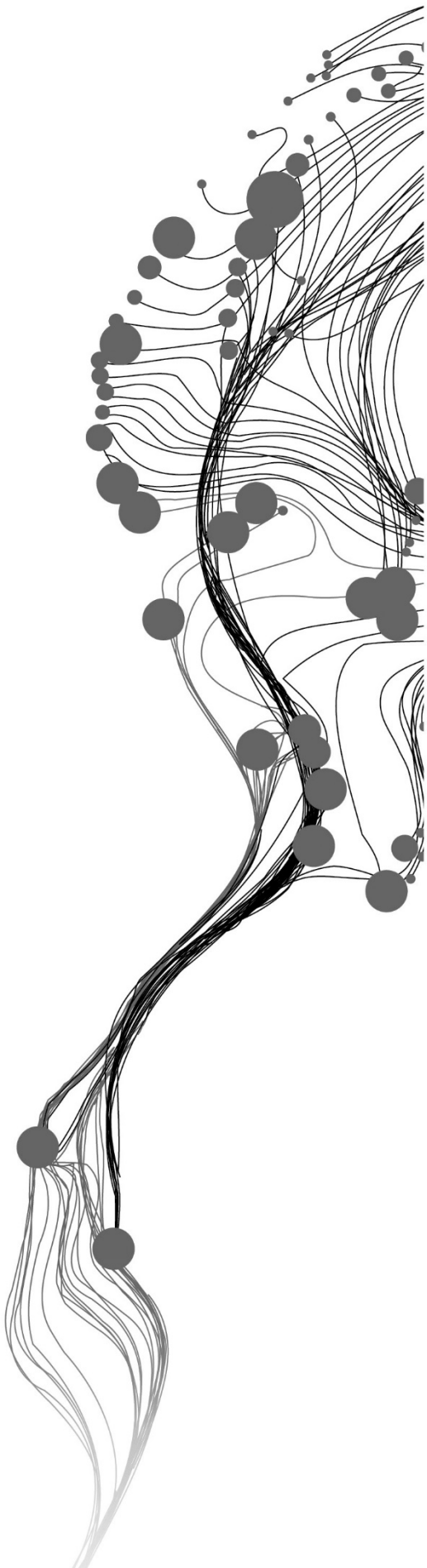
FEASIBILITY OF REAL-TIME UAV FLIGHT PATH PLANNING FOR URBAN MONITORING

SHICHEN HU
June, 2021

SUPERVISORS:
Dr. F. C. Nex
Dr. B. Alsadik



Feasibility of Real-time UAV Flight Path Planning for Urban Monitoring

SHICHEN HU
Enschede, The Netherlands, June, 2021

Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.
Specialization: Geoinformatics

SUPERVISORS:
Dr. F. C. Nex
Dr. B. Alsadik

THESIS ASSESSMENT BOARD:
Dr. C. Persello (Chair)
Dr. M. Koeva (External Examiner，ITC PGM department)

ABSTRACT

The unmanned aerial vehicles (UAVs) have become popular in many research fields in recent years. The UAV has the advantage of small size, simple structure, high mobility, and low cost. The flexibility of UAVs provides a new possibility of environment exploration, especially for those dangerous places where are difficult to reach. However, UAVs also have limitations due to their small size. The small size means limited payload and flight time. For applications that the target of interest is not the whole scene but specific objects within the scene, the traditional way is to fly twice. Use the first flight to obtain the location of the target, then fly another mission. However, the result of UAV applications also depends on environmental conditions. For instance, wind speed and illumination conditions also play a vital role in the data acquisition process. It is hard to keep the environmental factor being consistent between two flights.

This research proposed an updatable flight path according to the target of interest in one flight. The algorithm was divided into three parts: global planner, local planner, and object detector. The global planner is a pre-defined strip flight according to the simulation scene. The local planner is a circular flight path around the target with a lower flight height. The trigger between these two planners is an object detector. This object detector is a YOLO v3 detector with a Squeeze net backbone. The detector was first trained on the Aeroscape dataset and then performed a transfer learning process on the synthetic image dataset created manually. The whole algorithm was developed and tested using the MATLAB and Simulink platform. The simulation environment was established using Unreal Engine from Epic Games. The final output of this model is nadir view images along the whole flight path with object detection results added.

The training result on Aeroscape dataset achieved an average precision of 90%, and after the transfer learning process, the average precision increased to 100%. The object detector performs better at higher flight height in the simulation environment. The entire model runs stable in the Simulink and can have a target inspection from all directions.

Keywords: UAV flight planning; Object detection; YOLO v3; Simulink

ACKNOWLEDGEMENTS

Time flies, and graduation comes quietly. At this moment, when the thesis is finished, I want to thank all the professors and colleagues who helped me in the process of completing the thesis.

Thanks to my supervisor, Dr. F.C. Nex, for his patience and kindness when I faced many troubles implementing the algorithm in Simulink. All the comments and suggestions that he gave became the beacon of this research.

Thanks to my second supervisor Dr. B. Alsadik, for the fast reply to my email and all the help on mathematics and algorithm designation.

Thanks to my previous chair, Prof. Dr. ir. M.G. Vosselman for all the essential questions and helpful suggestions about presentations.

Thanks to my current chair Dr. C. Persello for taking the time to participate in my MSc research defense so that I can graduate as scheduled.

Thanks to S.M. Tilon, MSc, for helping me to achieve a proper YOLO network.

Thanks to drs. J.P.G. Wan Bakx for the explanations of a lot of procedures about graduations and caring for me.

Thanks to the two ladies from student affairs, M.C.F. Metz – Bekkers and T.B. van den Boogaard – Burke, for their so much love and concern.

Thanks to the MATLAB staff for the help and suggestions on Simulink.

Thanks to all the staff from the ITC building and ITC Hotel for making my two years living in the Netherlands very pleasant and comfortable.

Thanks to all my friends for caring for me over the years, especially under the situation of COVID-19. Their company and encouragement gave me the strength to move forward.

Finally, I would like to thank my parents. Their support and understanding behind the scenes have always been my source of strength. The grace of nurturing is nothing in return.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1. Research Background

In recent years, the development of unmanned aerial vehicles (UAVs) has provided new possibilities for many research fields. UAV has the advantages of small size, simple structure, easy to operate, high flexibility, low cost, high accuracy, etc. Therefore, it is widely used in many applications, e.g., search and rescue (SAR), monitoring, 3D reconstruction, and environment exploration[1]. At the same time, UAVs also provide the possibility for efficient and detailed data acquisition. The convenience and flexibility make it easy to support many applications, especially for those places which are dangerous or difficult for human beings to reach. The main mechanism of using UAVs is an efficient spatial data acquisition process, a low-cost solution for mapping the ground, a reliable source for spatial information [2].

However, UAVs also have their limitations. Plenty of technical issues and application problems still exist that need to be solved[2]. Besides, the limitation of commercial UAVs like the DJI Phantom series [3] also needs to be considered. For instance, the limited payload and restrict flight time due to its small size, the effect of wind at high flying altitude, and the fluctuation of the connection between UAV and its ground controller all may significantly impact the final result of applications [2].

Due to different applications, there is a need for different flight paths and camera conditions. Usually, obstacle avoidance, 3D reconstruction, and unknown environment exploration are the three typical topics in UAV path planning. For obstacle avoidance, the main idea is to find a collision-free path given a start point and an endpoint. For 3D reconstruction, the main issue is to guarantee a sufficient overlap between images to establish stereo image pairs. Finally, for unknown environment exploration, the difficulty is that the decision process becomes much more complex. Hence, the optimization of path planning becomes indispensable.

There are many methods for UAV flight planning. For instance, many commercial flight planners like Pix4D [4] and DJI-TERRA [5] can help users define simple flight paths. However, there are only several parameters that are provided for users [1]. Nevertheless, this method can overview the study area, and it is wildly used in many applications.

However, in unexpected situations, such as sudden natural disasters or investigations in an unknown environment, real-time adjustments are needed during the flight. The aim is to find out how to obtain more relevant information according to the actual situation, and in this kind of application, the overview information is insufficient.

Also, for patrolling and surveillance applications, to avoid occlusions and optimize the flight path in the distance, the flight path needs to be designed to fulfill the specific requirement [6]. Within the whole study area, if we are only interested in some unique targets in the study area, there is no need to get information about every object. Instead, only the information about the target of interest is needed. All of these specific applications need different strategies to accomplish different purposes.

1.2. Motivation

In an unknown environment, path planning becomes a fundamental problem for the exploration of UAVs. Due to flight time and energy limitations, finding an optimal path for a specific application is an unavoidable issue. In the situation mentioned in the previous section, the UAV needs to make a decision by itself while flying based on the pre-existing information [7], which means that an autonomous flight is required due to the dynamic nature of the real world. Autonomy is used in various fields related to UAVs, such as path planning, motion planning, and communication [8].

There are many methods for UAV path planning. According to the different purposes, various methods are proposed. As for representative techniques, two main directions are illustrated. One is sampling-based techniques, i.e., rapid-exploring random tree (RTT) [9] and A-star (A*) [10]. The other is artificial intelligence-based techniques, i.e., greedy algorithm, genetic algorithm, and traveling sales man problem[8]. However, hybrid approaches are recommended in the real world due to the uncertainty [11]. Besides, the continuous development of deep learning and neural networks also provides new possibilities for UAV path planning.

In many cases, when using UAVs to conduct investigations, operators do not have a detailed understanding of the investigated areas because the traditional flight path usually covers the entire study area in sequence under the condition of satisfying the overlapping relationship of adjacent photos. Thus, a simple strip flight plan is likely to lead to incomplete data acquisition when monitoring a specific target. Moreover, the conventional flying method usually uses a certain flying height, and the camera angle is unchangeable (nadir view or side-looking). Therefore, the strip flight path cannot be adjusted according to the real-world environment.

Then an algorithm for replanning the path is needed to complete the monitoring task. Most of the path-replanning algorithms are devoted to restoring the whole observed object, which is a complete 3D reconstruction process [12]–[14]. In their cases, a large number of images are needed to produce precise 3D models. However, for monitoring, only a complete observation of the target is required. Taking static vehicles as an example, if we want to know detailed information like the license plate number of vehicles in the urban area, we should only focus on vehicles as much as possible and ensure that all vehicles will be observed. For other objects in the study area, they can be ignored. Therefore, this algorithm for replanning should be more focused on information extraction rather than 3D reconstruction. Besides, the entire 3D reconstruction process is also quite time-consuming.

Therefore, to obtain more detailed information for a specific target, the flight plan needs to be adjusted according to the situation. For example, in an unknown environment, the location of targets is also unknown, so the UAV needs first to detect the target object and then fly towards this object. Then, after finishing detailed data acquisition, it keeps looking for the next target, which means the object detection needs to be in real-time. The automation of re-planning the UAV flight path in real-time in terms of the environment makes the data acquisition process more effective and informative.

This research attempts to develop a novel path planner that combines pre-set path planning, the global planner, and tailored path planning, the local planner, in the simulation environment. As a result, the simulated environment does not need to consider environmental factors such as weather, windspeed, and light conditions compare to test the algorithm in the field. Besides, there is no risk of damage to the UAV.

The UAV flight path will automatically update based on the target of interest. Then, all the images can be obtained in the same flight. With a high-speed object detector and the current UAV position, the target location can be obtained on the fly, and then, the UAV will switch from global planner to local planner. Finally, a simple local planner is proposed to complete the inspection of the target. The whole data acquisition process will keep running until it covers the entire study area.

1.3. Problem Statement

UAVs are used in various fields due to their flexibility. However, most applications use the traditional strip flight plan and post-data processing procedures. As illustrated in section 1.2, this kind of data acquisition process can lead to an incomplete data acquisition process in some applications. Besides, the UAV image data is sensitive to environmental factors like weather and lighting conditions. Therefore, it is hard to fly the UAV again with the same condition. To overcome this issue, a complete data acquisition process in a single flight would be ideal.

Another problem is, in terms of the regulations for UAVs among 19 countries [15], the maximum flight height cannot be higher than 150 meters. When using the same camera setting, the higher the flight height, the larger the field of view. In consideration of covering the whole study area in the shortest time, the flight height should be set as high as possible under regulations. In this case, when focusing on small static objects in the study area like vehicles, it is hard to extract information from 100 meters away.

Therefore, the proposed research will be to achieve a new method of flight path planning by combining different techniques. Take advantage of the CNN-based object detection method to achieve a near-real-time object detector, and then combine the traditional way of coverage flight planning with a more detailed local planning method following the object itself with lower flight height, finally, attempt to achieve a complete and effective data acquisition procedure.

1.4. Novelty

Many path planning algorithms are available, but the problem still exists in locating and identifying target aspects [8]. Therefore, instead of providing complete coverage of the study area, only focusing on the target part can be another choice to save time and reduce data redundancy. This study presents a first attempt to combine different planning modes and an online CNN-based object detector to achieve a flexible path planning process according to the objects in a simulation environment.

This study was implemented using the UAV toolbox developed by MathWorks. This new release toolbox in 2020 provides a new possibility for the simulation of UAVs. Instead of using Robot Operating System (ROS) with C + +, Simulink becomes another choice to achieve simulation without designing the control system of the UAV. Besides, it is easier to define the algorithm with a graphical programming environment and simulate different realistic situations. Moreover, with cooperation with Unreal Engine, a more complex simulation scene can be created, indicating that simulation can provide various testing environments without risk of damaging the UAV before the test of the algorithm in a real environment.

1.5. Research objectives and Research questions

The overall objective is to develop an algorithm that allows the flight path of UAVs to be updated according to a specific target. The algorithm will be implemented in the simulation environment, and the targets used in the simulation are vehicles. This algorithm comprises three parts: global planner, local planner, and object detector. For the study area that needs to be covered, both the global planner and the local planner are developed. The global planner is the pre-defined strip flight, and the local planner is a circular flight path around the target. Based on the images obtained in real-time, an object detector is proposed to search for a vehicle. Once a vehicle is found, switch the global planner to the local planner. When the local planner is finished, switch the local planner back to the global one, and continue searching for the next object of interest until it covers the whole area.

The UAV should change successfully from global planner to local planner and vice versa. For the global planner, the strip flight should be defined according to the simulation scene. For the local planner, the circular flight path should be localized around the target. For object detection, the method needs to work efficiently and precisely to detect vehicles while flying.

Sub-objective 1: To implement an efficient local planner algorithm able to acquire complete information about the vehicle.
Corresponding research questions:

1) What are the available methods for UAV flight planning for a specific object?
2) How to determine the completeness of the acquired images?
3) How to switch between the local planner and the global planner?

Sub-objective 2: To implement an object detection process to respond in real-time or near real-time.
Corresponding research questions:

1) What are the available CNN-based algorithms for real-time object detection?
2) Which image dataset is suitable for training the network to detect vehicles from a nadir view?
3) How to implement the network in the simulation environment?

Sub-objective 3: To embed the developed algorithm on a simulated virtual environment like Simulink.
Corresponding research questions:

1) Is Simulink suitable for the simulation of UAV path planning?
2) Is Simulink able to embed an object detection algorithm?
3) How to execute an updatable UAV flight plan in Simulink?

2. LITERATURE REVIEW

2.1. UAV Platforms

At present, there are two kinds of UAVs in common use, one is the UAV that can take off and land vertically (VTOL), also known as multi-rotor UAVs (figure 2.1(a)), and the other is the UAV with fixed wings (figure 2.1(b)). Fixed-wing UAVs have the advantages of fast flight speed and long flight distance, making them very suitable for detecting long strip features such as roads [16]. But the disadvantage of these fixed-wing UAVs is that they cannot hover in the air and require more sophisticated ways to take off, while their payloads are limited [17],[16],[18].

The VTOL UAV has multiple rotors, allowing it to take off and land vertically. The representative commercial VTOL UAVs can be DJI phantom series [3]. Such UAVs can hover in the air and observe targets from various angles [17]. But the main disadvantage of vertical takeoff and landing UAVs is that the flight time is relatively limited. Most of their batteries support a flight length of about 30 minutes, which means that the VTOL UAV is not suitable for long-distance monitoring [19],[17]. So, a hybrid UAV arises to be a new solution (figure 2.1(c)). Hybrid UAVs have the advantages of both types of UAVs, which can take off and land vertically and fly long distances. But the main disadvantage of a hybrid UAV is that its load is similar to a fixed-wing UAV [17].

(a)Multi-rotor UAV[20]

(b)Fixed wing UAV[21]

(c) Hybrid UAV[22]

Figure 2.1 Different UAV platforms

2.2. UAV Flight Planning

In the process of 3D restoration modeling with UAV, the flight path and view of the UAV are essential parameters to be considered [1]. UAV has a wide range of applications, and different applications have different requirements for flight planning, the angle of instruments carried on UAVs, flight time, and so on. The angle of the sensor determines the feature of the data. For orthophoto products, the nadir view is often used, and for 3D reconstruction, an oblique view can provide more information about the façade. The flight time depends on the energy of UAVs. Therefore, when considering the flight path, the energy-consuming issue also needs to be included. Meanwhile, how to deal with the occlusion problem during flight is also very important [1].

Methods to plan the flight path of UAVs can be divided into two categories. One is to make a flight plan before the flight (off-the-shelf flight planners) [23]. The other is to change the flight plan in real-time according to the changes of ground features (explore-then-exploit methods) [23].

2.2.1. Off-the-shelf Flight Planner

The off-the-shelf flight planner is the conventional way of the strip flight. Some parameters are included in this planner: camera setting, ground sampling distance, flight height, the proportion of image overlap, viewing angle, and flight time. Take the Pix4D capture [24] as an example of off-the-shelf flight planner software.

The Pix4D capture is a free mobile app available both on Android and iOS, which means users can easily plan and control UAVs from their mobile phones. It supports UAVs from three companies, DJI [25], Parrot [26], and Yuneec [27]. Then, five types of missions are available: polygon, grid, double grid, circular, and free flight [28]. The first three types of flights are all strip flights. The difference between polygon and grid is that the shape of the study area is different. Polygon means irregular shape, and grid means rectangular shape. Then the double grid means to fly a two consecutive grid mission in two perpendicular directions.

After selecting the type of mission, several parameters are provided to adjust the flight plan: angle of the camera, front overlap, side overlap, and drone speed. Usually, the front overlap (along-track overlap) is recommended to set to 80 percent, and the side overlap (across-track overlap) is recommended to set to 60 percent. After finishing all the settings, start the mission in the field, and the UAV will fly autonomously according to the pre-defined flight plan.

Off-the-shelf flight planners can be very convenient but also have some limitations. For example, it is easy to have complex occlusion relationships in dense urban areas, and the route set in advance may lead to incomplete data acquisition [1]. And this method does not adequately consider the geometric characteristics of the features.

Another approach consists of two phases. According to a conventional way, the first phase covers the whole research area uniformly, like the off-the-shelf flight planners. Then, through these images, to build a relatively rough 3D model. Then, a new flight trajectory is calculated in the second stage [1]. But even flying two times is still time-consuming, especially when the research area is relatively big. And the illumination condition also has a considerable effect.

2.2.2. Explore-then-exploit Flight Planner

There are many methods for explore-then-exploit flight planning. Next-best-view is one of them. A next-best-view (NBV) planning algorithm was developed simply back in the 1980s [29]. It merely defines an octree model for empty, occupied, and unseen, but later generations have continuously improved this idea to realize automatic 3D measurement [30]. Furthermore, an iterative linear method to quickly calculate the multi-view-stereo(MVS) problem is developed to estimate an initial model of the object and then solve the next position [31]. Another next-best-view algorithm is based on the reflectional symmetry feature from the rough model built after the first flight [32].

