Team F

Individual Lab Report 5

Progress Review 4

Yuchi Wang

Teammates

Danendra Singh

Pulkit Goyal

Rahul Ramakrishnan

Pratibha Tripathi

November 22, 2017

1 Individual Progress

My primary contribution since the last PR was the development and implementation of a custom GPS navigation controller for the Bebop 2 Drone and implementing a low pass filter for April Tag detection. The controller design involved several subtasks: calculation of the required heading and the difference in angle between the heading and the yaw, designing the controller logic and tuning the parameters, and expanding the GPS navigation code to multiple waypoint navigation.

1.1 Heading calculations

Recall that in the last PR, we were able to calculate the distance between two GPS locations. This information is important for GPS navigation as it allows us to set a margin parameter to specify our desired accuracy. However, the distance alone isn't sufficient information for autonomous navigation as the drone also requires a heading direction. Specifically, the drone needs to know which heading it needs to go to reach the target destination from its current location. This is also known as the forward azimuth and it is expressed in degrees from the north axis to the target. The equation for calculating this value is provided below, where λ is the longitude and φ is the latitude, and this calculation was implemented in the navigation controller.

$$\theta = \text{atan2}(\sin\Delta\lambda\cos\varphi_2, \cos\varphi_1\sin\varphi_2 - sin\varphi_1\cos\varphi_2\cos\Delta\lambda) \tag{1}$$

In addition, the drone needs to know its current heading such that it can minimize the error between its own heading and the forward azimuth. Fortunately, this information is already published by the Bebop Autonomy driver and can be read directly from the `states/ardrone3/PilotingState/AttitudeChanged` topic. With this information, Danendra and I tested that the Bebop 2 Drone is able to align its heading with the forward azimuth. However, there were a significant number of challenges that were encountered when testing this functionality which slowed down our progress (discussed in section 2).

1.2 Controller design and tuning

The basic idea of the navigation controller is simple: move in the direction of the GPS waypoint until the drone is within a certain radius of the location. The heading calculations described previously provides the drone with the direction of movement while last PRs work on GPS distances provides the radius margin. In terms of implementing the controller, there were two main approaches that I considered.

First, I note that the drone is omnidirectional and thus it does not need to face a certain orientation before moving in that direction. As a result, a possible controller implementation could simply provide x-y translation commands in a ratio determined by the difference in heading. For instance, if the required heading to the target destination was 90°of the drones current heading, then the appropriate move command would be $(0, -1, 0)$, moving the drone in the $-y$ direction. If the required heading was 45°, then the move command would hypothetically be $(1/\sqrt{2}, -1/\sqrt{2}, 0)$. The issue with this control architecture is that Bebop 2's move commands are not specified as m/s but instead in roll, pitch, yaw values (as a percentage of their max angles). Because the max angles are different for roll and pitch and the aerodynamics of the Drone are different when flying in different orientations, a command of $(1/\sqrt{2}, -1/\sqrt{2}, 0)$ does not actually move the drone at exactly 45°. It may have been possible to calibrate the roll and pitch commands but this approach would've been more complex than what was actually implemented.

The first implementation of the controller separated the rotational movement from the translational movement. The basic premise is to first orient the heading to face the target destination and then issue a move forward $(1, 0, 0)$ command, repeating the sequence until the drone is within a certain radius margin of the final location. The first controller implemented this concept but it performed very poorly during testing (further discussed in Sec 2). When Danendra and I tested the controller, we noticed that the flight was rather stiff and the drone moved slowly - a video of our first flight can be found in

Section 5, item 1. Varying the margin parameters helped increase the smoothness of the flight but it also decreased the accuracy and thus was not an acceptable compromise. The current controller combines and calculates the translation and rotation values in proportion to the distance and heading error from the target location. The relation between the speeds and the error were initially linear (P controller) but further experimentation revealed that a non-linear scaling from minimal speed to maximum speed results in better performance. A video of the current controller is shown in Section 5, item 2.

1.3 Expansion to multipoint navigation and code refactoring

Although the videos shows multipoint navigation capability, the original form of the controller only allowed for one GPS target. Incorporating a multipoint GPS navigation system involved changes to the code to process multiple waypoints and during the test process, it also motivated the other refactoring changes described below.

Initially, the target GPS location was hard coded but it became apparent that this approach would not work for multiple GPS waypoints. Creating a new variable for each GPS waypoint unnecessarily complicates the codebase while hardcoding each value in an array is time consuming and does not generalize well to different number of waypoints. In addition, recompiling the source code each time just seems like bad practice so I decided to migrate the GPS waypoints to use ROS param for run-time resolution. Several other run-time parameters mostly used for tuning purposes were also moved to the .launch file and retrieved via rosparam. They include the minimum and maximum angular and translational speed of the drone, the alpha value for april tag detections, and the margins.

