Zewail City for Science and Technology
University of Science and Technology
Aerospace Engineering

Object Detection and Control of Autonomous Racing Drone

A Graduation Project
Submitted in Partial Fulfillment of
B.Sc. Degree Requirements in
Aerospace Engineering

Prepared By

Aya Abdallah Atia	201701090
Mohssen Elshaar	201700493
Mohamed Kamel	201700402
Ezat Eldohimi	201700380
Rana Shalaby	201700634

Supervised By
Dr. Mohamed Elshenawy
Signature

2021/2022

بسم الله الرحمن الرحيم

﴿يَٰٓأَيُّهَا ٱلَّذِينَ ءَامَنُوٓاْ إِذَا قِيلَ لَكُمۡ تَفَسَّحُواْ فِي ٱلۡمَجَٰلِسِ فَٱفۡسَحُواْ يَفۡسَحِ ٱللَّهُ لَكُمۡۖ وَإِذَا قِيلَ ٱنشُزُواْ فَٱنشُزُواْ يَرۡفَعِ ٱللَّهُ ٱلَّذِينَ ءَامَنُواْ مِنكُمۡ وَٱلَّذِينَ أُوتُواْ ٱلۡعِلۡمَ دَرَجَٰتٖۚ وَٱللَّهُ بِمَا تَعۡمَلُونَ خَبِيرٞ١١﴾

"سورةُ المجادِلة، الآية الحادية عشر"

الحمد لله رب العالمين، له الحمد الحسن، والثناء الجميل، ونشهد ألا إله إلا الله وحده لا شريك له، ونشهد أن محمدا عبده ورسوله .. أما بعد،

مضت خمس سنوات ننهل فيهن العلم ونراكم الخبرات، وبذلنا فيهن جهودا بعد الجهود، وخضنا تجارب شتى؛ فكنا نصيب مرة ونخيب مرات، .. وإنا قد اجتهدنا أن نخرج هذا العمل على نحو يليق بما تعلمناه، ونزعنا عصارة من تطورنا على مجمل هذا العمل وتفصيله، وحرصنا على أن نتلافى الخيبات قدر الإمكان، فكان التوفيق من الله بنجاحات متتالية ما كانت في الحسبان .. ألا إن لكل مجتهد نصيب!

وإنا لا يسعنا إلا أن نطلب المزيد من المعرفة، ورحلتنا مع العلم مستدامة إلى حين آخر مجهول! ولا يسعنا كذلك إلا أن نستحضر معية الله عز وجل فيما هو آت، كما رأينا آثارها طيبة حانية فيما قد مضى ..

فاللهم إنا نسألك علما نافعا، ورزقا طيبا، وعملا متقبلا ..

Abstract

Recently, the number of applications that require aerial oversight and accurate navigation has increased; from modern agricultural and aerial surveillance, delivery services, and rescue to military operations. Hence, there is a tendency to automate these processes using autonomous drones is highly needed. An Unmanned Aerial Vehicle (UAV) is a platform that can perform in-air missions without human control. In some applications, perception and classical control methods fail and need improvements to be used in the operations done in complex environments like forests. Modern methods like deep supervised learning require huge datasets which are very expensive, and sometimes dangerous, to obtain. In this project, we try to develop an autonomous quadrotor that can detect and traverse through waypoints without collision at a high speed in space using AirSim. In addition to improving the results from previous work. RGB camera is used for collecting information about the environment and a cross-modal VAE (CM-VAE) utilizes them for perception. Imitation learning is applied on UAV to learn the control policy. We deployed the algorithms using AirSim on different tracks and gates and achieved a better performance (higher velocity on more complex tracks) than the most recent recorded results in the literature. Our Model A performed better in terms of mean velocity (150%), overall stability (1/5 completed tracks in literature compared to 5/5 completed tracks by our Model A), and detection range. Even-though the range has improved, it is still limited and has a room for improvements. Model B had a higher average mean velocity (121%) than Model A, but at the cost of an uncontrolled oscillatory behaviour which greatly reduces its stability. Both models had an issue with collisions where they would not respond to wall collision and more often than not get stuck upon any collision.

Declaration

We declare that:

- The thesis has been composed by the students whose names are shown on the cover page
- The thesis represents our own work
- The work has not been submitted for any other degree or professional qualification except as specified.

Acknowledgements

This project would not have been possible without the support of many people. Many thanks to our advisor, Dr. Mohamed Elshenawy, who read our numerous revisions, guide us through the project, and helped in clarification for many concepts.

Thank you to our professors who enriched us with knowledge through the five years of the academic learning to reach this stage.

Thank you to our families who were the main reason for us to be here after Allah and providing us with the encouragement to keep going.

And finally, thanks to our friends who stood by us during this long journey, always offering support and love.

Personal Acknowledgements

Thank you ***Father*** for your continuous guidance and useful advisement. Thank you ***Mother*** for your priceless efforts and support. I know that I would have never succeeded without Allah's care through you, and I hope that I become the person you have wished me to be one day. Thanks ***Abo-Salah***, you were always there during my hard times. Thanks ***Hema*** for your tolerance and thoughtfulness. Thanks ***Abka*** and ***Khaled***, you made the trip easier. Thanks to those who stayed, and thanks to those who left, .. Alhamdullillah.

Mohssen Elshaar

Personally, i would love to give special thanks to those who accompanied me during my non-academic journey:

"Mother, Father, Siblings, Mohammed Ehab, Yusif Ghonim, Badr Adel, Mohammed Sherif, Mohammed Ashraf, Mouris Gamil, Marwan Mostafa, Belal Bishr, Mostafa Ashraf, Omnia Badie."

And the ones in my academic journey:

"Mohammed Fouad, Amir Saif, Hatem Yehia, Mohammed Zayed, Ahmed Lila, Youssef Mostafa, Norhan Mohammed, Mai El-Somkhraty, Ahmed Hamdy, Mohammed El-Shafey, Ahmed Yasser, Ahmed Gamal, Mohammed Tarek, Zeinab Mohammed, Ramy Temraz, Ashrakat Ahmed."

Each and every one of you has impacted my life in a different way, I would not have made it without your continuous support.

Ezzat Eldokhimi.

Thank you to my family who always believed in me and supported me

"My father, My mother, My sisters Esraa and Manar, and My brother Mohamed."

A special thanks to the most supportive partner and my love Ahmed Hassan who always encouraged me to do my best and believed in my abilities.

To my friends:

"Rana Saeed, Fatema Moanes, Touka Mohamed, Bassant Yasser, Alaa Gamal, Nouran El-Nakeeb, Mayar Atef, Mai El-yamani, Hagar El mashriqi, Elaf Badr, Anhar El shemy, Tsneem omara."

Aya Abdallah.

I would like to thank My family who supported me along my journey.

'My father, My mother and My sisters yara and haya'

I would also like to thank my beautiful friends.

"Touka Mohamed, Fatma Moanes, Nourhan Mohsen, Alaa Gamal, Amal Hani, Mai El-yamani, Nouran El-Nakeeb, Rana El-Semary"

Rana Galaat Shalaby.

Contents

List of Figures

List of Tables

List Of Abbreviations

Chapter 1

Introduction

1.1 Motivation

An unmanned Aerial Vehicle UAV is a platform that performs in-air missions without human control. Recently, the number of applications that require UAVs has increased in both civilian and military fields. Across many civil application domains, the use of UAVs has been growing rapidly. Drones nowadays are used in real-time monitoring, providing wireless coverage, remote sensing, search and rescue, delivery of goods, security and surveillance, precision agriculture, and civil infrastructure inspection [1]. In addition to that, 'Meaning-Less Human Control', one of the the latest war studies in 2021 noted that human control over UAVs is unneeded [2]. In their report, some challenges related to in-air human control were suggested. For example, high speed operations are too fast for human response, and Complex tasks with unbearable maneuvers cannot be handled by human pilots. As a result of this deficiency in human control, automated systems integration is increasing widely in the air industry, and human interference and role continues to change from active controllers to more passive supervisors [2]. Tasks required for UAV autonomy can include Sensing (Perception), Path planning, obstacle avoidance, navigation, and attitude control. In this work we are focusing on obstacle detection, collision avoidance and path planning.
Classical active autopilot methods usually work by reading the aircraft's properties (speed, position, orientation, ..etc) and computing actuators' output before correcting for errors in the next cycle and keeps improving the error with each cycle that passes accounting for static and dynamic errors [3]. While this method might be reliable for a real aircraft with satellite connection, it may be unsuitable when the information from Global Positioning System (GPS) or satellite connections of precise position is not provided. This work is related to "The Game of Drones" competition in which the information is not always available. Instead, path information must be perceived using a visual monocular camera which provides relative positions instead of absolute ones [4].

1.1.1 Economic and global context

At this time, UAV applications are widely used in different fields such as agricultural and forests surveillance as it can oversight plant growth and detects forest's fires, rescue operations as it is dangerous for humans to perform, and delivery services which saves

cost and time to deliver products rather than the traditional methods for delivery that can stuck in the traffic. Our product has a wide range of customers all over the world due to its various applications. Moreover the target customers for our product are the owners for any business that has products to be delivered, the government and the military department for surveillance operations, and agriculture owners. Having this product will help in performing dangerous operations that no human can do, accurate surveillance with accurate sensors, and saving cost and time for delivery.

1.1.2 Societal and environmental context

This product affects the society and the environment positively sometimes as it helps in decreasing the traffic due to lots of delivery and its role of saving people's lives. On the other hand, Privacy, acceptance, and security are increasingly being replaced by operational considerations such as interaction with and impacts on other airspace users. Recent incidents demonstrate that unrestricted drone use can cause problems for other airspace users [5]. So there is a need for management response to organize the rapid and efficient development in drone usage even while facilitating innovation [5].

1.1.3 Cultural

The commercial and private utilization of drones is changing many ecosystems. It has been used for military purposes in many countries and it is widely increasing in non-military roles with different considerations [5]. As well, the culture's response to our product depends on how this culture is aware or responds toward innovative and different ideas and its conditions are well prepared.

1.2 Overview of the key challenges

Solving this problem is challenging as it requires a management response to avoid problems. First, the high dimensional nature and drastic variability of the input image data while navigating. Also, training in the real world is dangerous and expensive and real data is difficult to collect, so the challenge is to train the model using only simulated data.

1.3 Project scope

The problem we are solving is divided into two parts; first, developing perception models for gates and obstacles detection so that the **UAV!** (**UAV!**) can pass through them safely. There are different sensors that are used for perception like radar and LIDAR but the RGB camera is used for data in this project due to its ability to provide richer information about the environment. Second, path planning for the drone to fly through all gates smoothly with the minimum time. There are many methods used for path planning and control for UAV from classical control methods to modern methods using artificial intelligence that proved its efficiency. That's why imitation learning is used in the path planning and control of the UAV as it gives better performance in unknown environments.

1.4 Proposed methodology

Generally, our methodology is training on collecting data until we get a good performance then deploy the results on the simulated environment on AirSim. Data collection is done by RGB camera using AirSim to get 300k images each labeled with gate pose. Consequently the CM-VAE is used for encoding the images into latent vectors. Training on big data using neural networks results in a good performance for pose extraction. Furthermore, the latent vectors are utilized for the control model that is an imitation learning model to get the control best action and actuators speed after the training while imitating the expert.

1.5 Summary of the implementation

The project can be broken down into two main sub-units: the perception and control units.

The perception unit is a variational autoencoder (VAE) whose latent space is jointly trained by one encoder, Image Encoder (q_{img}), and two decoders. The encoder adopts the DroNet [citation] architecture, the first decoder is the image decoder Image Decoder (p_{img}) and the second is the gate decoder Image encoder (p_{gate}). The entire system is

trained using two input modalities, input images and ground truth poses of the gates in the images, using two modes of training at the same time, supervised learning for gate reconstruction and unsupervised learning for image reconstruction. Thus, the overall compound loss is composed of three terms, image reconstruction loss, gate reconstruction loss, and K-L loss. After the training phase, we only use the encoder q_{img} to produce the latent vector z which will be used in the training of the control policy, the second unit. The control unit is a Behavioural Cloning (BC) model in which the latent space vector extracted by the VAE is used to predict the suitable velocity vector. The model is a neural network that is trained using an expert's dataset. The expert is a drone that plans and follows a minimum snap trajectory based on the provided gate poses. The expert puts one future gate into consideration while solving the planning problem, and after tracking down the planned trajectory, it plans a new trajectory to the gate that follows. During its flight, the expert's dataset is generated. The dataset is mainly of 2 components, images of what the drone perceives, and the corresponding velocity vectors at the exact moments of these perceptions. After the network is trained, the full model (Perception and Control unit) is ready to be deployed and subjected to different tests.

1.6 Report Organization

The next chapter includes the literature review about the evolution of using perception and control methods for UAV. Moreover, a market survey, regulations, and laws. Consequently, a proposed design for our work and alternative designs, details of the project implementation, discussion and project impact, and lastly the conclusion and future work.

Chapter 2

Literature Review

2.1 Sensing

UAV state estimation can be done using two types of sensors, exteroceptive and proprioceptive. Proprioceptive sensors are relating to stimuli produced internally from the system, typically position and acceleration sensing. Exteroceptive sensors are relating to external stimuli, for example, visual stimuli like camera inputs. The most popular exteroceptive sensors used in UAVs are GPS and cameras. Gyroscopes and accelerometers are examples of propriceptive. Each sensor is suitable for some kind of mission, however, there are limitations that can severely affect the performance of every type of sensor. For example, GPS performance is prone to the number of available satellites. Accelerometers and inertial navigation systems' error propagation can affect the performance of the UAV by providing erroneous estimations for the velocity and position. Ideally, we can fuse information from different sensors to get better state estimation. However, for small UAVs, it is impractical to install several sensors onboard [6].

