Flying Through Gates using Deep Learning

Ahmad Kourani, Joyce Abi Saleh, Ali Shibli, Majd Wardeh

***Abstract*— We address the problem of flying a quadrotor through narrow gates using onboard sensors and computing resources. Our system utilizes a Neural Network that takes as an input the image of the gate in addition to the velocity and acceleration of the drone, and outputs controller-based commands, represented by the Euler angles (raw, pitch, yaw) that the drone must follow to traverse the gate safely. This path generation algorithm computes the relative position $\{x,y,z\}$ of the gate in space and performs the proper predictions for the next euler angles to follow. This path must keep the drone facing the gate and minimize the collision probability. Unlike previous works that assume that the pose of the gate is known as a priori or use traditional computer vision algorithms or motion capture systems, our proposed method aims to learn to pass through feasible gates using a data-based neural network approach.**

I. Introduction

Training drones to perform autonomous tasks has become a booming area in research. Being autonomous means dealing with tasks of increasing complexity that emerge from the new applications of these flying machines. Flying through gates is one task encountered in many special and useful applications of drones, yet it is one of the most challenging and most important tasks in the field [1]. Nowadays, drones are finding increasing applications in the safety and security sector, thus getting a drone to fly through a gate can help explore areas not easily accessible to humans. It can also help in preventing casualties in emergencies or high risk situations for example in unstable buildings or during fires or explosions [2].

In this work, we aim to tackle the challenge of a drone traversing through a gate of unknown position using the inputs of the on-board camera. A simple structured indoor setup for this problem is shown in Fig. 1. To clarify the problem at hand, we introduce the following brief description, which is also summarized in Fig. 2. The gate traverse mission consist mainly of four step:

1) Pose (i.e., position and orientation) estimation of the gate and the drone: knowing the location of both the drone and the gate is the prerequisite knowledge for any gate traverse algorithm. To acquire the drone and gate poses, researchers either refer to an indoor Motion Capture System (MCS)[1] to deliver accurate feedback [4], or rely on visual odometry to estimate the relative drone-gate position and orientation [3].
2) Generating the traverse trajectory: it is the trajectory starting just before entering the gate to just after exiting it. The conventional approach is to interpret the problem as a projectile mathematical model due to the under-actuation property of the quadrotor[2] [3], [1].
3) Generating the approach trajectory: to meet the requirements of the traverse trajectory, which are an initial lunch point, velocity, and angle, the quadrotor must follow a trajectory starting from its initial position that respects its dynamics and any other needed constraints such as maintaining a field of view over the gate[3]. Different methods are applied for the approach trajectory planning task, including traditional or learning based

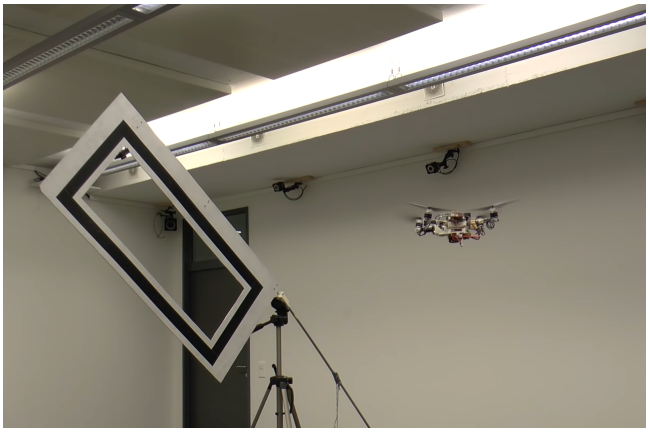

Fig. 1. The tackled problem: a quadrotor tries to traverse a gate using a front-facing camera and its onboard sensors and computing resources [3].

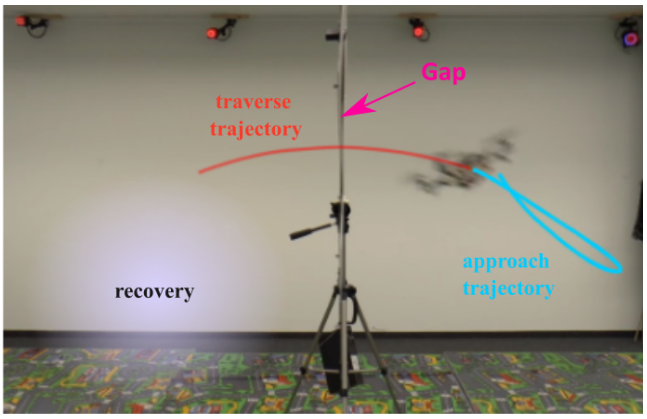

Fig. 2. The quadrotor in the gate traverse mission, including the following tasks: approach trajectory, traverse trajectory, and recovery [3].

[1]An indoor motion capture system uses multiple fixed infrared cameras and computer vision algorithms to detect markers fixed on the targeted object, and estimates its position and orientation relative to a predefined frame.

[2]An under-actuated system is one that all its states cannot be controller simultaneously.

[3]Loosing the gate visibility will result in loosing pose estimation of the gate if done using visual odometry, thus failing to complete the algorithm.

methods.

4) Recovering safely after traversing the gate: the main goal of the recovery step is to restore the drone to a stable and safe state while getting ready to receive new missions. This step is not the main scope of research in this field, and will not be elaborated on in this work.

