The University of Haifa RBD Labs

Learning Data in Real-Time Systems 203.3274

Autonomous Drone Scanning and Mapping

Authors

Waseem Tannous

Zinat Abo Hgool

Supervisors

Prof. Dan Feldman

Mr. Sagi Lotan

Mr. Bar Saguy

Table Of Contents

System Overview

The objective of the course is to make a drone (DJI Tello) **autonomously** scan the area, find the exit and navigate between the rooms in a closed area or a house.

The code was developed using Python. An external SLAM software was used (ORB_SLAM2 [1]) to scan the area and to generate a point cloud which is then analyzed and used to make decisions of the drone's next destination.

Link to the github repository: https://github.com/waseemtannous/DroneScanning

Link to the project video: https://youtu.be/lqBRYkWSmjI

Link to the ORB_SLAM2 repository: https://github.com/raulmur/ORB_SLAM2

Setup

Camera Calibration

In order to make the ORB_SLAM2 work efficiently, we need to calibrate the input we receive from the camera. Since it is open-source, we were able to change the code and many other parameters.

In the “Camera Calibration” module, we used a standard code we found in the opencv website [2] which takes some photos of a chess board as an input and returns all the values and parameters used to calibrate and undistort the images from a specific camera.

These parameters are then used as an input for the **mono_tum** program, which is the monocular (single camera input) version of the ORB_SLAM2. These values are changed in the YAML file which is the config file for the mono_tum.

Mono_tum

As said before, this is the monocular version of the ORB_SLAM2. Since the program is open-source, we have the ability to change how the program operates.

As a default, the program gets pictures as an input which are all the frames cut from a specific video. Since this is not ideal for our situation, we ought to make some changes to the code (mono_tum.cc).

Using the help of [3], we were able to change the input of the program to be the live footage of the computer’s webcam using opencv in C++.

After doing some research about the drone, we were able to find that it transmits live video using a UDP connection. We were able to connect to the drone’s camera broadcast using this connection in the opencv.

At this stage, the SLAM was getting live input from the drone as long as it’s stream was active.

In order for the SLAM to work in all configurations listed above, we added a parameter which indicates in which way we want to use the program, either in photo/sequence mode, webcam mode or drone’s camera mode. This parameter can be changed dynamically without re-compiling the code as it is received from the terminal. Instructions on how to run the program are on line 40 in the mono_tum.cc file.

We also added a function that saves the point cloud generated by the program in CSV file format.

Mono_tum config file

A config file is needed to run the mono-tum. It contains the camera calibration parameters and some other relevant parameters (DRONE_PARAMS.yaml).

Some of the parameters we changed are nLevels, minThFAST and iniThFAST.

According to what we read about the program, these parameters control how much the program is sensitive to contrast and edge detection. The more sensitive it is, the more points it detects in the frame.

Controlling the drone using code

First of all, the computer had to be connected to the drone’s wi-fi hotspot. Then the drone’s python library “djitellopy” is used to send commands to the drone.

We were able to control the drone’s speed, height and some other moves using the API’s functions. We tested many commands in order to understand how the drone reacts to the functions.

How The System Works

The idea is to make the drone aware of its surroundings, find an exit point (door) and make the decision to exit the room. We implemented a 3-step algorithm to accomplish this goal:

1. Make a 360 degree scan using the SLAM.
2. Analyze and detect the exit point.
3. Fly the drone towards this point.

The algorithm repeats until no exit is found.

1. Room Scanning

This part needed two working threads. The first one moves the drone 360 degrees, the second one runs the ORBSLAM2 program.

In the first thread, according to multiple tries and errors, we found that the ideal movement parameters are: 15 degrees each rotation, height at about 160cm, after each rotation move up and down 20cm in order to detect the points better and wait 3 seconds between each rotation (each step) so that the SLAM can detect the points.

Since the SLAM gets the points in real-time, it runs separately in another thread.

After the drone has landed, the SLAM saves the point cloud as a 3 coordinate system in a CSV file.