Apart from this, many other methods related to path planning are developed various from a different aspect. For area coverage of unknown terrain, a tree-like structure is designed [33]. The main idea is to build a tree in a vertical direction to deal with the non-uniform distribution of targets of interest. Instead of using the lawnmower pattern to cover every area, this method provides an online coverage path planning technique for shorter coverage paths. Due to this tree-like structure, the targets of interest will be covered with a higher resolution because of flying height change. This method wisely uses the difference in the field of view caused by the flight height change to obtain high-resolution images of the objects of interest. Still, it does not consider the energy consumption of the vertical movement of the UAV.

Another triangular mesh representation structure is developed for inspection path planning [13]. They proposed a two-step optimization paradigm to find the viewpoints that meet full coverage requirements and the low cost of the connecting path. This method focuses more on the 3D structure of the target object. The object is represented by a triangular mesh. For each triangle in the mesh, make sure that there is an acceptable viewpoint that, from this viewpoint, the corresponding triangle is visible. And the next step is to find a path that can inspect the whole 3D structure at the lowest cost. This algorithm is more complex and performs a complete 3D reconstruction [13].

Besides, deep learning becomes a research hot spot in recent years. It can also be used in UAV surveillance. Implement the learning method to allow the UAV to identify the region of risk and area of interest, then increase the time of visit in the area of interest and decrease the visit frequency in the risky region [14]. This method will get more data about the area with a higher possibility of the target of interest, and based on this, the surveillance path is calculated. Although this method considers the possibility of importance in the research area, the similarity between real target and decoy can cause a problem. And the applicability of this method still needs to be validated. In contrast, there is also an application of a supervised learning method in 3D reconstruction, where an original 3D convolutional neural network is developed to predict the next-best-view (NBV) [34]. Unlike many other pieces of research that treat the NBV problem as a search problem, they modeled this problem as a classification problem, which means each prediction is based on the partial model that was obtained before. Apart from this, the training samples for this network become important.

According to the reviewed pieces of literature, the UAV flight planning methods are various from application to application. Also, most next-best-view algorithms are committed to building a complete 3D model of ground objects. Therefore, combining next-best-view and scene understanding to make the data acquisition process of ground features more effective and flexible is still an urgent problem.

2.3. Object Detection

Detecting objects in UAV images can be challenging due to irregular shapes, different image scales, various viewing angles, and so on [35]. On the other hand, Convolutional Neural Network (CNN) becomes a new technique that can achieve more efficient and accurate results than other techniques [36]. Due to this situation, the CNN-based object detection model has been developed fast. There are two frameworks mainly used for CNN-based object detection [36].

The first framework is the one-stage framework, and the other is the two-stage framework. As for the one-stage framework, for instance, You Look Only Once (YOLO) [37], it considers the object detection procedure as a regression problem and classifies the anchors directly. For the two-stage framework, Region-based Convolutional Neural Network (R-CNN) can be the milestone. It uses a selective search to search for the object location candidates. The features are extracted separately for each candidate [38]. However, the drawback of R-CNN is that the detection process is very time-consuming. To speed it up, Faster R-CNN [39] takes the whole image as the input of one single CNN to extract the features and also implements the Region Proposal Network (RPN) to increase the calculation speed further [35].

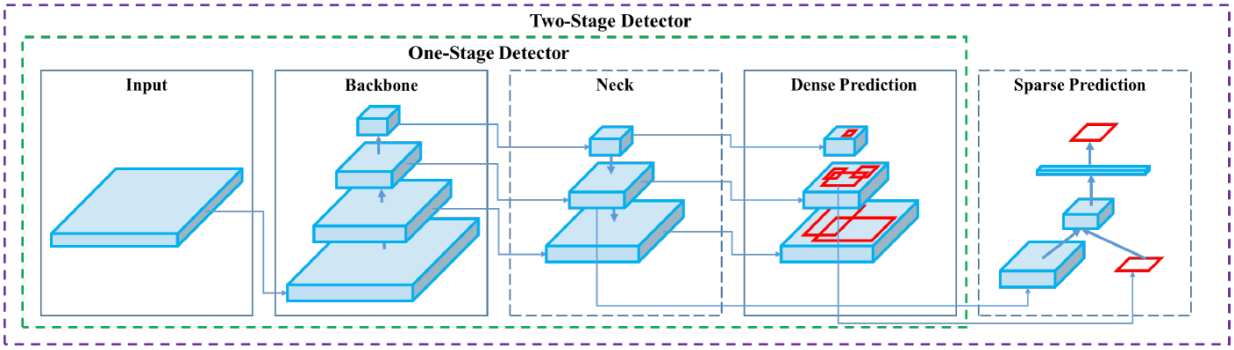


Figure 2.2 Object detector[40]

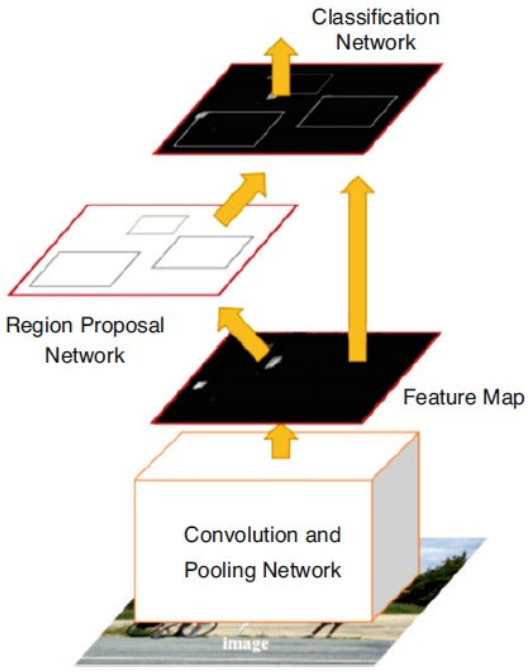


Figure 2.3 Faster R-CNN network architecture[41]

A Faster R-CNN(Fast Regions with CNN features) network was developed for UAV reconnaissance images [41]. In order to improve the accuracy of small object detection, the Adaptive Image Division algorithm (AID) was developed. Within the AID algorithm, images were divided into blocks in terms of the UAV flight parameters and the payload parameters, and then resize these blocks to the original image size to enlarge the object. Then use these resized images as the input of the network. The network structure is shown in figure 2.3, and the main three parts are convolution and pooling network, region proposal network (RPN), and classification network. The mean average precision of all detection results can reach 0.83.

However, for real-time applications, YOLO can achieve extremely fast calculation speed while maintaining reasonable accuracy. In 2016, the first version of YOLO had published, use the same backbone as Faster R-CNN, which is VGG-16. Although the mean average precision of YOLO is almost 7% lower than Faster R-CNN, the calculation speed is three times faster. YOLO performs worse to localize objects correctly but can distinguish better between object and background [37]. Besides, YOLO is not sensitive to nearby small objects, such as small birds in groups, but this problem has been improved in YOLO version 2 and version 3.

YOLO version 2 involves more deep learning tactics, including batch normalization, convolutional with anchor boxes, direct location prediction, and multi-scale training. Also, YOLO version 2 proposed a new classification model called Darknet-19 [42]. Darknet-19 contains fewer parameters than VGG-16 but still can achieve 72.9% top-1 accuracy and 91.2% top-5 accuracy on ImageNet. A new mechanism for joint classification and detection is proposed in YOLO9000 to increase the number of classes of the detection. By developing a new structure of the labels called WordTree, multiple datasets can be combined. Finally, YOLO9000 becomes a real-time object detector for more than 9000 object categories [42]. In 2018, YOLO version 3 has been released. YOLO version 3 developed a new backbone structure called Darknet-53, shown in figure 2.4 [43]. And YOLO version 3 introduced a feature pyramid network (FPN) to detect objects in different scales. It still runs extremely fast compared to RetinaNet and Faster R-CNN and achieves almost the same accuracy. Two years later, YOLO version 4 has made significant progress from YOLO version 3. YOLO version 4 has a new backbone architecture, the CSPDarknet53, and an adjusted neck, which improved the mAP (mean Average Precision) by 10% [40]. Moreover, it can be used on a single commercial GPU such as 1080Ti. With the bag of freebies and the bag of specials, YOLO version 4 becomes a very powerful and high-speed object detector.

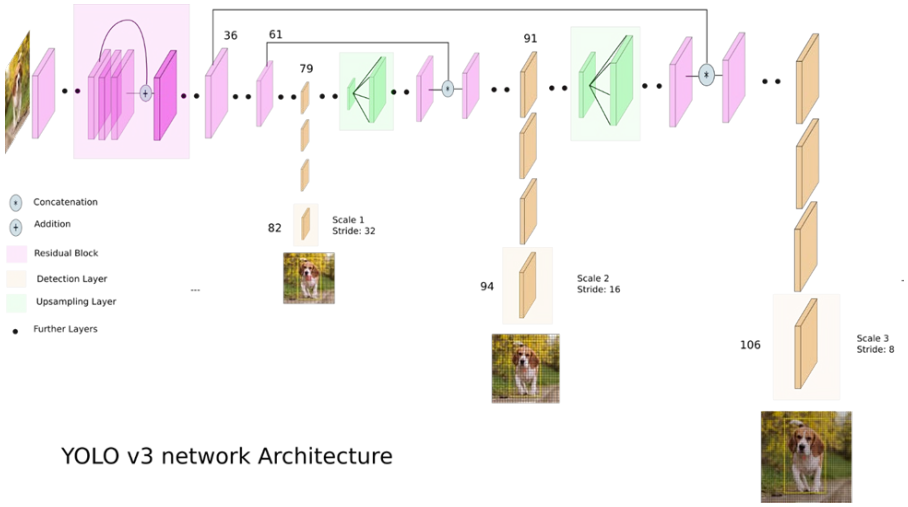


Figure 2.4 YOLO v3 network architecture[44]

3. SIMULATION ENVIRONMENTS

3.1. Simulink Model

Simulink is a graphical programming environment for modeling and simulating dynamic systems based on MATLAB. It is convenient to develop hybrid systems in Simulink with plenty of prebuild blocks. Especially, the UAV toolbox is designed for modeling, simulating, and testing various algorithms and applications. Besides, users can design and test flight algorithms, UAV missions, and flight controllers in the Simulink environment. Therefore, for implementation and testing, Simulink becomes an ideal choice. The entire model was built in MATLAB 2021a. The newest version of the UAV toolbox has two new parameters, which are weather and sun position. These parameters make the simulation environment closer to the real world so that more applications can be simulated, for example, the influence of shadow and canopy.

3.2. Unreal Scene

Within the UAV toolbox, MATLAB provides a customizable simulation environment called US City Block. This scene is visualized by Unreal Engine from Epic Games. The Unreal Engine is a powerful tool for creating a real-time 3D scene with plenty of impressive functions. It has been widely used in 3D game creation. Users can customize the scene with the Unreal Editor and the UAV toolbox interface. The Unreal Editor is an integrated development environment that supports various systems (i.e., Windows, macOS, and Linux) [45]. Moreover, it provides hundreds of free prebuild models, which contains complex scene, textures, or objects, that can be implemented directly.

By following the help document provided by MATLAB [46], users can run the simulation in the Unreal Editor and export an executable model that can be used directly in Simulink, as shown in figure 3.1. So far, this co-simulation framework only supports Unreal Engine version 4.23 (The Unreal Engine needs to be installed separately.). And the additional requirement is to install version 15.9 or a higher version of Visual Studio 2017. In this study, the US City Block was built based on the real-world city block in Chicago, United States.

Figure 3.1 Simulation 3D scene

The top view of the US City Block simulation scene used in this research is shown below in figure 3.2, where the extent of this area is 443.3 meters by 300 meters. There are fifteen intersections within the scene, and traffic lights and barriers are placed accordingly.

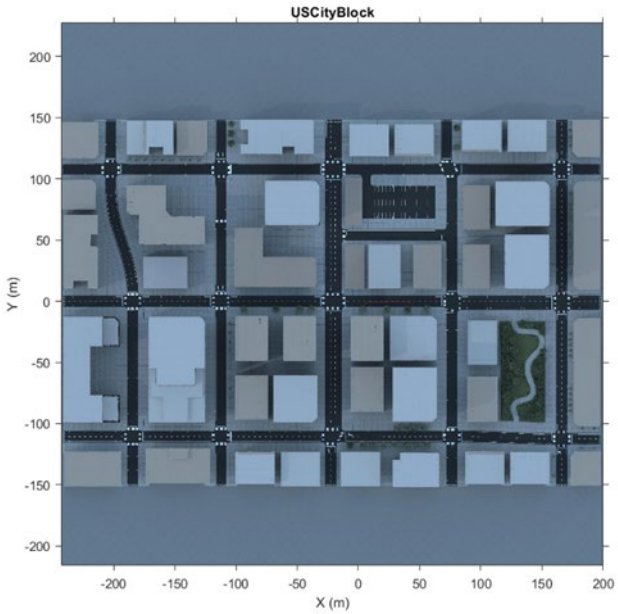
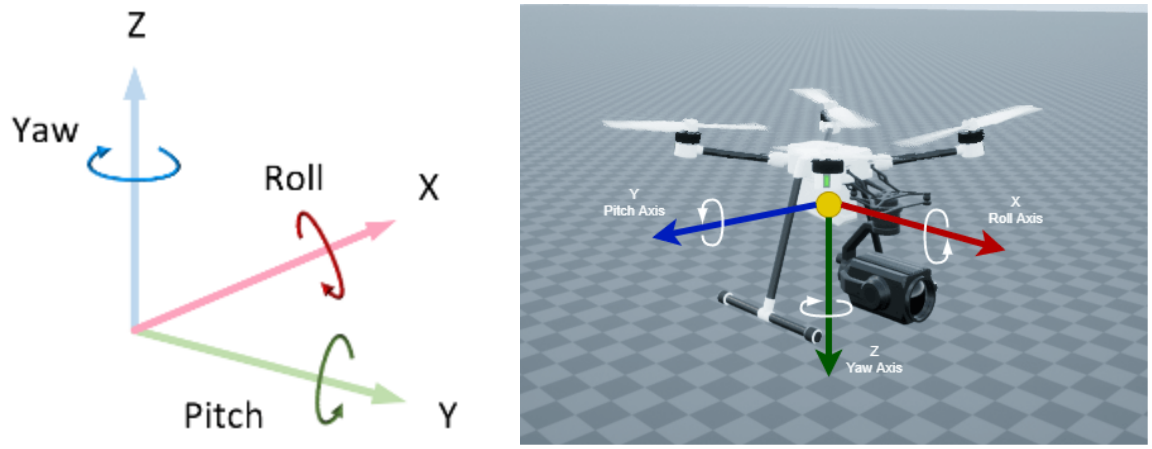


Figure 3.2 US City Block

The world coordinate system is selected as a right-handed and earth-fixed (inertial) coordinate system (Figure 3.3). It is an inertial coordinate system, which means the inertial reference frame does not have any linear and angular acceleration and angular velocity [47]. Another coordinate system used in the simulation is the body coordinate system which is fixed on the UAV itself. The origin and the orientation are both set on the moving UAV, and the UAV is assumed to be a rigid object.

The three dimensions of these two coordinate systems are defined in table 3.1 and table 3.2, and the directions are shown in figure 3.3.



(a)World coordinate system in Unreal Engine (b)Body coordinate system

Figure 3.3 Two coordinate systems used in simulation scene [47]

Table 3.1 World Coordinate in Unreal Engine [47]

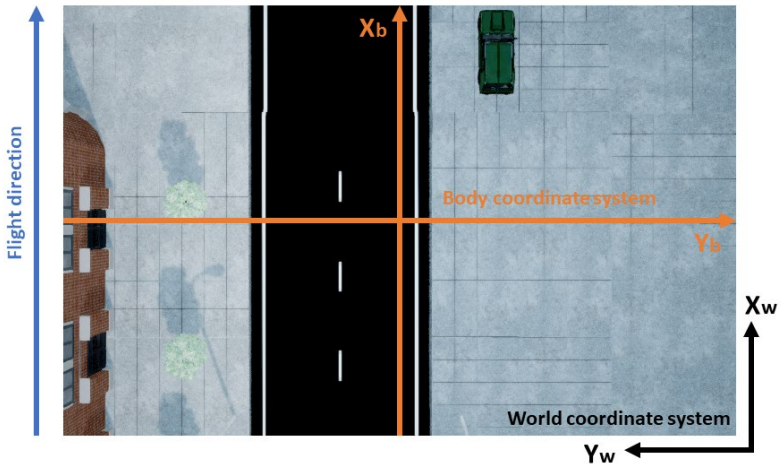
Axis	Description
X	The X-axis is in the forward direction of the vehicle. Roll - right-handed rotation of X-axis
Y	The X and Y axes are parallel to the ground plane. The ground plane is a horizontal plane normal to the gravitational vector. Pitch - right-handed rotation of X-axis
Z	In the Z-up orientation, the positive Z-axis points upward. Yaw – left-handed rotation of Z-axis

Table 3.2 Body coordinate system in Unreal Engine

Axis	Description
X	The X-axis points towards the front direction of the UAV, which is also along the flying direction. Roll - right-handed rotation of X-axis
Y	The Y-axis is on the right side of the X-axis and perpendicular to it. Pitch - right-handed rotation of Y-axis
Z	The Z-axis points downwards, and perpendicular to the X-Y plane meets the requirements of the right-hand system. Yaw - right-handed rotation of Z-axis

As illustrated in figure 3.5, the black coordinate system represents the world coordinate, and the orange coordinate system represents the body coordinate system. The image was captured by the nadir view camera. Therefore, the relationship between the two coordinate systems is that their direction of the Y-axis is opposite. Besides, as defined in table 3.1 and table 3.2, the Z-axis direction in two coordinate systems is also opposite. Thus, the world coordinate system has an upward Z-axis, while the Z-axis points downwards in the body coordinate system.

Figure 3.4 Relationship between two coordinate systems



In this study, the object of interest is the vehicle. Although the original US City Block environment does not contain vehicles, some free packages provided by Unreal Engine can be used to add some static vehicles

in this scene. Here, a package called Vehicle Variety Pack contains five types of vehicles shown in figure 3.5. Four of them were used in the scene, except the white one.

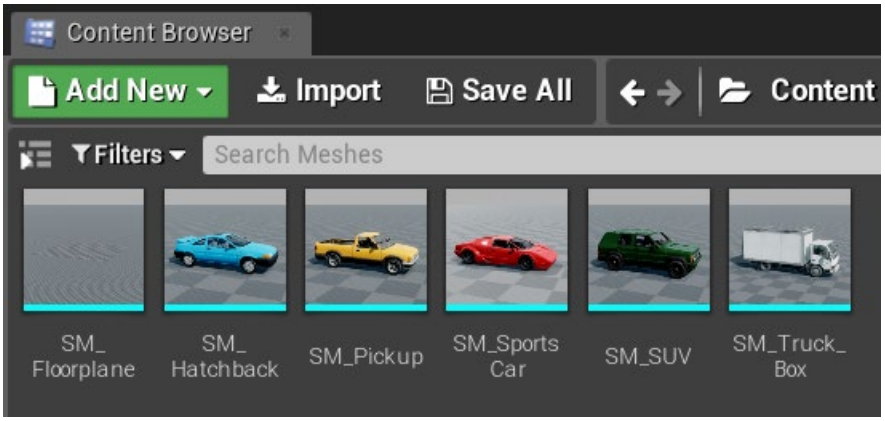


Figure 3.5 Five types of vehicle from Vehicle Variety Pack

In the real world, obstacle avoidance should be considered. However, this is beyond the scope of this research. Therefore, vehicles were placed at crossroads, where there is enough space to fly. There were nine vehicles added along the flight path shown in figure 4.2.

