Aer 1217 Report
Final Project: Autonomous Drone Geodashing

Team DLI:
Boyang Cheng, Chunwai Cheung, Jingwei Zhang

1. Introduction

Geodashing is a competition to see who can arrive at dash-point locations randomly chosen by computer before the deadline. The AER1217 final project consists of two phases. In Phase 1, the objective is to achieve georeferencing on both obstacles and four selected landmarks in Toronto. By using the coordinates and diameters of obstacles and coordinates and yaw angles of landmarks obtained from Phase 1, an optimized path which enables the drone to visit all landmarks without crashing into any obstacle is generated in Phase 2.

2. Code Structure

Two ROS packages, `localize_object` and `aer1217_ardrone_simulator`, are created to achieve the Geodashing function, which contain the implementation of Phase 1 and Phase 2 respectively. The path planning algorithm is implemented as part of the `aer1217_ardrone_simulator` package.

2.1 Code Structure for Phase 1

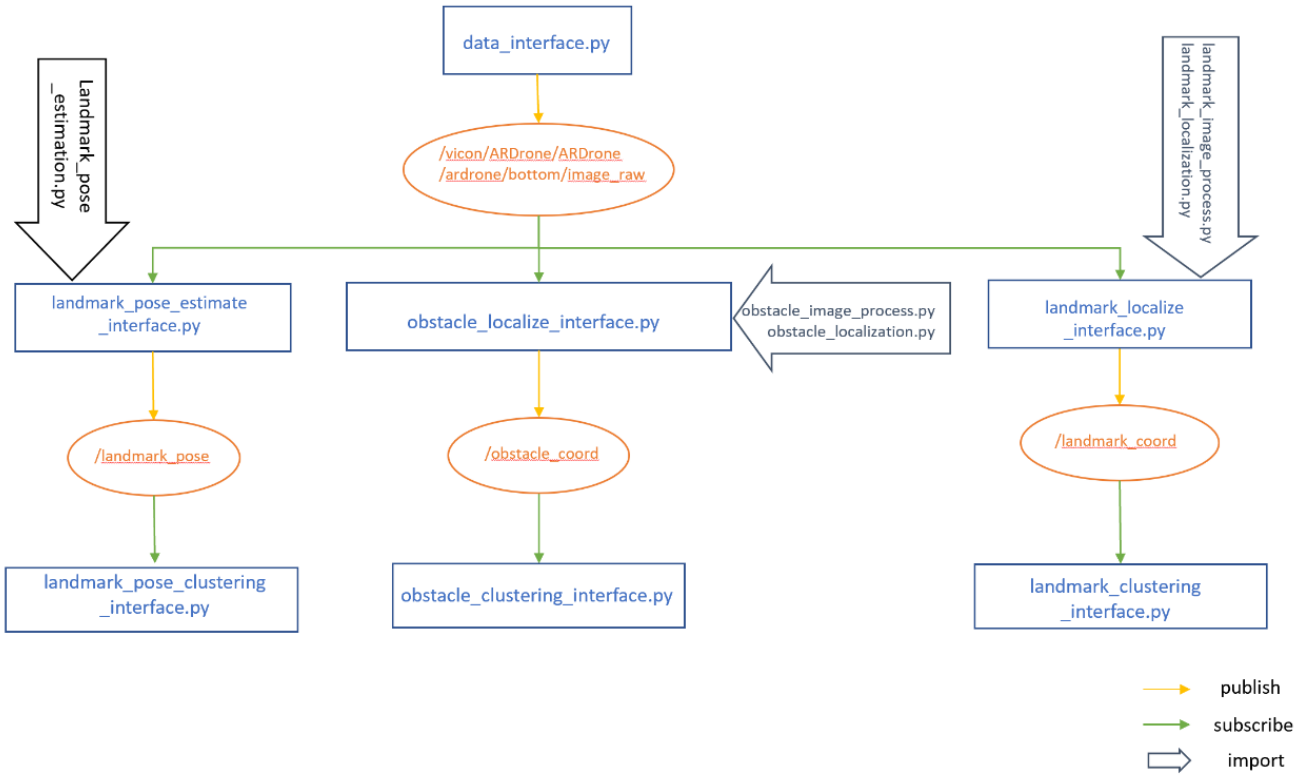

Figure 1: ROS code structure for the `localize_object` package.

