Institute for Aerospace Studies
UNIVERSITY OF TORONTO

AER1217: Development of Unmanned Aerial Vehicles

Project: Autonomous Drone Racing Competition

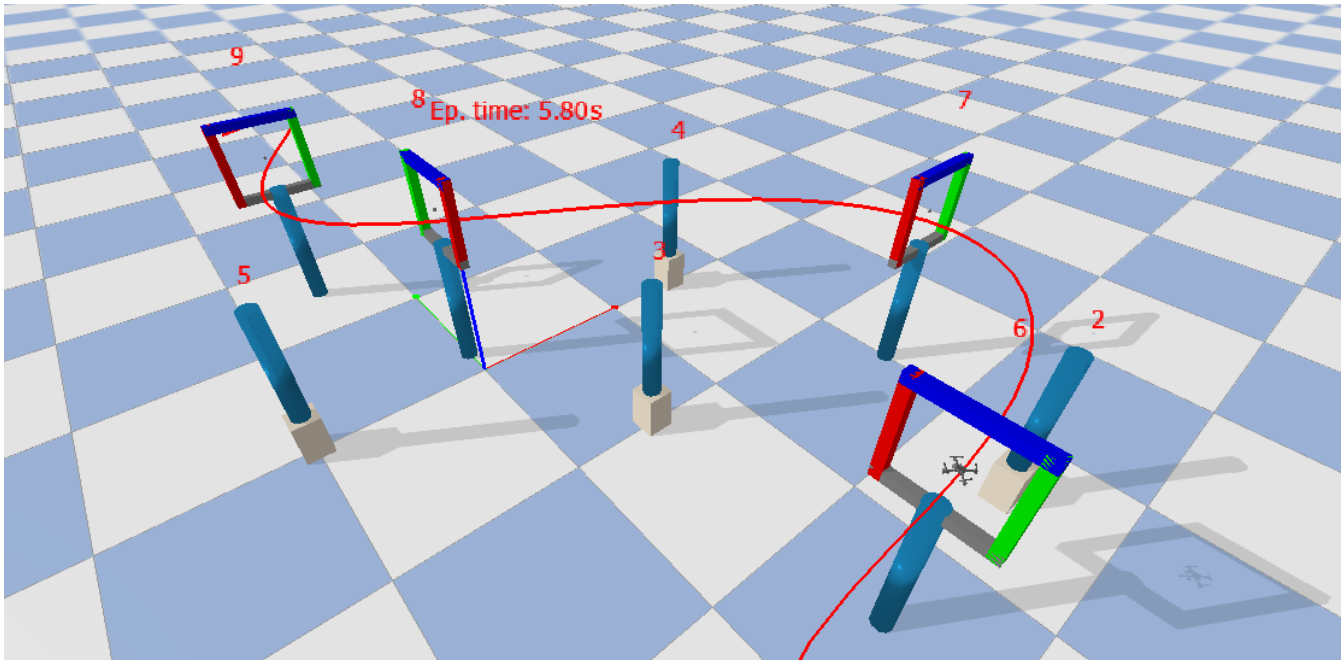

Figure 1: Final project PyBullet simulation environment for autonomous drone racing. A Crazyflie nano-quadrotor is required to fly through the gates and reach the target point, as fast as possible, without collision.

1 Introduction

Drone racing has become a popular sport in which professional human pilots fly small quadrotors through complex tracks at high speeds. Combining cutting-edge robotics technology and high-speed competition, drone racing is creating a new era of sports in the physical world, virtual environments, and the metaverse. Human pilots undergo years of training to master the sensorimotor skills involved in drone racing competitions. Today, robotics researchers are developing autonomous algorithms to achieve or even outperform expert human pilots in racing competitions. A key challenge is planning a time-optimal trajectory to complete the task while satisfying certain constraints. In the final project, we will hold an autonomous drone racing competition, which emphasizes the motion planning problem in robotics.

The final project includes two phases:

- **Phase 1:** In the first phase, you are given a Pybullet simulation environment to develop the algorithm. The task is to design a planning algorithm to navigate, as fast as possible, through the four gates and reach the target point while avoiding obstacles. The sequence of the gates will be provided during the project demonstration day. In the simulation, we provide accurate pose measurements of the quadrotor, positions of the gates, interfaces to a low-level controller, and noisy positions of the obstacles. You are required to show that the proposed algorithm can safely navigate the quadrotor through the gates while

dealing with the uncertainties of the obstacles.