2.1.1 Obstacle detection

Obstacle detection is an essential module in UAV autonomy as it reduces the damages resulting from collision and pilot error. Obstacle avoidance can be done using several types of sensors: Ultrasonic, Infrared Sensor (IR), Radar, etc. Nevertheless, all of them have very limited range and do not provide much information as visual obstacle avoidance. In this project, we focus on visual sensing as a mean to enable collision avoidance and path planning. Other sensors such as accelerometer can be used in other system modules. Visual sensing provides more information about the environment, like color and texture, more than other types of sensors. Common visual sensors (Figure 2.1) are:

- Monocular Camera: which is suitable for applications where there is limitation on the size or the weight of the drone. Its compactness and low price makes it very common, however, it cannot provide depth information.
- Stereo Camera: which is basically a pair of monocular cameras, it can provide information about the depth based on the two views obtained from each camera.
- RGB-D Camera: which can provide visual information about the environment as well as a depth map using an infrared sensor. However, it is typically used in indoor

applications due to its limited range.

- Fish-eye Camera: is a monocular camera with a wider angle view, it is suitable for obstacle avoidance applications.

Figure 2.1: Common Visual Sensors

2.1.2 Visual Obstacle avoidance

There are two main methods for visual obstacle avoidance: optical flow based or map-less navigation methods and Simultaneous Localization And Mapping Simultaneous Localization And Mapping (SLAM) based methods or map based navigation [7]. SLAM methods generally work by simultaneously updating a map while keeping track of the agent in it. Optical flow methods are very similar to how the human eye works, however, it provides limited accuracy in terms of distances of the obstacles which limits its usage in some applications that require precise state estimation. SLAM methods, on the other hand, can acquire precise maps of the environment at the cost of algorithmic complexity [8]. All-Kaff developed an optical flow method for object detection inspired by bio-mimicry

where they utilized the change of the received image's size with the distance to estimate how far away the object is [9]. With their built method, they were able to achieve an accuracy of 92.5% of obstacle avoidance, however, the range of the detection is quite short, hardly 90-120 cm. Moreover, their algorithm suffered with obstacles approaching from directions other than the forward direction. Rogerio Bonatti used SLAM based AI models for both the detection and the control of the UAV which impacted the detection significantly as the algorithm struggles with detecting objects with motion not towards the sensor [10]. They developed a Deep Neural Network (DNN) to be trained on simulation data to estimate the coordinates and the orientation of an obstacle in space. They were able to to achieve obstacle detection in ranges of 2-10m, which is great compared to [11] with the cost of lower accuracy.
One variant of the SLAM techniques is the ORB-SLAM, which uses a monocular camera as the main sensor for gathering data then pinpoints features in one key-frame to track them in the next key-frame. The main advantage for this technique is having its solutions being near real time, but one disadvantage is its high dependency on features, a plain wall would not be tracked and therefore not be registered as an obstacle, another one is its inability to accurately detect the boundary of the detected obstacles [12].
Sunggoo Jung proposed multiple modifications to a deep convolutional neural network for multi-box object detection called Single Shot Detector (SSD). The original model offered 2 FPS worth of performance which was insufficient, and hence VGG-16 base model was replaced with AlexNet, also fc6 and fc7 were both replaced with a convolutional layer with three connected AlexNet layers. Fc8 was deleted. All led to an increase in the performance to 12 FPS then further increased to 30 FPS by deleting unnecessary layers at the cost of reduced precision (0.07 compared to the original model) which was mitigated by tuning the network parameters [13].

2.2 Path planning

Path planning includes both, finding a geometrical path from the initial state to the target, and avoiding obstacles and threats along the path. Depending on whether a priori information of the environment is known or not, path planning is divided into global and local path planning. Global path planning; in which all the required information is known

to the drone before starting to move but it is limited to static environments. On the other hand, local path planning can handle both static and dynamic environments, where the drone observes only a part of the unknown environment each step through sensors, and thus, the environment is detected during the movement part by part [14].

2.2.1 Global Path Planning

Since all the required information is known to the drone before starting, this type of planning is limited to static environments. There are many global path planning algorithms to solve the problems of path search such as Dijkstra's algorithm, A* algorithm, and D* algorithm [9]. Most of these algorithms minimize a cost function $f(n)$ for each state n. The function is usually a combination of $g(n)$: the current accumulated cost between the initial state and state n, and $h(n)$: a heuristic estimate of the minimum cost from state (n) to the goal state.

Dijkstra's algorithm is An efficient breadth-first search method to find a least-cost path between nodes in a graph. The cost function is $f(n) = g(n) + 0$, which uses a negligible estimate of the estimated distance to the goal. The main disadvantage of this method is that it does not use the information obtained from the environment (uninformed search).

In A* algorithm the cost function is $f(n) = g(n)+h(n)$. It is more effective than Dijkstra's algorithm because it considers the location of the goal and mainly explores the goal state (informed search).

D* algorithm is a dynamic version of A*, where the arc cost is changed while the operation (online re-planning). The backward path (from goal to current position) is searched. It is more effective than A* in complex environments. However, the high cost of memory is the main disadvantage [14].

2.2.2 Reactive(local) path planning

Due to the lack of information about the environment, the problem of free navigation without a prior knowledge about the environment is a bit of challenging. In contrast to global path planning, a local/reactive navigation method works well in dynamic and initially unknown environment. The following paragraphs represents some algorithms related to local path planning [14].

Particle Swarm Optimization

Particle Swarm Optimization (PSO) algorithm is a stochastic optimization technique works by a swarm of particles that move around in the search space based on the intelligent of swarms. In UAV path planning problems, particle movements are achieved by their own best known position in the search space and the swarm best known position. The improved particles' positions are calculated considering an evaluation function then update the position if it is better than the previous one and the process is repeated until a good solution for the path is discovered. This method is robust and fast in known threat environments, however, it can fall in a local optima but combining it with the genetic algorithm solves this problem [15].

Ant Colony Optimization

The main concept of Ant Colony Optimization (ACO) is the walking path of ants that is used to express the feasible solution of the problem. In a path planning problem, all the paths of the whole ant group constitute the solution space for the optimization problem. The concentration of pheromone accumulated on shorter paths increases, and the number of ants choosing the path increases. Eventually, the whole ant will concentrate on the best path and the corresponding solution is the optimal solution to the path planning optimization problem [16].

Trajectory Optimization

Trajectory Optimization uses two alternative methods for trajectory generation: Mixed Integer Linear Programming (MILP) and a modification of the A* algorithm [17]. The UAVs trajectory generation is faced as a 3D optimization problem under certain conditions in the Euclidean space, characterized by a set of decision variables, a set of constraints and the objective function. The A* algorithm for UAV's trajectory generation it has the advantage to present a good performance when the terrain is quite irregular and the manoeuvres must be done close to the ground's surface. The main drawback is its high computational cost in 3D path planning problems [17].

Spline based path planning

In 2001, McLain and Judd proposed a relatively efficient method for path planning [18]. The method can deal with pop-up threats as well as static obstacles. The first stage of this method is constructing a coarse path from UAV current position to the target,

based on the threats position. The environment is mapped through a voronoi diagram to find the best path avoiding threat locations. Through a search graph algorithm, a minimum cost path can be found. However, the paths will probably be infeasible due to the UAVs turning radius constraints; where unflyable sharp corners that are resulted from joining the path segments. The main advantage is that construction and search steps are inexpensive. The second stage is constructing a fine path using information from the first stage. In order to refine a feasible optimum path using cubic splines, the path is built incrementally, parts of the path near the UAV are quickly calculated, while the farther parts are built after the UAV gets nearer.

Q-learning path planning

Q-learning algorithm is one of the model-free reinforcement learning methods and it is used to find the optimal action selection strategy in Markov Decision Process (MDP). The basic idea of Q-learning is that Agent learns an action value function to maximize the cumulative rewards obtained from the environment. In this algorithm, the agent receives feedback from the environment based on the agent's action and learns the optimal policy by trial and error. Immediate feedback(reward or punishment) will provide to the agent every time it acts [19].

The agent learns from the environment's feedback with no prior training data. The agent fills the q-table by exploring and exploiting each action (up, down, right, left, forward, backward) for each state until convergence. That's why, the agent will perform well and be able to take the best action in each state through the real flight.

Deep-Sarsa based path planning

For the first time in 2018, Deep-Sarsa was developed to be used in autonomous path planning and obstacle avoidance of UAVs in a dynamic environment [20]. Deep Sarsa is an on-policy algorithm combining traditional Sarsa - which is a slight variation in the Q-learning algorithm - with neural networks. Sarsa uses the action performed by the current policy to learn the Q-value, while neural networks are used instead of the Q-table where Q-values are stored.

The UAVs start training with no knowledge about the environment, and for exploration, UAVs take random actions: up, down, left, right and still, until they have gained enough experience and understand the situation.

The model's input used for real experiments contains 14 components with different infor-

mation from different considerations, namely relative position between UAV and targets, the relative position between UAV and obstacles, obstacles' moving direction and the rewards. The model's output is a possibilities array for the five available actions for UAV in each step. This model was simulated in an environment in 'ROS-Gazebo' [20].

2.3 Market survey: Drones and their Autopilot systems

Global Drones Components market are segmented by components into: Frames, Controller, systems, Battery, Propulsion systems, Camera, Navigation systems, and Wings categories. While they are segmented by Drone type into: Fixed Wing, High-Altitude Long Endurance (HALE), Unmanned Combat Aerial Vehicle (UCAV),Medium-altitude Long-endurance (MALE) Unmanned Aerial Vehicle, Vertical take-off and landing (VTOL). And based on applications, they are segmented into: Military & Defense, Civil Commercial, Logistics & Transportation and Construction & Mining [21].

Drone autopilot system components are shown in Figure 4.1.

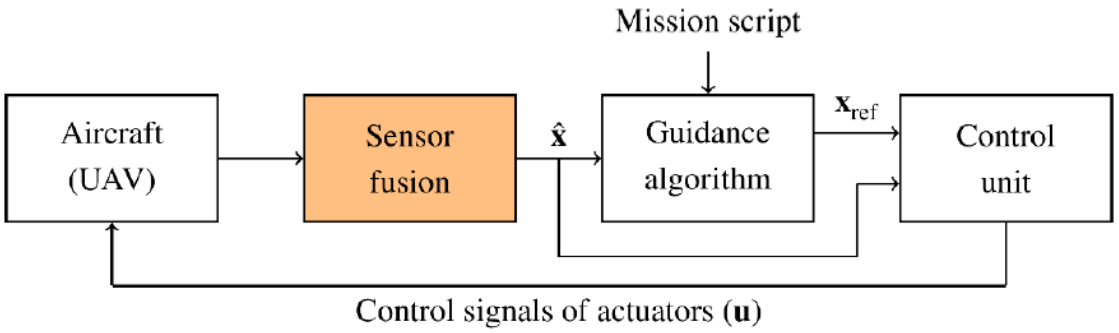


Figure 2.2: Autopilot Main Components

Sensors

Based on the sensor, the market includes inertial sensors, image sensors, speed & distance sensors, position sensors, pressure sensors, ultrasonic sensors, altimeter sensors, current sensors, light sensors, and other sensors. The inertial sensor segment is further

segmented into accelerometers, gyroscopes, tilt sensors, and magnetometers. The image sensors segment is further divided into thermal, infrared, and multispectral. Speed and distance sensors are further divided into radio detection and ranging (radar), LIDAR, and proximity. The position sensor segment is further sub-segmented into GPS and Global Navigation Satellite System (GNSS). The image sensors segment is projected to register a higher Compound annual growth rate (CAGR) by 2027 due to the increasing application of aerial photography in the commercial sector [21].

Control unit

The global drone flight control systems market is propelled by the advancing technological approach particularly focused on enhancing the aircraft market by including Artificial Intelligence (AI) and other influential technologies. Artificial intelligence is the most appealing technology in the current global market scenario with the benefit of enabling to acquire real-time data. For instance, University of Zurich implemented an advanced control quadcopter with a model predictive control were able to perform extreme acrobatic maneuvers. Numerous other control techniques have been applied to quadcopters as such as Proportional Integral Derivative (PID), Linear Quadratic Regulator (LQR) [22].

Flight control platforms

Examples of Open-Source Flight control platforms with autopilot systems are derived from this survey [23]:

- PX4: An ARM-based platform consisting of a Flight Management Unit (FMU), PX4-IO, and extra Memory all integrated on a single board. The project follows the Berkeley Software Distribution (BSD) License.
- PX2: Made through the collaboration of PX4 and Ardupilot teams, consisting of triple IMU's, a maximum of 3 GPS modules, a single DF17 connector holding all I/Os. It follows the CC-BY-SA-3.0 License.
- Paparazzi: Based on STM32F7 Microcontroller Unit, contains autopilot software for fixed-wing, hlicopters, multi-rotors and hybrid aircrafts. It follows the GPL License.
- CC3D Atom: Both based on LibraPilot. They contain all types of LibraPilot compatible stabilization hardware onboard, and are very flexible, they can be configured to fly aircrafts starting from fixed-wings to octocopters. Follow the GPLv3 license.

- ArduPilot Mega (APM): Based on an ArduinoMega, it can control autonomous fixed-wings, multi-copters, helicopters, ground rovers, and antenna trackkers. They follow the GPLv3 license.
- FlyMaple: Based on an Arduino style ARM processor called maple exclusively made for quadcopters. It's hardware is designed for balancing vehicles which require Inertial measurement units (IMUs) and high performance realtime controllers. Follows the GPLv3 license.
- Erle-Brain 3: It is a Linux system based on a Raspberri Pi integrated with an autopilot shield called PXFmini with multiple sensors, IO, and power electronics. Follows the CC BY-NC-SA license.

2.4 Regulations and Laws

UAVs usage is regulated by the national aviation authority (NAA) of the country. Nevertheless, the International Civil Aviation Organization (ICAO) has been exploring the usage of drones, and they have serious safety concerns regarding that technology [18]. In this work, we will use a simulation environment and the necessary approvals will be obtained when testing the actual system.