The structure of the `localize_object` package is shown in Figure 1 above. Their functionalities and topics published/subscribed are described below:

Data Publishing: The ROS node `data_interface` publishes VICON data and images from `FinalProject2021-04-09-21-13-44.bag` to `/vicon/ARDroneCarre/ARDroneCarre` and `/ardrone/bottom/image_raw` respectively.

Obstacle Localization: The node `obstacle_localize_interface` subscribes images and VICON messages published by the `data_interface` node. Images are then processed by `obstacle_image_process.py` to detect obstacles. The coordinates of the center points and a reference point located on the boundary of detected obstacles are then computed using `obstacle_localization.py`. The reference point is used to calculate the diameter of obstacles by calculating the doubled distance between the coordinates of the reference point and the center of the corresponding obstacle in the inertial frame. Obstacle coordinates and diameters are then published to the topic `/obstacle_coord`.

Obstacle Clustering: The node `obstacle_clustering_interface` subscribes coordinates and diameters of detected obstacles from `/obstacle_coord`. The clustering process is based on the distance of each detection. Coordinates of each detections are then averaged to estimate the coordinate of each obstacle.

Landmark Localization: Similar to the functionality of `obstacle_clustering_interface`, the `landmark_localize_interface` node subscribes images and VICON messages published by `data_interface` and publish landmark coordinates to topic `/landmark_coord`. Image processing and calculations of coordinates of landmarks are achieved in `landmark_image_process.py` and `landmark_localization.py` respectively.

Landmark Clustering: Similar to the functionality of the `obstacle_clustering_interface` node, `landmark_clustering_interface` subscribes `/landmark_coord` to obtain coordinates of detections and estimates coordinates of each landmark by applying a distance-based clustering.

Yaw Angle Estimation for Landmark: The `landmark_pose_estimate_interface` node subscribes images and VICON messages published by data_interface.py and publish landmark pose to the topic `/landmark_pose`. The yaw angle of landmarks is computed by importing the helper class `landmark_pose_estimation`.

Yaw Angle Clustering for Landmark: The `landmark_pose_clustering_interface` node subscribes to estimation of yaw angles for all detections from `/landmark_pose` and estimates yaw angles of each landmark by applying a distance-based clustering.

2.2 Code Structure for Phase 2

All nodes and classes for generating the optimized path are shown in Figure 2 below.

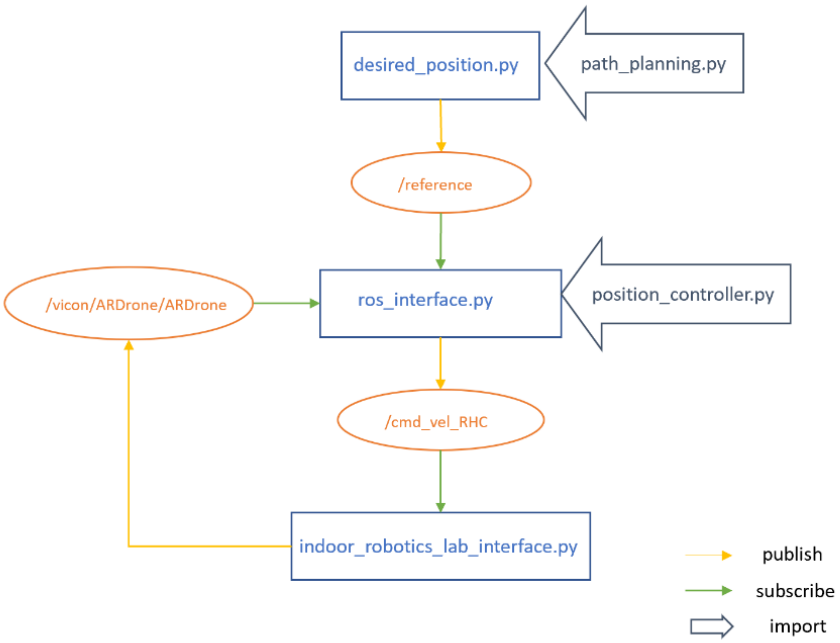

Figure 2: ROS code structure for the `aer1217_ardrone_simulator` package

`desired_position`: This node publishes the desired positions of the drone to topic `/reference`. Desired positions are computed by importing the RRT path generator `path_planning.py`.