- **Phase 2:** In the second phase, we will execute the algorithm you developed in a real-world experimental setup. We will check if the proposed algorithm can navigate safely through the gates with a predefined sequence and the amount of time it takes to complete the task.

2 Simulation Environment Setup

The Pybullet simulation environment we provided is based on the open-source project `safe-control-gym`. We recommend Ubuntu 20.04 on a mid-tier laptop and GPU. Open Terminal, clone the git repository, and check out to the `aer1217-course-project` branch:

```
mkdir aer1217-project
cd aer1217-project
git clone https://github.com/utiasDSL/safe-control-gym.git
cd safe-control-gym
git checkout aer1217-course-project
```

While not mandatory, we recommend using Anaconda to manage the software environment. If Anaconda platform is not installed already, check out the official website for the installation. Create and access a Python 3.8 environment using `conda`.

```
conda create -n aer1217-project python=3.8
conda activate aer1217-project
```

Install the `safe-control-gym` repository.

```
pip install --upgrade pip
pip install -e .
```

Now, we install `pycffirmware`, which is the Python bindings of the official Crazyflie firmware. Note, `pycffirmware` needs to be installed in the same folder of `safe-control-gym`.

```
cd ..
git clone https://github.com/utiasDSL/pycffirmware.git
cd pycffirmware/
git submodule update --init --recursive
sudo apt update
sudo apt -y install swig
sudo apt install build-essential
cd wrapper/
chmod +x build_linux.sh
./build_linux.sh
```

Then, we can run the scripts in `aer-course-project/` folder to get started.

```
cd ../../safe-control-gym/aer-course-project/
```

```
python3 final_project.py --overrides ./getting_started.yaml
```

3 Project Description

3.1 Gates and Obstacles

An overhead view of the simulation environment and its schematic diagram are provided in Figure 2. There are **four gates** and **four obstacles** in the environment. The quadrotor is required to navigate through all the gates before reaching the target point. The accurate positions of the gates and the nominal positions of the obstacles are provided in `getting_started.yaml`. Please do not modify the original sequence of the gates and obstacles. The heights of the four gates are set to be **one meter**. We injected uniform noises $\mathbf{U}(-0.2, 0.2)$ to the obstacles' nominal x and y positions to represent the uncertainties of the environment. The starting and final target points are $(-1.0, -3.0, 0.0)$ and $(-0.5, 2.0, 1.0)$, respectively.