2. Finding Exit Points

First we needed to build a point cloud using the CSV file. We used the “pyntcloud” and “open3d” libraries to build the point cloud from x, y, z arrays.

In order to fully see the room, we used the x, z coordinates which enabled us to see the room from the ceiling view. That way, we can see clearly the room’s borders and the exit point.

First, we prepare the point cloud by cleaning and deleting the random and noisy points by dividing the points into inliers and outliers found in the point cloud object. We added multiple cleaning functions in the file “PointCloudCleaning.py” found in [4].

After removing the outlier points, we continue with the point cloud made from the inlier points.

The next step is to find the room’s borders and walls by drawing a rectangle around it. First, we use the inlier point cloud, we extract the x and z coordinates to be able to see the room from the ceiling view. We calculate the “average” center point by averaging the x and z coordinates. Then, we divide the points into 4 groups which contain the points above the center point, below it, left to it and right to it accordingly. We average the distance of each group to the center point and find an average distance (average point) from this side to the center point. We repeat this step for each side. Now each side has an average point which the rectangle lies on. Each average point will approximately be in the center of all points from this side, so in order to bound all the points within the rectangle, we need to multiply its coordinates and move it by a factor of 2. Finally, we find the coordinates of the bounding rectangle. (fig. 1 & 2).

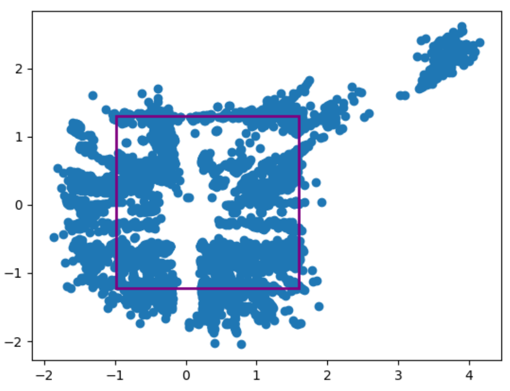

Fig.1 (average coordinates)

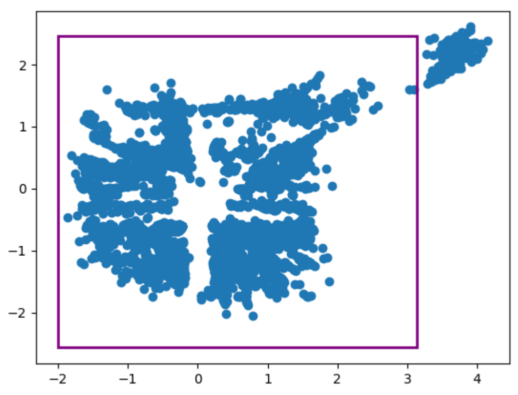

Fig.2 (after multiplying by 2)

In order to recognize the exit point, we use a cluster finding algorithm on the points that lie outside of the rectangle found previously. In other words, we need to find a cluster of points outside the room. We use the method “Hierarchical Clustering” using the algorithm “fclusterdata” from the library “scipy.cluster.hierarchy”. We used the help of [5] and [6].

That way we find all the clusters found outside the rectangle (room). We return all of them. Each cluster represents an exit from the room.

3. Navigate Towards The Exit

In order to navigate toward the exit, we need to find the cluster’s center point. For each cluster we calculate the center point by averaging all of the points coordinates.

Since we could have multiple exits, and since there are many factors determining which exit to choose, we agreed and decided to take the furthest exit from the drone.

In order to get to this point, the drone needs to rotate at a specific angle and then move some distance towards this point. The angle is calculated using the TAN function and then we add a correction of 0, 90, 180, 270 according to the quarter that the point lies on. The distance is measured using the euclidean distance between 2 2D points.

After some experimentation, we found that 1 unit in the ORB_SLAM2 is about 160cm in real life, so the distance is multiplied by 160.

Another Algorithm

We tried to implement another way to find a bounding box of the room using an AI based algorithm.

