Introduction to Robotics (236972)
Course Project

January 15, 2023

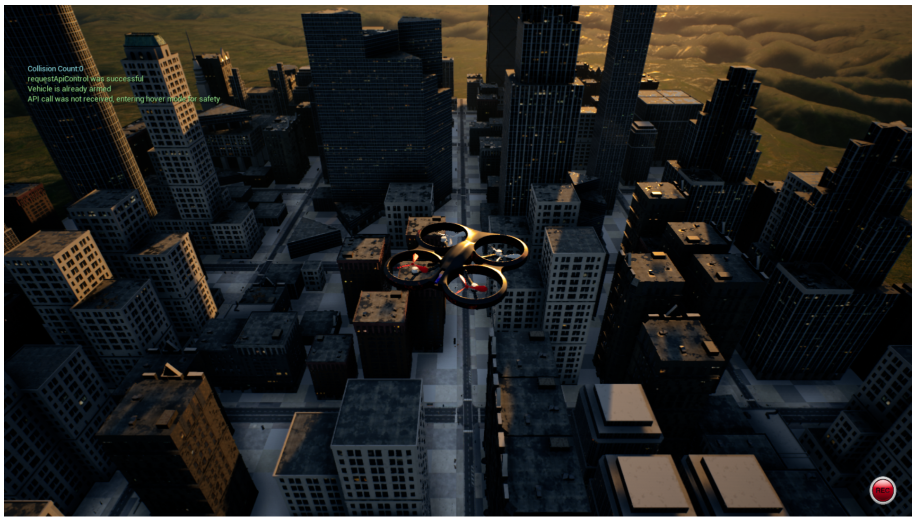
Introduction

As part of the Introduction to Robotics course, a hands-on project is essential as a way to get a real-world feel of working with robots. In this project, a student will experience working with sensors, and motion control and even perform a complicated task.

Simulation

Simulations are one of the essential tools used by the robotics and autonomous cars industry. Simulations are used to help develop a new system from the early stages of design and to evaluate and integrate highly complex algorithms at later stages.

This project gives you the opportunity to experience the advantages of simulations by using Airsim. AirSim is a simulator for drones, cars, and more, built on the Unreal Engine. AirSim is an open-source, cross-platform software supporting hardware-in-the-loop with popular flight controllers such as PX4 for physically and visually realistic simulations. It is developed as an Unreal plugin that can simply be dropped into any Unreal environment. Within the AirSim environment, the drone is controlled by using a Python/C++ API; in this manner, the drone can be configured to the project's specific needs.



```
Collision Count:0
requestApiControl was successful
Vehicle is already armed
API call was not received, entering hover mode for safety
```

Figure 1: Drone Simulation

Project Task

The task to be done in this project is to develop code that implements a simple trajectory planning algorithm (full details in course lectures). The drone receives a start position and will fly at a constant height in the fastest way toward a goal position. When it reaches the goal position, the drone will hover, and arriving at the destination will be declared. Sensors can detect obstacles in the drone's flight path that does not allow continuing in a straight line. When such obstacles are detected, the drone should employ some method to go around the obstacle until the path toward the goal is clear.

The main grading factor is the time performance of the implementation. i.e., a lower time, higher grade. In addition, penalties will be awarded on each hit of an obstacle.

A test run consists of placing the drone in some start position and letting the algorithm run until the drone reaches the target and stops, with a generous time limit of a few minutes (depending on the obstacle setup). When the time limit expires, the run is considered failed. Obstacles are the buildings in the city. A prior list of obstacles at a specific height will be given to develop and train the algorithm. Note that a different start position for a lower flight height will be given in the project presentation so that when the drone flies at a constant height, not all the obstacles will be known. The scenario will consist of a known obstacle and an additional unknown set of obstacles.

Extra Credit

An extra task that can provide extra credit is letting the drone traverse the environment, intending to create a map of the obstacles. In this case, the output is a matrix (image) of scale 1m per pixel representing the work-space, where each cell contains 1 if an obstacle exists at that location and 0 if not. Performance is measured using the time to completion and a subjective evaluation of the resulting matrix. The timing and evaluation of this task are separate from the primary task.

Technical Details

Motion

A "fly to position" command will control the drone's motion. The command is a three-dimensional cartesian coordinate of the desired location in meters referenced to the world frame in NED (North-x, east-y, Down-z) coordinate system and a scalar representing the speed in meters per second. Please note that a physical model is implemented, meaning the drone will ignore a command it cannot follow. For example, if the desired distance between two points is $2[m]$, and the desired speed is $10[m/sec]$, then the time for the drone to complete the desired distance is $0.2[sec]$. The physical model will not be able to move the drone $2[m]$ in $0.2[sec]$, and therefore, the command will be ignored. The threshold for a valid command is $0.6[sec]$, so, for the desired distance of $2[m]$ between two points, the speed must not be greater (and even less) than $2 \times 0.6 = 1.2[m/sec]$.