Chapter 3

Proposed Design

3.1 Problem Definition and Project Purpose

Using Microsoft AirSim API for data collection and high-level vehicle control, it is required to go through all the track gates with the minimum number of collisions and minimum time depending on the first-person view (FPV) camera as the only source of information. We split the problem into two separate subproblems; perception (extraction of useful information about the environment from the input images) and control policy determination (what action to take in each situation). In the deployment phase, we combine the two modules, perception and control policy) into an integrated system.

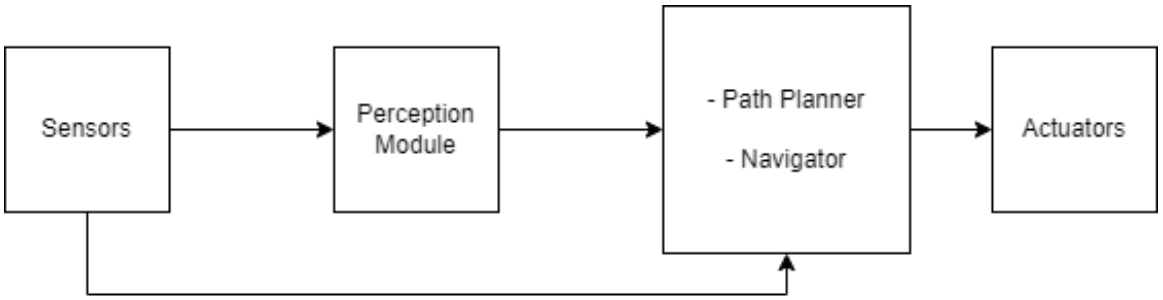


Figure 3.1: Project Flow

3.2 Proposed Designs for Perception Module

3.2.1 Requirements

Ideally, the perception module should provide the following:

1. Transformation information (distance and orientation angles) about the gates insight.

2. High versatility detection; as it would be required to detect gates with different shapes, colors, and sizes in average and harsh environmental conditions, ex: fog, poor lighting, ..etc.

By achieving this, the control policy will be able to take the required action. However, extracting rich and accurate information from raw 2-D images is not an easy task, thus,

perception modules can be designed to solve a specific problem, detecting rectangular gates in daylight for example. making such assumption makes the implementation a lot easier. Hence, we make our sets of assumptions, try to come up with multiple designs, and compare the different approaches to implement one.

3.2.2 Approach 1: Identifying the Gate Center

A naive approach used by the USRG team in the Game of Drones competition [24]. The key idea here is once the gate is identified at a given frame, the drone should aim at its **observed** center.

Assumptions

- The observed image will be treated as 2D projection of the environment.
- If multiple gates are observed, the nearest gate is the one that occupies more area of the image.

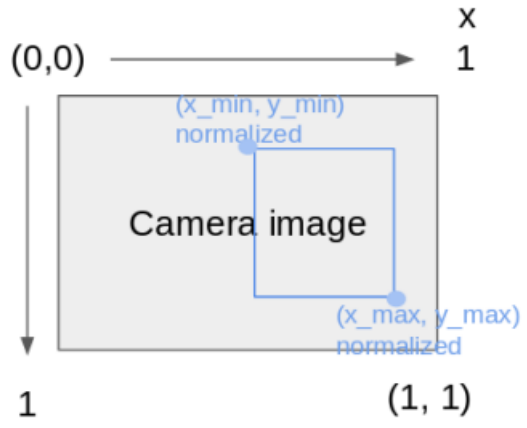
Design

Once a gate is detected, the drone aims to the its *observed* center. To achieve this, we:

- Need to detect the gate. This can be done by using *MobileNetSSD* Neural Network Neural Network (NN) as they provide faster object detection than the other state-of-the-art alternative YOLO [24] and MobileNet is suitable for mobile vision-based applications [25]. The network is trained on a dataset collected using AirSim. The dataset contains images of gates labeled with a bounding box and the coordinates of the vertices of the gate in the image frame from Figure 1.
- Determine the center of the observed gate in the image frame by averaging the coordinates of the gate vertices.

[b]

Figure 3.2: The output of the NN.



[b]

Figure 3.3: Illustration of how to find the gate center in the image frame.

3.2.3 Approach 2: Extracting relevant transform and visual information from the scene

This approach tries to extract a lower dimension representation of the input image that resembles the needed transform information for navigation.
Given a large dataset of gate images labeled with their spacial information like relative spherical coordinates to the drone, and yaw angel; a VAE can reduce an image t into a latent vector $Z_t \in R^N$, the where Z is a vector containing transform and visual information about the gate pose and background environment [26]. We adopt

the approaches used in [26] and [27]. [27] introduced a cross-modal method based on the VAE framework to estimate 3D joint configuration of hand poses from raw 2D images. The same approach was applied in [26] to determine gates orientation in space using two input modalities, RGB images and gate pose identified as $[r, \theta, \phi, \psi]$, the relative spherical coordinates of the gate to the drone and the yaw angle, respectively.
Due to its generative nature, VAE is capable of producing a continues, smooth, and regularized latent space unlike standard autoencoders [27].

Assumptions

- A model can be built to predict a gate pose if trained on sufficiently large set of poses.

Design

- We use AirSim to generate a large dataset of images containing variations of distances and orientation angles of gates. the images are labeled with their corresponding poses
- Using couple of encoder-decoder pairs, we train a joint latent space R^N with both modalities so that we can reconstruct an output in both modalities with only one modality input (RGB image).
- According to [27], in CM-VAE, The objective function that should be maximized is

$$E_{z\sim q(z|x_j)}[logp(x_i|z)] - D_{KL}(q(z|x_i)||p(z)) \tag{3.1}$$

  where x_i, x_j are the input data modalities, $q(.)$ is an encoder network, and D_{KL} is the Kullback-Leibler divergence.

3.3 Proposed Designs for Navigation Module

3.3.1 Requirements

The requirements for this task is planning a path to pass through the center of 10 checkpoints gates while avoiding hitting the gates' corners in the minimum time.

3.3.2 Approach 1: Deep Reinforcement learning

Deep reinforcement learning (DRL) is proven to effectively learn several reward driven skills including playing games and navigating complex environment [28]. The quad-rotor will be trained in a MDP to learn to plan the optimized path. The agent will be trained with Policy Proximal Optimization (PPO) which is an advantage actor-critic algorithm that tries to be conservative with policy updates [28]. The main advantage of using an algorithm depending on Markov decision process is creating paths with non-holonomic constraints. Furthermore, policy proximal optimization (PPO) is easy to implement and can be reproduced with fewer hyper-parameters than other algorithms.

Assumptions

There is no perception for the gates or other objects but the gates' positions are known from a competitor drone.

Design

reinforcement learning (RL) allows the drone to learn the best action to take in each state during the training then the drone apply this learning to pass through the gates safely. Learning procedure is discussed as follows:

– The actual drone racing track from AirSim APIs will be used with applying some modification in some parts.

- During the learning process the training agent will be started at different locations to learn to recover its path
- During training, agent gets reward for each step taken in the environment. Completing all subtasks (gates) and finishing race will be considered one episode of MDP.
- The competitor drone will be used here to collect guidance data via API calls. A simple path planning algorithm is used for the competitor.
- IMUs sensors will be used for raw data which contains information about linear velocity, angular velocity, and gyroscope data for pitch, roll, and yaw data. GPS data will also be used. These data will work as inputs for the actor-critic model with the number of actions
- The drone has seven legal actions that are hovering (making nothing), forward, backward, right, left, up, and down.
- The actor-critic model output will be mapped to its corresponding action then to the velocities for the drone's actuators.
- The implemented low-level drone PI controller will be used as we are interested in path planning; Not improving PID controller with the drone velocities as its input.
- We will Use 3 layered neural network as a function approximator with enough training time to result in a good performance and reduce the error in the velocity output.
- The center positions for the gates are known so that we can decide if the drone has passed the gates or has hit their corner.
- After training the drone, it will be able to fly on different environments.

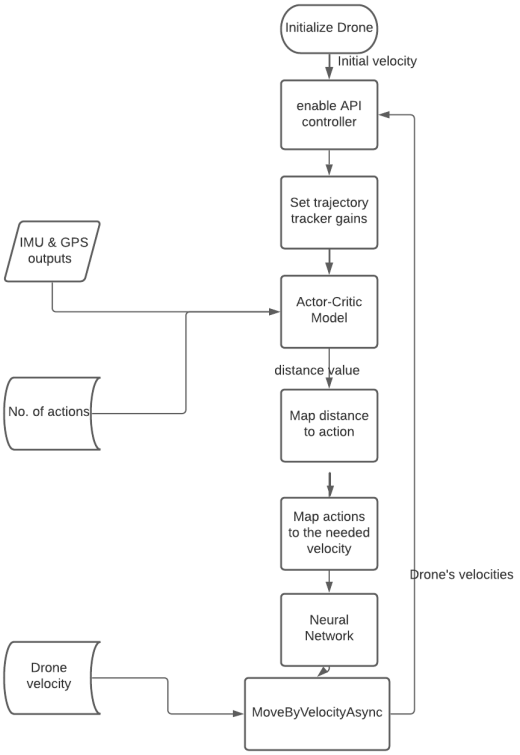


Figure 3.4: Block Diagram of the RL Model design.

3.3.3 Approach 2: Imitation learning

An alternative control system is an Imitation Learning (IL) model that was proposed by Bonatti and Madaan [26]. Imitation learning is simply learning from demonstrations and different algorithms in order to aquire a specfic behaviour [29]. The purpose of imitation learning is to efficiently learn a desired behavior by imitating an expert's behavior; an expert could be a human or another algorithm. It was used in many robotics applications for example; AlphaGo, the algorithm which was able to beat a human Go master, initializes a deep neural network policy from human demonstrations [30]. Generally, imitation learning is branched into replicating the desired behavior that is called behavioral cloning BC and learning the hidden objectives (reward function) of the desired behavior from the demonstrations that

is called Inverse Reinforcement Learning (IRL). However, our work focuses on the Behavioral Cloning (BC) techniques [29]. BC methods learn a direct mapping from states to actions or trajectories without recovering the reward function.

Why using imitation learning?

Collecting expert demonstrations to cover all situations is usually too expensive and time-consuming so the demonstrative data set does not cover all possible situations [29]. As a result, the learner often encounters states which were not encountered by the expert during demonstrations, which means that the target domain distribution is different from the source distribution. Therefore, domain adaptation is closely related to imitation learning.
Imitation learning is closely related to reinforcement learning (RL) through obtaining a policy that maximizes an expected reward. In RL, we employ a reward function that encourages a desired behavior. In imitation learning, however, we assume optimal expert demonstrations which basic reinforcement learning does not provide, and which provide prior knowledge allowing for significantly more efficient methods. It demonstrates a potentially exponential decrease in sample complexity when learning a task by imitation rather than trial-and-error reinforcement learning, and empirical evidence has long demonstrated such benefits.

Behavioral Cloning

BC methods learn a direct mapping from states/contexts to trajectories/actions without recovering the reward function. Behavioral cloning can be an efficient way to reproduce the desired behavior. The controller of a robotic system with an imitation learning consists of the upper-level controller that plans the desired trajectory based on the provided context. Furthermore, the lower-level controller determines the control input to achieve the desired trajectory, so imitation learning learns these controllers. While learning trajectory; imitation learning aims to learn the policy to produce the desired trajectory. Nevertheless, in action-state space learning; it aims to learn a policy that generates a control input for a provided state or context. Using the data sets of expert's demonstrations, a policy can be learned as the direct

mapping from the context to the trajectory or from the state to the control input. This learning problem can be defined as a supervised learning problem in which a policy can be recovered by solving a simple regression problem. We call this approach behavioral cloning. Recording a set of expert's demonstrations such as trajectories is the first step in the behavioral cloning algorithm. Then a policy representation and objective function that demonstrate the similarities between the desired behavior and the learner's policy are chosen. The policy parameters are then optimized using the policy demonstrations. Behavioral Cloning is the method we used in our control model as it's easy to implement with efficient performance [29].

Design choices for Behavioral Cloning

– The appropriate surrogate loss function is chosen to quantify the differences between the demonstrated behavior and the learned policy. Since it affects how to train the policy, selecting the appropriate loss function is crucial to achieve efficient learning. The quadratic loss function is the most common choice for the loss function in BC. L_1 loss function, log loss function are other usable loss functions for BC. In our behavioral cloning model we use the mean squared error loss function that is commonly used in regression. It calculates the mean of the squared differences between the demonstrated and the learned values [29].

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y})^2$$

where y is the demonstrated value and $\hat{y}$ is the predicted value.

– The appropriate regression method should be used for a better system performance. For example; Gaussian Model, Neural Network, Linear Regression, and Decision Tree are regression models for model-free behavioral cloning [29]. In our model Neural Network is used. One should choose the regression model with the suitable complexity. Simple models that can be trained using linear regression are easy to train but not informative. Complex models can be trained by a neural network, however, training needs large numbers of training

data. Furthermore, imitation learning cannot be addressed as simple supervised learning in many applications. In this case, a method for reducing from imitation learning to supervised learning with interaction is applied.

Assumptions

– Future gates don't affect the objective function of the current gate tracking problem.

