Reinforcement learning for inspection path planning of linear infrastructures

Martin Jánošík - 201911252@post.au.dk

Master's Thesis,

Department of Electrical and Computer Engineering

January 4, 2022

Advisor: Rune Hylsberg Jacobsen, rhj@eng.au.dk

AARHUS
UNIVERSITY
DEPARTMENT OF ENGINEERING

Abstract

The purpose of this work is to examine the possibility of Reinforcement learning for inspection path planning of linear infrastructures with a single unmanned aerial vehicle. The work is applying this technique to the power grid inspection. Current power grid inspection techniques are gathered and presented. Through knowledge gained from the literature, a proof-of-concept is developed and tested in the simulation. The simulation is performed in a developed learning environment that is reproducible by other researchers, by following a standard during the development. The validation of underlying Reinforcement learning algorithms is performed on three scenarios with various levels of complexity. The results are presented in the final part of the document for each scenario comparing the following Reinforcement learning algorithms Q-Learning, Sarsa, Deep Q-Learning. The analysis of the results provided useful insight into the importance of correct algorithms and rewarding systems in Reinforcement learning.

Contents

Acknowledgment

I would hereby like to express my gratitude to my supervisor Rune Hylsberg Jacobsen from Aarhus University, for his patience and guidance. Furthermore for the opportunity of working in such an interesting and challenging area. I have learned a lot in the process of working on this project.

1 Introduction

1.1 Motivation

The motivation of the project is to make an autonomous path planning for an inspection of linear infrastructures applying reinforcement learning using a depth camera. Reinforcement learning will allow using drones that are not programmed but taught to behave in a specific manner. This behavior can cover more edge cases than regular programming and is able to provide more precise navigation in various environments. It can be observed in current trends that Reinforcement Learning is gaining popularity in various industries ranging from traffic control to autonomous cars which are sensing the whole environment around them.

Reinforcement Learning, enables machines to solve more complex decision-making tasks with small prior knowledge. Reinforcement learning has the ability to extract and learn abstraction from data. This is especially helpful when there is a lack of dataset, which is required when performing supervised learning. This is enabling to solve tasks by inputting data from various sensors in a high dimensional state, which was difficult a few years ago [1]. For example, reinforcement learning is used in autonomous car navigation which is a complex task with many outputs from the environment such as pedestrians, signs, lines, or other traffic. Simple techniques fall short due to a lack of ability to adjust to the dynamic environment which is often changed in the real world. The reinforcement learning is better at generalizing the objectives.

This project will demonstrate the path planning on the power grid elements. The reinforcement learning would allow faster and cheaper inspections of the power grid elements which are vital for almost all individuals and businesses.

Currently, our society is heavily dependent on electrical energy. Electricity is used on daily basis in every developed country. This presents many challenges tied to distribution through the electric power grid in order to deliver energy without any interruption causing blackout. These blackouts are becoming more frequent with the aging of the current power grid. It is already seen in Europe and the US, power cuts are becoming more and more frequent [2]. The blackout can be lasting from a few minutes up to days. This can cause many disturbances in the economy due to businesses being unable to operate or stopping their production. The power grid needs to be monitored and inspected in order to prevent the above-mentioned blackouts.

Once the drone is learned to fly along with the cables this behavior can be simply extended to other power line elements. The autonomous flying should be performed with a single UAV, configured as a quadcopter. UAVs and quadcopters are used interchangeably in the report. The UAV should be able to autonomously fly to the position of the pilot with further ability to recognize a power line and continue to another

pilot alongside the cable while keeping the cable in the field of view. This project will also provide quantitative and qualitative comparisons of multiple reinforcement learning methods stated below.

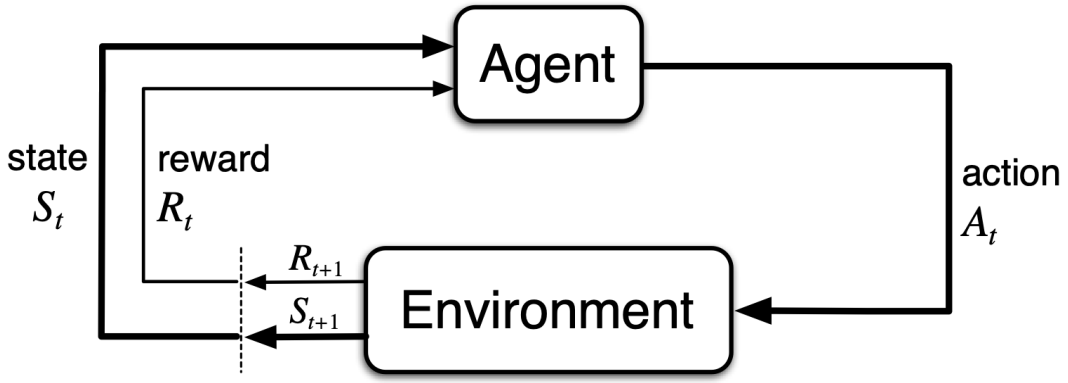


Figure 1: The agent-environment interaction in a Markov decision process from Richard S. Sutton et al. [1]

More detailed information related to the functionality and the implementation of the underlying algorithms is mentioned in the subsequent sections. The picked algorithms are one of the simplest, but with big differences under the hood providing an interesting benchmark.

1.1.1 Q-learning

The Q-learning reinforcement learning method is providing the agent with the capability to learn optimal acting in the given domain. The agent is continuously improving with trying an action in a particular state and evaluates its consequences against the set of rules returning reward or penalty. During the selection, of the next action, the Q-learning does not use the behavior policy. Q-Learning will converge to a solution that is optimal under the assumption that, after generating experience and training, it switches over to the greedy policy [3]. This switch will result in picking a correct action over the exploration.

The Q-learning reinforcement learning method is suitable for tasks where training performance is irrelevant and can result in dangerous behaviors. Usually, it is used in a simulation or in a controlled environment.

The details about the Q-Learning algorithm are presented in the implementation section 4.10.

1.1.2 Sarsa

The Sarsa reinforcement learning method is also providing an agent which is continuously improving in the given environment. Contrary to the Q-Learning, Sarsa is using behavior policy which allows the method to generate experience in the environment. Sarsa will converge to a solution that is optimal under the assumption that we keep following the same policy that was used to generate the experience. In order to converge, we need to incorporate some element of randomness into the policy.

The Sarsa reinforcement learning method is suitable for tasks where training performance is relevant contrary to the Q-learning. The agent should be performing well in the learning process. This is important in the case of learning on the expensive robot or in the environment where the agent can cause some damage.

The details about the Sarsa algorithm are presented in the implementation section 4.11.

1.1.3 Deep Q-Learning

The Deep Q-Learning reinforcement learning method is also providing an agent which is continuously improving in the given environment. If a problem is in a low-dimensionality it is reasonable to store and use Q-values for each state and action. The problems are raising in more complex problems where the use of the neural network as an approximator to the Q-value can be useful. The Deep Q-Learning is built on top of the Q-Learning algorithm and is sharing similar details.

The details about the Deep Q-Learning algorithm are presented in the implementation section 4.12.

1.2 State of the power grid inspection

This section provides an overview of current power grid inspection techniques that are considered state of the art. The general inspection methods will be presented in the following sections. This section provides a summary of current inspections and their underlying advantages and disadvantages.

Nowadays the power grid inspection is normally carried out by workers walking near power lines or using helicopters in case of inaccessible or remote locations. This approach has not been changed in past years. [4] The Inspection usually provides useful insights about future steps with the power line. This could prevent blackouts if measurements are taken in place. Problem is that the inspection is expensive and time-consuming. It is far more practical to use unmanned aerial vehicles (UAVs). While an autonomous inspection can be done with a UAV using a magnetometer that is sensing

a magnetic field of alternating current (AC) [5]. This solution is not working when the inspection is performed on direct current (DC) which is used to power railways or during the power outage.

In the following sections, the current means of the inspection are presented with underlying advantages and disadvantages.

1.2.1 Foot patrol inspection

The foot patrol inspection is highly accurate if the power line elements are visible from the ground. The inspection is usually performed by two inspectors with the aid of binoculars, in some cases with the infrared (IR) camera or corora detection camera [6] utilizing the Ultraviolet (UV) light spectrum [7].

The inspection is usually monotonous, slow, tedious, and suffers from the subjective opinion of the inspectors. Other disadvantages are the requirement to have powerlines in the line of sight and variety of the terrain which can cause problems for the inspectors to move. In case of bushes or hard terrain under the inspected powerline, the inspection can be delayed or even unable to perform. Furthermore, the given terrain can cause an endanger to the inspector's life [7].

1.2.2 Land vehicle aided inspection

Another land-based inspection is performed with the aid of the vehicles. Usually is a regular car in case of easily accessible terrain for example powerline alongside the road. The inspection can be also performed by All-Terrain Vehicles (ATV) in case of rough and unpaved roads. The vehicles are containing mapping sensors that are mounted on top of them. The typical sensors are camera, laser scanner, GPS) [7].

This type of inspection is providing almost similar accuracy as foot inspection but with the additional advantage of increased inspection speed. Another advantage is reduced risk and workload on inspectors. The disadvantages are also similar to the foot patrol since this inspection is also requiring the line of sight and suffering from the different terrains. In case of an inaccessible road along the powerline, this inspection can be paired with the foot patrol inspection [7].

1.2.3 Climbing robots inspection

Another type of inspection is performed with the aid of climbing robots. Usually, the climbing robots are able to move along the conductors and overcome various obstacles during the inspection on the power lines. The biggest advantage of this inspection is the ability to inspect lines that are not in the line of sight from the ground. Another

benefit is inspection within the proximity of the powerline which is greatly increasing the accuracy.

The disadvantages of this inspection type are increased engineering and technical effort due to many interferences and electromagnetic fields emitted from the power lines [7]. Also, the climbing robots need to be specifically designed and constructed for a specific type of tower and powerline [8] which often renders them unusable on other types. If the climbing robot encounters a new type of obstacle during the inspection it is often unable to overcome it.

1.2.4 Helicopter-assisted inspection

The inspection performed from the helicopter is usually requiring three personnel on the board. The pilot, inspector, and the cameraman operating the camera. The helicopter-assisted inspection is often used when the coverage areas are large or when there is a need for a quick inspection in order to prevent human life hazards. This inspection is providing advantages of higher speed and accessibility in comparison to the previous methods [7].

The disadvantages of this inspection type are the higher cost associated with the operation of the helicopter. Another big disadvantage of this method is the low detection rate and effectiveness of the inspection [7].

1.2.5 Satelite image inspection

The satellite image inspection is performed on the images gathered using the satellite sensors capturing visible and Near-InfraRed wavelengths. The objective of this inspection is usually to monitor vegetation alongside the power line. The advantage of this process is the ability to cover larger areas [4].

This method of inspection is highly dependent on the weather because the clouds and rain can obscure the view rendering images unusably. There is also a higher cost associated with gathering images but the ability to cover large areas is decreasing this cost [9].

1.2.6 UAV inspection

The inspection performed using the unmanned aerial vehicle (UAV) is bringing many advantages over all the previously mentioned methods. This is also the reason for increased interest in the usage of UAVs over past years from researchers. Currently, UAVs are used or researched in various activities ranging from monitoring the power lines to inspecting pylons [7]. The UAVs can be equipped with various sensors such

as cameras (color, stereo, IR), radars, GPS, .. etc. This flexibility allows performing various operations required to inspect different parts of the power grid. Contrary to the climbing robot [8] the UAV does not suffer from the need for adaptation for different types of power grid objects in most cases.

The biggest advantage of the UAV is reduced risk to human life and decreased cost. Contrary to the land-based inspection the drone can overcome the inaccessible terrain and gain a position within the line of sight. Investigation performed in the work of D Jones et al Requirements for aerial inspection of overhead electrical power lines [10] reported that the usage of the UAVs is would perform similarly or better than helicopter-based inspection while maintaining low cost and low risk to the human life. Furthermore, the inspection could be faster than the foot patrol. The performance of the UAV inspection can be heavily impacted by the choice of correct sensors and construction of the vehicle, which is still a great challenge to achieve [7].

The UAVs are suffering from similar disadvantages as the helicopter related to the nature of inspection from the air. The flying vehicle is affected by the wind and requires camera stabilization, a skilled drone operator to perform an inspection. The need for a skilled drone operator can be removed if the drone is able to sense and track the object of interest. This is the central focus of this work. Furthermore due to the limitations of the sensors the UAV is able to inspect and detect only the most visible conditions on the power grids.

This category can be further split into rotor-based UAVs, using motors to stay in the air, or wing-based UAVs. Each category brings its advantages and disadvantages. The rotor-based UAVs are suffering from limited speed and flying time due to batteries. But they offer a big benefit of the ability to hover one spot and great maneuverability contrary to the wing based which needs to move all the time to stay in the air and bigger turning radius. The biggest benefits of wing-based UAVs are speed and flight time. They are suitable for tasks when there is a large area to cover without the need to perform tight maneuvers as mentioned in [11].

1.3 Problem statement

The objective of the project is to examine the feasibility of reinforcement learning on the UAV in order to perform flight close to the power grid. This task is challenging due to various reasons. The power grid elements are often in unknown environments or surrounded by obstacles. The UAV should decide on the next action based on input from the sensors and perform the next action autonomously. The flight in close proximity would enable autonomous inspection of the underlying elements such as pylons, lines, insulators. The UAV should be able to perform flight in an unknown environment.

The challenges in the project are the training of a drone in a simulated environment performing autonomous flight while maintaining the viewing angle on the power grid

elements which are changing due to the movement of the UAV. Another challenge is to perform autonomous flight without prior knowledge about the environment and placement of power lines. Choosing the correct reinforcement learning algorithm is crucial due to many constraints on the onboard computer such as memory or computational power. The research questions are formed from the challenges and they need to be addressed in order to develop a proof-of-concept. The proof-of-concept will help with the assessment of the different reinforcement algorithms in comparison to the task complexity. The task complexity in reinforcement learning is often changed by the amount of action which can be performed by the Agent or by increasing or decreasing the amount of the observations from the environment. The research questions are as follows:

- Which reinforcement learning model is most suitable for autonomous UAV navigation with the specific goal to inspect power grid?
- How does task complexity impact various reinforcement learning models?
- What are the current limits of different reinforcement learning methods?
- What are the computational requirements to perform reinforcement learning methods?
- How to simplify complex data from drone sensors to perform simulation and learning faster?

These questions are the main focal points, the project will attempt to address.

This project is mainly aimed at the Reinforcement learning part in connection to navigation in unknown surroundings and safely navigate UAV alongside power grid to enable data collection in future work.

1.4 Method

The project will start out with literature research on power grid inspection to gain insights into the current state of the industry. This information should provide a foundation for a better understanding of the underlying problems in the inspection. The current inspections will be assessed with respect to their advantages and disadvantages. After the inspection research, the navigation and reinforcement learning models for autonomous drone navigation will be explored. Reinforcement learning algorithms should be chosen based on the research paper's analysis and quantitatively summarized. The various chosen reinforcement learning algorithms should be used to train drones to

perform autonomous flight in close proximity to the power grid. The performance of the algorithms should be analyzed and quantitatively summarized.

Then a concept of the system for power grid detection and autonomous navigation will be designed. The concept is utilizing the onboard camera together with GPS and down-facing sonar to sense the environment, which will be then used with a chosen reinforcement learning algorithms to provide instructions to the UAV regarding its next movement. The reinforcement learning should be able to utilize the perceived environment and interpret in order to take the next steps which in this case will be a movement of the UAV.

The proof of the concept will use Robotic Operating System and PX4 or SJTU flight controller. Time will be spent on learning how to use and integrate Reinforcement learning into the Robotic Operating System and PX4/SJTU. The implementation of underlying technologies will be executed upon completion of the concept. The implemented proof of the concept will enable reinforcement learning model training and provide information in terms of accuracy and performance. The implementation provides a demonstration of how the system should work in the practice and contribute to the power grid inspection research.

The implementation is based on the research questions with prioritization for the power line detection using an RGB-D camera The development will use open-source packages to aid with the development and speed up the process. Another important part of the project is to develop a 3D simulation environment of the inspected power grid. This simulation environment will enable rapid testing of proposed solutions and reduce time and costs. Real-life testing is heavily affected by external conditions such as rain or wind. Moreover, during the real flight, the hardware might be damaged due to failures in the system which will result in the project delay and extra costs. The simulations should provide useful data that validate the feasibility and functionality of the project. The performance and functionality will be assessed and discussed. The simulation will be optimized to reduce the computational requirements and reduce training time. The optimization should be assessed and discussed. The drone itself is not performing any data analysis in terms of inspection.

2 Related Work

This section provides an overview of current detection techniques performed on various power grid elements, that are considered state of the art. Firstly, the power lines detection methods will be presented in the following section 2.1. This section provides a summary of related work focused on detecting power lines using various methods. Next, the sections 2.2 and 2.3 presents the summary of the detection methods for a insulators and pylons. Lastly, the summarization of Reinforcement learning used in combination with UAV will be presented in section 2.4.

2.1 Power line detection

The power line detection problem is covered by a considerable amount of research papers. Research papers are published from two distinct research communities focusing on aircraft navigation and safety and the second one is focusing on power line inspection. The UAV requires both use cases during the flight. To perform safe navigation and inspect power lines. The underlying issue is the size of the powerline in the gathered image which is not more than 3 pixels in an image in case of visual detection. [12].

There are many research papers focusing on detecting lines using vision sensors, or infrared sensors, contrary fewer available research papers are focused on the usage of microwave or magnetic sensors. Power line detection is usually compromised of multiple stages beginning with building an image containing a set of line segments. The next step is to further filter the line segments based on geometric properties. The texture is often not usable due to a lack of details. The above is the base algorithm for line detection. Since then, many researchers improved this in various ways.