`ros_interface`: This node subscribes to VICON data and desired positions. VICON data is used for position and altitude feedback. It also publishes pose commands to `/cmd_vel_RHC` based on the desired positions received. Position commands are calculated using `position_controller.py`.

`indoor_robotics_lab_interface`: This node subscribes pose commands of the drone. Then the commands for each motor of the drone are computed and applied. The node publishes the current location of the drone to `/vicon/ARDrone/ARDrone`.

3. Procedure

3.1 Phase 1

Pixel locations of obstacles and landmarks are obtained by using image processing techniques. At the same time, the pixel location of a reference point is picked up on the boundary of each obstacle. They are then converted into 3D world coordinates by using the inverted camera intrinsic matrix. Unlike photos taken in the real world, photos do not have to be unwrapped to eliminate the distortions. The diameter is then calculated by doubling the distance between the coordinate of the center of each obstacle and the corresponding reference point. The pose of the landmarks is determined in the pose estimation processes using a SIFT detector and homography matrix finding algorithms.

3.2 Phase 2

The optimized path is generated by using the RRT path planning algorithm. Implementations of the position controller from Lab 2 is re-used to generate commands for the drone to follow the desired path.

4. Algorithm Overview

4.1 Algorithms in Obstacle Localization

Based on shapes and colors, obstacles coordinates are found by applying the image processing functions to the images taken by the drone. Firstly, the function `cv2.threshold` is applied to binarize all input images. Secondly, a Gaussian blurring process is used to filter out small, dotted noises. After that, `cv2.findContours` is used to find the contours on the images. The coordinate of the center of each contour is then computed and compared with the coordinate of the center of the minimum enclosing circle of the contour obtained using `cv2.minEnclosingCircle`. If the distance of the centers of the contour and the minimum enclosing circle is smaller than a pre-define threshold, the contour is considered a circle and a potential obstacle. A threshold on the radius of circles is also applied to filter out small circular dots that affects the accuracy of the estimations.

For calculating the diameter of each obstacle, a point that is one radius away from the center of each obstacle, which is a point located on the boundary circle of each obstacle, is picked up as the reference point for calculating the diameter of each obstacle later. The transformations for converting the 2D pixel locations of the center of the obstacle to the coordinate in the inertial frame is also applied to the reference point. Once the coordinates in the inertial frame of both points are obtained, the diameter of each obstacle can be computed by calculating the doubled difference between the coordinates of center of the obstacle and the reference point in the inertial frame.

The inverted camera intrinsic matrix, K is then applied to convert the pixel locations, (x_s, y_s) of the centers of detected circles, which are potential obstacles, to normalized 2D coordinates, (x_n, y_n). It is worth noticing that the 3D world coordinates in camera frame can then be computed by combining the 2D coordinates and the height of the drone, z, from the VICON message. The following equations shows this process for this conversion:

Normalized 2D coordinates in camera frame:

$$\begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix} = \begin{bmatrix} 604.62 & 0.0 & 320.5 \\ 0.0 & 604.62 & 180.5 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

3D coordinates of the obstacles in camera frame:

$$r_C^{PC} = \begin{bmatrix} x_n \\ y_n \\ 1 \end{bmatrix} \cdot z = \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$$

Therefore, 3D coordinates of obstacles in inertial frame can be derived as followed:

$$r_E^{PE} = \begin{bmatrix} x_E \\ y_E \\ z_E \end{bmatrix} = T_{EC} r_C^{PC} = T_{EB} T_{BC} r_C^{PC} = T_{EB} \begin{bmatrix} 0.0 & -1.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0125 \\ 0.0 & 0.0 & -1.0 & -0.025 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}^{-1} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}$$

