
                    angle2 = 90 - angle2
                    print("Vehicle angle from offset: ", angle2, "\n")

                    c3 = c3 * math.sin(math.radians(int(camangle)))

                    file.write(str(label)); file.write(",");
                    file.write(str(int(x+(w/2)))); file.write(",");
                    file.write(str(int(y+(h/2))));
                    file.write(","); file.write(str(c3)); file.write(",");
                    file.write(str(angle2)); file.write(",");
                    file.write(str(md)); file.write("\n");
                    t = t + 1

                if not (label == "TA15" or label == "truck" or label == "car" or
                label == "motorbike" or label == "skateboard" or
                label == "person"):
                    cv2.circle(frame,(center_x,center_y),10,(0,255,0),2)
                    print(label)
                    cv2.putText(frame," UNDEFINED OBJECT; BE CAREFUL ",
                    (x,y+30),font,1,(255,0,0),2)

    for c in cnts:
      #print('Test: ', add)
      add = add + 1
      # approximate the contour
      peri = cv2.arcLength(c, True)
      approx = cv2.approxPolyDP(c, 0.01 * peri, True)
      # ensure that the approximated contour is "roughly" rectangular
      if len(approx) >= 3 and len(approx) <= 500:
        # compute the bounding box of the approximated contour and
        # use the bounding box to compute the aspect ratio
        (x, y, w, h) = cv2.boundingRect(approx)
        aspectRatio = w / float(h)
```

```
        # compute the solidity of the original contour
        area = cv2.contourArea(c)
        hullArea = cv2.contourArea(cv2.convexHull(c))
        solidity = area / float(hullArea)
            # compute whether or not the width and height, solidity, and
        # aspect ratio of the contour falls within appropriate bounds
        keepDims = w > 0 and h > 35
        keepSolidity = solidity > 0.9
        keepAspectRatio = aspectRatio >= 0.4 and aspectRatio <= 2.0

            ####DRAW EVERYTHING
        #cv2.drawContours(frame, [approx], -1, (0, 0, 255), 4)

        if keepDims and keepSolidity and keepAspectRatio:

                    if(y == 0):
                        y = 1
                    if(x == 0):
                        x = 1

                    xdiff = (xo/x)
                    ydiff = (yo/y)

                    if xdiff < 0.5 or xdiff > 1.3 or ydiff < 0.5 or ydiff > 1.3:
                        cv2.putText(frame," UNDEFINED OBJECT; BE CAREFUL ",
                        (x,y+200),font,1,(255,0,0),2)
                        #print("ERROR")
                    else:
                        cv2.drawContours(frame, [approx], -1, (0, 255, 0), 4)

                        x1 = approx[0][0][0]; y1 = approx[0][0][1]
                        x2 = approx[3][0][0]; y2 = approx[3][0][1]

                        cv2.putText(frame,"TA15", (x,y+200),font,1,(255,0,0),2)
                        cv2.line(frame, (x1, y1), (x2, y2), (255, 0, 0),
                        thickness=line_thickness)
                        x3 = x2 - x1; y3 = y2 - y1
                        x4 = x1 - x2; y4 = y1 - y2

                        CentrumX = approx[0][0][0]-50; CentrumY = approx[0][0][1]+50

                        #Where to go
                        #Back
                        cv2.line(frame, (x1, y1), ((x1 - x3*2)+100,
                        (y1 - y3*2)+100), (0, 100, 255), thickness=line_thickness)
                        cv2.line(frame, (x1, y1), ((x1 - x3*3),
                        (y1 - y3*3)), (0, 0, 255), thickness=line_thickness)
                        cv2.line(frame, (x1, y1), ((x1 - x3*2)-100,
                        (y1 - y3*2)-100), (0, 100, 255), thickness=line_thickness)
                        #print(x1, y1, x2, y2)
                        #Front
                        cv2.line(frame, (x2, y2), ((x2 - x4*2)-100,
                        (y2 - y4*2)-100), (0, 0, 255), thickness=line_thickness)
                        cv2.line(frame, (x2, y2), ((x2 - x4*3),
                        (y2 - y4*3)), (0, 0, 255), thickness=line_thickness)
                        cv2.line(frame, (x2, y2), ((x2 - x4*2)+100,
                        (y2 - y4*2)+100), (0, 0, 255), thickness=line_thickness)

