Angad Sidhu
Team B Arcus
Logan Wan, Maitreya Naik, Clare Cui
ILR 9
March 22 2017

Individual Progress

My project goals are to investigate bugs that our advisor has indicated to us and to determine if the navigation stack is capable of navigating to a waypoint specified by user. The majority of time spent was on diving into the motion manager stack. The objective was to determine both how it works and identify possible failure points. I was able to determine the overall structure of the motion manager which is illustrated in Figure 1 below.

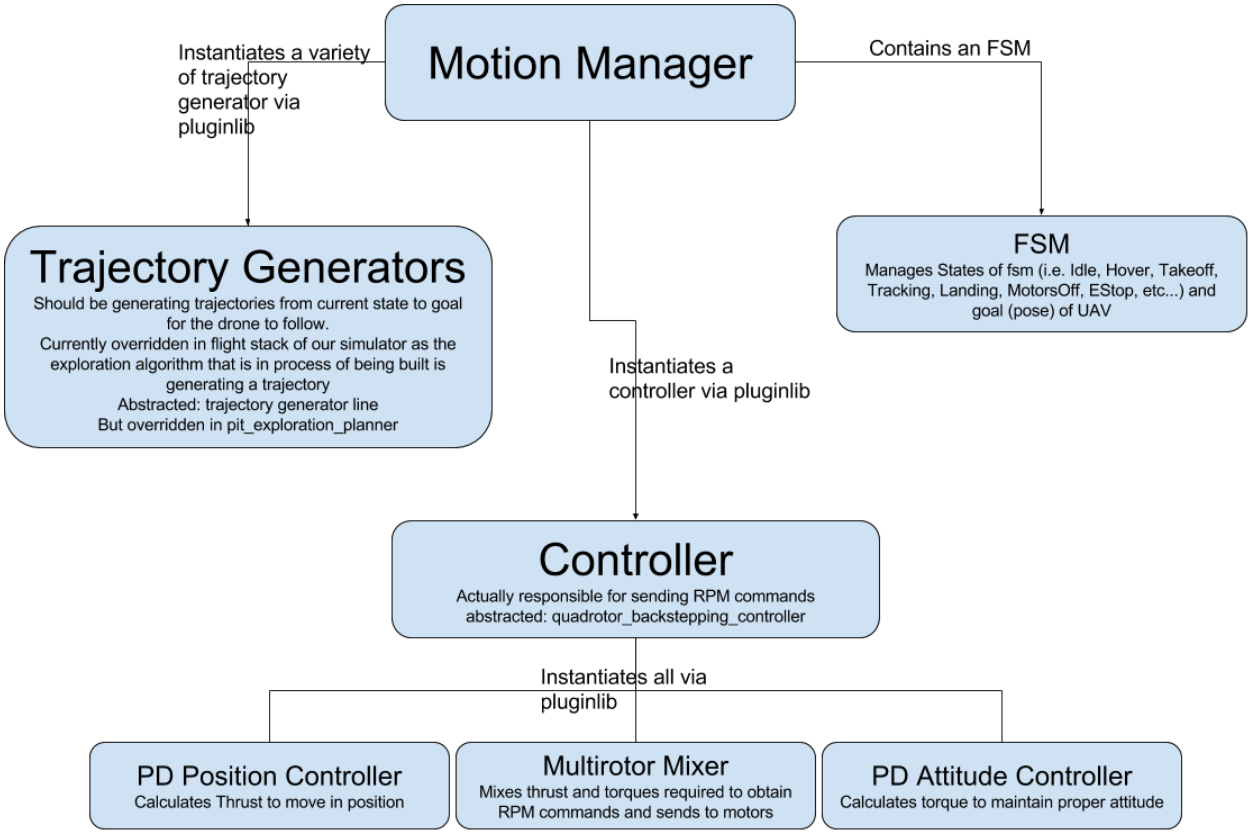

Figure 1. Summary of Motion manager architecture

We start at the top with the motion manager. This only concretely holds an FSM. The FSM is responsible for managing the state of the UAV. This includes its current state (like pose and velocities), the state in the FSM and the current goal. The motion manager then holds trajectory generators which are selectively applied based on the state of the quadrotor when a goal is specified. It also holds a controller which is responsible for executing trajectories. Both the trajectory generators and controllers are abstracted and concrete classes are dynamically loaded in via pluginlib. There are two trajectory generators being used in our specific sandbox. One being a “line generator” which generates poses along a straight line from current position to goal position. This applies when the drone is taking off or landing. When the drone is tracking a trajectory instead it relies on the “onboard” trajectory generator. This relies on the pit exploration