During early adoptions of the power line detection, the Yan et al. in Automatic extraction of power lines from aerial images [13] introduces a new method to detect the powerlines. Previously the Hough transform was used, but the algorithm is consuming a lot of memory and detecting many false positives from objects which appear as a line on the image such as roads. The usage of Radon transforms with the addition of Kalman filter to merge edge detection allowed them to detect more of the 90% of total power length.

The early algorithm adoptions using pure transformations methods are heavily affected by the complexity of the background. To overcome this issue, the neural networks were proposed in Towards automatic power line detection for a UAV surveillance system using pulse coupled neural filter and an improved Hough transforms from Zhengrong Li et al. [14]. The comparison and proposed methods are shown in figure 2.

According to the work of Joshua Candamo et al. in the Detection of Thin Lines using Low-Quality Video from Low-Altitude Aircraft in Urban Settings majority low flying aircraft in the United States army such as helicopters are more often lost due to power

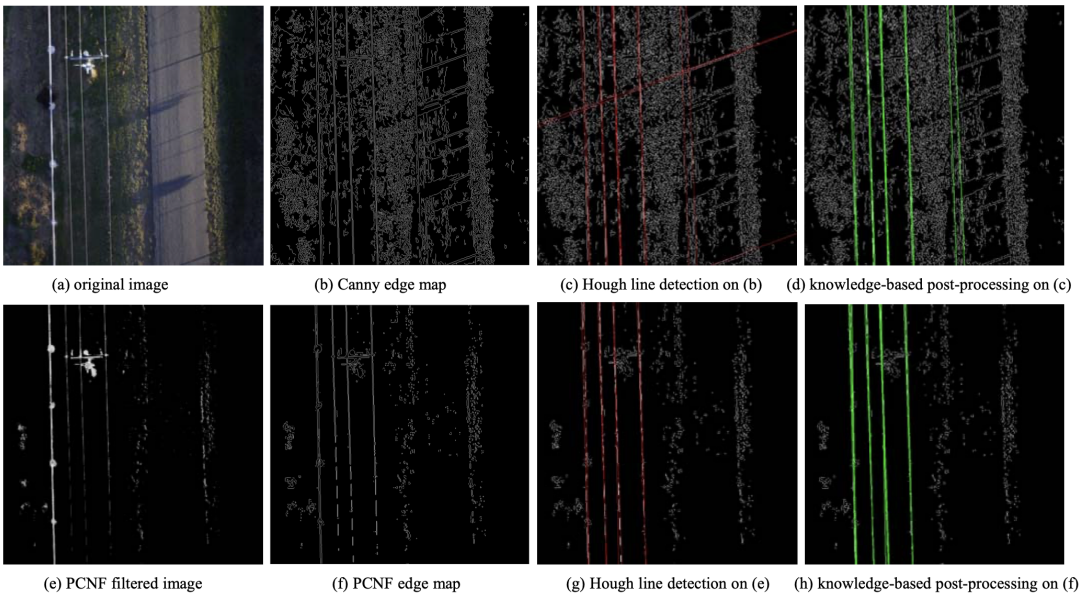


Figure 2: Comparison of power line detection results from Zhengrong Li et al. [14]

lines than during the combat. The work intrudes usage of the low-quality video sensor to detect power lines during low-altitude flights. Their algorithm was used in a heavily cluttered environment. In order to assess, the algorithm performance the researchers prepared a dataset for testing and validating. The dataset is containing artificially created pictures of power lines or wires used on hanging bridges with additional noise and background such as clouds or buildings, trees, etc. The resulting algorithm utilized the time continuity property of the video, meaning that if the power line is detected in one frame it should be presented in another frame with some transformation in case of moving aircraft. The proposed solution achieved a 65% detection rate on their dataset with a 25% false-positive rate. The previous related works achieved only 48% with a big amount 75% of misclassified objects as lines. [15]

Another research focused on aircraft navigation and safety is extending the previous work by proposing millimeter-wave radar video in An Algorithm for Power Line Detection and Warning Based on a Millimeter-Wave Radar Video from Qirong Ma [16]. The proposed solution is solving issues with visibility constraints on regular video cameras, which require ideal weather conditions. The previous work of Joshua Candamo et al. [15] is not working when there is heavy rain or fog. The Millimeter-Wave Radar is using a similar algorithm to detect the power lines. The usage of this algorithm with different sensors achieved similar results, but the wave radar is able to sense the DC cables which are emitting Bragg patterns. Researchers adjusted the previous algorithm to provide as many detections as possible regardless of the false positives. The detected powerlines are further categorized into two categories (Real lines, False positives) using Support vector machines utilizing the Bragg patter property. The outcome of this

method is increased detection rate to 99.84% and decreased false positives to 0%. [16].

The work Image Based Visual Servo control for Fixed Wing UAVs tracking linear infrastructure in wind introduced by Steven Mills et al. [17] investigates the accuracy of Image Based Visual Servoing controlled by a feed from a downward-facing camera on the fixed-wing. Their work is focusing on tracking powerlines and adjusting flying paths to mitigate the effect of the wind. Researchers investigated the previous work done to track the object using the interaction matrix which is relating the motion of powerlines within the image plane to UAV motion. The previous works generalized the task to center the feature vertically on the image plane for a down-facing camera aligned with the roll axis of the UAV. They attempt is to mitigate the effects of the wind which can change the course over the ground causing the misalignment with the tracked feature introducing steady-state error. To mitigate this effect the integral control or wind correction angle must be introduced. The integral control is actively controlling the plane using the flaps to reduce the tracking error to zero. This approach resulted in improved tracking capability but caused significant overshoot and introduced bank angle. The second approach used the direction of the wind and simply turned the UAV against the wind which mitigated the effects of the wind and the UAV stayed directly above the tracked feature. Furthermore, this approach avoided overshooting during the correction. [17]

The work of Luis F. Luque-Vega et al. in the Power Line Inspection Via an Unmanned Aerial System Based on the Quadrotor Helicopter [7] explored the topic of power line joints detection and qualitative inspection by designing and building UAV based on quadrotor drone. The UAV used the thermal infrared camera and color camera to detect the power lines joints by overlaying the images and subtracting the background from the image. The joints are increasing the electrical resistance which caused increased temperature. They assumed that the temperature on all joints should be the same. If the temperature is too high it might indicate a faulty joint causing a lot of resistance. The computation was done on the ground station which was communicating with the drone. Furthermore, the ground station provided an interface to control flight parameters. They concluded that the information provided by the thermal and color camera is sufficient to perform a qualitative inspection and detect some common faults.

Alexander Cero'n et al. in the Power line detection using a circle-based search with UAV images [18]. proposed a new method for power lines detection. The proposed method utilized the geometric relationships, which is distinguishable from the previous approaches which often used gradient information. Power line detection is done using the drawing of circles utilizing the computer graphics primitives which removed the need for GPU. The proposed technique is firstly detecting valid points using the Bresenham algorithm [19] and from the valid point the center of the circle is created, then the circle is drawn. The algorithm is detecting pixels on the edge of the circle. The pixels should be symmetrical and form a line through the valid center point, this technique is called

Circle Based Search (CBS). The proposed technique was performing better than the previous Hough approach in terms of detection rate and speed, the CBS finished the task two times faster in the worst-case scenario. The detection rate is was similar to the EDLines in terms of the speed and detection rate [18].

The recent research Survey and Evaluation of Sensors for Overhead Cable Detection using UAVs from Nicolaj Haarhøj Malle et al. [5] provided an overview of the current state of the art sensors that can be used to detect overhead power lines using UAVs. The researchers executed tests on low to high voltage cables and were looking for cable distance estimation as a measurement from multiple sensors. The sensors used in this test are using various technologies to sense the environment.

mmWave sensor - Millimeter-wave (mmWave) is a special class of radar technology that is transforming microwave echo signals into electrical signals. The radar system transmits the short-wavelength electromagnetic waves which are then reflected by objects in the path. The reflected signals are captured and translated. The radars can determine the velocity, angle of the objects, and range [20].

stereo camera - A stereo camera is simultaneously photographing an object using two cameras usually on the same horizontal axis. This is causing a horizontal displacement in image points due to horizontal translation from the camera. We can estimate depth using the disparity. The disadvantage of the camera is problems with the visibility in the case of low light or bad weather such as fog.

high-resolution ToF - The time-of-flight camera is using the light pulses which are then reflected back in the image plane. The depth estimation is established by the measurement of the round trip of the light signal emitted from laser or LED. The advantage of the ToF is the simplicity and compactness of the device in comparison to the stereo camera. In comparison to the Lidar, there are no moving parts. The disadvantage of the ToF is the issue with the background where the emitted light might be suppressed by surrounding light sources, this is the case only in the case of using visible light in the sensor. Another problem might be interference from other laser pulses published from different devices.

Lidar - An acronym for the Light detection and ranging, emitting light in the pulses to measure ranges. Lidar is more complex and precise than the ToF camera. Contrary to the ToF it is building point clouds from emitted pulsed laser signal. ToF is creating depth maps based on the reflected light, usually using an RGB camera. Another benefit of the Lidar is better readability of the point cloud contrary to the produced depth map from ToF [21]. Lidar and ToF are often interchanged due to a similar principle.

analog magnetoresistive sensor - The magnetoresistive sensor is changing the internal resistance by the presence of the external magnetic field. The sensors have

generally wide usage from consumer electronics to the automotive industry. The sensors are used to detect the position, speed, and direction of magnetic fields or objects emitting a given field. The analog sensor is usually outputting a 0 to 5-volt signal. The sensors are small and able to provide precise measurements [22].

digital magnetoresistive sensor - The digital magnetoresistive sensor is working on the same principle as the analog but the output is usually from 0 to the inputted reference voltage.

(a) mmWave sensor for UAV

(b) stereo camera

(c) ToF for UAV

(d) Lidar sensor for UAV

(e) analog magnetoresistive sensor

(f) digital magnetoresistive sensor

Figure 3: Most commonly used sensors on the drone to detect structures or obstacles.

They tested the distance measurements from 5 meters and concluded that all sensors overestimated cable distance but provided relatively consistent performance. The worst result was from the mmWave sensor with a mean error of 0.123m and the best was the Lidar sensor mean error of 0.025 which is an important measurement useful for our project together with the stereo camera, which performed with a mean error of 0.085m. The tests are recorded and available to replay, together with a created dataset of overhead lines images [5].

According to the [12] no one attempted to detect power lines from multiple angles. Many research papers focused on detecting the powerlines from one specific viewpoint. The creation of the before-mentioned algorithm would expand the potential use of UAVs.

Method	Detection rate	False positives
High quality camera using Radon transform in [13]	90%	N/A
Low quality camera using Thinned SUSAN in [15]	79%	60%
Low quality camera using Non-thinned SUSAN in [15]	86%	40%
Low quality camera using Canny not preprocessed in [15]	80%	59%
Low quality camera using Canny preprocessed in [15]	88%	41%
Millimeter-wave radar [16]	99.84%	0%

Table 1: Overview of notable power line detection methods from previously stated works.

2.2 Insulators detection

The insulators detection topic is not broadly examined as powerlines due to a lack of interest from the aircraft community. The insulators are placed on pylons and pylon is more simple to detect than an insulator, due to different materials of the difference in the construction materials and size. The insulators are examined mostly in terms of inspection. There are fewer research topics focused on detection.

The work of M. Arastounia et al. in the Automatic extraction of insulators from 3D Lidar data of an electrical substation [23] explored the possibilities of automated methods to create a 3D model of the electrical substation with the LIDAR scanner. The Insulators detection and extraction is one of the key objectives. In order to reduce the size and computational requirements the first step is to remove the ground points using histogram gradient. Then the dataset is further split into a set of objects. as shown in figure 4. This can be done due to the nature of the repetitive pattern of objects in the electrical substation. The next step is to identify the insulator in the given subset. This task is performed by a new proposed approach to determine the direction and distribution of neighboring points. The radius is an important variable that can massively affect the end result of the detection. With the extracted points the distribution direction is compared using the principal component analysis which is further compared with the unique characteristics of the insulators. The proposed use of principal component analysis reduced computational requirements and achieved 98% accuracy in insulator detection. The undetected insulators were highly occluded and

many important features were missing.

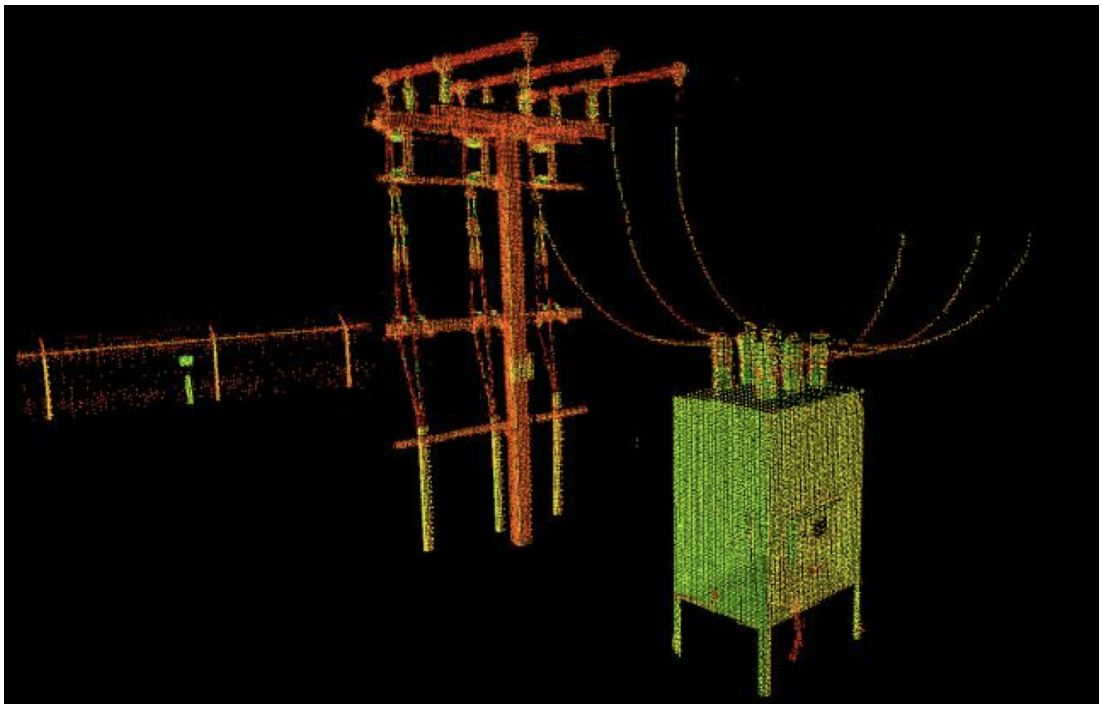


Figure 4: The subset of Lidar dataset from M. Arastounia et al. [23]

2.3 Pylon detection

Pylon detection is also not a broadly researched topic due to the interest in the power line inspection instead of pylon as mentioned in [24] and [25].

In the work of Jittichat Tilawat et al. Automatic Detection of Electricity Pylons in Aerial Video Sequences [25], the researched topic was to automatically keep the gimbal-mounted camera focused on the pylon during the manual inspection from the manned helicopter. This task was performed manually by the inspector controlling the camera gimbal through the joystick which can lead to human errors. This is also required to fly with the inspector onboard which caused risk and cost. The researchers proposed to use straight-line detection to detect the power lines and then search for a pylon. The achieved result with this method was 97% true positive detection and 40% false detection rate. The false detection rate is not causing issues due to detection in close proximity to the pylon [25].

Another research Towards Autonomous Detection and Tracking of Electric Towers for Aerial Power Line Inspection from Carol Martinez et al. [8] focused on pylon

detection and tracking using the neural network to classify if the image contains the pylon and isolate pylon from various backgrounds. This model was trained on four types of pylons. The dataset used for training was created from inspections carried out using a manned helicopter. Once the pylon is detected in the given image frame the tracking algorithm calculates the estimate of movement in the image plane. This approach is increasing computational efficiency because tracking is a simple task when compared to detection. Once the tracking algorithm is not able to provide another estimation, the detection will be used as a fallback. With this approach, researchers achieved approximately 90% true positive detection of the pylons [8].

2.4 Reinforcement learning on UAV

The work of the Iker Zamora et al. in the Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo [26], introduced OpenAi gym for robotics. The OpenAI gym is a useful toolkit for reinforcement learning. The researchers implemented this toolkit into the ROS package ultimately providing a standard for robotics. This enabled broad usage of reinforcement learning in a standardized way with comparable results. The implementation was used on the simple turtle robot with only 3 actions (forward, left-right) using Q-Learning and Sarsa. The results were compared between Q-Learning and Sarsa. They also implemented a simple model for the UAV but it was not benchmarked in their work.

According to Motion Planning by Reinforcement Learning for an Unmanned Aerial Vehicle in Virtual Open Space with Static Obstacles from Sanghyun Kim et al. [27] the reinforcement learning is capable of performing autonomous flying in an open space with static obstacles. The researchers implemented a proximal policy algorithm to perform motion planning for the UAV. The experiment is performed in the Gazebo simulation with the ROS and OpenAI Gym. The UAV used a depth camera for sensing the environment together with the information about the angle relative to the goal pose and distance from the goal pose. These pieces of information acted as observations for the reinforcement learning resulted in actions that can be moved forward or turn left or right (Yaw). The trained model performed well with the 81% goal rate.