Prior Obstacle List

A CSV file containing a list of obstacles for a specific height will be given to train your algorithm. Fig. 2 illustrated the given obstacles list at $100[m]$. The obstacles are the buildings in the model, and there are different buildings of different heights. Fig. 3 shows a DSM (Digital Surface Model) generated from the simulation model. The Obstacle list is a slice at a given height so that, at lower heights, more obstacles will appear in the drone's flight path. The file includes the number of points, vertex coordinates of each obstacle polygon, and the obstacle type number. Note that the drone will fly at a different height in the presentation, adding obstacles for you to detect with the Lidar while also changing some of the given obstacles' shapes and dimensions.

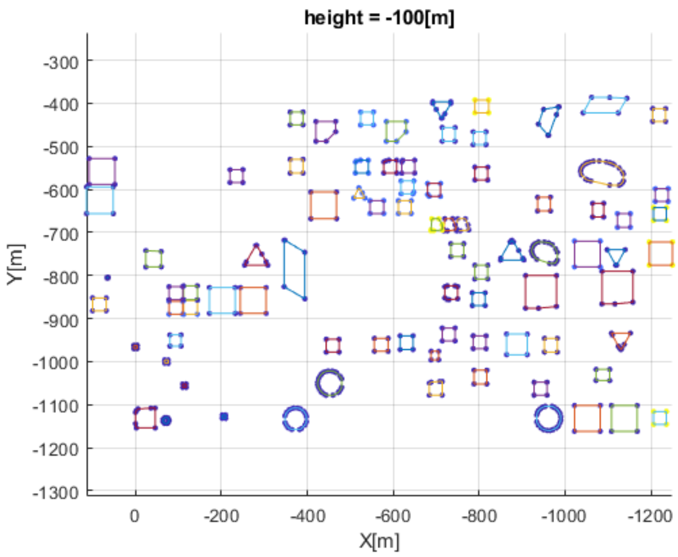


Figure 2: Obstacles Polygons

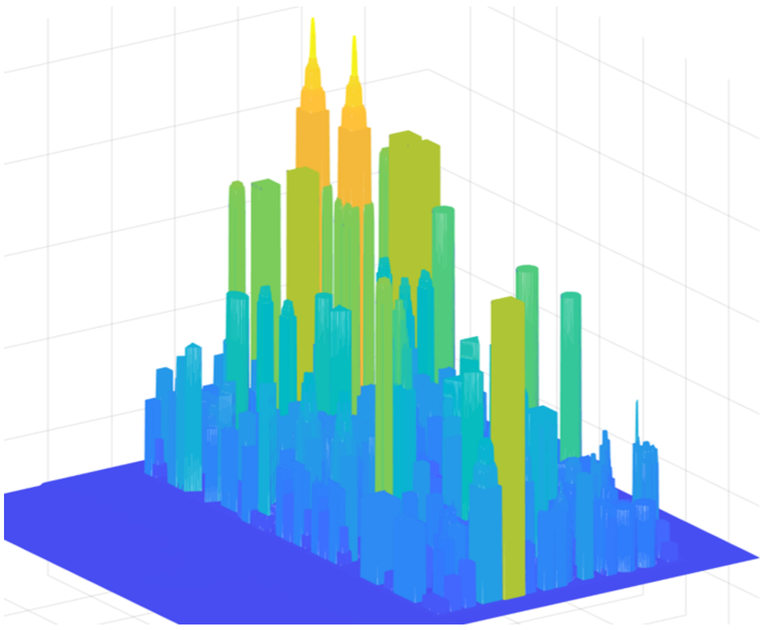


Figure 3: Digital Surface Model

Sensors

Position and Orientation

A 6 DoF (Degrees of Freedom) vector containing a position and orientation is available. The position values of the body are in meters and are referenced to the world frame (model origin) in NED (North-x, east-y, Down-z). The orientation values are the Euler angles of the body given in radians and are also referenced to the world frame. Note that the position will not be precisely equal to the desired location.

Obstacle Detection

Obstacles are detected using a Lidar sensor with a single rotating laser. The Lidar provides distance information for a set of angles around the drone. The FOV (Field of View) will be $[-90°, 90°]$, and the maximum measured distance to an obstacle will be $35[m]$. The measurements are given as an array of a three-dimensional cartesian coordinate of the detected obstacles' location in meters referenced to the drone's body frame. Note that FOV and the maximum range can be changed in the project presentation, and the algorithm should be somewhat robust to the change of setup.