Design

- Expert's Actions

Expert data will be generated using minimum snap trajectory planner method as proposed by Mellinger and Kumar [31]. In order to facilitate the automated generation of trajectories, the states and the inputs of the quad-rotor model should be written as algebraic equations of four selected outputs and their derivatives. The flat outputs are $\sigma = [x, y, z, \psi]$ where $r = [x, y, z]^T$ are the coordinates of the center of mass in the world coordinate system and ψ is the yaw angle. A trajectory,$\sigma(t)$, is defined as a smooth curve in the space of flat outputs:

$$\sigma(t) = [t_0, t_m] \rightarrow R^3 \times SO(2) \tag{3.2}$$

This method generates an optimal trajectory transitions through specified m keyframes smoothly while staying within specified safe corridors. Conveniently, the trajectories in the flat output space are written as piece-wise polynomial functions of order n over m time intervals as following:

$$\sigma_T(t) = \begin{cases} \sum_{i=0}^{n} \sigma_{Ti1} t^i & t_0 \leq t < t_1 \\ \sum_{i=0}^{n} \sigma_{Ti2} t^i & t_1 \leq t < t_2 \\ \vdots & \\ \sum_{i=0}^{n} \sigma_{Tim} t^i & t_{m-1} \leq t \leq t_m \end{cases} \tag{3.3}$$

The aim is to find trajectories that minimize functionals which can be written using

these basis functions. The optimization program is required to solve this problem while minimizing the integral of the k_r derivative of position squared and the k_ψ derivative of yaw angle squared as following:

$$\begin{aligned} \min \int_{t_0}^{t_m} \mu_r || \frac{d^{k_r} r_T}{dt^{k_r}} ||^2 + \mu_r \frac{d^{k_\psi} {\psi_T}^2}{dt^{k_\psi}} dt \\ \text{s.t.} \qquad \sigma_T(t_i) = \sigma_i, \quad i = 0, ..., m \\ \frac{d^P x_T}{dt^P}|_{t=t_j} = 0 \quad or free, \quad j = 0, m; \dot{P} = 1, ..., k_r \\ \frac{d^P x_T}{dt^P}|_{t=t_j} = 0 \quad or free, \quad j = 0, m; \dot{P} = 1, ..., k_r \end{aligned} \tag{3.4}$$

where μ_ψ and μ_r are constants that make the integral non-dimensional, $\sigma_t = [x_T, y_T, z_T, \psi_T]^T$, $\sigma_i = [x_i, y_i, z_i, \psi_i]^T$.

It is known that human reaching trajectories tend to minimize the integral of the square of the norm of the jerk (the derivative of the acceleration) [32]. However, trajectories that minimize the integral of the square of the norm of the snap, the second derivative of the acceleration, k_r=4, will be generated in this system as the inputs of the quad-rotor model appear as functions of the fourth derivatives of position. k_ψ=2 is used since one of the inputs appears in the second derivative in the yaw angle.

- BC Models

  Inputs of the behavioral cloning model is mainly an encoded CM-VAE latent space vector Z_t composed of the main features that recognize the gate image. This vector should include both modalities, the RGB component of the image, and the relative pose of the next gate to the current aircraft frame. There are many types of image features extractors. We are mainly using CM-VAEs, and direct regression from images to features.

  As explained in the VAE section, producing this vector requires training of a full network including both encoding and decoding (reconstruction) steps. The decoding step, especially, can be done through different ways. Depending on the number of decoders applied on the gate poses part of the latent space vector, the data input is set to be either constrained or unconstrained. If a single gate pose decoder is used,

the vector is unconstrained. However, based on our knowledge that each gate pose component is independent from all other components (e.g, the distance between gate and drone is independent of the gate's orientation), constrains could be applied to the gate pose vector. This can happen by using four different decoders (1 for each component or feature) instead of using only one. Outputs of these two networks are two CM-VAE latent space vectors composed of 10 components each, namely Z_{con} and Z_{unc}. Instead of using an auto-encoder, pure supervised regression can be used as well to transform images into a vector of gate poses as features. This method produces vector Z_{reg} composed of only 4 components, those which represent each gate relative pose features.

We have 3 different options for training our policies. Policies BC_{con} and BC_{unc} are trained based on the constrained and unconstrained data, where Z_{con} and Z_{unc} vectors are used as inputs to the behavioural cloning neural network. Z_{reg} is used as a data input for training a separate policy, namely BC_{reg}. The output of training data set is the expert's behavior represented in a 4 component velocity vector, including both linear and yaw velocities of the drone.

For BC_{con} and BC_{unc}, a 4 dense layer network is used. The first layer is of 256 units. This number is halved for the second hidden layer, and through the hidden layers, the number of units of each layer is the half of the previous layer's number of units. The output velocity vector components requires 4 separate units at the output layer. The activation function for the input and hidden layers is a "ReLU", while a linear activation function is used in the output layer. For BC_{reg}, a convolutional neural network is used. It consists of 10 2D convolution layers, along with 2 dense hidden layers with a 'ReLU' activation function, and an output layer with a linear activation function. The first 4 convolutional layers are of 32 filters (including the input layer), the following three are of 64 filters, and the last 3 are of 128 filters. Kernel L2 regularization is applied to all convolutional layers excluding the first and the last. The hidden layers are of 64 and 32 units, and the output layer is of 4 units. The objective is to find the optimal policy parameters Θ^* and Φ^* that minimizes the expectation of the distance between our control policy and the expert, taken over different observed states (equation 1). The expert policy operates with full knowledge of the environment, while our policy has only access the observation (feature vector).

$$\Theta,\Phi = \underset{\Theta,\Phi}{\operatorname{argmin}}\ E_s[D(\pi^*(E), \pi^{\Phi}(q^{\Theta}_{RGB}(I))] \tag{3.5}$$

where D is the distance between the two policies, π^* is the expert's policy, I is the observation, ${q_{\mathrm{RGB}}}^{\Theta}$ is the encoding function producing the latent space vector, and π^{Φ} is the trained policy.

BC Navigation Models summery			
POC	$\mathrm{BC_{unc}}$	$\mathrm{BC_{con}}$	$\mathrm{BC_{reg}}$
CM-VAE	1 decoder used for all gate poses components (2 decoders in total)	1 decoder used for each gate pose component (5 decoders in total)	No Auto-encoder used (direct regression of the image into a feature vector)
Input	Vector of 10 components	Vector of 10 components	Image regressed into a Vector of 4 components
Output	Vector of 4 components	Vector of 4 components	Vector of 4 components
Network layers	- Hidden layers (256, 128, 64 and 32 units) - Output layer (4 units)	Similar to $\mathrm{BC_{unc}}$	- 10 2D Convolutional layers (32, 32, 32, 32, 64, 64, 64, 128, 128 and 128 filters) - 2 Hidden layers (64 and 32 units) - Output layer (4 units)
Activation functions	'ReLU' for the input and hidden layers, linear for the output layer	Similar to $\mathrm{BC_{unc}}$	different combinations for the convolutional layers, 'ReLU' for hidden layers and linear for the output layer

Table 3.1: Summery of BC Navigation models

According to [26], BC_{unc} and BC_{con} had quiet similar performance while they outperformed all other models. We BC_{unc} for this proposed design since we cannot assume that the gate poses are fully independent of each other. The gates we are required to detect is not of one color, and its characteristics change significantly as we change its poses.

3.4 Summery of proposed designs

Perception Models		
Design	Pros	Cons
Approach 1 (Aiming at Gate Center)	Easy in implementation. Does not need large dataset to be trained on.	Does not take the orientation of the gate into consideration. Will not give an optimal path in most of the cases as it plans on the projection of the gate, not the original position.
Approach 2 (NN to esimate the Orientation of the Gate)	Gives richer information about the relative coordinates of the gates. Gives a better path as it has more information.	Needs large dataset that includes samples of the possiple orientations of the gate. We will not be able to experiment several modifications as the NN Will require hours to be trained .
Navigation Models		
Design	Pros	Cons
Approach 1 DRL	Creating paths with non-holonomic constraints	Complexity of policy learning
Approach 2 IL	Simplicity and facilitated implementation	High computational cost

Table 3.2: Summery of proposed designs

3.5 Selected Design

Based on the previous analysis of different models, for the perception module, we choose the second approach: CM-VAE to estimate the gate poses. For the navigation module, we choose the unconstrained BC imitation learning model. [26] used this design to detect and fly through red gates at slow speeds. Our job is to detect checkered race gates, and to pass through them all in minimum time (higher speed).

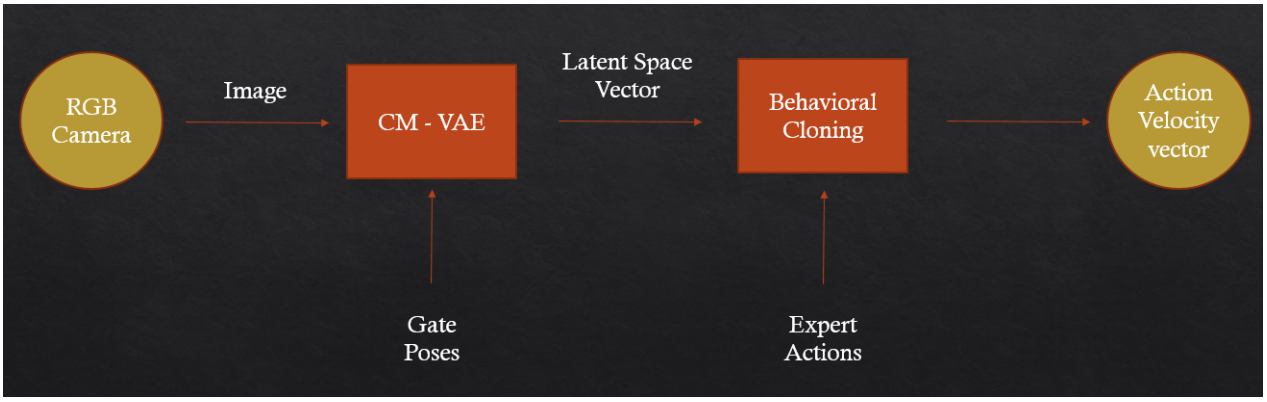


Figure 3.5: Selected Design Flow

Chapter 4

Implementation

4.1 Pecreption Unit

4.1.1 Perception Training data

Using *airsimdroneracingvae* package Application Programming Interface (API) to generate a dataset of 300k gate images and their corresponding poses and save the images in png format and write the gate poses into a Comma-separated-value (CSV) file.

- We define ranges for the values of the drone coordinates (X, Y, Z) and orientation angles (θ, ϕ, ψ) in the world frame.
- For each data point, we uniformly sample a pose (position and angles) for the drone from these ranges and set the drone pose accordingly.

```
    UAV_X_RANGE = [-30, 30] # world x quad
    UAV_Y_RANGE = [-30, 30] # world y quad
    UAV_Z_RANGE = [-2, -3] # world z quad

    UAV_YAW_RANGE = [-np.pi/4.0, np.pi/4.0]
    eps = np.pi/10.0  # 18 degrees
    UAV_PITCH_RANGE = [-eps, eps]
    UAV_ROLL_RANGE = [-eps, eps]

    # create and set pose for the quad
    p_o_b, phi_base = randomQuadPose(UAV_X_RANGE, UAV_Y_RANGE,
UAV_Z_RANGE, UAV_YAW_RANGE, UAV_PITCH_RANGE, UAV_ROLL_RANGE)
    self.client.simSetVehiclePose(p_o_b, True)

```

- We do the same two steps for the gate pose.
- Uing the AirSim API we record the image from the camera of the drone (client) and the gate pose relative to the drone frame.

```
    # request quad img from AirSim
    image_response = self.client.simGetImages([airsim.
ImageRequest('0', airsim.ImageType.Scene, False, False)])[0]
    # save all the necessary information to file
    self.writeImgToFile(image_response)
    self.writePosToFile(r, theta, psi, phi_rel)


```

4.1.2 Variational Auto-Encoder

For the multi-modal learning sake, we implement three neural networks, one encoder and two decoders, and train them together with respect to the compound loss function. Generally, the outputs of the VAE encoder network are a mean μ and a standard deviation Σ vectors. The latent vector z is then sampled from a multi-variant normal distribution parameterized by the given means and standard deviations. This is known as the reparameterization trick and it is necessary for backprobagation to run through a sampler neural network [35]. The latent vector is sampled according to the equations

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\Sigma} \odot \boldsymbol{\varepsilon}. \tag{4.1}$$

where $\odot$ is element wise multiplication and

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \boldsymbol{I}) \tag{4.2}$$

Then, the same latent vector will be used in both decoder networks to estimate the gate pose and regenerate the image. We could have used only one decoder, i.e. gate decoder, nevertheless, we wanted to make sure that the learned representation capture not only the geometric information but also also information about the RGB space of the image.

Encoder

We use the DroNet[33] figure 4.1 which mainly consists of three convolutional residual blocks figure 4.2 followed by three Fully Connected (FC) layers. Each residual block consists of a batch normalization layer through the channel axis followed by a Rectified Linear Unit (ReLU) non-linearity, then a convolution layer followed by a batch normalization layer through the channel axis and a ReLU non-linearity. Figure 4.1 shows all the details for each layer in the architecture; The input image first goes through a convolutional layer with 32 filters before being fed to the residual blocks. All layers use same padding. The layers of the three residual blocks use HE initialization and L2 regularization with 1×10^{-4} regularization factor. The output of the final residual block is flattened and passed through a ReLU non-linearity. The model is terminated by three FC layers with ReLU activation, the first has 64 neurons, the second has 32 neurons, and the final layer has twice the number of neurons as the latent vector dimension i.e. 20.

