Project Report - Autonomous Drone Geodashing

Team E: Aditya Jain, Apurv Mishra, Praharsha Abbireddy

1 Introduction

This project focuses the use of almost the entire robotics pipeline - perception, planning and control - to play the game of autonomous drone geodashing. The task is to reach a set of landmark locations in a simulated world in the shortest time possible.

First, data collection is done in a gaezbo world containing circular targets and landmark images on the ground. With this data, our **perception** module does georeferencing for the circular targets and landmark locations to build an occupancy map. Next, our **planning** module uses the occupancy map and a given landmark visit sequence to find an optimal path. Finally, the designed **controller** generates the required control input for the drone to accurately traverse the trajectory.

2 Overview of the Robotic System

Figure 1 illustrates the different steps and processes in our pipeline with the outputs at each step.

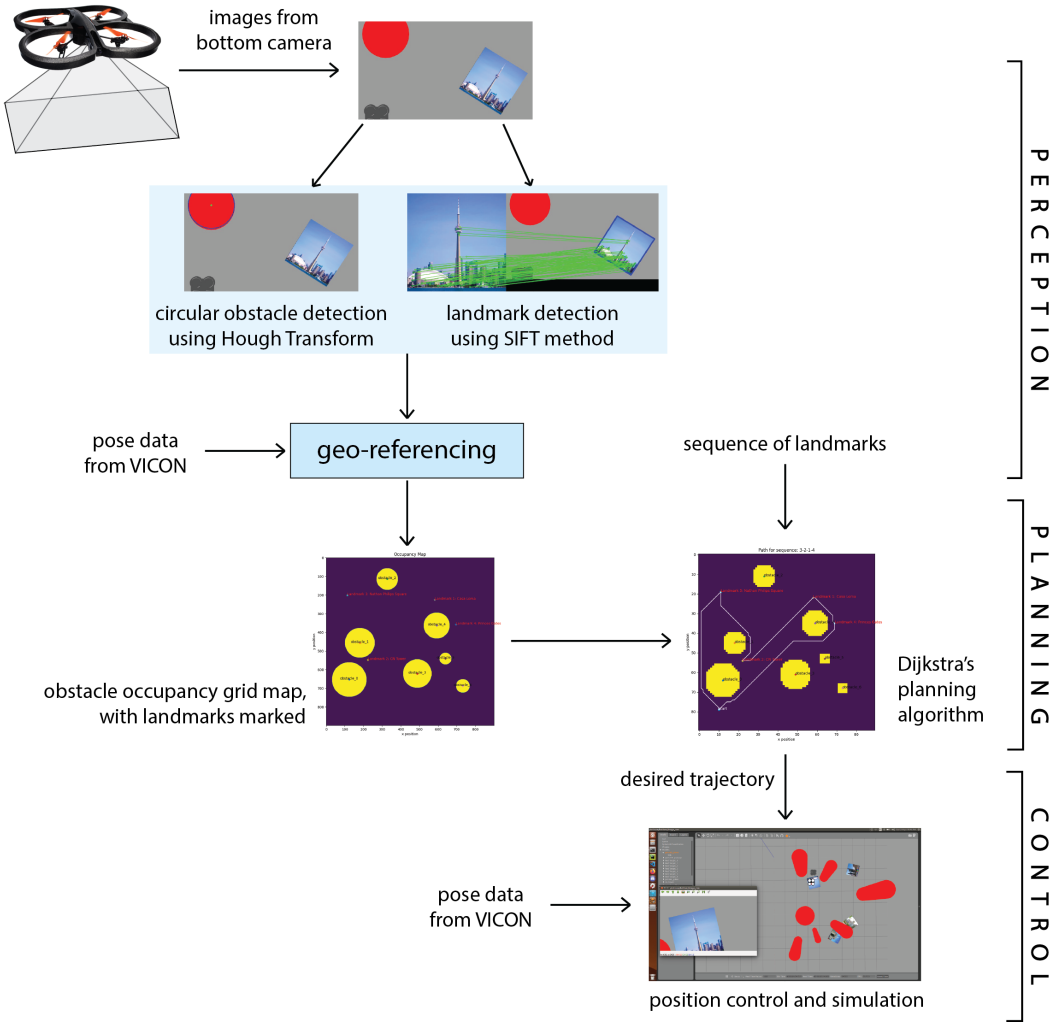


Figure 1: Overview of the robotic system

The provided data - images collected from bottom-facing camera and pose information from VICON system - is extracted from the bag file. We use computer vision techniques to estimate the positions of obstacles and landmarks, along with their sizes and orientation respectively. After completing georeferencing, an occupancy grid map is generated using the obstacle data and landmarks are marked at their respective positions. This map serves as input for the dijkstra's algorithm responsible for planning an optimal trajectory for a given sequence of landmarks to visit. The last step is the design of a PD controller to control the motion of the quadrotor in the gazebo simulation to follow the trajectory generated by the path planner.