In addition, I implemented a logging functionality into the codebase. Previously, debugging the code was accomplished primarily through print statements to cout but this method is crude, doesn't generalize well to larger programs, and the bebop driver itself also outputs a significant number of warning and error messages to cout. As we added more features, these extraneous error messages increased and it soon became very hard to find relevant log information in the terminal. As a result, a logging feature was created so that each callback function can record its time of callback, the values it reads, and the members that it changes. A high level diagram of the entire controller schematic is shown in Fig 1.

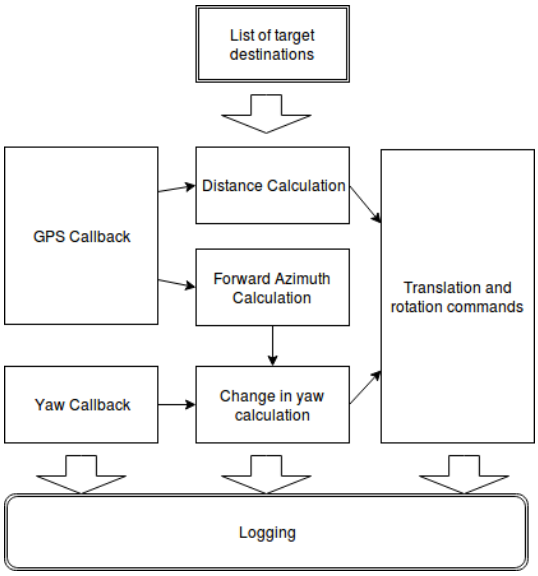

Figure 1: Schematic of controller

1.4 April Tag stabilization

Unstable localization with April Tags due to the concatenation of multiple transforms has been a persistant problem for the last two PR. Sasanka, our PhD advisor, suggested that we try implementing a simple low-pass filter to stabilize the readings. Following his advice, we implemented the filter in the following form:

$$y_i = \alpha x_{i-1} + (1 - \alpha) y_{i-1} \tag{2}$$

After some tuning, we managed to get significantly better localization with $\alpha = 0.00003$. Using this data, we plotted the pose of each April Tag in rviz as an arrow and also displayed the poses and distance from the *home_frame* (represented by april tag 0 and whose pose is at the origin). The localization with the filters is very accurate - well within the margins that we have set for FVE. In Fig 2, 8 April Tags are placed around Tag 0 in a radius of 1m and facing outwards. The visualization of their pose is shown in Fig 3 - one can see that the distances are within cm's of 1m.

Figure 2: Picture of 8 April Tags placed in a circle of radius 1m around tag 0

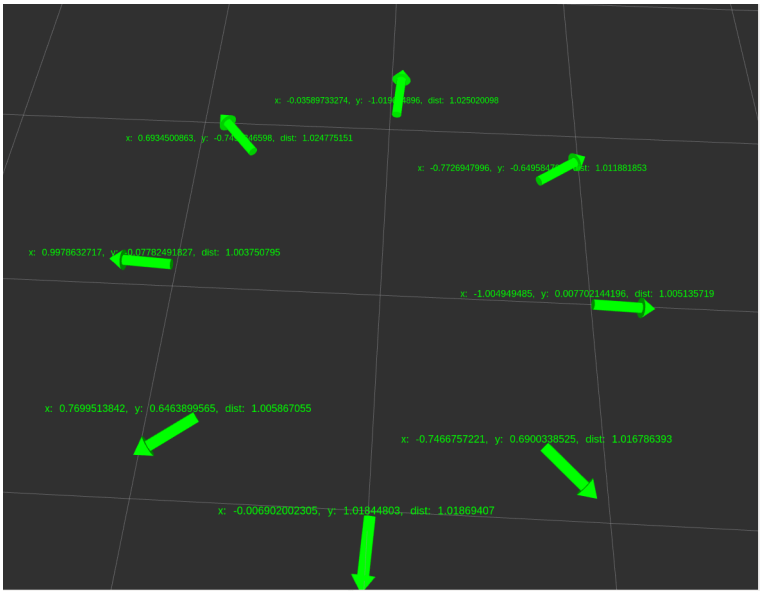

Figure 3: Visualization of the 8 April Tags in rviz

A disadvantage of using low pass filters is the slow response time of the readings. In our tests, we noticed a delay of approximately 1s between the moving of an april tag and its new location reflected in rviz. However, this isn't a significant issue for us as we are using April Tags to represent mostly stationary objects (ie, the location of the paths). The only object that we are tracking which will be moving is the Husky. At the moment, this does seem to be a challenge that we will have to overcome and we currently do not have a solution in mind. We plan on exploring the exact implications of the low pass filter on localizing the Husky after the FVE.