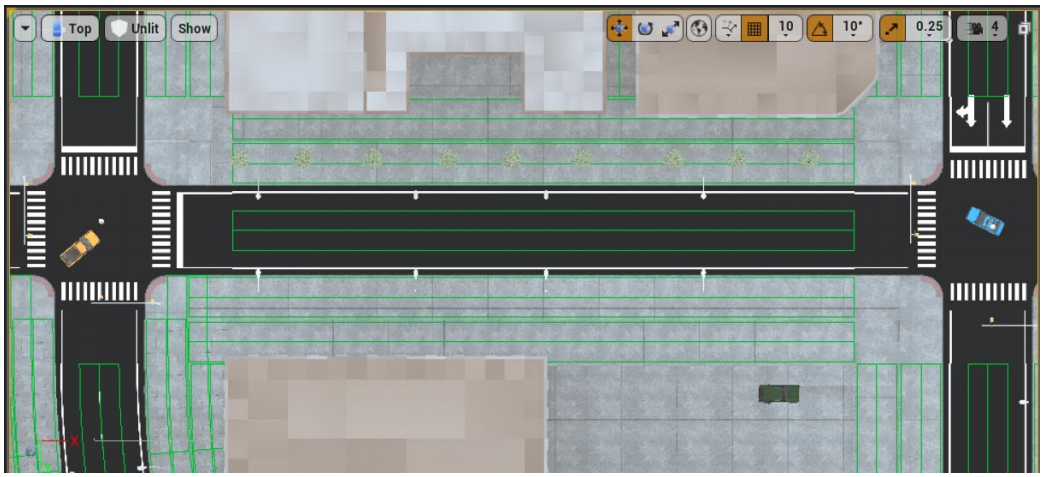


Figure 3.6 Vehicles in the simulation Scene

4. METHODOLOGY

The main idea of this research is to develop an algorithm for UAV flight path planning, which allow the UAV to change flight path according to the target of interest. In this study, take the vehicle as the target of interest. The flight planning strategy was divided into two parts, one is the global planner, and the other is the local planner. The global planner is similar to the conventional flight planners mentioned in section 2.2.1. The local planner is a localized circular flight path around the vehicle. Then, once detect the vehicle, switch the global planner to the local planner to obtain more information. Finally, when the whole global flight plan is complete, the flight ends.

The whole workflow is shown in figure 4.1. From the place where the UAV takeoff, first use the global planner, and at the same time, try to identify the vehicle according to the images captured by the UAV in the simulation scene. Once a vehicle is found, then switch to the local planner. At this stage, change the flight plan, reduce the flight altitude, complete a local circular flight path around the vehicle. When the circular flight path is complete, return to the global flight altitude, and continue target detection in the study area until the whole study area is covered.

For the global planer, conventional strip flight can be a reasonable choice, which is illustrated in section 4.1. As for the object detection process, a YOLO version 3 detector has been developed, which is further described in section 4.2. As for the local planner, a simple circular flight path is explained in 4.3. The whole algorithm was implemented in Simulink, and the model is explained in 4.4.

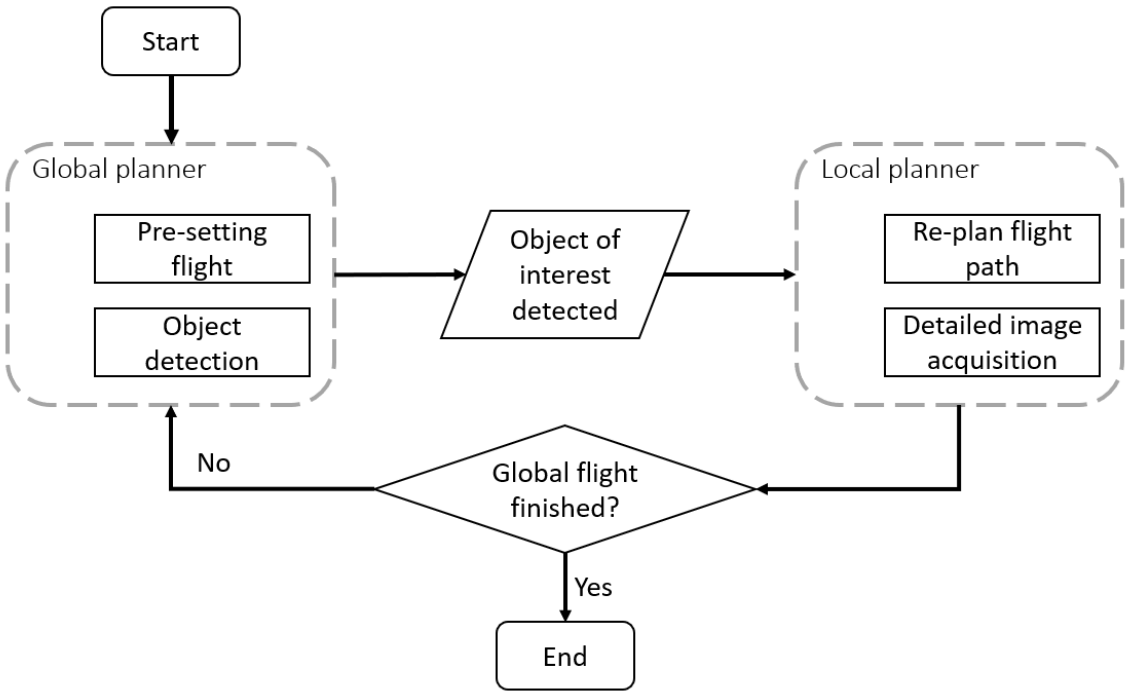


Figure 4.1 Overall workflow

4.1. Global Planner

The global planner is a pre-defined flight path according to the simulation scene. As shown in figure 3.2, in the unreal scene, the distance between buildings is relatively narrow. Therefore, the vehicles were placed in the open area near the intersections. In terms of this situation, the global planner was defined along the road shown in figure 4.2. Therefore, only the along-track overlap was considered because there was no vehicle between strips.

4.1.1. Camera Settings

In this study, the UAV carries a simulated camera[48] as the data acquisition sensor. This simulation camera is a model based on an ideal pinhole camera model [49], [50] and a lens in order to represent a full camera model. The lens distortion is defined as radial and tangential [51]. However, to simplify the process, in this study, lens distortion is not considered. Table 4.1 shows the specifications of the camera where all the parameters of the camera are specified in pixels. The relative rotation is the rotation angle of the camera relative to the UAV. Therefore, for a nadir view, the pitch angle was set to 90 degrees. The sample time of the camera is every second, and the accordingly overlap between two images acquired is greater than 95 percent.

Table 4.1 Camera parameters

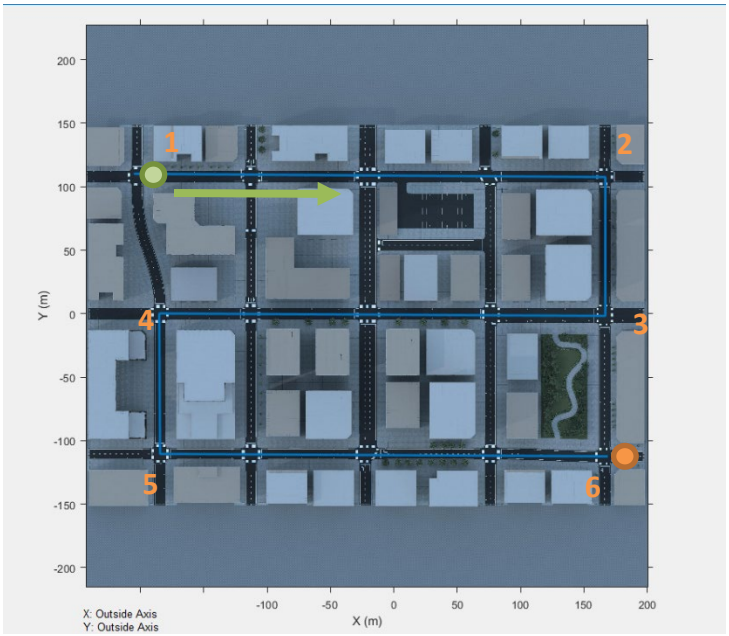
Parameter	Value (pixels)
Focal length	[1109, 1109]
Optical center (principal point)	[376, 240]
Image size (col, raw)	[480, 752]
Relative rotation [roll, pitch, yaw]/degree	[0, 90, 0]

4.1.2. Waypoints

This global flight path was defined manually according to the MATLAB help document [52]. The waypoints, including the takeoff and landing points, were extracted from the US City Block scene. As shown in figure 4.2, the blue line is the flight path. The UAV will take off at point 1 and then follow the sequence from point1 to point 6. The flight height was set to 50 meters, considering the size of the simulation scene and the height of the buildings within this scene.

The global flight path was broken down into small pieces every four meters in order to achieve an 80% front overlap between every waypoint. In this case, there are 322 waypoints in total, including the takeoff and landing point. However, the camera acquired image every second, which means more images will be obtained along the flight path.

Whenever the UAV arrives at a waypoint, it will decide where to fly next. If there is a vehicle between two waypoints, then the UAV will fly towards the vehicle. If there are no vehicles detected, the UAV will keep flying according to the pre-defined waypoints along the flight path.



Green point: start point
Green arrow: flight direction
Orange point: end point
Blue line: flight path

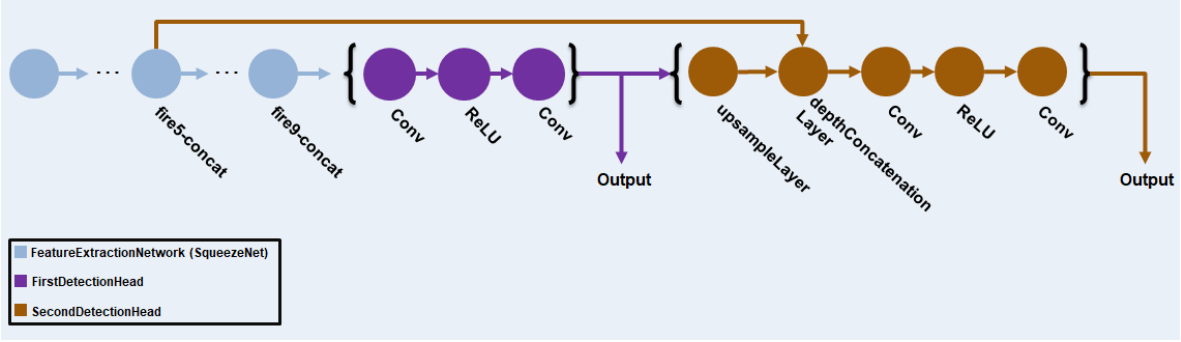
Figure 4.2 Global planner flight path

4.2. Object Detection using YOLO v3

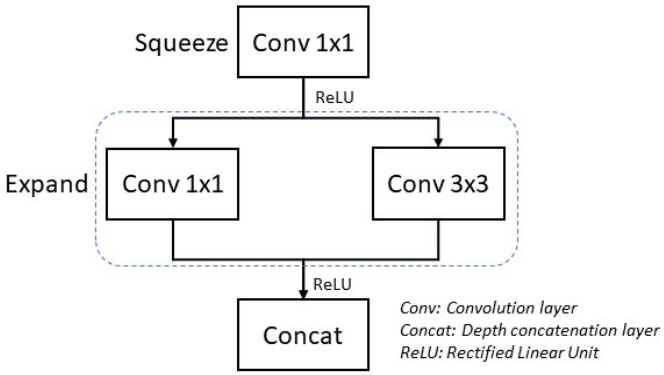
YOLO is a very representative one-stage object detection framework. Considering the compatibility in MATLAB, YOLO version 3, the newest version available in MATLAB was selected as the detector in this research.

Anchor boxes are used in YOLO version 3 to classify the object. Anchor boxes are a series of pre-defined bounding boxes based on the size of the object in the training dataset. These boxes are the initial bounding box guesses. With these first guesses, the network will not predict the exact bounding boxes. Instead, it will predict the probabilities and offset corresponding to the anchor boxes [53]. YOLO version 3 predicts three attributes for each anchor box: intersection over union (IoU), anchor box offsets, class probability.

MATLAB R2021a provides a yolov3ObjectDetetor object. Using this object, users can create a pre-trained YOLO v3 object detector using the COCO dataset or customize the network using a different base network available in MATLAB [54]. The original YOLO v3 developed a new network architecture called Darknet-53 [43], containing 53 convolution layers. However, in this research, with limited available training samples, a simple backbone can save more processing time but still preserve accuracy at the same time. Therefore, in this research, according to the document of MATLAB [55], the Squeeze net [56] was used as the backbone. Squeeze net is a small CNN structure but can reach the same accuracy as Alex net. Small CNN has several advantages: fewer parameters, faster training process, feasible updates, and smaller memory requirement [56]. In this research, the Squeeze net was pre-trained on the ImageNet dataset [57].



(a) Detection heads[55]



(b) Fire module

Figure 4.3 Network structure

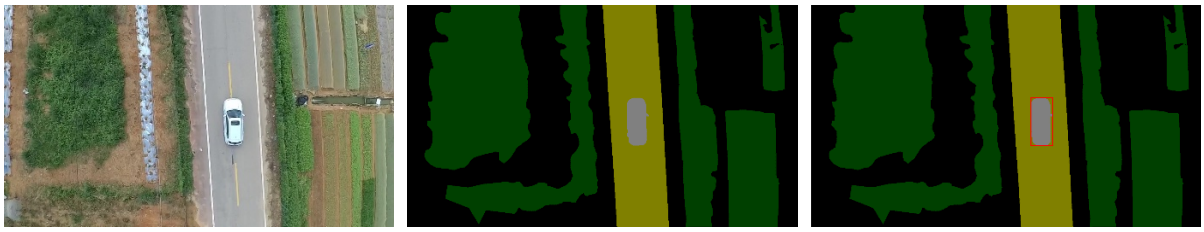
The main structure of the Squeeze net is the fire module, as shown in figure 4.3 (b). The fire module is comprised of one squeeze layer, which is a convolution that only has 1x1 filters, and an expand layer that contains a mix of 1x1 and 3x3 convolution filters and then concatenates the output of the expand layer [56]. Based on the original Squeeze net, two detection heads were added. As illustrated in figure 4.3 (a), the first detection head was added after the ninth fire module (fire9-concat), the second one was added after the fifth fire module (fire5-concat). The output of the first detection head was up-sampled into the same size as the output of the fifth fire module by replicating neighboring pixel values, and then the second detection head takes this concatenation layer as input. The detection head contains three layers, one 3x3 convolution layer, one ReLU layer, and one 1x1 convolution layer. Due to the up-sampling process, the second detection head was twice as big as the first one. Therefore, the second detection head was more sensitive to small objects [55]. The input size of the network is set to 227x227x3 to shorten the training time. Then, the image will first resize into the network size and then forward into the network.

4.2.1. Data Preparation

In this research, the specific object detection task is detecting vehicles from a nadir view. The AeroScapes aerial semantic segmentation benchmark is used as the training dataset of the YOLO v3 detector. The AeroScapes contains 3269 720p images captured by a commercial done from various flight heights (5-50 meters) and the corresponding ground truth labels for 11 classes [58].

From the AeroScapes dataset, first, select all the images that contain vehicles, and then select images that contain complete cars by visual inspection. Finally, 591 images were selected, randomly choosing 90% of them used for training, and the rest was used for testing. Bounding boxes were created based on the multi-class segmentation images. Within the multi-class segmentation images, the color of vehicles is grey, and its RGB value is (128,128,128), as shown in figure 4.4 (b).

Use this color information to filter out other objects in the image. Then create the bounding box around the vehicle (figure 4.4 (c)). The bounding boxes are defined in a vector that contains four elements: x, y, width, height. The x and y described the upper left corner, and the width and height defined the size of the bounding box, and all these values are in pixels.



(a)training sample image (b)Multi-class segmentation (c)Bounding Box

Figure 4.4 Training data preparation

Then, create text files containing the location of bounding boxes corresponding to each image, then save images and text files in the same folder. Finally, create the ground truth table using the imageLabeler-API [59]. As for data augmentation, only implement three simple techniques in the training set: color jitter augmentation in HSV space, random horizontal flip, and random scaling by 10 percent. The random result was shown in figure 4.5, and the yellow rectangles are the corresponding bounding boxes.

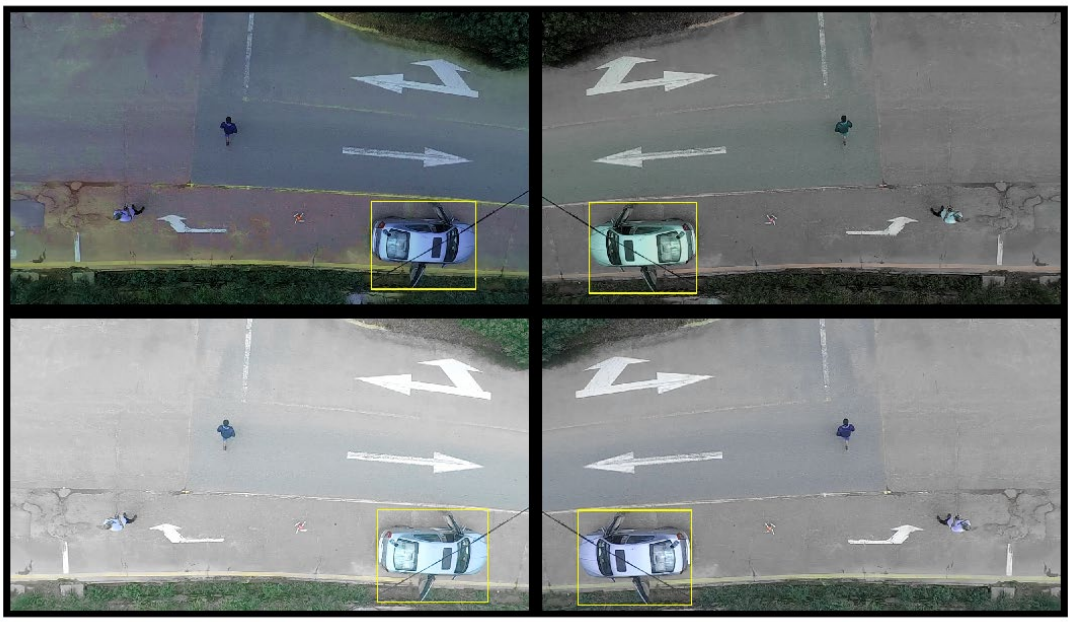


Figure 4.5 Data augmentation on AeroScapes dataset

4.2.2. Training Process

The training dataset contains various scales of vehicles. Therefore, the size of the anchor boxes is also various from each other. These anchor boxes were estimated using a k-means clustering algorithm based on the IoU distance metrics [60]. IoU is defined as the formula 4.2.1. As shown in figure 4.6, IoU measures the proportion of the overlap between the anchor box and the ground truth. Six anchor boxes were estimated based on the training data, and the mean IoU of these anchor boxes was 0.6743. To ensure the anchor boxes are well defined, the mean IoU needs to be greater than 0.5.