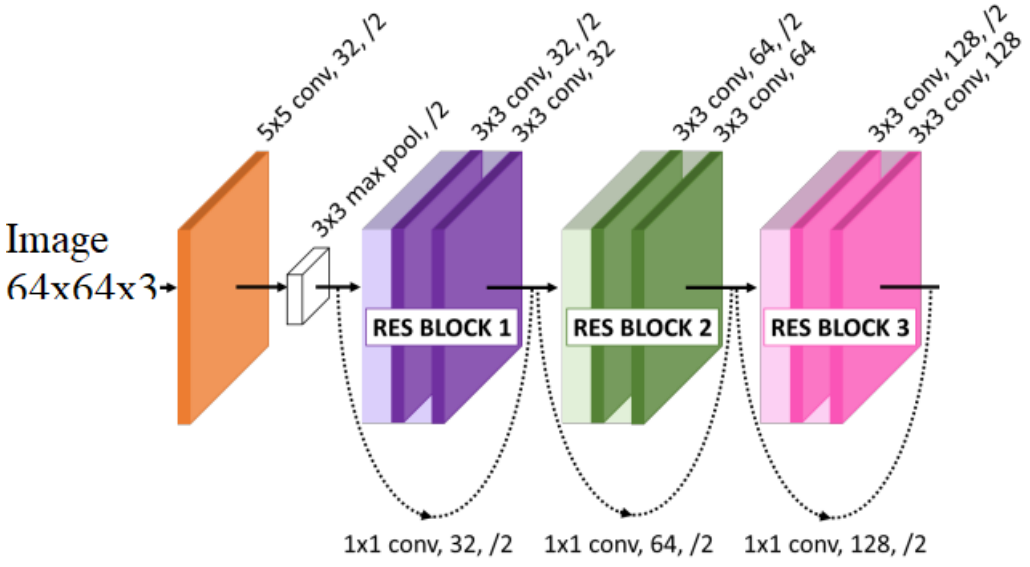


Figure 4.1: DroNet Architecture

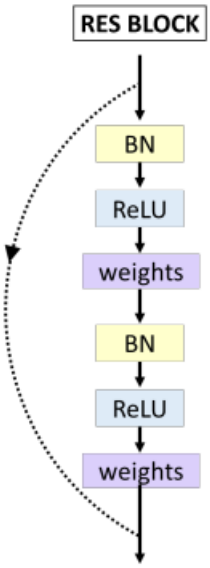


Figure 4.2: Residual Block

Image Decoder

The image decoder takes the latent vector $z \in \mathbb{R}^{10}$ as input and tries to reconstruct the original image 64x64x3. It consists of a 1024 unit FC layer then six transpose convolutional layers with the following specifications:

```
# for 64x64 image
deconv1 = Conv2DTranspose(filters=128, kernel_size=4, strides=1,
    padding='valid', activation='relu')

deconv2 = Conv2DTranspose(filters=64, kernel_size=5, strides=1,
    padding='valid', activation='relu', dilation_rate=3)

deconv3 = Conv2DTranspose(filters=64, kernel_size=6, strides=1,
    padding='valid', activation='relu', dilation_rate=2)

deconv4 = Conv2DTranspose(filters=32, kernel_size=5, strides=2,
    padding='valid', activation='relu', dilation_rate=1)

```

```
deconv5 = Conv2DTranspose(filters=16, kernel_size=5, strides=1,
    padding='valid', activation='relu', dilation_rate=1)

deconv6 = Conv2DTranspose(filters=3, kernel_size=6, strides=1,
    padding='valid', activation='tanh')
```

Gate Decoder

The gate decoder is an FC network that takes the latent vector z as input and outputs an estimation for the gate pose information $[r, \theta, \psi, \phi]$. Other than the input and output layers, there are four ReLU activated layers with 512, 128, 64,16 hidden units respectively.

```
layer_1 = Dense(units=512, activation='relu')
layer_2 = Dense(units=128, activation='relu')
layer_3 = Dense(units=64, activation='relu')
layer_4 = Dense(units=16, activation='relu')
output = Dense(units=4, activation='linear')
```

4.2 Cross Modal Training

First, the images and corespondent gate poses are stored into numpy arrays. The images are read in BGR format, and the model is trained in this format as well. The entries of the image and gate arrays are mapped to range $[-1, 1]$ to speed up the training process.

The data is then split into 90% training and 10 % validation sets.

We set the batch size to be 32, learning rate to be 1×10^{-4}, and using Adam optimizer.

```
batch_size = 32
epochs = 51
learning_rate = 1e-4
optimizer = tf.keras.optimizers.Adam(lr=learning_rate)
```

A linear combination of mean square error between the reconstructed (images, gate pose) and the ground truth (image, gate pose) and Kullback-Leiber Divergence (D_{K-L}) as an optimizing objective function.

```
@tf.function
def compute_loss(img_gt, gate_gt, img_recon, gate_recon, means,
    stddev):
    """
    Args: ground truth image, ground truth gate pose, reconstructed
    image, reconstructed gate pose, means an std vector from the
    encoder.
    """
    # compute reconstruction loss
    img_loss = tf.losses.mean_squared_error(img_gt, img_recon)
    # img_loss = tf.losses.mean_absolute_error(img_gt, img_recon)
    gate_loss = tf.losses.mean_squared_error(gate_gt, gate_recon)
    kl_loss = -0.5 * tf.reduce_mean(
        tf.reduce_sum((1 + stddev - tf.math.pow(means, 2) - tf.math.
    exp(stddev)), axis=1))

    return img_loss, gate_loss, kl_loss
```

We keep track of the training and validation losses during the training process. Fig.4.4 shows the loss component due to image reconstruction, fig.4.2 shows the loss component due to gate pose reconstruction, fig.4.5 shows the D_{K-L} loss component, and fig.4.6 shows the summation of theses components.

After 50 epochs, both reconstruction training errors were less that 0.02, and their test errors were less than 0.023, while the KL loss approached zero. At the end of the day, the total training error was around 0.03 and the total testing error was approximately 0.044.

From the training graphs we can observe that the gate reconstruction components converges after approximately 25 epochs which the other components keep improving at slower rate.

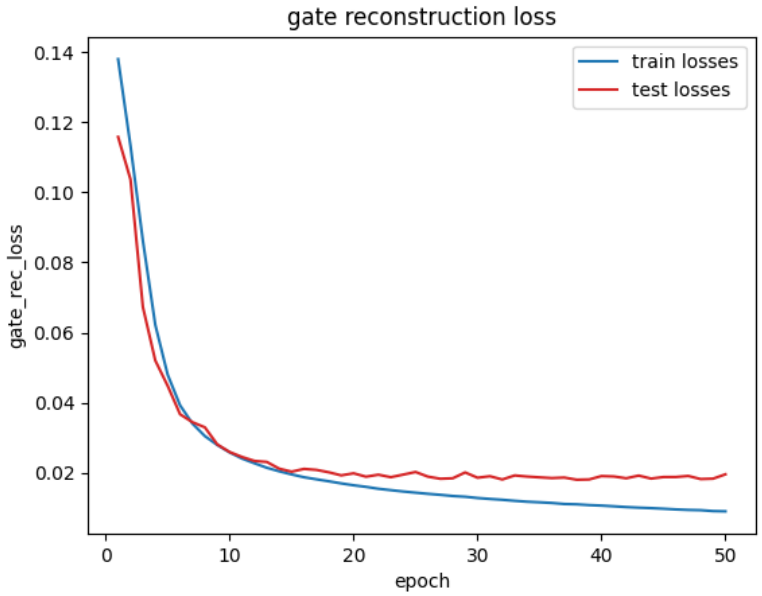


Figure 4.3: Gate Reconstruction losses

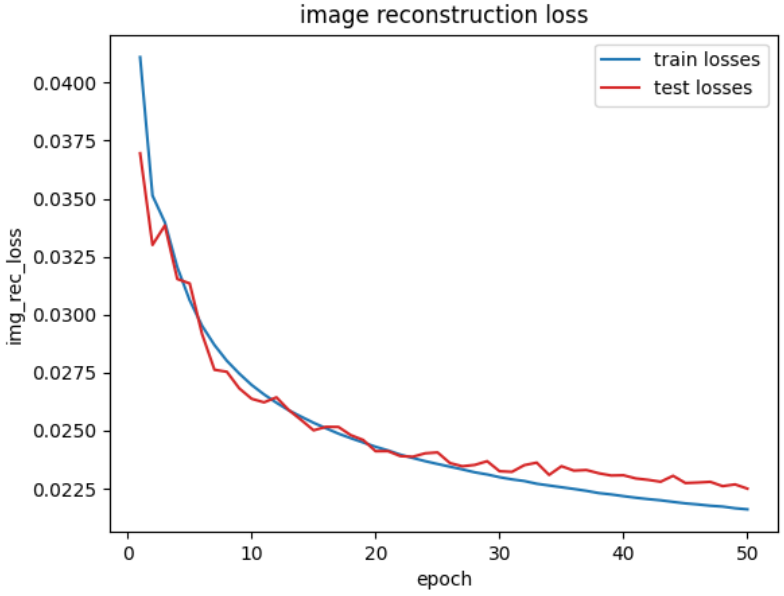


Figure 4.4: Image Reconstruction losses

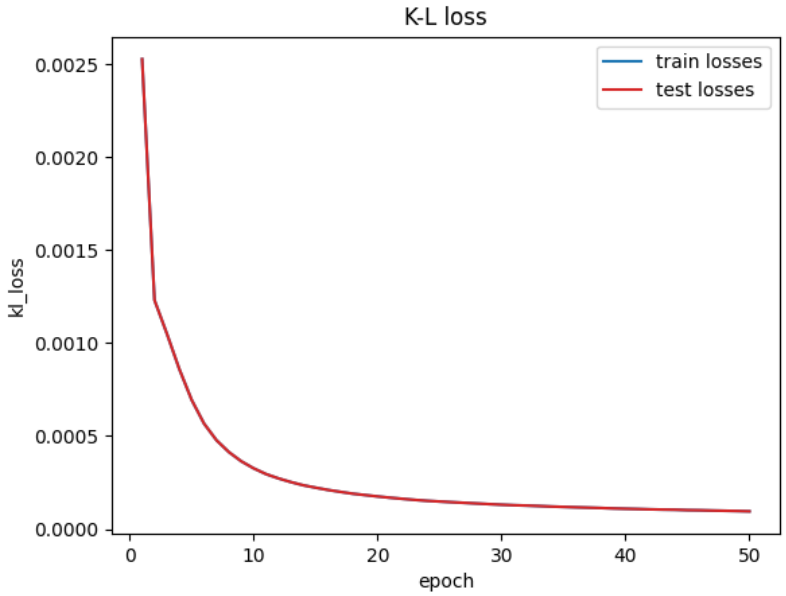


Figure 4.5: KL losses

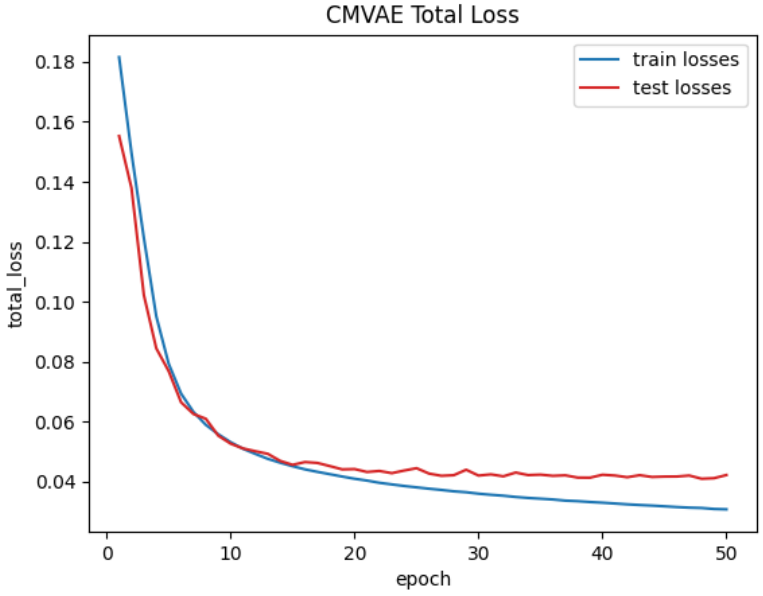


Figure 4.6: Total losses

CMVAE Evaluation

We evaluated the performance of the perception unit on a sample set of $1k$ images.Figure4.7 shows that error density of the estimated quantities, we see error density concentration at zero which is a positive indication.

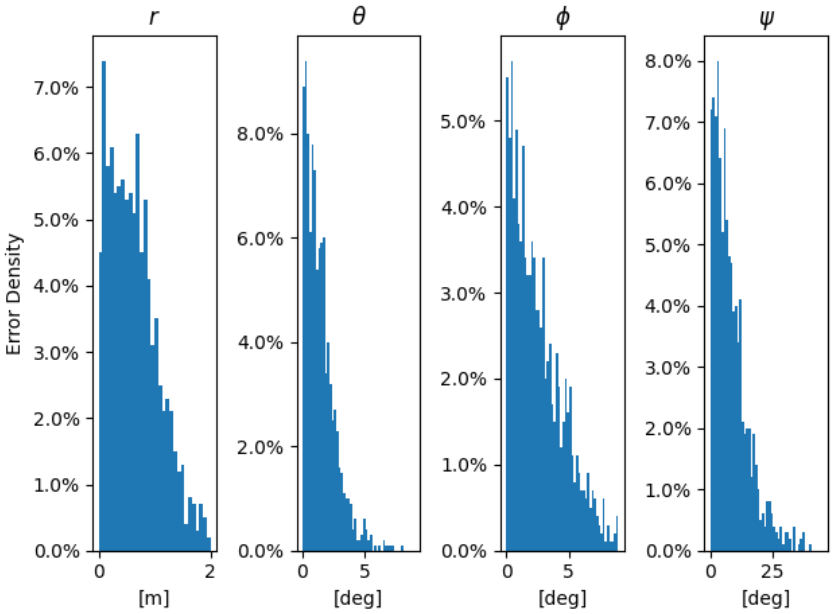


Figure 4.7: Error Densities

Figure4.8 displays 50 image- decoding pairs. We can see that the decoder network can restore much of the RGB information and preserving good estimate of the gate pose just from 10 scalar numbers.

The reconstruction is quite poor in cases where the gate is observed from its side or when the gate is far from the camera. This can be the result of using **MSE!** (**MSE!**) as an objective function; the resulting error between the original image, which is mostly green, and a blurry greenish reconstruction is already low.

Figure 4.8: Reconstructed Images

Finally, the smoothness of the latent space manifold with respect to the gate poses and image outputs is a desirable property (i.e., similar latent vectors correspond to similar gate poses). Intuitively, our single cross-modal latent space should lead to such smooth latent space representation, and our next analysis confirms that such properties emerge automatically. In figure4.9 we show the decoded outputs of a latent space interpolation between the encoded vectors from two very different simulated images. Both images and their decoded poses are smoothly reconstructed along this manifold.