All these prementioned high level tasks require a low level control algorithm capable of stabilizing the drone and following assigned trajectories. This is either accomplished though control theory, or through learning based algorithms.

Challenges that arise specifically in flying a drone through a narrow gate include detecting the gate's pose relative to the drone, relying on the drone's onboard sensors and generating the trajectory to traverse the gate, in addition to physical damage during algorithm development.

Achieving the gate traverse mission autonomously is not yet fully addressed in literature. Furthermore, many attempts have been made to incorporate machine learning based techniques to address this problem due to advantages of reducing computational power requirements and facilitate the design process. In the following, we aim to develop a novel learning-based approach that addresses the state estimation and path planning problems of the quadrotor gate traverse mission.

Falanga *et.al.* tackled the gate traverse problem using onboard sensors and computer [3]. A monocular camera was utilized to detect a rectangular shape gate and estimate its pose. The gate's pose is used to estimate the relative pose of the drone with the help of an Inertial Measurement Unit (IMU) sensor. Once both the poses of the gate and the drone are estimated, a traversing trajectory and an approaching trajectory are generated using optimization methods. However, the authors used a traditional computer vision algorithm to detect a rectangular-shaped gate decorated with a black rectangle on a white plane. This made the detection and localization of the gate a trivial task. In [5], the authors use machine learning methods to generate an approach trajectory of the drone to traverse the gap, but feedback was considered available from a MCS, which makes to mission non-autonomous.

In this work, we intend to use a supervised learning algorithm in order to train a Neural Network (NN) to perform the task of safely guiding the quadrotor through the gate. The inputs to the network are an image taken by an onboard front-facing camera, and the current linear and angular speed and acceleration of the quadrotor. The output of the network is a series a N waypoints consisting of the $\{x, y, z\}$ coordinates in addition to a desired yaw angle, describing a short-term trajectory that leads the quadrotor to traverse the gate.

Our main contribution lies in creating a gate traverse-trajectory generator system relying on a Neural Networks (NN) that identify the presence and pose of the gate in a first phase, followed by the drone's navigation coordinates without relying on any additional vision or control algorithm. Fig. 3 depicts the proposed system's pipeline in form of two neural network. The inputs are a picture of the gate taken from the drone's front facing camera in addition to the angular and linear velocities and accelerations provided by the onboard Inertial Measurement Unit (IMU). The network then outputs the short-term path consisting of $\{x, y, z\}$ coordinates and a yaw angle used to navigate the drone. Using sensor and camera data, the visual inertial odometry (VIO) algorithm estimates the drone's pose relative to the world frame. The waypoints from the short-term path in addition to the drone's current position is then sent to a controller which will output motor voltages in order to navigate the drone accurately through the gate.

II. Related Work

A. Seminal and State-of-the-Art Work

1) Introducing the gate traverse problem: The problem of quadrotor's aggressive maneuvers was initially defined by Mellinger *et.al.* [6], which extends to the narrow gate traverse maneuver. The proposed methodology consisted of generating a traverse trajectory and an optimized feasible approach trajectory, relying on dynamic modeling, controller switching and trial and error. The traverse trajectory was based on a series of basic trivial actions while the quadrotor switches from one control mode to another (position control & attitude control). Trajectory planning was performed onboard and optimized to respect the quadrotor dynamics. The problem of pose estimation and autonomous flight was not addressed in this work, as feedback was available from a Motion Capture System (MCS). Thus quadrotor heading and computational efficiency was not a concern.

2) Path planning and control with Machine Learning: The traditional work in autonomous navigation relied on the perception-planning-control approach [7]. These conventional approaches were memory and computationally expensive, especially when applied to aggressive dynamic maneuvers, having to apply expensive optimization techniques in real time. However, recently, there has been research on mapping (or imitating) the motion planning and controller function by neural networks [1]. The approach proposed utilizes 2 fully connected networks: one for motions planning and one for the controller. They relied on 2 datasets for each network: the first one uses training data as random trajectory plans generated from onboard optimizer, and the second network uses data generated from the controller. The overall (policy) network outputs, given the specific position, velocity, and acceleration of the drone, the corresponding control commands from Roll, Pitch, and Thrust. The input to the network includes as well the position of the gap, which is retrieved from the motion capture system. Finally, the network parameters were fine tuned by reinforcement learning, using Microsoft-AirSim simulator. However, this approach assumed known quadrotor and gate position, thus excluding any uncertainty in the quadrotor-gate relative position, while not taking the quadrotor heading (yaw angle) into account. The quadrotor heading is directly related to the onboard camera's field of view, thus not respecting proper orientation will result in loosing the track of the gate in real-world vision based missions.

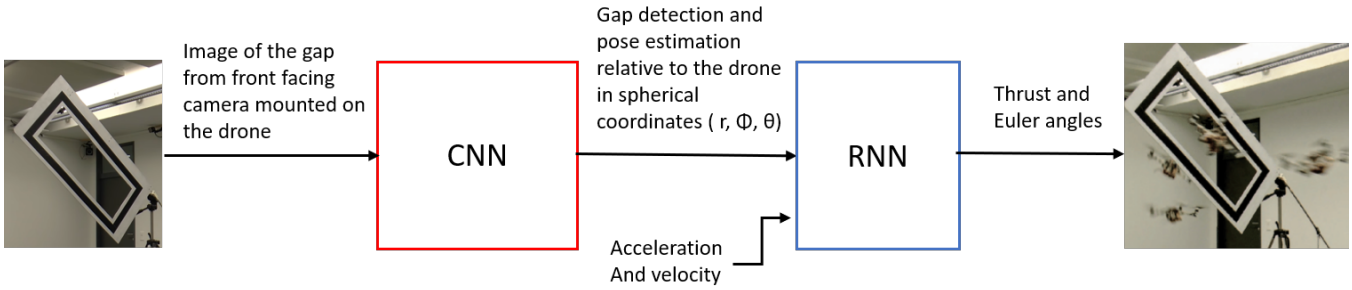

Fig. 3. The pipeline of the proposed methodology.