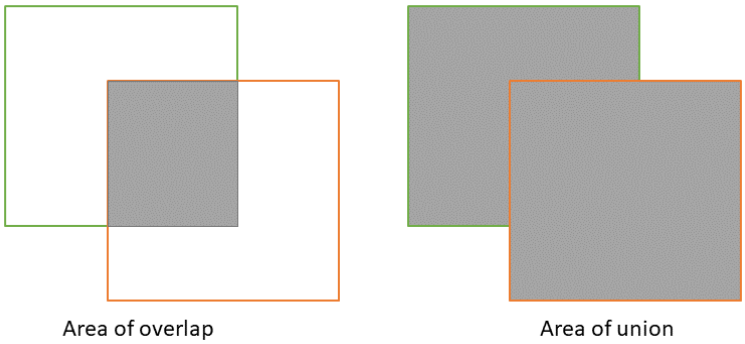


Figure 4.6 Intersection over union

$$IoU = \frac{Area\ of\ overlap}{Area\ of\ union} \tag{4.2.1}$$

The training parameters are defined in table 4.2. A minibatch queue was created to read data in batches, and the minibatch size was set to 16, considering the size of the training dataset. The current learning rate during the training process depended on the iteration number and the warmup period. A warmup period was introduced to stabilize the gradients at high learning rates, which means during the first 500 iterations, the learning rate would increase exponentially based on the formula (4.2.2):

$$learning\ rate \times \left(\frac{iteration}{Warmup\ period}\right)^4 \tag{4.2.2}$$

After the warmup period, set the learning rate equals to the max learning rate if the remaining number of epochs is less than 60 percent. Then if the remaining number of epochs is more than 60 percent but less than 90 percent, scale the learning rate by 0.1, otherwise scale the learning rate by 0.01 [55]. The penalty threshold was a threshold to filter out the detections that overlap with the ground truth less than 0.5.

Table 4.2 Training parameters for Aeroscape dataset

Parameters	Value
Number of Epochs	50
Max learning rate	0.001
L2 regularization factor	0.0005
Minibatch size	16
Warmup period	500
Penalty threshold	0.5

Due to the number of training samples and the depth of the network, the network was trained for 50 epochs, 1700 iterations in total. Therefore, the warmup period was set to 500 iterations, the first 30% iterations to stabilize the training process.

The loss used in this training process consists of three different loss functions, which are box loss, object loss, and class confidence loss.

- Box loss: Calculate the mean squared error using formula 4.2.3 between the prediction values and the ground truth values for each value in the bounding box vector, then sum it up.

$$loss = \frac{1}{N}\sum_{m=1}^{M}(P_m - T_m)^2 \tag{4.2.3}$$

Where P_m is the prediction from the network, T_m is the corresponding target value, M is the number of predictions of the network, N is the total number of observations in P.

- Object loss and class confidence loss: The binary cross-entropy function was used to calculate the object loss. For each network prediction P_n and the target value T_n, the loss value is calculated using the formula 4.2.4.

$$loss = -\frac{1}{N}\sum_{n=1}^{N}(T_n \log(P_n) + (1 - T_n)\log 1 - P_n) \tag{4.2.4}$$

Where N is the number of observations.

4.2.3. Transfer Learning

In order to implement the object detector in the simulation environment, the network needs to be tuned on the simulation dataset. Accordingly, as defined in section 4.1, the UAV global planner is first applied, and then images are saved while flying. After collecting the simulation images, manually label the cars using the Image labeler provided by MATLAB to create the bounding boxes for transfer learning (figure 4.7). There are 492 images collected for transfer learning in total. All images were taken from 50 meters in height. The camera settings were the same as in table 4.1. In this synthetic dataset, only images that contain complete vehicles were selected as training samples. Within the simulation dataset, 90% of images are used as training samples, and the rest 10% used as testing samples.

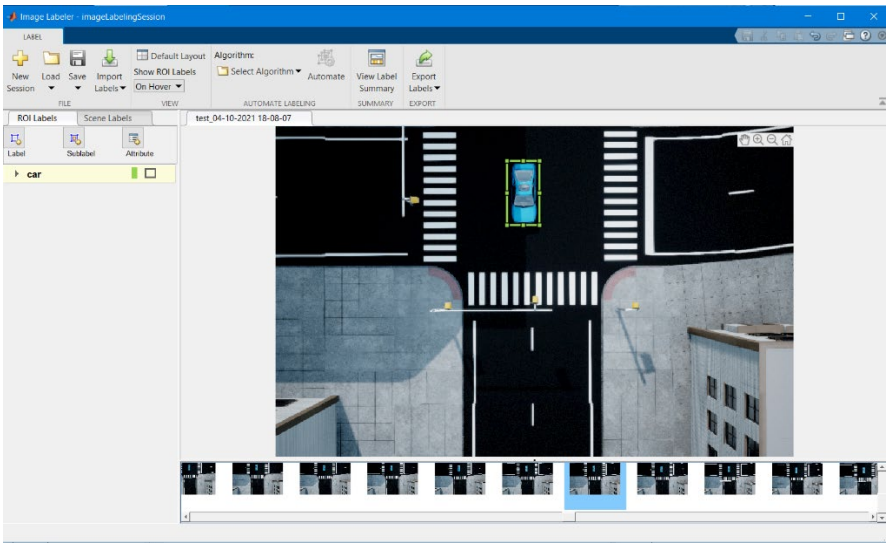


Figure 4.7 Manually labeling process

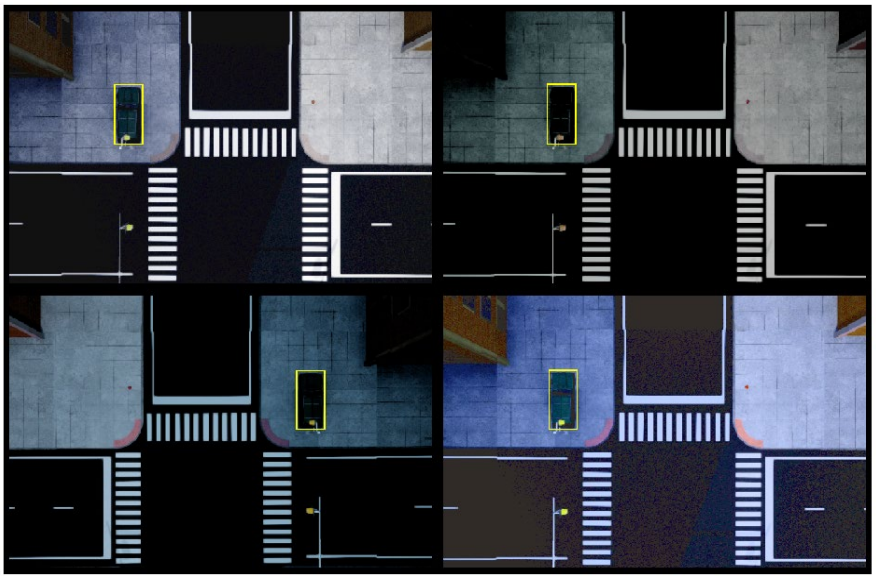


Figure 4.8 Data augmentation on simulation dataset

Then follow the same process of data augmentation as mentioned in 4.2.1. The results are shown in figure 4.8. Considering the feature of the local flight path, which is a circle, data augmentation techniques were added to improve the performance during the local flight path. Apart from the data augmentation process mentioned in section 4.2.1, a random rotation of 30 degrees and a random scaling by four times were added in order to obtain a multi-scale dataset. The new augmented data was shown in figure 4.9.

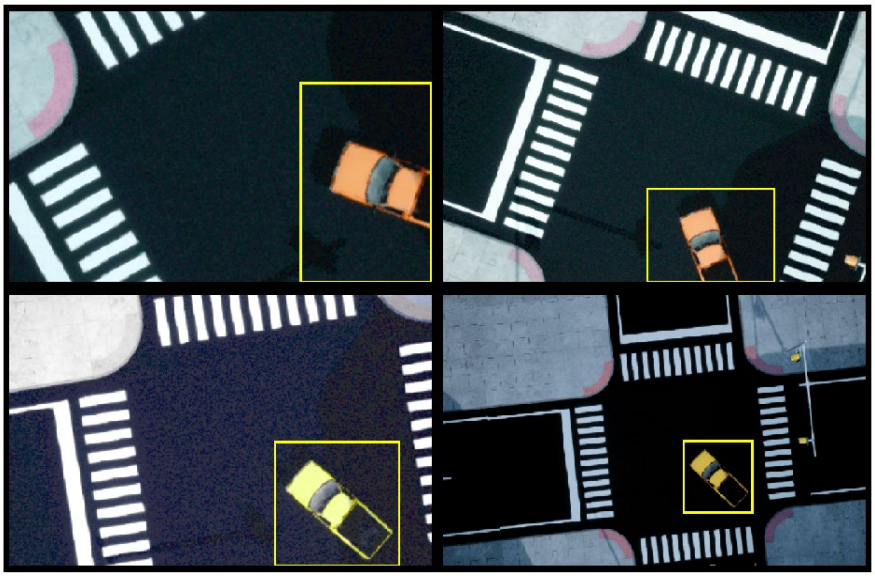


Figure 4.9 Rotation and scaling data augmentation

As for transfer learning, the last learnable layers need to be replaced. The final two convolutional 2D layers were replaced by new convolutional 2D layers called new_conv1 and new_conv2 with the same size as the old ones (figure 4.10). Then all the other layers were frozen except the final two layers, which means their learning rate factor was set to zero. After setting the learning rate factor, the network is finally prepared for training on the simulation image set.

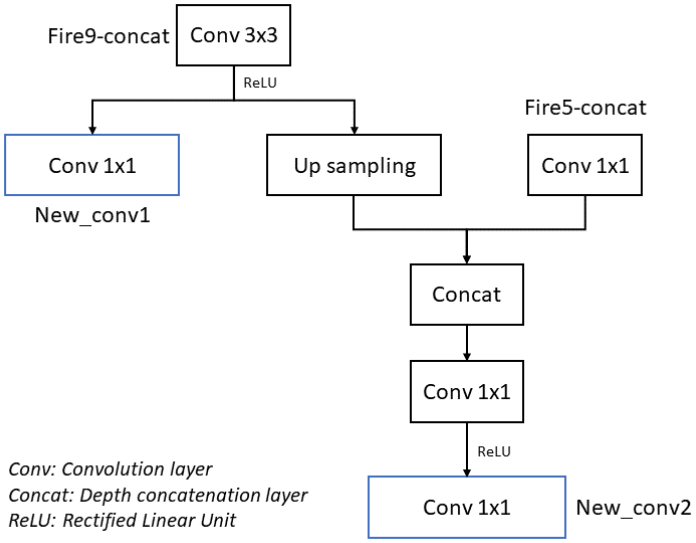


Figure 4.10 New network architecture

The new anchor boxes were estimated based on the simulation dataset. Because the flight height of the synthetic images was constant, all the vehicles are on the same scale. Therefore, the size of the anchor boxes is similar to each other, and the mean IoU is 0.9049. The training parameters for transfer learning are shown in table 4.3.

Table 4.3 Training parameters for transfer learning

Parameters	Value
Number of Epochs	25
Max learning rate	0.001
L2 regularization factor	0.0005
Minibatch size	8
Warmup period	100
Penalty threshold	0.5

The minibatch size was set smaller because the number of images available for transfer learning is smaller. Then set a shorter warmup period to speed up the training process. Besides, when implementing transfer learning, there is no need to train so many epochs. Therefore, the number of epochs was set to half of the previous training process.

4.3. Local Planner

The main idea for the local planner is a more detailed inspection of the target. Once the target has been detected, first calculate the location of the target without altitude. Then the UAV would fly towards the target, and a simple circular flight path was defined, in which the center is the target location, the flight height is 20 meters, the radius is three meters.

The flying height and the radius of the circular path mainly depend on the size of the camera. The larger the camera, the larger the possible field of view. Therefore, even images taken at a low flying height can still cover a larger area. However, suppose the camera is relatively small. When the flying altitude becomes lower, because the field of view becomes smaller, there may be situations where the vehicle cannot be captured. Nevertheless, large image size means long processing time, with the image size and camera setting defined in table 4.1, 20 meters flight height and three meters radius were selected.

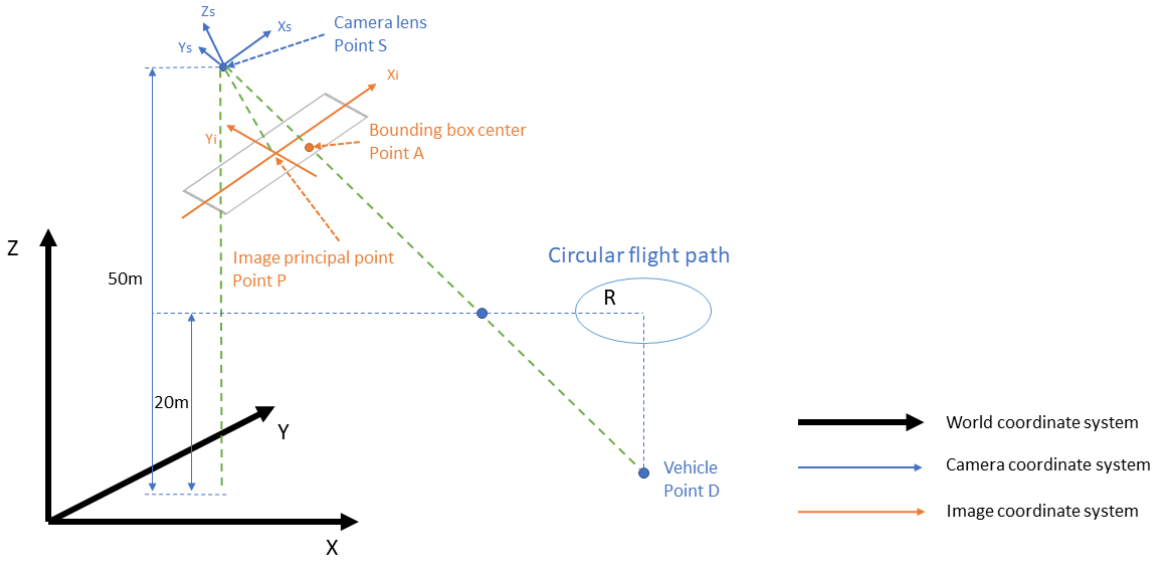


Figure 4.11 Local planner

As shown in figure 4.11, if a vehicle has been detected, then use the value of the bounding box center to calculate the vehicle location without altitude using the collinearity equation. Then take this coordinate as the center of the circular path. Once finish the circular flight path, return to the global flight path at 50 meters height and keep flying until the landing point of the global planner.

The collinearity equations are shown in formula 4.3.1.

$$\begin{bmatrix} x_a - x_p \\ y_a - y_p \\ -c \end{bmatrix} = \lambda R \begin{bmatrix} X_d - X_s \\ Y_d - Y_s \\ Z_d - Z_s \end{bmatrix} \tag{4.3.1}$$

$$x_a - x_p = \frac{r_{11}(X_d - X_s) + r_{12}(Y_d - Y_s) + r_{13}(Z_d - Z_s)}{r_{31}(X_d - X_s) + r_{32}(Y_d - Y_s) + r_{33}(Z_d - Z_s)}(-c) \tag{4.3.2}$$

$$y_a - y_p = \frac{r_{21}(X_d - X_s) + r_{22}(Y_d - Y_s) + r_{23}(Z_d - Z_s)}{r_{31}(X_d - X_s) + r_{32}(Y_d - Y_s) + r_{33}(Z_d - Z_s)}(-c) \tag{4.3.3}$$

Where $-c$: the focal length of the camera

$(x_a \quad y_a)$: the coordinate of point A in the image system

$(x_p \quad y_p)$: the coordinate of point P in the image system

$(X_d \quad Y_d \quad Z_d)$: the coordinate of point D in the world system

$(X_s \quad Y_s \quad Z_s)$: the coordinate of point S in the world system

$\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$: the rotation matrix between camera coordinate and world coordinate

As it is illustrated in figure 4.11, if the vehicle has been detected, then the center of the vehicle on the image (point A), the lens (point S), and the vehicle on the ground (point D) would be on the same line. Here comes an essential concept from photogrammetry [63], collinearity equations. For one point on the image, two equations can be defined: formulas 4.3.2 and 4.3.3.

Within these equations, nine parameters are defined, shown in table 4.4. The exterior orientation contains six parameters, three for the rotation angle between the camera coordinate system and the world coordinate system. The other three is the camera position in the world coordinate system, also known as translation. The interior orientation contains three parameters: the principal point of the image and the camera focal length. The principal point is defined as the point of the vertical projection of the lens on the image. Ideally, the principal point and the image center should be the same point, but there is an offset in a real camera. Therefore, in the formulas 4.3.2 and 4.3.3, this offset is eliminated.

Table 4.4 Exterior orientation and interior orientation

Exterior orientation			**Interior orientation**		
Rotation angle	$(\omega \quad \varphi \quad \kappa)$	Yaw, pitch, roll	Principal point	$(x_p \quad y_p)$	Center of the image
Translation	$(X_s \quad Y_s \quad Z_s)$	The position of the camera in the world coordinate	Focal length	c	The distance between lens and image

Hence, the transfer relation between these three coordinates, which are world coordinate, camera coordinate, and image coordinate, is illustrated in the formula (4.3.1). With both interior and exterior orientation are known, the vehicle location can be solved using the formula (4.3.2) and (4.3.3).

The images acquired from the local planner use the same camera settings as the global planner. Along this local circular flight path, the image will be obtained every second.

4.4. Simulink Implementation

The entire model was implemented based on the MATLAB example called UAV Package Delivery [61]. This example illustrated many functions that the UAV toolbox could achieve about multi-rotor UAVs, including obstacle avoidance, cuboid scenario simulation, and Photorealistic Simulation. However, the purpose of this example is more to introduce general blocks in the UAV toolbox, so it covers many other contents. The following only introduces the parts applied in this research.

4.4.1. Flight Planning

The UAV Package Delivery model contains four main parts: ground control station, external sensors, on board computer, and multi-rotor. Every part is encapsulated into a system, as shown in figure 4.12.

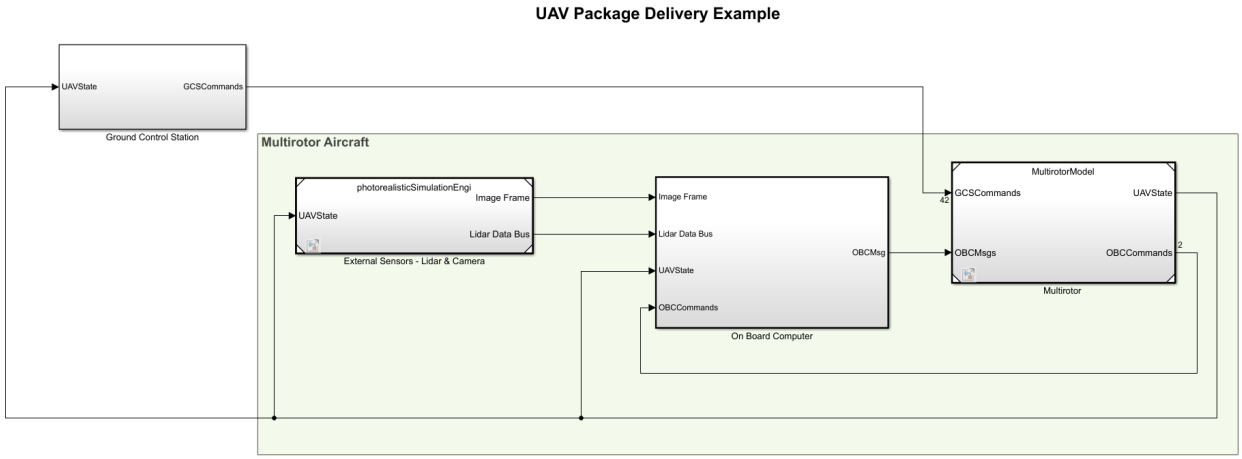


Figure 4.12 MATLAB UAV package delivery example [61]

The ground control station is the part to set the flight path point by point. The whole global planner was defined there. The execution of the flight depended on a block called UAV Path Manager [62]. This block has five inputs shown in table 4.5: current UAV pose, UAV mission data, mission command, UAV home location, and a boolean variable called IsModeDone.