The transformation matrix, T_{EB} can be computed using the associated VICON information of the image:

$$T_{EB} = \begin{bmatrix} R & l \\ 0 & 1 \end{bmatrix}$$

Where R is the rotation matrix and l is the translation vector and they are expressed as followed:

$$R = quaternion\_matrix\left(\begin{bmatrix} vicon.transform.rotation.x \\ vicon.transform.rotation.y \\ vicon.transform.rotation.z \\ vicon.transform.rotation.w \end{bmatrix}\right)$$

$$l = \begin{bmatrix} vicon.transform.translation.x \\ vicon.transform.translation.y \\ vicon.transform.translation.z \end{bmatrix}$$

4.2 Algorithms on Landmark Localization

Landmarks are localized by following similar method as the obstacle localization process. The function `cv2.Canny` is firstly used to binarize and detect edges of the images. Then, Gaussian blurring methods are applied to blurring the edges to make the features of potential landmarks continuous. Then, `cv2.findContours` is applied. The coordinate of the center of each contour is then computed and compared with the coordinate of the center of the minimum enclosing rectangle of the contour obtained using `cv2.minEnclosureCircle`. Since the landmarks are rectangular, the contours with curves are expected to be excluded. As opposed to the obstacle detection process, if the distance of the centers of the contour and the minimum enclosing circle is greater than a pre-define threshold, the contour may be a potential landmark. This process also excludes the shadows of drones since there are curves in the boundary of the shadow of the drone. In addition, `cv2.minAreaRect` is used to find the minimum rectangular that fits each contour. If the minimum fit rectangular is a square, which means that the ratio between the width and the height is smaller than a pre-defined threshold that is close to 1, the contour is considered a potential landmark and the center of the minimum fit rectangular is treated as the pixel location of this potential landmark. The inverted camera intrinsic matrix is then applied to realize the conversion to 3D coordinates in the inertial frame by following the same procedures and formulae described in Section 4.1.

4.3 Algorithms on Landmark Pose Estimation

Rotational poses of landmarks are also important in this project. Since photos of landmarks is placed on the ground, only the yaw angle is non-zero and needed to be estimated. To compute the yaw angles in the pose of each landmark, a SIFT detector is firstly utilized to detect key points and descriptors of both the input images and the four raw landmark images. Before starting matching features of the input images to the raw landmark images, the same image processing scheme used in the landmark image processing scripts, `landmark_image_processing.py`, is applied to detect landmark to ensure the orders of detected landmarks of both the landmark localization and the yaw angle computing process are the same for matching corresponding information with the correct landmark names.

Then, a FLANN (Fast Library for Approximate Nearest Neighbors)-based feature matcher provided by OpenCV is applied to associate features of the input images and each raw landmark image with k=2 for the KNN match process. FLANN is a collection of algorithms that fast compute the nearest neighbor searching processes for a large dataset. In this case, "`FLANN_INDEX_KDTREE`" algorithm is applied. For the two neighbors obtained from the feature matching processes, a Lowe's ratio test is applied to check the validity of each match. The match is considered valid if the following inequality is satisfied:

$$distance\ 1 < Lowe's\ Ratio \cdot distance\ 2$$

Where distance 1 is the distance to the first neighbor and distance 2 is the distance to the second neighbor.

The inequality implies that the first neighbor is at least closer than the second neighbor by the Lowe's ratio, which indicates that the first neighbor is significantly closer comparing to the second neighbor, and the first match is considered a valid match.

Then, the `findHomography()` function from OpenCV is applied to the valid matches to compute the homography matrix, which represents the transformation between the landmarks in the two images and is used to find the rotational angle between the two images. The rotation angle obtained from the homography matrix is treated as the yaw angle of the landmark in the input image.

4.4 RRT Path Planning Algorithm

1) Initialize: Initialize a tree containing the coordinate of the current node, the parent node, and the cost of each node to the start point. The start point is added to the tree in the initialization process.

