MASTER IN COMPUTER SCIENCE PROFESSIONAL FOCUS IN DATA SCIENCE

Study on passive and active computer vision paradigms related to UAVs movements for electrical pylons analysis

Maitre, Guillaume

Award date:
2020

Awarding institution:
University of Namur

UNIVERSITÉ DE NAMUR

FACULTÉ D'INFORMATIQUE

Study on passive and active computer vision paradigms related to UAVs movements for electrical pylons analysis.

Guillaume Maitre

RUE GRANDGAGNAGE, 21 ● B-5000 NAMUR(BELGIUM)

Acknowledgements

First of all, I would like to thank Prof. E. Tuci for introducing me to the field of robotics, opening up opportunities and supervising this thesis. I would also like to thank Professor A.Lyhne Christensen for welcoming me to the MMMI in Odense, and for her supervision and advice, both on-site and remotely.

Secondly, I would like to thank the University of Namur and the Faculty of Computer Science for having made me grow intellectually in a friendly and attentive space.

Moreover, this dissertation would not have been possible without the help of Qualitics. They have provided a computer and expertise in the field of pylons analysis.

Finally, I would like to thank my family, relatives and friends who supported me, pushed me to go further and advised me throughout my studies. I have a special thanks to Antoine Hubermont for his advice and our enriching discussions on the U-Net model.

Abstract:

This master thesis aims to make a first step in the research of using simulation for UAVs in the domain for computer vision applications. We will compare passive vision and active vision paradigms. Passive vision is a system where the computer takes a decision depending on the actual image it receives. Active vision, on the other hand, manipulates the viewpoint to investigate the environment and retrieve better pieces of information. We were able to achieve a fully running passive vision U-Net model. That model can successfully navigate around a pylon following a path that covers the whole pylon. The active approach was more complicated. It was only possible to do a reflection plus some unsuccessful tests. A time comparison was made. We have also compared the complexity of modifying in case of enterprise use. U-Net showed to be easier to use and change compared to the active vision paradigm by a far hand. Finally, we open the possibilities of future researches in that still yet to exploit domain.

Keywords: Computer vision, Active vision, Passive vision, Autonomous UAV, Intelligent system, Artificial intelligence, U-Net, CTRNN

Résumé:

Ce mémoire vise à faire un premier pas dans la recherche de l'utilisation de la simulation pour les drones dans le domaine des applications de la "computer vision". Nous comparerons également deux paradigmes de vision, respectivement la vision passive et la vision active. La vision passive est un système dans lequel l'ordinateur prend une décision en fonction de l'image réelle qu'il reçoit. La vision active, en revanche, manipule le point de vue pour étudier l'environnement et récupérer de meilleures informations. Nous avons pu réaliser un modèle U-Net de vision passive entièrement fonctionnel qui peut naviguer avec succès autour d'un pylône en suivant un chemin qui couvre l'ensemble du pylône. L'approche active était plus compliquée et n'a donné lieu qu'à une réflexion et à des tests infructueux. Une comparaison a été faite par rapport au temps d'entrainement et à la complexité de modification en cas d'utilisation dans une entreprise. U-Net s'est avéré plus facile à utiliser et à modifier que le paradigme de la vision active de loin. Enfin, nous ouvrons les possibilités de recherches futures dans ce domaine qui n'a pas encore été exploité.

Mots-clefs: Computer vision, Vision active, Vision passive, Drone autonome, System intelligent, Intelligence artificiel, U-Net, CTRNN

Contents

Chapter 1

Introduction

This work has for objective to make a first step in the research of transforming the electrical pylon flaws analysis and detection fully autonomous. The growing grid infrastructure offers a vast market in need of a faster and reliable way of detecting the flaws in their infrastructures. This work aims to look at two approaches that can potentially solve the problem of moving around an electrical pylon safely while only using computer vision.

Currently, power line pylon inspection is a critical and costly task due to the time and effort required. The inspection requires a helicopter, a <4k camera with a professional photographer to take long-range photos from the helicopter, an expert to analyse the photos. This means that electrical companies can be interested in a newer way of analysing power line pylons. For example, by having the possibilities to do multiple power lines at the same time without requiring a lot of human resources or extremely expensive equipment's. Especially for the electrical installations that are in remote access location such as in a forest or mountain. Like so, the use of unmanned aerial vehicles (UAVs) is a potential way to reduce the cost and avoid to use a helicopter, Automate the system to have multiple UAVs doing the same power line or other power lines.

Taking photos can be done either by a pilot using the UAV remote-controller to move the camera and takes the required images. Or it can also be possible to work on an automated system that also controls the camera to take photos automatically. A script could make this outside of the movement-based Artificial Intelligence(AI) or a script embedded in the movement's AI.
Then, the image captured could be analysed by AI onboard by another model looking at errors and then discard images that do not have a sign of problems.

An autonomous UAVs could also detect problems and store each potential damage detected (such as rust, crack, broken insulator) on a pylon in a dedicated database for further analysis. This can helps avoid wasting time looking at pictures of pylons that do not have any problem.
Working on such a complete system could be an extensive research that can take some times. So we will stay on the movement AI to start this extensive research that could be continued for future works by the end of this work. This continuity can be done in collaboration of Qualitics (a Belgian start-up that does

electrical instalment audit) to transform their UAVs fleet into an autonomous power-line pylon analysis system.

In this master thesis, we will attempt to compare a state of the art approach of image segmentation AI system called U-Net to detect the pylon and make all the movement required for the analysis and a system based on the active vision paradigm.

The U-Net model is well known to be capable of good results on object detection and localisation without a lot of segmented training data. This is an interesting model because the domain we are researching is quite unexplored, and the few data-set that could exist are kept for private use. Thanks to Qualitics we can have a nice amount of electrical pylon image that we can make the segmentation on to train our model. We will also train our model on a simulator. This can also be very useful to generate a huge amount of images to train our U-Net model with simulated data. The system using U-Net is going to detect the pylon and from that decides to move around the pylon the best way possible to ensure the smooth execution of the pylon analysis

After, we will create a new model based on active vision principle that will use a genetic algorithm (GA). GA is a method of computer learning inspired by evolution in biology [1]. It searches for one of the best solutions of a function by calculating the fitness of parameters represented by a gene. We mutate the best genes to optimise it. GAs will be used to generate efficient movement strategies according to the input from the camera. The approach will be based on the gaze system and continuous-time recurrent neural network(CTRNN). CTRNN is a fully interconnected artificial neural network where a system of differential equations chose when the nodes are activated. CTRNN is used to solve this complex problem of movement in a heavily constrained three-dimensional movement space without having to rely on a training data set of segmented images.

Also, the final section for the research part will compare the two models over what we were able to achieve this time constraint and the accessibility of each model to perform modification on them.
In terms of results, the U-Net model was straightforward to create and train with a short time of learning thank to the RTX GPU. The downside of U-Net was the creation of the data set with segmented images. Here, it is avoided with the simulation generating segmented images. The intelligent fly controller receiving information from our U-Net model was able to perform a full analysis of a simulated electrical pylon in a mean time of 5.13 minutes.

The CTRNN active vision model, on the other hand, ended up being way too long to train and optimise. It has resulted in the inability to make a working model able to analyse a whole pylon. We decided to stop the training after multiple failures and the learning speed being of more than twenty-four hours per test. The main advantage here is the capabilities to train directly on the simulated scenarios and the fact that we do not need to segment the images.

The comparison has shown that the U-Net model is a better option for enterprises. This is due to its simplicity and ease to modify and adapt to new constraints. The CTRNN is still a domain to explore that could deliver better results for the reality gap. But, it requires a lot of time of training and is not the most suitable option in a domain that could request a frequent change in constraints.

For the reality gap, we can only talk about the U-Net model that was testable. The model was showing difficulties to adapt to real pylon detection of multiple forms, for some never seen. We lost around twenty per cent of the F1-score between the results of the U-Net model on simulated pylons and real pylons.

Finally, we will explain what can be done for further research on the two computer vision paradigms to make the autonomous UAVs a robust solution to do the whole analysis of a pylon. We will also discuss the other possibilities that can be explored with more time, especially for the active vision paradigm that is open for a lot more works.

Chapter 2

State of the art

In this chapter, we will list all the technologies related to UAVs from the actual history of how they ended up as we know them right now. We will also describe each module, such as sensors, that can be put on them to make them more autonomous and aware of their environment. This list of technologies is created thanks to a large literature review made for this research.

By the end of this chapter, you will have the knowledge necessary to understand all the decisions made in this master thesis and the theory behind them.

2.1 UAV

In this section, we will explain what the UAVs are. This includes everything related to them, such as the definition, the history and what is composing a UAV.

2.1.1 Definition

In this thesis, drones or UAVs are air crafts without an onboard human pilot. Most often, the control of the UAV is done through remote control and a human operator on the ground. Sometimes, a UAV can be controlled by a computer (either onboard or outboard).

The size and shape of a UAV can vary depending on what the UAV will be used. From around 10×2.5 cm for a Black Hornet Nano [2] for example to about 39.9 meters wingspan for a Global Hawk military UAV [3](both being controlled by human). The shape can be categorised as follow:

Type	Number of propellers	Example
Rotor UAVs	Single-rotor UAVs	Helicopter UAV
	Multi-rotor UAVs	Civilian's quad-copter UAV
Fixed-wing UAVs	Single motor	Delta wing UAV for agricultural surveillance
	Multi-motor	Model making of real plane
	Jet engine	Military UAV for long-distance operation

2.1.2 Controlling UAVs in history

UAVs have a vast history of military usage since 1849. Those were proper tools for training, attack and recon activities.

UAVs have made their first apparition July of 1849 as balloons carrying fire-bombs. They were moving with the wind in the desired orientation. Austrians used them to bomb Venice during the 1849 siege of Venice. But, this technique failed due to changing wind condition. UAVs of that kind were rapidly forgotten due to the lack of control and efficiency. [4][5]

During World War 1 and World War 2, UAVs had begun to be reused, first for training, and then for attacking (German V1 flying bomb). For both uses, most of them were mock plane with no remote control and straight flying aircraft that would either be destroyed in flight (for training) or explode on landing (for attacking). A large amount of them was calibrated before the fly to stay in the air for a precise time before a mechanism would push them to dive on the ground. Each nation and UAV using a different approach for how to perform that action.[4][5]

After World War 2, remote controlling begun to be more common, with wireless technologies getting more robust and reliable. It's also at that moment that richer aircraft-model enthusiast got into remote controlling and UAV.[4][5]

Then in 1975, the Israelis have shown a UAV capable of live video streaming at long range to helps the pilot manoeuvring the UAV. Which is still the most commonly used way to fly a UAV around.[4][5]

From 2000 and onward, thanks to the miniaturisation and prices dropping for complex electronics system, UAV were more accessible to civilians. Which led to more common usage in agricultural, recreational, aerial photography and scientific domains. At the same times, researches started on autonomous UAV using either distance sensors, cameras, GPS, barometer, gyroscope or all of them at the same time for various utilities.[5]

2.1.3 Civilians UAV components

Other than the frame, motors, batteries, antenna that you would expect on UAV, the more advanced one like we will be using further have some intersecting electronic devices to help without having to program them. Those components have been explained to us by Qualitics.

- Flight Controller (FC), the core of the UAV, it translates the instructions from the pilot/AI and the data from sensors to control the power of each motor. The FC helps the UAV to fly as requested by the pilot.
- Electronic Speed Control (ESC) is managed by the FC to regulate the electrical power supplied to one motor. It is requested to have one for each motor.
- Inertial Measurement Unit (IMU), this is a chip with an accelerometer, a gyroscope and other tools that can help to determine the current moving state of the UAV. If the UAV is not moving and commands are not supplied, the IMU will send data to the FC to keep the UAV hover at the same position.
- GPS, it is helping the UAV to know its world position, and also retain his take-off position so it can get back to this starting position autonomously in case of trouble.
- Sensor, LiDAR, radar, sonar, camera, those devices can generate multiple inputs for the security of the UAV. It can be communicated to the FC and the pilot of the UAV. Those data can be interpreted by both the human and computer to help them avoid an obstacle or to follow a detected object.

With all those components associate together, we can obtain a well designed and very secure UAV capable of performing a large range of actions depending on the pilot and the task to be done. Each year, the miniaturisation of sensors and computing power bring more and more capabilities to the UAV to perform new complex tasks that were previously not achievable.

2.1.4 Conclusion on UAVs

The UAVs are becoming more and more capable. They are now getting more attention in the industrial sector and from artists such as photographers or even painters. This mainly military technology is growing in the public sector, with a wide range of use from recreational hobby to highly complex tasks execution such as search and rescue in dangerous places to navigate.

This technology also went from a high difficulty to control to a more semi-autonomous flying machine capable of taking decisions over the pilot. The AIs can avoid objects, stabilise itself and even capable of returning to their starting point autonomously in case the signal between the pilot and the UAV is lost.

UAVs have changed the way some photographers can take photos by giving a whole new level of perspective to work on. Artists do not have to pay a huge amount of money to use them. We can now see a lot of shooting of landscape and city skyline by professional that would have required the use of a plane or a helicopter ride to be done before.[6]

Those UAVs are becoming a really useful resource to use to accomplish tasks in remote locations or requiring to fly over particular areas that are not typically easy to access (indoor or outdoor). This rapidly growing piece of technology has still a lot of abilities that are yet to be discovered.

2.2 LiDAR & human, the current approach of Qualitics

Qualitics use LiDAR for checking the distance between the UAV and the pylon during an analysis. But, the LiDAR does fail to be used in a situation where the UAV and the object being analysed are separated from more than 10 meters. This failure is due to the quality of the LiDAR and the object being often too thin. This can be explained by the LiDAR being unable to echo the light on the object continuously leading to instabilities (getting noises on LiDAR segments, detecting further objects and detecting nothing) in the detection. We have tested several configurations with the LiDAR used by Qualitics for cable detection to optimise and augment the distance at which we detect the cable with enough stability. Those tests, realised on real cables, were able to get enough stability up to 6.5 meters, over this, the number of instabilities would grow exponentially until the complete loss of the cable detection.

As for the pylon analysis, the pilot does everything. He is just helped to keep the right distance between the pylon and the UAV thanks to the LiDAR.

The advantages of this approach are:

- Extremely safe to use, LiDAR is an exact way of calculating the distance every 0.04 second without needing much post-processing. This leads to a very smooth control of the UAV to life adjusts its position according to the eight projected segments by the LiDAR.
- The pilot can make smalls adjustments if he thinks that the photos taken during the analysis are not correct (ISO, brightness).

The problems with this approach are:

- Problems related to stability issues over longer distances between the pylon and the UAV.
- It requires the pilot to be near the pylon at a correct distance to judge how the inspection is going. Leading to need to move with a car between each pylon. It can sometimes be difficult when accessing some pylons in the middle of a forest, a field or any remote locations that are not close of a road.
- The UAV can do autonomous electric line following, but then, the pilot needs to land the UAV, to start the procedure of pylon analysis. Which lead to a time lost and battery lost.

Even if this method is correct, it is showing some cons that cannot be avoided. Also, we know that we are already pushing the UAV's LiDAR to its maximum, so the only way we could make it better would be to change to another LiDAR with better performance. But this change would result in more weight on the UAV, more power consumption and reduce the flying time.

2.3 Other classic approaches

2.3.1 Flying with Path planning

Most of the other companies working in autonomous UAVs for electrical infrastructures inspection use a less complicated approach that asks the pilot to prepare the flying path before going to the mission. This solution is similar to [7]. This approach is the easiest one to create but also results in many troubles. Because the UAV is just able to perform basic obstacle avoidance while following the traced path on a computer, the UAV is just going straight to high-interest points marked on the map and proceed to take photos requested depending on the parameters without adjustments.

If everything is correct and well placed without errors of the UAV positioning sensors, this technique would clearly be the best. Still, unfortunately, even high precision GPS tend to have difficulties when getting close to the high voltage cable, sometimes creating a small error in positioning.