planner for generating a trajectory and communicating it to the controller directly. There is a lot of obfuscation in the controller as well. The concrete implementation of controller that we are using is the quadrotor backstepping controller. This specific implementation combines thrust and torques from an pd position and pd attitude controller with a multirotor mixer. The multirotor mixer is abstracted based on the number of rotors a UAV has.

Challenges

The biggest challenge I faced was the huge amount of packages. There was essentially a package for almost every class. The motion manager stack is about 15 packages with maybe 50 classes in it. This obfuscation was very difficult to dig through, but eventually using liberal use of project wide searches I was able to get a clearer picture of the packages. In order to even determine where to start I used rqt_graph which is depicted in Figure 2.

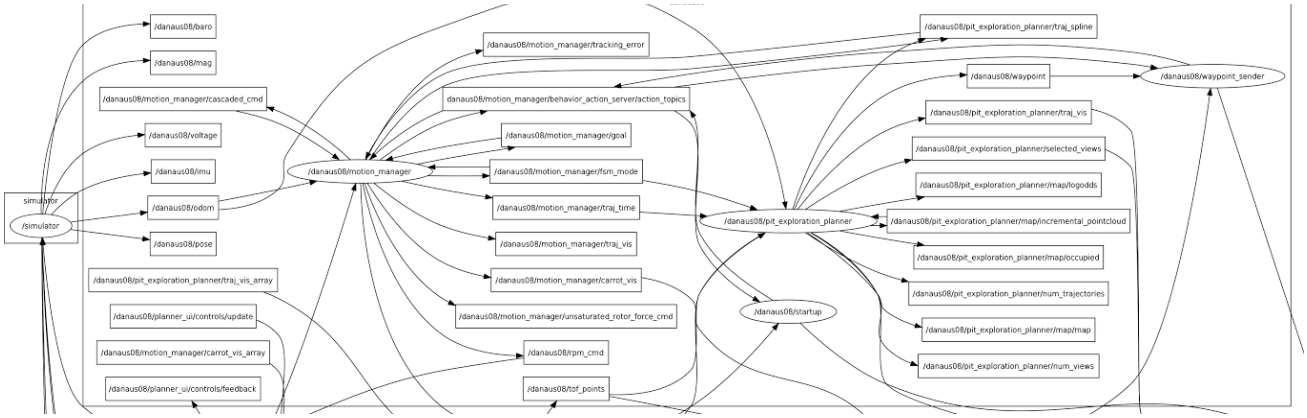

Figure 2. rqt_graph of motion manager.

From here I saw a few interesting things like the pit exploration planner publishing to the traj_spline which led me to discover that pit exploration planner was overriding the default trajectory generator. Similarly I was able to trace a route from the /rpm_cmd topic to the /waypoint topic published from the planner. This helped highlight the route of information.

Teamwork

Logan was working on getting the rgb images and back-projecting point cloud data into the image. This allowed him to obtain a colorized point cloud.

Clare worked on the flip side of this in making sure the mapping framework was capable of holding the RGB data.

Maitreya worked with me in investigating the sandbox and spent time on tuning gains and looking into potential flight issues.

Plans

I will be working on figuring out the motion manager bug. I have a few suspicions as to where it may lie. For example, an FSM is very strict in its transitions and all of the dependents of the FSM could depend on things all being set up in a specific manner. The “onboard” trajectory generator breaks this paradigm by providing trajectories out of process of the motion manager. Similarly the bug appears to have a very high error for a very short amount of time after publishing a trajectory so I think that should be the right approach. I also will look into generating a trajectory from current state to the goal state.