The algorithm “findBestBoundingBox” can be found in the “ExitFinding.py” file.

The algorithm sorts all the points by the ‘y’ coordinate and then divides all the points into sub-groups of size of log(n), where n is the number of all points in the cloud. For each sub-group (slice of the point cloud), we calculate the bounding box of all these points and calculate the fitness of each box. The box with the lowest fitness is then chosen to represent the room’s borders.

We define fitness by the difference between all the points in the point cloud that lie outside of the box and the points that lie inside the box, in absolute value. Here we also take the xz coordinates for the same reasons of the first algorithm.

This algorithm is not ideal because if we look at each slice of the point cloud, some slices have points that are not distributed among all the space, so the bounding box of each slice may not be relevant for the entire points.

The code for this algorithm was not deleted and could be used in the future for different approaches of this project.

Test and Run

We started testing the algorithm in Jacobs building room 209, a simple square room with one exit. The border was drawn and an exit point (orange) was found.

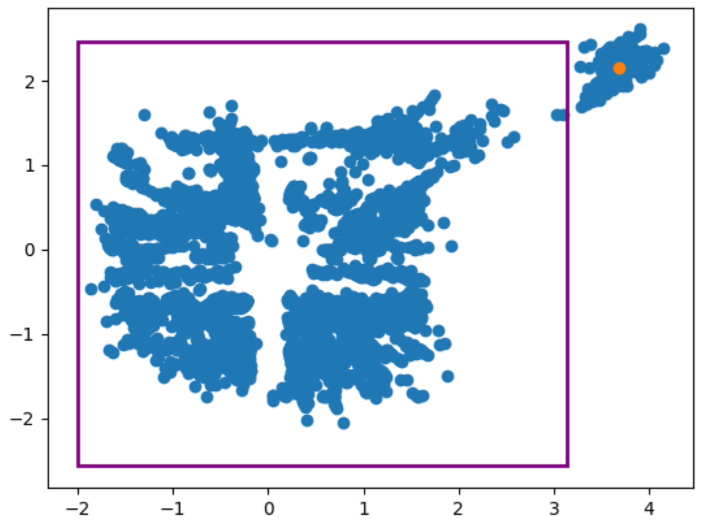

Fig.3

After a successful test, we moved on to a more complex area like a house. The house we tested in is represented by the following blueprint:

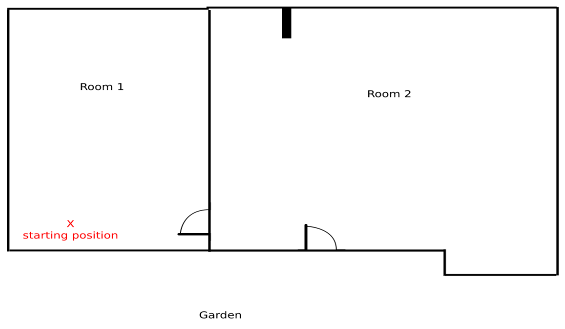

Fig.4

We started the drone in room 1, making a 360 degree scan and then processing the point cloud we got. Fig.5 represents a 2D drawing of the room. Fig.6 is the result of using the inlier points, or in other words deleting all the outlier and noisy points. Fig.7 is the result of the algorithm, where we can clearly see the borders of the room in purple and the exit point in orange.