Those small errors can make a mission a total failure, by having photos out of point, incorrect centring or even photos without the point of interest requested visible. This can lead to the remake of a mission from the beginning to redo all those failed photos. Most of the times, this request some days to prepare and get the correct weather. Since UAVs are not able to fly in some complex weather conditions.

Those hazardous comportment's make this technique not enough reliable for large scale inspection, especially in term of reliability and quality of photos for post-analysis.

2.3.2 Fully sensors-driven system

This project [8] shows a sensors way to interact with the world for the UAVs. This software uses proximity sensors and distance sensors of each type possible to verify if the UAV will not make any lousy movements that could result in a crash. This software is meant to be used in an inside loop with an AI or a human. The human or AI will decide on the movement, and the software will start to overwrite the command that can cause a crash or be too dangerous. For example, if the UAV is getting to close to the cable. The software will make the UAV slightly goes on the opposite way. This piece of software is developed to be as reusable as possible.

This software would be a nice add-on for a version of our U-Net model. It would add the capability of objects avoidance without requiring much work from our side.

2.3.3 Single camera environment perception

In [9], they put in place a first attempt of autonomous flight in a forest without aids from the path following system or any additional sensors or GPS. The approach here is to identify clear flight areas and predict the behaviour of the UAV in this unknown environment. To do so, the model as to extract from the image by a first model the NED (North, East, Down) position and the orientation quaternions with the second model. This multitasks Regression-based learning takes advantage of having different neural networks for each task.

When they are all combined, they are capable of showing promising results in unseen environments. This method is suitable for search and rescue missions since it is having a very aggressive exploratory behaviour and do very well in term of obstacle avoidance.

This technique can be interesting to adapt in term of navigating around the pylon without needing any distance sensor and thus reducing the weight and battery usage for our use case. One important thing is to consider is the fact that we need to decide the precise objective of analysing pylon compared to this type of problems where you just have one point to go to.

2.3.4 Using stereo-vision

Stereo-vision and visual optometry is also a great way to achieve a decent obstacle avoidance and taking explicit photos at the same time [10]. Still, it requires to have precise intrinsic parameters and calibration before obtaining favourable results. It also has a cost in term of autonomy because it requires two cameras to work. Those stereo-vision systems give a cloud of points created by two cameras. When those two images are interpolated together, the system is able to tell the distance of the pixels in the images. As an output, most cameras give a heatmap or a grayscale image with darker points being closer and lighter point being further. The system then enables the UAV to navigate while avoiding the obstacle easily. The main problem with this technique is the need for precise parameters to obtain correct data. This is creating considerable complexity and making the UAV more complicated and expensive.

This technique will not be used in this master thesis, but it is worth to be aware of it. For example, this could be useful for other tasks or if other techniques fail to provide correct results, stereo-visionè could be an alternative.

2.4 Machine learning techniques

In this section, we will go through some machine learning methods that can be interesting to use.

Right now, there are already algorithms that proceed to detect a pylon with boosted decision trees joined together to make a strong classifier to localise a pylon in a variety of background [11]. They use Aggregate Channel Features(ACF) to get a different channel of the RGB image as input for the decision trees and train the ACF with AdaBoost optimiser to help with the training time. All the decision trees are made to detect a particular feature in the image. By doing so it makes each one of them strong at one object detection. When they are combined, they detect all the part of a pylon. They do end up with very good accuracy and with a false-negative rate of 2%.

Some deep-learning approaches were tested in [12], where 4 different types of models (SSD, F R-CNN, YOLO, R-FCN) were tested to compare their efficiencies at detecting different kinds of insulators on a pylon. The results of this paper show that R-FCN (Region-based Fully Convolutional Network) and SSD (Single Shot MultiBox Detector) tend to have better results than the two other. This type of models could also be interesting for future works. In example, the detection of different parts of the pylon for advanced analysis could be made following this paper.

No U-Net implementation for this usage as been tested even if this deep neural network performs very well [13] on classification problems. Usually, where little to no data are existing in terms of annotation and masked image for segmentation. It is also a model that can learn quite fast on modern consumer-grade GPU. This model will be considered since it can be trained with a very low number of images if needed. This is a very important aspect because the availability of a segmented data set of pylons is non-existent. It is also time-consuming to create a segmented data-set.

In term of computer vision, Genetic Algorithms (GA) are not frequently used. Some GAs appeared for hyper-parameters optimisation [14].

Also, there are now libraries that make GA able to run on GPU to parallelise the work and make it faster to learn [15]. They are using folders and multiple genomes comparison at the same time. Simultaneously, they update the folders which contain the model by removing the eliminated ones and keeping the winner and its offspring. If the simulator enables to run multiple instances at the same time, this technique will be very useful to accelerate the training time of our GA system.

Finally, there is also an approach to neuro-evolution called NEAT (Neural Networks through Augmenting Topologies) [16] that can be useful after further reading. But this is not in our time frame for this master thesis.

2.5 Reinforcement Learning (RL) and Evolutionary Learning (EL) for optimisation.

For full autonomous movements control that can be represented by a Markov decision process [17], it can be interesting to use algorithms that can help to dynamically optimise the decision process such as the weight of the branch between different state. For example, Reinforcement Learning [18] is one of the most popular ways of doing that in the artificial intelligence domain. RL will repeat the same type of mission to learn from it with a system of reward per good tasks done. Depending on different chosen parameters, a function can be created to calculate the amount of reward given to the algorithm that just performed the mission. In example, time, energy, collision, area constraints can be applied to the reward function to optimise the algorithm at its finest. As for the cycle of learning in RL, the robot will perform an action, then the environment of where the robot is will be analysed and interpreted. Depending on this environment, a bonus or a penalty will be given to the total reward. After that, the next action is assigned and done. At the end of a mission, the total reward will tell how well the robot performed. It will then make some back-propagation on the model weight to do the optimisation of parameters. Then, it will test the new form and see if it works better. After a moment, RL should have explored the map of possible parameters by optimising at each step. RL is also used in game theory and operational research since it shows excellent results in optimisation.

As for Evolutionary Learning [19], like RL, it aims to optimise a solution to solve a problem, but this time, it is a population-based meta-heuristic optimisation algorithm that will research one of the optimised solutions. Depending on the genes pool generated, the algorithm will start to compare the strength of each individual depending on a fitness function. Then it will make a selection of the best one and use a mechanism such as mutation, reproduction, recombination to generate a new-gene pool to find the new best fit for the problem. It can go for a long time depending on the need and complexity. Every run can keep x number of the best solutions of the previous iteration.

To know which one between RL and EL is to use for our case, we can look at [9]. This paper explains to us that RL shows extremely good result in all the Markov decision process task and explain why it is used so much. Since a lot of problems can be transformed into a potential Markov chain, but when it comes to non-Markov chain task, for example, multitask decision making, the EL paradigm is performing better in term of training and final solution reaching. This gives us the right pieces of information of which type of algorithm we will need to choose to train our active vision version of the pylon analysis.

2.6 U-Net

U-net [13] is a powerful deep neural network that first does a convolution of the image it gets in input for object detection. This convolution is used to extract features. Then, it starts a deconvolution path to make a precise positioning of the detected object by reconstructing the image. It is a great tool that performs well in the detection of malicious cells in the human body. U-net has the great advantage of being able to be trained on a reduced data-set. It can also run in less than a second per image on a 2015 GPU (NVIDIA GTX Titan 6Gb). The performance on a 2019 RTX GPU (NVIDIA RTX 2080 8Gb) using the TPU's (Tensor Processing Unit) in it are better. If we use an implementation of U-Net using the library Keras [20] and TensorFlow [21]. The training on a one thousand images data set can be done in less than forty minutes.

U-net is going to be used to make the prototype of a pylon detection and used as a benchmark for other implementations. It aims to give the position of each part of the pylon on the image. Then, it can generate action depending on the positions of them.

2.6.1 Moving with U-Net

U-Net can be used for movement in the passive vision paradigm. We detect the pylon with our neural network, and we use that prediction to take decisions related to it to make the best movement possible. This requires to create a set of rules to interpret the image and take a movement decision.

2.6.2 Convolution neural network for UAVs movement

In recent year, a new way of looking at the control of robot and UAVs as been researched. The aim is to get a system that can be trained in simulation and also be used in real life in various environments. There is variability in terrains such as going from forest to field environment and vice versa that can be met when using UAVs. For example, the use of UAVs in search and rescue mission is something fundamental and to be looked at. Autonomous UAVs tends to be well appreciated for this kind of task due to their capacity of covering a lot of space without requiring to have a human piloting them.

Online Deep Reinforcement Learning [22], as showed promising results of being able to navigate through multiple unseen environments that can be dangerous of access from humans. In their research, they developed a model in AirSim that has to find one of the best ways to get to a certain point on the map without having a GPS and can only use camera and accelerometer to understand its position in the world from the starting point. They believe that having a simple approach can help the model to get better adaptability than the classic Q-learning system and Long Short-term memory. Since those had shown difficulties to overcome unseen place and scenarios. The trained model in the paper, Extended Double Deep Q-Network (EDDQN) show nice results of adaptability and capacity to be used on different UAVs, either specialised or commercial one on many forms of terrain and weather condition. EDDQN use a double inputs model, one that takes the 84*84pixels grayscale image to generate a vector of it thank to multiple convolutions. In the same manner as the convolution phase of the U-Net model. Then it concatenates this vector with the second network output from the local map fed to obtain the four outputs that will decide the action taken. The reinforcement learning is then applied during the learning part, and they do use a reward function depending on going to already visited places, getting to the objective and time. We will follow the advice of staying as simple as possible by only using the basic U-Net model without much modification to its way of working. It is also important for us to stay simple since we have to also work on the active vision paradigm.

2.7 Active vision.

2.7.1 Definition and explanation.

To fully explain the active vision, *"Adaptive active vision"* [23] is a fascinating PhD thesis to read. To summaries this paradigm, we need to go back to basic biology, as to how living creatures work and how we are learning to move in the world. [24] helps to understand the facts behind how we are learning to interact with the world by a passive and active way. Our vision and movement for decision making are bounded together and it is very important. As the paper shows, multiple pair of kittens are put into a carousel to learn how to move and interact. Half the kittens are learning actively to move around the carousel and the other half is learning in a passive way by being stuck and moved around the carousel by the active learning kitten. Then, they have to go through different experiments such as paw opening reflex, chose the right ascending surface (swallow or direct), eyes reaction to incoming objects.

The results of this experiment show that the active learning kitten can perform every test easily and correctly and the passive learning kitten as a lot of difficulties to perform well into the world when facing the real world.

Getting back to [23], vision can be divided by 3 part. First, there is the gaze stabilisation such as head movement to keep your gaze on a specific object and compensate for any movement of the body. The second part, especially with primates, is the object following gaze which is smooth, like looking at a car passing next to us, our brain predicts the future position of the looked object. That's also why some unpredictable objects are often harder to follow smoothly since our brain will make false predictions. The last part is what [23] calls intentional shaking, which is the fact, for example, to move your eyes around a new room and pointing without realising different points of interest. Human most often does not realise that they are doing eyes movement or controlling them when they are not focused especially on that. A lot of UX expert, for example, are looking into this field of vision when developing a new interface to avoid or catch the gaze of the user on something.

So, for active computer vision [25], we can consider an observer as active when it can control itself over different geometric parameters of its sensor or gaze. Those controls have the capability to manipulate the sensor to reduce some constraints related to the observer phenomena, like turning around, changing the angle of the sensor, change its position and even the sensor parameters. [25] shows results that active vision is able to solve fundamental computer vision problem way more efficiently. In short, some issues which are non-linear for passive vision can end up linear with the active vision paradigm, like depth computation and shaping contour.

2.7.2 How does it work with autonomous vehicles?

The active vision or at least the use of evolutionary learning such has genetic algorithms has been used to control UAVs for navigation. In particular, when you need to navigate to a position or an area in the scenario of search and rescue, see [22]. In [25], they explain a model in chapter 5 that use active vision to control a car on a simulator that needs to stay on the road and avoid obstacles that are on the road. To do so, the model uses two sub-models, one for the car control and the other for the gaze control. The two models are fed with the same input, which is the image in greyscale and four parameters, the speed, the three others are relative to the gaze and camera controller.

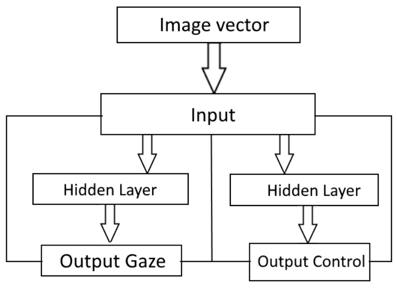


Figure 2.1: Single input, Multiple neural network system

This chapter also helps us by giving some information about the type of network to put in place. They have tested in a first time feed-forward neural network but without success in term of control. They then tested Continuous Time Recurrent Neural Network (CTRNN) for the controller. Thanks to this type of network, they were able to keep a certain memory of previous states without having to use long short term memory neural network. CTRNN is also a type of artificial neural network characterised by the use of a system of differential equations for the node's activation.

As for output, the first model for the control of the car is straightforward and have two outputs, the acceleration can be positive and negative and the steering [-1, 1]

The second model extracts three outputs. The first two are for the exact future coordinate of the gaze on the image. The third one is the features extraction method. Like either, take the average greyscale or the centre.

In this dissertation, we will try to achieve the active vision paradigm by using CTRNN since this seems to work for a car quite well. The main problem here will be to create one and try to find out if it works on a UAV vehicle that can move in multiple ways compared to a car. The complexity of movement that UAV are suffering from might be causing this model to struggle to work well.

2.8 Reality Gap

The reality gap effect [26] in robotic is an effect caused by the divergence between a simulated environment and a real environment due to the lack of realism. It will be interesting to study how our working model can perform in real life pylon, while only being trained on simulation. This is a critical paradigm to take into account when you are working on production AI trained on simulation.

In the paper, the author talks about abstraction around the sensors and actuators to facilitate the development of robots in simulations that can directly be used in real life, by reducing the behaviours and leaving the control to an abstracted fly controller. For example, with a fly controller that possess all the information related to the UAVs, we can just send a simpler order to it. It would be like "go up one meter" and makes us able to avoid sending complete information to each motor and everything related to movements. Abstraction helps to reduce the gap between reality and simulation with tools that can simplify the communication between the AI and the body of the robot.

The gap can come from the visual effect and the fidelity of the simulation. Usually, the simulation does not have good visual effects and good quality graphics. It can grow the reality gap in computer vision systems. Nowadays, graphics are getting a huge improvement. But, it equipped to be equipped with a good GPU. However, there is still place for improvement over the visual and quality of three-dimensional models. We need to keep in mind that more we want to look like the reality, the more resource it will consume and thus take resources that could be used for the AI model.

Also, the gap can come from the physic engine deployed in the simulation. The physic is crucial for robot simulation because it can alter the way the robot will perform in the real world. Having a lousy physics would involve trouble in power sent to the UAV for a particular movement. In the case of RL or EL, it is essential to have a physics as close as possible to reality. Since everything will be optimised to the simulation physic and not the real world. It is important to watch and test how well the model performs on real-life before sending the UAV with the trained model in production.

After that, some events can occur in the real world that does not happen in simulation or are harder to simulate. For example, dynamic obstacles, it is not complicated to develop static obstacles such a fence, wall, trees in a simulation. But, some objects that have the ability to move can be harder to generate in simulation. For example, birds, leaves and branches from trees moving to the wind. Those can be very dangerous for a UAV and change drastically the way the UAV interacts with the real world.

2.9 Image processing

Image processing will be an important task in the master thesis to extract information from what we receive in the U-Net version of this project. From the image preparation to the post prediction analysis, multiple stages, pre and post-processing, had to be performed to get all the requested results to make a decision.
In this section, we will explain some techniques that will be used during the master thesis to extract pieces of information from the 2500x2500 pixels image taken by the UAV.

