DRDO DGRE's Vision Based Obstacle Avoidance Drone

Group 19

Introduction

- Unmanned Aerial Vehicles (UAVs) have high potential to support search tasks in unstructured environments. Small lightweight agile UAVs such as multi rotor platforms with limited sensors are suitable for detecting the objects of interest in clattered outdoor areas.

- The given DRDO problem statement was to construct an autonomous flight that navigates from one point to another avoiding the obstacles on its way.
- After reaching the final point (i.e at the specified arcomarker), the message to be printed was "*Marker ID : 0, Landed*".

Approach to solve the problem

- The strategy to solve the problem with our autonomous drone was to take a path from the initial point to the aruco marker which is as short as possible.

- The approach goes as below:
  - Autonomous takeoff and navigating a certain distance.
  - Image recognition with the help of darknet/yolo and openCV.
  - OpenCV - aruco marker detection and darknet/yolo - Obstacle avoidance
  - Collect the data produced after recognition for further analysis.
  - Further analysis included decision making for the shortest path and obstacle avoidance.

Things we tried out!

- We wrote a script for the autonomous takeoff of the drone.
- We simulated the navigation of the drone autonomously but the waypoints were predefined for the path it had to follow.
- We could subscribe to the depth camera but unfortunately we weren’t able to subscribe to the downward facing rgb camera .
- We worked on the path planning algorithm as well. We have the script written for it but since the rgb camera didn’t workout for us, we were unable to integrate it with the script written.

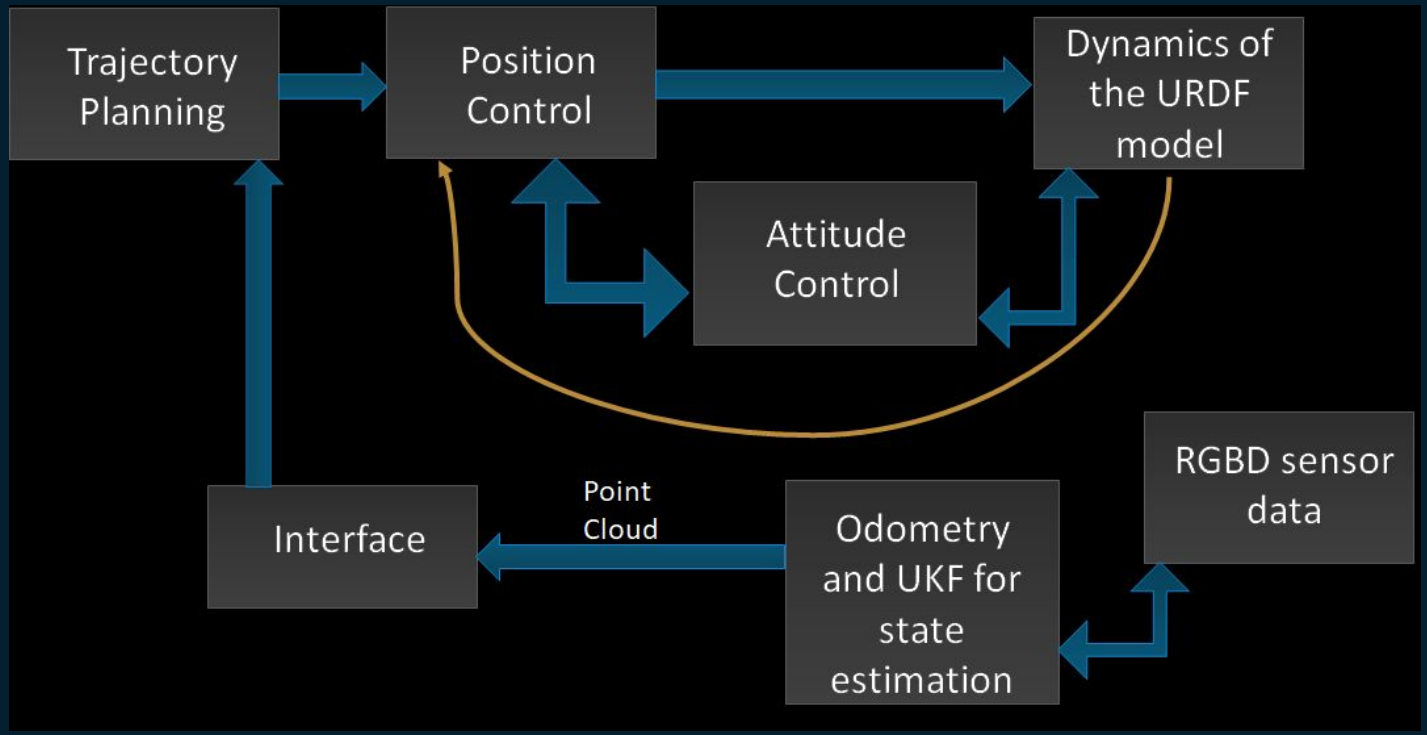
Image Processing

- We were able to subscribe to the topic *depth_camera/rgb/image_raw* which is the forward facing camera.
- We were able to retrieve data from the forward facing camera which was in the format described below.
- We were not able to subscribe to gimbal_2d which is downward facing camera. Due to this, we were not able to complete the obstacle avoidance algorithm since the path_planning algorithm written by us didn’t get the data and hence failed in integration.
- We tried the simulation with the way points predefined.
- Yolo/darknet required too many computations to perform the tasks and gpu was taking a lot of time for processing the data.