B. Navigation Using Deep Learning

In this subsection, we present papers that used deep learning to enable drones and Cars to navigate using a monocular camera. These works corroborate that Convolutional Neural Networks can directly control vehicles to achieve the navigation task. In [8] authors developed a CNN that safely steers a drone in the streets of a city. The CNN had two outputs, the first one (regression) steers the drone (controls the yaw angle) and the second output (classification) predicts the probability of collision (which controls the linear speed of the drone). In [9], Yang et al. introduced a navigation system that was capable of steering a UAV through corridors from a monocular camera, using two consecutive NN, going form RGB images as input, to predicted depth as first network output, then to navigation path as the second's. Their results demonstrated a 20% increase in accuracy compared with direct prediction.

In [10], An end-to-end learning approach to gradual path planning for autonomous self-driven cars was proposed. The algorithm takes one picture from onboard camera and generates a short term path consisting of a sequence of 5 steering angle commands.

Xu et al [11], propose an end-to-end model which given the agent's present state, predicts its future motion path for autonomous driving vehicles. The input of the network consists of an image, in addition to the vehicles previous state signals. The network consists of an LTSM which uses the sequence of prior events to predict the ideal next motion action and a CNN is used to analyze and extract visual cues. An LTSM is also used to fuse past and current states into a single state which will contain all past sensor information. The output of the system consists of a probabilistic distribution over different possible actions. Our system adopts a slightly similar approach however our NN outputs the exact coordinates needed for navigation rather than a probability for a set of actions.

C. Deep Learning in Autonomous Drone Racing

In Autonomous Drone Racing competitions a drone is required to traverse several gates autonomously as quickly as possible using onboard resources. Jung *et al.* published the first paper that presents an end-to-end navigation system for Autonomous Drone Racing [12]. They demonstrated that color-based gate detection fails in situations when light conditions are changed or when more than one gate is captured overlapped. For these reasons, they presented a CNN approach for gate detection using The Single Shut Detection algorithm [13]. The CNN is based on AlexNet architecture but modified to speed up the inference time. Velocity commands were computed to align the optical center of the images with the center of the gates to control the drone.

Muller *et al.* proposed an end-to-end CNN based system that pilots a UAV to race in a photo-realistic simulator [14]. The CNN took raw images from the simulator as input and outputs thrust, and angular rates command for the UAV, taking into account its dynamics. In the training stage, data was recorded from humans, with different levels of experience, piloting the aircraft after training on the racing tracks. The authors proved that their system is capable of beating mid-level humans in tracks that have never been seen before nether by the CNN nor the human pilot in the simulator. Since the proposed system was end-to-end, the control commands were low-level, meaning that it will require a high frame rate throughput computer, and in case the dynamical model of the drone is changed, the CNN must be retrained.

In [15], Kaufmann et al. answered the question: "Can we learn high-level commands (*e.g.,* waypoints) in order to take advantage of existing optimal-control algorithms for UAVs in drone racing?" by presenting an approach that combines the learning of CNN in perception with the path-planning and a model predictive control (MPC) to fly the aircraft. The deep network took a 320×240 RGB image, capturing a gate, and output two measurements; the pose of the next gate relative to the UAV and the variance of this prediction. After representing these outputs in the world frame, The prediction of the captured gate was filtered using an Extended Kalman Filter. Low-level waypoints were then generated using traditional methods and an MPC controller was used to follow the path.

In [16], Kaufmann et al. worked on traversing the gates with optimal trajectories. They trained a CNN on images taken from a quadrotor flying in a racing track with predefined positions of the gates and with an optimal global planned trajectory traversing all of them. The CNN outputs a tuple $\{\vec{x}, v\}$. $\vec{x}$ is a two-dimensional vector that encodes the direction to the new goal (coming gate), and v is a normalized desired speed to approach it. The system took the CNN

output and back-projected $\vec{x}$ along the camera projection ray to get the 3d point in the local camera frame. Once the 3d point in the local frame of the UAV is computed, an interpolating trajectory going from the current pose of the drone to the goal point is computed. In [15] and [16] traditional path-planning algorithms were used and there was no learning component it them.