2.9.1 Pre-processing

For this part, we take the full RGB classic image and start by making a simple resizing of the image from 2500*2500 pixels to a more usable 256*256 pixels size, that can be analysed faster and still have enough data to distinguish the pylon. After that, all the pixels are divided by 255, so we now have a float array. Dividing the array by 255 makes the RGB value be between zero and one. By doing so, we can now make our image go trough model for prediction.

2.9.2 Post-processing

After the prediction, the masks of the image are extracted, and a clarification of the image is done. So depending on the probability of a pixel, we either gave it the value one or zero. If the pixel as a probability of more than .75, it's changed to a one else it's zero. Like that, we end up with three arrays corresponding to the three objects that we want to detect, with either 1 or 0 as pixels value.

Now that the prediction is fully processed, we can start to create a coordinate box of our detected pylon. To do so, multiple techniques are used to do so. To detect the left and right edge of our pylon, we combine two algorithms (Canny Edge Detection and Hough Line detection) to extract those coordinates. As for the top and bottom part of the pylon, we simply extract the lowest and highest point of the detected Upper part of the pylon.

2.9.3 Canny Edge Detection

We will use Canny Edge Detection to produce edges extraction. This algorithm will be useful to help the second algorithm (Probabilistic Hough transforms for lines) to extract the lines needed to create a position box of the pylon. By doing so, we will be able to centre the UAV by positioning the box in the centre of the image.
Canny Edge Detection[27] is a popular and robust way to extract the edge of

an image and goes through multiple stages to do so.

First, it will make a noises reduction by using a 5x5 Gaussian filter. This will smooth the image and remove some noisy pixels that could have been faultily detected by our model.

Then the algorithm goes to find the intensity gradient of the image. It does so by filtering the smoothed image with a Sobel Kernel in both horizontal and vertical direction. In that way, we can have two images (G_x) for the horizontal direction and (G_y) for the vertical direction. With both images, we can extract the edge gradient and direction for all the pixels.

$$EdgesGradiant(G) = \sqrt{G\binom{2}{x} + G\binom{2}{y}}$$

$$Angle(\Theta) = tan^{-1}(\frac{G_y}{G_x})$$

Then, a non-maximum suppression is done; in this part, the algorithm cleans the image only to leave all the edge of the image. To do so, it checks for every pixel if it's a local maximum compared to its neighbours in the direction of the gradient calculated in the second part. This removes all the pixels that are not an edge in the image.

The last stage decides if the detected edges are real or not. To do so, two threshold values are used to determine it. Depending on the selected values, the algorithm decides if an edge is worth keeping depending on its intensity gradient.
Three cases can be considered:

- The intensity gradient of the edge is higher than the maximum value of the threshold. The edge is a "sure edge" and kept.
- The intensity gradient of the edge is between the two threshold value, but one part of it is higher than the maximum threshold value. Then this edge is also considered as a "sure edge".
- If the edge is always under the maximum threshold value, then, it's removed as it's not an edge.

Those manipulations should make the image remove the remaining noisy pixels that could have survived if the minimum length of edge value is not to small.

2.9.4 Probabilistic Hough transform for lines

Probabilistic Hough transforms for lines will be used in the master thesis to extract all existing vertical lines. This algorithm will use the image processed by the Canny Edge Detection algorithm to detect the lines. This will help us remove the left and right side position of the pylon.

Hough transform [28] is a well-known algorithm for natural shape (such as lines, circles and ellipses) detection in digital images for objects detection. Those standard shapes can fully describe an object. Due to imperfection in either the image or the edge detector, some shape might have some basic deviation or noises that make them harder to detect. So Hough algorithm uses a grouping of possible candidates and makes a voting system to extract which one is the right object from all the available possibilities.

The probabilistic version of the Hough transform algorithm will not go through all the possible points but instead, just take different random subsets of points that are sufficient for line detection. This is making the line detection as good as the classic approach while also taking less computer power.

2.10 Three-dimensional space

UAVs are flying objects that use spacial rotation and the speed of their rotors to move in the air. As such, this section will introduce you to two different ways to interpret the three-dimensional space for movements that are used in the simulation AirSim.

2.10.1 Euler angles

Euler angles are going to be the system we use to communicate with our UAV. They have the advantage of being simple and require less input to work. The AirSim APIs does use already Euler angle for half of its APIs related to movement.

Euler angles are the most comment way and easiest way for humans to interpret angle in the space. It is made out of three specific angles named Yaw, Pitch and Roll. An angle is characterised by a radiant number representing the value of the object position compared to its spacial origin position. AirSim uses those value such as comprise in the interval of $[-\pi, \pi]$ which can be interpreted in degree as $[-180, 180]$. Whenever the pitch is positive, it means that the nose is facing up and vice versa. A positive roll means that we are rolling on the right side (right side facing down) and when we have positive yaw, we rotate around the centre axis of the object on the right.
Fig:2.2 is an excellent example to understand how Euler angles works.

Euler angles are straightforward to understand and to manipulate in term of human understanding, but mathematically, they are limited by the phenomenon called "gimbal lock". This phenomenon happens when the pitch axis gets too close to +/- $\frac{\pi}{2}$. This result in the two other angles to not be correctly calculated. Therefore, it is impossible to determine which one of the two is changing, resulting in errors than can block the gimbal.

2.10.2 Quaternions

Quaternions are the second type of system to describe the orientation and direction of an object in AirSim. We will encounter quaternion in AirSim when retrieving outputs from our accelerometer and positioning system.

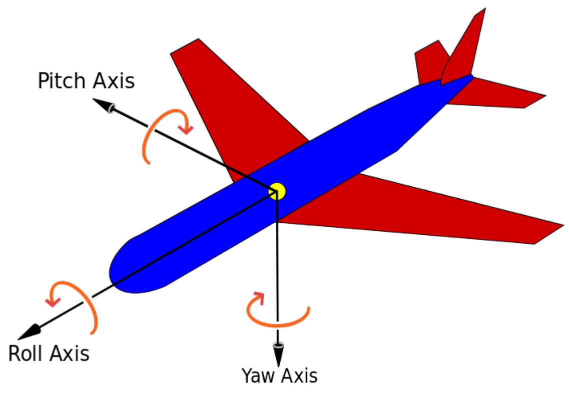


Figure 2.2: Yaw, Pitch, Roll angle interpretation

In [29], we have an in-depth explanation of what a quaternion is, How to use it and finally how to transform it in a more readable format such as Euler angles.

In short, we could describe it as a quadruple of real numbers. The first three numbers (x, y, z) can be interpreted as a vector. Those three numbers are considered the imaginary part by the mathematician Hamilton. w is the scalar of the quaternion. It is the purely real part. The main particularity of these complex numbers is the loss of the commutativity of the multiplication.

In this master thesis, we are interested in the rotation quaternion that can be represented by its imaginary part as a vector. That vector has its origin at the origin of the Euler's Cartesian plane. If we take into account the Euler's rotation theorem, any rotation in a three-dimensional space can be interpreted as a given angle θ around the Euler axis. The vector $\vec{u}$ can represent the direction. This means that our vector can represent a rotation as our imaginary part and θ as our w.

Like so, we can deduce the radiant angles in the Euler plane by doing those three functions:

- Yaw: $\arctan(\frac{2(wx+yz)}{w^2-x^2-y^2+z^2})$
- Pitch: $-\arcsin(2\,(xz-wy))$
- Roll: $\arctan(\frac{2(wz+ya)}{w^2+x^2-y^2-z^2})$

2.11 Ways of moving

In this section, we will introduce you to what Qualitics uses to move the UAV during the analysis of a pylon. We will then proceed to explain our way of thinking on how our IA should move to analyse the pylon completely without requesting the need of a human to interact or move the UAV to some specific places.

2.11.1 The actual way

Fig 2.3 is showing the way Qualitics is doing this right now. Qualitics is using a LiDAR that estimates the distance to the pylon and keeps it to a certain minimum. The pilot manually does the rest, there are 3 phases, the first one calibrating the UAV and taking front-facing images, the second gets the top-down photos and the last one the bottom-up photos.

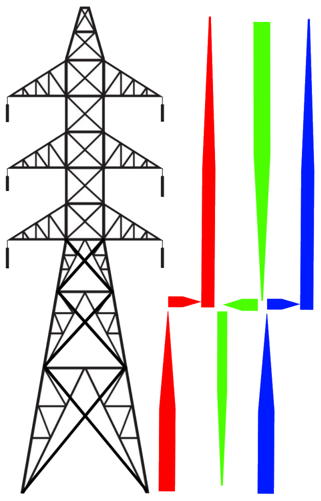


Figure 2.3: The actual way of doing the pylon analysis

We can interpret fig 2.3 with the arrow representing the direction of the UAV. Here is an explanation:

1. The red arrow, this is the first step of the analysis. The UAV is on the ground, approximately four meters away of the pylon and the camera gimbal is locked at 0 degrees of pitch. The UAV will take-off and ascend following the pylon. The UAV will keep the same distance until it reaches a certain level below the first insulator. I will then proceed to move away from the pylon until they are approximately four meters away from the first cable. The UAV will then continue to do its ascending until it reaches the top of the pylon. this step is used to calibrate the UAV for the rest of the inspection.

2. The green arrow is the second step of the analysis. The UAV will descend vertically at the same security distance of four meters and follow the same path taken by the red arrow. The difference here is the pitch angle of the gimbal will be locked at -30 degree to get the top-down angle images from the pylon.

3. The blue arrow is the last step. Like the other two, it will follow the same path in the ascending way. This time, the pitch of the gimbal will be locked at $+30$ degree to get bottom-up images of the pylon. After that third step, the analysis of the side of the pylon is done, the UAV can land while keeping the security distance with the pylon.

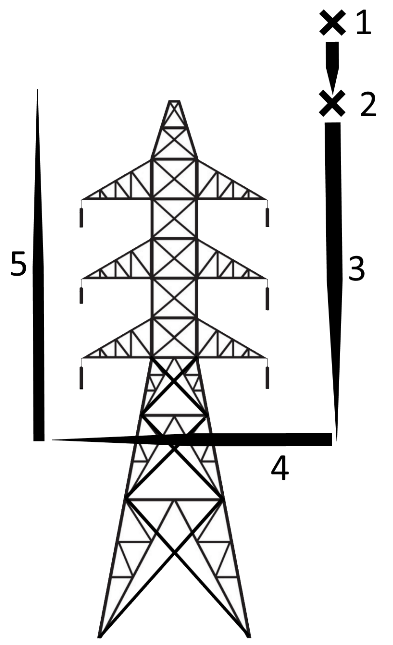


Figure 2.4: A supposed way of doing the full analysis in one fly.

4. this technique has to be repeated for the second side face. Since going between two cable is not compliant to safety measure due to interference caused by the cable to radio signals. So only two faces can be analysed.

2.11.2 How we would like to proceed with the analysis

Fig 2.4 is about a new possible way of moving. Here, we can stop descending when we are at the top of the base structure and go around the pylon. This approach can be interesting in the case of the use of the electric line following system. We would directly start from the top of the pylon where the UAV would normally arrive. By directly starting from the line following system, UAV would take less time and battery loss since it would be able to directly start from a better positioning to actually avoid the first step of the previous technique. This would also require an autonomous calibration system to avoid the first step.

The interpretation of fig 2.4 is a follow:

1. This is the point where we switch from the line following system to the pylon analysis system. This can be done by using a pylon detection model in the line following interface and make it stop when the pylon ends up being detected in the middle of the image resulting in the UAV stopping. It would stabilise itself and start to go a bit lower to adjust the UAV to the height of the pylon before starting the analysis.

2. This is the starting point of the pylon analysis. It's important to have a very specific starting point related to a precise point of the pylon to calibrate the rest. So, when an image is taken with a flaw, we can precisely tell where the flaw is by referring to the movements that the UAV have

done during the analysis. It as to be calculated from a specific point since most sensors are subject to noises when getting to close of the high tension cable leading to bad flaws positioning after the analysis.

3. This is the descending part of the pylon analysis movement. During this period, the UAV uses the U-Net model to estimate the distance and the position of the pylon to centre himself as best as possible. The UAV will descend to a certain level, approximate to four meters lower than the upper part of the pylon, to stop its descending and stabilise itself before going to the next step.
4. This is the most delicate phase of the pylon analysis. The UAV will drift on the left while pitching forward to advance. It will advance until it reaches the middle of the front face of the pylon. Once it is there, the UAV will yaw to look directly at the pylon, proceed to centre himself and adjust the height. During that time, the gimbals can make take multiple photos of the front face of the pylon by changing its pitch value. Then, it would start to drift again on the left to finish this task. After drifting on the left, the UAV will centre itself to the pylon and prepare for the next step.
5. This is the last step of the first half of the loop. We ascend on the other side of the pylon by always keeping it to the centre. The UAV will always adjust the distance to the pylon. The UAV will continue until it reaches the top of the dome of the pylon.
6. After all those steps, the UAV will do the step $3, 4, 5$ again. By doing so, we can achieve a complete scan of the pylon. Those are the same steps so we can just program rules for the points above and not for the rest of the pylon.

Thanks to this manner, we can make a scan as complete as the actual one, without needing to land and take-off multiple times. This solution could also be adapted in the case of the UAV not being able to go too low due to obstacle or potential dangers that could result in a crash. For example, if there is a forest under the UAV that block its analysis. The UAV could stop to descend and start to ascend to go by the top of the pylon to performs the switch.

Chapter 3

Technical information

For this master thesis, we need to get a list of all the constraints that are required to specify and train our autonomous UAV properly. It is important to take into account hose constraints, so our model fits the laws and security rules. First, to know the liberty of freedom we get access to when moving around the pylon. It is also helping us shape the core of our solution and its feasibility. After that, the pylon itself has a structure somewhat complex to analyse and navigate around. The fact that we are flying around high voltage infrastructure can cause harm to some sensors and generate noises and electromagnetic interferences.

3.1 Constraints related to moving around a pylon

- Always keep a safe distance from the pylon. It is requested to stay at a minimum of 4 meters away from cables and insulators. This constraint is highly important due to the high voltage capacity of those cables that can cause serious harm to the UAV and generate unpredictable comportment if too many noises are generated.

  It is also important to keep a safe distance in case of a UAV hardware malfunction. For example, if one rotor break, the UAV can crash and might lose control, which could result in a propeller hitting the cable, this might not be capable of cutting through the cable. Still, it can cause heavy damages resulting in the need for changing that damaged cable. This security distance should continuously be followed.

- Always use the same starting point. This constraint is more related to the inability of the GPS or the barometer to produce precise information related to the positioning. It is then essential to find a way to always start at the same relative point and calculate all the movement with to the accelerometer. This is one of the sensors that is not infected by the noise generated by the high voltage proximity. So, by using the same point every-time, we can expect to get precise positioning of the UAV with trusty information.

  For this master thesis, we will use the top of the pylon as a starting point.

When we are coming from the wire following AI system, it is the most unique part of the pylon we can find easily and close to where we arrive.

- Get top-down and bottom-up angle photos from each side face to get all the possible problems that can be detected. It is essential to have different angles to get as many information as possible on the corner pieces that compose the pylon. Getting those bottom-up and top-down image can help us get the most information on the pylon by only doing two types of photos. And like so, produce a nice map of each pylon with the localisation of flaws.
- Obstacle avoidance. While the UAV is on high altitude, the probability of touching an object is low; But, we still have to take into account active (moving objects like birds and leaves) and passive (non-moving object such as fences and threes) obstacles. Especially when the UAV is descending along the pylon.

  This constraint is logic and essential to keep in mind since it would require to detect all the possible objects related in the pylon analysis that could be obstacles. It would need to create a safe zone of detection with sensors all around the UAV to detect any incoming objects as fast as possible to avoid any contact.

  Most of the time, sensors can be more useful for the passive obstacles than when it comes to active obstacles which require a longer distance of detection to compensate the potential speed of both the UAV and the incoming object.

  The risk is getting more important when we get closer to the ground. There are much more obstacles, such as threes and potential fences that can get in contact with our UAV.
- The camera is mounted on a three motor gimbal able to stabilise in all directions and change the orientation of the camera on all three angles (roll, pitch, yaw). But, the pitch angle is limited. It can only go on [-90], [$+45$] degrees vertically and 360degrees horizontally.