2 Challenges

There were many challenges associated with testing the gps and navigation functionalities of the Bebop 2 Drone. First, our decision to perform the FVE outdoors brings along many inconveniences associated with the weather. As winter gets closer, the temperature drops and snowfall is a real concern. We have significant difficulty testing outside in these weather conditions as snowfall will make the laptop wet, the cold weather is not conducive to debugging with exposed fingers, and the earlier sunset means less time for flying with visual odometry. These challenges affected all stages of the controller development.

In addition, there were a number of non-weather related challenges when Danendra and I were debugging the controller. First, the initial testing location was the backyard of B Floor NSH but it turns out that location is extremely inconvenient for flying drones. It is situated right next to the parking lot and is immediately proceeded by a corner, which means that there are many cars passing by and we cannot see their approach from far (due to the corner obscuring our view). As well, the environment was crammed which limited the space where we could fly the drone and the tall buildings resulted in numerous dead zones for the GPS. As well, the places that did have GPS also had drifts of ridiculous proportions (50m). There were also some bugs in our code due to the conversions from degrees to radians and the aforementioned environmental issues made it very hard to narrow down the source of bugs. Eventually, we talked to Dimi who suggested that we test our drone in the field (next to Hamerschlag Hall). We got much better results afterwards.

The first implementation of the controller produced stiff movement due to the discrete nature of GPS updates and a hardcoded magnitude for the move commands. The basic algorithm first compared its current heading with the required heading. If it was outside a margin m_r, it would rotate at v_r. Otherwise, it would compare its current distance to target against a margin m_d. If it was outside this

margin, it would move forward at a speed v_d. Unfortunately, this algorithm performed very poorly in our tests. If v_r was too high, the drone often overshot and oscillated because yaw readings only update at a certain frequency. This also meant that it never moved forward. If v_r was set too low, the drone rotated too slowly. If m_r was set too low, we would theoretically achieve higher accuracy but in practise the drone just overshot the margin. We solved this issue by implementing a proportional controller: if the heading error is large, then the drone should rotate fast. Otherwise, it should rotate slower as to not overshoot its heading. Furthermore, we combined the translational commands with the rotational commands so that we are guaranteed that the drone progress forward.

3 Teamwork

Danendra worked with me on the testing portion of the navigation controller. Due to the dangers of flying a drone in a populated area, testing the drone necessitated multiple people to operate the drone and also look out for bystanders. In addition, Danendra worked on the PDB soldering with Pratibha. There has also been significant progress on the Husky subsystem thanks to Pulkit, Rahul, and Pratibha. They have been experiencing a number of mysterious issues with the Husky's onboard miniPC and they have managed to solve most of them. First, the CMU-SECURE network seems to have issues with the Husky image as `apt-get` commands often didn't work. This was very troublesome as it slows down any development progress if they are unable to install the necessary packages. Furthermore, they found out that the configuration identity of the GPS module conflicted with the identity of the Husky serial port. As a result, whenever they plugged in the GPS, the communication to the Husky itself would disappear since the OS prioritized the GPS over the Husky. They have resolved this by switching back to the SE100 GPS. Rahul has also worked on the networking with the Bebop 2. He has managed to configure the Bebop 2 as a WiFi client, thus allowing us to finally run all our systems on one network.

4 Future plans

Our immediate plans for the future is to prepare for the FVE. To that end, Pulkit, Rahul and I will focus our attention toward the Husky while Pratibha and Danendra will work on the PDB. The April Tag localization and Bebop 2 GPS navigation is already complete so those are not high priorities for the upcoming weeks. For our FVE, we have decided to perform Husky GPS waypoint navigation. At this point, this task has not been started. However, we have a clear approach to accomplishing this task before FVE. Clearpath has a template in ROS for GPS navigation so our first plan is to adapt their package to the sensors and ROS topics that we have. If that does not work, we will adapt the GPS navigation code from the Bebop 2 to the Husky. This should provide us with adequate performance for FVE as the overall navigation algorithm is the same and the codebase is written in such a way that it abstracts away the underlying system.

5 Video Links

1. Multipoint navigation with stiff controller
   `https://drive.google.com/open?id=1Rn768_7JlKf0QKdr5Q5_jtTDgHWBOKfe`
2. Multipoint navigation with current controller:
   `https://drive.google.com/open?id=1nHqpn8OfOKwHF1lc4WRYtS6e2n9W11q2`