For each iteration:

2) Sampling: Find the next sampling point by pick up a point on the searching area (in this case, [0, 9, 0, 9] since the range on the x-axis and y-axis are both from 0 to 9) for searching towards that point. Denote p as the probability of randomly generating the sample for the next step, then the probability that the sample point of the next step is set to be the final target is (1-p). The purpose of random sampling is that since obstacles exist, if the algorithm always search towards the final target and the searching path is blocked by an obstacle, the searching process will not be able to continue.
3) Find Nearest Neighbor: Find nearest point in the tree to the sample point by going through the tree and calculate the distance between the sample point and each node of the tree.
4) Find Next Node: A step length, which is the radius of the searching range, is pre-defined for the algorithm. If the distance between the nearest point in the tree and the sample point is smaller than the step length, the sample point is taken as the new node to be added into the tree and set the nearest point as its parent node. The cost of the new node is computed as the sum of the cost of the nearest node and the distance between the new node and its parent, the nearest node.
5) Check Node Collision: Check if the new node falls in the range of any obstacle and its buffer zone or out of the range. If the node falls in an obstacle or its buffer zone, the node is invalid.
6) Check Path Collision: The distance between the line section between the new node and its nearest neighbor and the center of each obstacle is computed. If the distance between the line section and the center point of an obstacle is smaller than the radius of that obstacle and the closest point on the line formed by the new node and its nearest neighbor to the center of the obstacle falls on the line section (not on the extension of the line section), the path is considered collided with that obstacle.
7) Check Neighbors to Update Cost: For the newly added node, the algorithm goes through the tree to find the parent that will lead to smaller cost (cost = parent['cost'] + dist(new node, potential parent)). If a potential parent node with lower cost is found and the path between the new node and the potential parent does not collide with any obstacle, the parent of the new node is updated as the potential parent. The cost is also updated to the corresponding value.
8) Get Path: After a pre-defined number of iterations, the tree building process will be terminated. If the final target is not in the tree, the algorithm goes through the tree to check if there is any node within

the search range of the final target. If there is no neighbor found for the final target, there is no available path existing. Otherwise, the final target is connected to the nearest neighbor and the tree is back tracked by checking parents towards the start point. The list containing all points obtained by back tracking the tree is then reverted to obtain the path from the start point to the target.

9) Compute Path for Yaw Command: The points between the start point and the target are counted. Then, `linspace` is applied to generate a yaw path from the expected yaw value of the start landmark (pose + 90 degrees) to the expected yaw angle of the targeting landmark.
10) Add Intermediate Points: To smoothen the motion of the drone, intermediate points are added to each section of the path.

4.5 Discussion on Parameters Setting of RRT

- Random sampling probability: the probability to random sample the direction of search at the next step. If the probability is large, then algorithm will have a larger capability of exploration and thus result in a higher possibility of finding a valid path. However, the cost of the generated path may increase since a larger portion of searches are conducted towards a random direction.
- Maximum iteration: the maximum steps in the tree building process. A larger number of iterations increase the probability of finding the target and a valid path but also increases the computational cost and searching time.
- Searching range (step length): the maximum searching step for each iteration. A larger step length increases the possibility of finding the target and a valid path. However, the path will be less smooth, and the cost of the generated path may increase.
- Width of buffer zone: the width of buffer zone around each obstacle. It uses to keep the drone away from obstacles to reduce the possibility of colliding into any obstacle. A larger buffer zone reduces the probability of drone of collapsing into obstacles. However, larger buffer zones increase the difficulty of finding a valid path since the available space for placing samples decreases.

5. Simulation Result

Obstacle and landmarks' coordinates, diameters and pose are found in the `localize_object` package.

