ILR09: Progress Review 10

Maitreya Naik

Team B: Arcus

Teammates: Logan Wan, Clare Cui, Angad Sidhu

ILR09

Mar. 22, 2017

1. Individual Progress

Since the last progress review, I was responsible for tuning the controller gains for the rebuilt drone, and figuring out the current trajectory generation and motion management software architecture along with Angad.

1.1. Controller Tuning

Most of the tuning is done on a firmware developed in the RASL [1] for the px4 [2] flight controllers. This is done in collaboration with the pilot and lab post-doc fellow, Moses Bangura [1]. Most gains are tuned via trial and error, and the results are shown in the plots in Figure 1 below.

Figure 1: Flight Plots with the blue curve in both denoting the thrust, and appropriate flight state demarcated as "Grounded" or "In Flight". Top – Plot of pitch rate with set point in Red and measured values in green. Bottom – Plot of Yaw rate with set point in Gray and measured values in Yellow.

The roll rate plot looks similar in tracking to the pitch rate plot. The abrupt set points were to test the drone's speed of reaction. The yaw rate set point was corrected at every 4-5 seconds to maintain constant heading. Most importantly, the yaw drift arises from an integral error building up over time which was hard to mitigate with a high integral gain without loss of stability. Hence, the yaw drift has been corrected as much as possible with mitigation required in manual flight at every 8-10 seconds.

When we attempt autonomous flight, we expect the yaw to be mitigated more punctually by the on-board controller than is possible by manual human operation. The yaw is more difficult to control than roll and pitch as it's a motion that relies on motors spinning in one direction to spin faster than the others, while bearing lower output magnitudes (as seen by comparing the values between the pitch rate and yaw rate graphs while in flight in Figure 1). Figure 2 explains the 3 motions and appropriate motor control. Another possibility for slow control may be the inability of the motors to produce the required torque.

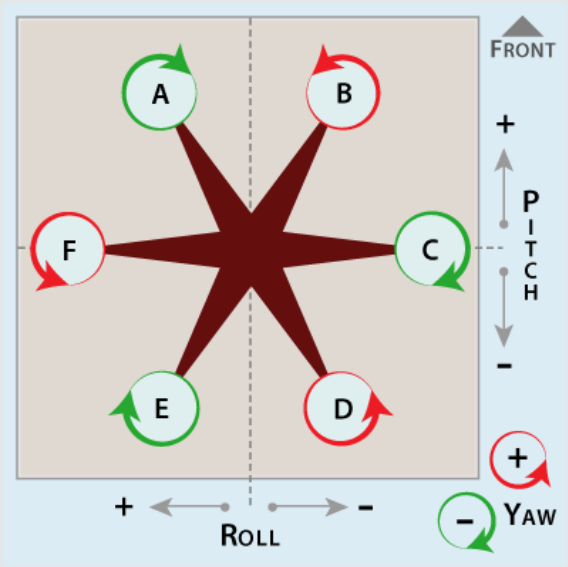

Figure 2: Hexrotor Yaw Control is achieved by raising the speed of motors in the same spin direction. Roll and Pitch rely on the motors in the respective half of the plane spinning faster in either direction. Courtesy [4]

1.2. Trajectory Following and Motion Management

Figure 3 summarises the high-level architecture of the motion management currently in use.