Software API

The software implementation is provided in Python. The Python implementation uses version 3.8x and is tested only on Windows 10. The simulation requires some computational hardware such as a GPU and a strong processor. Therefore, the simulation can run on a personal machine or the ISL server for development and testing.

Installing API dependencies on a personal machine

- Download anaconda `https://www.anaconda.com/products/individual`
- Install anaconda with default configurations
- Run anaconda prompt from the ”start menu”
- In the terminal, enter command ”conda create −−name robotics python=3.8”
- Enter the command: ”conda activate robotics”
- Enter the command: ”pip install airsim” which installs the AirSim package
- Use any IDE which supports anaconda (Pycharm, VScode, etc.). Choose the interpreter as anaconda with the name of our environment (which we called ”robotics”)
- The API zip contains 3 files: documented API, ”DroneClient.py”, ”DroneTypes.py”, and ”main.py” for quick start and assert all works

Configuring API dependencies on an ISL server

- Start ”PyCharm” Community Edition 2022.3.1
- Check ”I confirm...” and click ”Continue”
- Click ”Don’t Send”
- Click ”New Project”
- Name ”pythonProject” as ”robotics”
- Click ”Previously configured interpreter”
- Click ”Add interpreter” button and then ” Add Local interpreter...”
- Choose ”Conda Environment”
- Choose ”Existing environment”
- Choose ”Use existing environment: robotics”
- Click ”OK”
- Uncheck ”Create a main.py...”
- Click ”Create”
- From the ”DroneAPI” folder on the desktop, copy the three files to the desktop
- Drag the files from the desktop to the robotics folder in ”PyCharm”
- Click ”Refactor”

Game Simulator

The simulator will run only on x64 Windows 10 platforms.

Game Simulator on a personal machine

- Unzip the simulator package
- At `C:/Users/<user>/Documents/` create the folder ”AirSim”
- From the unzip package copy the ”settings.json” file and paste it at the new ”AirSim” folder
- Run the ”exe” file to start the simulator (it may install dependencies at the first run)
- To exit the game, click on the ”Ecs” button on your keyboard

Game Simulator on an ISL server

Run the simulation using any `run_XXX.cmd` on the desktop. To exit the game, click on the "Ecs" button on your keyboard.

The first time you run the simulation, do the following steps:

- Choose Vehicle: "No"
- Hit "ESC" or click "X" to close
- Copy "settings.json" from the desktop to "`Documents/AirSim/`" and replace existing
- Open "settings.json" (in "`Documents/AirSim/`") and modify line 5: "ApiServerPort: 41451"
    1. Your port number is your username's number (groupXX) + 50000
    2. For example: group23 should change to 50023
- Save the file and exit the editor
- Run the simulation again

Python API

The "DroneClient()" includes connect, get, and set methods that allow the user to control the drone and receive data from the sensors:

- connect() - establishes a connection to the game simulator
- isConnected() - a Boolean to state if a connection was established
- setAtPosition(x, y, z) - sets the starting position of the drone referenced to the world frame
- flyToPosition(x, y, z, speed) - sets the desired destination point and flight speed referenced to the world frame. Note that if this command does not lock the drone's action, i.e., another command can be set in mid-flight, and the drone will change course.
- getPose() - receives the position and orientation data.
- getLidarData() - receives the Lidar's point cloud data as a flat list of 3-dimensional coordinates referenced to the drone's body. X - positive in the direction of flight, Z - positive in the downward direction, and Y complies with the right-hand rule.

General Notes

Code

All supplied code is provided in zip files uploaded to a SharePoint project site at `https://technionmail.sharepoint.com/sites/IntroductiontoRobotics-236927`.

There are two main directories in the repository:

- PyProj - containing the DroneAPI zip file with 'DroneClient.py", "DroneTypes.py", and "main.py" code
- Simulator - 3 zip files with the simulator package and a "settings.json", "obstacles_100m_above_sea_level.csv" files. Each zipped package is generated to run in a different resolution of your choosing.

Connecting to ISL server via Remote Desktop

- Fill in your details in the google doc. This will be your group's user on the server `https://docs.google.com/document/d/185787A8uJC3qR7fPxfxo1uvlIVxboE1_0ZMe5TJZrIQ/edit?usp=sharing`
- To connect to the server outside the Technion, use your VPN and "remote desktop" with IP: `132.68.38.38`
- To connect to the server from the Technion use "remote desktop" with `newton4.cs.technion.ac.il`
- The password will be given to each group individually