In the work of William Koch et al. Reinforcement Learning for UAV Attitude Control [28] the novel idea of using reinforcement learning as PID control for the UAV. The Proportional-Integral-Derivative (PID) control is proving stability during a flight. The PID control is evaluating sensor data and controls the motors accordingly via motor controllers. The PID control tries to mitigate the oscillations or other unwanted movements chased by the external factors. The researchers implemented Reinforcement learning to create an intelligent PID controller able to mitigate effects from more unpredictable and harsh environments. The resulting PID controller decreased 44% in wrong actions and reacted from 1.15 to 2.5 times faster on various axes. The

Reinforcement learning used various models from the OpenAI Baselines project which is providing algorithms for the researchers.

3 Concept / System Design

This section is providing a high-level overview of the proposed solution on the topic of reinforcement learning for autonomous power lines inspection on the UAV. Furthermore, the section presents and describes significant hardware and software components needed to fulfill the task. The foundation of reinforcement learning is also presented in the section to provide a basis for the implementation together with the introduction of OpenAI Gym standardization.

3.1 System overview

The focus of this work is on creating a standardized learning environment for performing flight in proximity to the power grid elements. The work is also focusing on optimizing the learning environment to save resources and time during the learning procedure. The learning environment is created with reusability and tries to provide a general solution that can ultimately result in the usage of a trained model on various UAVs with different sensors. This can be performed under the assumption that the given UAV can provide required observations from the environment such as horizontal distance from the cable, global coordinates, current height, etc. The trained model is not affected by the method of gathering the information which allows fast comparison between multiple sensors in the real world. A figure 5 presents a high-level overview of the learning system architecture which is comprised of simulation part, robot controller, and then reinforcement learning. More detailed information about each section will be presented later. All the underlying components are connected to each other and communicate using Robotic operating system topics, this will be explained in section 4.1.2.

The figure 5 shows only the software part of the learning environment because during the initial learning no hardware is used, only simulated models generating data from a simulated world created in Gazebo. Furthermore, the basic data interaction between the different major components is also presented.

In the next sections, all the major components will be presented and described with additional details starting with the depth camera which is used in this experiment to detect a distance from cable. The sensor for detecting can be changed since the learning environment is not aware of the sensor type. The task of a depth camera or any other sensor capable of detecting distance from the cable is to detect a distance of the UAV from inspected cable. This sensor should be mounted on the front of the UAV. The depth camera is returning images and point clouds which can be utilized to calculate the distance from cable across two axes, horizontal distance and vertical distance.

The next sensor which is utilized in the learning model is down-facing sonar, which is providing a distance from the ground. This information is important to maintain

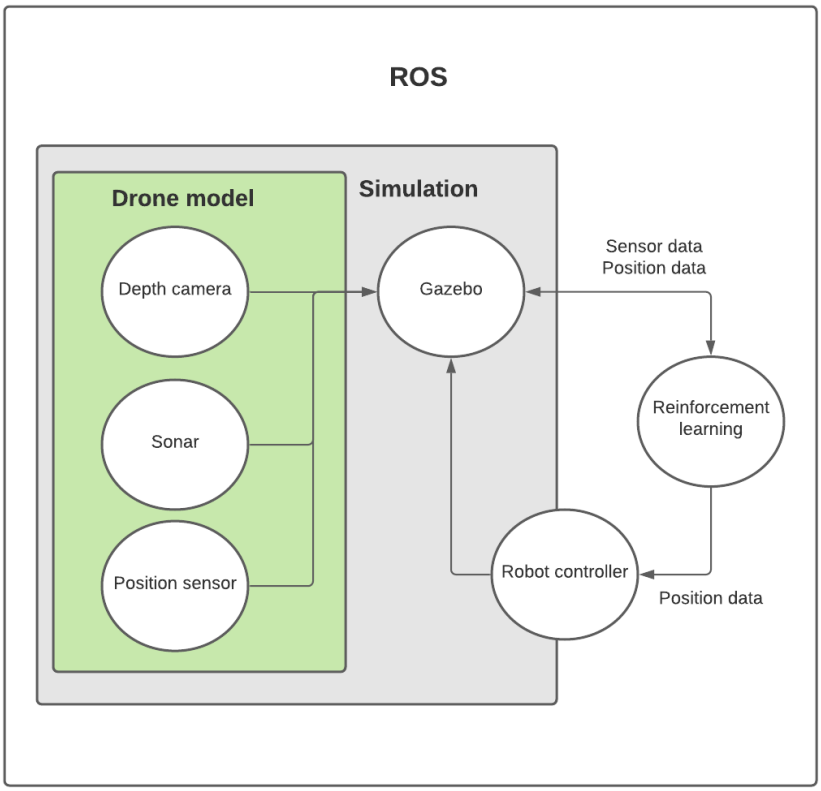


Figure 5: The high level overview of the learning system

height above the ground and prevent the UAV from flying too close to the ground or other objects such as vegetation. The learning model is also not aware of the type of the sensor. The learning model should receive only the distance from the ground. This information can be gathered from other sensors such as GPS.

Lastly, the position sensor is providing the required information about the global position of the UAV. This information is crucial together with information about starting point and endpoint which is usually the pylon. The position information is used in the learning environment to help a UAV decide which direction to go and maintain the UAV within the predefined geofence. Furthermore, this information is detecting the goal of the task which is to fly safely along with the power line elements to the end position.

All the produced pieces of information from the sensors in the simulations are then gathered in the reinforcement learning component and the UAV is deciding on the next action based on the observations. The next action is then sent to the robot controller which is then updating the state of the model in the simulation. The section 3.3 is providing more details about the reinforcement learning component and mentions all the underlying functionality.

The underlying system will be decentralized and distributed which is beneficial in terms of scalability and a faster environment, due to being able to shut off parts of the system or restart them independently. The project focuses on the development of the learning environment without the emphasis on sensor data collection and means of cable detecting and tracking. The learning model will be used on various tasks starting with simple ones such as moving only on one axis and ending with difficulty where the UAV needs to maintain all the 3 axes autonomously., using different training algorithms. All the results will be discussed and summarised providing insight into the usage of reinforcement learning on the UAVs. The inspection task is outside of the scope of this project.

3.2 Reinforcement learning

There are three main categories of machine learning techniques which are supervised, unsupervised, and reinforcement learning. Supervised learning is trying to predict a target value or categorize based on the data with the label in the training part. The unsupervised learning technique is not having the labels in the training data and the algorithms are trying to find patterns or detect anomalies.

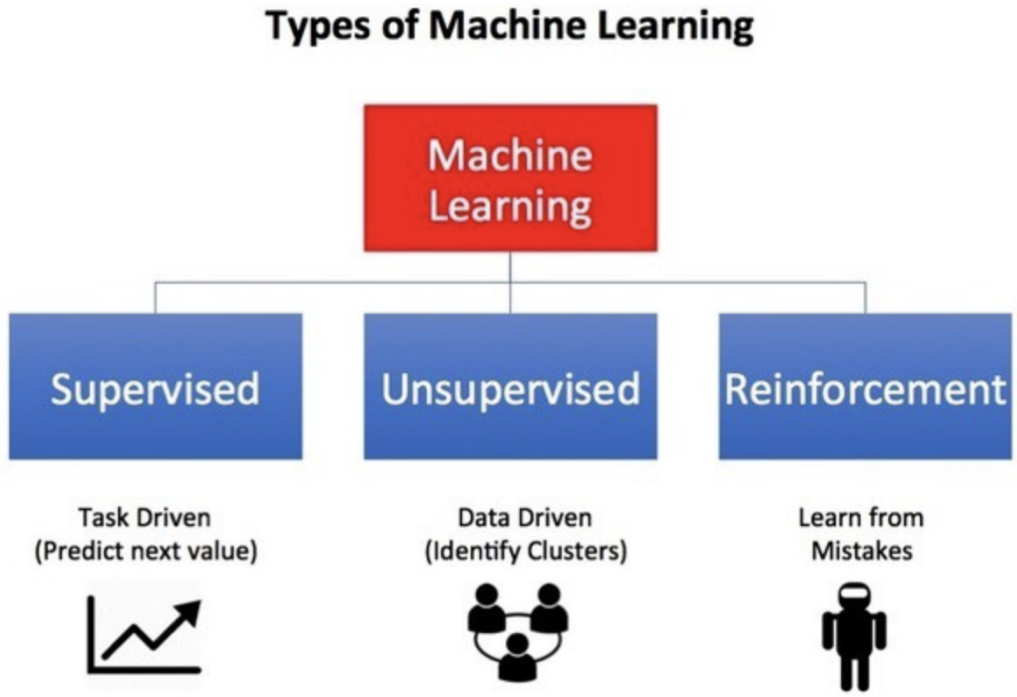


Figure 6: The division of the machine learning techniques from Reinforcement learning 101 [29]

Reinforcement learning is another specific category of the machine learning technique that enables an agent, UAV in terms of this project, to learn in the interactive environment by trial and error using feedback from its own actions and observations from the environment. The agent is rewarded or punished for his actions which simulate positive and negative behavior. The reinforcement learning algorithm is similar to the optimization where the task is to find the best methods to earn the maximum reward. Reinforcement learning takes place when the calculation of the exact solution for the goal is hard or complex.

The reinforcement learning technique is furthermore divided into model-free and model-based learning. Model-free learning purely relies on the sample from the environment and predicts the next state from experience memory. The algorithms which are using model-free learning are for example Q-learning, SARSA, Monte Carlo Control, and more. On the other hand, model-based learning is using a model of the problem which can provide predictions of the next state and rewards. This technique suffers from the inability to regularise the solution and in the case of an inaccurate model, this can cause huge instability and inaccuracy in the resulted agent. This project is focusing on model-free learning which is suitable for real-world problems.

Key terms used in Reinforcement learning:

Enviroment - Simulation or physical world where the agent (UAV) operates, this is in context of this work a simulation in Gazebo

State - Current simulation of the agent, in the context of this work the state contains the current position, goal coordinates, distances from the ground, line, and desired position.

Reward - Positive or negative feedback from the environment.

The Reinforcement learning technique is suffering from the exploitation and exploration dilemma, in other words, should the agent pick optimal decision and assume that current knowledge is reliable? Or should the agent pick decision which is sub-optimal but it will allow to gather new information which could potentially improve knowledge in the future? The general goal is to find a balance between both of the approaches to obtain optimal results. There are many methods to resolve this issue. The most common is an $\varepsilon - greedy$ method where the agent is picking the most optimal action with a probability of $1 - \varepsilon$. ε is the exploration value which is usually decreasing over the time of the learning period. The ε parameter is usually set high at the beginning to enable exploration behavior and lowered in later steps to achieve correct behavior [30].

3.3 Open AI Gym

According to the survey performed by Monya Baker in 1,500 scientists lift the lid on reproducibility [31] more than 70% of researchers failed to reproduce experiments from another work or scientists. Furthermore half of the researchers are not able to reproduce their own experiments.

OpenAI Gym is a toolkit which is providing a variety of simulated environments such as Atari games, 2D or 3D physical simulations. Furthermore, the OpenAI Gym is providing standard API to communicate between algorithms and the learning environment. This allows the creation of more general environments and fast prototyping. This also allows other researchers to easily reproduce results and extend work [32].

The OpenAI Gym is providing abstraction only for the environment in order to maximize convenience and allow various types of agents. The agent takes three parameters from environment abstraction as input at each timestamp. These parameters are:

Observation - Environment-specific object which is representing the observation of simulation or physical world, for example, the distance of the UAV from the power line, GPS position.

Reward - the amount of positive or negative feedback from the environment achieved by the previous action.

Done - indicator whether it's time to reset the environment, in our domain it can be the case when the UAV is outside of the geofence, collided with the power line, or the UAV reached its desired position.

Info - diagnostic information for debugging, this information is not allowed to be used for learning.

The OpenAI Gym is also providing a guideline on how to measure the performance of the reinforcement learning algorithm in an environment. The proposed measurements are to measure the final performance which is referring to the average reward per episode and the amount of time required to learn or amount of steps required to finish the task. This provides useful insights into the sample complexity.

Another benefit of the OpenAI Gym is strict versioning for the environment, if the environment is changed the results will be incomparable. The OpenAI Gym requires to accompany all the environments with the version number in specific format *Name-V0* which after a change in the environment should be incremented to *Name-V1*.

Lastly, the OpenAI Gym is providing a monitor for the environment, which tracks the time, steps, and reset events. The monitor functionality can record videos, produce learning curves.

3.4 Firmware

The initially chosen firmware for the UAV in this work is the PX4 firmware [33] which is governed by the Dronecode Foundation to maintain and overlook the direction and standards of the firmware. The firmware is an open-source project backed by world-class developers from industry, academia and maintained by a worldwide community.

The PX4 provides a flexible ecosystem with prebuild libraries for autonomous flying, computer vision. These libraries are interoperable because of shared standards. Another advantage of the PX4 is Software-In-The-Loop(SITL), making it possible to simulate the flight stack on the computer. This is a key feature required in this project. The choice of the PX4 firmware is due to being familiar with the functionality and mature community backing the project. Current usage of the PX4 is wide from the consumers, state-of-the-art research, and industrial applications.

The next firmware used in this is project is SJTU Drone developed by the Shanghai Jiao Tong University which is enabling simulation with various sensors such as downward-facing sonar, IMU, and forward-facing depth camera [34]. The software is also proving a model for the simulation together with the implemented sensors. The model is displayed in the figure 7.

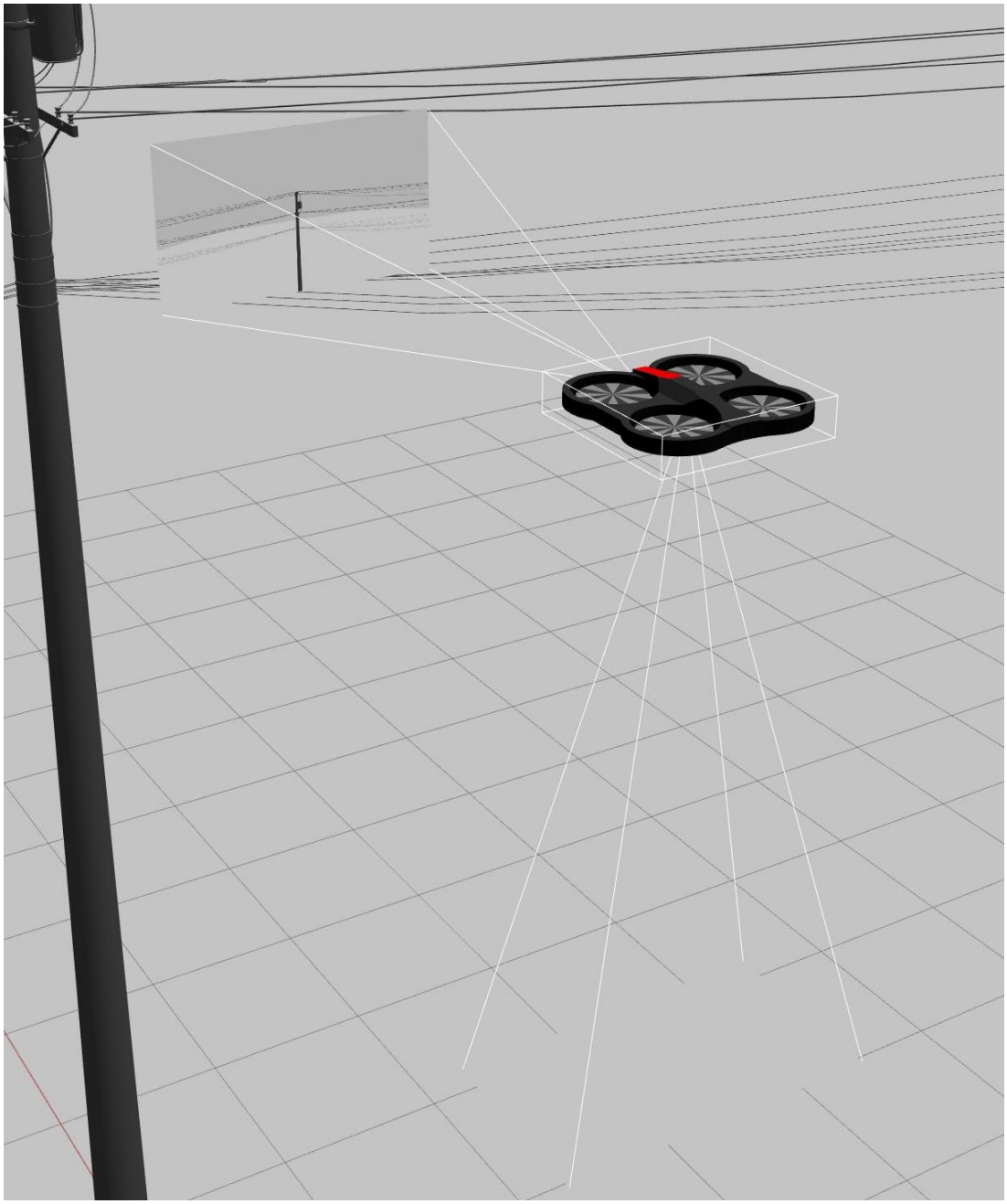


Figure 7: SJTU Drone model with visible RGB-D camera image output and down-facing sonar.

4 Implementation

The following sections serve as documentation for the implementation of the reinforcement learning model for the linear structure utilizing the OpenAI standard. This section also explains the implementation decision and details of the components previously described in the section 3.1 together with the description of learning environments and different tasks for the UAV to perform. Furthermore, the section is proving details about underlying algorithms used in Reinforcement learning.

The implementation is mostly written in Python 3 programming language with dependency on various python libraries and compiling functionality of ROS environment. The python requirements are stated in the Github repository of this project [35]. Some functionality is implemented in C++ which is also dependent on the ROS. The installation of the ROS is performed according to a manual from the PX4 [33] with a small change to use the latest ROS 1 Noetic which is natively supporting Python 3.