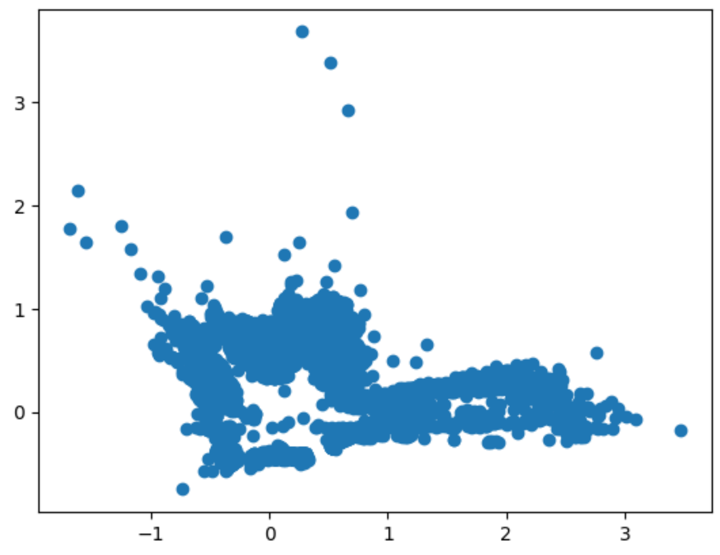

Fig.5 (x-z coordinates)

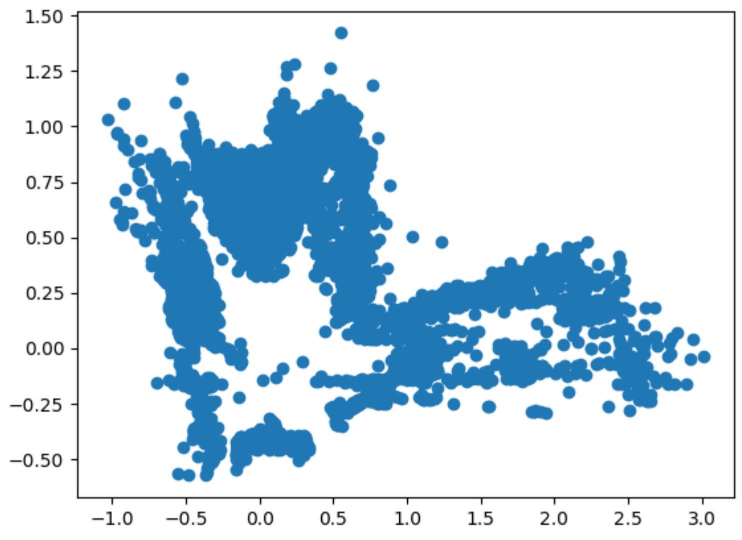

Fig.6 (inlier points)

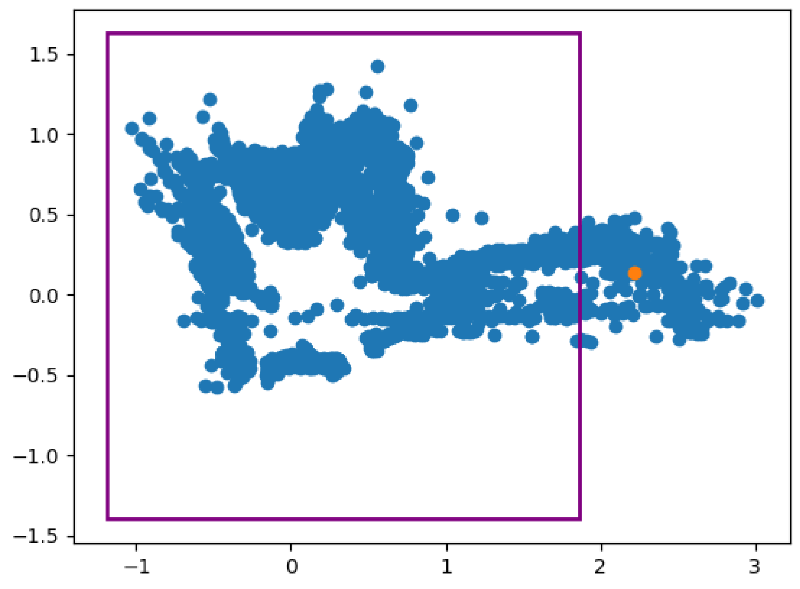

Fig.7 (bounding box and exits)

After exiting the room, the drone stops, lands and starts this process all over again for room 2. See Fig.8 & 9 & 10.