Table 4.5 Input of UAV Path Manager

Input variables	Value
Pose	[x; y; z; course angle]
Mission data	Mode, position, parameters
Mission command	0; 1; 2; 3;
Home	[x; y; z]
IsModeDone	0/1

The current pose of the UAV is defined by a vector with four factors. The x, y, and z are the UAV location in the north-east-down (NED) coordinate defined in meters. The course angle is the heading angle defined in radius. In other words, the angle between the direction of the current velocity vector and the north direction and the range of course angle is $[-\pi, \pi]$. The mission data is a series of points that specifies the flight path. It is defined in a MATLAB struct with three properties: mode, position, and params. The mode defines the character of the point, which contains six different modes: takeoff, waypoint, orbit, land, RTL, and custom.

Among them, orbit means fly along the circle defined by corresponding parameters, RTL stands for return to the launch position, and custom means that users can define custom mode by themselves. At last, the params defines the parameters related to each mode in a four-column vector. For waypoint mode, yaw angle and transition radius need to be specified. Nonetheless, as for multi-rotor UAVs, the transition radius can be set as zero. For the orbit mode, the radius of the orbit, the turn direction, and the number of turns need to be specified. For other modes except for the custom, params can be set to zero.

As for the mission command, it defines the way to execute the mission. Four possible values are listed:

- 0: all the mission point will be executed sequentially
- 1: the UAV will hold still on the current mission point
- 2: the UAV will repeat the whole mission after arriving at the last point
- 3: the UAV will change into the RTL mode

Besides, the home factor is a three-column vector that specifies the home location of the UAV, and the IsModeDone factor is the Boolean variable to check whether each point has been executed or not.
Therefore, in the ground control station part, the pre-defined flight was defined in the format of mission data and feed into the Multirotor part.

The multirotor contains two subsystems: the guidance logic of the flight and the low-fidelity control system. Both of them were developed by Mathworks. The core of the guidance logic is designed based on the UAV Path Manager block mentioned above. The full guidance model is shown in figure 4.13.

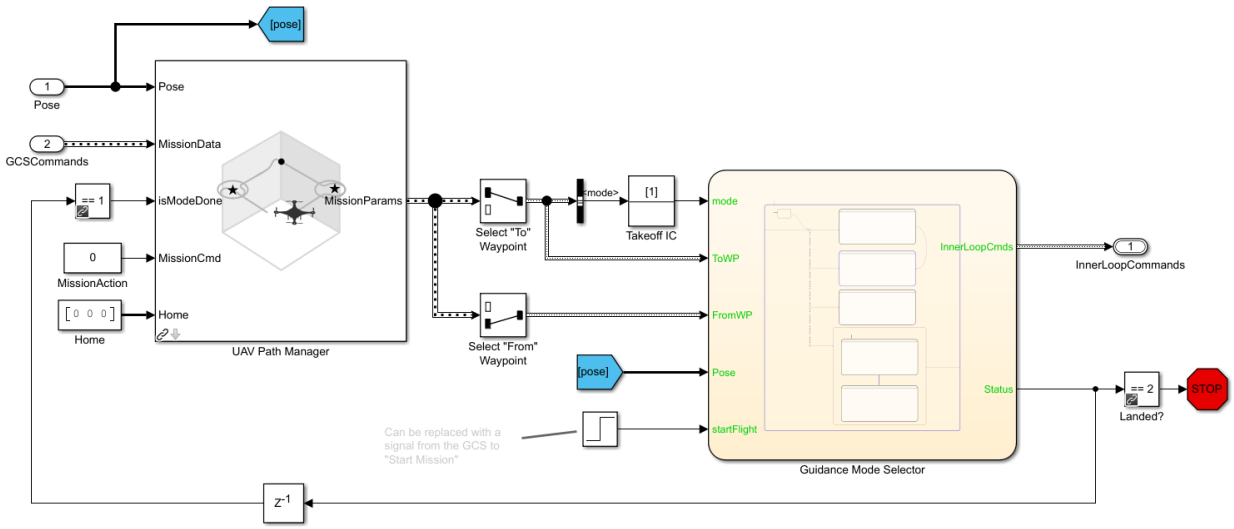


Figure 4.13 Full guidance logic[61]

The full guidance logic subsystem uses the UAV Path Manager block to distinguish the current waypoint and the next waypoint. The next waypoint is passed into the Guidance Mode Selector Stateflow chart. The Guidance Mode Selector Stateflow chart will decide how to fly from the current waypoint to the next waypoint according to different modes and generate the necessary position control commands used in the low-fidelity control system [61]. The main method of the low fidelity control system is to use a simple PID (proportional, integral, and derivative) system (position and acceleration control in figure 4.14) to simulate the motion of the UAV. Three aspects were involved, which are position, acceleration, and attitude.

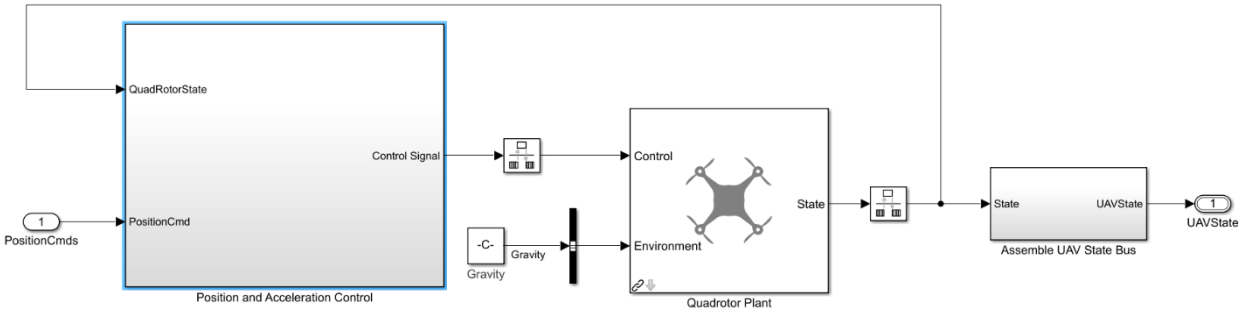


Figure 4.14 Low-fidelity control system

Then, the control signal is passed into the guidance block (Quadrotor Plant in figure 4.14). The guidance block will finally compute the current UAV state, which is the current position and attitude. These are also the six exterior orientation parameters.

The blocks used to achieve the photorealistic simulation environment are shown in figure 4.15. For configuring the 3D simulation scene, the Simulation 3D Scene Configuration block is needed. Besides, there is a Simulation 3D UAV Vehicle block to show the UAV in the 3D environment. The camera is modeled in a simulation 3D camera with parameters specified in table 4.1, and the output image is an RGB image in uint8 format. The LiDAR sensor was disabled (grey) in this study.

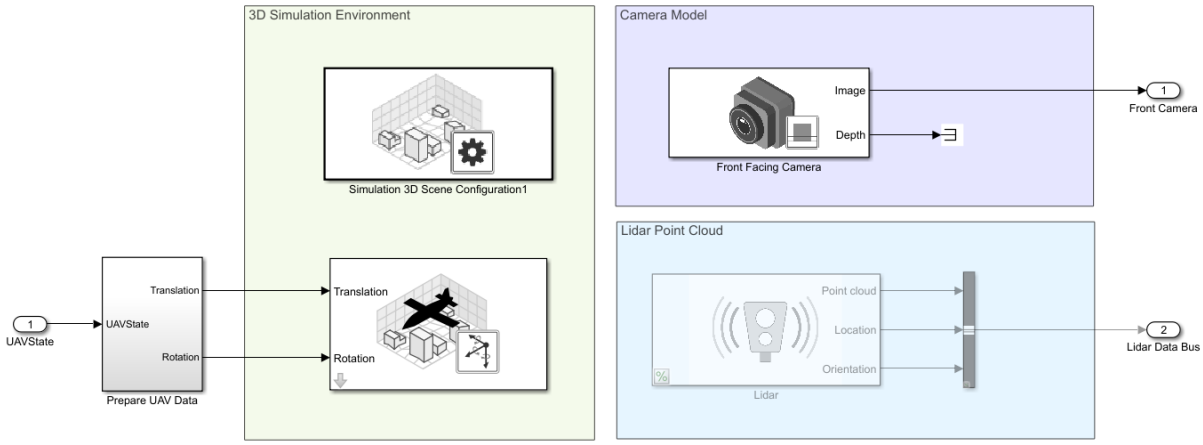


Figure 4.15 External sensor

In summary, the whole UAV flight was executed based on the waypoints defined in the ground control station system and the guidance logic in the multi-rotor system. The images were captured by the simulation camera every second, and the 3D simulation environment was configured through the simulation blocks.

4.4.2. Object Detection

With the image captured by the simulation camera, the object detection function was added to import the fine-tuned object detector in section 4.2.

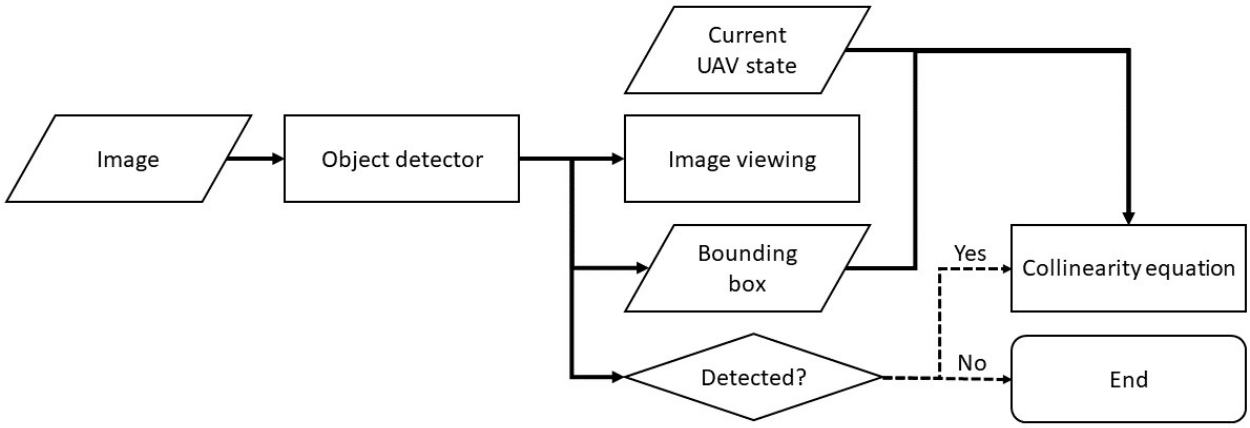


Figure 4.16 Object detection workflow

The workflow of the object detection process is illustrated in figure 4.16. The output of the object detection function contains three elements: annotated image, bounding box vector, and a Boolean value. The annotated image was passed to the image viewer to allow a real-time visual inspection. The bounding box vector was the input of the collinearity equation. Another input is the current UAV state, which contains the position and attitude values. In this model, the offset between the UAV and the camera was ignored. Hence, the current UAV state is the current camera position and attitude, and these are also the exterior orientation parameters. Finally, the Boolean output is to decide whether there was a car in the image or not. If a vehicle has been detected, then use the value of the bounding box to calculate the vehicle location without altitude using the collinearity equation. Otherwise, the four elements in the bounding box would be zero.

The center of the circular flight path can be decided by using the equations mentioned in section 4.3. The next step is to forward this new point into the full guidance logic (figure 4.13) to update the flight path.

4.4.3. Flight Path Update

The ground control station subsystem needs to accept runtime changes, to achieve flight path re-planning, which means the flight path must be dynamic while the simulation was running. As mentioned in section 4.4.1, the pre-set flight path was saved in a MATLAB struct with three attributes. Therefore, the general idea of updating the waypoint list is to insert the new waypoint after the current waypoint.

In section 4.1, the global flight path was broken down into small pieces every ten meters. In this case, there are 322 waypoints in total. After reaching every waypoint, there would be an *if-else* statement to decide how to fly next using the IsModeDone parameter, as shown in Figure 4.17. If the vehicle has been detected during the flight, then the new waypoint calculated by the collinearity equation exists. Then update the flight path by inserting the new waypoint in the old flight mission. If there is no vehicle, then fly as the pre-defined flight path. The length of this waypoint list was set to 400, which is far longer than 322. Therefore, the overflow problem was not considered. The insert process is shown in figure 4.19. When a new waypoint exists, insert his new waypoint

With the new waypoint, the new UAV mission will be created and forward to the ground control station part. Then, within the ground control station subsystem, another *if-else* statement exists to decide whether use the old mission data (global planner), or the new mission data, as shown in figure 4.18.

There is a memory block [63] in figure 4.17. The reason why using memory block is that the new mission data was calculated based on the result of image object detection. However, before the UAV takes off, there is no image available. In this case, there will be an algebra loop [64] within the Simulink model, which means a block using the output of this block as its input. Since Simulink is a continuous dynamic system, there needs a delay to break this loop.

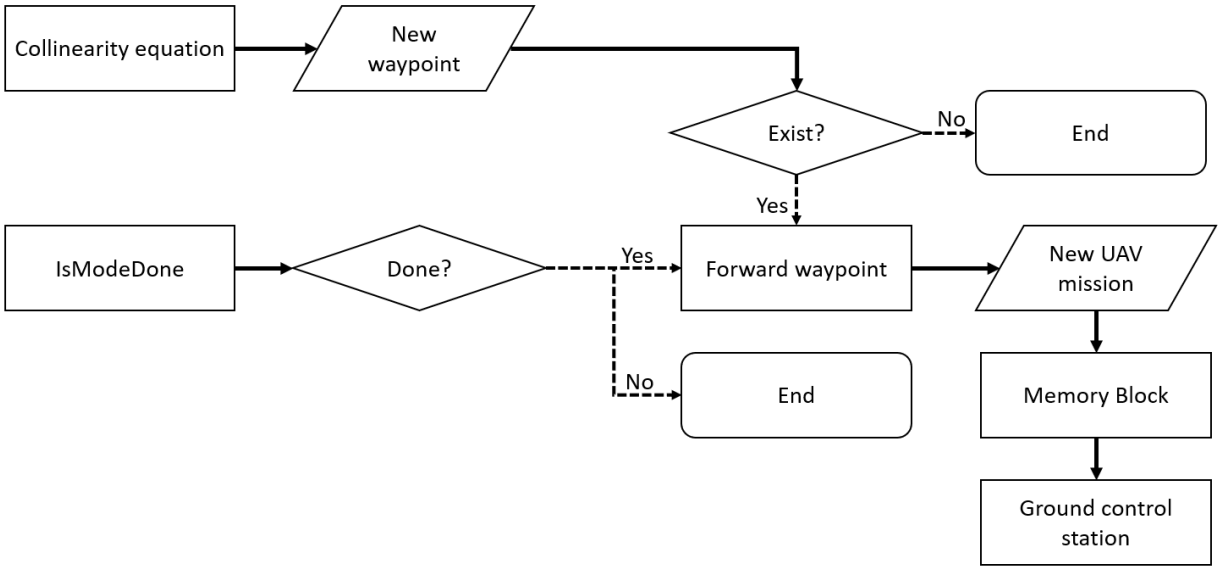


Figure 4.17 UAV mission update workflow

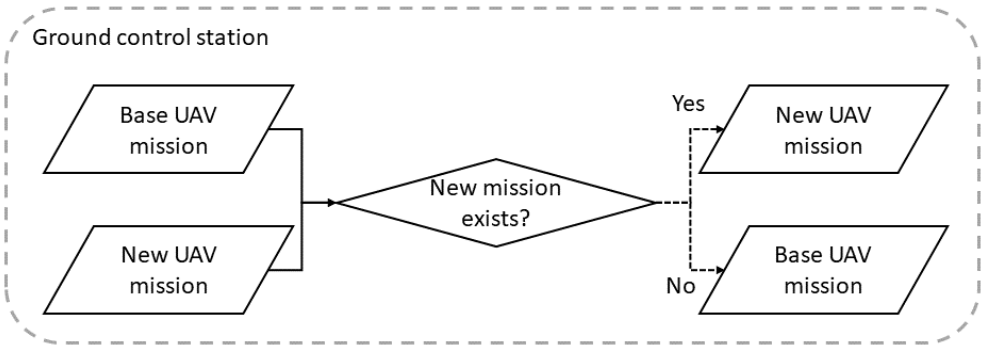


Figure 4.18 Dynamic waypoint update system

As shown in figure 4.18, to update the flight plan, an *if-else* statement was used to decide which mission will be used. For example, if the new UAV mission (figure 4.17) exists, pass this new mission to the guidance logic (figure 4.13). Otherwise, keep using the original flight plan defined by the global planner.

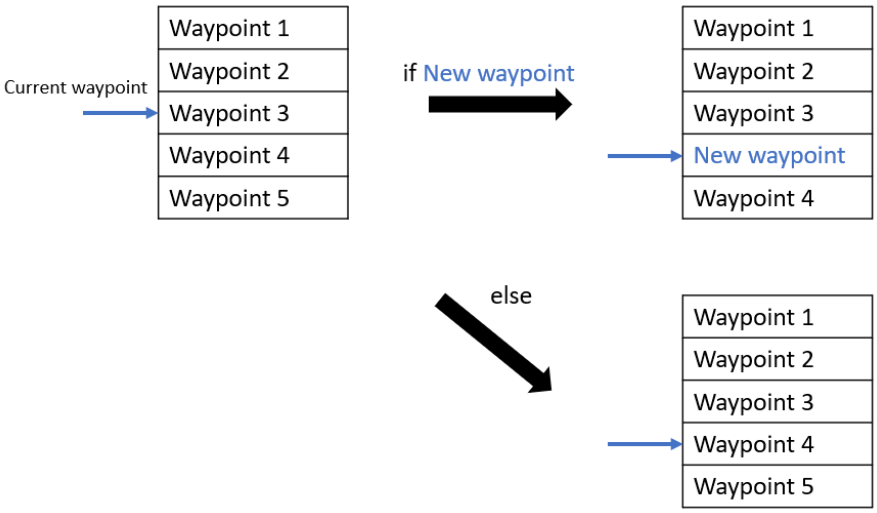


Figure 4.19 Insert a new waypoint

As illustrated in figure 4.19, when a new waypoint exists, insert this new waypoint into the original flight mission and then create a new mission. Otherwise, just fly towards the next waypoint as the original flight plan.

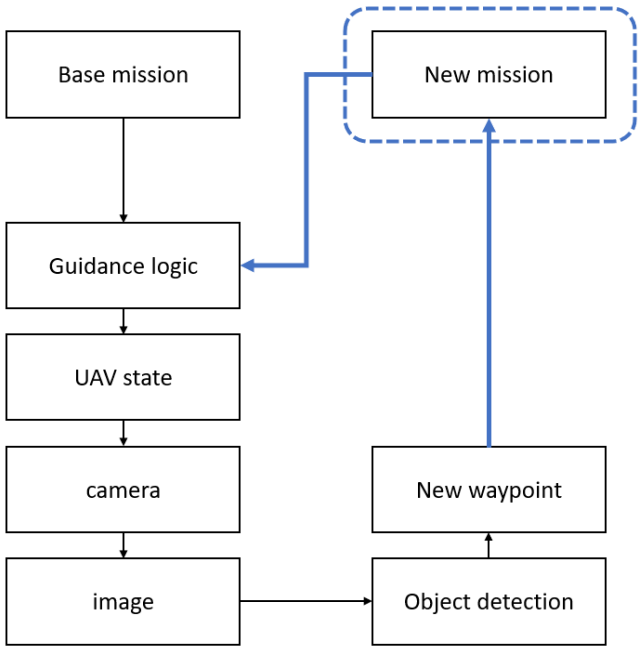


Figure 4.20 Algebra loop

As illustrated in figure 4.20, the new mission becomes both input and output of this model. Hence, a memory block has been added between the new mission and the guidance logic. The function of the memory block is to hold and delay its input by one iteration, and then there will be a time difference within the model so that the algebra loop can be solved. At last, this model will keep running until it reaches the final landing point.