4.1 Robotic Operating System

The Robot Operating System (ROS) acts as a meta-operating system that shares some middleware properties such as loose coupling and marshaling by message passing and framework properties by using message passing and utility classes. Furthermore, ROS is providing functionality that is similar to the OS for example hardware abstraction, package management.

ROS is helping to solve hard tasks, problems that are often trivial for humans, but complex for robots. It is a collection of libraries, tools that simplify the process of controlling robots. ROS and the libraries are maintained as open-source by communities ranging from academics to international corporations.

Compared to most other middleware systems and frameworks for robots, ROS imposes little policy on the developer, both in terms of API and license issues. The code reusability is a primary goal of the ROS which often simplifies the development process together with a variety of libraries available to use. ROS is designed to support other robot software frameworks to support many use cases, together with many programming languages such as C++, Python, and more. Another benefit of ROS is its suitability for large runtime systems with large development teams.

The Robot Operating System is chosen as the system architecture for the development of the prototype in this work due to the above-mentioned benefits and large variety of libraries for UAVs and machine learning which are ready to use with great documentation which is simplifying the development.

The main components of ROS1 and ROS2:

ROS Master Responsible for communication between nodes, managing names, and registering nodes and services.

Node A single process running in ROS system. Node is registered at ROS Master before taking any other actions.

Topic Topics are channels over which nodes send (Publish) and receive (Subscribe) messages. Topic names are unique and can be nested in namespaces.

Service Services enables to call node function from another node. A service call is suitable for occasional tass which takes some time to complete. The service is returning some goals which can be for example some reading from a sensor.

Action Actions are also called on nodes and provide similar functionality as service. The only difference is that the feedback is available all the time. For example, if the action is *move UAV to position* the feedback can provide current distance from the goal.

The distribution in the ROS is achieved by using publish and subscribe communication patterns which makes it possible to communicate across different nodes in different languages. The notes are publishing or subscribing messages with specific topics while maintaining a defined message format. The messages are working in the many-to-many communication pattern which can be a limiting factor in some use cases because of lack of reply concept. In order to solve this requirement in some use cases, the services and actions are introduced. The services and actions are operating on a request/reply messaging pattern which can be used for one-to-one communication.

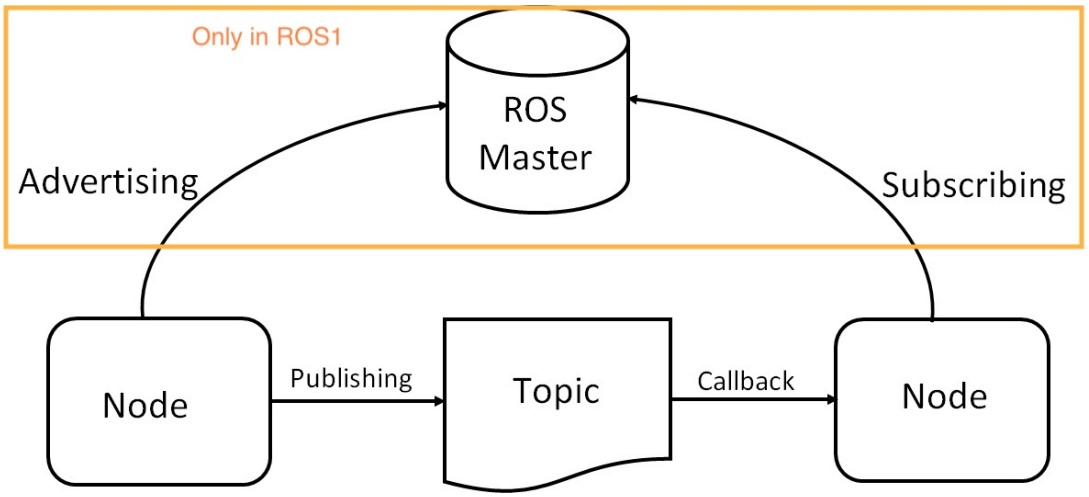


Figure 8: A high level model of ROS system components

4.1.1 ROS 2

Firstly the ROS2 was chosen as the underlying system for the learning environment due to various benefits over the ROS1. The biggest deciding point is native support for a new version of Python 3.5 which is important in order to use new reinforcement learning techniques. Another benefit of the ROS2 change is the message format and transport protocol which is based on DDS standards. This change is allowing ROS2 to provide various Quality of Service policies that can be configured according to specific needs [36].

Quality of Service include configuration for a following set of policies:

History Keep only up to N messages or keep all messages

Depth Size of the message queue

Reliability Best effort (try to deliver a message, but it may fail) or reliable delivery which means that delivery is a guarantee through retrying

Durability Messages are persisted until someone is able to read them or no messages are persisted and only the newest is available

Furthermore, ROS2 removed a need for a ROS master, which is responsible for nodes discovery and handling traffic over the network. This functionality is centralizing the system. In new ROS2 this ability is a move to all the nodes without the need for the ROS master. The ROS2 is containing more changes that are outside of the scope of this project. The high-level comparison between ROS1 and ROS2 is provided in figure 9.

Unfortunately, after the start of the development and implementation of the PX4 Firmware 4.3 and Gazebo simulation 4.2 it was found that plugins for new ROS2 are not mature enough to continue the development of the prototype due to a lack of interest from the community to move plugins to the new ROS2. The crucial component for the OpenAI 4.6 is not available to use and is supporting only ROS1. Implementation of a such complex library would be outside of the scope and is requiring extensive knowledge of Reinforcement learning and the new ROS2 messaging system. The above-mentioned problem was a reason to move back to the older ROS1.

4.1.2 ROS 1

The ROS1 was chosen as the underlying system for the learning environment after the unsuccessful implementation of the ROS2. The ROS1 is no longer releasing new versions and it is reaching the end of life in the latest version which is ROS Noetic. At the time of the writing, the ROS1 was offering more mature plugins and better support for reinforcement learning.

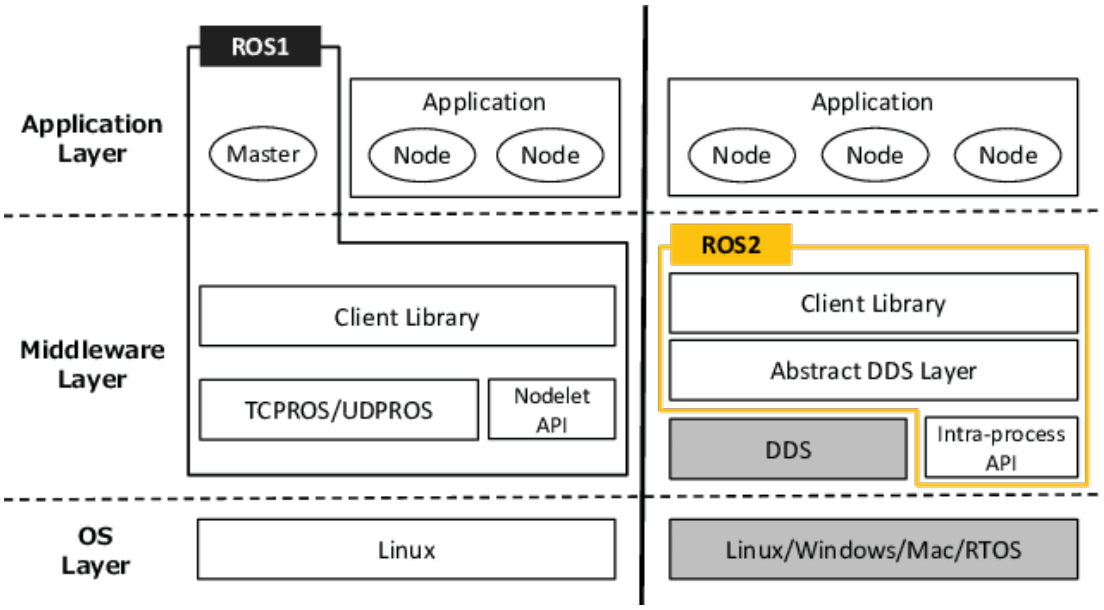


Figure 9: A high level overview of changes between ROS1 and ROS2 from [37]

4.2 Gazebo

The simulation is enabling faster training and removes costs such as hardware required for a UAV construction, hardware damage, or power grid problems related to colliding a drone. Due to these reasons, the simulation is an important part of the project.

The simulation performed in this project is created in the Gazebo, which is suitable for testing computer vision and is commonly used with the Robotic operating system. Furthermore, the Gazebo simulator is recommended to use by the PX4 Firmware. It is a powerful 3D environment simulator for robots, with the ability to simulate collisions, physical interaction between the environment and robots. The accuracy in the Gazebo is achieved by reproducing objects' mass, velocity, friction, and other attributes which enables realistic behavior when objects are held, pulled, or pushed [38].

The dynamic structures in the simulation are composed of rigid bodies connected with the joins. This type of structure is also used in the construction of robots. The joints are moving by applying different forces on them such as angular or linear force. Furthermore, all the objects and aspects of the simulation are changeable, for example, the gravity, time, or lighting. In the case of surfaces the friction, mass, and more. This functionality is enabling rapid development and testing of robot or multi-robot systems. The Gazebo is supporting various sensors and is easily extendable by the wide range of plugins created by the community.

Other alternatives which are proposed and supported by the PX4 firmware are:

JSBSim - The simulator focused on simulating flying dynamic models such are planes, quadcopters, or hexacopters. Furthermore, it can provide a realistic environment together with the wind and realistic flight dynamics [33]. The disadvantage of this simulator is a problem with compatibility and usage in ROS.

jMAVSim - The simulator also focused on simulating flying dynamic models more specific only on quadcopters. The simulator is providing scenarios only to more basic actions such as take off, fly, land, or dynamic fail conditions, such as loss of GPS signal [33].

AirSim - A cross-platform simulator which is providing realistic visualizations in terms of physics and visualization. The disadvantage is high computational resources consumption [33].

The above-mentioned simulators were not chosen due to previously mentioned disadvantages which are not present in the Gazebo. The possibility to extend Gazebo functionality and ease of use with the ROS are the deciding factors.

The simulation environment for this work is created as a new world in the Gazebo, together with the power grid elements as models. Furthermore, the world is easily adjusted to various robot models, which can spawn in the world and be controlled by the scripts which are called "plugins" in Gazebo terminology. The plugins need to define the control for each movable part in the robot. More details about used robot models are provided in sections 4.3 describing PX4 models and 4.4 describing SJTU model. The figure 10 is showing a created learning environment created for the learning without spawned robots.

4.3 PX4 Autopilot in ROS 2 and ROS 1

The PX4 Autopilot is handling all the low-level flight control of the UAV with underlying mechanisms to control motors, peripheral components, and more. The PX4 is utilizing a translation layer between ROS2 and PX4 which is providing a bridge between ROS2 messages and PX4 messages. The bridge is using messages definition which is able to serialize and deserialize messages coming out or into the PX4.

Contrary to the ROS2 the ROS1 is using MAVLink messaging protocol which is designed for the drone ecosystem. The protocol is providing the predefined format of standard messages for exchanging data between various systems such are flight controllers, ground stations, companion computers, and more. MavLink is utilizing a hybrid publish-subscribe pattern with topics and point-to-point patter with the ability to retransmit in order to ensure packet delivery. Another benefit of the MAVLink is the ability to detect packet loss, packet authentication while maintaining low bandwidth

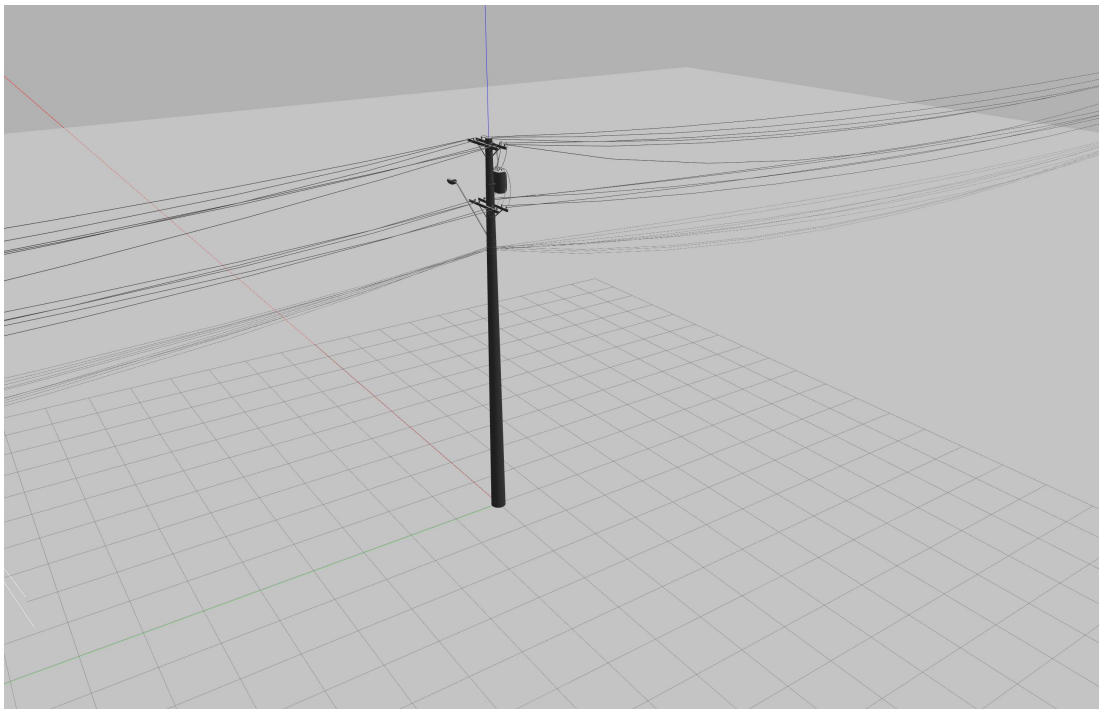


Figure 10: The finished learning environment without the UAV model in Gazebo

which is often crucial for UAVs due to resources limitations. Connection to the ROS1 is utilized through the MAVROS library which is acting as a bridge between MAVLink and ROS1 messages. Figure 11 is showing the communication between various components of the system.

Furthermore, the PX4 Autopilot is providing various drone models with different sensors for the Software in the Loop functionality which will be used in this work. The provided model which is going to be utilized in this work is 3DR Solo (Quadrotor) which is already equipped with the front-facing RGB-D camera to push the messages to the ROS1 or ROS2. The down-facing sonar is added by editing the provided model. The model is provided for a variety of simulators but usage of the Gazebo is strongly encouraged. The PX4 community provided all the necessary libraries to control the drone and translate the topics to the ROS.

After the re-implementation of the PX4 Autopilot with the ROS1, it was discovered that Software in the Loop is not supporting environment reset which is an essential step during reinforcement learning. Reinforcement learning needs to reset the environment due to various reasons which are mentioned in section 3.3. This is not possible due to complex PX4 firmware controlling the motors and sensors of the UAV. If the PX4 detects sudden movements it tries to correct the movements or disabling the whole UAV due to various failures. Due to these issues, another UAV model without low-level flight control needs to be used and implemented.

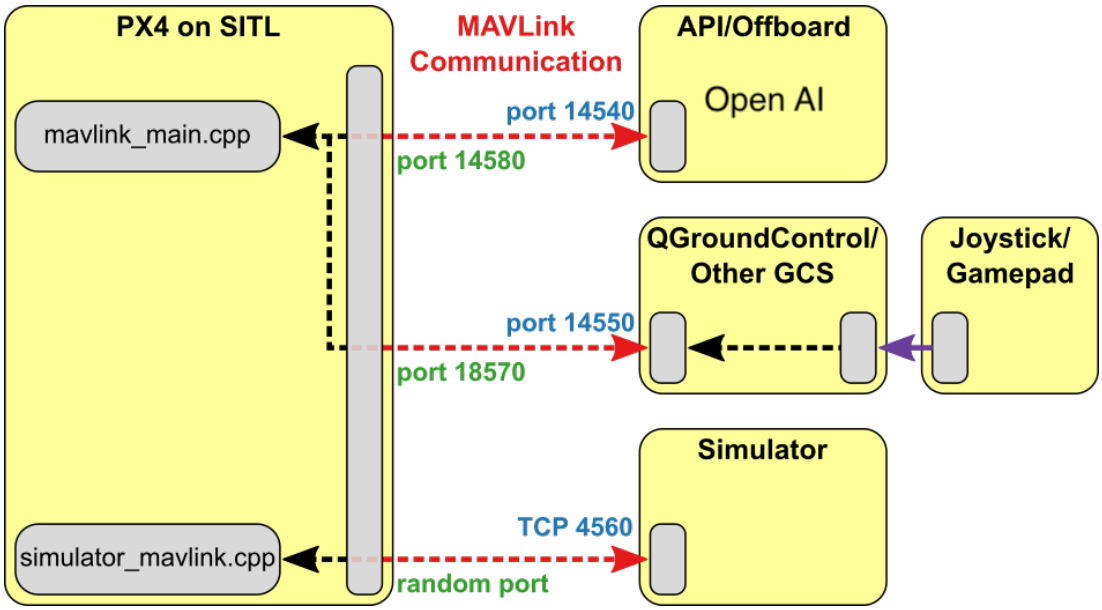


Figure 11: The MAVLink communication between various system components [33].

4.4 SJTU Drone in ROS

The SJTU Drone is a ROS package containing a simulation program for the Gazebo. The choice to use the SJTU Drone package is due to direct implementation in a ROS without any low-level flight control. The drone is simply moved by a command in the gazebo to change a position. SJTU drone is not containing flight controller code or any complex mechanics. This is highly suitable for reinforcement learning because the simulation can be easily reset without a need to reset or change the underlying firmware of the drone. The angle of yaw, pitch, or roll is simply calculated from velocity in a given direction contrary to the PX4 where the angles are calculated from a thrust difference between the motors.