  The camera also has a maximum resolution. This means we can not get more details passed the resolution of the camera, which is 5280x3956 pixels. We can interact with the field of view angle, ISO, HDR and other related function related to photography to make some details more visible or change the perception of the camera.
- Centring the UAV to the pylon is also essential. The more we are centred to the pylon, the better it will be for complete analysis. Being too much out of focus of the pylon can results in the camera not retrieving enough data. This can lead to a higher possibility of not detecting flaws due to the angle generated by the perspective.
- One of the most important constraints is the time taken to analyse the pylon, going too fast can result in blurry images or going too slow can make the UAV unable to perform the whole pylon analysis with a single battery. It can also cause the UAV not to be able to make more than one pylon between two charges. This would require an operator to be always close to the UAV to change the battery at every pylon.

From those constraints, we can see that there are two optimisations possible. One is the distance between the UAV and the power line and the pylon. The distance needs to be of at least 4 meters and be stable to help with the focus of the photos for further investigation in the image.

The second would be the path of the drone around the pylon, to take as much correct top-down and bottom-up photos of the pylon. It is also interesting to get as many different images and the fastest way possible while respecting all the constraints.

3.2 The electrical pylon conception

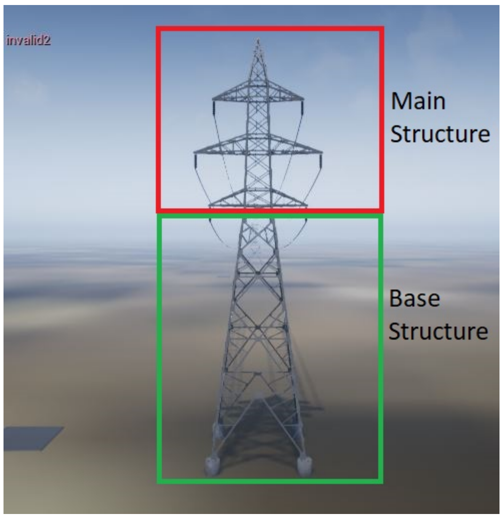


Figure 3.1: Two parts of the pylon conception

As we can see in Fig 3.1, the pylon is divided into two-part. There is the base structure, commonly inclined from a larger base to a smaller top edge. It usually has no cables or dangerous part linked to the base structure. It is used to elevate and make sure the upper structure is stable and high enough to avoid problems with ground obstacles.

The second part, the main structure, is the most important piece of the pylon, it is composed of a straight central structure with a pointy top edge named the dome and multiples arms. The main part of the structure can widely vary in shape and size depending on the country and where it is situated. On the side edge of every arm is situated insulator that is linked to the high voltage cables. The top cable, on the dome, is commonly named the ground cable.

As stated earlier, the starting point could be the Dome as it is always at the top level. It will be the thing we see every time we do get in contact with a pylon while following cables. We have to correctly place the UAV with the top edge of the dome at the centre of the images flow of the UAV to get a precise positioning related to the pylon.

Chapter 4

Data collection

In this section, we discuss how the data set was created and what kind of data we extracted from it. We first talk about the camera that the UAV is going to be using during the whole dissertation. To help us understand the format of the photos that we will use. Then, we are going through the creation of the data set and the choice made in it to get as many different types of images as possible.

4.1 Camera settings

First of all, it is important to talk about how the photos are taken to know the size and shape of each image extracted for analysis. The simulation has multiple settings that we can be tweaked to obtain whatever type of photos we want. The camera parameters that we used to take the photo in the simulation whereas following:

- Height: 2500 pixels
- Width: 2500 pixels
- FOV: 80°
- AutoExposureSpeed: 1000
- MotionBlurAmount: 0

The high resolution is because the UAV takes photos with the same camera for control and the analysis. So the image has to be on a higher resolution to be able to zoom in as much as possible. This is needed to search for flaws (either by AI or UAVs by humans) with precision.

It was decided to go for a square shape for the image to avoid distortion when transforming the photo for the movement algorithm that use the resized photo of 250x250 pixels resolution.

The FOV or field of view is the area the camera can cover at a given time. By using a FOV of 80 degrees we are able at 4 meters to see the pylon and the surrounding correctly, making it easy to navigate back to the pylon in the case the wind would push the UAV on the side, making the UAV not centred to the pylon.

AutoExposureSpeed is the speed at which the camera can adapt on a millisecond scale [30]. Still, if it is put too low, it can generate artefacts on photos. In order to reduce the danger of having artefacts and also because we are in an environment where we don't change luminosity and distance often, we can set it to 1000 which is correct for us and high enough (ArSim ask to not go under 100 on this parameter) to avoid any artefacts.

The motion blur was also deactivated because of it as a tendency of making the simulator more unstable due to computational power requested for such a useless parameter. It can also cause artefact when it is on, as [30] state in its documentation.

4.2 creation of a simulated data set for the U-Net system

The training data and the validation data are all coming from the same three-dimensional pylon model taken from different angles, distances and light sources in the AirSim simulation [30] with a different background. We then proceed to execute a simple algorithm of data augmentation to boost the number of images generated rapidly by making a horizontal inversion of all the pixels.

The images provided by AirSim are made to be realistic, especially for the pylon, which is photo-realistic. For this data set, we took some liberties by making one of the background less realistic by putting nothing but a raw dirt terrain without anything around. We then have made a more natural background for the same pylon with some threes and houses.

Here is a more in-depth explanation of the different backgrounds that can be seen on fig 4.1:

1. The "plain ground" is made to be the simplest environment possible. Thanks to this environment, it is possible for us to get a full view of the pylon without any obstacles. The pylon is popping-out with this contrasting background. This makes nice and clean images. It is interesting to have those kinds of non-complex images for the U-Net system to train on. We can compare this environment to a pylon in a field or a plain that we can see when going around in real life. There is a lot of pylons that are crossing countries by going through fields.

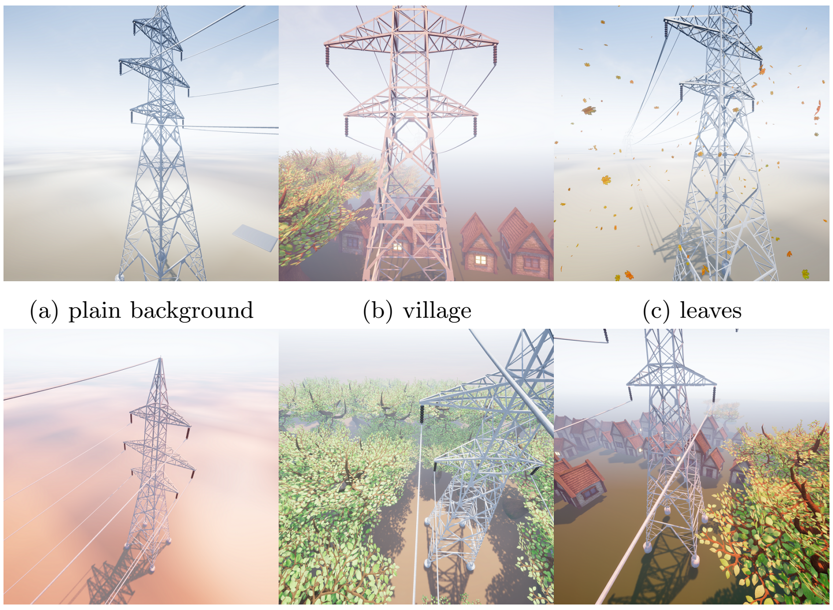
2. The second environment is surrounded by dense vegetation composed of threes. This enables some photos with pylon semi-visible through threes and background with colour ranging from dark green to light brown. The complex shapes of the threes are useful to get branches that go on the field of view of the camera. We can compare this background to the power lines that need to go through a European forest. This is also making the pylon contrast well with the background.

3. The last environment is representing a suburb place. This is where you get a mix of small houses and threes. The environment is familiar in real life since power lines are mean to bring electricity to villages and cities.

The gray wall houses and brown to black roofs make the pylon contrast less with its background. This environment is created to be the most challenging one for the AI to train.

All those environments are subjects to lighting variation, from colour (simulating sunrise and sundown) to luminosity intensity. Like so we can make different contrast and even change the colour of the pylon by making the sun and sky look more orange just like a morning sun.

Other effects can be produced, falling leaves, fog, rain, snow. All of them being sensible to tree parameters, wind direction, wind intensity and the intensity of the effect itself. The effect can generate a nice amount of noise for the data set to create more difficult images where different objects can obstruct pylon visibility. We did not go too far on the intensity of fog and rain effect since UAVs require a clean weather to fly. But we can still expect some fog from time to time just like a small rain can happen while doing an inspection.



(a) plain background (b) village (c) leaves

(d) sunrise (e) forest (f) village

Figure 4.1: Example of images from our data set

It was then easy and fast, with the help of the simulation and all the environment created, to generate images for the data set representing various scenarios and possible positioning of the UAV. After one hour, the data set was filled with 1483 different images. Half of them being generated by the data augmentation algorithm in 27 minutes.

To add to the utility of those scenarios, the simulation directly gives the segmented image related to the photo. On this segmented image, we can choose each object or group of objects a specific colour. In that way, we were able to generate segmented images that take the pylon in two parts, the main structure and base structure, on different colours, then put the rest of the segmentation in black. This gain of time is precious in our restricted time windows for this dissertation, making the precise segmentation alone can take up to 40 minutes per pylon to do it properly, so being able to generate instantly segmented masks of an image is a non-negligible point for the simulation.

From this data set, around twenty per cent of the images (297 images) were chosen to be validation data, and then the rest of the data set was taken to be the training set (1186 images). Typically, that number of images is more than enough to train a model such as U-Net since this model was made to be reliable on a minimal data set of around 30 images at the beginning [31] to predict the segmentation mask of one object (cell).

Finally, to test our model on real images, we can create a minimal data set of real images, but this requires to make proper masking by hand that takes an hour as stated before around 40 minutes per images. Having about 30 real images masked to test our model precision in real life is a very interesting thing to have for results analysis. This would be mainly used to test the capabilities to be directly used in the real world.

Chapter 5

Implementation of the passive vision paradigm with the U-Net model

In this chapter, we are talking about our passive vision system that is going to be an implementation of the state of the art U-Net model like [13] describe it and a set of rules. We will analyse the first results we had and all the step we have made during this master thesis to obtain our final model.

We are also discussing the controller part of the U-Net implementation, how we achieve to get a UAV to fly autonomously around an electric pylon. This includes the approaches taken, analysis of the outputs of our U-Net model and difficulties encountered during this part of the project.

5.1 The model architecture

Our architecture is the most common U-Net architecture you can found. This architecture is adapted to our needs and data. This has been chosen due to its capabilities to be adjusted easily with any forms of feeding. This is also a very effective architecture to learn on a low quantity of images.

Our model is implemented in python 3.14 and extensively use the library TensorFlow and Keras. Those two libraries are useful and full of resources to help us construct an optimised convolution neural network ready for GPU computation.

Here is a complete explanation of our architecture:

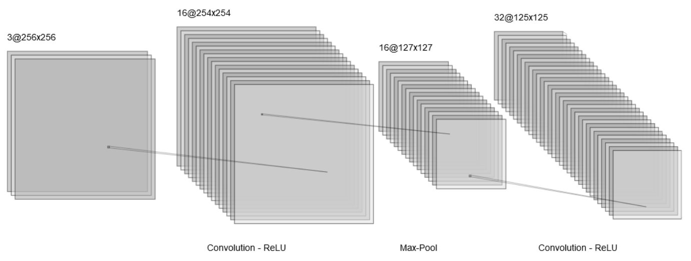


Figure 5.1: the beginning of you convolution network. Since we have numerous layers (16), we are not showing it completely.

1. We start our CNN with our resized base image. The shape of our image is 256x256 pixels square image and composed of three layers. Those three layers represent the RGB colour spectrum. It as to be noted that our RGB spectrum goes from $[0, 1]$ as a R and not from $[0, 255]$ as an Z. So if someone wants to use our trained model, this is important to note that all the channels must be divided by 255.

2. After that, we start a loop of Convolution and Max-pooling to realise our features extractions and detection. Each step will be made of one convolution and one max-pooling

   Our convolution work by multiple of 16 for layers and square filters of size three. This result in the number of layers at each convolution step to grow by sixteen. During the convolution part, we can see the number of layer going from $[16, 32, 64, 128, 256]$.

   With the Max-Polling, we divide the size of the layer per two at each step by using a square filter of two pixels. In case the number of the pixels is odd, we do add a pixel padding so we can divide it per two. This padding is represented by a column of zero at the end and a row of zero at the bottom. We do not use the max-pooling at the last convolution when the number of layers is at 256.

3. Now that the convolution is finished, we can proceed to do the inverse and Upscale our image to its original size and extract three layers representing the background, the main structure and the base structure. We are using a loop of convolution to reduce the number of layers at the end and some up-sampling to make the reduced image bigger.

   Our deconvolution works the same as the previous point but this time we go with a number of layers ranging like so $[256, 128, 64, 32, 16, 3]$. Thanks to those steps, we can reduce the amount of layer to the three needed layer.

   The up-sampling is here to multiply the size of our image to the initial one. We do this by concatenating the up-sampled image from the previous step to the related image from the convolution phase and the up-sampling result. This gives us a better result than simple up-sampling.

   When reducing the layer from 16 to 3, we use the activation called "soft-max". This means that the addition of the values present at the same place in the three layers should be equals to one. We can then assume that each pixel has a precise role and not all three possible roles. A pixel can not be predicted at 80% background and at the same time predicted 80% main structure. This has been the most interesting and logic final activation found.

When the convolution and deconvolution have been done, we obtain an output representing the prediction of the three masks: the main structure, the base structure and the background. As explained earlier, each pixel can only be one thing and not the other two. After getting our outputs, we can now leave the U-Net model on the side and go to the image analysis part of the fly controller system.

5.2 Fly controller

The fly controller receives a live feed of images from the UAV's camera that can be processed. The controller works like a state machine when the pylon analysis is started.

This approach is rules-based, which mean that we have to insert rules for every behaviour that the UAV needs to respect or to control the action the UAV will do now and then. This technique is friendly to use when there are not too many rules. For a more complex system, this can turn into a puzzle of rules that can be counterproductive.

Since we are limited in the thesis to only use the camera live feed, we can only make rules that can be related to it. No other levels of rules from sensors have to be taken into account. We also don't have to worry about the battery in the simulation. This reduces the scope of rules necessary to complete the challenge. But, it is also making it more challenging to analyse distances or to detect obstacles.

Our fly-controller work like this:

1. The camera takes an image. The image taken is 2500x2500 pixels and use the RGB spectrum; we proceed to resize it to the size of 256x256 pixels. Like explained before, the system divides all three layers by 255. Like so, we get an image ready to be fed into our U-Net model.

2. Now the system will be making the prediction. During this step, we use our U-Net model to extract the three masks by giving the resized image as input and getting those three masks in output. The details on how it works being explained earlier, we are not talking much about it here.

3. From the output, the system is going to extract the information of the pylon. We can start to precisely calculate the position of the main structure of the pylon. The processing of the image is done with a canny function to get only the edges in the image. As explained in the state of the art, this function is very efficient to extract the borderlines of forms present in an image. We feed the canny algorithm with the mask representing the main structure of our pylon. This algorithm cleans our image, removing some noises and also making borders out of the shape of the pylon.

   Then, a Hough lines algorithm made to extract lines proceeds to detect if there is any line of more than seven pixels long while being one pixel wide.

Those lines have to be vertical, to do so, we take only the lines that have their both edges on approximately the same X-axis coordinate. Like that, we get the coordinate of every vertical line in the image. The processing then proceeds to find the vertical line that is the most on the left and the one the most on the right to have the horizontal limits of the pylon.

After that, we try to detect the most upper point on the image processed by canny to know where the top is. Then we check to find the point that is the most on the bottom of the main structure prediction to see where it end.