3 Perception: Georeferencing

3.1 Image and Pose Data from Bag file

During simulation experiments for data collection, the data published on */ardrone/bottom/image raw* and */vicon/ARDroneCarre/ARDroneCarre* topics is recorded. The bag file is then used to fetch image and pose data for post-processing.

The bag file recorded 2,141 images and 21,831 pose messages. The bag file is then played in a time-synchronized fashion to save the images and corresponding pose (position and orientation) of the drone at each time instant of image capture.

3.2 Circle Detection using Hough Transform

To detect circles in the images, we use Hough Circle Transform [5] from OpenCV library. After a few iterative runs, the following parameters of *cv2.HoughCircles*() [6] are found to be optimal for the task: *minDist*=50, *param1*=40, *param2*=25, *minRadius*=25 and *maxRadius*=100.

For each image in the bag file, the red channel is extracted and converted into a binary image by thresholding the grayscale image with a value of 250. Following that, an averaging blur filter of kernel size 5x5 is applied to the binary image. This minimizes the detection of false positives. This processed image is then passed to the *cv2.HoughCircles*() function, which returns the circle centres and their corresponding radii. A total of 330 out of 2,141 images had circle detections. Figure 2 shows a few samples from the detection algorithm.

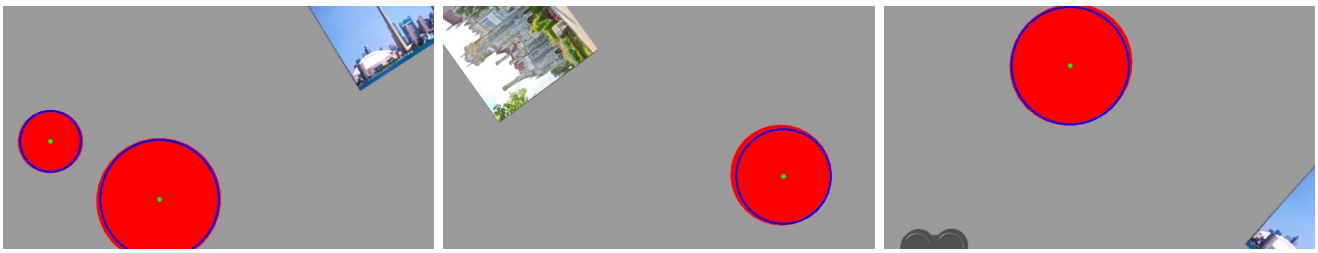


Figure 2: Examples of Circle Detection

3.3 Landmark detection using SIFT

There are four landmarks to be detected - CN Tower, Nathan Philips Square, Casa Loma and Princes' Gates. To detect the landmarks in the images, we use the Scale-Invariant Feature Transform (SIFT) [7] from the OpenCV library. As we are provided with the original landmark images, the SIFT algorithm is used to detect and compute matching keypoints between the original image and

the images fetched from the bag file. We set a minimum threshold of 75 keypoints to be matched for considering a correct classification.

The keypoints matching process of each camera image and landmark forms a total of 8,564 pairs. We use the k-dimensional tree [1] data structure from the Fast Library for Approximate Nearest Neighbors (FLANN) [4] for matching. We find top two candidates from each frame for each keypoint in the original landmark image. The good match among these is decided as per the Lowe's ratio test for both distances where $(distance1) < 0.5 * (distance2)$. Here, distance is the difference between positions of the original keypoint and the two candidates, with the factor of 0.5 decided after a few iterative runs.

Next, we check if the number of good matches is greater than the threshold of 75. If so, the keypoints are extracted from both images and a 3×4 transformation matrix, M, is calculated using the $cv.findHomography()$ [3] function from OpenCV library. The position of landmark in the image is taken as the mean of the matched keypoints and the orientation is calculated from the rotation part of M using the equation $\psi_{landmark} = -tan^{-1}\left(\frac{M_{2,1}}{M_{1,1}}\right)$. A landmark is detected for a total of 445 times in 2,141 images. Figure 3 shows a few samples from the detection algorithm.