Table 1: Obstacle Center Coordinates and Diameters in Inertial Frame

Obstacles	Central Point Coordinates [m]	Diameters [m]
1	(1.1018794698099934, 7.975980010083926)	0.2887832097789294
2	(4.520090363242409, 7.583265622456868)	0.5949794072093386
3	(8.262144497462682, 8.394538034711566)	0.40550610167639595
4	(5.999486982822418, 3.6608902335166933)	0.7908300936222077
5	(8.64246536254883, 1.7197444105148316)	0.39559750735759736
6	(2.3262507915496826, 3.685031943851047)	1.0071580277548895
7	(3.0342175563176474, 4.350731809933979)	0.5096020599206289

Table 2: Landmark Center Coordinates and Yaw Angle

Landmarks	Central Point Coordinates [m]	Yaw Angle [rad]
Nathan Phillip Square (3)	(1.9134980865887232, 6.632558992930821)	2.851107433578201
Princes Gates (4)	(8.734049844741822, 4.789200067520141)	-0.47667693164104996

Casa Loma (1)	(7.144379251533085, 5.809121251106262)	0.615793492321352
CN Tower (2)	(3.230806689513357, 1.426224526606108)	-1.323698549743104

The detected obstacles and landmarks are plotted in Figure 3 below:

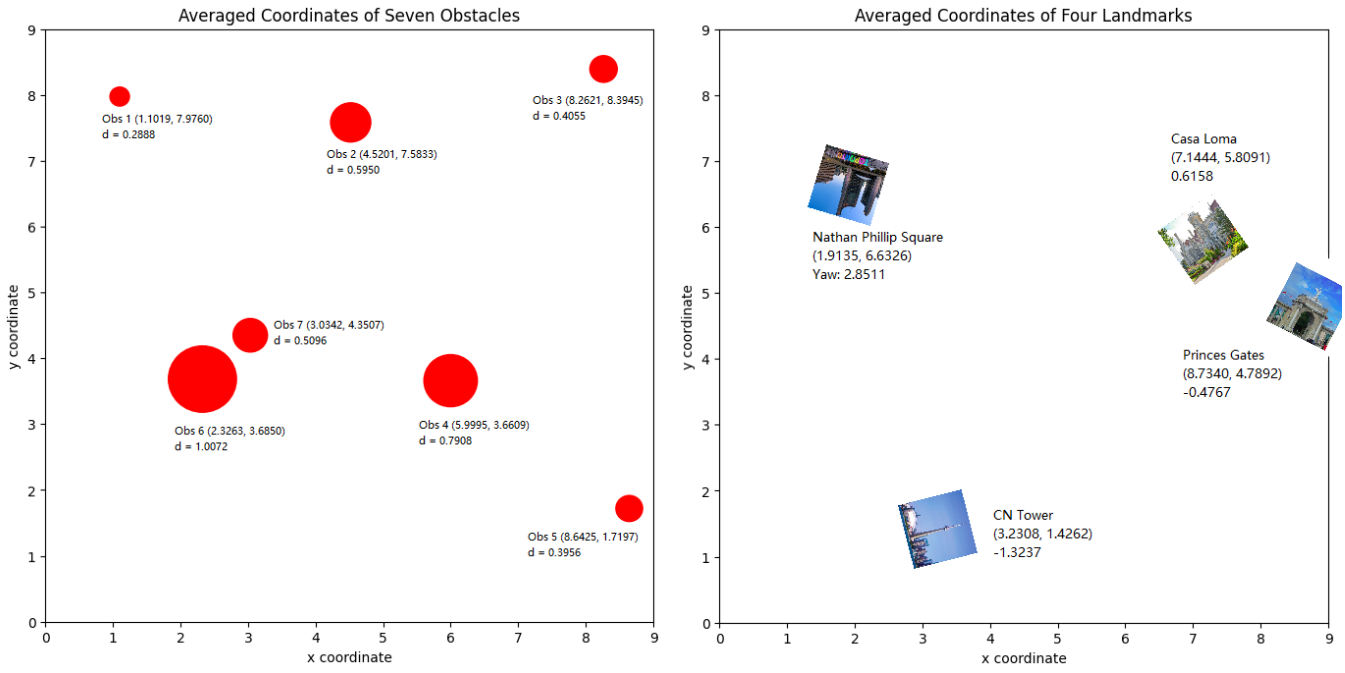

Figure 3 Obstacles and Landmarks

A graph of the optimized path that reaches landmarks while avoids the obstacle is shown in Figure 4. In this case, the sequence (3, 2, 1, 4) is applied.