Finally, we end up with a proper box that gave us all the coordinate of the position and the relative size of the pylon in the image.

4. Now that we have the position of our pylon. The controller can start to move the UAV by rolling either left or right to centre the UAV. The controller is continuously looking at the new position of the pylon to keep it centred. If the UAV is already centred, we don't have to move, and we can go to the next step.

   This step can counter the effects of the wind and potential unbalance of the UAV that can make it drift slowly on one side.

5. Now that UAV is centred, it is time to adjust the distance between the pylon and the UAV. The system is doing it with the width of the predicted box of the pylon. Just like the last step, we use the width of the predicted position of the main structure to guess the distance between the UAV and the pylon. This step would be safer and more reliable if some sensors were used.

   This step also fights the effect of the wind, and the unbalance of the UAV that can make it drift passively. This is doing it passively while checking the security distance.

6. Finally, we Check if the UAV is not too high or too low compared to the main structure of the pylon. This is done by looking at the position of the prediction box. It then proceeds to take one of those decisions:

   - Depending on what part of the pylon analysis the UAV is at, it can move either down or up. This action only occurs when we are on the outer side of the pylon. Those movements are mean to be the one used when doing the bottom-up and top-down inspection of the pylon. We always use the same rotors speed and same time of execution, so no parameters have to be taken from the position of the pylon to adjust the climbing speed.
   - If the UAV is low enough, it can proceed to go around the pylon to start analysing a new face of the pylon. It is necessary to be low enough compared to the main structure of the pylon to go around the pylon to avoid any collision with cable. There is also risks of collisions with objects unrelated to the pylon.
   - If the UAV is too high compared to the main structure, We can start to analyse the top-down view analysis and proceed to go down. This how we know when to switch from the bottom-up to top-down pylon

analysis. We wait for the dome of the main structure to be around the same height as the UAV by looking at its position box.

- If all the faces have been analysed, the UAV can go back to the wire following AI. by elevating itself back to its original starting position and start back to follow cables.

By choosing to work that way, we can have a UAV doing a full pylon inspection without colliding with the pylon. We have also avoided going multiple times on the same path. We theoretically reduced the potential time taken to do the analysis. All those movements can make us respect all the constraints related directly to the pylon and enable us to do all the requested action to complete the full pylon analysis. Of course, optimisation and adding rules is possible, especially the one related to obstacle avoidance. It would require to get proximity sensors to generate data to verify the rules developed for this topic. Also, it has been chosen to be slow for the ascend and descend to be the most secure and stable as possible.

We can see on [31] the very first implementation of the fly controller controlling the drone as an example. As we can see, there is still a lot of optimisation possible.

5.3 Training

This section will be dedicated to talking about the different training steps that we went through to achieve the model capable of doing the full pylon analysis [31].

For the training, we are going to be using our training data set composed of 1186 high-resolution images. Since those images are too huge to enter in our model, we load all the images in our instance and resize them to a 256x256 pixel's size image.

We resize the image to 256x256 pixels because our GPU memory is not able to support a bigger file size. Once we have the outputs of the fed image in the model, we can compare it to the real masks of the image since it is supposed to be the same value in the image array. The accuracy is calculated on the comparison of the predicted output and the actual output that it should make.

To analyse the results, we are using the F1-score and the accuracy to deduce the correctness of the model. First, here is the explanation of the two approaches tested in terms of objects detection, only pylon detection or background and pylon detection.

5.3.1 The first model, only detecting the pylon

The first approach used for the U-Net model was made by just trying to predict the pylon's main structure and the pylon's base structure. They are the only parts of interest to us for guessing the movements. The last deconvolution used is a softmax activation which means that a mask is an independent prediction to the other one.

The model has resulted in poor results. The analysis shows that this might be because it is trying to give meaning to the background. An observation of the results was showing some opposition on the two layers of detection, and the model was ”finding” potential pylons everywhere on the image.

5.3.2 The second model, background and pylon detection

For our second model, we decided to use the results of our first model to think about what was going wrong. It was then decided to go for an approach where we would detect not only the two part of the pylon but also the background. It was also decided to switch from softmax to sigmoid, where we would have an additive proportion between the masks of the outputs. The probabilities would be closer to either one or the other than independent results for each mask, like softmax would make.

The data set had to be rebuilt with a third mask extracted from the simulation. It was easy to get this information thanks to the easy to use function from the simulator APIs. It requested just an hour to prepare everything to be ready to train again.

The obtained results were more precise than the first version, and we can now precisely distinct each part of the pylon. This is possible thank to the fact that we use the background detection and also the sigmoid activation. Our AI is trained to give some meaning to the background now.

5.3.3 Comparison of the two models

We will shortly discuss the results of the two approaches in term of accuracy and F1-score.

Test function	Accuracy	F1-score
Softmax (without background)	0.445874	0.213562
Sigmoid (with background)	0.997854	0.876616

As we can see, the results are drastically different and reflect what we were seeing during the training of each model. This shows us two things to analyse:

- The first is the difference between the two models. This huge gap is undoubtedly due to the fact that we use the sigmoid activation, leading to either one of the mask dominating the two others. If we had used softmax on a three masks detection, this would have resulted in a worse result. This is important for our model to get one distinct element to pop out per pixel. Also, using sigmoid without the background would have been a terrible idea since we don’t want to detect anything else than the pylon. With sigmoid, the algorithm does the addition to reach one on all the pixels.

- The second point we can look at is the difference between the Accuracy score and the F1-score. The results of the F1-score are low because it is taking into account the false negative as a relevant element to calculate the mean precision of our function.

  The false-negative is the fact that an element that should be considered positive but it showed negative. In our case, it means parts of the pylon are ignored or too little for detection or not seen because of the low-quality image. But, at this point, we don't have many pieces of information to deduce how the false-negative affect our score.

5.3.4 Optimiser choice

After this, we proceeded to test different optimisers on our sigmoid system to see if we would get better results. It would be interesting to have a reduction in the difference between the accuracy score and the F1-score.

To do that, we test multiple optimisers: Adam, Nadam, AdaMax and AdaDelta. We test them on the same training data set and verification data set.

From the four tested optimisers, here are the results:

Test function	Accuracy	F1-score
Adam	0.997854	0.876616
Nadam	0.989625	0.844557
AdaMax	0.964584	0.765917
AdaDelta	0.756952	0.423695

The results are showing that we have two optimisers that perform better, Adam and Nadam, the two other optimisers are performing worst. The difference is highly visible when we look at the F1-score, where we get an eleven per cent difference between the first (Adam) and the third (AdaMax).

All the optimisers are an adapted version of Adam. Those gradients descend optimisers are different but still use some of the same features. But one thing that makes Adam stronger, and considered the default one in most of the case, is that Adam uses bias-correction help toward the end of the optimisation. ¿hen the gradients start to be sparser. It can also be said that Adam can tend to be better at getting out of local minimum during the gradients descend.

We have observed during the training that some optimisers have difficulties to improve their results over the multiple epoch. AdaMax and AdaDetla are the two optimisers that showed the most problems. On the other hand, Adam and Nadam have shown outstanding capacities to obtain better accuracy at every epoch. They stopped getting a better score at around 0.97-0.99 of accuracy.

We stop the training with an early stopper to avoid the model to overfit when the model doesn't perform better after five epoch.

Finally, on those results, we can see that the difference between the accuracy and the F1-score is even more marked in this second experiment. We can see that the number of false-negatives goes even more up. This means that we can say that one mask might be able to tanks the accuracy for the other, while the two other masks are making a considerable amount of false negatives. Next, we are going to try to clarify this hypothesis.

5.3.5 The false-negative hypothesis

This hypothesis is that false-negatives cause a lower score in terms of f1-score. Because they are taken into account, those false-negatives happen when a pixel in one mask is not marked false while being true or 0 while being 1. The accuracy checks just the percentage of similarities. So having one layer with good precision with a lot of true pixels belonging to that mask can lead to higher accuracy while other layers with lower accuracy have way less weight in the accuracy calculation.

While analysing the different masks, we can see that the background detection is the one taking most of the space in an image.

We can clearly state that when the background is taken into account in the accuracy. It makes the accuracy goes way higher because it takes most of the image space. For example, sky, house, threes around pylon parts.

Also, the detection between the upper and lower parts can lead to more errors. They are very similar and cause the model to have more difficulties in separating them well. This is resulting in a higher potential false-negative between the two. It results in the model to make more errors that are hidden by the accuracy from the background detection.

The results can explain the difference in the F1-score that we met at the next step of the training of our neural network. The next step will be to test our model on real images and also check test the difference.

5.3.6 Pylon detection on real image

Now we are trying our model on a small test data set with 10 real images annotated by hand. It was made during the master thesis to see how effective the models are on real images. The lack of images is evident for this part. But creating the masks for the segmentation to calculate the score is a long task. We don't have much time to prepare more images during this master thesis.

A result of prediction in fig 5.2 shows the same image with its different predicted masks. We can see the shape of the pylon, but there are still some errors appearing. Artefacts appear on the intern part and the arms. Those parts of the pylon are very thin in general compared to the four central vertical beams.

Here we can see the accuracy of the detection on real pylon with and without the background prediction taken into account an the F1-score next to it.

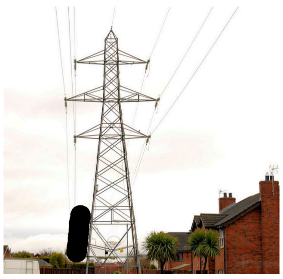
Accuracy	With background	without Background	F1-score
Adam	0.842190	0.603505	0.585651
Nadam	0.800957	0.403360	0.374902

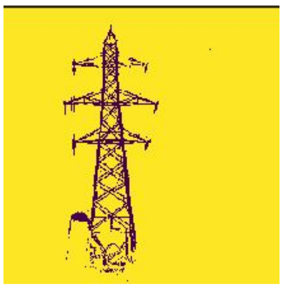
We can see that the background is tanking the score of accuracy. When it is removed, it looks more like the F1-score.

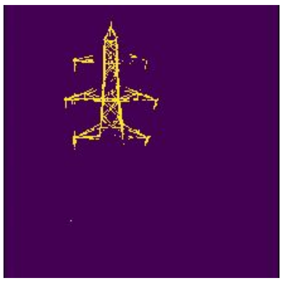
Also, we can see an impact of the reality gap here with the score clearly dropping lower and making the use of our models certainly not stable enough for the real usage.

The results can be explained because, in this real pylon data set, the images have pylons of different architectures. The differences helps us to test the capacity of the model to adapt and resist the reality gap. So most of the best detections are from pylons that have the same architecture as the ones in the simulation. Just like the one on fig 5.2.

So pylons with really different shapes of the main structure tend to be very badly detected. Most often, the base part is well detected. But the model tends to get artefacts of base pylon where there is none. Most of the time, this is the background being detected as a pylon part.


(a) Bottom part pylon prediction


(b) Background prediction


(c) Upper part pylon prediction

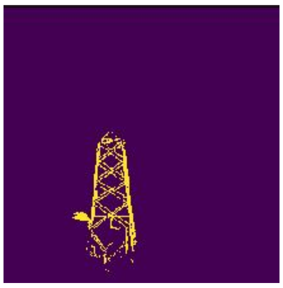

(d) Bottom part pylon prediction

Figure 5.2: Result of detection on a real normal shape pylon.

5.4 Conclusion on passive vision

From all those experiments that we have tested during the training of our U-Net model, we can deduct that the best model is the Adam model with background detection and sigmoid activation at the last deconvolution. This model can get pretty good results on the simulation. It can help us to build a robust fly controller but is going to lack precision when it comes to real pylons.

It might be possible to obtain better real-world results by adding real images to the training data set. By doing so, we want to make the model recognise the real and simulated pylon correctly by only doing one training. This could minimise the effects of the reality gap on the fly controller.

Chapter 6

Research around the active vision

This chapter will talk about the research done in this master thesis. We will go through the pre-processing of the image to make it understandable for the neural network. After that, we will talk about different gaze approaches. Then we will explain in depth our model architecture and how the training went.

6.1 Preparing the four gaze approaches

First, we need some images modification process for the implementation of our active vision system. In a first attempt, we are looking at the gaze system just like in chapter 5 of [23]. Four gaze versions can be looked at. Those four approaches can be trained to find the most effective way to get a solution to our problem.

First of all, the image is set to a dimension of 250*250 pixels in grayscale. The scale for the intensity of grey is [-1. 1]. -1 being black and 1 being white. The grey intensity makes the pixel be one value and not an array of 3 values for RGB. This is a good way to transform our data into an easy to interpret tensor.

The three first models will use the same CTRNN to be trained with. Only the input data set to the neural network will be different. This neural network will consist of two subs neural network that will have the same inputs image.

The UAV control neural network will consist of 29 inputs. The first 25 inputs for each pixel from the gaze window and 4 inputs for the gimbal quaternion. For the outputs, we are looking at five outputs, four for the new directional quaternion and one for the motor control.

The gaze control neural network will have 29 inputs, 25 inputs from each pixel of the gaze windows and 4 inputs for the gimbal quaternion. As for the outputs, two inputs will be retrieved. Those will be the new movement of the gaze.

The first approach, named ZOOMED-GAZE, will be using a system of gaze on a window of 25*25 pixels. From that 25*25 pixels window, we will make 25 distinct "pixels" composed of 5*5 pixels of the gaze window. Then, we calculate their value by doing the means of all the pixel's value in the 5*5 square. This technique will be used to determine if a smaller gaze window is essential for the results. The method should help the algorithm by reducing the loss of quality in the image. Because, if you are going for bigger windows but keep that 25 pixels inputs, you can lose pieces of information in the reduction. This technique can also enable more gaze movement possibilities.

The second approach, named NORMAL-GAZE, will be using the same size of gaze windows as [23]. We will consider this approach as the "State of the Art" approach. This technique will focus on a 50*50 pixels window for the gaze. That gaze will be divided in a 5*5 pixels size image. Just like ZOOMED-GAZE, NORMAL-GAZE will calculate the value of each pixel of the 5*5 image by doing the means of the pixels in that zone. This approach will check if the standard state of the art gaze parameters can work for our problem to solve compared to an autonomous car on the road.

ZOOMED-GAZE and NORMAL-GAZE aim to see if the size of the gaze can influence the result. We can also see if the gaze parameters are useful to optimise in the future.

The third approach, named GIMBAL-GAZE, would be to make no modification to the image but generate output to move the gimbals so the UAV can change its sight without yawing around. This technique would make the UAV work like it had eyes to look around with a certain degree of freedom.

The last approach, named NO-GAZE, will just use the resized grayscale image to develop a solution without using "gaze". It will be used as a sort of benchmark. We will still reduce the number of inputs by reducing the image to a 10*10 pixels image. The value of one pixel of the image will represent the mean value of greyscale of a square of 25*25 pixels at the same position in the picture. The method will result in a hundred inputs to be sent into the CTRNN.

Due to the time it takes for the training and the time-related restriction of the master thesis, only the latter approach will be tested. Considering that [23] shown that gaze control does not give much improvement in car control. Further studies could check if that type of model with gaze control can be effective for the three-dimensional capabilities of movement from the UAV.

6.2 Neural Network

Continuous-Time Recurrent Neural Network (CTRNN) is the type of neural network selected. This fully connected neural network had to be made from scratch in python since no standard libraries exist that could help us on the creation of such a network. In order to use GPU, we have to keep in mind the use of tensors. The library PyTorch will be used to manipulate the different tensors generated to create the CTRNN. By creating our library for this neural network, some abstractions have been possible to be made to help at the creation of very custom and easy to modify CTRNN. By doing so, some modification can be made very easily to the neural network such as changing the number of inputs, hidden-layer nodes and outputs and other parameters that will be explained in further details. In those next sections, we will discuss the different parts that make the neural network.