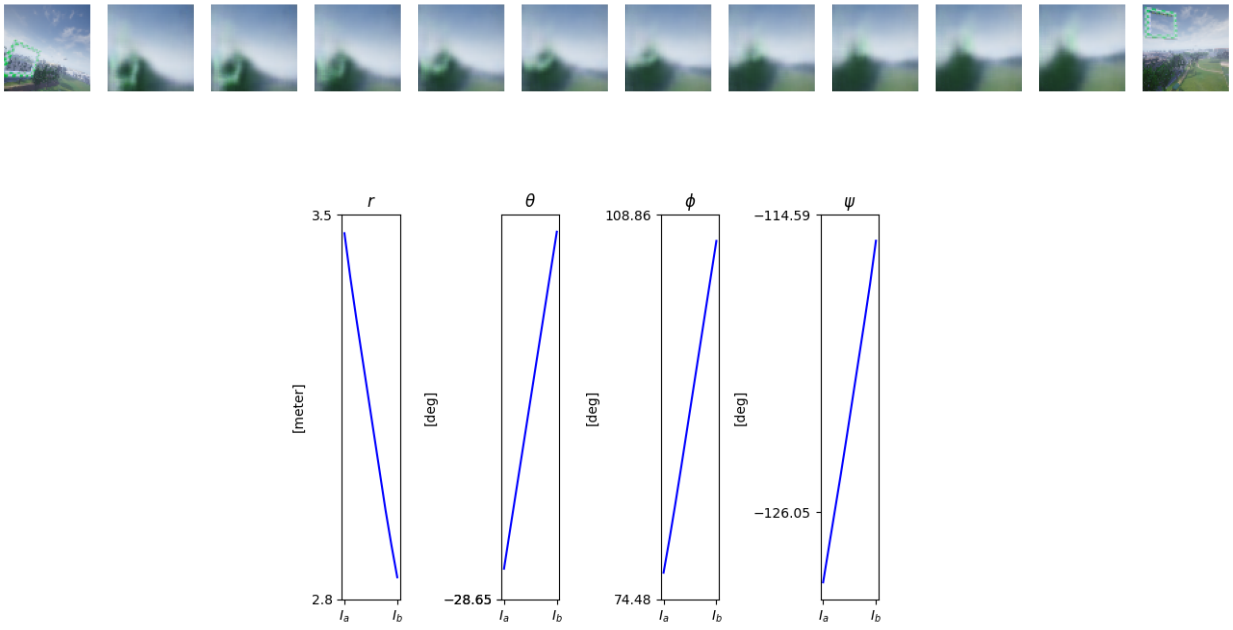


Figure 4.9: Interpolated gates

4.3 The Expert

4.3.1 The Expert's dataset

The dataset is built up from 2 components, gate snapshot images, and the required velocity vector of the drone at each snapshot. The images will be fed to the trained auto-encoder network to extract the gate features, and then used in the training dataset along with the velocity vectors. We generated more than 40,000 images with their corresponding velocity vectors.

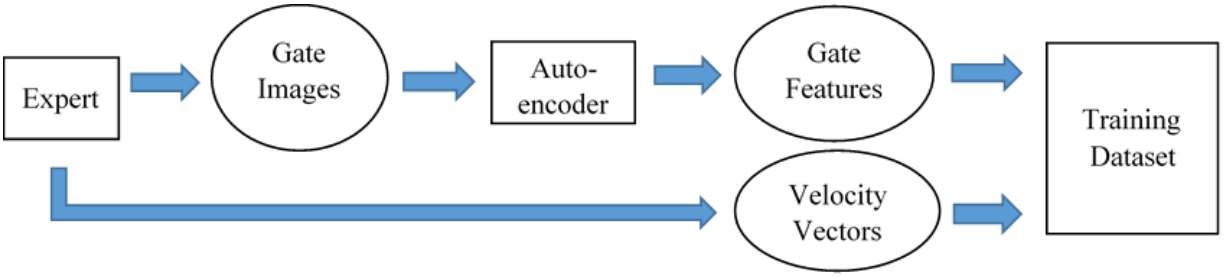


Figure 4.10: Dataset building up

4.3.2 The training environment

We need to guarantee the diversity in the gates' orientation and background, as well as their distances from the drone when generating gate images. Putting that in mind, we decided to use one of the racing environments provided by the neurIPS 'Game of Drones' competition, namely 'Soccer Field'. This environment contains different background objects (e.g. the pitch ground, side pitch, trees, clouds and sky, etc.). We also extracted gate positions of one of the race tracks provided by the competition for training, namely 'Qualifier Tier 3', where the gates can be spread all around the field. However, the average distance between each gate and its following was larger than the maximum distance our RGB camera could operate. To solve this issue, we interpolated new gate positions between the old gates, where the distance between each gate and its following doesn't exceed 8 meters. In order to generate a dataset that is big enough, the drone should pass through a large number of gates,

and take a larger number of images. We created a closed track of 60 gates. From far away, the track seem to be an irregular circle. However, the track is big enough so that the drone doesn't move in a circle, but in a random path. The drone will make 15 lab during the generation process to make the dataset large enough.

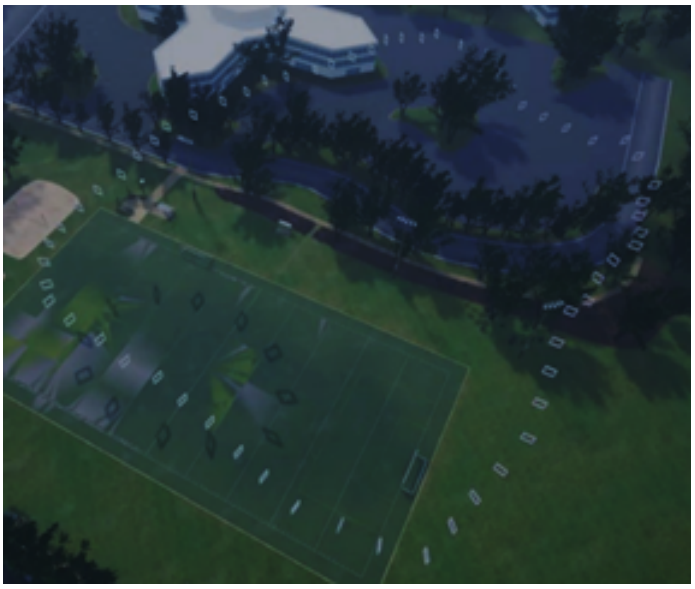


Figure 4.11: Training track

4.3.3 Expert's Trajectory generation

As explained in the design section in this report, we have chosen the minimum snap trajectory generation model for our expert. We set the position error P gain to 5.0, the velocity error P gain to 0.04 and the yaw error P gain to 0. All D gains were set to 0.0, except that of the yaw error was set to 3.0. Considering a horizon of one gate into the future, the drone could generate a minimum snap trajectory towards the center of the next gate. The suitable maximum acceleration and velocity of the expert were determined through trial and error. There was an obvious trade-off between the drones' velocity and how close the gates were to each other. To demonstrate this, we have tried to deploy the expert in different interpolated tracks with varying maximum distance between each consecutive gates. As the gates gets

closer to each other, the drone will have hard time maneuvering them at high speeds, and thus, for a collision-free flight, the velocity is required to be tightly constrained. On the other hand, letting the distances increase enables the drone to fly safely at higher velocities. For our expert, we deployed the drone at an average velocity of 4 m/s.

4.3.4 Trajectory refinement and rotation

In order to have a safer and smoother trajectory, the drone should be oriented to face the next gate whenever it is possible. Thus, adding some constrains on the orientation while solving the trajectory optimization problem was very useful. Constrains on the drone's orientation were conducted through a rotation matrix that was simply extracted from the gate orientation itself. The drone gradually rotates with its next trajectory, in order to finally pass through the gate with the required orientation (facing the gate). This in a way also guaranteed that most of the generated images will contain a gate, or a part of a gate.

```
def get_gate_facing_vector_from_quaternion(self, airsim_quat,
                                           scale=1):

   # convert gate quaternion to rotation matrix.
     q = np.array([airsim_quat.w_val, airsim_quat.x_val,
                   airsim_quat.y_val, airsim_quat.z_val],
                            dtype=np.float64)

     n = np.dot(q, q)
     if n < np.finfo(float).eps:
       return airsimdroneracinglab.Vector3r(0.0, 1.0, 0.0)
     q *= np.sqrt(2.0 / n)
     q = np.outer(q, q)

     rotation_matrix = np.array([[1.0 - q[2, 2] - q[3, 3],
     q[1, 2] - q[3, 0], q[1, 3] + q[2, 0], [q[1, 2] + q[3, 0],
      1.0 - q[1, 1] - q[3, 3], q[2, 3] - q[1, 0]],
      [q[1, 3] - q[2, 0], q[2, 3] + q[1, 0],
      1.0 - q[1, 1] - q[2, 2]]])

```

```
21        gate_facing_vector = rotation_matrix[:, 1]
22
23        return airsimdroneracinglab.Vector3r(
24        scale * gate_facing_vector[0],
25        scale * gate_facing_vector[1],
26        scale * gate_facing_vector[2])
```

4.3.5 Connection delay (Code – simulation lag)

After having the expert's model and the training environment ready, we faced an unexpected problem with the packet transmission between our code and the simulation. For illustration, the code may reach the trajectory generation from the 6th gate to the 7th, while the simulation is still in its initial position. This resulted in random errors in the trajectory generation, where the drone is set to move to wrong gates. To solve this problem we changed the logic flow of the trajectory generator. Instead of allowing the code to generate the trajectories after each other directly, we introduced a threshold for each trajectory generation. The threshold represents the maximum allowed Euclidean distance between the drone and the next gate. Whenever the distance between the drone and the next gate is less than that threshold, the code is allowed to generate the trajectory to the after-next gate. If the threshold is not yet broken by the drone, the code will do nothing but re-checking the drone position until it breaks the threshold.

4.3.6 Data recording

AirSim provides an option of recording the simulation as snapshots taken by the drone front RGB camera at high frequency. This is the images of the expert's dataset were generated. Pressing 'r' triggers the start button of the recording option in AirSim. However, we needed to automate this option as well, and 'Pynput' library just came in handy. We could easily automate a press of the 'r' button using commands like 'press()' and 'release()'. AirSim binaries provides an automatic recording for the drone velocities as well. The images and the velocity vectors are organized together by the time stamp at which each snapshot was taken.

4.3.7 Testing the Expert

The competition's first tier is basically a trajectory generation challenge. The drone is required to pass through gates of a certain track, where the position and orientation of each of these gates is given to the drone as gate poses. Based on that, the 1st tier is a perfect test for our expert's model. We deployed our expert to track provided by the competition for this tier and we recorded the results. The drone finished the challenge with zero collisions in 66.7 secs/lab, and with a maximum velocity of 5.3 m/s. This result puts us in the 6 place (out of 12) within the 1st tier leader-board. The test result was very good, and at this point we were comfortable that the expert is ready to generate satisfying actions.

4.4 Image Segmentation

Through the encoding stage, the resulted image gives a poor efficiency and there is a high error between the encoding and decoding images and gate pose due to the small differences between the used gates' color and the background's color; in our case the checkered gates with green and white colors are not different from the green environment. Furthermore, there are two ways to improve the efficiency of the encoder and decrease the error between the real image and the reconstructed image (decoded image). The first one is increasing the number of the training data (images) that requires a high computation cost and sometimes it is hard to get big data. The second is to make some pre-processing on the image before entering the encoder to give better efficiency. The pre-processing done for images is whether subtracting, segmenting the gate from the background, or changing the contrast of the image to improve the decoding.
Background removal techniques:

- Edge-based background removal: This technique detect a continuous edge path by detecting edges in the image. All items outside of the path are referred to as background.

- Foreground detection: The technique of foreground detection detects changes in

the image sequence. To extract the foreground from an image, the background subtraction method is performed.

– Machine-Learning Based Approach: To distinguish the foreground from the background, this technique employs a variety of machine learning algorithms.

The implemented techniques are background removal or image filtering using OpenCV and MediaPipe framework:

– Background removal using MOG MOG2 functions but they don't work for our case as they are used in videos as they highlight the moving objects in the video.

– Background removal using function selfie segmentation from the cvzone package worked well on a red gates as there is a good difference between the gate's color and the background's color. On the other hand, it didn't work on the checkered gate because of the high similarity between the gates and backgrounds color which make it hard to segment the gate from the environment. This model segment the objects that is less than 2 meter to the camera so it didn't work on far gates

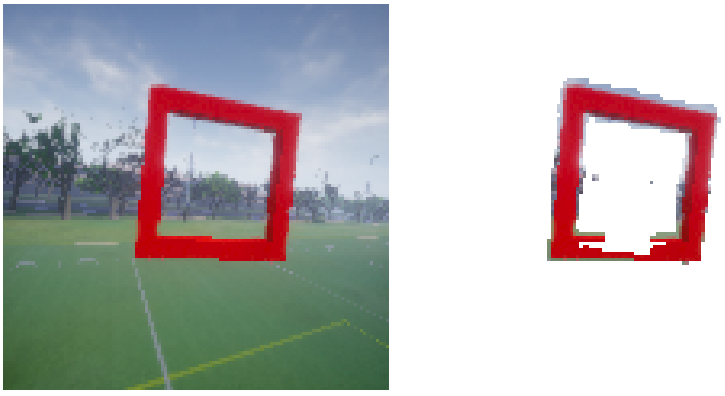


Figure 4.12: Image segmentation for red gate

– mage filter using filter2D function to sharpen the image so that the colors became more obvious but there is not obvious changes on the far checkered gates.