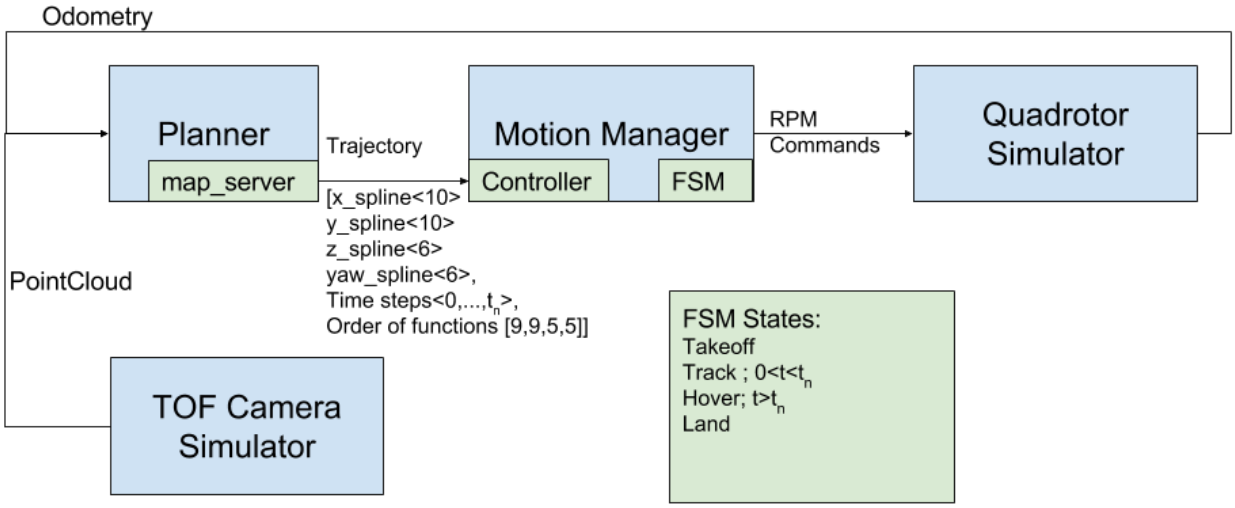

Figure 3: Current motion management high-level architecture

The motion manager contains the Planner which acquires odometry data from our quadrotor simulator (will be replaced by the "state-estimation" on our vehicle) and point cloud data for obstacle detection from the TOF Camera Simulator (will be replaced by LiDAR point cloud data). The planner stands for "pit_exploration_planner" node which generates a trajectory spline as a function of time and sends the calculated spline coefficients for x, y, z, and yaw to the "motion manager" controller. The "motion manager" also has an FSM functionality, with some states listed in Figure 3. As can be seen, the FSM is in track mode for the time steps received from the planner and otherwise stays in Hover or other states. Coming back to the motion manager's "controller", this calculates appropriate motor RPMs and sends it to our Quadcopter simulator.

2. Challenges

The biggest challenge I faced was my laptop having a breakdown out of the blue. Hence, I had to set up the RASL's common laptop with our project sandbox and dependencies, which took a lot of time. Apart from the standard "ros-indigo-desktop-full" package and "OpenCV 2.4", I had to set up the Armadillo package (libarmadillo) and its dependencies BLAS, ATLAS, and ARPACK (libblas, libatlas, and libarpack respectively) which are linear algebra libraries for C++. Since RASL has its private GitLab server, and multiple SSH keys on the lab laptop, adding permissions and configurations took about 4-5 hours, as issues would be visible only when trying to build the project sandbox and its multiple packages.

Meanwhile, in the current planner-motion manager-simulator architecture, there is an issue that the vehicle would sometime not follow the trajectory spline sent out by the planner. I think this bug might root in the fact that the trajectory following is based on a time-based goal than a closest-trajectory-point goal, which may lead to the drone's commanded displacement vector deviating from the trajectory spline.

3. Teamwork

Clare was responsible for further understanding the map_utils package and in creating an occupancy grid representation and filling occupied voxels with random colors for proofing.

Logan was responsible for further understanding the map_server package and colorizing a point cloud with RGB data received from a rgb camera and tof camera simulator. Clare and Logan were working a separate branch and mostly isolated packages than Angad and I, hence, the RGB camera simulator is not currently visible in our version of the planner-motion manager-simulator architecture.

Angad was responsible for studying the current motion manager pipeline, while specifically focusing on the trajectory generation, while I was focusing on the trajectory tracking and collision avoidance

4. Plan

Moving forward, Logan and Clare will be integrating their parts of the mapping software and improve the colorization of the point cloud to be more reliable and continuous than the current state. They will then proceed to interface the simulated mapping software with our current system. Angad and I will be debugging the current trajectory formation and tracking, before testing out on a smaller quadrotor in the lab cage area.