5. RESULTS AND DISCUSSIONS

In this chapter, the proposed flight planning approach will be experimented with using Simulink. The results of the object detector and the flight path were shown by images obtained while the simulation was running. The strength and weakness of the result has been discussed.

5.1. Object Detection using YOLO v3

5.1.1. Training Result

The precision and the recall are used to evaluate the detection result on the test dataset. Precision is defined as the true positives divided by all the positives of the detection (5.1.1). The recall is defined as the true positives divided by all the real positives (5.2.2).

Precision:

$$precision = \frac{TP}{TP + FP} \tag{5.1.1}$$

Recall:

$$recall = \frac{TP}{TP + FN} \tag{5.1.2}$$

Where TP is the true positives, FP is false positives, and FN is false negatives.

In an ideal condition, precision equals one at all recall levels. In actual situations, the more convex the P-R curve to the upper left, the better the neural network results. Here in figure 5.1, in the test set, the lowest precision is greater than 0.93, which means most vehicles in the test set had been detected. The average precision is the mean precision of all detection results. With an average precision of 90 percent, this trained YOLO v3 object detector performs well on the test dataset.

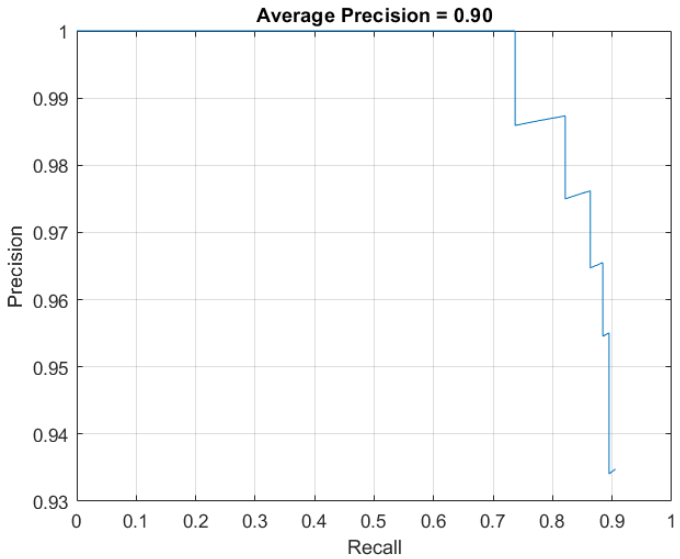


Figure 5.1 P-R curve on Aeroscape dataset

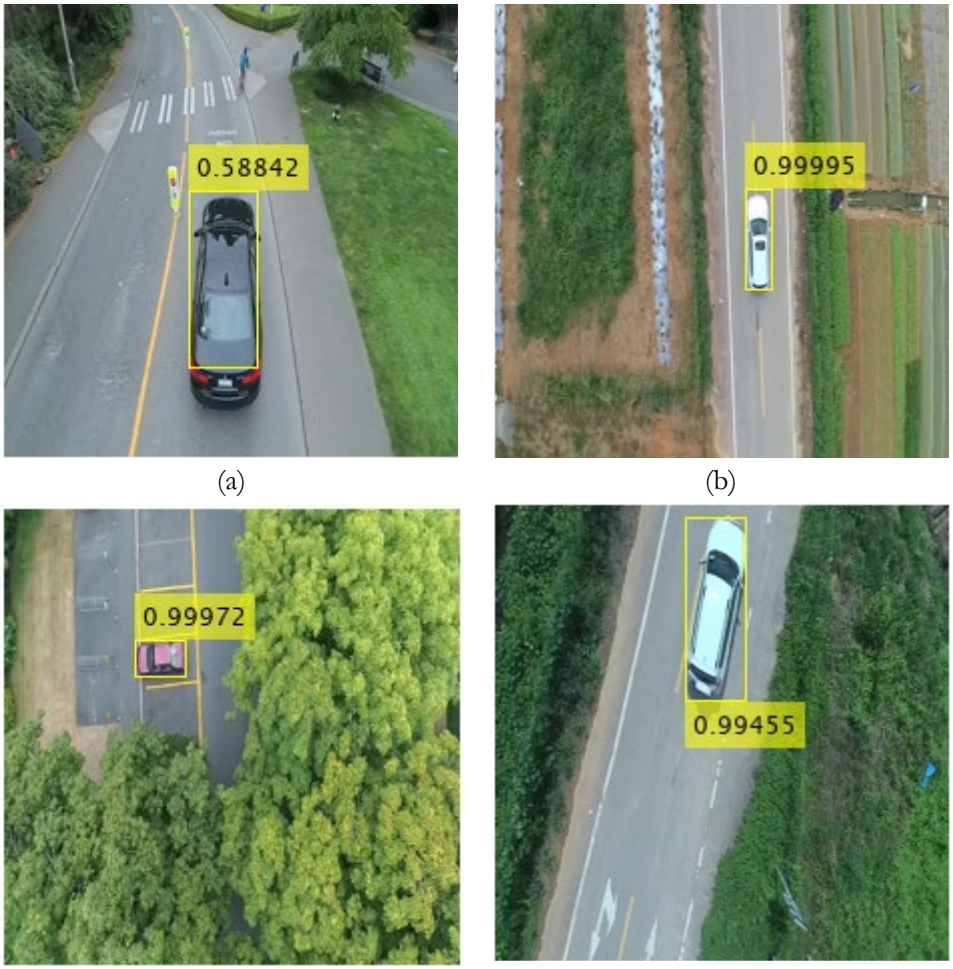


Figure 5.2 Detection result of Aeroscape dataset

The detection result of training on Aeroscape dataset is shown in figure 5.2. From figure 5.2 (b), (c), and (d), the network has better results in detecting vehicles that were farther away with a high confidence score, while the detection results of vehicles that are closer, which was also very accurate (figure 5.2 (a)), but the confidence score is low.

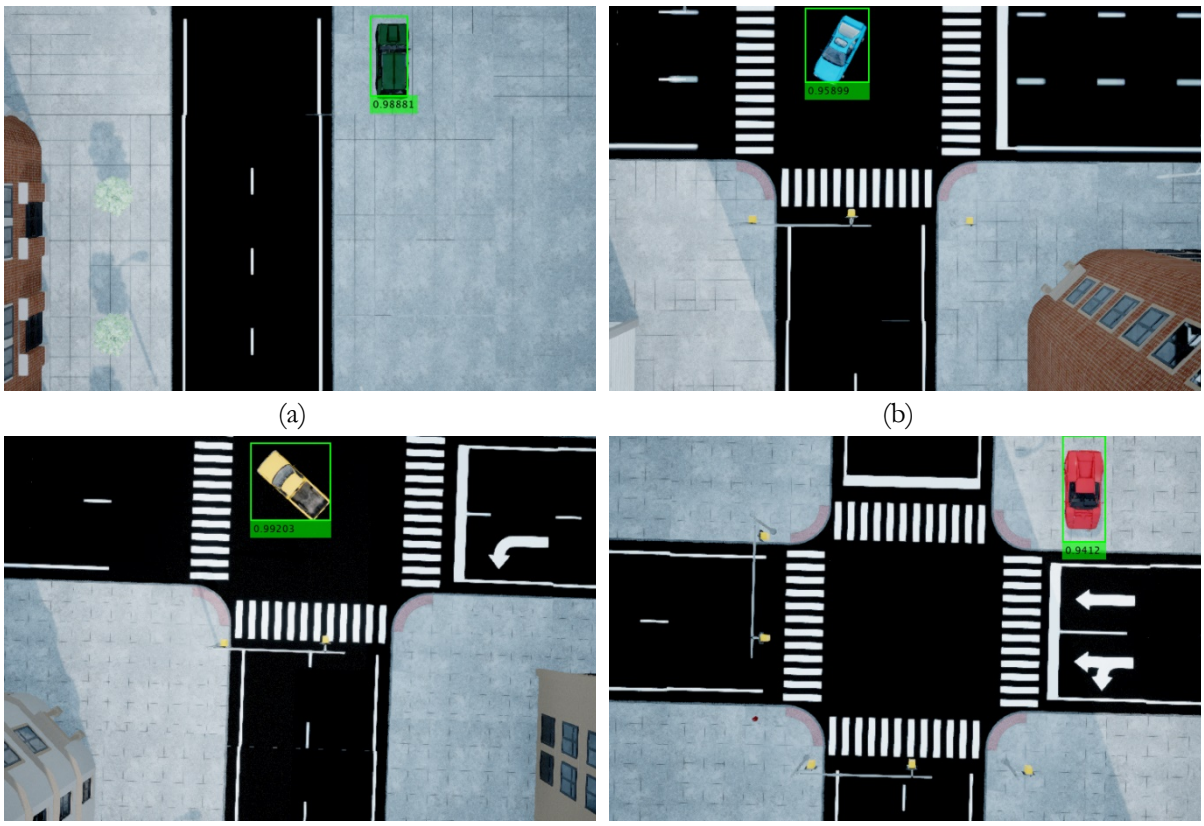
After performing the transfer learning, the average precision increase to 100%. This is because the synthetic dataset is simple and limited. Therefore, the network can perform very well in the test dataset. Moreover, the test result using rotation and scaling data augmentation method also achieve an average precision of 98%.

The overall training results accessed by the P-R curve and average precision indicate that the network performs well on the test dataset. However, the test dataset used in both Aeroscape dataset and the synthetic dataset was only 10% images of the whole dataset. Therefore, the results obtained on this limited test dataset cannot fully reflect the actual accuracy of the network.

5.1.2. Object Detection Result in the Simulation Scene

When running the object detector in the simulation environment, the performance becomes different at two different flight heights. At 50 meters flight height, all types of vehicles can be detected precisely with high confidence scores shown in figure 5.3. This result consists of the testing result obtained in section 5.1.1. However, at 20 meters height, the detector performed worse.

The object detector worked well at 50 meters height, but there were some false positives during the takeoff process shown in figure 5.4. Therefore, to avoid the false detection problem at the beginning, the collinearity function can only be triggered after the takeoff process. This is because that only 50 meters images were used as training samples, and the simulation dataset was too small.

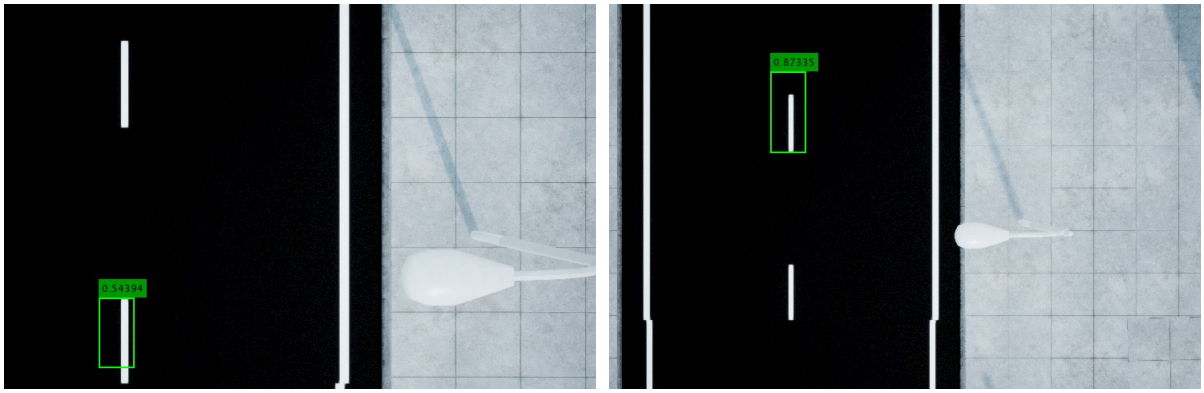


(a) (b)

(c) (d)

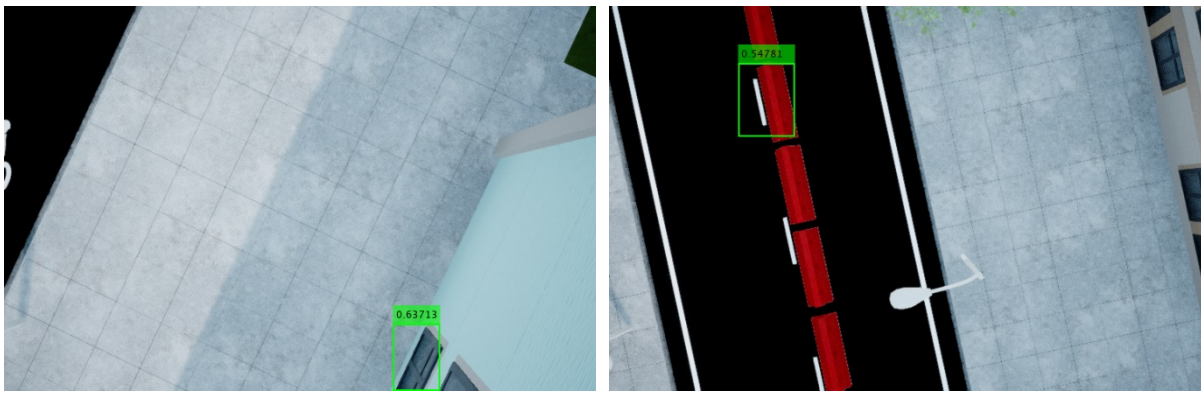
Figure 5.3 Vehicle detection

These false positives happened during the takeoff procedure, which recognized the rectangle road marks as the vehicle (figure 5.4). One possible reason could be that the shape of the road mark is similar to the vehicle so that their patterns are similar when at a low flight height. Besides, when training the network, the image was first resized into 227x227. During the resize process, the bounding box and the vehicle can be compressed and become thinner, which may be similar to the road mark.



(a) (b)

Figure 5.4 False detection during takeoff



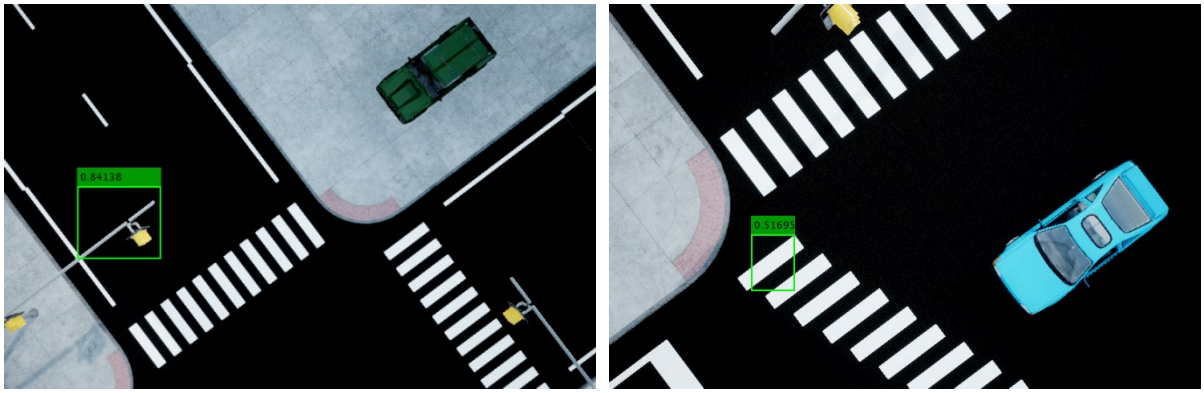
(a) (b)

Figure 5.5 False detection on the window and road barriers

However, there also were false detections at 50 meters height. This implied the insufficient training sample for the simulation environment. From figure 5.5 (a), the window close to the ground were detected as the vehicle. This is because the window also has a similar pattern to the window in the vehicle. This kind of error can also cause another problem, which is the error of the target location. As assumed in section 4.3, the target location was calculated based on the assumption that the altitude of the vehicle is zero. If the object detector recognized the window as a vehicle, then the target location will be wrong because the altitude of the window is not zero.

As for (b) in figure 5.5, one possible reason is the insufficient distribution of vehicles. There was a red vehicle in the training dataset (figure 5.3 (d)), but here along this road with red barriers, there was no vehicle placed. Therefore, the detector cannot distinguish the barrier and the real vehicle.

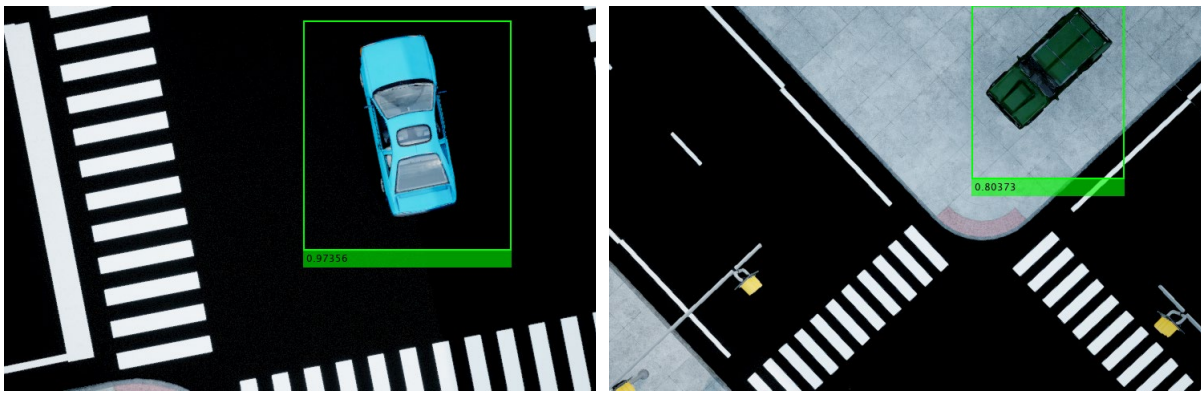
Like the situation of taking off, when the UAV was descending to complete the circular flight, false detections also existed, as shown in figure 5.6. Those false positives during ascending and descending processes indicate that the trained network performs poorly at lower flight height. Although two detection heads were developed in this detector to achieve multi-scale object detection, the result is not ideal.



(a) (b)

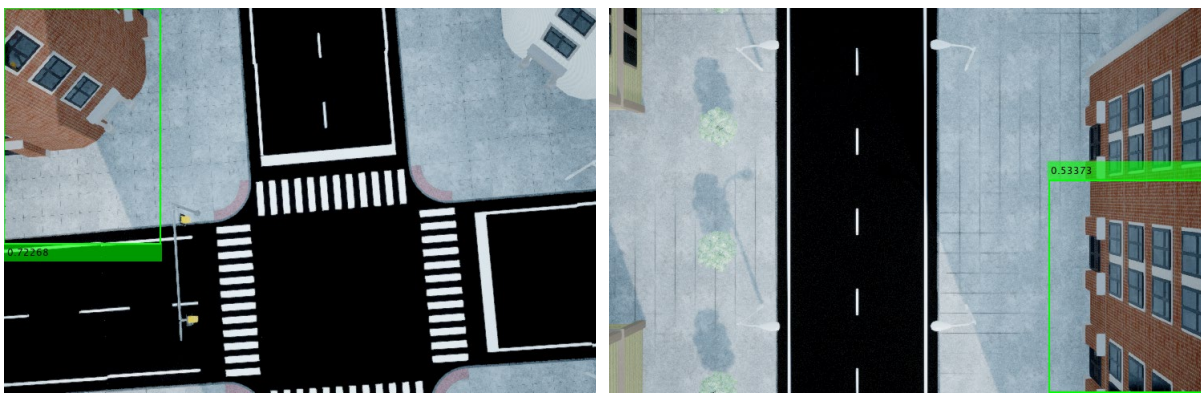
Figure 5.6 False detection at 20 meters

From above, the main drawback of the object detection process is the insufficient detection result at 20 meters high. By improving the data augmentation method, use this new augmented dataset as the input of the transfer learning process and test the new detector in the simulation environment. The results are shown in figure 5.7. The detection result at 20 meters height indeed improved by using new data augmentation strategies. Nevertheless, the bounding box predicted was not so precise.