                        layedCir = cv2.circle(frame, (CentrumX, CentrumY),
                        radius=200, color=(0, 0, 255), thickness=1)

                        CentrumX = ((((Multiplier*CentrumX)) +
                        ((1 - Multiplier)*OldCentrumX)))

                        CentrumY = ((((Multiplier*CentrumY)) +
                        ((1 - Multiplier)*OldCentrumY)))

                        OldCentrumX = CentrumX; OldCentrumY = CentrumY

                    xo = x
```

```
                yo = y

    #camangle compared to north

    cv2.circle(frame,(int(frame.shape[1]/2),int(frame.shape[0]/2)),2,
    (0,255,0),thickness=-1)

    print (wsave, wsaveV)

    #CircleBufferForSafety
    if (wsave < 150 and wsave != 0):
        layedCir3 = cv2.circle(frame, (int(xsave), int(ysave)),
        radius=100+wsave, color=(0, 0, 255), thickness=1)
    wsave = wsave * 1.1
    wsave = int(wsave)

    if (wsaveV < 150 and wsaveV != 0):
        layedCir2 = cv2.circle(frame, (int(xsaveV), int(ysaveV)),
        radius=100+wsaveV, color=(0, 0, 255), thickness=1)
    wsaveV = wsaveV * 1.1
    wsaveV = int(wsaveV)

    cv2.imshow("Image",frame)

    key = cv2.waitKey(1) #wait 1ms the loop will start again

    if key == 27: #esc key stops the process
        cv2.imwrite("ResultDot.png",im)
        break;

cap.release()
cv2.destroyAllWindows()
```

B Motion detection

This motion detection is based on Rosebrock basic motion detection, [56]. Of course, the focus was to prove the statement of using multiple detection methods, and Rosebrock's method of using motion detection was both fast and accurate. The main changes that have been done are to fit the motion detection into the implementation, extractions of data, and filtering the motions for the UAV-videos to work. Movement of the UAV was an error that needed to be solved for this to work. Also, safety circles were inserted.

```
#Verison: 0.1.8
#Author: Rhn15001, Rasmus Hamren
#Applying UAVs to Support the Safety in Autonomous Operated Open Surface Mines

#Inspired by reference:
#A. Rosebrock, "Basic motion detection and tracking with python
#and opencv," PyImage-Search, accessed: 2021-01-04. [Online],
#Available: https://tinyurl.com/ycwmrwxa

from imutils.video import VideoStream
import argparse
import imutils
import cv2
import matplotlib.image as mpimg
import statistics
import sys
import math

line = 0

font = cv2.FONT_HERSHEY_PLAIN

i = 0
x = 1; y = 1; xo = 0; yo = 0; ho = 0; wo = 0; xv = 0; yv = 0; ao = 0; w = 0
h = 0

loop_stopper = loop_stopper2 = 0

a = 0
r = 0; oldC = 0

line_thickness = 2

radius = 0
radiusO = 0
circleO = 0

square = 0

heightD = 20
camangle = 20

md = (int(heightD)/(math.sin(math.radians(90)-math.radians(int(camangle)))))
md = int(md)

file = open("outMotion.txt","w")
file.write("label,x,y,l,angle,offset\n");

im = mpimg.imread("ResultTemp.png")
cv2.imwrite("ResultDot.png",im)
im = mpimg.imread("ResultDot.png")

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-v", "--video", help="path to the video file")
args = vars(ap.parse_args())