III. Methodology

A. Preliminary proposal

In this work, we aim to design a neural network that will mimic the role of the autonomous path planner of a quadrotor for the gate traverse mission. This is achieved by estimating the pose of a gate in 3D space, then generating a short path to approach and traverse the gate. The inputs to this network are the quadrotor states, consisting of position, velocity, and acceleration,

The preliminary architechture consist of two main sub-networks:

- a convolutional neural network (CNN) with the goal of detecting the location of the gate with respect to the drone, given different shapes of the gate. The output of this sub-network is the pose estimate of the gate in 3D space, which is then used as input to the second sub-network
- a recurrent neural network (RNN), that takes the state of the drone in addition to the pose of the gate at each time step and outputs a finite series of recommended trajectory. The reason choosing this network tipe is that the data is a time series. This network will be trained on two sets of data:

B. Data Description

The first sub-network (CNN) will be trained on pictures of various gate shapes taken from different angles and locations. The gate's pose vector in the world frame is defined as: ${}^{w}X_{g,i} = [x_{g,i}, y_{g,i}, z_{g,i}, \eta_{1,g,i}, \eta_{2,g,i}, \eta_{3,g,i}]$, where $x_{g,i}$, $y_{g,i}$ and $z_{g,i}$ are the coordinates of the gate in a predefined inertial frame in $SO(3)$, and $\eta_{1,g,i}$, $\eta_{2,g,i}$ and $\eta_{3,g,i}$ are the Euler angles defining the orientation of the gate.

The second sub-network (RNN) will be trained simultaneously on two sets of input data:

1) N points representing different relative poses of the gate in the camera fixed spherical frame of reference ${}^{s}X_{g,i} = (r, \theta, \phi)$, $i = 1, ..., N$,
2) M points representing different initial states of the quadrotor ${}^{w}\dot{X}_{q,j} = [\dot{x}_{q,j}, \dot{y}_{q,j}, \dot{z}_{q,j}, p_{q,j}, q_{q,j}, r_{q,j}]^T$, $j = 1, ..., M$, representing its linear and angular velocities estimated using the onboard IMU.

These input can be fed to traditional path planning algorithms for quadrotors that rely on optimization techniques to generate the ground truth output data, defined as an optimal path $P({}^{s}X_{g,i}, {}^{w}_{q,j})$ and associated to the i^{th} gate point and j^{th} quadrotor state point.

Fig. 4. Pictures of the gate in different positions (a), (b), (c) and with no gate (d).

IV. Experiments and Results

A. Data Collection

To train the model, we referred to our own generated data, as the problem is hardware and software dependent, and few work has been done on the paper topic so far. The data generation process incorporated knowledge from different disciplines, including virtual environment, frame of reference rotations and transformations, path planning, modeling, and control.

1) Data Collection for the Gate detection Network: In order to train the convolutional neural network to predict the 3D pose of a given gate relative to the drone, the vision data was simulated. The virtual environment was based on flight goggles simulator's [18] 3D reconstruction of an abandoned factory with gates built in Unity 3D. To generate the data, a camera was placed as a child to a gate and programmed to move to a random position within a specified range along each of the x-, y- and z- axes as to make sure the gate remains visible. In addition, the rotation was also set so that depending on the camera's position, it is always facing the fixed gate. At each iteration where the random position of the camera is updated, a numbered picture of the gate taken from that camera is saved along with the camera's x-, y- and z- positions, the rotation quaternion defining its orientation, and its x-, y- and z-axes Euler angles. Next, given the relative position of the gate with respect to the camera frame ${}^{C}X_{G} = [x_r, y_r, z_r]^T$ is computed, and then converted to spherical coordinates by applying the following transformation:

$$\begin{aligned} r &= \sqrt{x_r^2 + y_r^2 + z_r^2}, \\ \theta &= \text{atan}(\frac{z_r}{x_r^2 + y_r^2}), \\ \phi &= \text{atan}(\frac{y_r}{x_r}), \end{aligned} \tag{1}$$

Furthermore, several gate-free pictures of the abandoned factory were generated in order to train the CNN on detecting the presence of a gate. Samples of the generated pictures are presented in Fig. 4.

2) Data Collections for the Trajectory Generation Network: We applied polynomial based trajectory generation

TABLE I

QUALITATIVE COMPARISON OF CHALLENGES ADDRESSED BY PREVIOUS WORK

#	Challenges	1, gate detection	2, state estimation	3, path planning	4, control	data-driven approach?
1	Seminal [6]	external	external	optimization	control theory	NO
2	S-o-t-A 1 [1]	external	onboard	optimization	control theory	NO
3	S-o-t-A 2 [17]	onbaord	onboard	optimization	control theory	NO
4	S-o-t-A 3 [3]	external	external	neural network	neural network	YES
5	This work	onboard, CNN	onboard	neural network	control theory	YES

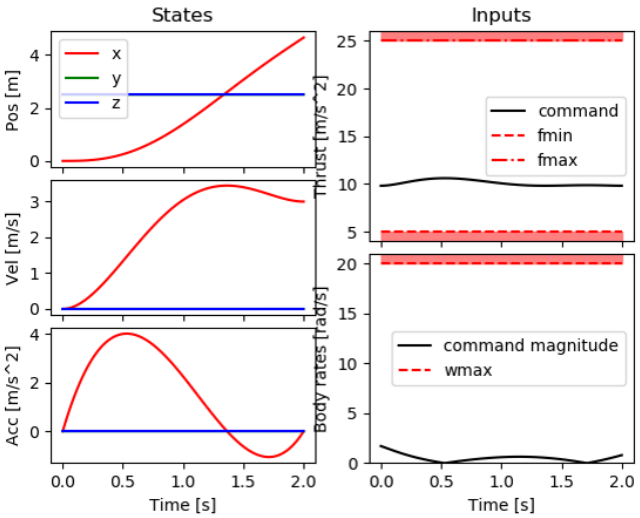

Fig. 5. A sample of the trajectory data set [19].

algorithm [19], along with the gate traverse optimization algorithm [3] to generate the training data for the RNN network. The trajectories were chosen based on the criteria that minimizes the line-of-sight angle with the gate, as this will reduce the gate's pose estimation error. The trajectory generation procedure can be described by the following pseudo-code:

1) Define the pose of the gate,
2) Define a cuboid space at one side of the gate (i.e. a $5m$ side),
3) Discretize the space to n_p points along each axis,
4) Define an orientation space for the drone's camera within reasonable bounds (i.e. $\pm\pi/3$),
5) Discretize the camera's orientation space into n_o points,
6) Define the camera's field of view (i.e. $\pi/2$),
7) For each point in the defined spaces:
   a) If the gate is visible to the camera:
   b) Calculate the required end point parameters that allows the drone to traverse the gate [3],
   c) Generate a trajectory (x, y, z components) that meats the predefined starting and end points characteristics [19],
   d) Generate a heading angle that minimizes the line-of-sight angle between the camera frame and the center of the gate [3].

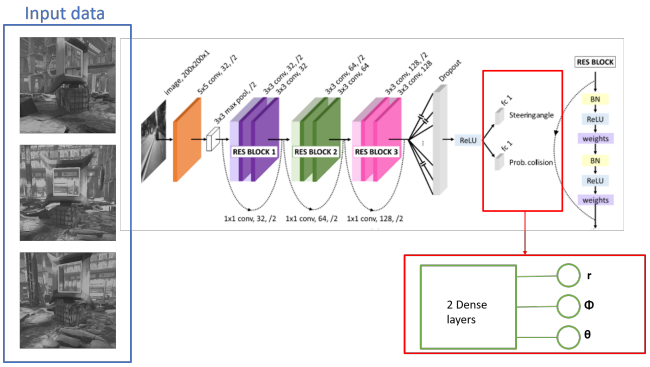

Fig. 6. *DroNet* modified architecture

B. Approaches

C. Convolutional Neural Networks for Pose Estimation

In order to estimate the pose of the gate from a single 2D image, Convolutional Neural Networks are used. We propose two CNN architectures, one based on *DroNet* and the other is based on Inception.

1) Dronet-based Model: Similar to Kaufman's [15] approach, the CNN used is based on the Dronet [8] architecture which consists of 10 convolutional layers and 3 residual blocks. The input to the network is a 200 by 200 gray scale image. The convolutional layers are followed by two fully connected layers, one which outputs the steering angle required to avoid the obstacle and the other indicating the probability of collision. In our work, we keep the residual blocks and replace the last layer with two dense layers followed by an output layer with three outputs corresponding to the spherical coordinates $\{r, \theta, \phi\}$ as seen in Fig. 6. Even though the input data that DroNet is trained on is quite different from our own (pictures of streets with obstacles consisting of cars, buildings, humans...), the network is trained to detect and localize obstacles in order to avoid them which is similar to our network's requirements. The network is retrained from scratch on the generated data from Unity. The Relu activation function is used for the two dense layers and a linear activation function for the last layer since this consists of a regression problem.

2) Inception-based Model: The second proposed CNN model is motivated by *PoseNet* [20]. *PoseNet* is a convolutional neural network that estimates the 6-DOF pose of a camera in a scene from an RGB image. The papers' authors showed that CNNs designed for classification tasks can transfer well to learn regression tasks. They used Inception

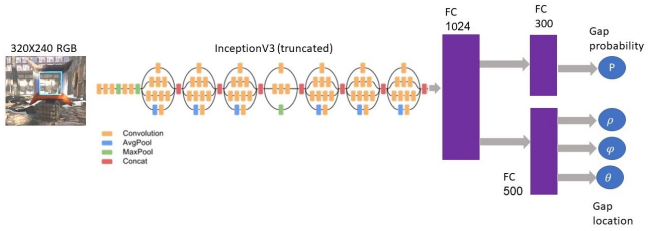

Fig. 7. The architecture of the Inception-based proposed model.

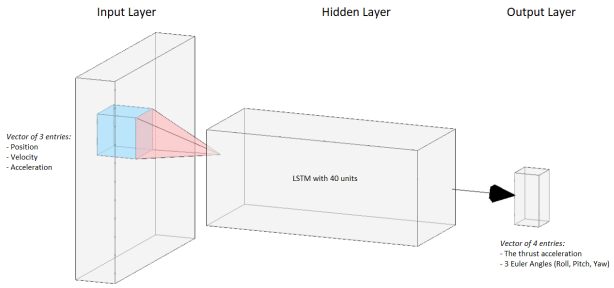

Fig. 8. RNN Design

V1 [21] to predict the pose of the camera represented in X, Y, Z coordinates as well as a quaternion for the orientation. Inspired by *PoseNet*, We developed a model based on Inception V3 [22] that takes a 320×240 RGB image and output the probability of gate in that image and its estimated position in 3D.

Figure 7 shows the architecture of the proposed model. We truncated the Inception V3 network by removing the deeper conv layers since the throughput of the proposed network must be high, and also, deeper convolutional layers are trained to detect task-related features. The feature maps of the resulted CNN are flattened and connected to a 1024-neuron fully connected layer with ReLU activation function. This layer is connected to two dense layers; The first one with 300, neurons followed by one neuron with a Sigmoid activation to predict the gate probability. The other has 500 neurons connected to three linear output neurons to predict the position in Spherical coordinates.