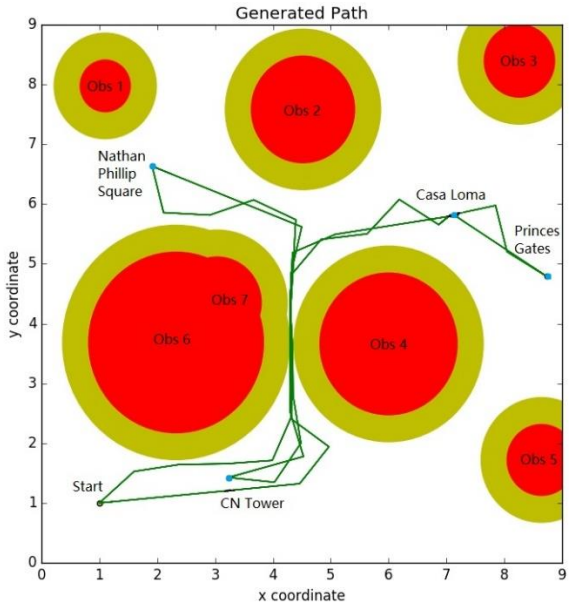

Figure 4: Optimized path planning

Red circles are obstacles and yellow regions represent buffer zones of obstacles, which prevent the drone from crashing into the obstacles. Polylines in green represent the path of the drone. As we can see, the drone is placed at (1, 1) at first. Then, the drone goes between the two largest obstacles No. 6 and 4 and

reaches Nathan Phillip Square. After that, the drone goes between Obstacle 6 and 4 again to reach CN Tower. The drone then goes between Obstacles 4 and 6 one more time and arrives at Casa Loma and Princes Gates. The drone finally goes back to the starting point via the tunnel between Obstacle 4 and 6.

Graphs of actual positions, desired positions, and errors between them in x, y, z directions and yaw angles for the drone are shown in Figure 5:

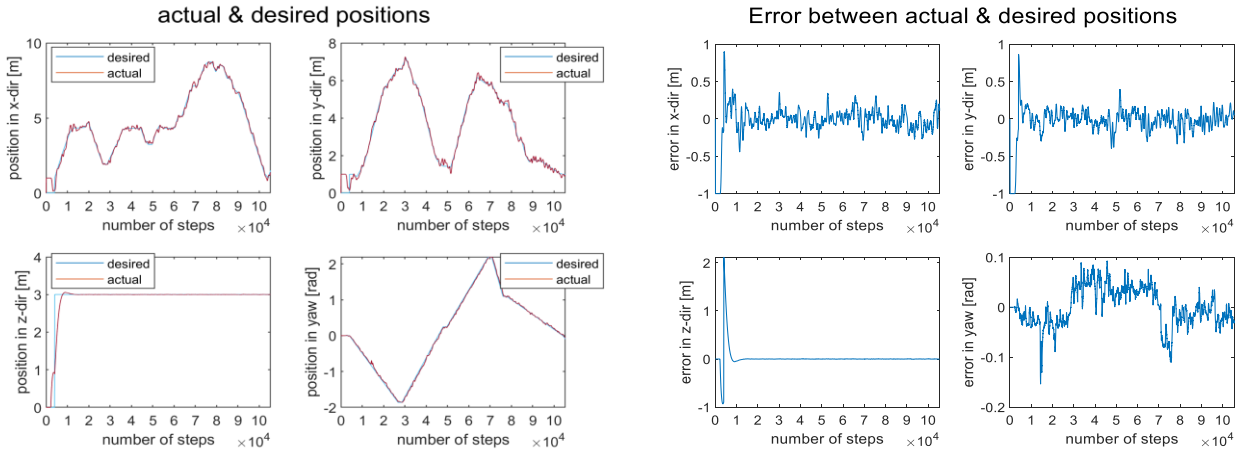

Figure 5: Drone poses (VICON, Desired, Error)

As we can see from figures above, the performance of the position controller is as expected during the simulation. Actual and desired positions are always approximately the same. Spikes are observed at the beginning of the error plots in x, y, and z directions and then stabilize at around 0. Yaw error is around 0 at all steps. The error spikes in x, y, and z directions exist at the beginning since the drone is placed at (0, 0) at the very beginning instead of the supposed start point (1, 1) when waiting for the path generator.