Furthermore, the SJTU Drone contains a noise generator for the current pose which is simulating wind and GPS signal problems. The noise is helping the reinforcement learning to become more robust and should improve the model in real-world scenarios where these types of noises are common. Figure 12 is displaying a high-level overview of the SJTU drone system together with Gazebo and ROS 1.

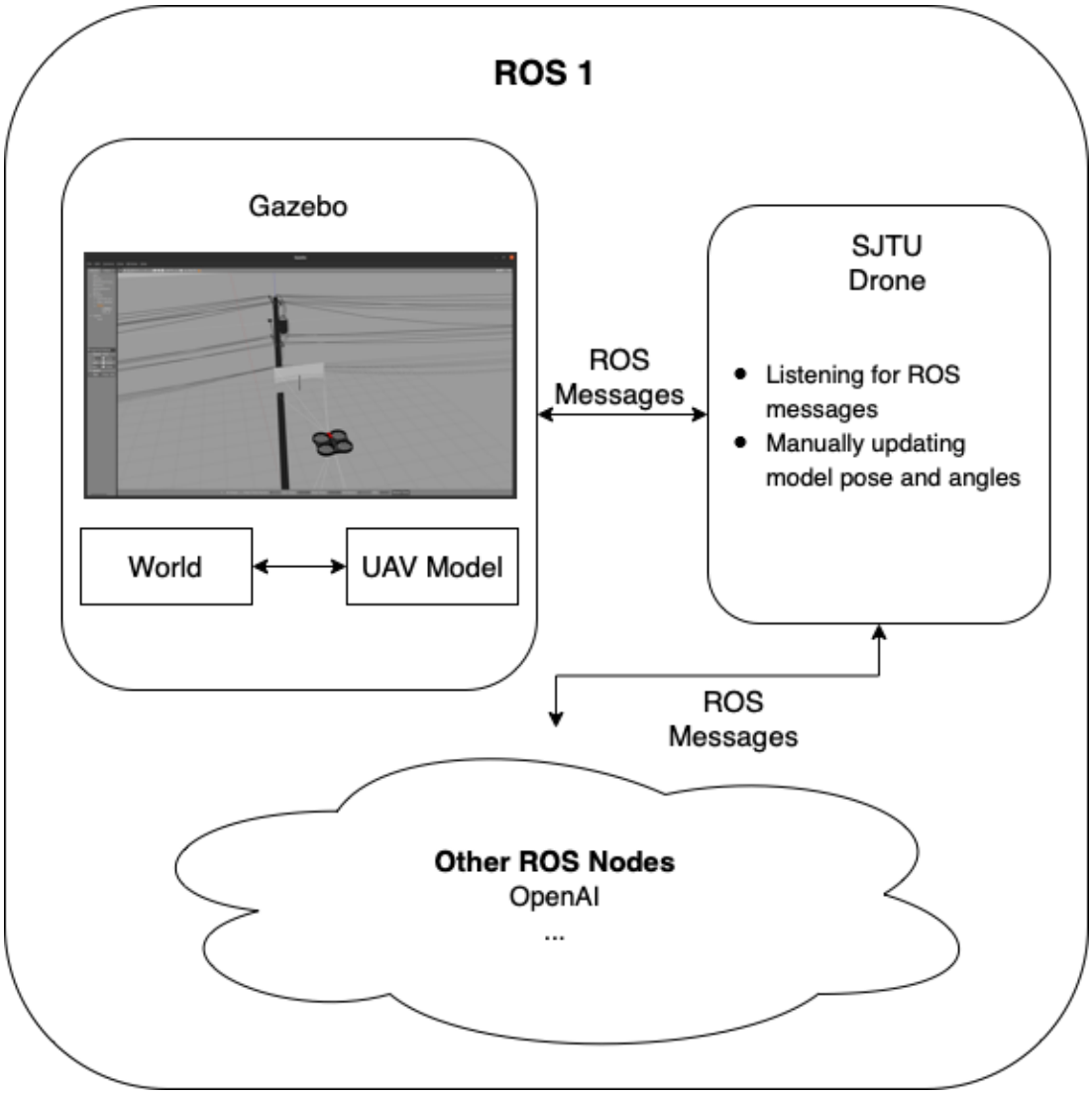


Figure 12: A high-level overview of packages within the ROS 1 using SJTU Drone.

4.5 Simulation and code optimization

After the successful implementation of ROS 1, Gazebo simulation together with the SJTU Drone on the development computer. The development computer is a laptop which means all the components are built for a laptop often being more underpowered. It was ensured that the computer is not overheating which is causing a thermal throttle resulting in lowered computational power.

The development computer is compromised out of the following components and will be serving as a benchmark for this project

CPU - Intel Core i7-7700HQ (2,80-3,80GHz)

Memory - 32GB DDR4 2400MHz

Integrated GPU - Intel HD Graphics 630

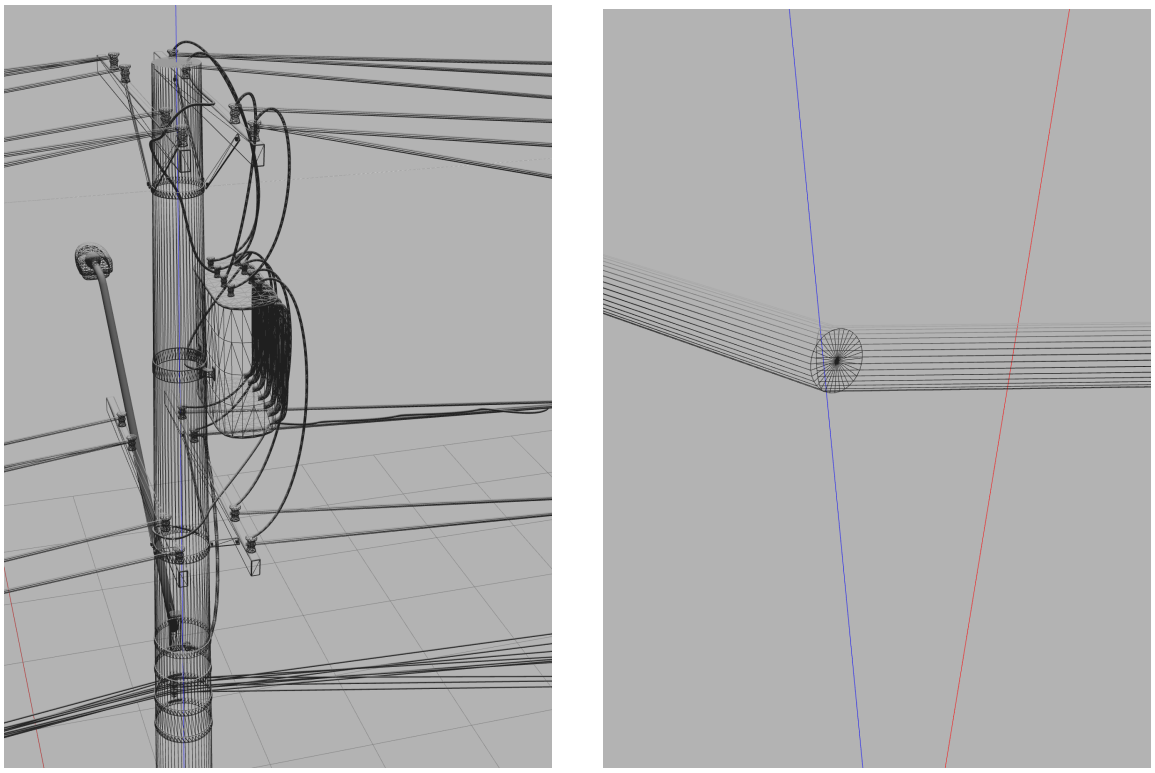
Dedicated GPU - NVidia GeForce GTX 1050 4GB DDR5

Operating system - Ubuntu 20.04 LTS (Focal Fossa)

Due to various issues with the drivers the dedicated NVidia GPU is not used during the simulation and only Intel GPU is utilized.

The simulation was using 45% of CPU resources for the physics engine and ROS messages. The dedicated GPU was used utilized for the 98%. The simulation was able to run at 1 second in simulation per 1 second in real-time with the same limitation by the world constraints. Even without the time limitation, the simulation would not run much faster due to the GPU bottleneck. The first step was to implement a new world with a simplified 3D model of a power line. The original environment is containing 50 000 polygons for the power grid elements. The polygon is a straight-sided shape with 3 or more sides, defined by three-dimensional points and the straight lines that connect them.

The figure 13 is showing a side-by-side view of polygons in an original world containing 50 000 polygons and a new simplified environment with the 200 polygons.



(a) The complex world with 50 000 polygons (b) The optimized world with 200 polygons

Figure 13: Side by side view of polygons in different worlds

The new optimized world reduced the integrated GPU usage by 53% to average usage of 45% while the CPU and memory usage was not heavily affected. The memory usage was lowered only by 200 megabytes and CPU reduced utilization by 6%. Furthermore, the CPU usage needs to be optimized. This can be done by increasing step size which is by the default 0.001 seconds. This means that all the collisions, messages from the UAV sensors are generated every 0.001 seconds. For reinforcement learning, there is no need to have such a granular step and it can be changed to 0.01 seconds.

With this optimization, the simulation is capable to run 61 seconds per 1 real-time second with the maximum utilization of the CPU. After experimenting with the time factor, it was found that the 5 seconds per 1 real-time second is optimal and is not resulting in too much inconsistency in the flying physics of the simulated UAV. Additionally, the faster simulation allows faster learning due to being able to run 5 times faster than a real-time. Another optimization is not using the topics from the camera which is computationally heavy and not necessary for the simulation, because the cable detection is outside of the scope for this project. All the positions are manually calculated from the simulation. Furthermore, the simple SJTU Drone is using fewer resources due to the lack of actual firmware for the UAV. This did not reduce the GPU

usage which stayed the same at 45%, but it reduced the CPU utilization even further to 21% when running the simulation with 1 second per 1 real-time second. The memory usage was not changed but it is not a limiting factor on the benchmarking machine. The disk utilization was also monitored but it was not changing significantly and it was not a limiting factor. The disk utilization was below 5% during the experiments.

Even though the sensors were removed and the precision of the simulation is decreased this should not affect the end results. The increased step size is still precise for the flying needs and UAVs with the companion computers are not able to compute at that speed with real-world data. The removal of the sensors is beneficial due to the increased flexibility of the model which can then be used with the variety of the sensors.

The table below is summarizing all the optimizations steps with the average resource usage.

Optimization step	CPU usage	Intel GPU usage	Memory usage	Achieved time ratio
Complex 3D model and 1:1 speed ratio with granular steps 0.001s	45%	98%	3.5G/32G	1:1
Simple 3D model and 1:1 speed ratio with granular steps 0.001s	39%	45%	3.3G/32G	1:1
Simple 3D model and 1:1 speed ratio with granular steps 0.01s	21%	45%	3.3G/32G	1:1
Simple 3D model and 5:1 speed ratio with granular steps 0.01s	42%	60%	3.3G/32G	5:1
Simple 3D model and Unlimited speed ratio with granular steps 0.01s	99%	65%	3.3Gb/32Gb	65:1

Table 2: The average resources utilization during various optimization steps. All the results are already without the reading from the sensors. The speed ratio of 65:1 means 65 seconds in simulation per 1 real-time second (65 times faster time). All the results are gathered during One-axis training with the Q-Learning algorithm. The simulation was rendered in 4K resolution at 30fps

4.6 OpenAI Gym in ROS

The OpenAI is providing a set of libraries that allows comparing reinforcement learning algorithms on common ground which is called Environments. Unfortunately, the originally build Gym is not supporting the ROS environment with the usage of Gazebo simulation. The work of Iker Zamora in Extending the OpenAI Gym for robotics: a toolkit for reinforcement learning using ROS and Gazebo [26] created a package for a ROS1 which is providing an environment for robotics.

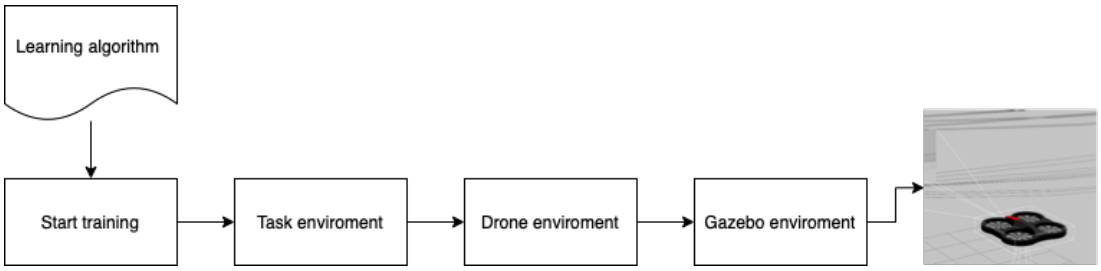


Figure 14: A high level overview of the OpenAI Gym package for the ROS

4.6.1 Gazebo environment

The implemented package is providing a structure to organize ROS code and a connection bridge between OpenAI and Gazebo called *GazeboEnviroment*. The bridge is handling the reset and pause functionality of the simulation when it is necessary during the training. It also takes care of all the steps that need to be done during the training steps or training reset. The class is containing the required OpenAI Gym functionality which is a step, seed, reset and close functionality.

4.6.2 Robot environment

The next item is called *RobotEnviroment* which is in the case of this work called *DroneEnviroment*. The *DroneEnviroment* is handling all the underlying functionality of the UAV. During the initialization, it is checking if the topics which are controlling the drone are available. Furthermore, the topics from the sensors are verified which are providing data from the IMU, front-facing depth camera, and down-facing sonar. The topics are also containing information about the current position and velocity. *DroneEnviroment* is containing functionality to publish position topic, takeoff, and land topics. Ather functionality which is implemented is to read all the data from the sensors and store them.

4.6.3 Task environment

On top of the *DroneEnviroment* the *TaskEnviroment* is implemented which is specifying the usage of the robot for a specific task. In the case of this project, three types of tasks are created. Tasks are furthermore specified and described in the following sections 4.7, 4.8, 4.9.

Generally, the task environment specifies the requirements for the learning environment. The class is specifying which actions to perform and how to perform them. The next requirement is to specify observations resulting from the action and then it specifies how to calculate a reward for a given action. It is also detecting if the current episode is finished.

4.6.4 Training script

The Training script is defining how to set up the learning algorithm and specifies which robot environment to use and what task should be performed. The training script is completely independent of the environment. The algorithm can be changed without the need to modify an environment structure.

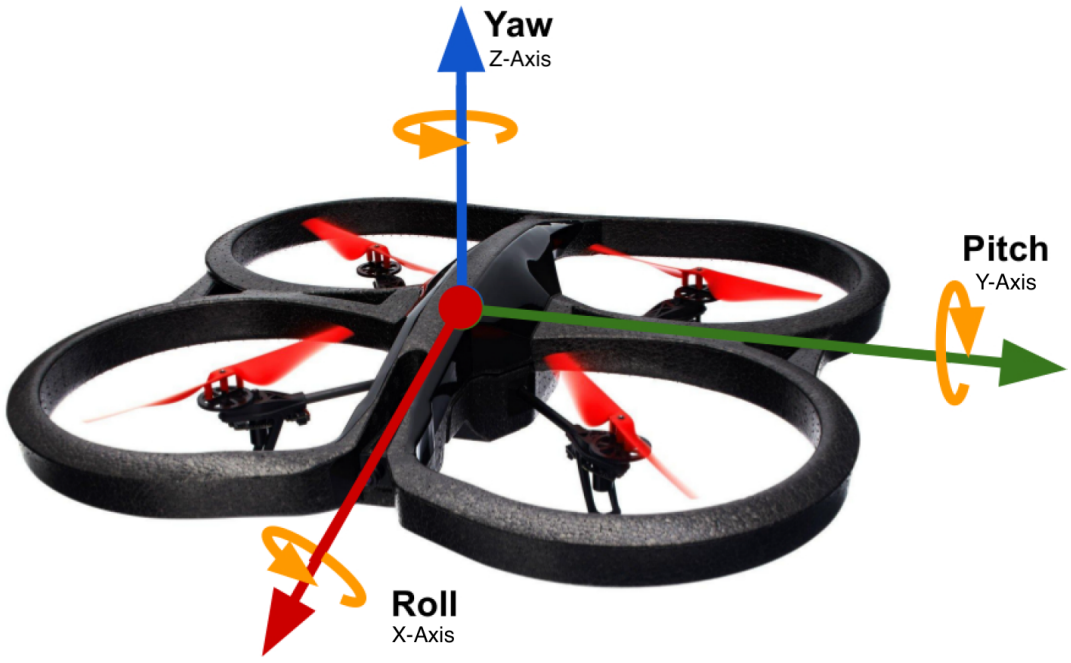


Figure 15: UAV body frame with the local coordinate system displaying the movement axes and rotations

4.7 One-axis movement

The first implemented task for the drone is to perform a movement only across one axis. This is the simplest task in this scenario. The UAV needs to perform three actions: *left*, *right*, and *stay*. Other axes are guarded manually and adjusted periodically without affecting the learning process. The safeguarding is necessary due to noise implemented in SJTU Drone simulating the wind and GPS signal which is resulting in a movement over time in the axis which is not controlled by the Reinforcement learning and causes flying outside of the boundary or movement in a wrong direction in case of yaw noise.

Reinforcement learning is choosing from these steps according to the observations from the environment which are following:

Y axis position - The global position in terms of the Y-axis. The Y-axis in this scenario is affected by the left or right movement.

Roll angle - The angle of the drone when it is moving to the sides. The explanation about UAV axes and rotation movements are displayed in figure 15

Y axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a GPS Coordinate.