# If video input is none, open webcam (only works in real Ubuntu eviroment)
if args.get("video", None) is None:
  cap = VideoStream(src=0).start()
  time.sleep(2.0)
```

```
# otherwise, read video input
else:
  cap = cv2.VideoCapture(args["video"])

# initialize the first frame
firstFrame = None

# loop over the frames of the video
while True:

  # grab the current frame and initialize
  frame = cap.read()
  frame = frame if args.get("video", None) is None else frame[1]

  # if a frame is none, end of video
  if frame is None:
    break

  if(frame.shape[1] > 0 and loop_stopper == 0):
      pl = po = (frame.shape[1] * 0.021875)
      loop_stopper = 1

  # resize, convert it to grayscale, and blur it
  gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
  gray = cv2.GaussianBlur(gray, (3, 3), 5)
  gray = cv2.Canny(gray, 200, 200)

  # save frame buffer
  if firstFrame is None:
    firstFrame = gray
    continue

  # absolute difference
  frameDelta = cv2.absdiff(firstFrame, gray)
  thresh = cv2.threshold(frameDelta, 25, 255, cv2.THRESH_BINARY)[1]

  # create threshold and find contours
  thresh = cv2.dilate(thresh, None, iterations=1)
  cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)
  cnts = imutils.grab_contours(cnts)

  # saving buffer in frames, can be changed but can not be used on UAV if other
  for c in cnts:
    if i == 1:
      firstFrame = gray
      i = 0
    # if the contour is too small, ignore it
    if cv2.contourArea(c) < 200:
      continue

    # compute box

    (x, y, w, h) = cv2.boundingRect(c)

    w = wo*0.7+w*0.3
    h = ho*0.7+h*0.3

    square = cv2.rectangle(frame, (int(x), int(y)), (int(x) + int(w),
    int(y) + int(h)), (0, 255, 0), 2)

    xv = x - xo
    yv = y - yo

    #creating radius and safety circles, but since no real size is being
    #introduced this is will be for future work
    radius=int(80*(w/h))
    radius = radiusO*0.95+radius*0.05
    radiusO = radius
```

```
    circle = cv2.circle(frame, (int(x+w/2), int(y+h/2)), int(radius),
    color=(0, 0, 255), thickness=1)

    #Buffer
    circleO = circle
    if(a == 15):
      xo = x; yo = y; wo = w; ho = h
      a = 0
    ao = a

    #Circle print
    im = cv2.circle(im, (int(x), int(y)),
    radius=4, color=(0, 0, 255), thickness=-1)

    angle2 = math.atan2(frame.shape[1]/2 - int(y+(h/2)),
    int(x+(w/2)) - frame.shape[1]/2) * 180.0 / math.pi

    angle2 = 90 - angle2

    #Prints xy coorinates on the frame
    #print(int(x), int(y))
    p1 = [frame.shape[1]/2, frame.shape[0]]
    p2 = [x, y]
    distance = math.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2))

    ans = md/(frame.shape[0]/2)
    distance = ans * distance

    pl = h
    pl = pl*0.6+po*0.4
    po = pl
    lp = 2.3/pl

    c2 = math.sqrt((((int(frame.shape[1]/2)-x) * (lp))*
    ((int(frame.shape[1]/2)-x) * (lp)) + (md*md)))
    c3 = (c2 + distance)/2

    file.write("motion"); file.write(",");
    file.write(str(int(x+(w/2)))); file.write(",");
    file.write(str(int(y+(h/2)))); file.write(","); file.write(str(c3));
    file.write(","); file.write(str(angle2)); file.write(",");
    file.write("0"); file.write("\n");

  square = cv2.rectangle(frame, (int(x), int(y)),
  (int(x) + int(w), int(y) + int(h)), (0, 255, 0), 2)

  circle = cv2.circle(frame, (int(x+w/2), int(y+h/2)),
  int(radius), color=(0, 0, 255), thickness=1)