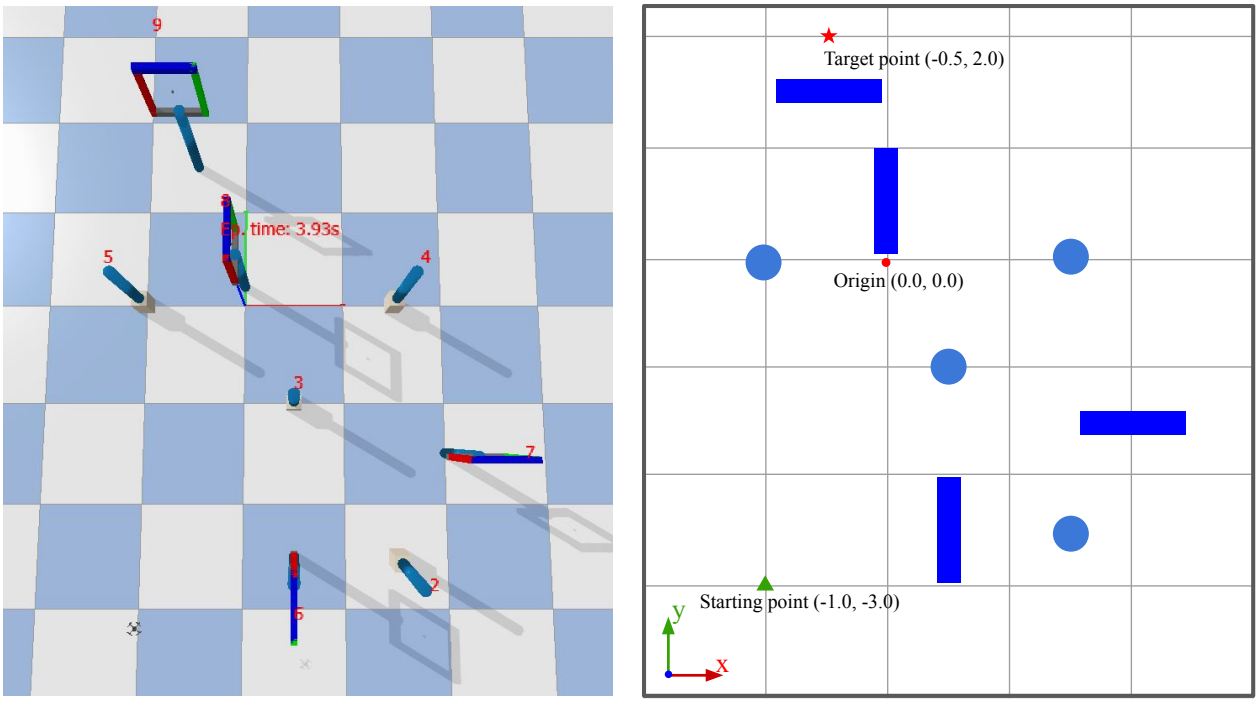

Figure 2: (Left) Overhead view of the simulation environment. (Right) Schematic diagram of the simulation environment. Obstacles and gates are indicated as circles and squares, respectively.

The dimensions of the gates and obstacles are shown in Figure 3. The inner edges of the square gates are 40 cm in length. The height of the gate's center is one meter. The obstacles are pillars with 12 cm in diameter and 1.3 m in height. Passing through the gates from either direction without collision counts for one successful pass. Repeatedly passing the same gate only counts for one pass.

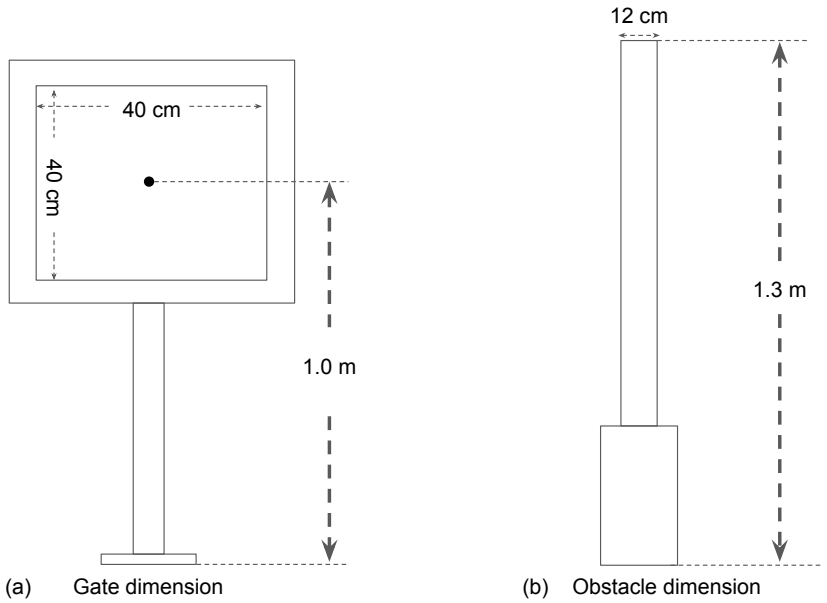

Figure 3: (Left) Overhead view of the simulation environment. (Right) Schematic diagram of the simulation environment. Obstacles and gates are indicated as circles and squares, respectively.

3.2 Trajectory Planning

As described above, the x and y positions of the obstacles are corrupted with uniform noise. Hence, the designed planning algorithm should take into account the uncertainties to ensure safe navigation. The sequence of the gates will be provided in the project demonstration day. You are required to plan the trajectory based on the provided sequence and navigate the quadrotor in simulation and physical experiments. We will record the total time it takes for the quadrotor to finish the trajectory and reach the final target point in experiments. As a drone racing competition, we will rank the teams based on their flight time in completing the task, which will account for 5% of the final score. The environment boundaries are set to be $x \in [-3.5, 3.5]$, $y \in [-3.5, 3.5]$, and $z \in [0, 2]$. The planned trajectory should be within this range. Please do not fly outside these boundaries.