(a) (b)

Figure 5.7 Detection results with new data augmentation process



(a) (b)

Figure 5.8 False detection of the new detector at 50 meters high

Table 5.1 Detection results of two detectors

	Global planner			**Local planner (i.e., blue vehicle)**		
	True detection	False detection	Miss detection	True detection	False detection	Miss detection
Detector 1	9	8	0	462	101	477
percentage	52.9%	47.1%	0%	44.4%	9.7%	45.9%
Detector 2	7	16	2	1045	30	81
percentage	28%	64%	8%	90.4%	2.6%	7.0%

As illustrated in table 5.1, detector 1 refers to the previous version detector, and detector 2 refers to the new detector using new augmentation methods. The performance of detectors can have three categories, true detection, false detection, and miss detection. True detection stands for detections that detected vehicles as the vehicle. False detection stands for detections that detected other objects as a vehicle. The miss detection in the global planner part stands for the vehicle without circular flight execution. The miss detection in the local planner part stands for the lost tracking issue in the local planner.

For detector 1, when tested in the global planner, it detected all vehicles in the scene but also including eight false detections. Still, it achieved a 52.9% accuracy. However, detector 2 performed much worse than detector 1 when using the global planner. It only had seven true detections and even missed two vehicles, but 16 false detections, which is as twice as detector 1. However, when tested in the local planner, the result became the opposite. For detector 1, the true detection and the miss detection are almost the same. But for detector 2, the precision was largely improved.

From this experiment, the main problem of the object detection process is still the lack of training samples. Adding rotation and a scale factor in the data augmentation process can help to improve the detection result at lower flight height, but this also sacrificed the detection accuracy at 50 meters height. Also, with a loose bounding box, when calculating the orbit center of the local flight path, there might be an offset between the bounding box center and the vehicle. As a result, the local path was not located at the exact location of the vehicle, and images obtained from the local planner became incomplete.

5.2. Flight Path Planning

5.2.1. Flight Trajectory

The flight trajectory was captured using the UAV animation block. Part of the flight trajectory was shown in figure 5.9. After take-off, the UAV will first follow the flight path defined by the local planner at 50 meters height. Then, once a vehicle has been detected, the UAV will fly towards the vehicle.

When it reached 20 meters height, the circular flight path would be executed. Once the circular flight is complete, switch back to the global planner and keep flying. When switching from the local planner to the global planner, there was a small fluctuation. This is the result of the low-fidelity control system. While ascending, there is a small buffer process to stable the flight. This small fluctuation will also happen when the UAV turns.

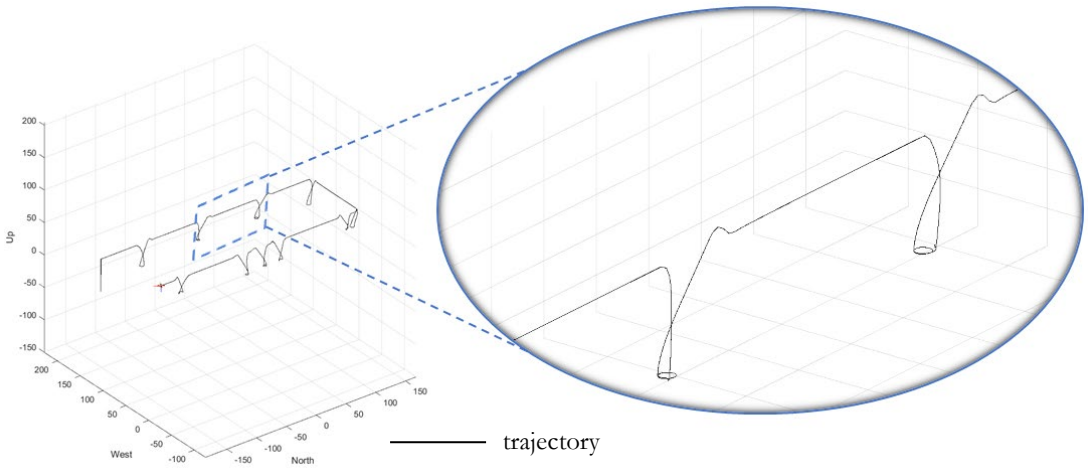
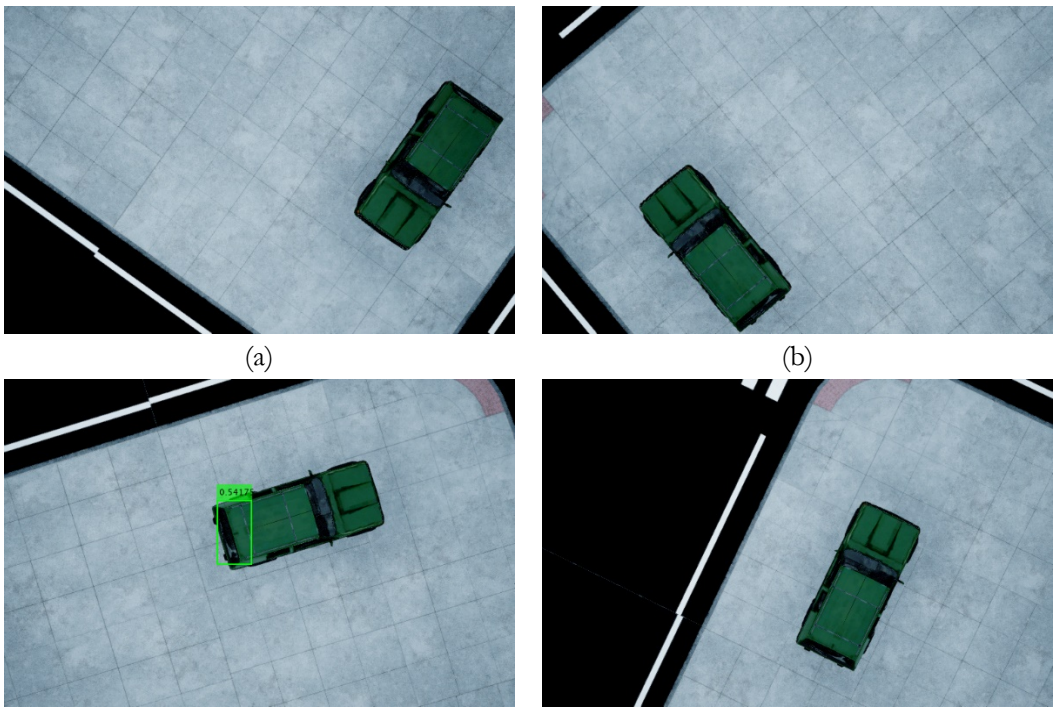


Figure 5.9 Flight trajectory

5.2.2. Vehicle Inspection

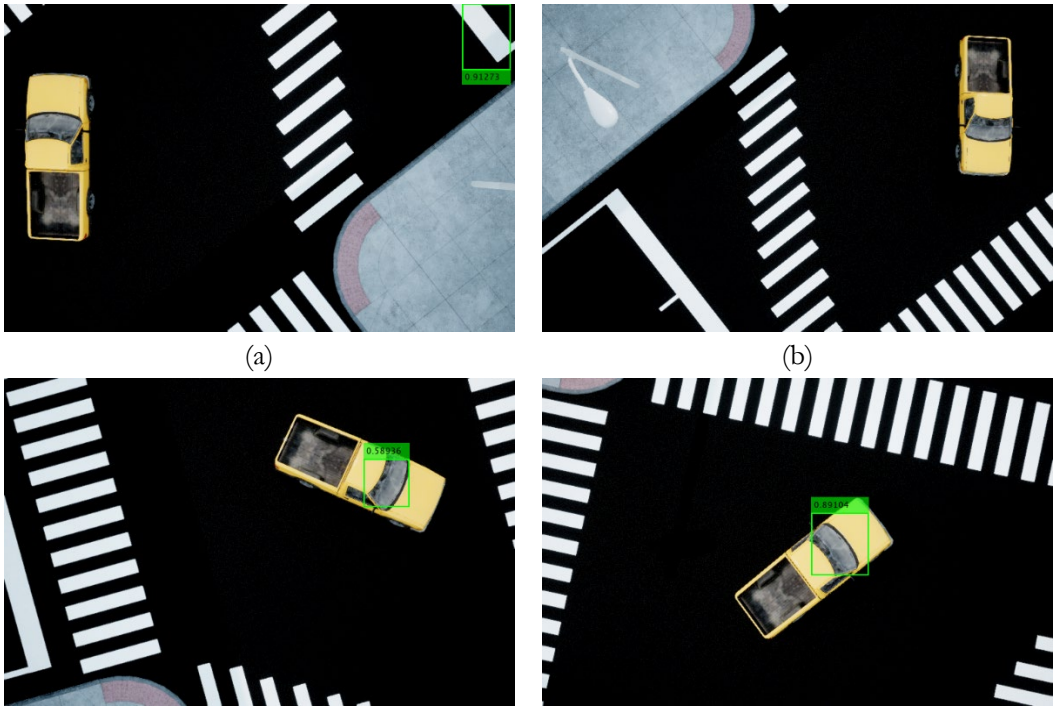
The circular flight path was achieved by using the orbit mode of the UAV path manager block. The main purpose is to get closer to the target and obtain a detailed inspection. On average, for each vehicle, around 673 images were acquired. With a nadir view, the side face of the vehicle was not so clear in the image. But in the case of the green one (figure 5.10) and the yellow one (figure 5.11), the side windows and doors, even the wheels, can be seen a little. For all four cases, the left and right sides can be seen better than the front and back sides of the vehicle.



(a) (b)

(c) (d)

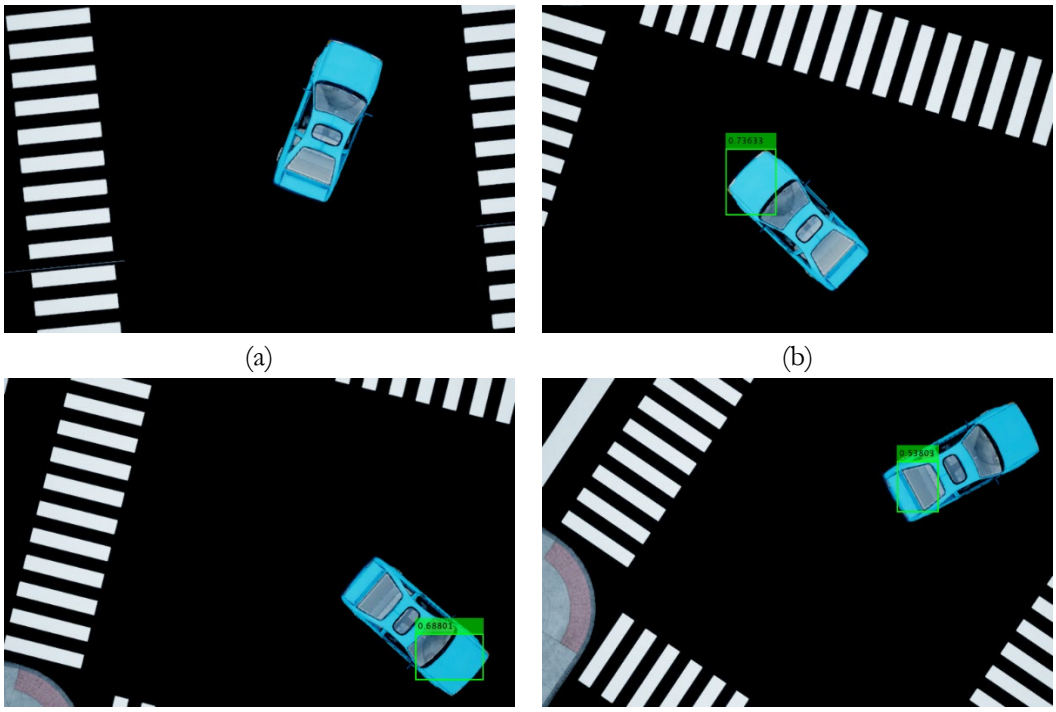
Figure 5.10 Green vehicle inspection



(a) (b)

(c) (d)

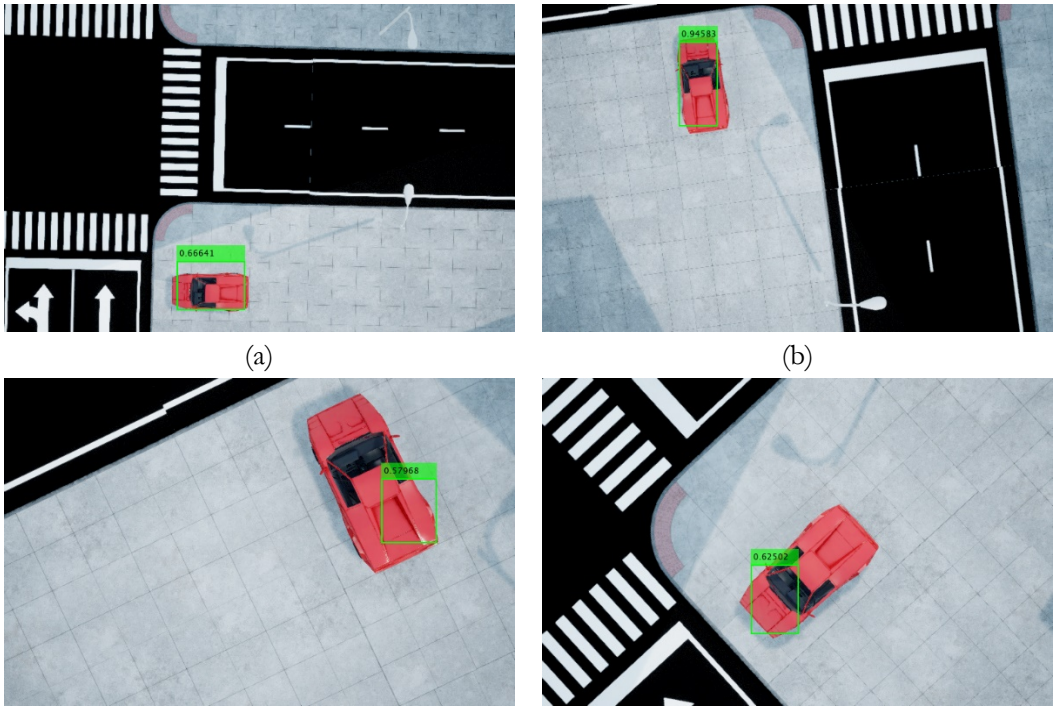
Figure 5.11 Yellow vehicle inspection



(a) (b)

(c) (d)

Figure 5.12 Blue vehicle inspection



(a) (b)

(c) (d)

Figure 5.13 Red vehicle inspection

Although the side face of the blue one (figure 5.12) and the red one (figure 5.13) was not so clear, the flight path allowed the UAV to capture vehicles in different directions and at different scales. Besides, from the detailed images, the object detector also performs worse at 20 meters flight height. At lower flight height, the object detector only detects part of the car, and the false detection problem still exists.

The whole model can run smoothly, with no errors, but it took nearly six hours to cover the whole area of the scene, which is very slow. This is caused by the memory block included in the model will slow down the process (figure 4.17). Moreover, many MATLAB functions were used in the model. For instance, within the object detection function, the "coder.extrinsic" command was used to call another customized function outside the Simulink model, this will lead to an extra process, which is the packaging process of the block data, these data will be packaged into MATLAB arrays, and this takes additional time and temporary memory. If large amounts of data have been passed through the block, then the running speed will decrease remarkably [65].

Apart from the speed, all functions defined in section 4 were realized properly. The blocks provided by the UAV toolbox can be easily implemented in the model. The graphical modeling method can be convenient for knowing the transmission of the signal but also easily result in a mess of connections when the model becomes complicated.

5.3. Discussions

5.3.1. Object Detection using YOLO v3

In the training process, two types of the dataset have been used. The idea is to combine real-world data and simulation data in order to increase robustness and prevent overfitting. The detection result tested on the test dataset was pretty good accessed by the P-R curve. However, from the result of the simulation, the overfitting issue still exists. Besides, although the YOLO v3 detector is an object detector for multi-scale application, the result is not so ideal either. This indicates that even using two different size detection heads, the scale difference between 50 meters and 20 meters is still larger than the scale difference between two detection heads. This is also because the network input size was set to a small value compared to the original image size.

From the result in section 5.1.2, the most serious problem is the lack of training samples. The accuracy illustrated in table 5.1 indicated that the size of the training dataset used in this study is not enough to train a convolutional neural network adequately. Although there are many public datasets available for object detection, i.e., the ImageNet [57], Microsoft COCO [65], and some other dataset with specific objects, for example, CelebFaces Attributes Dataset (CelebA)[66], which is a large dataset for faces especially. However, datasets of UAV images are very limited. Apart from the Aeroscape dataset, only a few datasets are available for vehicle detection, for example, UAVid [67] and UAVDT benchmark [68]. Although the dataset can be created by fieldwork, with the COVID-19 situation, fieldwork becomes difficult to arrange, and the Dutch weather is not so friendly to UAVs in winter.

If enough training samples can be obtained, changing the data augmentation method can be a solution to improve the network performance on the synthetic dataset. In section 5.1.1, by adding rotation and scaling in the data augmentation process, the detection result at 20 meters in the simulation scene had been significantly improved. This indicates that data augmentation strategies can be a solution to improve the performance of the network. Nevertheless, the detection result became worse at 50 meters height. Hence, the most important issue is still the number of training samples.

Moreover, this model cannot distinguish multiple targets along the flight path, and only the latest detected target will be monitored. If multiple detections happened along the flight path, only the last one would be recorded. Then, if the last one was a false detection while there was a vehicle along this path, this vehicle will be skipped. This mechanism also increased the possibility of miss detection in the global planner.

In this study, the detection result at 50 meters height is more important because the global planner was set at this level. Besides, the false detection at 50 meters level will result in false detection at 20 meters level. Therefore, the detector should improve the performance at 20 meters level as much as possible while maintaining the performance at 50 meters level.

If the number of training samples is sufficient, then using different data augmentation methods can help to increase the variety within the training set. With enough training data on both flight heights, the object tracking issue in the local planner can be improved. Besides, when performing transfer learning, all layers were frozen except the output layers. Since the transfer learning was from real dataset to synthetic dataset, free more layers may be another option to improve the network performance.

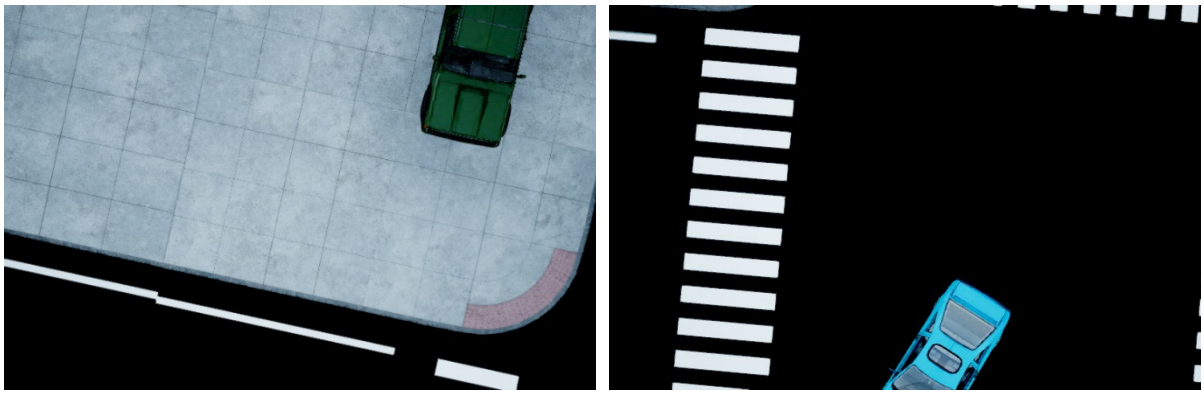
5.3.2. Flight Path Planning

The entire flight path planning process is successfully implemented in Simulink, and the UAV can automatically update the flight path according to the object of interest. There will be slight fluctuations when switching between the two flight planners. In a single flight, the basic information about the study area can be obtained at a higher flight altitude, and detailed information about the object of interest can be obtained at a lower flight altitude.