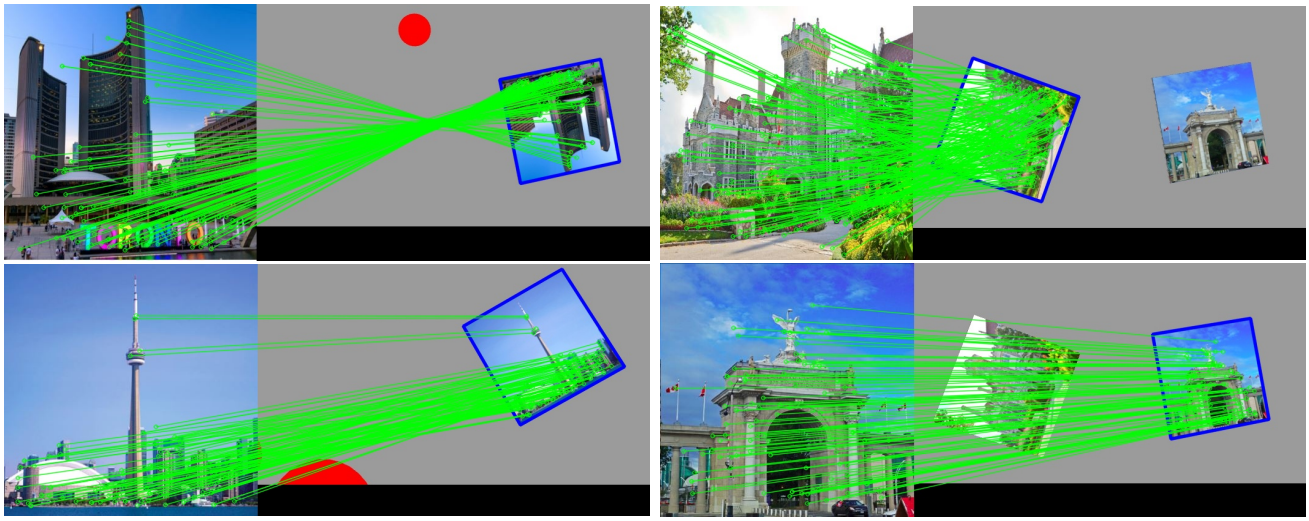


Figure 3: Examples of Landmark Detection and Matching

3.4 Georeferencing Algorithm

After the targets are detected in the images, the pixel locations of the centres has to be transformed to VICON reference frame using drone's synchronized pose information. Below is the sequential procedure:

1. **Transformation from world to camera frame**:
   Vehicle's origin in camera frame, $v_{camera} = v_{body} \times T_{CB}$,
   where $v_{body} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, $v_{camera} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix}$ and $T_{CB} = \begin{bmatrix} 0.0 & -1.0 & 0.0 & 0.0 \\ -1.0 & 0.0 & 0.0 & 0.0125 \\ 0.0 & 0.0 & -1.0 & -0.025 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$
2. **Normalized Camera Coordinates**: The normalized camera coordinates of the vehicle's origin, $v_{normal} = [x_c/z_c,\ y_c/z_c,\ 1]^T$

3. **Pixel Coordinates Transform**: Finally, the vehicle's origin in image frame, $v_{image} = K \times v_{normal}$, where camera matrix $K = \begin{bmatrix} 604.62 & 0.00 & 320.5 \\ 0.00 & 604.62 & 180.5 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$ and $v_{image} = \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$

4. **Distance between circle centre and vehicle origin**: The euclidean pixel distance $dist_{pixel}$ is calculated between the vehicle origin (x_i, y_i) and target centre (x_c, y_c). Only circles that are less than 275 pixels from vehicle origin and landmarks less than 400 pixels from vehicle origin are considered for further processing. These values are chosen after iterating over a range of values and choosing the one which gave best results.

5. **Target Location in Inertial Frame**: First, the euclidean pixel distance $dist_{pixel}$ is converted to metres $dist_m$ using height of the drone in metres, $pose_z$, and the FoV parameters of the bottom camera: $dist_m = dist_{pixel}\,(2\tan(64\,\text{deg}) \times pose_z)/\sqrt{640^2 + 360^2}$

   Then, the angle between the positive x-axis and the line joining vehicle origin and target centre is calculated: $angle_{target} = -\tan^{-1}((y_c - y_i)/(x_c - x_i))$

   Finally, the target centre in inertial frame is found:

$$x_{c\_inertial} = pose_x + dist_m\, cos(angle_{target} + pose_{yaw}) \tag{1}$$
$$y_{c\_inertial} = pose_y + dist_m\, sin(angle_{target} + pose_{yaw}), \tag{2}$$

   where $pose_x$, $pose_y$ and $pose_{yaw}$ are the vehicle's x position, y position and yaw respectively for the particular image capture.