6. Discussion

6.1 Challenges, Solutions and Enhancement When Facing Similar Problems:

- Incorrect number of clusters for obstacles or landmarks may be obtained if detections of obstacles and landmarks are not accurate enough. For example, a detection for an obstacle may sometimes be clustered as a new obstacle since the distance of the detection to the existing average of the supposed obstacle is greater than the pre-defined threshold.
  Solution: Increase the tolerable distance for a cluster. Also, reduce the data updating rate from `FinalProject2021-04-09-21-13-44.bag` to reduce the probability of time lag between image and corresponding VICON data to enhance the accuracy of the coordinate of each detection.
  Future Enhancement: Implement an algorithm to check the distance among the average coordinates of each cluster. If the distance between any two clusters is smaller than the distance threshold, the two clusters will be merged as one cluster (treat them as one obstacle/landmark).

- Sometimes irrelevant objects, such as the shadow of the drone, are misclassified as landmarks.
  Solution: Increase the threshold for the minimum tolerable distance for the between the center of the mininum enclosing circle of contours and contour centers to get rid of any contours with arcs. Then, only contours that are close to a rectangular are selected. Since the four corners of the shadow of the drone are arcs, the shadow of the drone will then be excluded.

- Small noises and paired contours are recognized as obstacles.
  Solution: Add a threshold and only consider contours with the number of parts greater than the threshold to ensure that noises will not be recognized as obstacles/landmarks by mistake. The issue and solution is shown in Figure 6. In this case, since the left contour has only 2 parts, even the distance between the center of the contour and the center of its minimum enclosing circle is small, it cannot be treated as a circle or a valid obstacle detection. On the contrary, it is obvious that the right contour with more than 10 parts represent a valid detection.

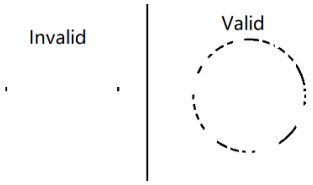

Figure 6 Contour Parts Threshold

- The drone may crash into obstacles due to the control overshoot and a very close path to obstacles.
  Solution: Add a buffer zone around obstacles to prevent the drone from entering the surrounding areas of obstacles. However, this method may reduce the path finding speed and increase the failure probability of finding a valid path.
  Future Enhancement: Tune PID gains or apply modern control techniques (LQR, etc.) for a more robust control system to reduce overshoot.

- RRT algorithm is slow when generating paths.
  Solution: Parameter tunings can be applied. For example, the step length can be increase and the number of iterations can be reduced to speed up the algorithm and increase the probability of reaching the target. But the path may be less optimal. Parameter tuning of RRT is described in Section 4.5.
  Future Enhancement: We can store the neighbors of each node to a list and add the list to each node as a feature in the tree to reduce the searching time. Also, since the collision checks for paths are time-consuming, we can keep the search range/step length small and get rid of path collision checking since if both the sample node and its neighbor do not collide with obstacles, the probability of any part of the small line section between them to collide with obstacles is close to 0 given the existence of buffer zones around each obstacle.

6.2 Things that do not go well:

- Generally, the obstacle and landmark localization algorithms work well but the yaw angle calculation algorithm for landmark pose estimation takes longer time to complete, which results in the algorithm to use less images than the localization algorithms and thus less accuracy on estimating the yaw angles. In the future, lower data publishing rate can be used. Also, we can ply the .bag file for several times or try other feature matching algorithms to enhance the accuracy of yaw angle estimations.
- It is hard to balance the optimality, generating speed and the probability of the drone to collide into obstacles when generating paths using the current implementation of RRT. For the current group of parameters, it is guaranteed that the drone does not collide into obstacles, but the optimality of the path may be sacrificed.