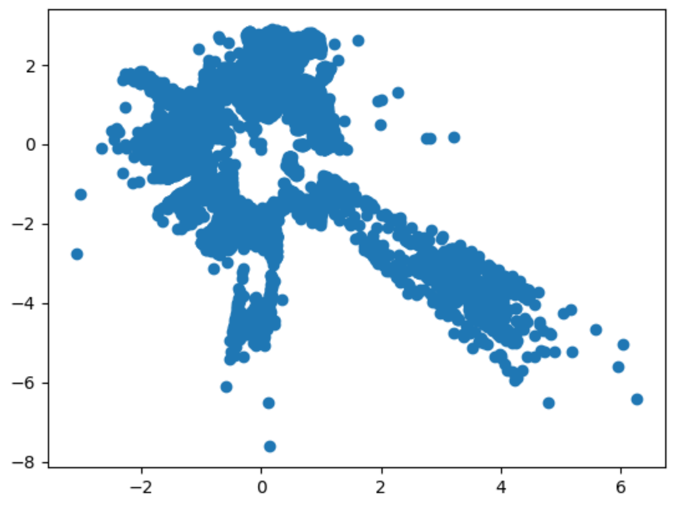

Fig.8 (x-z coordinates)

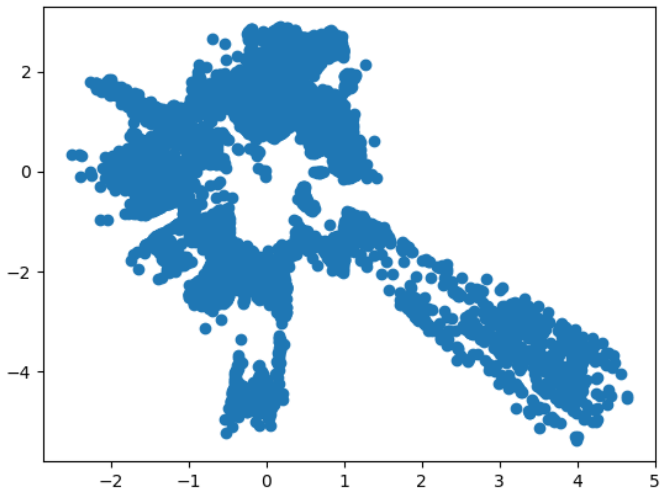

Fig.9 (inlier points)

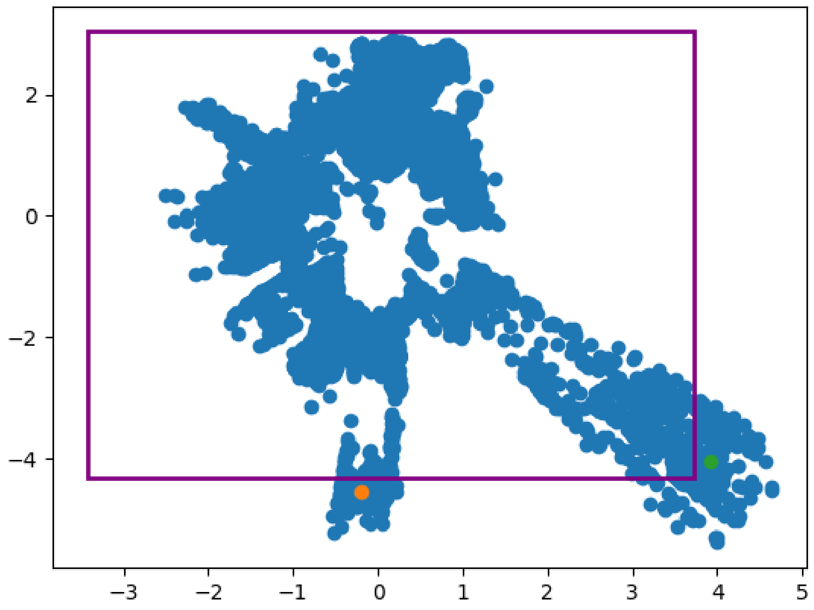

Fig.10 (bounding box and exits)

In Fig.10, two exits are found, so as mentioned earlier, the furthest point is chosen (exit 2 shown in Fig.11).