3.5 Results and Discussion

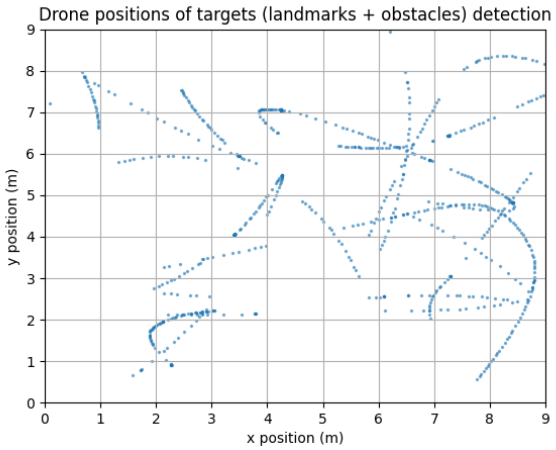
Figure 4a shows the drone positions for each frame in which an obstacle or a landmark is detected correctly. After georeferencing, we calculate the positions of the targets in the inertial frame. The centers of detected circular obstacles are shown in figure 4b which are further refined and clustered using a density-based clustering method called Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [2]. The parameters are tuned to obtain 7 clusters (figure 4c), each corresponding to one obstacle. The final circle centres are the centroid of each cluster and the landmark positions are found by taking mean of calculated positions from all the frames. The final georeferencing results are shown in figure 4d.

The final positions and radii (before scaling up 3 times) for each circular obstacle (in meters) in the inertial reference frame is given below:

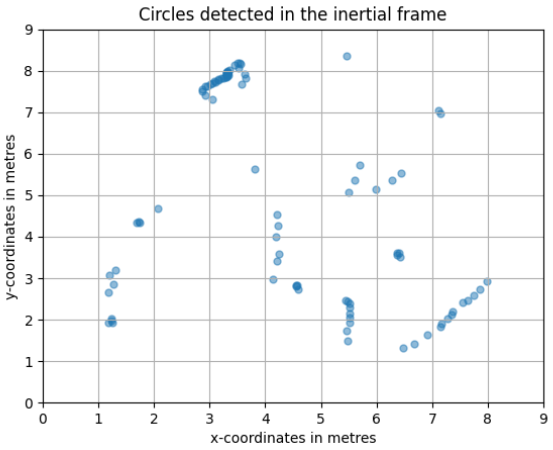
- **Obstacle #1:** x = 1.23, y = 2.46, radius = 0.47;
- **Obstacle #2:** x = 1.80, y = 4.43, radius = 0.45;
- **Obstacle #3:** x = 3.27, y = 7.85, radius = 0.29;
- **Obstacle #4:** x = 4.89, y = 2.78, radius = 0.39;
- **Obstacle #5:** x = 5.92, y = 5.36, radius = 0.40;
- **Obstacle #6:** x = 6.39, y = 3.58, radius = 0.16;
- **Obstacle #7:** x = 7.32, y = 2.12, radius = 0.18;

The final positions (in meters) and orientation (angle from positive y-axis in the inertial frame, in radians) for each landmark is given below:

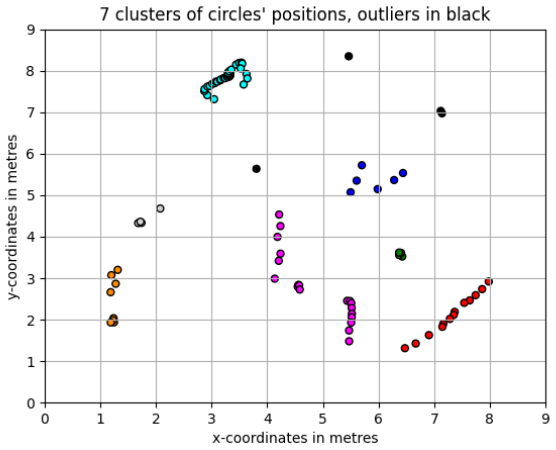
- **Casa Loma (landmark #1):** x = 5.82 y = 6.74, orientation = 0.60;
- **CN Tower (landmark #2):** x = 2.23, y = 3.52, orientation = -1.34;
- **Nathan Philips Square (landmark #3):** x = 1.14, y = 6.98, orientation = -3.37;
- **Princes' Gates (landmark #4):** x = 6.94, y = 5.41, orientation = -0.50;



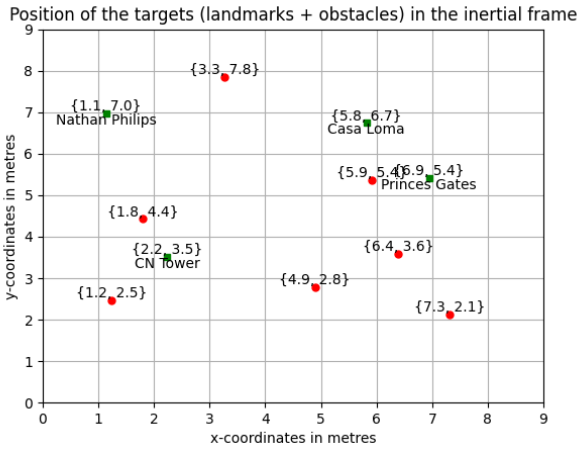
(a) All drone positions containing targets