The figure 16 is showing which axes are controlled by the reinforcement learning and which are manually safeguarded during this learning task. The goal is the same as in other learning tasks and is displayed in figure 20 as yellow box.

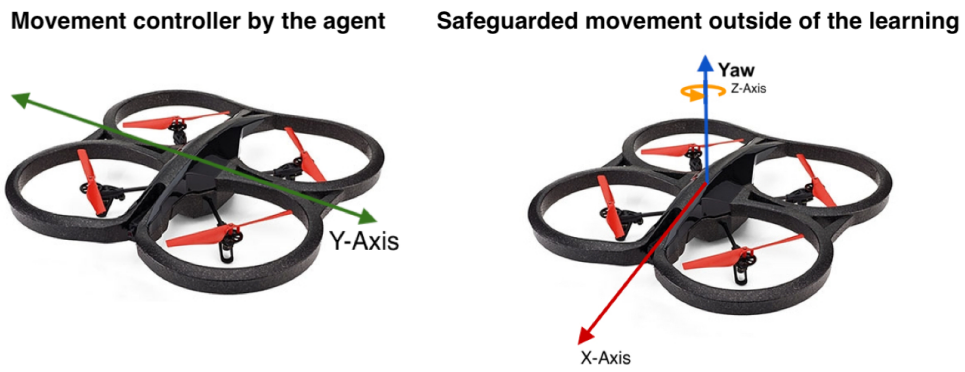


Figure 16: The visualization of axes for the one-axis movements learning environment together with the safeguarded axes.

4.8 Two-axis movement

The second task is to perform a movement across two axes X and Y. This scenario is more complex and should require more training to achieve and steps to achieve a goal. The distance is the same but the UAV now needs to adjust a position in terms of the X-axis which can be changed due to noise in the SJTU Drone. The safeguarding is also required as in the previous case but only on Z-axis and Yaw terms. In this task, there are additional actions that can be performed by the agent. A movement to the front and back.

Reinforcement learning is choosing from these steps according to the observations from the environment which are following:

X axis position - The global position in terms of the X-axis. The X-axis in this scenario is affected by the front or back movement.

Y axis position - The global position in terms of the Y-axis. The Y-axis in this scenario is affected by the left or right movement.

Pitch angle - The angle of the drone when it is moving to the front or back. The explanation about UAV axes and rotation movements are displayed in figure 15

Roll angle - The angle of the drone when it is moving to the sides. The explanation about UAV axes and rotation movements are displayed in figure 15

X axis distance from cable The distance from the tracked object. The distance is in decimal values and in meters. The data can be gathered from any type of sensor.

X axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a GPS Coordinate.

Y axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a GPS Coordinate.

The figure 17 is showing which axes are controlled by the reinforcement learning and which are manually safeguarded during this learning task. The goal is the same as in other learning tasks and is displayed in figure 20 as yellow box.

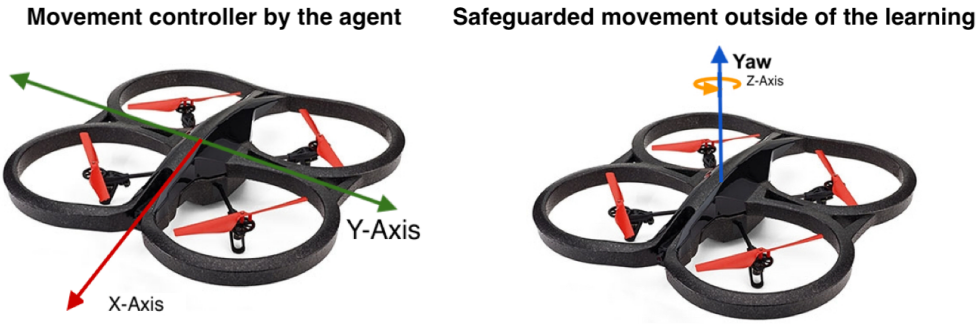


Figure 17: The visualization of axes for the two-axis movements learning environment together with the safeguarded axes.

4.9 Three-axis movement

The last task for the UAV which is implemented is to perform movement across all the axis X-axis, Y-axis, and Z-axis. This is the most complex task during this project and it will require extensive training. The safeguarding is performed only for the yaw rotation, while the drone needs to maintain drifting across other axes while keeping the movement closer to the goal position which is displayed in the figure 20 as yellow box. UAV is performing all the actions except the YAW. This means that Reinforcement learning needs to learn how to achieve tasks using the following actions: *left*, *right*, *stay*, *forward*, *backward*, and lastly *up*, *down* which are introduced in this task.

Reinforcement learning is choosing from these steps according to the observations from the environment which are following:

X axis position - The global position in terms of the X-axis. The X-axis in this scenario is affected by the front or back movement.

Y axis position - The global position in terms of the Y-axis. The Y-axis in this scenario is affected by the left or right movement.

Z axis position - The global position in terms of the Z-axis. The Z-axis in this scenario is affected by the up or down movement.

Pitch angle - The angle of the drone when it is moving to the front or back. The explanation about UAV axes and rotation movements are displayed in figure 15

Roll angle - The angle of the drone when it is moving to the sides. The explanation about UAV axes and rotation movements are displayed in figure 15

X axis distance from cable The distance from the tracked object. The distance is in decimal values and in meters. The data can be gathered from any type of sensor.

Z axis distance from cable The distance from the tracked object. The distance is in decimal values and in meters. The data can be gathered from any type of sensor. If the UAV is below the line, the values are negative.

X axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a GPS Coordinate.

Y axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a GPS Coordinate.

Z axis goal position - The goal point which needs to be achieved. In the real scenario, it can be a height from GPS.

The figure 18 is showing which axes are controlled by the reinforcement learning and which are manually safeguarded during this learning task.

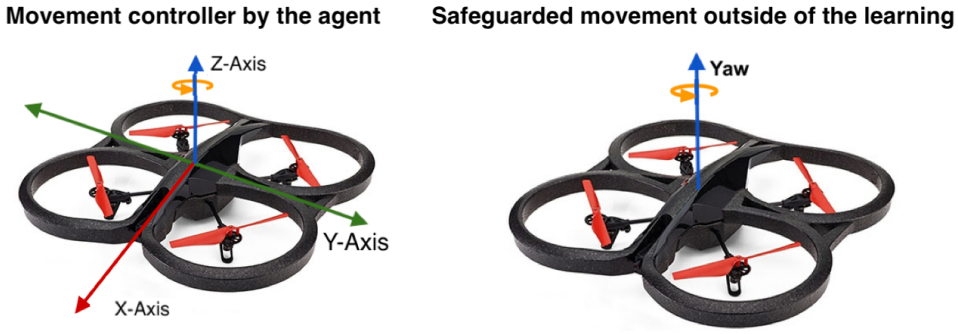


Figure 18: The visualization of axes for the three-axis movements learning environment together with the safeguarded axes.

4.10 Q-Learning

The Q-Learning algorithm presented in 1 is a model-free reinforcement learning algorithm that does not use transition probability distribution. The Q-Learning learns from the experience obtained from each step. Q-learning needs only one step trajectory (s, a, r, s'). Namely, it consists of an agent that will interact with an external environment represented by states. At time t, the agent will be given a reward r after taking an action a in state s. Then the agent will transfer to the next state s'. Using the recursive

operation, we can calculate the expected total reward in the future corresponding to each state and action, $Q(s,a)$, called the Q-value, which is updated using the formula displayed in 1.

$$Q_{(s,a)} \leftarrow (1-\alpha)\cdot Q_{(s,a)} + \alpha\cdot(r+\gamma\cdot\max_{a'} Q_{(s',a')}) \tag{1}$$

The calculated Q-value of each state and action is stored in a Q-table. The Q-learning is $\varepsilon - greedy$ algorithm which means that with the ε probability, the action will take action randomly. This is required in order to find a global optimal and not get stuck in the local optimum. As mentioned previously in 1.1.1 the Q-Learning is off-policy which means the action taken is different from the best action in the current state. This means that the action used to update policy in Q-Leaning is different from the true action.

Algorithm 1 Q-learning: Learn function $Q : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$

Require:
States $\mathscr{S} = \{1,\ldots,n_s\}$
Actions $\mathscr{A} = \{1,\ldots,n_a\}$, $A : \mathscr{X} \Rightarrow \mathscr{A}$
Reward function $R : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$
Black-box (probabilistic) transition function $T : \mathscr{S} \times \mathscr{A} \to \mathscr{X}$
Learning rate $\alpha \in [0,1]$, typically $\alpha = 0.1$
Discounting factor $\gamma \in [0,1]$
procedure QLEARNING($\mathscr{S}$, A, R, T, α, γ)
    Initialize $Q : \mathscr{S} \times \mathscr{A} \to \mathbb{R}$ arbitrarily
    while Q is not converged **do**
        Start in state $s \in \mathscr{S}$
        while s is not terminal **do**
            Calculate π according to Q and exploration strategy (e.g. $\pi(s) \leftarrow \arg\max_a Q(s,a)$)
            $a \leftarrow \pi(s)$
            $r \leftarrow R(s,a)$ ▷ Receive the reward
            $s' \leftarrow T(s,a)$ ▷ Receive the new state
            $Q(s',a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha\cdot(r+\gamma\cdot\max_{a'} Q(s',a'))$
            $s \leftarrow s'$
    return Q

4.11 Sarsa

The Sarsa algorithm presented in 2 is a model-free reinforcement learning algorithm that does not use transition probability distribution. The Sarsa is also learning from the experience and working in a similar manner as a Q-Learning. The difference is

lying in choosing the next action which performed on-policy. This means that Sarsa is calculating the optimal action a' based on the next state s' resulting in $Q_{(s',a')}$ which is used to update Q-value contrary to the Q-learning which is using the maximum Q value. The algorithm might be less accurate because there is less exploration performed contrary to the Q-Learning. This difference is visible in the formula 2.

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma Q(s',a')) \tag{2}$$

Algorithm 2 SARSA: Learn function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$

Require:
Sates $\mathcal{S} = \{1, \ldots, n_s\}$
Actions $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{S} \Rightarrow \mathcal{A}$
Reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
Black-box (probabilistic) transition function $T : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{X}$
Learning rate $\alpha \in [0,1]$, typically $\alpha = 0.1$
Discounting factor $\gamma \in [0,1]$
procedure SARSA($\mathcal{S}$, A, R, T, α, γ)
  Initialize $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ arbitrarily
  while Q is not converged **do**
    Select $(s,a) \in \mathcal{S} \times \mathcal{A}$ arbitrarily
    while s is not terminal **do**
      $r \leftarrow R(s,a)$ ▷ Receive the reward
      $s' \leftarrow T(s,a)$ ▷ Receive the new state
      Calculate π based on Q (e.g. $\varepsilon - greedy$)
      $a' \leftarrow \pi(s')$
      $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (r + \gamma Q(s',a'))$
      $s \leftarrow s'$
      $a \leftarrow a'$
  return Q

4.12 Deep Q-Learning

The Deep Q-Learning algorithm presented in 3 build on top of the Q-Learning removing the limitations in higher dimensionality as mentioned in 1.1.3. The Neural Network is acting as an approximator to the Q-Value and it takes the following form 3.

$$f : R^n \rightarrow R^m \tag{3}$$

The n is the size of the state space and m is the size of the action space. The function takes input s which is the state and outputs the Q-Values for each possible action [39]. The neural network is implemented using Tensorflow and is consisting of 4 fully-connected layers with ReLU activations. The implemented network is displayed in the figure 19.

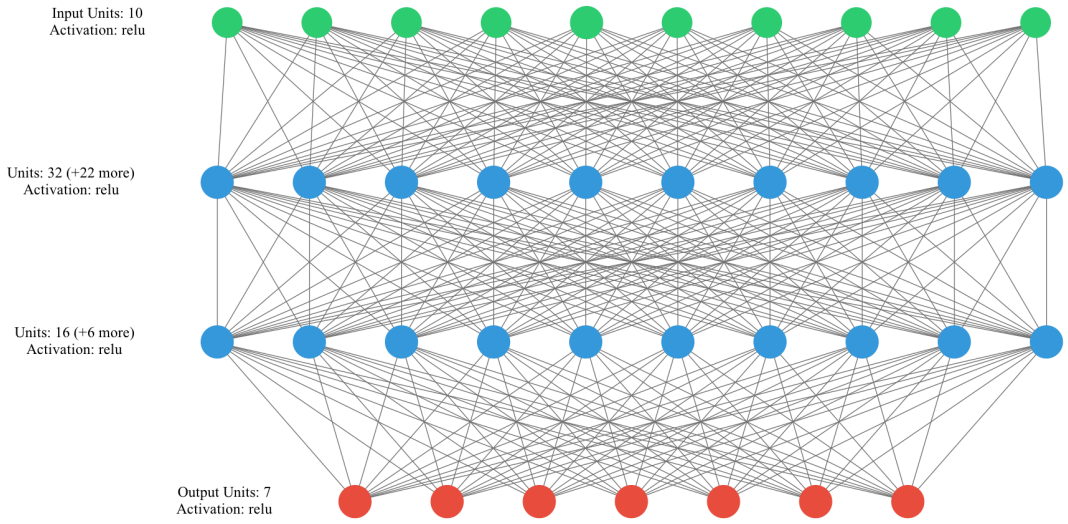


Figure 19: The visualization of the Neural network for the Deep Q-Learning on the Three-axis environment task with 10 observations as input and 7 actions as output.

The Deep Q-Learning can result in inefficient policy in case of a high correlation between actions and weights. This issue can be resolved by using Experience Replay which is similar to memoization which stores the experience tuple and it can recall the rate occurrence to provide better results. The stored results should be pooled randomly and this helps to break a correlation between action and weights.

Algorithm 3 Deep Q-Learning with Experience Replay

Require:

Initialize replay memory $\mathscr{D}$ to capacity $\mathscr{N}$

Initialize action-value function Q with random weights

for $episode = 1, M$ **do**

Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$

for $t = 1, T$ **do**

With probability ε select a random action a_t

otherwise select $a_t = max_a Q(\phi(s_t), a; \theta)$

Execute action a_t in simulator and obtain reward r_t and observations x_{t+1}

Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathscr{D}$

Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathscr{D}$

Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$

Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$

5 Results

This section will go over each of the learning environments and introduce the results of the UAV simulations. The experiments were run in three different learning environments One-axis movement 5.2, Two-axis movement 5.3 and Three-axis movement 5.4. The reward and training parameters were tested only on One-axis and Two-axis environments due to computational and time constraints. The gathered metrics from the environments are compared based on performance from the different algorithms. The results section should provide insights into whether the developed learning environments are feasible to use with reinforcement learning for inspection path planning of linear infrastructures

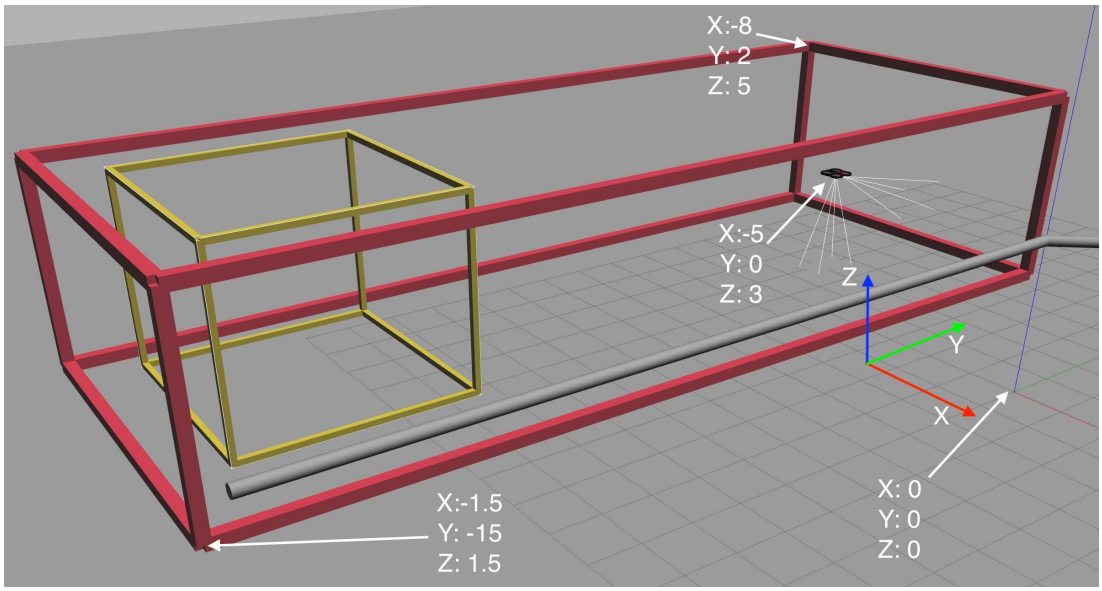


Figure 20: The optimized learning world with the workspace which is acting as geofence displayed in red and goal space displayed as yellow. The UAV is displayed at the starting point.

5.1 Experiment setup

The optimized world is used as a learning environment for the experiment due to computational benefits over the complex environment. The parameters for a world and task are displayed in a table 3. In all the experiments the UAV has a predetermined goal position that needs to be achieved. The position is displayed as the yellow box in the figure 20. If they reach the goal they will receive a reward and the simulation is done for a given attempt. Then the simulation will reset and the drone will start again from a starting point which is also displayed in figure 20.

The movement is guarded by the red box, which is acting as a geofence for the UAV. If the UAV moves outside of the box the simulation is also done and the UAV will receive a punishment for a wrong action by a predefined amount. The rewards and punishments are individual for each task. The reward and punishment are heavily affecting the learned behavior and need to be individually tailored for each different task.