  cv2.imshow("Motion detection", frame)
  a = int(a) + 1
  i = i + 1

  key = cv2.waitKey(1) & 0xFF

  # if the `q` key is pressed, break from the lop
  if key == ord("q"):
    break

cap.stop() if args.get("video", None) is None else cap.release()
cv2.destroyAllWindows()
```

C Gridmap

The map code is divided into two parts, the calculations part and the showing part. Of course, the code uses inputs from object detection and motion detection to estimate specific objects, but by using that inputs, detecting lines from textfiles gives the updates and where the object is placed estimated from the UAV.

The calculation part is inspired by [54], using the detection data from the previous run by the object detection. Using the estimated values, such as distance to object, angle to object, etc., the calculation is being done to define the longitude and latitude the object is located.

The other part is a printing tool, using pandas [57] and matplotlib [58]. This gives the interface and estimation on where the object are placed.

```
#Verison: 0.1.8
#Author: Rhn15001, Rasmus Hamren
#Applying UAVs to Support the Safety in Autonomous Operated Open Surface Mines

#Calculations is inpired by following:
#C. Veness, Calculate distance, bearing and more between
#latitude/longitude points, accessed:2021-01-04. [Online].
#Available: https://www.movable-type.co.uk/scripts/latlong.html

import numpy as np
import pandas as pd
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math

line_count = 0
line_count2 = 0

#Tracking of DJI drone long/lat from csv
df = pd.read_csv("r2.txt")

#Copy TA15
file_TA15 = open("out_TA15.txt","w")
file_TA15.write("x,y,l\n");

#Copy Person
file_person = open("out_person.txt","w")
file_person.write("x,y,l\n");

#Copy Motion
file_motion = open("out_motion.txt","w")
file_motion.write("x,y,l\n");

#Orginal from objectdetection
out = pd.read_csv("out.txt")
file1 = open("out.txt","r")

#Orginal from motiondetection
out2 = pd.read_csv("outMotion.txt")
file2 = open("outMotion.txt","r")

#Counter line
for line in file1:
    if line != "\n":
        line_count += 1
line_count = line_count - 1

#Counter line
for line in file2:
    if line != "\n":
        line_count2 += 1
line_count2 = line_count2 - 1

#Startpos from drone
#startpos = [16.428125, 59.405547]
startpos = [16.42733, 59.405226]
```

```
#angle compared to north, - to left, + to right
angle = -120
#angle = 0

#Radius of the Earth
R = 6378.1

#FoorLopp of calcilation to calculate long/lat
for x in range(line_count):
    if(out.label[x] == "TA15"):
        brng2 = math.radians(angle+out.angle[x])

        d = out.l[x]*0.001 #Distance in km

        #Current lat from drone point converted to radians
        lat1 = math.radians(startpos[1])
        #Current long from drone point converted to radians
        lon1 = math.radians(startpos[0])

        lat2 = math.asin( math.sin(lat1)*math.cos(d/R) +
        math.cos(lat1)*math.sin(d/R)*math.cos(brng2)) #New long

        lon2 = lon1 + math.atan2(math.sin(brng2)*math.sin(d/R)*math.cos(lat1),
        math.cos(d/R)-math.sin(lat1)*math.sin(lat2)) #New lat

        lat2 = math.degrees(lat2)
        lon2 = math.degrees(lon2)
        #print(lat2, lon2) #Test

        #write to file_TA15
        file_TA15.write(str(lon2)); file_TA15.write(",");
        file_TA15.write(str(lat2)); file_TA15.write(",");
        file_TA15.write(str(d)); file_TA15.write("\n");

file_TA15.close()

for x in range(line_count):
    if(out.label[x] == "person"):
        brng2 = math.radians(angle+out.angle[x])

        d = out.l[x]*0.001 #Distance in km

        #Current lat from drone point converted to radians
        lat1 = math.radians(startpos[1])
        #Current long from drone point converted to radians
        lon1 = math.radians(startpos[0])