(b) Obstacle centers post-filtering



(c) Clustering using DBSCAN



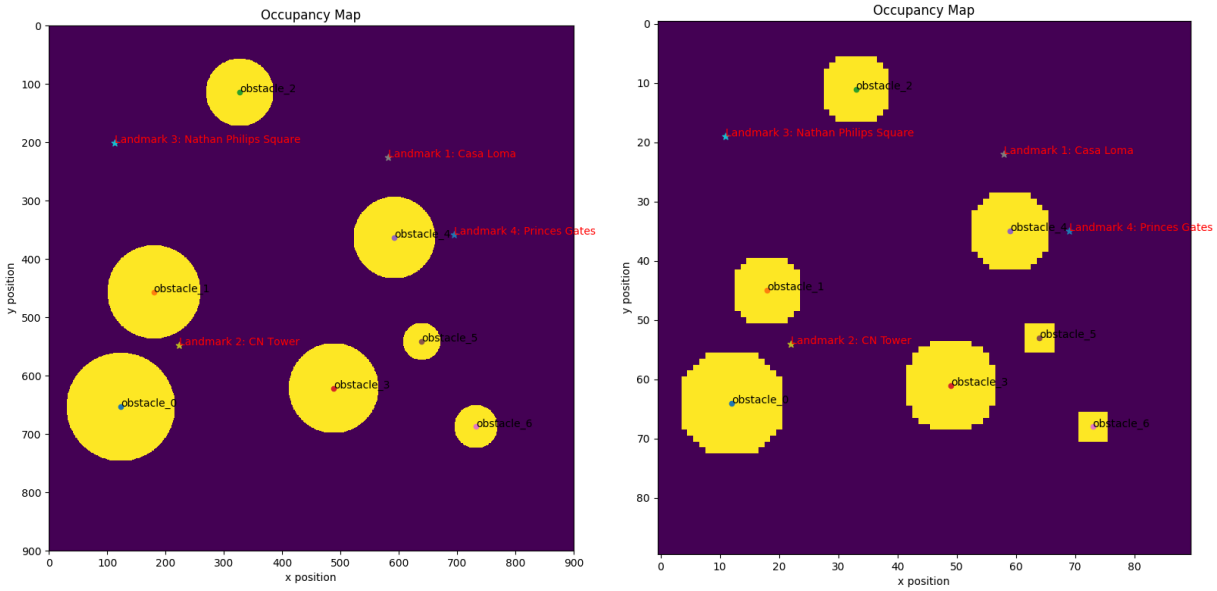
(d) Final landmark and obstacle positions

Figure 4: Results from georeferencing: obstacle and landmark positions

3.6 Occupancy Grid Map

After the circle centres and their corresponding radii are found in the world reference frame, an occupancy map is built. An occupancy map is a 2D grid by definition in which each of its cell has a binary value - 0 for free and 1 for occupied. The area in the map occupied by the circles are assigned the value of 1 whereas the rest of the map has a value of 0. Figure 5 shows the occupancy grid along with the position of the landmarks in the map.

Figure 5a is a map of size (900, 900) which offers a resolution of 1cm, whereas, figure 5b is a (90, 90) map with a resolution of 10cm. For the purpose of planning, we use the downscaled (90,90) version of the map, the reason for which will be discussed in section 4.2.



(a) (900, 900) Map

(b) (90, 90) Map

Figure 5: Occupancy grid map. Yellow regions are occupied by the obstacles and purple locations are free.

4 Trajectory Planning: Dijkstra's Algorithm

A trajectory planner finds a feasible path between two points in a map. For this project, we implemented a search-based planning algorithm called as Dijkstra's algorithm. It is a optimal path planner, i.e., it finds the shortest path between two points in the map, if one exists. In section 4.1, we discuss the implementation of the algorithm and the results in section 4.2.

4.1 Algorithm

Algorithm 1 describes the Dijkstra's algorithm. Beginning with the start position x_{start} in the priority queue Pr_Q, the first element of Pr_Q is deleted from the queue and its neighbouring cells that haven't been visited yet are added in Pr_Q. The queue is sorted based on *cost-to-come* of the cells from x_{start}. The process is repeated with the sorted queue until the goal position x_{goal} is reached or Pr_Q is empty. When x_{goal} is found, the shortest path is backtracked from x_{goal} to x_{start}.