The table below is summarizing all the simulation parameters together with general task parameters.

Gazebo parameters	
Speed ratio	5 : 1
Step size	$0.01s$
Global learning task parameters	
Speed for XYZ axes movement (forward, up, ...)	$0.5m/s$
Speed for Yaw around Z axis	$0.3rad/s$
Initial position of UAV: X	-5
Initial position of UAV: Y	0
Initial position of UAV: Z	3
Boundary box: X-max	-1.5
Boundary box: X-min	-8.0
Boundary box: Y-max	2.0
Boundary box: Y-min	-15.0
Boundary box: Z-max	5.0
Boundary box: Z-min	1.5
Goal box: X-max	-2.0
Goal box: X-min	-7.0
Goal box: Y-max	-10.0
Goal box: Y-min	-14.0
Goal box: Z-max	5.0
Goal box: Z-min	1.5

Table 3: Summarisation of the Gazebo and learning task parameters used in the experiments.

During the experiment, the Deep Q-Learning algorithm parameters were not changing, the parameters for learning are introduced in table 4 and reused in each task together with the other parameters.

Deep Q-Learning parameters (contains also parameters stated above)	
Learning rate for Adam optimizer	0.001
Batch size	0.001
Min Epsilon	0.01

Table 4: Summarisation of the Deep Q-learning parameters used in the experiments. The parameters are merged with global and Q-Learning / Sarsa parameters

5.2 One-axis movement

This section presents the result from the One-axis movement experiment using the optimized learning environment. As mentioned previously it is the most simple task where the drone needs to perform only three actions. This task was mostly used to experiment with different rewards and punishments due to the fastest time to train. The experimenting was used to improve the overall performance and gain knowledge about the effects of the rewarding/punishing system. The knowledge was then used to set rewards and punishment in the more complex task where this type of experimenting is not feasible due to time constraints. Only the most notable experiments are mentioned and displayed in the Appendix A. The first trial was performed with the wrong params which are displayed in the table 11. These parameters resulted in going left and right without reaching the end goal. This behavior is most likely caused by too many steps and rewards for not crashing. The behavior can be observed in the figure 27.

After fixing this issue with the lowering amount of the steps and increasing end reward the Agent figured out that he can move a few times towards a goal and then stop for the rest of the episode without punishment. This behavior can be observed in the figure 28 with the parameters from table 12. This was only a partial fix and the agent was not able to finish the task. The training was interrupted due to observed behavior before reaching the expected amount of episodes.

The next step was to introduce a punishment for stoping for a given amount of steps. These parameters were finetuned during many attempts and the result is presented in table 13 and figure 29. This resulted in the correct behavior. The mean of the steps during the training was reduced from 220 to 32 steps per episode.

There were many attempts to reduce this number even further. The best solution was to introduce a small punishment when the UAV is moving further from the goal. This punishment is also used in the final version named as *Punishment for the movement in a bad direction* which is displayed in the table 6. The final finetuned parameters were

then used for other algorithms and results are presented in the table 5 and figures 21 and 22.

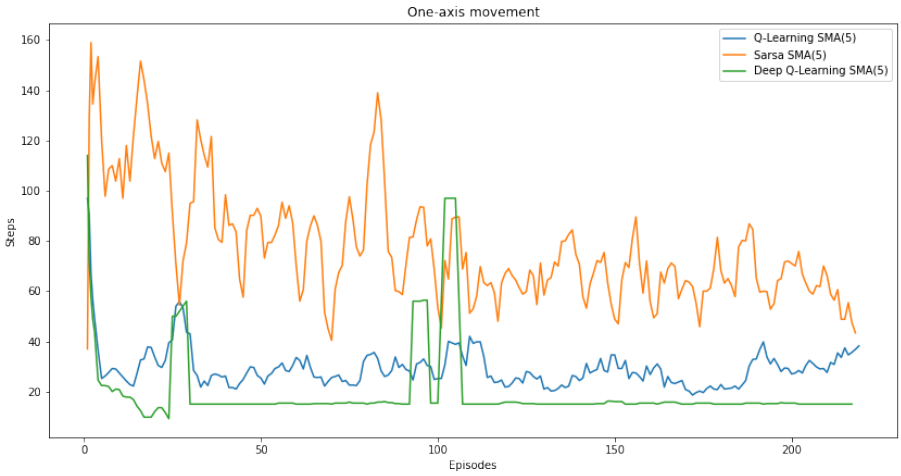


Figure 21: The plot shows Steps per Episode using a Simple moving average of 5 in a final attempt during One-axis movement

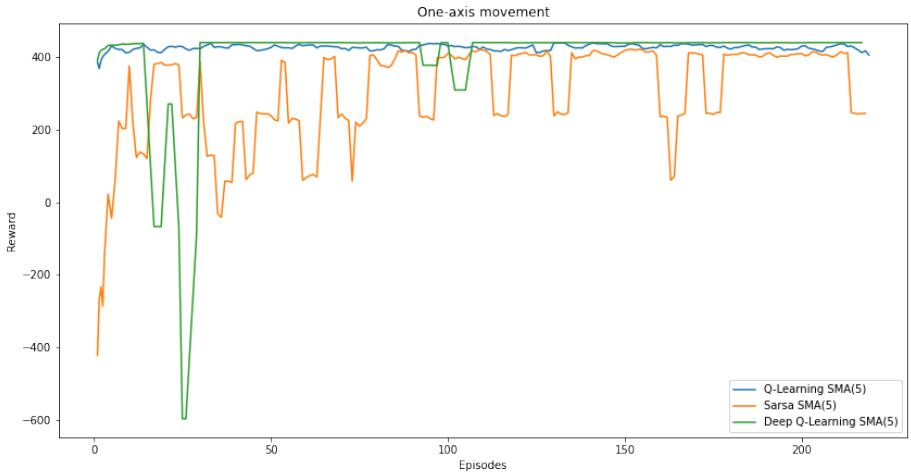


Figure 22: The plot shows obtained Reward per Episode using a Simple moving average of 5 in a final attempt during One-axis movement

The table below is summarizing the results from a final attempt in the One-axis movement experiment

	Q-Learning	**Sarsa**	**Deep Q-Learning**
# of episodes	220	220	220
Average number of steps per episode	29.15	77.58	19.51
Average reward per episode	426.27	305.74	399.60
Average time per episode in a flight time	47s	2min1s	49s
% of reaching a goal	100%	87.73%	95.43%
% of movement outside of a boundary box	0%	10.45%	2.74%
% of running out of the steps	0%	1.82%	1.83%

Table 5: Summarisation of the final results from the One-axis movement experiment.

The table below is summarizing the final parameters for the One-axis movement experiment

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	220
# of steps per episode	220
Learning task parameters	
# of actions (left, right, nothing)	3
Punishment for the movement in a bad direction (futher from the goal on Y-axis)	-1
Maximum consequent stops	3
Punishment for stopping longer than the Max consenquent stops	-10
Reward for movement closer to the goal	5
Reward for not crashing	0
Reward for successfully reaching goal	400
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	-400

Table 6: Summarisation of the final Learning task parameters used in the One-axis movement experiment.

5.3 Two-axis movement

The next experiment is performing a movement across X-axis and Y-axis using the reward parameters from a previous experiment. The parameters need to be adjusted to support the additional axis, but it can be done in a similar manner. The task has also experimented with many types of parameters but the difference is not significant so results are omitted except the final trial. During the training, the simple algorithm *Q-learning* and *Sarsa* were unable to converge to the solution. This is probably caused due to the higher dimensionality and complexity of the problem.

The newly introduced reward and punishment parameter was safeguarding the correct distance from the cable. For example, the correct distance from the cable is $3m$

due to sensor constraints, then the drone will be rewarded by *Reward for the movement in a correct distance from cable* .. if the following formula is met $2 \leq x \leq 4$. In case that the formula is not met the agent will receive punishment without finishing the environment. Furthermore, the amount of steps is increased in order to provide enough time to explore the environment.

The final finetuned parameters were used for all algorithms and results are presented in the table 7 and figures 23 and 24.

Figure 23: The plot shows Steps per Episode using a Simple moving average of 40 in a final attempt during Two-axis movement

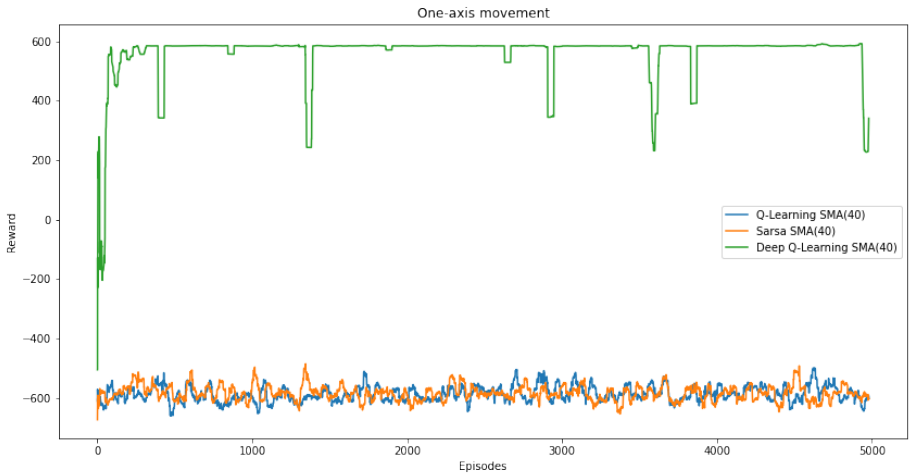


Figure 24: The plot shows obtained Reward per Episode using a Simple moving average of 40 in a final attempt during Two-axis movement

The table below is summarizing the results from a final attempt in the Two-axis movement experiment

	Q-Learning	**Sarsa**	**Deep Q-Learning**
# of episodes	5000	5000	5000
Average number of steps per episode	51.25	48.43	18.24
Average reward per episode	−586.2	−583.72	560.63
Average time per episode in a flight time	1min20s	1min16s	46.8s
% of reaching a goal	1.58%	1.42%	98.94%
% of movement outside of a boundary box	98.42%	98.58%	0.7%
% of running out of the steps	0%	0%	0.36%

Table 7: Summarisation of the final results from the Two-axis movement experiment.

The table below is summarizing the final parameters for the Two-axis movement experiment

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	***5000***
# of steps per episode	***500***
Learning task parameters	
# of actions (left, right, nothing, forward, backward)	***5***
Punishment for the movement in a bad direction (futher from the goal on X-axis and Y-axis)	***-5***
Reward for the movement in a correct distance from cable less than 1m from an optimal distance $5m$ ***in both directions***	***0***
Punishment for the movement in a wrong distance from cable more than 1m from an optimal distance $5m$ ***in both directions***	***-5***
Maximum consequent stops	3
Punishment for stopping longer than the Max consenquent stops	-10
Reward for movement closer to the goal	***6***
Reward for not crashing	0
Reward for successfully reaching goal	***500***
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	***-500***

Table 8: Summarisation of the final Learning task parameters used in the Two-axis movement experiment. The text marked as "***this***" is a newly introduced parameter or a change from the previous experiment

5.4 Three-axis movement

The next experiment is performing a movement across X-axis, Y-axis, and Z-axis using the reward parameters from a previous experiment. Similar to the previous experiment the parameters need to be adjusted to support the additional axis. The experimenting

with different rewards was also performed, but it showed that previously used parameters are more optimal. Therefore there are similar to the Two-axis movement task. This environment is the most complex so the result is similar to the previous case where the simple algorithm *Q-learning* and *Sarsa* were unable to converge to the solution. Their ability to reach a goal is decreased even further as it is presented in the table 9. Also, the previous task showed that even after many episodes Q-Learning or Sarsa will not converge so the training episodes were reduced to 3000 episodes. This is enough for Deep Q-Learning to find a global optimum. The results are summarised in the table 9 and figures 25 and 26.

The table below is summarizing results from a final attempt in the Three-axis movement experiment

	Q-Learning	**Sarsa**	**Deep Q-Learning**
# of episodes	3000	3000	3000
Average number of steps per episode	63.51	49.69	34.1
Average reward per episode	-1352.46	-1231.88	$870,77$
Average time per episode in a flight time	$1min35s$	$1min15s$	$1min16s$
% of reaching a goal	0.9%	1.42%	94.77%
% of movement outside of a boundary box	99.1%	98.58%	4.07%
% of running out of the steps	0%	0%	1.16%

Table 9: Summarisation of the final results from the Three-axis movement experiment.

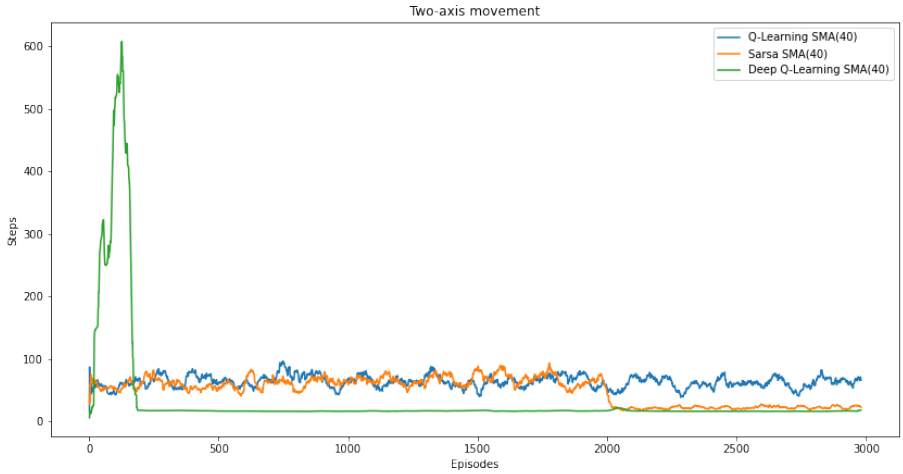


Figure 25: The plot shows Steps per Episode using a Simple moving average of 40 in a final attempt during Three-axis movement

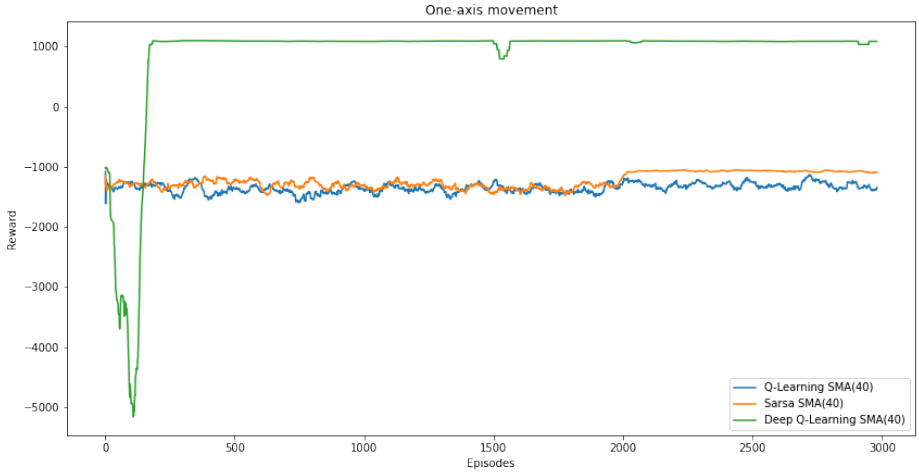


Figure 26: The plot shows obtained Reward per Episode using a Simple moving average of 40 in a final attempt during Three-axis movement

The newly introduced reward and punishment parameter was safeguarding the correct height relative to the cable. For example, the height of the cable is $3m$ due to sensor constraints or field of view in the case of the camera, then the drone will be rewarded by *Reward for the movement in a correct height relative to the cable* .. if the

following formula is met $2 \leq x \leq 4$. In case that the formula is not met the agent will receive punishment without finishing the environment.

The table below is summarizing the final parameters for the Three-axis movement experiment

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	***3000***
# of steps per episode	***1000***
Learning task parameters	
# of actions (left, right, nothing, forward, backward, up, down)	***7***
Punishment for the movement in a bad direction (futher from the goal on X-axis and Y-axis)	***-5***
Reward for the movement in a correct distance from cable less than 1m from an optimal distance $5m$ in both directions	0
Punishment for the movement in a wrong distance from cable more than 1m from an optimal distance $5m$ in both directions	-5
Reward for the movement in a correct height relative to the cable, less than 1m from an optimal height $3m$ ***in both directions***	***0***
Punishment for the movement in a wrong height relative to the cable, more than 1m from an optimal height $3m$	***-5***
Maximum consequent stops	3
Punishment for stopping longer than the Max consenquent stops	-10
Reward for movement closer to the goal	6
Reward for not crashing	0
Reward for successfully reaching goal	***1000***
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	***-500***

Table 10: Summarisation of the final Learning task parameters used in the Three-axis movement experiment. The text marked as "***this***" is a newly introduced parameter or a change from the previous experiment

6 Discussion

The Reinforcement learning experiments are showing that choosing the right algorithm for the task is crucial. The performance of the underlying algorithm is heavily affected by the correct rewarding system which can even cause complete inaccuracy. The experiments showed promising results, performing well during different tasks.

The choice of the correct algorithm and the rewarding system can heavily affect flying time as it is visible during experiments with the simplest One-axis movement environment where the number of steps varied a lot depending on the algorithm and parameters. When a correctly trained model is used in a real-world scenario the benefits are increased mapping coverage using one battery and decreased flight time which is one of the biggest problems on the UAVs. The problem is that there is no library or a tool to choose reward parameters and it is completely subjective to a developer. Due to computational requirements exhausting searching is also not feasible. The proposed work starting from the simplest task is partially solving the problem above. During the learning in the simple task, the developer can observe behavior and logs from the simulation and understand the mechanics behind the decisions. This becomes harder with the more complex tasks.