        lat2 = math.asin( math.sin(lat1)*math.cos(d/R) +
        math.cos(lat1)*math.sin(d/R)*math.cos(brng2))

        lon2 = lon1 + math.atan2(math.sin(brng2)*math.sin(d/R)*math.cos(lat1),
        math.cos(d/R)-math.sin(lat1)*math.sin(lat2))

        lat2 = math.degrees(lat2)
        lon2 = math.degrees(lon2)
        #print(lat2, lon2)

        file_person.write(str(lon2)); file_person.write(",");
        file_person.write(str(lat2)); file_person.write(",");
        file_person.write(str(d)); file_person.write("\n");

file_person.close()

for x in range(line_count2):
    if(out2.label[x] == "motion"):
        brng2 = math.radians(-angle+-out2.angle[x])

        d = out2.l[x]*0.001 #Distance in km
```

```
        #Current lat from drone point converted to radians
        lat1 = math.radians(startpos[1])
        #Current long from drone point converted to radians
        lon1 = math.radians(startpos[0])

        lat2 = math.asin( math.sin(lat1)*math.cos(d/R) +
        math.cos(lat1)*math.sin(d/R)*math.cos(brng2))

        lon2 = lon1 + math.atan2(math.sin(brng2)*math.sin(d/R)*math.cos(lat1),
        math.cos(d/R)-math.sin(lat1)*math.sin(lat2))

        lat2 = math.degrees(lat2)
        lon2 = math.degrees(lon2)
        #print(lat2, lon2)

        file_motion.write(str(lon2)); file_motion.write(",");
        file_motion.write(str(lat2)); file_motion.write(",");
        file_motion.write(str(d)); file_motion.write("\n");

file_motion.close()

out_TA15 = pd.read_csv("out_TA15.txt")
out_person = pd.read_csv("out_person.txt")
out_motion = pd.read_csv("out_motion.txt")

ruh_m = plt.imread('map.png')
ruh_m2 = plt.imread('ResultTemp.png')


img_cropped = ruh_m[77:141, 57:121, :]

fig, ax = plt.subplots(figsize = (12,9))

ax.scatter((out_TA15.x), (out_TA15.y), zorder=2,
    alpha= 1, c='y', s=300, marker='s')

ax.scatter((out_person.x), (out_person.y), zorder=2,
    alpha= 1, c='g', s=300, marker='s')

ax.scatter(startpos[0], startpos[1], zorder=2,
    alpha= 1, c='r', s=12)

ax.set_title('Plotting Data on Volvo Eskilstuna, ' +
'Green = Person, Yellow = TA15, Red = Position of Drone')

#Sets limits
xmin = 16.42372
xmax = 16.43711
ymin = 59.40307
ymax = 59.40811

xmin1 = df.longitude.min()*0.999995
xmax1 = df.longitude.max()*1.000005
ymin1 = df.latitude.min()*0.999995
ymax1 = df.latitude.max()*1.000005

ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)

BBox = (xmin, xmax, ymin, ymax)

#Grids
ax.set_axisbelow(True)
ax.minorticks_on()
ax.grid(which='major', linestyle='-', linewidth='0.5', color='red')
ax.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

#Background
ax.imshow(ruh_m, zorder=0, extent = BBox, aspect= 'auto')
rectangle = patches.Rectangle((xmin1, ymin1),
```

```
    xmax1-xmin1, ymax1-ymin1, fill=False)

ax.add_patch(rectangle)

plt.plot(xmin,xmax)

#Motion detection on gridmap
fig, ax2 = plt.subplots(figsize = (12,9))

xmin2 = 16.42600
xmax2 = 16.43100
ymin2 = 59.40507
ymax2 = 59.40611

ax2.set_title('Motion detection interfearing')
ax2.scatter((out_motion.x), (out_motion.y), zorder=2, alpha= 1, c='g', s=12)

ax2.set_xlim(xmin2, xmax2)
ax2.set_ylim(ymin2, ymax2)

plt.show()
```