4.2 Results and Discussion

Planning for a map of size (900, 900) has a high computational cost. It took hours to find an optimal route between just two landmarks. Since we wanted to have a system of on-demand sequence of landmarks to visit, we chose to plan in a (90, 90) map. Additionally, we maintain a padding of 3 cells around the obstacles while planning. This is to accommodate any uncertainties in the control input and avoid hitting obstacles during path tracking in simulation.

Algorithm 1 Dijkstra's Algorithm

$Pr\_Q.Insert(x_{start})$ and mark x_{start} as visited
while Pr_Q not empty **do**
    $x \leftarrow Pr\_Q.GetFirst()$
    $Pr\_Q.DeleteFirst()$
    if $x = x_{goal}$ **then**
        return success, path $x_{start} \rightarrow x_{goal}$
    end if
    for all $x' \in neighbour(x)$ **do**
        if x' not visited **then**
            Mark x' as visited
            $Pr\_Q.Insert(x')$
        end if
    end for
    $Pr\_Q.Sort(cost\_to\_come)$
end while

Figure 6 shows planning results for four different landmark sequences. For any given sequence, the drone starts from {1,1} in meters and returns to this start location after visiting all the landmarks. For any given sequence, it takes ∼20 sec for planning.

5 Control: PD Controller

After finding the trajectory for the desired sequence of landmark visits, the points are fed into the position controller consecutively, from the start to end. The */vicon/ARDroneCarre/ARDroneCarre* topic provides us with the current state of the quadrotor in the inertial reference frame. The difference between desired state and the current state is used to control the maneuver of the quadrotor. The controllers and equations used are mentioned below:

1. Translation along x and y axes: Proportional Derivative (PD) controller

$$\ddot{x}_{cmd} = k_{p,x}\, x_{error} + k_{d,x}\, (x_{error} - x_{error,old}) \,/\, \Delta t \tag{3}$$
$$\ddot{y}_{cmd} = k_{p,y}\, y_{error} + k_{d,y}\, (y_{error} - y_{error,old}) \,/\, \Delta t \tag{4}$$

   where $\ddot{x}_{cmd}$ = commanded linear acceleration along x-axis, $\ddot{y}_{cmd}$ = commanded linear acceleration along y-axis, $x_{error} = (x_{des} - x_{cur})$, $y_{error} = (y_{des} - y_{cur})$

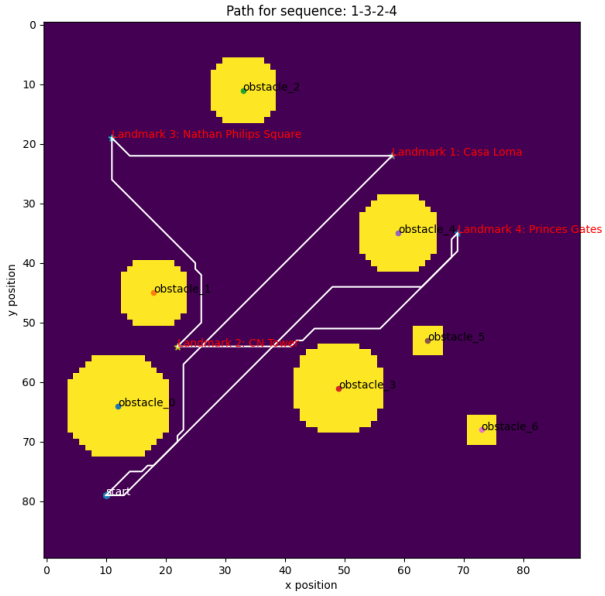
2. Translation along and orientation about z-axis: Proportional (P) controller

$$\dot{z}_{cmd} = k_{p,z}\, z_{error} \tag{5}$$
$$\dot{\psi}_{cmd} = k_{p,\psi}\, \psi_{error} \tag{6}$$

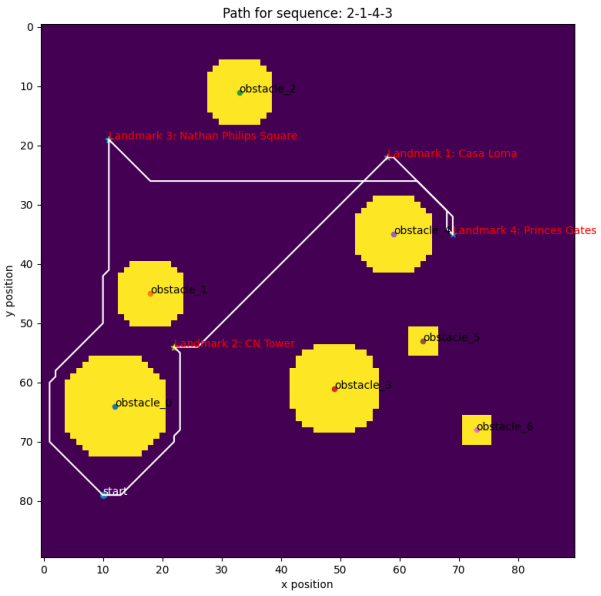
   where $\ddot{z}_{cmd}$ = commanded linear acceleration along z-axis, $\ddot{\psi}_{cmd}$ = commanded angular acceleration along z-axis, $z_{error} = (z_{des} - z_{cur})$, $\psi_{error} = (\psi_{des} - \psi_{cur})$