During the Reinforcement learning of the One-axis movement task, it is clearly visible that the more complex algorithm Deep Q-Learning is faster in reaching the proper estimation of a Q-value. After reaching the value it is maintaining great performance during the rest of the training even with some deviations where the algorithm is randomly trying to assert if there is no other global optimum. On the other hand, the Sarsa took the longest time to find global optimum, and by the look at the figure 21 it will still need more episodes to reach it even further to provide a proper estimation of a Q-value. This can be caused by the conservativeness and carefulness of the algorithm which can be good in some use cases. The Q-Learning performed overall very well. luckily it was able to find a global optimum during the first episode which helped in the future training. This was purely random, and in some cases, it took 2-3 episodes to find a global optimum if the algorithm picked actions to opposite actions and went out of the boundary.

During the Reinforcement learning of the Two-axis movement task, the simple algorithm Q-Learning and Sarsa are unable to convolve due to the increased dimensionality of the problem. The algorithm reached the goal a few times during the learning but it can be assumed this is random. Deep Q-Learning is able the accurately estimate the Q-value and guide an Agent towards a goal with a few steps. It is also successfully cooping with the random noise in the SJTU Drone which is causing drifting of the UAV. The resulted accuracy of 98.94% shown in table 7 is an exceptionally good result.

During the last experiment with the Three-axis movement task, the outcome was similar to the previous task, and Q-Learning and Sarsa are unable to find optimal

Q-value. During this experiment, it is visible that Deep Q-Learning is struggling at the beginning often reaching outside of the boundary box until it figured the correct behavior. This struggle is normal given the complexity of this task. In the end, Deep Q-Learning is able to cope with the noise in the environment and successfully reach a goal with the accuracy of 94.77% shown in table 9. The accuracy would be even higher due to averaging if the training would run more steps as in the figure 26 it is showing consistent behavior.

7 Conclusion

After the implementation and experimentation with the Reinforcement learning environment to perform an inspection on the linear structures, it can be said that the learning environment is able to provide a baseline for future improvement and measurably compare different algorithms as it was performed with the Q-Learning, Sarsa, and Deep Q-Learning. After the experiments in the simulation, it is clearly visible that reinforcement learning can be potentially used in the inspection and is able to perform autonomous flying in a simple task. Further improvements for the learning environments are needed with the implementation of more complex tasks where the UAV is fully controlled by Reinforcement learning. Another improvement in terms of task can be a change to continuous actions instead of discrete which were used in this work. The use of Actor-Critic where the action will be directly velocities instead of predefined movements can lead to another interesting result and more robust behavior.

The research questions that were presented in Section 1.3 were explored during this work through the use of the literature, then by the designing and implementation of the proof of the concept which then enabled to perform simulations to gather data which are used for the answers. The following text will present the answers to the research questions.

According to the results from the simulation, the most suitable algorithm is Deep Q-Learning which is able to perform well in the simple task and also in complex tasks where the simple algorithms are unable to provide a solution. During the work, the limits of the Reinforcement learning were not reached for the more complex Algorithm. But the simple Q-Learning and Sarsa were able to solve only tasks with a small dimensionality of 3 actions and 3 observations. The computational requirements of Reinforcement learning are fairly high but it can be optimized as shown in section 4.5 with a more detailed discussion about computational requirements. The simplification of the complex data from the drone is not performed because the data were completely removed and replaced by the manual calculation inside of the simulation.

From the results gathered, the reinforcement learning environment is functional and serves as a proof-of-concept in a simulated environment. It was able to provide useful metrics from the reinforcement learning algorithms and perform intended tasks autonomously.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Aichinger M Kuhn M Weymann M Schmid G Bruch M, Münch V. Power blackout risks. *CRO forum*, page 28, 2011.

[3] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[4] Leena Matikainen, Matti Lehtomäki, Eero Ahokas, Juha Hyyppä, Mika Karjalainen, Anttoni Jaakkola, Antero Kukko, and Tero Heinonen. Remote sensing methods for power line corridor surveys. *ISPRS Journal of Photogrammetry and Remote sensing*, 119:10–31, 2016.

[5] Nicolaj Haarhøj Malle, Frederik Falk Nyboe, and Emad Ebeid. Survey and evaluation of sensors for overhead cable detection using uavs. In *2021 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 361–370. IEEE, 2021.

[6] Lan Chen, Lin Lin, Mimi Tian, Xingming Bian, Liming Wang, Zhicheng Guan, et al. The ultraviolet detection of corona discharge in power transmission lines. *Energy and power engineering*, 5(04):1298, 2013.

[7] Luis Luque-Vega, Bernardino Castillo-Toledo, Alexander Loukianov, and Luis González-Jiménez. Power line inspection via an unmanned aerial system based on the quadrotor helicopter. pages 393–397, 04 2014.

[8] Carol Martinez, Carlos Sampedro, Aneesh Chauhan, and Pascual Campoy. Towards autonomous detection and tracking of electric towers for aerial power line inspection. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 284–295. IEEE, 2014.

[9] Van Nhan Nguyen, Robert Jenssen, and Davide Roverso. Automatic autonomous vision-based power line inspection: A review of current status and the potential role of deep learning. *International Journal of Electrical Power & Energy Systems*, 99:107–120, 2018.

[10] Dewi Jones and Graham Earp. Requirements for aerial inspection of overhead electrical power lines. In *RPVs International Conference, 12 th, Bristol, United Kingdom*, 1996.

[11] Hu Sheng, Haiyang Chao, Cal Coopmans, Jinlu Han, Mac McKee, and YangQuan Chen. Low-cost uav-based thermal infrared remote sensing: Platform, calibration and applications. In *Proceedings of 2010 IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications*, pages 38–43. IEEE, 2010.

[12] François Mirallès, Nicolas Pouliot, and Serge Montambault. State-of-the-art review of computer vision for the management of power transmission lines. In *Proceedings of the 2014 3rd International Conference on Applied Robotics for the Power Industry*, pages 1–6. IEEE, 2014.

[13] Guangjian Yan, Chaoyang Li, Guoqing Zhou, Wuming Zhang, and Xiaowen Li. Automatic extraction of power lines from aerial images. *IEEE Geoscience and Remote Sensing Letters*, 4(3):387–391, 2007.

[14] Zhengrong Li, Yuee Liu, Rodney Walker, Ross Hayward, and Jinglan Zhang. Towards automatic power line detection for a uav surveillance system using pulse coupled neural filter and an improved hough transform. *Machine Vision and Applications*, 21(5):677–686, 2010.

[15] Joshua Candamo, Rangachar Kasturi, Dmitry Goldgof, and Sudeep Sarkar. Detection of thin lines using low-quality video from low-altitude aircraft in urban settings. *IEEE Transactions on aerospace and electronic systems*, 45(3):937–949, 2009.

[16] Qirong Ma, Darren S Goshi, Yi-Chi Shih, and Ming-Ting Sun. An algorithm for power line detection and warning based on a millimeter-wave radar video. *IEEE transactions on image processing*, 20(12):3534–3543, 2011.

[17] Steven Mills, Nabil Aouf, and Luis Mejias. Image based visual servo control for fixed wing uavs tracking linear infrastructure in wind. In *2013 IEEE International Conference on Robotics and Automation*, pages 5769–5774. IEEE, 2013.

[18] Alexander Ceron, Flavio Prieto, et al. Power line detection using a circle based search with uav images. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 632–639. IEEE, 2014.

[19] Michael L. V. Pitteway and Dereck J Watkinson. Bresenham's algorithm with grey scale. *Communications of the ACM*, 23(11):625–626, 1980.

[20] Cesar Iovescu and Sandeep Rao. The fundamentals of millimeter wave sensors. *Texas Instruments*, pages 1–8, 2017.

[21] Thinal Raj, Fazida Hanim Hashim, Aqilah Baseri Huddin, Mohd Faisal Ibrahim, and Aini Hussain. A survey on lidar scanning mechanisms. *Electronics*, 9(5):741, 2020.

[22] S. Lovati. Fundamentals of digital magnetic sensors, 2019.

[23] Mostafa Arastounia and DD Lichti. Automatic extraction of insulators from 3d lidar data of an electrical substation. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci*, 2:19–24, 2013.

[24] Gunho Sohn, Yoonseok Jwa, and Heungsik Brian Kim. Automatic powerline scene classification and reconstruction using airborne lidar data. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci*, 13(167172):28, 2012.

[25] Jittichat Tilawat, Nipon Theera-Umpon, and Sansanee Auephanwiriyakul. Automatic detection of electricity pylons in aerial video sequences. In *2010 International Conference on Electronics and Information Engineering*, volume 1, pages V1–342. IEEE, 2010.

[26] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

[27] Sanghyun Kim, Jongmin Park, Jae-Kwan Yun, and Jiwon Seo. Motion planning by reinforcement learning for an unmanned aerial vehicle in virtual open space with static obstacles. In *2020 20th International Conference on Control, Automation and Systems (ICCAS)*, pages 784–787. IEEE, 2020.

[28] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM Transactions on Cyber-Physical Systems*, 3(2):1–21, 2019.

[29] Shweta Bhatt. Reinforcement learning 101, 2018.

[30] Joseph Rocca. The exploration-exploitation trade-off: intuitions and strategies, 2018.

[31] Monya Baker. 1,500 scientists lift the lid on reproducibility, 2016.

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[33] Dronecode. Px4 autopilot, 2021.

[34] Shanghai Jiao Tong University. Sjtu drone, 2021.

[35] Martin Janosik. Reinforcement learning for inspection path planning of linear infrastructures - code repository, 2021.

[36] ROS2 Community. Ros2 design, 2021.

[37] Yuya Maruyama, Shinpei Kato, and Takuya Azumi. Exploring the performance of ros2. In *Proceedings of the 13th International Conference on Embedded Software*, pages 1–10, 2016.

[38] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[39] Timmy A Hussain. Combining deep q-networks and double q-learning to minimize car delay at traffic lights.

[40] Jingjing Zhang, Liang Liu, Binhai Wang, Xiguang Chen, Qian Wang, and Tianru Zheng. High speed automatic power line detection and tracking for a uav-based inspection. In *2012 International Conference on Industrial Control and Electronics Engineering*, pages 266–269. IEEE, 2012.

[41] Wu Jiang, Fei Wenkai, and Li Qianru. An integrated measure and location method based on airborne 2d laser scanning sensor for uav's power line inspection. In *2013 Fifth International Conference on Measuring Technology and Mechatronics Automation*, pages 213–217. IEEE, 2013.

[42] Weison Lin, Adewale Adetomi, and Tughrul Arslan. Low-power ultra-small edge ai accelerators for image recognition with convolution neural networks: Analysis and future directions. *Electronics*, 10(17):2048, 2021.

[43] Jia Liu, Jianjian Xiang, Yongjun Jin, Renhua Liu, Jining Yan, and Lizhe Wang. Boost precision agriculture with unmanned aerial vehicle remote sensing and edge intelligence: A survey. *Remote Sensing*, 13(21):4387, 2021.

[44] Guang Zhou, Jinwei Yuan, I-Ling Yen, and Farokh Bastani. Robust real-time uav based power line detection and tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 744–748. IEEE, 2016.

Appendices

A Results and Parameters from experiments

A.1 One-axis movement using Q-Learning - First attempt

The table and figure below is showing the first attempt at Reinforcement learning which resulted in wrong behavior due to incorrect training parameters. The agent figured that moving right and left will result in more reward than finishing a task. The table 11 is providing a summarization of parameters used in the experiment and figure 27 is showing the number of steps per episode and achieved reward.

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	400
# of steps per episode	500
Learning task parameters	
# of actions (left, right, nothing)	3
Punishment for the movement in a bad direction (futher from the goal on Y-axis)	N/A
Maximum consequent stops	N/A
Punishment for stopping longer than the Max consenquent stops	N/A
Reward for movement closer to the goal	10
Reward for not crashing	1
Reward for successfully reaching goal	200
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	-200

Table 11: Summarisation of the Learning task parameters used in the One-axis movement experiment. The N/A parameters were not implemented at a given time.

Figure 27: The plot shows Steps and Rewards using a Simple moving average of 5 per Episode in a first attempt

A.2 One-axis movement using Q-Learning - Stopped behaviour

The table and figure below is showing the attempt at Reinforcement learning which resulted in wrong behavior due to incorrect training parameters. Agent figured out that he can move a few times towards a goal and then stop for the rest of the episode without punishment. The table 12 is providing a summarization of parameters used in the experiment and figure 28 is showing the number of steps per episode and achieved reward.

A.3 One-axis movement using Q-Learning - Fixed stopped behavior

The table and figure below is showing the attempt at Reinforcement learning which removed the stopping behavior of the agent which was caused by the wrong parameters. Agent figured out the task and was able to successfully reach a goal. The table 13 is providing a summarization of parameters used in the experiment and figure 29 is showing the number of steps per episode and achieved reward.

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	400
# of steps per episode	220
Learning task parameters	
# of actions (left, right, nothing)	3
Punishment for the movement in a bad direction (futher from the goal on Y-axis)	N/A
Maximum consequent stops	N/A
Punishment for stopping longer than the Max consenquent stops	N/A
Reward for movement closer to the goal	5
Reward for not crashing	0
Reward for successfully reaching goal	400
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	-400

Table 12: Summarisation of the Learning task parameters used in the One-axis movement experiment. The N/A parameters were not implemented at a given time.

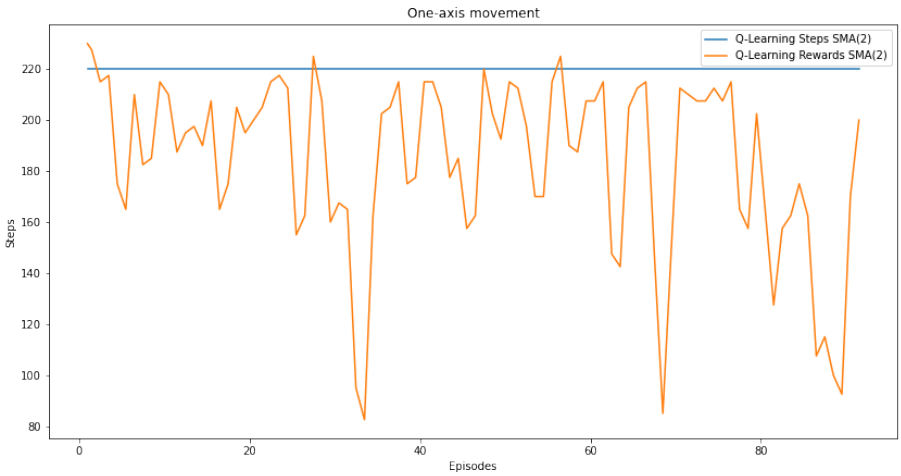


Figure 28: The plot shows Steps and Rewards using a Simple moving average of 2 per Episode in a stopped behaviour

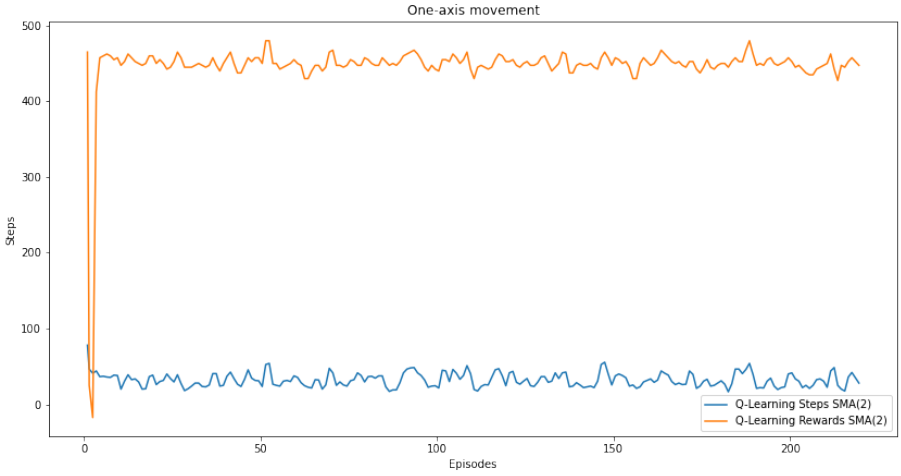


Figure 29: The plot shows Steps and Rewards using a Simple moving average of 2 per Episode in an attempt to fix the stopped behavior

Q-Learning / Sarsa parameters	
Alpha (learning rate)	0.1
Gamma (value of future reward)	0.7
Epsilon (randomness - $\varepsilon - greedy$)	0.9
Epsilon Discount	0.999
# of episodes	400
# of steps per episode	220
Learning task parameters	
# of actions (left, right, nothing)	3
Punishment for the movement in a bad direction (futher from the goal on Y-axis)	N/A
Maximum consequent stops	3
Punishment for stopping longer than the Max consenquent stops	-10
Reward for movement closer to the goal	5
Reward for not crashing	0
Reward for successfully reaching goal	400
Punishment for not reaching the goal (the episode is done due to the movement out of boundary or maximum steps are reached)	-400

Table 13: Summarisation of the Learning task parameters used in the One-axis movement experiment. The N/A parameters were not implemented at a given time.

B Github repository

The implemented proof of the concept together with the trained results is available in GitHub on the following URL:

`https://github.com/aykonsvk/ros-drone-reinforcement`

The code contains the worlds, training results, and models for Q-Learn and Sarsa algorithm in *ros-drone-reinforcement / reinforcement_drone /* folder. The folder also contains the launch files and all the code necessary to reproduce the experiments.

The system requirements are stated in the *README.md* file. Together with a link to the useful websites with the tutorials to set up ROS and Gazebo environment

The *README.md* file contains a link to the video displaying a short demo of behavior during the One-Axis movement with the Q-Learning algorithm.