D. Recurrent Network Block For Path Planning

Learning a path for the drone to follow can be formulated as a time series problem. Future points are dependent on two chronological sets of data: the current states and the previous states. For our purposes, training was done given one previous state and the current state of the drone, as described in the data collection section. Let the commands to the drone at time instance t be the vector $U = [F_t, \eta_1, \eta_2, \eta_3]^T$, with its components representing the drone's total thrust, roll angle, pitch angle, and yaw angle commands respectively. The output of the network is one future command step ahead, to be input to the controller. Thus, the problem can be thought of as a regression problem, which we are fitting new data to a curve similar to previous instances.

The network is composed of three main layers:

- An *InputLayer* having position of the gate in 3D (retrieved from the CNN), and velocity and acceleration of the drone (retrieved from the onboard sensors of the drone).
- A hidden *LSTMlayer* having 40 RNN Units to learn the time series pattern.
- An output *DenseLayer* outputting the target thrust and the three Euler angles (η_1, η_2, η_3) for the drone's controller to follow.

The design can be seen in Fig. 8. The loss functions used in determining loss rate is the Mean Square Error (MSE) metric. At instance i, given the ground truth state X_i, and predicted state $\hat{X}_i$, the MSE is calculated as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(U_i - \hat{U}_i)^T(U_i - \hat{U}_i), \tag{2}$$

E. Experiments and Results

In this section we show the experiments conducted to proposed models.

1) DroNet-based Model: The CNN based on *DroNet* is trained on 7000 samples of pictures with their corresponding positions. The data split consists of 80 percent for training and 20 percent for validation. The model was trained on a GeForce RTX 2070 GPU. The Adam optimizer is used with a Mean-Squared-Error (MSE) loss function to evaluate performance. The model was first trained using a transfer learning approach, pretrained weights were loaded onto the convolutional layers which were then made untrainable and only the last three dense layers were trained. The model was trained for 300 epochs with a batch size of 128. As seen in Fig. 9 both training and validation losses smoothly converge and reach final losses of 1.18 for validation and 1.13 for training. The model was then trained from scratch for 30 epochs using a batch size of 128. In this case the final loss achieved is 0.5 for training and 0.7 for validation as seen in Fig. 10 . The validation loss oscillates and has a noticeable variance with the training loss.

The average prediction time per frame in the case of the CNN based off DroNet, it is equal to 0.004 seconds meaning the network can predict the position of gates at a rate of around 240 frames per second.

2) Inception-based Model: In order to make a fair comparison between the DroNet-based model and the Inception-based model, both models are trained on the same data and with the same training/testing splitting percentage. First, We started trained the regression outputs that predict the gate's position. Three experiments were conducted:

1) Training the model from scratch: Figure 11 shows the training and testing losses. The model achieved a final MSE loss of 0.27 on training and 1.5 on testing. We can see that the model overfits, and that is not surprising since the model has a large number of parameters and trained on a relatively small dataset.
2) Freezing Inception pre-trained weights and training the added fully connected layers only: Figure 12 shows

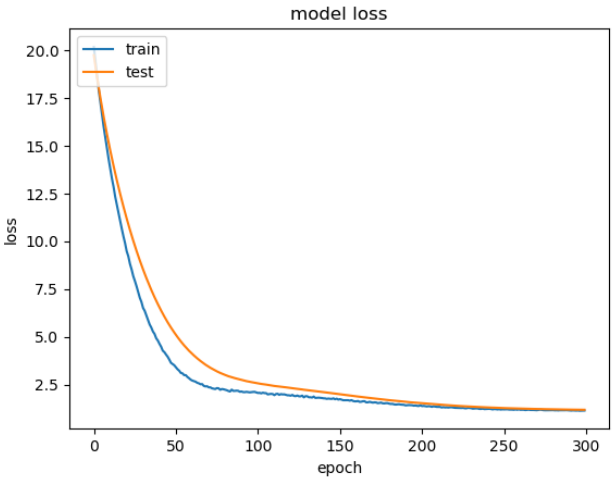

Fig. 9. Loss graph for the CNN based on *DroNet* using transfer learning

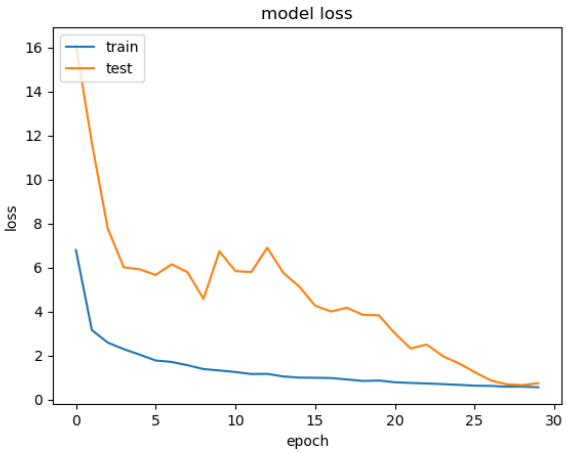

Fig. 10. Loss graph for the CNN based on *DroNet* trained from scratch

the corresponding losses. The Inception-based model achieved 0.052 and 0.19 on training and testing losses, respectively.

3) Finetuning Inception pre-trained weights: Figure 13 shows the corresponding training and testing losses. We can see that finetuning the weights did not result in a noticeable improvement on the losses.

After training the regression output, we trained the classifier that outputs the probability of the gate's existence. To do so, we froze the weights of the convolutional layers and the first fully connected layer (the one with 1024 neurons) and kept the 300-neuron layer and the Sigmoid output layer trainable. The loss function used is Binary-cross-entropy. Data augmentation techniques were used to increase the training data with a random zoom with a range of 10%, and with width and height shift of 15%.