The tuned gain values are:

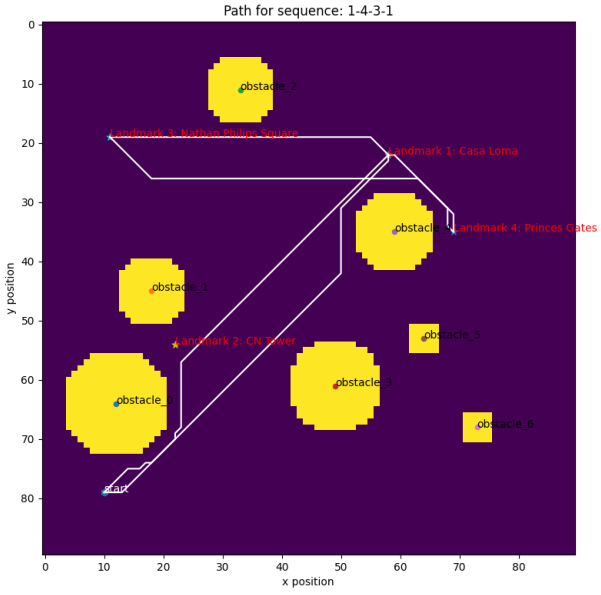
$$k_{p,x} = 4.0,\;\; k_{d,x} = 8.0,\;\; k_{p,y} = 4.0,\;\; k_{d,y} = 8.0,\;\; k_{p,z} = 6.0,\;\; k_{p,\psi} = 4.0$$



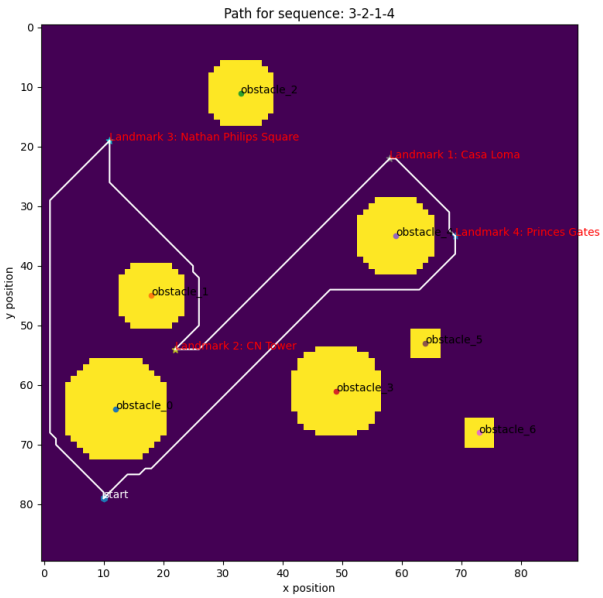
(a) Sequence 1-3-2-4



(b) Sequence 2-1-4-3



(c) Sequence 1-4-3-1



(d) Sequence 3-2-1-4

Figure 6: Planning results for different landmark visit sequences

The current state values from the VICON system and the calculated values from (3) and (4) of linear accelerations are used to determine the commanded roll and pitch using:

$$f_{cur} = \frac{\ddot{z}_{cur} + g}{cos\theta_{cur}\, cos\phi_{cur}}, \quad \phi_{cmd} = sin^{-1}\left(\frac{-\ddot{y}_{cmd}}{f_{cur}}\right), \quad \theta_{cmd} = sin^{-1}\left(\frac{\ddot{x}_{cmd}}{f\, cos\phi_{cmd}}\right) \tag{7}$$

After this, the commanded state values are published to */cmd_vel_RHC* topic in ROS to control the quadrotor in gazebo. The gazebo world is also updated with the pose information of obstacles and landmarks from section 3.5 of georeferencing.

5.1 Results and Discussion

The screenshots of the simulation are shown in figure 7 and the complete simulation video for the sequence 1-2-4-3-1 can be seen here, with start and end positions defined as {1,1} in meters. The desired positions are given in a sequence and the next position is given as an input only when the quadrotor is within a distance of 0.25 meters from the current desired position. Increasing this threshold causes the quadrotor to move faster while decreasing the same reduces the speed significantly. This particular value is chosen after a few test runs.