Marker Detection and Landing

- We created a python script that subscribes to the feed of the downward facing RGB Camera /webcam/image_raw and obtains the image
- This image is processed by OpenCV. Using the aruco library, we get the coordinates of the aruco markers and their id's.
- Once the id 0 is detected, its coordinates are marked. Using the dronekit API, we communicate with the SITL via Mavproxy to start the landing script.
- The drone is commanded to go to the coordinates of the marker '0' . Once there, it is shifted from GUIDED mode to LAND mode.

System Architecture



Trajectory Planning

Our main objective here is to generate and implement kino dynamically feasible trajectories with a considerable success rate.
Factors into consideration:-

1. Time boundedness, computational resources, capabilities
2. Regeneration of trajectories, efficiency and robustness.

Initial approach: The motion planning problem can be coupled into two parts:-

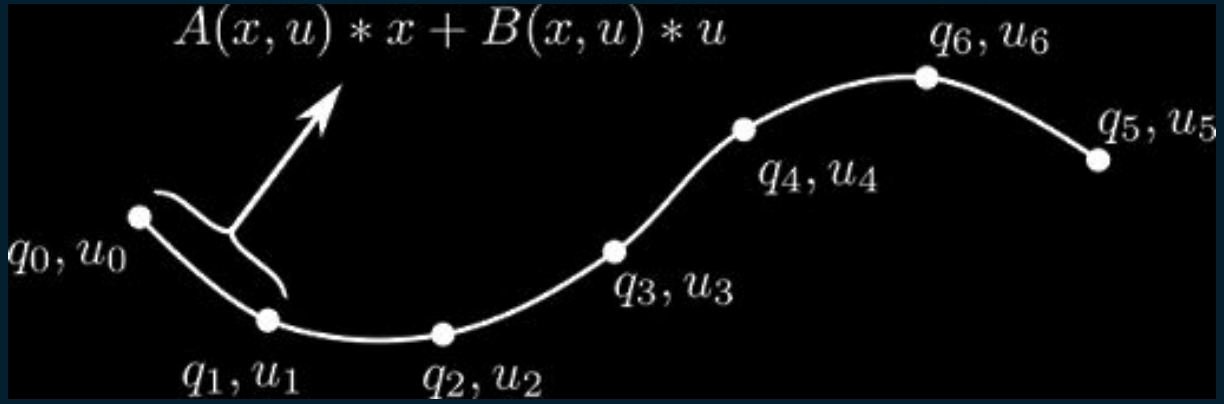
1. Front-end path searching

2.Trajectory optimization

4th order(minimum snap trajectory generation) : Piecewise polynomials linked generated by quadratic programming.

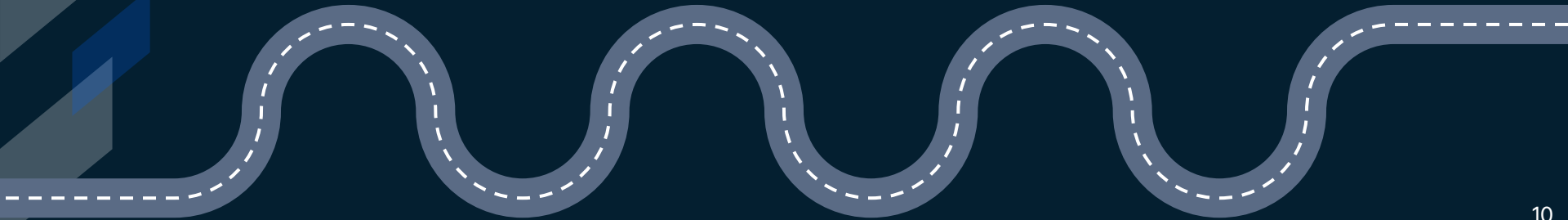
Trajectory Planning

- The control inputs are u=[u1,u2,u3,u4], where u1 is the net force on the body and u2,u3 and u4 net body moments.
- We then represent the control inputs in terms of derivatives of p(defined later).
- We define a trajectory as a smooth curve in the space of flat outputs.



Trajectory Planning

- Considering the differentially flat dynamics for quadrotor, we choose the corresponding output vector as $p = [x,y,z,\psi]^T$, here, ψ is the yaw angle.
- We define 'a' waypoints (timestamped) in the space, with corresponding yaw angles to positions.
- Interpolation will generate trivial trajectories amongst the waypoints.
- We generate trajectories in the output space over our given 'a' time intervals of 4th order(snap).
- We get cubic/higher order spline functions.



Optimization

- We want the trajectory to follow our max. error limits.
- Note that our objective is to minimize the quadratic cost, i.e. to minimize the functionals $L(\mathbf{q},\dot{\mathbf{q}},\mathbf{u},t)$ which are written using the individual polynomial functions. The optimal function should satisfy the Euler-Lagrange's equation.

Optimization

Cost function: *The problem boils down to:* Trajectories which minimize the functionals while simultaneously minimizing the integral of square of norm of snap (2th derivative of accelerations) and 2nd derivative of yaw squared.

- We call these as optimal trajectories.
- We can see that our input variables ui are related algebraically to the snap.

Now, we can enforce continuity conditions here.

Quadratic Program: Now, with all the 'p' vectors, we can introduce a decision variable such that the constraints on flat outputs and their derivatives are also satisfied. $f(\mathbf{q},\mathbf{u})\geq\mathbf{0}$

Optimization

Obstacles: The constraints for obstacle avoidance (integers) are based on the geometry of the bounding polyhedral.

Additional safety constraints:

- On each of the trajectory segments, we define a safe corridor width on the infinity norm.
- Constraints are added by introducing some intermediate points.

In a nutshell, we tried to implement computationally efficient and feasible trajectory generation algorithm with near optimal performance.

Team Members

Soham Kulkarni

Nisha M

Sanket Ranade

Siddhant Chandorkar

Aadil Salim

Advaith P