Figure 4.13: Image filter for far checkered gate

4.5 Control model training

For the behavioural cloning (BC) network a 4 dense layer network is used. The latent space vector size is 10 elements. The first hidden layer is of 256 units. This number is halved for the second hidden layer, and through the hidden layers, the number of units of each layer is the half of the previous layer's number of units. The output velocity vector components requires 4 separate units at the output layer. The activation function for the input and hidden layers is a "ReLU", while a linear activation function is used in the output layer.

```
dense0 = tf.keras.layers.Dense(units=256, activation='relu')
dense1 = tf.keras.layers.Dense(units=128, activation='relu')
dense2 = tf.keras.layers.Dense(units=64, activation='relu')
dense3 = tf.keras.layers.Dense(units=16, activation='relu')
dense4 = tf.keras.layers.Dense(units=4, activation='linear')
```

We trained the BC network with a 40,000 latent-space-vectors and their corresponding velocity vector. The number of epochs was set to 500 epochs, the batch size to 32 sample, and the learning rate to 10^{-2}. The training error at the end of training was approximately 3.04%, while the test error was approximately 3.67%. These results were very good, and so, we could proceed to the last step of this project. We are now ready for deploying our trained drone and see how it will perform.

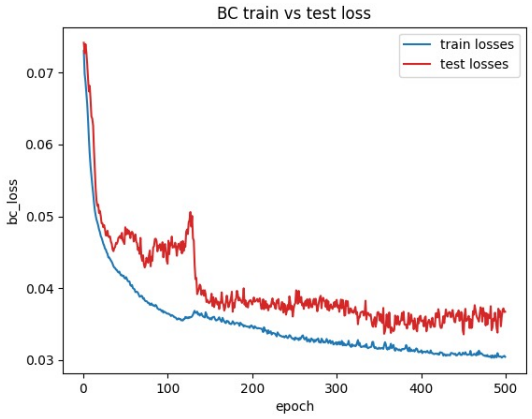


Figure 4.14: BC model training loss

Chapter 5

Results and Discussion

5.1 Overview of Performance measures

5.1.1 Evaluated Models

We have trained 2 models; Model A, and Model B with the main difference between them being the average, and max speed of the dataset used to train these models where model B was trained on a relatively faster dataset than Model A. Finally, we compare our fastest model (Model B) with the trained model from [26], which we named by: 'paper's Model'.

5.1.2 Performance measures

In this part, we're collecting multiple data during the drone's run-time and using it to compare the performance of different auto-pilot algorithms from different perspectives each decision iteration.

- **Velocity vector, and its speed magnitude**
  This is a simple one, we can use such information to draw a plot over the iterations and observe how the model behaves, does it just default to the max amount? Or is it smart enough to use different values for different situations?
- **Gate pass precision (Distance between the center of the gate and the drone, measured whenever the drone passes a gate)**
  How precise is the model? The more precise the model is does not essentially mean it is better; if we take the example of a circular track, the smaller the radius of drone's rotation, the less overall distance it will cover and essentially the more preferment it will be regardless of the gate precision; However, this value is good to keep track of in order to better understand the behavior of the model against different tracks.
- **Distance passed per iteration**
  This one is self explanatory, the bigger the distance passed the higher the velocity it decided to use but again, this is not a direct performance measure without

taking into account the path it took compared to the shortest path between the 2 gates; A model may be moving with a very high velocity but still make bad decisions which lead to longer pathways.

- **Time elapsed during each iteration**
  This is a direct measurement of how preferment the model is, how long did it take to come up with this iteration's decision? The lower this value is, the better.

- **Total distance passed**
  This one could be used to compare different model decision making performance, how smart was that model in choosing shorter pathways?

- **Total time elapsed**
  This is a direct performance comparison, the shorter the total time elapsed is, the faster the drone was (This takes into account many things, how fast the decisions were, how accurate, etc..).

- **Total fail recovery time (How much time did a drone take to recover from collisions)**
  Another direct performance comparison, an ideal model would have a number of 0.

- **Number of gates passed**
  This one is simple, it dictates whether or not the algorithm was able to finish the track without getting stuck at any point for a significant amount of time.

5.1.3 Track generation

The track itself was generated using pre-made square-ish shaped tracks with noise in the y direction (height); There are a total of 3 tracks, 8, 10, and 15 with the main difference between them being how spread apart each gate is from the consecutive one.

The main goal of these differences is to highlight how precise can the different models perform against different decision windows caused by the distance between each gate, where the smaller the distance is, the tighter the window the model will have to react to be able to track the next gate.

5.2 Results

There is a lot of data that was collected, but we're only previewing the ones with considerable differences between different models and tracks.

The decision time was calculated for each and every iteration with a value of 0.07436 seconds/decision it seemed to show nearly no change ($\sigma <= 0.020$) with the track, or the algorithm. Model B showed very little accuracy on some tracks so we tried suppressing them by capping their max speed to 5 instead of the original 8 and the acceleration to 20 instead of 30, hence and measurement that is suppressed will be denoted by (s). There are 2 types of tracks, circular and square-ish. The circular tracks we used to test Model A against the paper's model, while the square-ish ones were used to observe the effect of changing distance between gates as well as the performance differences between Model A and Model B. It is worth mentioning that all of the circular track measurements were simulated on a different device than the one used in the square-ish one which makes comparisons between the 2 tracks unreasonable, but we are still able to compare the relative values between measurements of the same track type.

5.2.1 Graphs

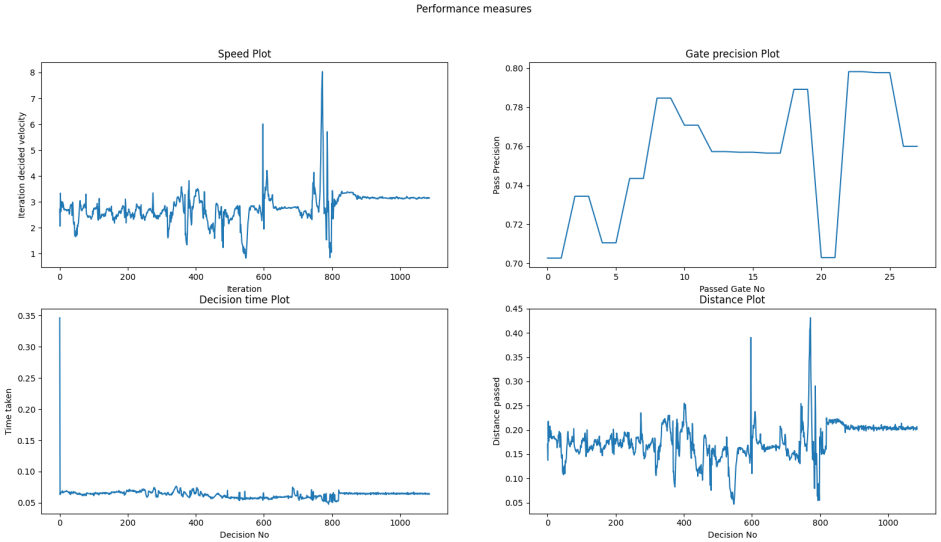


Figure 5.1: Model B on Track 8

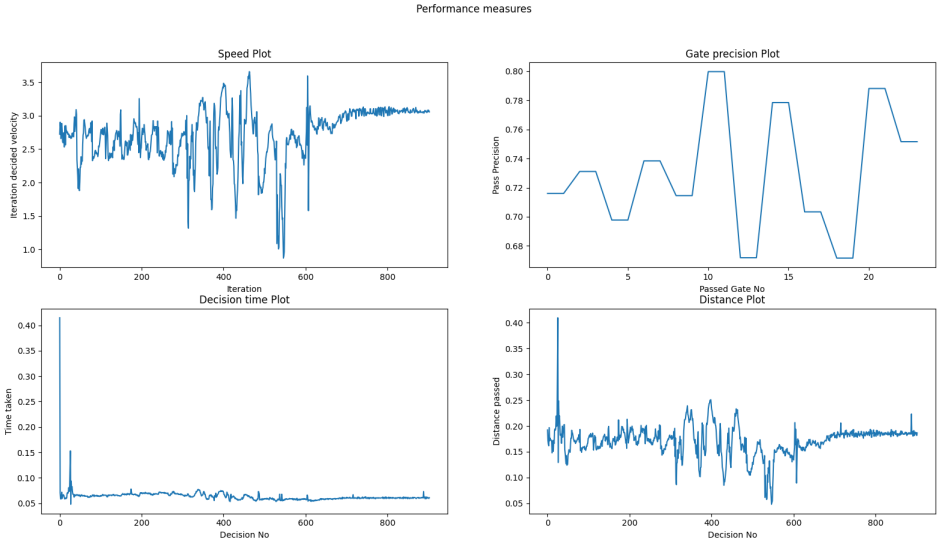


Figure 5.2: Model B(s) on Track 8

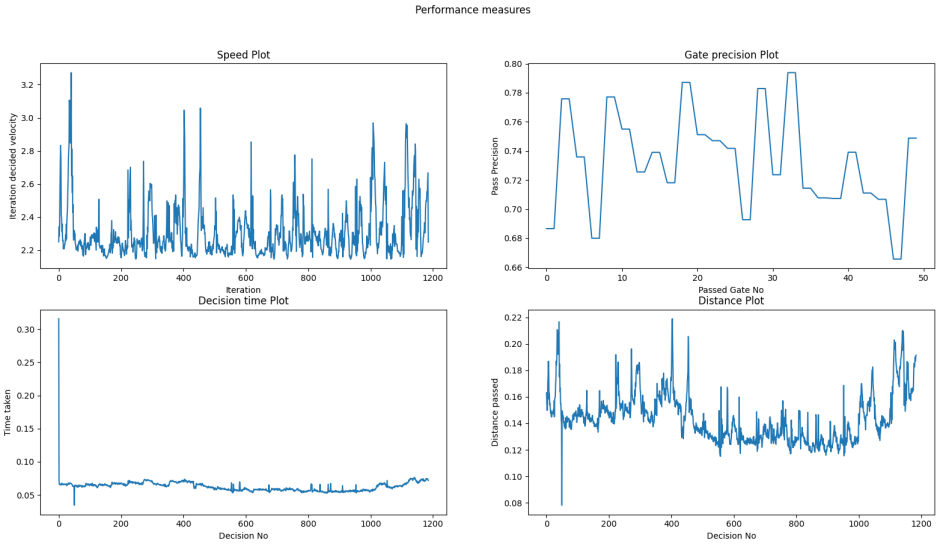


Figure 5.3: Model A on Track 8

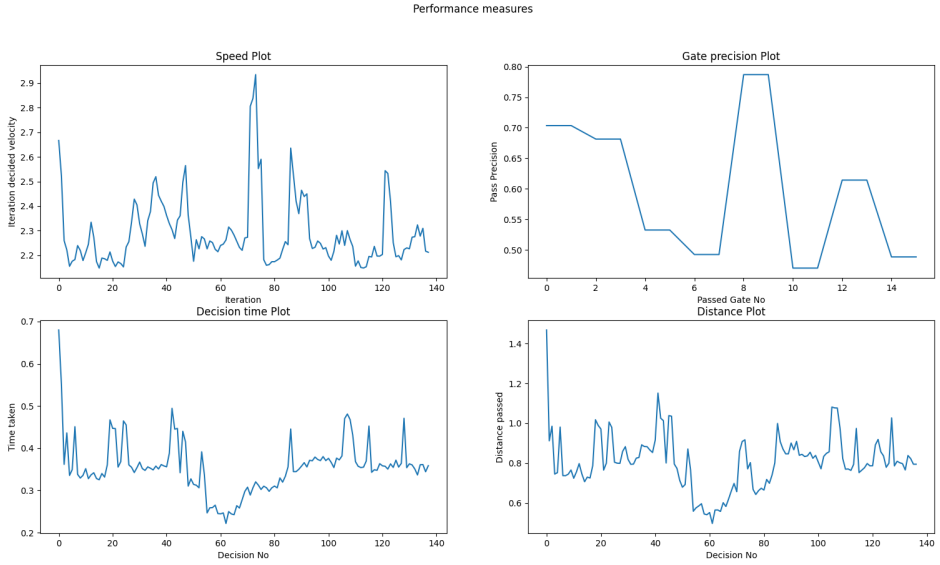


Figure 5.4: Model A on Circular Track R8

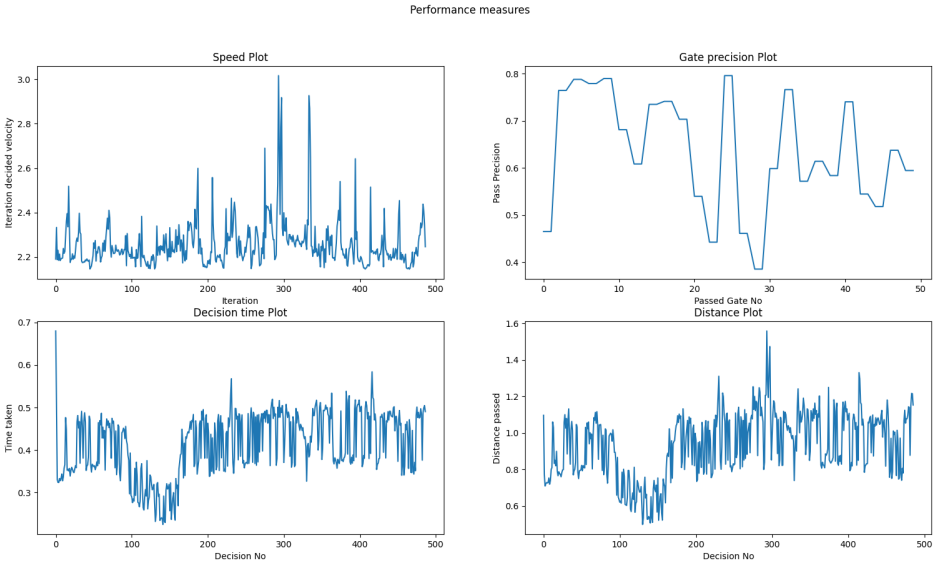


Figure 5.5: Model A on Circular Track R25

5.2.2 Tables

Model A vs Model B in track 8					
Model	Velocity	Gate precision	Total distance passed	Total fail Recovery time	Total number of gates passed
Model B	μ: 2.788 σ: 0.1511 Median: 2.80	μ: 0.678 σ: 0.0391 Median: 0.678	75.31	18.52	2/25
Model A	μ: 2.30 σ: 0.174 Median: 2.25	μ: 0.722 σ: 0.0432 Median: 0.720	181.89	0	25/25