Figure 14 shows the classifier accuracy. We can see that the network classifies the gate with 100% accuracy. This result was predictable since the gate had a red stripe making its detection relatively easy.

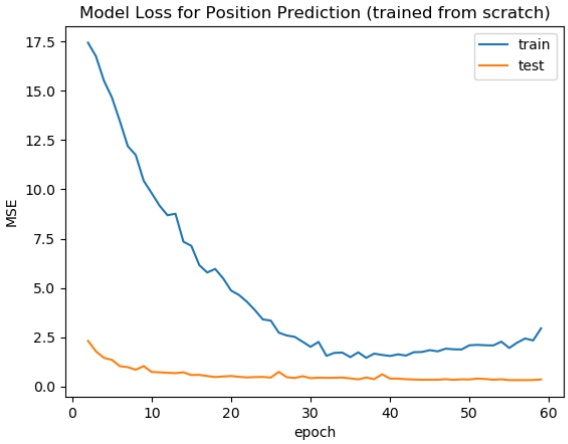

Fig. 11. The MSE loss of the position estimation of the Inception-based model trained from scratch.

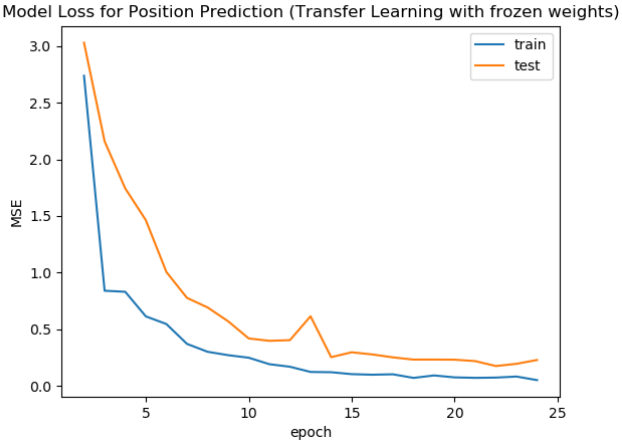

Fig. 12. The MSE loss of the position estimation of the Inception-based model loaded with pre-trained weights without finetuning them.

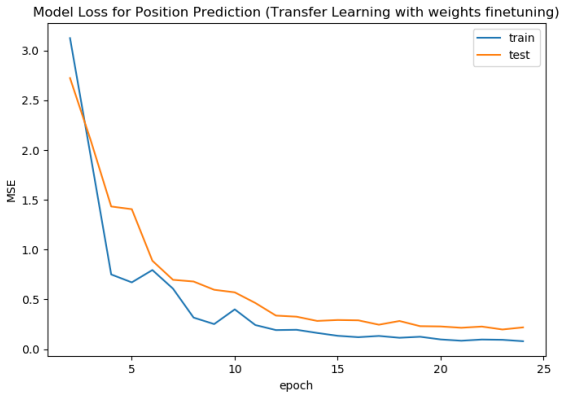

Fig. 13. The MSE loss of the position estimation of the Inception-based model trained with pre-trained weights with finetuning.

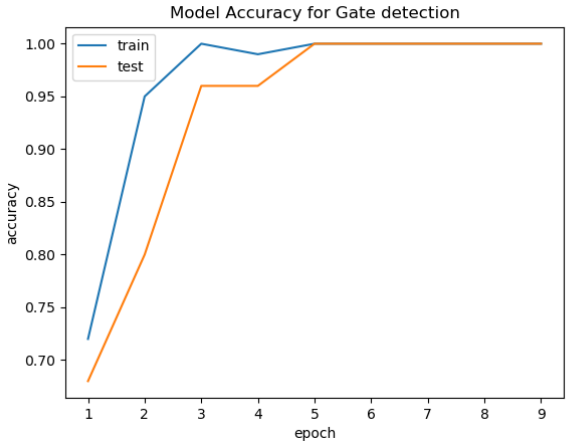

Fig. 14. The accuracy of gate detecting of the Inception-based model.

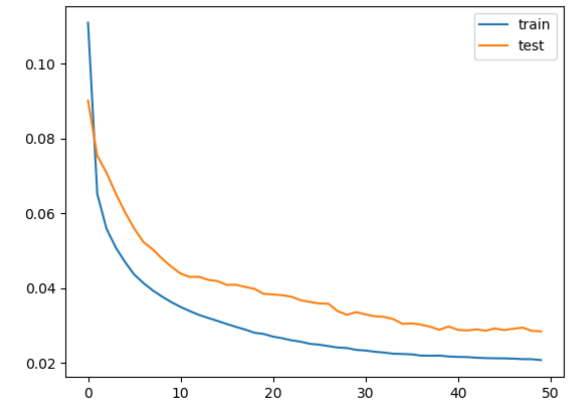

Fig. 15. RNN Loss Curve

F. Experiments on the Recurrent Network Block For Path Planning

For the RNN, the LSTM was trained on 10 thousand data instances, for 50 epochs, on 2060Ti GPU. The loss graph can be seen in Fig. 15.