3.3 Controller Interface

In the simulation environment, we provide the Crazyswarm's Crazyflie command interface, including high-level commands `takeoff()`, `land()`, `goTo()` and low-level fullstate commands `cmdFullState()`. The low-level command `cmdFullState()` sends the desired position, velocity, acceleration, and heading references to the onboard controller, which is beneficial for smooth trajectory planning. A complete command enumerate class and corresponding references are provided in `project_utils.py`. You are free to choose either high-level command or a combination of high-level and low-level commands to finish the task. Note, a `NOTIFYSETPOINTSTOP` command must be called to transfer drone state from low-level command (`cmdFullState`) to high-level commands (`takeoff`, `land`, `goTo`) (see the official documents for details). The `final_project.py` is the main script for the project, which imports the controller class defined in `edit_this.py`. All the available command interfaces are defined in `project_utils.py`.

3.4 Requirement

You are required to use the command interface to implement the proposed algorithm in `edit_this.py`. You are required to implement the path planning algorithm on your own instead of using existing packages. We have provided the necessary instructions, including example codes for waypoints generation, using low-level (`cmdFullState`) and high-level commands (`takeoff`, `land`, `goTo`), etc. You can search for strings 'INSTRUCTIONS' and 'REPLACE THIS (START)' in the scripts for detailed descriptions. We also provide a dummy utility module in `example_custom_utils.py`. Please use a similar file to add extra functions if needed. Also, please maintain the file structure and do not modify the `final_project.py` and `project_utils.py` to ensure a direct sim2real transfer in the second phase.

The evaluation procedures are as follows:

- **In the first phase**, the TAs will evaluate the algorithm in the simulation environments. We will provide a sequence of gates and check if the proposed algorithm enables the quadrotor to navigate safely through the gates and reach the target point while avoiding obstacles.
- **In the second phase**, the TAs will execute your algorithm on a dedicated course laptop to command a physical Crazyflie quadrotor to navigate in a real-world experimental environment (see Figure 4). The grades will be evaluated based on the number of gates successfully passed and the total flight time used to complete the task.

Each team will be required to use either **(1) search-based** or **(2) sampling-based** algorithms for trajectory planning, which will be announced on Quercus before the final project.

Figure 4: The real-world experimental setup for the final project. We will execute the algorithm on a real Crazyflie quadrotor and check if the proposed algorithm can navigate safely to the target point and count the amount of time it takes to complete the task.

4 Final Project Grading

The final project grading is separated into three parts: (1) simulation and experimental demonstration (25 %), (2) code (5 %), and (3) the project report (20 %). You are required to use either **(1) search-based** or **(2) sampling-based** algorithms for trajectory planning. The project demonstration is on **Apr. 20th, 2023**.

Grading is based on the following

- **Performance: (25 %)**
  - Pass each of the four gates in simulation (2 % each for a total of 8 %)
  - Reach the target point in simulation without collision (2 %)
  - Pass each of the four gates in experiment (2 % each for a total of 8 %)
  - Reach the target point in experiments without collision (2 %)
  - The rank of the total flight time in experiments (5 %)
    Note, the team's rank will be determined based on their total flight time in the real-world experiments. Only teams that complete the tasks will be ranked and eligible for this 5% score.
- **Code: (5 %)** include all code developed or modified by your team that was used in the simulation. The code should be commented appropriately.
- **PDF report (20 %)** maximum of 5 pages and contain the following:
  - Overview of the algorithms and equations used and considered for the final project, as well as the justifications behind your decisions (10 %).
  - Discussion of the simulation results (5 %).
  - Discussion of the difficulties faced during the final project, including what went well or as expected and what did not go well (5 %).

Please submit the code and final project report in a .zip format, to Quercus, latest by **April 22, 2023, 11:59 PM EST**. Students who require final grades by April 21 in order to meet graduation deadlines should inform Prof. Waslander in advance and submit the report by **April 21, 2023, 11:59 AM EST** (that's **noon**, not midnight).

5 Good luck!