Table 5.1: Model A vs Model B in track 8

Model B in tracks 8, 10, and 15				
Track	Velocity	Gate precision	Total distance passed	Total number of gates passed
8	μ: 2.788 σ: 0.1511 Median: 2.80	μ: 0.678 σ: 0.0391 Median: 0.678	75.31	2/25
10	μ: 2.709 σ: 0.359 Median: 2.789	μ: 0.724 σ: 0.045 Median: 0.7399	167.61	11/25
15	μ: 2.833 σ: 0.689 Median: 2.838	μ: 0.7359 σ: 0.0389 Median: 0.724	239.32	4/25

Table 5.2: Model B in tracks 8, 10, and 15

Model B, Model B(s) and Model A in track 8				
Model	Velocity	Fail Recovery time	Total time elapsed	Total number of gates passed
Model B	μ: 2.793 σ: 0.584 Median: 2.768	18.6	68.9	14/25
Model B(s)	μ: 2.722 σ: 0.403 Median: 2.788	6.6	57.2	12/25
Model A	μ: 2.306 σ: 0.161 Median: 2.254	0	74.24	25/25

Table 5.3: Model B, Model B(s) and Model A in track 8

Model A vs Paper's Model in circular track with radius 8				
Model	Velocity	Decision Time	Total distance passed	Total time elapsed
Model A	μ: 2.29 σ: 0.11 Median: 2.268	μ: 0.488 σ: 0.135 Median: 0.485	114.26	50.57
Paper	μ: 1.483 σ: 0.295 Median: 1.476	μ: 0.329 σ: 0.074 Median: 0.307	105.23	71.5

Table 5.4: Model A vs Paper's Model in circular track with radius 8

Model A vs Paper's Model in circular track with radius 25					
Model	Velocity	Decision Time	Total distance passed	Total time elapsed	Total gates passed
Model A	μ: 2.24 σ: 0.085 Median: 2.226	μ: 0.404 σ: 0.103 Median: 0.402	408.245	182.68	25/25
Paper	μ: 1.843 σ: 0.175 Median: 1.909	μ: 0.396 σ: 0.077 Median: 0.404	179.74	98.62	3/25

Table 5.5: Model A vs Paper's Model in circular track with radius 25

5.2.3 Comments

One very noticeable behaviour in in-complete graphs is the horizontal (nearly constant) values at the very end (150 iterations), this is due to the way the model behaves against collisions which is further described in the recommendations and future work section.

- **Square-ish tracks**
  We can get a sense of the stability of a model from observing the velocity curves in Figures 5.1, 5.2, and 5.3 where in Model A on track 8, the velocity changed from 2.2 to 4, compared Model B changing from 1, to 8, and from 1 to 3.5 after suppression. In Table 5.3, we can see a similar behaviour in the standard deviation(σ) where it jumps up from 0.161, to 0.4 in the suppressed Model B, and 0.584 in Model B. Model A appears to be much more stable numerically, and physically where Model B tends to "oscillate" instead of moving in a straight line, with increasing radius as the straight distance increases, which is supported by the standard deviation(σ) values of the velocities of Model B in Table 5.2 of 0.15, 0.359, and 0.689. This does not essentially make Model B useless, as it is still able to make much better decisions in shorter time periods compared to Model A, but it seems that it has a stability issue where it fails to stabilize itself onto a straight line. Another thing to note here on the same topic is how the velocity and acceleration suppression affected the stability of the model, reducing the overall "oscillatory" behaviour and the velocity standard deviation(σ), even though it was not enough to increase the number of gates passed as shown in Table 5.3 which suggests that the model itself is flawed and not just

that it is making decisions at faster speeds.
On a side note over all the experiments, the time taken to make a decision seemed to be relatively constant at around 0.075s.

- **Circular tracks**
  Even-though we used Model A in our comparison against the paper's model, it seemed to exhibit a much higher average velocity than the paper's with an increase of 154% in the R8 track and 121% in the R25 track in Tables 5.4, 5.5. One would assume this comes with an increased instability but this is not the case, Model A shows a standard deviation(σ) decrease to nearly 50% the value of that of the Paper's model over both circular tracks, these numbers translate to an overall performance improvement of 140% as dictated in the time elapsed section.
  Another note is on Table 5.5 of track R25, notice how only Model A was able to complete the track, actually in practice and out of a total of 5 attempts the Paper's model was able to finish track in only 1 time compared to 5 of Model A in R8 track and 0 times in R25 track. This essentially suggests higher precision for Model A compared to the Paper's.
  The last interesting behaviour is between Figures 5.4, 5.5 more specifically the gate precision graphs, ignoring the values they both seem to "look" similar which suggests that the dependency on the gate number overwhelms the effect of difference in the models. It does make sense that the gate number has an effect over the graph since the ultimate goal of the drone is passing through the the center of the gate after detecting it in both models, but it's interesting to see how dominant this effect is compared to the model differences.

Chapter 6

A Final word

6.1 Recommendations and Future work

There are four main points to consider in this part starting with the collisions, the trained models exhibited a behaviour where it would only have the goal of passing through gate, but be oblivious to the gate's walls and any other walls around it. Upon any collision, the model would not respond to it in anyway, instead try to to force his way in the same straight line and ultimately gets stuck unless it was lucky enough to only hit the very edge of the gate and is able to slip through. This behaviour is highlighted through the last 150 iterations in any of the graphs for incomplete runs, more specifically the speed graphs which is almost a horizontal line with nearly constant values. It is recommended for the model to be retrained against an expert that is aware of the walls, and tries to avoid collisions, or at least is able through feedback to recover from collisions.

The second point is the gate precision calculation algorithm, the way it is calculated at the moment is that we're computing the distance between the gate's center and the drone, at the moment of passing the gate, this method introduces an error of the value for the perpendicular distance from the point to the gate. Instead, the value should be the distance along the gate's plane between the 2 points.

The third one is the track generation method while the method itself is fine, AirSim does not allow the reassigning of the drone's orientation in a simple way, and thus if the first gate in any track is rotated enough, the drone itself may not be able to see it clearly and thus not detect it in the first place. A perfect track generation method in this case would make it so that the very first gate is place in-front of the (0, 0, 0) position with less than 45% rotation in any of the axis.

Another one is the area of effect of the gate detection model, which is very small causing the drone to lose track of relatively average distant objects, and thus limiting the number of gates it can detect. In addition to that, the perception unit could have performed better if the input images were manipulated in a way to make the gates more clear and different from the background prior to its encoding.

Also there is the decision time measurements in the circular tracks, it is highly encouraged to redo the measurements on a different machine and confirm or deny

the effect of the machine's physical capabilities on the decision time behaviour and standard deviation (Not the mean, since it is obvious that a stronger machine will make decisions faster).

The last one is the gate precision, which should be further explored by putting the model against different track shapes, both simpler and more complex. Also we can test it against other models that operates in a different manner in order to put a qualitative weight to the effect of both parameters on the gate precision. We could argue that a higher precision model does not entail higher performance, but it definitely has a direct relation to safety where a more precise model will be less prone to collisions with the gates' walls.

6.2 Conclusion

To sum up, intelligent racing drones' era has started. Drones are required to detect the obstacles they need to avoid, and they are required to learn how to avoid them as well. In this project we were able to develop a CM-VAE model that could extract the obstacle's (the gate) most important features in the form of a latent space vector. This vector is passed to the control model, which we chose to be a BC model. The training dataset was created through recording an expert flight in a similar environment, where both images of what the drone sees, and the drone's corresponding velocity vectors are saved. The trained encoder converted these images into latent space vector, and the BC network was trained afterwards. Our perception model showed very high performance on relatively close gates, while the performance dropped as the date distance from the drone increases. This is mainly because the gates' texture and colors were very close to the background's. Our control model showed success, as it performed better in both velocity and stability than other models in literature. However, there is yet a lot of work to do regrading the flight safety and speed; the drone should be able to travel faster without losing its stability. This in addition to increasing the perception range can result in a huge step in the drone racing technologies.

References

[1] “Unmanned Aerial Vehicles (uavs): A survey on civil applications and key research challenges,” IEEE Xplore. [Online].

[2] I. B. and T. Watts, “Meaning-less human control: Lessons from Air Defence Systems for lethal autonomous weapons,” Drone Wars UK, 27-Jul-2021. [Online].

[3] Jia, Y., Guo, L., Wang, X. (2018). Real-time control systems. Transportation Cyber-Physical Systems, 81-113.

[4] R. Madaan and A. Kapoor, “Game of drones competition gears up to aid Autonomous Systems Research,” Microsoft Research, 24-Dec-2019. [Online].

[5] R. Merkert and J. Bushell, "Managing the drone revolution: A systematic literature review into the current use of airborne drones and future strategic directions for their effective control", Journal of Air Transport Management, vol. 89, p. 101929, 2020. Available: 10.1016/j.jairtraman.2020.101929

[6] “UAV applications: Introduction | springerlink.” [Online].

[7] Bonin-Font, F., Ortiz, A., Oliver, G. (2008). Visual Navigation for Mobile Robots: A Survey. Journal Of Intelligent And Robotic Systems, 53(3), 263-296.

[8] Y. Lu, Z. Xue, G. Xia and L. Zhang, "A survey on vision-based UAV navigation", Geo-spatial Information Science, vol. 21, no. 1, pp. 21-32, 2018.

[9] Al-Kaff, A., Q. Meng, D. Martín, A. de la Escalera, and J.M. Armingol. 2016. “Monocular Vision-based Obstacle Detection/Avoidance for Unmanned Aerial Vehicles.”Intelligent Vehicles Symposium, IEEE, Gothenburg,Sweden, June 19–22.

[10] R. Bonatti, R. Madaan, V. Vineet, S. Scherer and A. Kapoor, "Learning Visuomotor Policies for Aerial Navigation Using Cross-Modal Representations", 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2020.

[11] Yathirajam, Bharadwaja. (2017). Obstacle Avoidance for Unmanned Air Vehicles Using Monocular SLAM with Chain Based Path Planning in GPS Denied Environments.

[12] A. Koubâa, A. Azar, and M. Abdelkader, "Unmanned aerial systems: autonomy, cognition, and control," in Unmanned aerial systems: Theoretical Foundation and applications, London i 3 pozostałe: Academic Press is an imprint of Elsevier, 2021.

[13] Jung, S., Hwang, S., Shin, H., Shim, D. (2018). Perception, Guidance, and Navigation for Indoor Autonomous Drone Racing Using Deep Learning. IEEE Robotics And Automation Letters, 3(3), 2539-2544.

[14] J. Zhang, "AI based algorithms of path planning, navigation and control for Mobile Ground Robots and uavs," arXiv.org, 03-Oct-2021. [Online].

[15] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Mathematical Problems in Engineering 2015.

[16] Y. Zhao, Z. Zheng and Y. Liu, "Survey on computational-intelligence-based UAV path planning", Knowledge-Based Systems, vol. 158, pp. 54-64, 2018.

[17] "UAV trajectory planning for static and dynamic environments." [Online].

[18] "(PDF) spline based path planning for unmanned Air Vehicles." [Online].

[19] C. Yan and X. Xiang, "A Path Planning Algorithm for UAV Based on Improved Q-Learning", 2018 2nd International Conference on Robotics and Automation Sciences (ICRAS), 2018.

[20] "Deep-Sarsa Based Multi-UAV Path Planning and Obstacle Avoidance in a Dynamic Environment." [Online].

[21] "Unmanned Aerial Vehicle (UAV) market size, growth, analysis," Allied Market Research. [Online].

[22] "Drone flight controller system market size, analysis, outlook 2021-2030," Allied Market Research. [Online].

[23] Ebeid, Emad Samuel Malki & Skriver, Martin & Jin, Jie. (2017). A Survey on Open-Source Flight Control Platforms of Unmanned Aerial Vehicle.

[24] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," arXiv.org, 29-Dec-2016. [Online]. Available: https://arxiv.org/abs/1512.02325.

[25] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for Mobile Vision Applications," arXiv.org, 17-Apr-2017. [Online].Available: https://arxiv.org/abs/1704.04861.

[26] R. Bonatti, R. Madaan, V. Vineet, S. Scherer, and A. Kapoor, "Learning visuomotor policies for aerial navigation using cross-modal representations," arXiv.org, 08-Mar-2020. [Online].

[27] A. Spurr, J. Song, S. Park, and O. Hilliges, "Cross-modal deep variational hand pose estimation," arXiv.org, 30-Mar-2018. [Online]. Available: https://arxiv.org/abs/1803.11404.

[28] "Long-term planning with deep reinforcement learning on autonomous drones," IEEE Xplore. [Online]. Available: https://ieeexplore.ieee.org/document/9259811.

[29] T. Osa, J. Pajarinen, P. Abbeel et al., "An algorithmic perspective on imitation learning," Foundations and Trends R in Robotics, vol. 7, no. 1-2, pp. 1–179, 2018.

[30] L. Rozo, J. Silvério, S. Calinon and D. Caldwell, "Learning Controllers for Reactive and Proactive Behaviors in Human–Robot Collaboration", Frontiers in Robotics and AI, vol. 3, 2016.

[31] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011, pp. 2520–2525.

[32] T. Flash and N. Hogan, "The coordination of arm movements: An experimentally confirmed mathematical model," The Journal of Neuroscience, vol. 5, pp. 1688–1703, 1985

[33] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," IEEE Robotics and Automation Letters, vol. 3, no. 2, pp. 1088–1095, 2018

[34] purr, J. Song, S. Park, and O. Hilliges, "Cross-modal deep variational hand pose estimation," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 89–98.

[35] M. W. Diederik Kingma, "Autoencoding variational bayes," in ICLR, 2014.