However, the flight path of the local planner is relatively simple. Although the circular flight path is simple and easy to implement, the information obtained will be target-oriented if a more complex path planning method is used. However, the complex path also means a longer computing time. Besides, this study did not consider the situation of obstacle avoidance and occlusion. In real-world applications, more complex situations should be taken into consideration.

The image size (752x480) used in this study is relatively small compared to a real sensor on commercial UAVs. For instance, the camera mounted on phantom 4 pro has an image size of 5472x3468 according to the support document of Pix4D [69]. The main reason is that processing larger images will take more time and memory, which may lead to the crashing down of MATLAB.

Based on the camera setting from table 4.1, the field of view calculated in the horizontal direction is about 37.5 degrees. With a relatively small field of view, the coverage of the ground in low flight height became restricted. Hence, when the UAV flew the circular flight path, the vehicle might not appear in the range that the camera could capture, and this results in an incomplete vehicle in the image (figure 6.1).



(a) (b)

Figure 5.14 Incomplete vehicle inspection

The radius of the circular flight path was already set in a small value, which is three meters, compared to the size of the simulation scene. Hence, the possible reason for this incomplete inspection is due to the small field of view at low flight height. Nonetheless, the inspection results in section 5.2.2 indicate that complete vehicles can still be captured. Therefore, within the local planner, a larger field of view is needed to cover a larger area. In other words, the focal length should be smaller when flying at a low flight height in the local planner.

6. CONCLUSION AND RECOMMENDATIONS

6.1. Conclusion

In this study, an algorithm that allows the UAV to update the flight path automatically according to the target of interest (vehicles) was developed and implemented in the Simulink platform. The whole algorithm was achieved in a simulation environment of city blocks created by MATLAB using Unreal Engine. The flying process was divided into two main parts, the traditional strip flight path called global planner and the circular flight called local planner. The link between these two planners is the object detection process. The global planner was defined based on the simulation scene. The local planner is a circle, which center is the location of the vehicle. The object detection was achieved using YOLO v3 with a Squeeze net backbone and two detection heads in different scales. The YOLO v3 object detector was first trained on the Aeroscape dataset, which contains images from the real world, and then perform a transfer learning on the simulation dataset, which was created manually. According to the results, here are the conclusions:

i. The object detector using YOLO v3 achieved an average precision of 90% on the Aeroscape dataset and increased to 100% after performing the transfer learning process on the synthetic dataset. When using this object detector in the simulation environment, the detector worked well at 50 meters height with a 52.9% accuracy but performed poorly at lower flight height, only achieved 44.4% accuracy. Adding rotation and scaling in the data augmentation process can improve the network performance at lower flight height, which the accuracy increased to 90.8%. However, the accuracy at 50 meters height dropped to 28%. These detection results are mainly affected by the size of the training dataset. The number of images used in this study is not sufficient for a decent training process but acceptable for use in a simulation environment.

ii. The circular flight path can be sufficient for a detailed data acquisition process. The image acquisition process was obtained from the video stream with a sampling frequency of one second. On average, around 673 images were acquired through this circular flight path, and about 20 thousand images were acquired in the entire flight mission. The vehicle can be captured from different directions and scales. However, the side face of the vehicle may not be so clear. More complicated path planning methods like next-best-view can be implemented in the future based on this simple flight path.

iii. Simulink has been proved as a proper platform for UAV applications, and the entire flight was stable. The UAV toolbox provided many functional blocks for flight planning and scenario simulation. Moreover, the CNN-based object detection methods can be developed in MATLAB first and imported directly into the Simulink. However, this model was defined for multi-rotor UAVs. Fixed-wing UAV is not so flexible to achieve turns. Besides, this model cannot distinguish multiple targets, and the speed of the model was relatively slow.

In conclusion, the flight path planning algorithm worked smoothly in Simulink. The Simulink platform with the UAV toolbox provided useful blocks for the simulation of UAV applications. As for object detection, more training samples are needed for a better performance of the object detector. With sufficient training samples, data augmentation can improve the detection result. As for the local planner, a more complex path planning algorithm can be considered in further research.

6.2. Recommendations

This study was the first attempt at using the UAV toolbox to design and test the UAV flight path planning algorithm all in one go. It proved the possibility of developing UAV applications in the Simulink environment. Therefore, based on this prototype, a more complicated model can be developed to achieve different objectives.

The Simulink platform has the advantage of using a graphical programming interface and built-in functions from MATLAB, which is more intuitive and convenient than starting everything from scratch. Moreover, Simulink supports code generation in C, C++, Verilog, VHDL, and Structured Text. This means once the Simulink has been developed successfully, the model can be easily tested on hardware. This significantly shorts the distance between simulation and reality.

As for object detection, if the training dataset is sufficient, then the object detection can be improved by using advanced object detection methods. Besides, MATLAB supports importing networks and network architectures from TensorFlow-Keras, Caffe, and the ONNX (Open Neural Network Exchange) model format [70]. However, this importing process needs to consider the version compatibility issue. The newest version available in python may not be supported by MATLAB yet.

As for the simulation environment, the model developed in this study only considered an ideal urban environment. The wind speed, illumination condition, and many other environmental factors were not included. Therefore, a more complicated simulation environment can be established. Besides, the distribution of the target was also in regular distribution. There was only one vehicle at every intersection. However, in the real world, there will be a high possibility that several targets of interest may appear at the same time. How to distinguish them and perform a sequential inspection one by one can be the next step.

As for the local planner, the proposed local planner was a circular path. Based on this, a more advanced algorithm can be developed to achieve an exploration. For example, introduce the Simultaneous Localization and Mapping (SLAM) algorithm or Next-Best-View (NBV) method in the local planner part could be an option. Besides, the obstacle avoidance problem was not considered. To prevent the UAV crash on the walls or anything else, using the LiDAR sensor may be an ideal option.

Besides, this algorithm focused on small objects like vehicles. For other kinds of objects, the detection method needs to be modified. For instance, if focusing on a large object like buildings, semantic segmentation may be a better option than object detection methods. Another thing is that the limitation of flight time was not considered in this study. The conventional flight time for a multi-rotor UAV is about 30 minutes. If the study area becomes large, then a single flight will not be enough. Hence, for a large study area, multiple UAVs can be used simultaneously. Moreover, the local planner should be executed sequentially for multiple objects according to a series of bounding boxes.

Nevertheless, this model is the first step to realize more UAV-related applications in Simulink and provides a new possibility for designing and testing more algorithms. Also, it provides a safer and faster simulation environment before the field test and reduces the risk of sensor damage that may happen in the field test.

LIST OF REFERENCES

[1] X. Zhou, Z. Yi, Y. Liu, K. Huang, and H. Huang, "Survey on path and view planning for UAVs," *Virtual Real. Intell. Hardw.*, vol. 2, no. 1, pp. 56–69, Feb. 2020.

[2] B. Alzahrani, O. S. Oubbati, A. Barnawi, M. Atiquzzaman, and D. Alghazzawi, "UAV assistance paradigm: State-of-the-art in applications and challenges," *J. Netw. Comput. Appl.*, vol. 166, p. 102706, Sep. 2020.

[3] DJI, "Phantom - Semi Professional Camera Drones - DJI," 2021. [Online]. Available: https://www.dji.com/nl/products/phantom?site=brandsite&from=nav. [Accessed: 13-Jun-2021].

[4] Pix4D, "PIX4Dmapper: Professional photogrammetry software for drone mapping | Pix4D," 2021. [Online]. Available: https://www.pix4d.com/product/pix4dmapper-photogrammetry-software. [Accessed: 09-Jun-2021].

[5] DJI, "DJI Terra," 2021. [Online]. Available: https://www.dji.com/nl/dji-terra. [Accessed: 09-Jun-2021].

[6] K. S. Lee, M. Ovinis, T. Nagarajan, R. Seulin, and O. Morel, "Autonomous patrol and surveillance system using unmanned aerial vehicles," in *2015 IEEE 15th International Conference on Environment and Electrical Engineering, EEEIC 2015 - Conference Proceedings*, 2015, pp. 1291–1297.

[7] S. Karim, C. Heinz, and S. Dunn, "Agent-based mission management for a UAV," in *Proceedings of the 2004 Intelligent Sensors, Sensor Networks and Information Processing Conference, ISSNIP '04*, 2004, pp. 481–486.

[8] S. Aggarwal and N. Kumar, "Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges," *Computer Communications*, vol. 149. Elsevier B.V., pp. 270–299, 01-Jan-2020.

[9] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, 1968.

[11] W. Khaksar, S. Vivekananthen, K. S. M. Saharia, M. Yousefi, and F. B. Ismail, "A review on mobile robots motion path planning in unknown environments," 2016, pp. 295–300.

[12] J. Tordesillas, B. T. Lopez, J. Carter, J. Ware, and J. P. How, "Real-time planning with multi-fidelity models for agile flights in unknown environments," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2019-May, pp. 725–731, 2019.

[13] A. Bircher *et al.*, "Structural inspection path planning via iterative viewpoint resampling with application to aerial robotics," *Proc. - IEEE Int. Conf. Robot. Autom.*, vol. 2015-June, no. June, pp. 6423–6430, 2015.

[14] M. Ramasamy and D. Ghose, "Learning-based preferential surveillance algorithm for persistent surveillance by unmanned aerial vehicles," *2016 Int. Conf. Unmanned Aircr. Syst. ICUAS 2016*, pp. 1032–1040, 2016.

[15] C. Stöcker, R. Bennett, F. Nex, M. Gerke, and J. Zevenbergen, "Review of the current state of UAV regulations," *Remote Sensing*, vol. 9, no. 5. MDPI AG, p. 459, 01-May-2017.

[16] M. A. Boon, A. P. Drijfhout, and S. Tesfamichael, "Comparison of a fixed-wing and multi-rotor UAV for environmental mapping applications: A case study," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 2017, vol. 42, no. 2W6, pp. 47–54.

[17] S. Tilon, F. Nex, and N. Kerle, "D5.1: Conceptual framework for remote sensing based RI monitoring," 2019.

[18] C. Johnston, "Technical Challenges For Small UAV Payloads," *Electron. Mil. Def.*, pp. 38–44, 2012.

[19] J. G. Mooney and E. N. Johnson, "A Comparison of Automatic Nap-of-the-earth Guidance Strategies for Helicopters," *J. F. Robot.*, vol. 33, no. 1, pp. 1–17, 2014.

[20] DJI, "Phantom 4 Pro," 2021. [Online]. Available: https://www.dji.com/nl/phantom-4-pro?from=p4p-or-p4a. [Accessed: 13-Jun-2021].

[21] senseFly, "eBee X Fixed-Wing Mapping and Surveying Drone." [Online]. Available: https://www.sensefly.com/drone/ebee-x-fixed-wing-drone/. [Accessed: 13-Jun-2021].

[22] Vertical Technologies, "DeltaQuad - VTOL Mapping Drone / UAV for large area mapping." [Online]. Available: https://www.deltaquad.com/vtol-drones/map/. [Accessed: 13-Jun-2021].

[23] C. Peng and V. Isler, "Adaptive view planning for aerial 3D reconstruction," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2019, vol. 2019-May, pp. 2981–2987.

[24] Pix4D, "PIX4Dcapture: Free drone flight planning mobile app | Pix4D." [Online]. Available: https://www.pix4d.com/product/pix4dcapture. [Accessed: 15-Jun-2021].

[25] "DJI - Official Website." [Online]. Available: https://www.dji.com/nl. [Accessed: 15-Jun-2021].

[26] "Parrot | European leader in professional drones." [Online]. Available: https://www.parrot.com/en. [Accessed: 15-Jun-2021].

[27] "Yuneec – Quadcopters & Aerial Drones." [Online]. Available: https://us.yuneec.com/. [Accessed: 15-Jun-2021].

[28] Pix4D, "Types of mission / Which type of mission to choose – Support." [Online]. Available: https://support.pix4d.com/hc/en-us/articles/209960726-Types-of-mission-Which-type-of-mission-to-choose. [Accessed: 15-Jun-2021].

[29] C. I. Connolly, "The determination of next best views," in *Proceedings - IEEE International Conference on Robotics and Automation*, 1985, pp. 432–435.

[30] M. Karaszewski, M. Adamczyk, and R. Sitnik, "Assessment of next-best-view algorithms performance with various 3D scanners and manipulator," *ISPRS J. Photogramm. Remote Sens.*, vol. 119, no. July 2018, pp. 320–333, 2016.

[31] R. Huang, D. Zou, R. Vaughan, and P. Tan, "Active Image-Based Modeling with a Toy Drone," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 6124–6131, 2018.

[32] R. Almadhoun, A. Abduldayem, T. Taha, L. Seneviratne, and Y. Zweiri, "Guided next best view for 3D reconstruction of large complex structures," *Remote Sens.*, vol. 11, no. 20, pp. 1–20, Oct. 2019.

[33] S. A. Sadat, J. Wawerla, and R. T. Vaughan, "Recursive non-uniform coverage of unknown terrains for UAVs," *IEEE Int. Conf. Intell. Robot. Syst.*, no. Iros, pp. 1742–1747, 2014.

[34] M. Mendoza, J. I. Vasquez-Gomez, H. Taud, L. E. Sucar, and C. Reta, "Supervised learning of the next-best-view for 3d object reconstruction," *Pattern Recognit. Lett.*, vol. 133, pp. 224–231, May 2020.

[35] H. Zhang, M. Sun, Q. Li, L. Liu, M. Liu, and Y. Ji, "Multi-scale Object Detection in High Resolution UAV Images: An Empirical Study," *Neurocomputing*, vol. 421, pp. 173–182, Sep. 2020.

[36] A. R. Pathak, M. Pandey, and S. Rautaray, "Deep learning approaches for detecting objects from images: A review," in *Advances in Intelligent Systems and Computing*, 2018, vol. 710, pp. 491–499.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 779–788.

[38] D. Cazzato, C. Cimarelli, J. L. Sanchez-Lopez, H. Voos, and M. Leo, "A survey of computer vision methods for 2d object detection from unmanned aerial vehicles," *Journal of Imaging*, vol. 6, no. 8. MDPI AG, p. 78, 04-Aug-2020.

[39] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017.

[40] A. Bochkovskiy, C.-Y. Y. Wang, and H.-Y. Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv*, Apr. 2020.

[41] X. Guo, X. Li, Q. Pan, P. Yue, and J. Wang, "An Object Detection Algorithm for UAV Reconnaissance Image Based on Deep Convolution Network," in *Lecture Notes in Electrical Engineering*, 2019, vol. 606, pp. 53–64.

[42] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-Janua, pp. 6517–6525.

[43] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv*, Apr. 2018.

[44] A. Kathuria, "What's new in YOLO v3?. A review of the YOLO v3 object… | by Ayoosh Kathuria | Towards Data Science," *Medium*, 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b. [Accessed: 28-May-2021].

[45] Epic Games, "The most powerful real-time 3D creation platform - Unreal Engine," *Unreal Engine 4*, 2021. [Online]. Available: https://www.unrealengine.com/en-US/. [Accessed: 02-Jun-2021].

[46] Mathworks, "Customize Unreal Engine Scenes for UAVs," *MathWorks, Inc.*, 2020. [Online]. Available: https://nl.mathworks.com/help/uav/ug/customize-3d-scenes-for-automated-driving.html. [Accessed: 02-Jun-2021].

[47] Mathworks, "Coordinate Systems for Unreal Engine Simulation in UAV Toolbox," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/uav/ug/coordinate-systems-for-unreal-engine-simulation-in-uav-toolbox.html. [Accessed: 02-Jun-2021].

[48] Mathworks, "Camera sensor model with lens in 3D simulation environment," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/uav/ref/simulation3dcamera.html?s_tid=doc_ta. [Accessed: 05-Jun-2021].

[49] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.

[50] J. Y. Bouguet, "Camera Calibration Toolbox for Matlab," 2015. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc. [Accessed: 05-Jun-2021].

[51] H. Janne and S. Olli, "A Four-step Camera Calibration Procedure with Implicit Image Correction," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1997, pp. 1106–1112.

[52] Mathworks, "Select Waypoints for Unreal Engine Simulation," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/driving/ug/select-waypoints-for-3d-simulation.html. [Accessed: 03-Jun-2021].

[53] MathWorks, "Anchor Boxes for Object Detection," *MathWorks, Inc.*, 2019. [Online]. Available: https://nl.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html. [Accessed: 01-Jun-2021].

[54] MathWorks, "Pretrained Deep Neural Networks," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html. [Accessed: 15-Jun-2021].

[55] Mathworks, "Object Detection Using YOLO v3 Deep Learning," *MathWorks, Inc.*, 2020. [Online]. Available: https://nl.mathworks.com/help/releases/R2020b/vision/ug/object-detection-using-yolo-v3-deep-

learning.html#mw_rtc_ObjectDetectionUsingYOLOV3DeepLearningExample_D2F0B11C. [Accessed: 30-May-2021].

[56] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 2016.

[57] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2010, pp. 248–255.

[58] I. Nigam, C. Huang, and D. Ramanan, "Ensemble Knowledge Transfer for Semantic Segmentation," in *Proceedings - 2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018*, 2018, vol. 2018-Janua, pp. 1499–1508.

[59] "GitHub - cuixing158/imageLabeler-API: Convenient image annotation tool API." [Online]. Available: https://github.com/cuixing158/imageLabeler-API. [Accessed: 26-Apr-2021].

[60] MathWorks, "Estimate Anchor Boxes From Training Data," *MathWorks, Inc.*, 2020. [Online]. Available: https://nl.mathworks.com/help/vision/ref/estimateanchorboxes.html?searchHighlight=estimateAnchorBoxes&s_tid=srchtitle. [Accessed: 31-May-2021].

[61] Mathworks, "UAV Package Delivery," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/uav/ug/uav-package-delivery.html. [Accessed: 03-Jun-2021].

[62] Mathworks, "Compute and execute a UAV autonomous mission," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/uav/ref/pathmanager.html. [Accessed: 04-Jun-2021].

[63] MathWorks, "Output input from previous time step," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/simulink/slref/memory.html. [Accessed: 06-Jun-2021].

[64] MathWorks, "Algebraic Loop Concepts," *MathWorks, Inc.*, 2021. [Online]. Available: https://nl.mathworks.com/help/simulink/ug/algebraic-loops.html. [Accessed: 06-Jun-2021].

[65] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8693 LNCS, no. PART 5, pp. 740–755.

[66] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep Learning Face Attributes in the Wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[67] Y. Lyu, G. Vosselman, G.-S. S. Xia, A. Yilmaz, and M. Y. Yang, "UAVid: A semantic segmentation dataset for UAV imagery," *ISPRS J. Photogramm. Remote Sens.*, vol. 165, no. 0934–2716, pp. 108–119, Jul. 2020.

[68] H. Yu *et al.*, "The Unmanned Aerial Vehicle Benchmark: Object Detection, Tracking and Baseline," *Int. J. Comput. Vis.*, vol. 128, no. 5, pp. 1141–1159, Mar. 2020.

[69] Pix4D, "Which cameras are supported in Pix4Dmatic – Support," 2020. [Online]. Available: https://support.pix4d.com/hc/en-us/articles/360037744571-Which-cameras-are-supported-in-Pix4Dmatic. [Accessed: 07-Jun-2021].