6.2.1 Gene file

This JSON file will provide the DNA of our UAV. Those genotypes represent the different parameters like the bias, the gain, the weight of each connection in the neural network. By using a JSON file and python, we can easily read and write in it through a library called pandas that directly transform the JSON file into an entirely usable data frame. This file has two variables:

- gene: this is a list of a decided length by the user. Each item of the list is a float of value in range $[0, 1]$ that represent a gene.
- fitness: This variable is either None or a float representing the fitness obtained after testing that genotype. The fitness is the mean of the six fitness score calculated during the test of the genotype. This value is always checked before running the test, so we do not waste time testing the same genotype twice.

Here is a small example of the file for a genotype of length 3:

```
{
    "gene":{"0": 0.3221, "1": −0,453, "2": −0,00443224},
    "fitness": None
}
```

6.2.2 Configuration file

This file is the core of how the neural network will be after being generated. This file is in JSON format for the ease of use. From this configuration file, we can extract information such as how much inputs we want in our system. How much hidden-layer nodes are going to be present. How much outputs we want to have. Thanks to this file, we can either test different configuration but also making it easier to interact with each layer without having to go through all the code of the neural network. This helps us to avoid making mistakes in the code of the neural network. It is still enabling us to have some freedom to change meta parameters of the neural network.

The JSON is represented as such:

```
[{
  "updateSec": 0.13,
  "numberInput": 100,
  "numberHidden":16,
  "numberOut": 8,
  "lowInputWts": −8.0,
  "highInputWts": 8.0,
  "lowInputBias": −4.0,
  "highInputBias":  4.0,
  "lowHiddenWts": −10.0,
  "highHiddenWts":10.0,
  "lowHiddenTau":  −1.0,
  "highHiddenTau":  2.2,
  "lowHiddenBias":−5.0,
  "highHiddenBias":  5.0,
  "lowOutputBias"   :  −5.0,
  "highOutputBias" :   5.0,
  "lowSensorsGain"   :  1.0,
  "highSensorsGain"  :  13.0
}]
```

6.2.3 CTRNN

As stated earlier, our neural network had to be created from scratch since there were no available libraries to fulfil our request to create a CTRNN. This is partially due to the complexity of that type of neural network.

First of all, we start by resizing the 2500x2500 pixels image we are getting from the UAV's camera to obtain a lower resolution 250x250 pixels image, so we don't have too much data to process afterwards. Also, the model does not request that many pixels to work as we will explain in further details next.

The reduced image is then transformed from RGB to a grayscale image. Like so, we can end up using fewer channels layer, this helps us reduce the size of the potential vector to use when we feed it to the CTRNN.

The next action taken is to transform the image into the vector that we will be feeding the CTRNN. We are simply doing this, by Average Pooling, Fig:6.2, the 250x250 pixels image and using a filter of 25x25. The pooling helps us reduce the size of the image to a 10x10 pixels image using the average value of each pixel in each filter. As a small explanation, the Average Pooling is a method used to reduce the size of an image. It is helping to reduce the computational power necessary to compute an image. It can also produce some features extraction. They are two crucial ways on how to do it, like here the Average Pooling, as shown in fig:6.1. We take the size of the selected filter and complete the new image by taking the average of each pixels value in the filter. We can then recreate the minimised image by putting the average value at the corresponding relative place of the filtered square in the new image array. The other pooling solution is Max Pooling. This one uses the same system of filters. This time, it is the maximum value of the pixels present in the filter square that is taken as the new value.

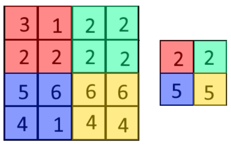


Figure 6.1: Result of an Average Pooling on a 1@4x4 image to a 1@2x2 image using a 2x2 filter

Once the Average Pooling is done, we end up with an image of 10x10 pixels that we can flatten into a vector. We put that extracted vector into a tensor ready for GPU usage. After all those steps, our vector is set to be fed into our CTRNN.

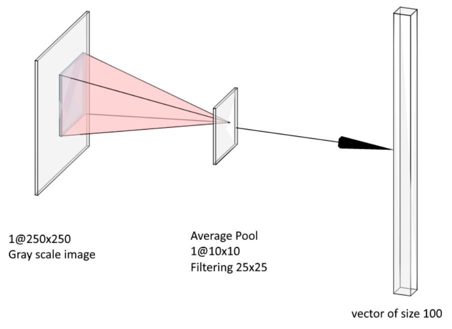


Figure 6.2: The image transformation to a 100 values vector

We can now use our vector in our CTRNN by inserting it as an input. After that, we use our API to do a step in our CTRNN to obtain the output needed to move the drone. We will now explain how our CTRNN is built. We can have a look at fig:6.3 to see the structure. As explained in the state of the art, the CTRNN is a fully interconnected neural network. This means that inputs propagate to each one of the hidden layer nodes, and each hidden layer nodes are connected to each other. This last constraint of the CTRNN explains why it was needed to create the neural network from scratch to achieve this in python. Then, all the hidden layers nodes are connected to all the outputs. The network's nodes are activated using a system of differential equations. With that complex system, we get a sort of memory between the activation of nodes and the last output vector. Like it is explained in [23]. //

Structurally speaking, All layers are represented by its own tensor. The tensor is itself composed of tensor that represents a node. A node is composed of its different parameters that are updated during a step.

- State: this is the actual state of the node.
- Outputs weight of current layer: It is an array of size of the number of nodes in the next layer that is composed of each weight.
- Self-weight of current layer: This is an array of size of the number of nodes in the actual layer. Only used for the hidden layer. The size is null for the rest.
- S: this is the output of the node when it is activated.
- Tau: it is the activation time parameter.
- Bias: it represents the bias value of the node.
- Gain: this is the gain value of the node.

We use the gene file to complete the values of those nodes' tensors, one element of the gene data frame goes to only one of the node. This makes us have a large spectrum of possibilities in term of genotype to search from.

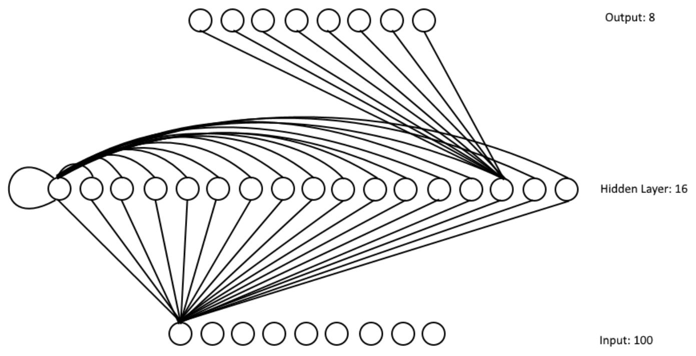


Figure 6.3: The CTRNN simplified structure

6.3 Genotype modification

One important thing in the training of a genetic algorithm is the way the best members of the population are mutated to reach the optimisation goal. Just like in the biological evolution, the offspring is created by either direct mutation (mimicking the comportment of some cellular organisms like bacteria) or the genes mixing like an organism that need two members to reproduce the race. To do so, we have defined four versions of the creation of a new gene that take one or two parents. This section will talk in-depth on all of the four different approaches used in the master thesis.

6.3.1 simple random mixing

This first approach of mixing two genes is the simplest one of the four. We generate a new gene by randomly choosing with a fifty-fifty probability at each index either the first or second gene to extract the gene element at the given index. Like so, we get a gentle mixing that does not take into account the fitness level of the two parent's genes. It makes us explore a certain range of solution without going to close to the same tested solution that could dominate if the partitioning was proportional to the fitness score.

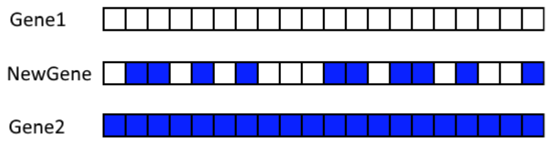


Figure 6.4: Example of simple genes mixing

6.3.2 Ten folding mixing

This second approach is a ten elements folding system combining two genes. This means that just like the first approach, we randomly pick the new genotypes every ten elements on one of the two parent's genes. The folding can be a useful method if some genotypes get some good relationships with each other. In that case, taking ten elements by ten elements reduce separation risk of those correlated elements.

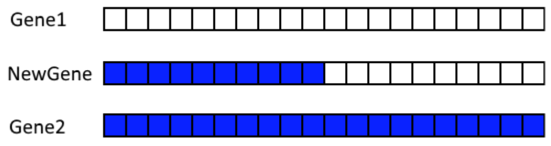


Figure 6.5: Example of folding genes mixing

6.3.3 Complex random mixing

This third approach is made to help the gene with the best fitness of the parents to propagate better. This technique allows the best parent to take more importance and get more chance to be the one giving its genotypes to the newly created gene. To calculate the proportion of mixing we use this function:
If $\Delta fitness <= 0$:

$$Proportion = (\frac{|\Delta fitness|}{fitness_2})$$

Else:

$$Proportion = 1 - (\frac{|\Delta fitness|}{fitness_1})$$

By doing so, the best of the two get more a bigger probability to have its genotypes sectioned for the creation of the offspring. The folding makes the new gene continue to research optimum closer to that parent's gene in the solutions range.

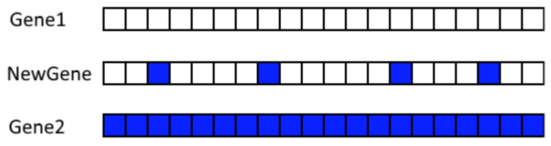


Figure 6.6: Example of complex genes mixing, $gene1$ having a probability of 80% of being choosen

6.3.4 Mutation

This last approach is the only one that only gets one gene in the process. It is made to produce a smaller mutation on a gene to expect potential better results in the same scope as the parent gene. The new element is calculated as follow:
If $randomNumber <= 0$:

$$newGene[i] = (oldGene[i] + (oldGene[i] * randomNumber))$$

Else:

$$newGene[i] = (oldGene[i] + ((1 - oldGene[i]) * randomNumber))$$

6.4 Abstracted simple fly controller

During the first iteration of the training the model, we went for very low-level control of the IA. We had four outputs on the model representing the three Euler angles and the rotor strength. but it has shown difficulties to keep the UAV flying properly or losing the control and crashing too often.

Those problems can be easily explained. First, we had the problem of having one output for the rotation speed of the rotors, which would lead to a crash of the UAV if the rotor speed would be set to less than 20% which result in a stalling situation. Stalling leads to the need for a higher rotor speed for an extended period to compensate for the dropping speed of the UAV. This can also lead to the UAV starting to roll over while falling. Rolling over is happening when the motor stops and the UAV does not have the roll and pitch set to zero degrees. Those behaviours are hard to compensate for that type of heavy UAV and a professional pilot.

Other behaviours related to the Pitch, Yaw and Roll were seen when using the low-level control system. For example, if all the controls were changed at the same times. This drastic change can lead to some instabilities and sometimes push the UAV to lose all the controls. Losing control can finish in either a crash or spin around without gaining back the stability.

It can be explained that the control's parameters can have a big impact on stability. This situation is not comparable to papers related to the autonomous car. Car system can simply give the acceleration and steering without a big chance of losing the integrity of the control. Here, we have to move around a three-dimensional movement space. We thus have to take into account the gravity that can result in trouble maintaining the stability of the UAV. So having movement every 0.13-second result in having a lot of risks related to stability if only one output is too excessive.

It was decided to switch to a more high-level control system with more outputs compared to the first model.
First, four different functions were created, each one having 0.13 second of action time. They are run in a sequence to avoid two or more parameters to be executed at the same time. This should help with the previous stability issues.

One function that controls the rotation of the rotor to help it go either up or down. It takes two inputs $[-1, 1]$ and $[0, 1]$, for the UAV to go up. The first input would have to be > 0 and second input would need to be > 0.55(the hovering speed). The second input would represent the speed of the motor. Like that, we have to get two inputs to have the UAV to go up or down.

Then a function that controls the yaw of the UAV. It would also use a system of dual input. Here, both of them are in the interval $[-1, 1]$. The first acting more like a confirmation to access the yaw and the second one being the yaw parameter. In example, to be able to yaw right, both inputs have to be < 1 to make the UAV yaw for a maximum of 1 $radiant/second$ during 0.13 second.

The third function controls the roll of the UAV. It enables the UAV to drift left and right during 0.13 second at a given radiant. To do so, like the yaw, we use a double inputs system. The two inputs are both values between $[-1, 1]$.

The fourth function controls the pitch to make the drone move forward or backward at a given rate for 0.13 second again.

The three last functions are made to keep the actual altitude to the UAV. For every function, if no consensus is reached between the two inputs. The UAV enters a state of hovering until the next action.

By using the first version of the fitness function with the high-level control system. The UAV had two distinct behaviour going on:

- Yawing around. The UAV starts to yaw in one direction. And sometimes, the UAV can balances left and right. The UAV would not even lose or gain altitude.
- The balancing effect. The UAV balances himself when it is close to being centred to the pylon. Sometimes, the UAV would slowly go up by small steps when balancing.

6.5 Fitness function

Our fitness function will be one of the hardest parts to create and test due to the difficulties to describe the task. The difficulties are coming from maintaining distance, the multiple checkpoints to reach.

Some of the important things to keep in mind are:

- Time constraint
- Keeping the right distance
- Avoiding to crash
- keep an eye on the pylon
- Distance to the checkpoint

Those parameters can be either optimised like time and keeping the right distance or directly be disqualifying such as crashing. Thanks to the simulation, we can have access to data such as the distance between objects by calculating a delta between the two objects. We can also have information related to the number of collisions. We can extract the direction of the UAV and check if he is facing the pylon. The time taken to execute one task can be calculated externally to the simulation. This solution will first be trained in an incredibly simple simulation like the one for the U-Net model. The scenarios will be without object close to the pylon. We will reuse the decorative background created for the data set creation.

6.5.1 Multiple-Objective functions

By making this approach, we will have to deconstruct the pylon analysis movements into multiple functions. Then, it would require to optimise all of them during different training. For example, we can divide the pylon analysis into multiple types of movement. For example, it would be: descend looking at a side face, ascend looking at a side face, switch from a side face to a front face, switch from front face to side face. We would have a system with one fitness function per type of movements and one general function that would sum each one of the sub-fitness functions. The pivot from one function to an another would be to get to a certain checkpoint that would be the end objective to reach for the sub-function and the starting point of the next sub-function.

6.5.2 Single objective function

In this approach, the function will have all the checkpoints for the pylon analysis. Since it's not a linear problem that we are facing, it's more likely to be quite a challenge to solve such a complex approach. One similar thing that could be done would be to use a state machine. It would require to have multiple single objective function and switch between the different AIs. Like so, we would have an IA for ascending, one for descending and one for turning the UAV around. Those being more linear problems to optimise.

6.6 Training

In this section, we will discuss the results of our experiments with the active vision paradigm. We will talk about what we can extract out of it and what we can be deduced.
Unfortunately, The time required to perform a test and try one version of the model takes around twenty-one hours to get enough epochs to reach a local minimum or the global minimum. We are not able to test all the possible fitness functions either.

By using the single objective function, we will need to create one function for each objective of the pylon analysis. Meaning it will require many training to have our UAV do the whole pylon analysis. Due to the time constraint, we will try to optimise the single objective method with a state machine. The state machine is the less complex one that could lead to a possible result. We will, as explained earlier, use the NO-GAZE approach as our studied approach since this is the one that requires the smallest neural network.

The training was done the same way for each try. The population will be of ten members in the population. Each gene is tested six times on two different setups of a pylon (the plain setup and the village setup). We will go through fifteen epochs while reproducing the two best members.

6.6.1 The first version of our function fitness

The first version of the fitness function was tested on both low and high-level control system.

To create the reward for each movement decision, we use $-\left(x^2\right)+1$ basic function to calculate a result reward for each different parameters to optimise. It gets us a nice and homogeneous distribution of reward when we are at either side of the best possible position. Out of having a fitness function, crashing would result in the test run to automatically stop and go to the next test and give a -100 score to the fitness score to punish an action that normally would result on the UAV not being usable again.