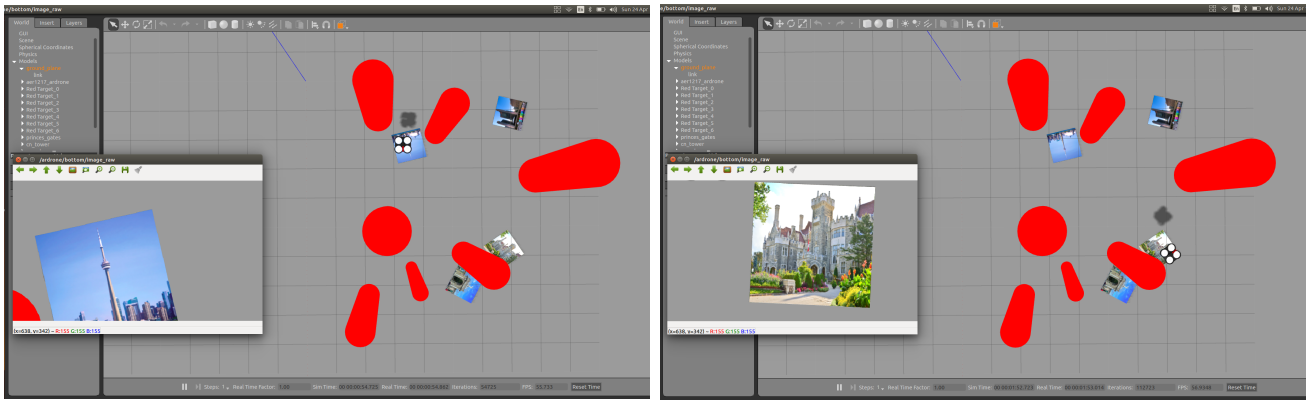


Figure 7: Results from simulation in Gazebo using ROS

The drone successfully followed the planned trajectory without collisions from the obstacles and remained inside the mentioned limits of the 9×9 area and a height of 3 m. The snapshots in figure 7 also shows the view from the bottom-facing camera and confirms that the drone orients itself to see an upright view once it reaches the landmark.

6 General Discussion

The general mobile robotics framework can be classified into four components:

1. **Perception:** At the very early stage, one needs to deal with the raw sensor data. In perception, the raw data from sensors (like CMOS, infrared, IMU, etc.) are converted into actionable information about the states of the environment and the robot.

2. **State Estimation:** Once we have the measurements from perception, those are recorded over time and used to correct our predictions by employing different numerical methods (like Extended Kalman Filter) to construct an accurate estimate of the state of the robot.

3. **Planning:** Given a task or goal, the primary job of a robotic system is to identify a sequence of optimal actions and follow those accurately to achieve its goal, based on the estimated robot state and state of its environment. The sequence of desired actions is generated at this stage.

4. **Control:** The desired sequence of actions needs to be executed reliably and in a controlled way, following some state and input constraints as well as making sure that the cost incurred is least. This is the focus in the control stage. Here, the robotic system is given control inputs based on the feedback from measurements and the desired states.

We would also like to discuss some challenges we faced during different phases of the project and potential solutions:

- **Perception**: As opposed to the data provided in lab 3, the number of circles in the project's image data is a small fraction. As a result of data scarcity, the confidence in clustering the circle centres is low and hence the final georeferencing results are bound to be inaccurate. Ideally, we would like to collect image data from multiple views and multiple times.
- **Planning:** One challenge is choose the trade-off between computational cost and accuracy. As we discussed in section 4.2, we chose to plan in a low-resolution map to obtain fast planning results. We could also try A*, which is also an optimal path planner and converges faster than Dijkstra's algorithm.
- **Control:** Initially our path tracking was slow. To fix this issue, we tried two things: 1) increasing P gain in the controller so that the desired state is achieved faster 2) as discussed in section 5.1, we increased the threshold of reaching the desired point.

References

[1] Jon Louis Bentley. "Multidimensional Binary Search Trees Used for Associative Searching". In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: https://doi.org/10.1145/361002.361007.

[2] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[3] *Find Homography*. URL: https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780.

[4] *FLANN*. URL: https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html.

[5] *Hough Circle Transform*. URL: https://docs.opencv.org/3.3.1/da/d53/tutorial_py_houghcircles.html.

[6] *Hough Circles*. URL: https://docs.opencv.org/3.3.1/dd/d1a/group__imgproc__feature.html#ga47849c3be0d0406ad3ca45db65a25d2d.

[7] David G Lowe. "Distinctive image features from scale-invariant keypoints". In: *International journal of computer vision* 60.2 (2004), pp. 91–110.