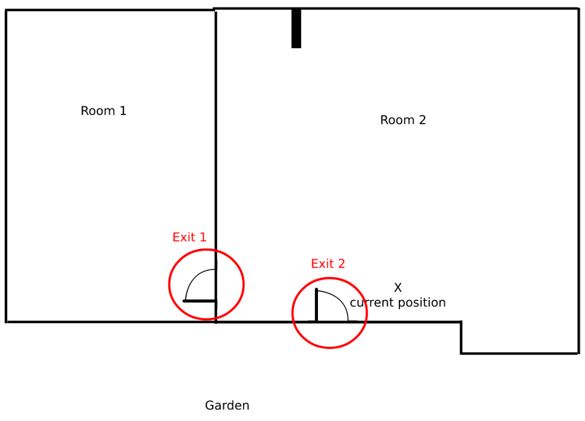

Fig. 11 (Two exits in the room. Exit 2 is taken)

The drone exits through the door and lands. Here The process stops and terminates.

<u>**** Important note:**</u> After each 360 degrees scan, the drone lands. This is because of the real-time ORB_SLAM2 detection. The scan takes about 4-5 minutes but the SLAM takes about 10 minutes to finish. Also, the drone has a built-in mechanism that tells it to land if a command is not received during a specific amount of time. The drone also lands when reaching a critical battery level to avoid damage. Not forgetting that the drone's battery lasts a theoretical 13 minutes, so no full algorithm run could be completed while the drone was flying. The process ran in three parts: first we made a 360 scan and landed, processed the data and lastly reconnected to the drone and sent it the relevant commands.

Contribution to other groups

- Starting from the program's installations, Waseem has prepared a file with all the problems he faced during the installations and downloads with a way to solve and overcome them. He sent the file to all the students in the course in our Whatsapp group (installation guide.pdf).

- We helped other groups with running the ORB_SLAM2 in a separate thread in the code.

- We sent the calibration parameters of the drone’s camera to some people after making a program that calibrates a camera.

- We helped others connect the ORB_SLAM2 to the drone and receive a real-time broadcast using UDP and Opencv in C++.

Why Should We Get A High Grade ?

- We installed Ubuntu natively on a computer and didn’t use a virtual machine as running an OS natively on a computer’s hardware translates to a smoother and faster code execution times as the OS has all cores and threads fully dedicated for it. This was extremely helpful as we are dealing with a real-time processing system.

- We added the functionality to run all the configurations available to work with using the monocular version of the ORB_SLAM2. We added a simple “flag” parameter which can be dynamically sent to the program and it indicates whether we want to receive footage from a specific folder, webcam or a network connected camera like the DJI Tello’s camera. This means that for each configuration, we don’t need to recompile the program’s code.

- We used our knowledge in Artificial Intelligence and built an algorithm based on some AI concepts. Although the idea of the algorithm wasn’t ideal for our use case, the code can be used and can be improved for future projects.

- Our code receives all it’s parameters from a configuration file named “config.json”. This means that our program can be compiled once and run multiple times with different parameters that can be dynamically and easily changed after each run. The default parameters are the ideal ones we found by testing the program multiple times.

Resources

[1] https://github.com/raulmur/ORB_SLAM2
[2] https://docs.opencv.org/4.5.1/dc/dbb/tutorial_py_calibration.html
[3] https://robot-vision-develop-story.tistory.com/10
[4] http://www.open3d.org/docs/0.9.0/tutorial/Advanced/pointcloud_outlier_removal.html
[5]
https://stackoverflow.com/questions/10136470/unsupervised-clustering-with-unknown-number-of-clusters
[6]
https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.fclusterdata.html#scipy.cluster.hierarchy.fclusterdata