G. Analysis of Results

Regarding the DroNet-based model, the transfer learning approach leads to a smoother convergence and a closer validation to training loss because the preloaded weight have been already trained and adjusted on large amounts of data and we are only training the last 3 layers. However the final loss (1.1) does not decrease below 1 which could be explained by the fact that DroNet was initially trained on a different data format (pictures of streets with cars, buildings and humans as obstacles) and to conduct slightly different tasks which consist of predicting the steering angle and probability of collision. As for training from scratch, the model was able to reach a lower loss (0.5), however the convergence is not as smooth and the validation loss has a 0.2 difference which can be explained by lack of enough training data given the depth of the CNN. Regarding the Inception-based model, on the other hand, training it from scratch resulted in overfitting on the training data. Whereas the Inception V3 CNN, trained on ImageNet dataset for a classification task, transferred well to do a regression task. The proposed Inception-based model trained with the preloaded Inception weights fixed, achieved lower loss of 0.052 and 0.19 on training and testing outperforming the DroNet-based model.

Regarding the inferecing time, DroNet-based model has a throughput of around 240 FPS compared with the Inception-based model which has around 15 FPS.

We can see that the Inception-based model has slightly higher accuracy predicting the position of the gate. However, DroNet-based model is 16 times faster than the Inception-based network making it more suitable for work on embedded-GPU devices such as the Nvidia Jetson TX2.

V. Conclusion

In this paper we provided a policy network for a quadrotor to traverse a gate located at a random location in 3D space.

To tackle vision, we tested 2 neural networks (CNNs) for 3D pose estimation and trained them on our own dataset. The training data is a set of gate shapes placed at different positions in 3D space and generated from simulation in Unity environment. The first network, *DroNet*, was inspired from the work of [8] in 2018. The second network, Inception, was inspired by the work of [20] in 2015. These two networks were modified where the last few layers were changed to fit our purposes. Multiple experiments were conducted, Training was done both from scratch and on the last portion of the network, leaving the rest frozen, to apply transfer learning to our data.

The Inception-based model had slightly higher accuracy predicting the position of the gate. However, DroNet-based model was 16 times faster than the Inception-based network making it more suitable for work on embedded-GPU devices such as the Nvidia Jetson TX2.

To tackle path planning, we presented a simple Recurrent Neural Network design. We trained it on data generated from a traditional path planner, optimizer, for drone navigation in space.

Our approach is genuine and is, up to our knowledge, the first to tackle vision and path planning within one network design for a drone to navigate and traverse a gate in 3D space.

This is an open-research area topic, and there is future work involved. One aspect would be in achieving a higher accuracy and lower loss for the network designs, and testing different architectures that may outperform our method. Another possibility would be in generating more gate shapes and training on different types of gates to achieve random gate shape detection and traversal in 3D space.

REFERENCES

[1] J. Lin, L. Wang, F. Gao, S. Shen, and F. Zhang, "Flying through a narrow gap using neural network: an end-to-end planning and control approach," *arXiv preprint arXiv:1903.09088*, 2019.

[2] D. Falanga, K. Kleber, S. Mintchev, D. Floreano, and D. Scaramuzza, "The foldable drone: A morphing quadrotor that can squeeze and fly," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 209–216, 2019.

[3] D. Falanga, E. Mueggler, M. Faessler, and D. Scaramuzza, "Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision," in *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 5774–5781, IEEE, 2017.

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

[5] X. Lin, Y. Yu, and C. Sun, "Supplementary reinforcement learning controller designed for quadrotor uavs," *IEEE Access*, vol. 7, pp. 26422–26431, 2019.

[6] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

[7] S. D. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. H. Eng, D. Rus, and M. H. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[8] 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.

[9] Z.-W. Hong, C. Yu-Ming, S.-Y. Su, T.-Y. Shann, Y.-H. Chang, H.-K. Yang, B. H.-L. Ho, C.-C. Tu, Y.-C. Chang, T.-C. Hsiao, *et al.*, "Virtual-to-real: Learning to control in visual semantic segmentation," *arXiv preprint arXiv:1802.00285*, 2018.

[10] M. Q. Dao, D. Lanza, and V. Frémont, "End-to-end deep neural network design for short-term path planning," in *11th IROS Workshop on Planning, Perception, Navigation for Intelligent Vehicle (PPNIV 2019)*, IEEE, 2019.

[11] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," 2016.

[12] S. Jung, S. Hwang, H. Shin, and D. H. Shim, "Perception, guidance, and navigation for indoor autonomous drone racing using deep learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2539–2544, 2018.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[14] M. Mueller, V. Casser, N. Smith, B. Ghanem, V. C. Center, and T. KAUST, "Teaching uavs to race using ue4sim," *arXiv preprint arXiv:1708.05884*, 2017.

[15] E. Kaufmann, M. Gehrig, P. Foehn, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Beauty and the beast: Optimal methods meet learning for drone racing," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 690–696, IEEE, 2019.

[16] E. Kaufmann, A. Loquercio, R. Ranftl, A. Dosovitskiy, V. Koltun, and D. Scaramuzza, "Deep drone racing: Learning agile flight in dynamic environments," *arXiv preprint arXiv:1806.08548*, 2018.

[17] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos, "Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 2799–2806, 2018.

[18] MIT, "Flight Googles." flightgoggles.mit.edu, 2020. [Online; accessed 7-May-2020].

[19] M. W. Mueller, M. Hehn, and R. D'Andrea, "A computationally efficient motion primitive for quadrocopter trajectory generation," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.

[20] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *Proceedings of the IEEE international conference on computer vision*, pp. 2938–2946, 2015.

[21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[22] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.