We can describe the fitness function like this:

- α represent the Δ of the orientation of the UAV and where the pylon is. 0 is when the UAV look in the exact direction of the pylon. value in degree.
- β is the Δ of the distance between the UAV and the pylon and the optimal distance the UAV should be.
- γ is the Δ of the distance between UAV and the perfect center of the pylon when facing it. To make the UAV get the pylon on the center of the image.
- δ is the altitude of the UAV.
- t_{delta} is the of the time taken for the UAV to achieve the task and the perfect time considered to perform the task (to go fast enough to be effective in power consumption and slow enough to let us take correct photo without blur).

$$\frac{1}{t_{delta}}\sum_{i=0}^{t}\left(\left(-\left(\frac{\alpha_t}{10}\right)^2+1\right)+\left(-\left(\frac{\beta_t}{1.75}\right)^2+1\right)+\left(-\left(\frac{\gamma_t}{5}\right)^2+1\right)+\left(\frac{\delta_t-\delta_{t-1}}{10}\right)\right)$$

As for the results, it is harder to discuss the obtained results since there is no real value besides the completion of the task. The fitness score does not tell anything real on how the model performs in-depth.

First, we will analyse the one obtained on the low-level fly control system. This model was a failure with no genotypes being able to perform our first task. The task was to descend the first side of the pylon. The objective of the task is to be at the correct altitude to go around and respect the constraints. The fitness function results, after three full tests with different starting population, are on an overall score of -74,4 each time for the two best members of the final population. Visually speaking, the UAV would just balance itself left and right centred to the pylon but without going up or down and yawing left or right indefinitely.

From those results, it was clear that the controls were to complex for learning and the overall stability. So the creation of the high-level control system was put in place and tested on the same fitness function.

The results from the second experiment were more interesting to look at but still failing to generate correct results. The overall stability of the UAV was better. The UAV suffers less from balancing and a better centring, but we were still a victim of the UAV not being able to go down. At least, we get a fix on the infinite yawing. Our fitness score after three tests over different population ended up on an average of -3 for the two best members of the population.

After seeing those results and since our fitness function was not giving proper results but making sense logistically, it was decided to try a different configuration file to see if the results would be different. Depending on those results, the first refraction of the fitness function would be done. Because we also have to check if the fitness function would be the cause of our problems.

6.6.2 Configuration file tests

In this section, we are looking at the hypothesis of wrong parameters. They can lead to an unstable neural network. We are making seven different configuration files with four files to proceed into resolving our hypothesis. From those seven files, four files have a common part in the parameters with the original one. Then we have three different files with no common elements. Also, it is to note that $numberInput, numberHidden, numberOut$ are going to always be the same for all the configuration files.

For this section, we only have made one test for each configuration file due to the training time taken. This has required five days to train all those configurations.

We did not have any better results than the original configuration file. The results from those different configurations were in the interval $[-6, -3]$. The result means the activation of the neurons is not much affected with different configurations, or our fitness function is the problem. The fitness function can lead us to a local minimum that always locks us during the genetic algorithm optimisation.

6.6.3 Second fitness function

Due to the poor results obtained during the last experiment. Our fitness function is possibly not on point. Here is a list of the behaviours seen during the analysis of the results of the UAVs.

- Balancing, difficulties to keep itself in the centre
- UAV maintains a steady security distance to the pylon
- UAV is not able to go down or up (some models have shown the abilities to do it but finish to disappear).
- Random yawing, The pylon ends up out of sight.

From those behaviours, we can deduct that some of the parameters do not weight enough in the final fitness score. We will try for this version of the fitness function to make the centring, and the distance to the objective be more important and weighted more in the equation.

Since we are using the base equation of $-\left(x^2\right)+1$, we can easily control its curve. By doing so, it can make a parameter more aggressive or less aggressive, depending on how it performs. We will try to adjust δ, γ and touch a bit at α

Here is our new fitness function:

$$\frac{1}{t_{delta}} \sum_{i=0}^{t} \left(\left(- \left(\left(\frac{\alpha_t}{10} \right)^2 * 2 \right) + 1 \right) + \left(- \left(\frac{\beta_t}{1.75} \right)^2 + 1 \right) + \left(- \left(\left(\frac{\gamma_t}{5} \right)^2 * 3 \right) + 1 \right) + (\delta_t - \delta_{t-1}) \right)$$

Unfortunately, we did not have better results by testing this solution. The results of the fitness function were now positive. This means at least one of the augmented parameters is being correct compared to before. Or, it is compensating more the loss generated by the other parameters.

In terms of number, we end up with an average score in the interval $[2, 73, 3, 81]$. This is not a huge difference from the last experiment. But we have to take into account that some parameters weight more now.

Visually speaking, the UAVs tend to have lowered their balancing effect. We even get our second-best member of the population being a UAV that stabilises itself close to the centre of the pylon and hovers without changing its altitude.

We could go further into the optimisation of our fitness function. For example, to try to make our UAVs go down since this the most problematic thing happening right now.

Unfortunately, due to the time constraint of our master thesis, the pursuit of a functional fitness function wasn't reached. Some discussions about this chapter will be made later. We have some ideas on how to better accomplish this task by using a better and correct fitness function.

Chapter 7

Comparison of passive and active vision

In this section, we will review what the differences are between active and passive vision. Unfortunately, due to the inability to finish the research on the active vision paradigm. We can only compare the data related to the time taken to train the two paradigms and capacity to obtain a working result. We are going to compare the capabilities of the two techniques and try to guess which one is the best related to what kind of expectations you can have.

7.1 Training time comparison

First, we will talk about the training time required to create a solution for both versions. We are going to do it as much as possible. We will revue multiple points related to the time taken in the different steps of the training:

- Creation of the simulations scenarios.
- Creation of a ready to use data set (segmented images).
- Training of the model.

7.1.1 Creating simulation scenarios

Creating correct scenarios in the simulation is an important task to be done. It is at the core of the training for a researched algorithm. The scenarios must stay consistent and close to the reality to ensure that the reality gap will be as much as possible reduced.

Here the time difference is the same since we can generate the same type of scenarios for both models. AirSim is a simulation based on the game engine Unreal Engine. It is easy to find assets of cities or other types of places. It is just essential to find the correct assets that are the most realistic as possible. Assets usually cost money but make us gain a lot of time. Once the assets are in possession, it is relatively easy to create a scenario. Even though it takes times, Unreal Engine is made around a drag and drops world creation, meaning

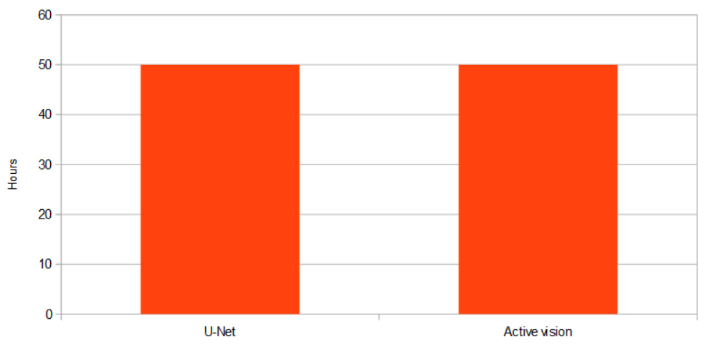


Figure 7.1: Time taken in hour to create five realistic scenarios

it does not require specific skills.

The time it takes to generate five correct scenarios would be of around fifty hours. The time is the same for both models since those does are for the same task.

7.1.2 Creation of the data set

The generation of the data set is something only related to the passive vision paradigm. We need to train our model to recognise pylons. For that, we need proper images so the controller can extract where the pylon is located. To train it, we also need a data set of images with the segmentation of the images included.

Segmenting an image requires precision to the pixel. Segmenting can take an incredible amount of time for such a ”simple” task. This can be subcontracted if a large number of images need segmentation for not much money. It requires software such as Photoshop or Paint.net to do the segmentation by hand.

The amount of time considered here is the time taken by someone that know how to use masks on paint.net. But, it also considers that the user is not necessarily comfortable with that kind of tools. As stated earlier, it takes approximately forty minutes per image to be precisely segmented. To obtain a nice data set usable for U-Net on one object detection system, it would require fifty images to be segmented requiring thirty-four hours.

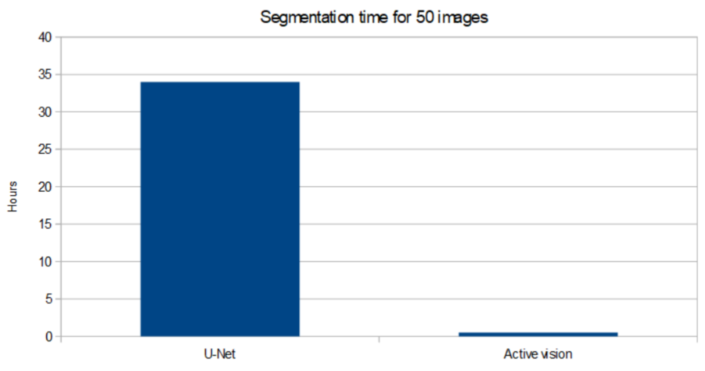


Figure 7.2: Time taken to create 50 segmented images

As we can see on Fig 7.2, the active-vision model gets the advantage on this part of the comparison over time. The difference is because it does not require to use segmented images during the training at all.

7.1.3 training the model

Now we are looking at the time required to train both models. To make clear what is taken into the training here is some explanation:

- U-Net: we count the training the model with the data-set, adjustments and test to obtain a functioning fly-controller.
- Active vision: We count the training of the model with genetic algorithm until it performs the requested task.

The procedure for the training of the U-Net model is an iterative process. It runs through the cycle of "train AI $\rightarrow$ analyse results $\rightarrow$ check the compatibility with the fly-controller $\rightarrow$ adjust the fly controller $\rightarrow$ Test on all scenarios". If one step does not meet our standard, we go back again from the beginning.

Thanks to the fact that U-Net is optimised on GPU. The training of the AI for the pylon detection takes a maximum of an hour of training. We then have to check the result and compatibility with the fly-controller. Checking takes around an hour to do. The longest step is adjusting the fly-controller to the results. We have to make the UAV more smooth in its movements. It takes multiple trials for each parameter for adjustment. The testing on all scenarios takes less than an hour in our case.

The total resulted approximatively to a time of training of 23 hours. This time is necessary to obtain a UAV capable of going around a pylon. It is also considering multiple iterations of all the cycle.

The procedure to generate a functional model with the active vision is a bit more complicated to estimate. Partially because we were not able to obtain a working model. From what we have experimented we can at least say that it does take weeks, for those reasons:

- We can only run one instance of the simulation and one model at a time. It is unstable to simulate a server. This means that only one genotype can be tested at a time. That flaw led to a huge amount of time being used to do one epoch. One epoch requires multiple tests per genotype to get a mean of the fitness score on different scenarios.

- The iterative process that adjusts the fitness function or detects errors in the fly-controller program. We can only detect problems on those two by testing and training our model. It does take time to iterate this process. It is also complicated to detect what is going wrong most of the time. The problem most often comes from either the fly-controller or the fitness function.

From what we have experimented, we can clearly state that more than a hundred hours have been taken without arriving at any results. But it was realised by someone unfamiliar with this approach. So, someone having more experience with this kind of method can potentially perform something working in that lap of time.

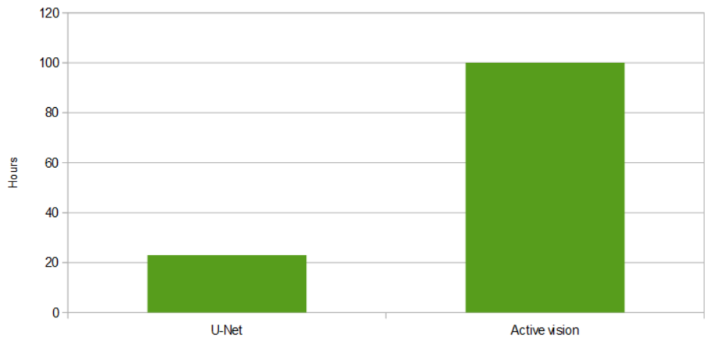


Figure 7.3: Time taken to train a model capable of making a pylon analysis.

Looking at fig 7.3, we can see that our U-Net model is faster to train. It is thanks to the fact that this model is straightforward to understand. Also, we put most of the effort into the rules systems and also the creation of the data set. We can also train the U-Net model on a server easily and make multiple models at each cycle to test. When, on the other hand, you can only have a single instance to train the active vision.

7.1.4 Overall time comparison

We will look at the average total time to finish this comparison between the time requested during the training. It has to be taken into account that this is an approximated time. The active vision would request to properly work, especially for the training of the model to have a precise estimation.

It is also important to remember that this does not take into account the coding time for both fly controller, the programming the two neural networks and every test of the fly-controller done before to experiment it. Since all those activities are not really related and can vary depending on the person behind and the coding capabilities.

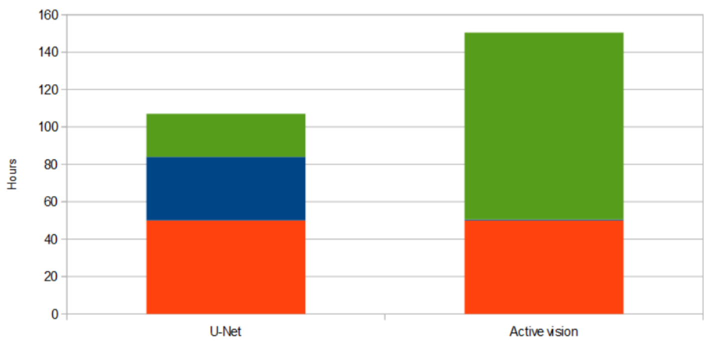


Figure 7.4: Total time taken for the training of a pylon analysis.

First of all, the U-Net model beat the active vision. Thanks to the fact that the training is exceptionally long for the active vision. That time difference is due to the long and complicated process that it needs to go through to try to find a solution. It also requires some iterations to find the correct fitness function parameters. The complexity is caused by the fact that we don't give any direct information to the UAV. The UAV does not know what it has to do when flying. Everything has to be taken into account in the fitness function since this is the only thing giving ”rules” to the UAV. If the fitness function is to complex, this might make it impossible to optimise the solution correctly. The optimisation can end up jammed in a local minimum and making the UAV never perform its task correctly.

After that, we can also discuss the number of segmented images. Here, our U-Net model is made to work with one and only one type of pylon. This makes it have less chance to be adaptive to different types of pylons. That means that for every kind of pylons, we will need to remake a new data set with the new pylon that we want to add. It also considers that fifty images are enough to train the model with a correct score. We can generate images from the fifty existing image with data augmentation also. By using data augmentation, we can easily get a two hundred images data set from the fifty base images. Normally, the active vision model would be able to adapt better to a new form of pylons to a certain extent. It does not base its fly control to the ability to detect a pylon precisely.

Finally, We will say that the U-Net model is the most viable option for pylon analysis. It still requires much more work to be able to avoid obstacles. It also requires to rely on sensors to detect potential objects on the side, below and up.

Unfortunately, we can not know how the active vision supports the frontal object avoidance from scratch. But, surely, it would also require some sensors to handle at least the safe descend. Having sensors would need to get an upper framework checking distances to avoid any crashes.

7.2 Comparison on the modification ease

This section will be about the complexity of modifications and new training capabilities. In example, if someone wants to modify the system. This person might not have the same amount of capabilities than the creator.

To begin with, when you are modifying the system. The availability of tutorials, forums, and external explanation is essential to get some knowledge. Those sources are also important if the person that created the model is not available, and the documentation is not clear enough.

For U-Net, it is simple to find an enormous quantity of websites with tutorials, advises and libraries in multiple languages. This makes modifications achievable for a lot of good software engineers. Also, the actual control is done by a set of programmed rules; those modifications are elementary to perform and does not require any retraining of the AI part.

For the active vision paradigm, there is way less information besides scientific papers and no basic libraries are supporting CTRNN from scratch. It is then requested to have deeper knowledge in case some modifications have to be done. The person must be able to work on a system programmed from scratch that can sometimes be less accessible. Also, retraining is mandatory in case of any change that can result in different output or any neural network architecture modification.

Next, the retraining and modification time is also reduced in U-Net thanks to the whole system being modular and not being one big unit that does everything. This makes adjustment way faster and only related to one part of the system. In comparison, it takes time to perform small modifications on the active vision due to the higher training time that will need to be done.

In a nutshell, The U-Net model is interesting in the case of usage in an enterprise. It is due to the facility to change the rules and the lower complexity of the system. This means it does not require an expert to train back the model. An easy process can be made to adjust the model to the security requirement of a client. For active vision, it differs since it would require a whole new training cycle. It can lead to a lot of work to recreate a new fitness function that matches the requirements. A modification on the fitness function demands an expert in the domain to perform it.

Chapter 8

Future works and discussions

In this chapter, we are talking about what could have been done to improve the research. The multiple possibilities to obtain better results or even points that were not tested in this master thesis. We are also discussing some of the problems met during this work.

8.1 Simulation and work environments problems

AirSim is our first and main environment, this powerful tool is handy and possesses a lot of options, but with all of this, comes some flaws that made our life harder during some steps.

- The impossibility to use AirSim on a server with stability for visual tasks (computer vision). Due to this impossibility, the training of our active vision system ended up being extremely long. When we are doing genetics algorithms related system, it is essential to divide the tests of each genotype to get those fitness results faster. Threading makes us able to perform a comparison between the fitness of the population to select the best ones. This environment flaw did not affect us during the test done with the U-Net model.
- There is a lack of documentation on complex functionalities. AirSim is a huge piece of software that require a reasonably big documentations. Unfortunately, AirSim does not own detailed and complete documentation. This has led us to either find documentation made by someone else or either run around the source code.
- The high computational cost of the simulator. Using a realistic visual simulation is something that requests an excellent CPU and GPU. Both are needed to support the calculation of the physics and also compute the graphical part of the simulation. The more realistic the simulation will be, the more resources it is going to require. This takes resources from our computer that are also used by the models. So, it is important to

limit the memory usage on the GPU. It is important to avoid crashes of the computer running out of resources.

After this, the second environment to be considered is our programming environment. This environment is where our used libraries are installed. For this project, two environments where requested, one for each paradigm.

We did have lost a lot of time with those coding environment. That loss of time is due to the NVIDIA drivers and CUDA version. They are both necessary to use our GPU for computational needs. Libraries are usually stable for a certain range of version of CUDA and NVIDIA drivers. This can leads to issues of two libraries requesting different versions of CUDA to work. This led us to mess the installation of our Windows installation due to driver malfunction. Usually, those malfunctions are coming from modifying the driver. It can happen when updating to a previous version of a driver or badly uninstalling a version of CUDA.

We also encountered libraries having requirement of other libraries but with different versions. It was leading to multiple malfunctions of libraries. It requested time to find where the problem was and to combine libraries to achieve the best stability possible.

Making everything work perfectly and stable is something that requested more than one week for each paradigm. That libraries optimisation has been requested many times for such a short time given. It is also a tricky task to do when you are not used to that kind of installations. In example, CUDA is an add-on the NVIDIA GPU driver that requires Windows command path modification.

8.2 U-Net model's future works

U-Net is an exciting model to study, and it is straight forward to get into it. In this section, we are going to discuss all the ideas of possible improvements that can be done to make this model better.

First of all, adding sensors is mandatory to make this solution more robust to obstacle avoidance. It was requested for this master thesis to perform a pylon analysis with a system composed of computer vision only. Those sensors can either be LiDAR, RaDAR or sonic sensor. The most important thing is that those sensors have to be extremely reliable and do not weight too much. Like so, they do not impact too much the UAV flying time. It is our main flaw in the U-Net model since right now, we are unable to achieve any form of obstacle avoidance.
Thanks to the sensors, we would be able to perform a way safer distance controlling. They can potentially make the model work better on different types of pylons. By performing better, we can also assume that we will be able to do the analysis faster. The safety will not only rely on computer vision. The computer vision has a cycle time of around half a second compared to a sensor capable of making updates every couple of milliseconds.

After that, we can test to stop detecting the two part of the pylon differently. This would be interesting to see how the model is getting better F1-Score

in general since that was a very complicated task to divide the pylon into two distinct parts. It will also make the segmentation easier to be subcontracted. Seeing how the model would be able to be more robust is a nice hypothesis to follow.
This technique would require to change some of the rules related to the height detection and ability to know when it can make a side switch. Before, we were using two parts to know when to switch. But, here we would need a new way to know when to switch. The thought method would be to now detect the cable in our model. Like so, when the camera detects a low number of cables on the upper part of the image and none on the bottom part. We can potentially say that we are low enough to perform the side switch. This technique would need to get extensive tests to make the best fly-controller possible. We need to see how it could work in term of stability and safety. But, this technique is a very interesting way to do the pylon analysis with the U-Net model.

Then, the use of real images in the training data set could be an interesting way of doing the training. Either only use real images or goes for a mix of simulated and real images. Those would be two manners to try that can compensate for the reality gap. We would also test the opposite of what we are doing right now. Test if a model trained on real images can perform well on a simulated environment. Of course, this makes us lose one of the important points of the simulation, which is to obtain data set easily. But if we can get a model that is performing well in simulation and real life, this would be great.
Using real data is also an interesting point to see how the model would adapt to different pylons of the same family. For example, a pylon which has only arms on one side of the pylon.
This would make a nice study on the reality gap in a U-Net based system.

Next, it would be interesting to detect precise pieces of the pylon to make analysis on them or do photos of those pieces with precise angle during the pylon analysis. U-Net would be an awesome tool to detect some pieces like insulator and joint that compose the pylon. Those pieces might require photos in precise angles to analyse them correctly. Using our already made neural network and feeding it with more object to detect would not be that hard. We can also train a separated neural network and make it work next to our model of pylon detection.
This model would be able to detect those pieces and take control for a short period of time. During that time, it would perform the analysis of the piece before returning the UAV to the place it has left the pylon analysis. Such a model is a potential added value for such a project. It can make the pylon analysis professional and precise.

Finally, detecting flaws on the pylon is also important. It would be interesting to perform a complete pylon analysis to detect flaws and their localisation. For that, U-Net is a perfect fit, thanks to its capability to achieve a nice segmentation of any requested object. This flaw detection could be for rust, crack, broken pieces and excessive wear of a piece. Having all those flaws detected would request a lot of data to be generated to achieve it. Still, it can make the pylon analysis autonomous from start to finish.

8.3 Active vision model's future works

The potential of the active vision was not explored due to the difficulties to obtain our first working model. So in our future works with this model, we can clearly and firstly finish what we have started to get a first glance at how it performs.

After that, it would be a must to study how the different type of active vision models that we have explained would perform to the same problem. This would make a nice study on the comparison of other models on an autonomous UAV. We could also highlight if the gaze is as important in our kind of problem. Active vision is a very uncommon paradigm that can lead to a lot of researches for UAVs.

Next, it would also be interesting to use this system to control the UAV decisions but also use a layer of U-Net to make some precise object detection and use some sensors to extract more inputs that could help the system maybe. Making a hybrid system could be an interesting challenge and lead to a network. This hybrid network can increase the stability and capability of modifying its actual task to perform another one. That can be useful for the pylon analysis. It might help to reduce the number of rules that we use to control the UAV.

8.4 Other possibilities

In this section, we will discuss what can be done with the idea of autonomous UAVs related to the pylon analysis. Those reflections aim to add more value to a project like that.

First, after detecting a flaw on a pylon, it would be interesting to have a multi-UAV-system. A second UAV would take off and fly to the detected defect and start repairing it. This UAV would have a multi-function arm capable of restoring some of the flaws that can found on a pylon. That system would be a great challenge and make the system even more autonomous. It could help the worker avoid to go up a pylon where it is dangerous to do some repairs. It would require a new system to be created, with a general controller that controls the fleet of UAVs and after that one to controls the repair UAV. It would also require some research to create miniaturised tools to perform the repair.

After that, it would be interesting to have a fleet system for the UAV that analyse the pylon. UAVs would go and analyse the same power line but propagate them-self to cover a maximum of that power line without checking twice the same place. This would be the optimum way of using totally autonomous UAVs to analyse the grid system at a large scale while reducing the considerably the time requested and the human cost. It would require either an interconnected online system with the main server calculating where each UAVs need to go or an offline system where a UAV gets a sector before taking off and as to only do that sector then leave back to the starting point.

With those two last points, it would make the system a complete solution for fully autonomous electrical pylon analysis, capable of doing everything autonomously and even repair what is possible to be repaired with a UAV.

Chapter 9

Conclusion

We can now conclude our work axed on the creation and research over the inspection of electrical pylon by autonomous UAVs.

This work was composed of four parts:

- Get a simulator working. We have to create scenarios. We have to generate a simulated data set and environments to train our two models. Then, if possible, experiment to see how they work on real-life data.
- Research on the passive vision. A U-Net neural network represents this part. This model uses the strength of image convolution and a set of rules to achieve the task.
- Research on the active vision. A CTRNN represents this part. A neural network fully interconnected that uses genetic algorithms to evolve.
- Make a Comparison the two paradigms as much as possible by the given results of the two above points. We then have to decide the one that is the most interesting for scientific and production point of view.

We have successfully created scenarios on the simulator AirSim. We also have made a realistic simulated data set. Thanks to AirSim easy to use APIs, both points were easy to do. Qualitics helped us by giving some pieces of advice on the different constraints that exist for pylon analysis and in result make our data set more realistic.
We then proceed to create our first model of AI that can perform a full electrical pylon analysis. That part was a success. U-Net is a very basic convolution neural network that can be done easily with both PyTorch and Keras. Thanks to that less complicated paradigm. This first model was achieved in a month. During this first part, we were able to familiarise with the simulator different ways of controlling the UAVs. We had the opportunity to create our first simple fly-controller.
U-Net is also known to be easy to train, which we verified by needing less than an hour to train our neural network. This model was able to move around the pylon without trouble by always keeping itself centred. It was also able to

adjust the distance to a minimum of four meters of the closest part of the pylon.

After that, we started the research on active vision. This more complex paradigm possesses a different way of thinking about the AI-powered control system. The fact that a neural network had to be created from scratch was already a big challenge to go through. This showed us a flaw, the complexity and unavailability of tutorials accessible to everybody. Then the complexity in terms of the fitness function for such a task. It requested much more training. That training time made us unable to finish the research on this paradigm. The biggest flaw from this paradigm is the overall complexity and that doing a change in the constraints request to retrain the whole CTRNN model. It requires at least to run the GA for twenty-four hours.

Finally, a comparison was made on the time and capacity to modify each model to check how those two paradigms performed. From this comparison, we have concluded that U-Net is a better model to work on production right now. It is easier to manipulate and faster to train. But in terms of research value, the active vision is way more interesting with that entirely unexplored method for such task. The time taken to train goes in favour of the U-Net model. Even with the image segmentation, U-Net can train faster than the CTRNN model. Also, the fact that we can change the constraints without retraining with U-Net is important to take into account for a production system. It requires to adapt to different constraints depending on the laws and clients.

In term of scientific added values, this project accomplishes tasks that were not made on any papers that we know. We used untested approaches to resolve the problem on a UAVs. For example, U-Net is used in the detection of objects such as cell or can be useful for a basic self-driving car. In our research, we do apply more dimensions to the controller than just a car would have. The difficulty to always focus on the same object is also present. We also managed to calculate the approximate distance from an object and the centring using U-Net and some image computation algorithms.
We also started the research on the active vision in this field, which is also not much explored in public publication. As stated in the future work chapter, there is still plenty of things to discover down this path. Some ideas were stated in this master thesis in case we would have had the time. We are more on reflection research for this part of the dissertation.

Thanks to this research, we can also state that for someone that would like to continue this work, the passive vision and U-Net are worth the time for a fast and robust implementation of the system for an enterprise that would like to get involved in the domain.
On the other hand, the active vision paradigm would suit more the academic world, with many possibilities in a complex research field that look promising.
For both paradigms, the reality gaps is an exciting field that can be explored. It is possible to research how to counter it or at least make it have a smaller effect on the fly-controller. We can imagine the capacity of a system being created and tested in a simulation that would be directly working correctly without any flaws nor risks in reality. In this sector, that would be an enormous gain of time, especially during seasons that are not favourable to UAV fly test.

In a nutshell, This work was the first step in possible future researches to see what is likely to be done and how effective it is. There are yet many things that need to be researched. We now have a clear view of the possible paths that can be followed. It only depends on the time given and the interest in science. This master thesis shows that this is possible to complete a full pylon analysis with a fully autonomous UAV.

Bibliography

[1] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[2] Wikipedia black hornet nano as of 4/05/2020.

[3] Wikipedia northrop grumman rq-4 global hawk as of 4/05/2020.

[4] History of drones as of 4/05/2020.

[5] consortiq, a short history of drones as of 4/05/2020.

[6] wikipedia. List of unmanned aerial vehicle applications, 2020.

[7] J. Kwak and Y. Sung. Autonomous uav flight control for gps-based navigation. *IEEE Access*, 6:37947–37955, 2018.

[8] imec. Safedroneware, 2018.

[9] Bruna G. Maciel-Pearson, Samet Akcay, Amir Atapour Abarghouei, Christopher J. Holder, and Toby P. Breckon. Multi-task regression-based learning for autonomous unmanned aerial vehicle flight control within unstructured outdoor environments. *CoRR*, abs/1907.08320, 2019.

[10] Stephen T Barnard and Martin A Fischler. Computational stereo. *ACM Computing Surveys (CSUR)*, 14(4):553–572, 1982.

[11] Bing Han and Xiaoyu Wang. Detection for power line inspection. *MATEC Web of Conferences*, 100:03010, 01 2017.

[12] Van Nguyen, Robert Jenssen, and Davide Roverso. Intelligent monitoring and inspection of power line components powered by uavs and deep learning. *IEEE Power and Energy Technology Systems Journal*, PP:1–1, 01 2019.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[14] Daniel Hein, Steffen Udluft, and Thomas A. Runkler. Generating interpretable reinforcement learning policies using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19, page 23–24, New York, NY, USA, 2019. Association for Computing Machinery.

[15] Leandro F. Cupertino, Cleomar P. Silva, Douglas M. Dias, Marco Aurélio C. Pacheco, and Cristiana Bentes. Evolving cuda ptx programs by quantum inspired linear genetic programming. In *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11, page 399–406, New York, NY, USA, 2011. Association for Computing Machinery.

[16] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[17] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley Sons, Inc., USA, 1st edition, 1994.

[18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[19] Melanie Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

[20] François Chollet et al. Keras. `https://keras.io`, 2015.

[21] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[22] Bruna Pearson, Letizia Marchegiani, Samet Akcay, Amir Atapour Abarghouei, James Garforth, and Toby Breckon. Online deep reinforcement learning for autonomous uav navigation and exploration of outdoor environments. 12 2019.

[23] G.C.H.E. de Croon. *Adaptive active vision.* PhD thesis, Maastricht University, jan 2008.

[24] Richard Held and Alan Hein. Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56:872–6, 11 1963.

[25] Weiss I. Bandyopadhyay A Aloimonos, J. Active vision. *International Journal of Computer Vision*, 1:333–356, 1988.

[26] Kirk Y. W. Scheper and Guido C. H. E. de Croon. Abstraction, sensory-motor coordination, and the reality gap in evolutionary robotics. *Artificial Life*, 23(2):124–141, 2017. PMID: 28513202.

[27] J Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986.

[28] N. Kiryati, Y. Eldar, and A.M. Bruckstein. A probabilistic hough transform. *Pattern Recognition*, 24(4):303 – 316, 1991.

[29] Ken Shoemake. Quaternions. `http://www.cs.ucr.edu/~vbz/resources/quatut.pdf`.

[30] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[31] Guillaume Maitre. Unet-v1.