Progress Review 11

Cole Gulino

Team C / Column Robotics

Teammates: Job Bedford, Erik Sjoberg, Rohan Thakker

ILR # 10

March 31, 2016

Individual Progress

During this sprint, my main focus was to get the Iris+ hovering consistently above the April Tag. During the week, we discovered that the rotation invariant April Tag pose estimates that were given by the node I wrote last week were prone to a lot of noise.

The noise was due to the fact that the orientation information about the quadcopter from the camera and the April Tag are not very accurate whenever the quadcopter is moving and rotating.

Because of this, we needed to filter the data in order to ensure that the outliers were rejected and not given as waypoints.

This code can be found on our github [1].

In order to filter the outliers, we implemented two methods: RANSAC on a line and RANSAC around a point. I implemented the RANSAC [2] on a line with an example shown in Figure 1.

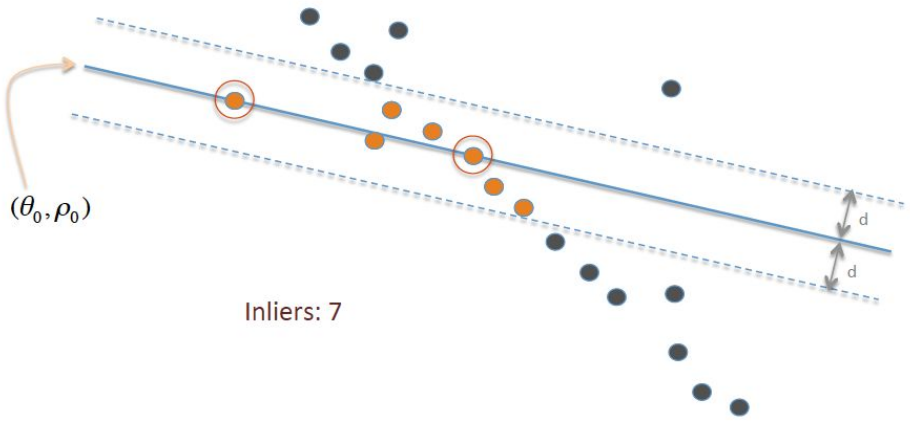

Figure 1) Example of RANSAC

In the algorithm, I am fitting a hyperplane for the (x, y, z) coordinates of the pose estimates from the rotationally invariant april tag that I wrote last PR which can be found on our github as well [3].

I set a threshold for an inlier and an outlier based on the orthogonal projection from the points to the line. The line is determined by randomly selected points along a rolling window of 10 readings from the rotationally invariant april tag node.

I also pass along the covariance which is the ratio of outliers to the total number of points in the window.

This covariance is used by a world modeling node (found on our github [4]) to determine the confidence of the point it has received. Those that do not meet a threshold of 0.3 are ignored.

Figure 2 shows the performance of the filter:

Figure 2) Performance Before and After Filtering

The upper plot is the pose estimates after filtering and the lower plot is the pose estimates before filtering.

There is a significant decrease in the number of outliers.

Challenges

The major challenge for this sprint was to get everything integrated. It was the major functionality that we could not get fixed. This was due specifically to a major time crunch. We did not have enough time towards the end of the sprint to get everything integrated.

We worked very effectively in our sub parts, but it took us a very long time to get all of the sub-functions into a suitable state that they could be integrated.

We have greatly reduced our build-develop-test cycle by introducing python scripts to our system. We are then able to change the python script in real-time to test changes. Erik did a great deal of work on that.

The major conflict with integration was that our code base was scattered over many very different branches. The merge conflicts took up most of the time that we delegated to integration. This did not allow us to test thoroughly.

Because the sub-functions are working and we have the python script framework, it should only be a small matter in order to get them integrated and working.

And now that master has all of the merged work from every branch, merge conflicts should be fewer than before.

Teamwork

There was a lot of pair work on the team during this sprint.

Job and Erik worked together in order to change their off board node to work by getting our filtered april tag data. This node then took that data and changed the quad's position setpoint in our PD controller. We then worked with Job in order to integrate the two. This is the demo that we showed on this video [5].

Once I had completed the RANSAC on a line, me and Rohan worked to find a way that we could improve on this. We decided that the line implementation only made sense, because we thought of the information with a fourth dimension of time. It made more sense to think of it as just a three dimensional point and the inliers could be decided based on a spherical range from the point.

So then we shifted to just searching all of the points in the window for the one with the most inliers (within some Euclidean distance from the point). This code is also implemented in the class found [1].

Figure 3 shows the results from the RANSAC based on a point representation:

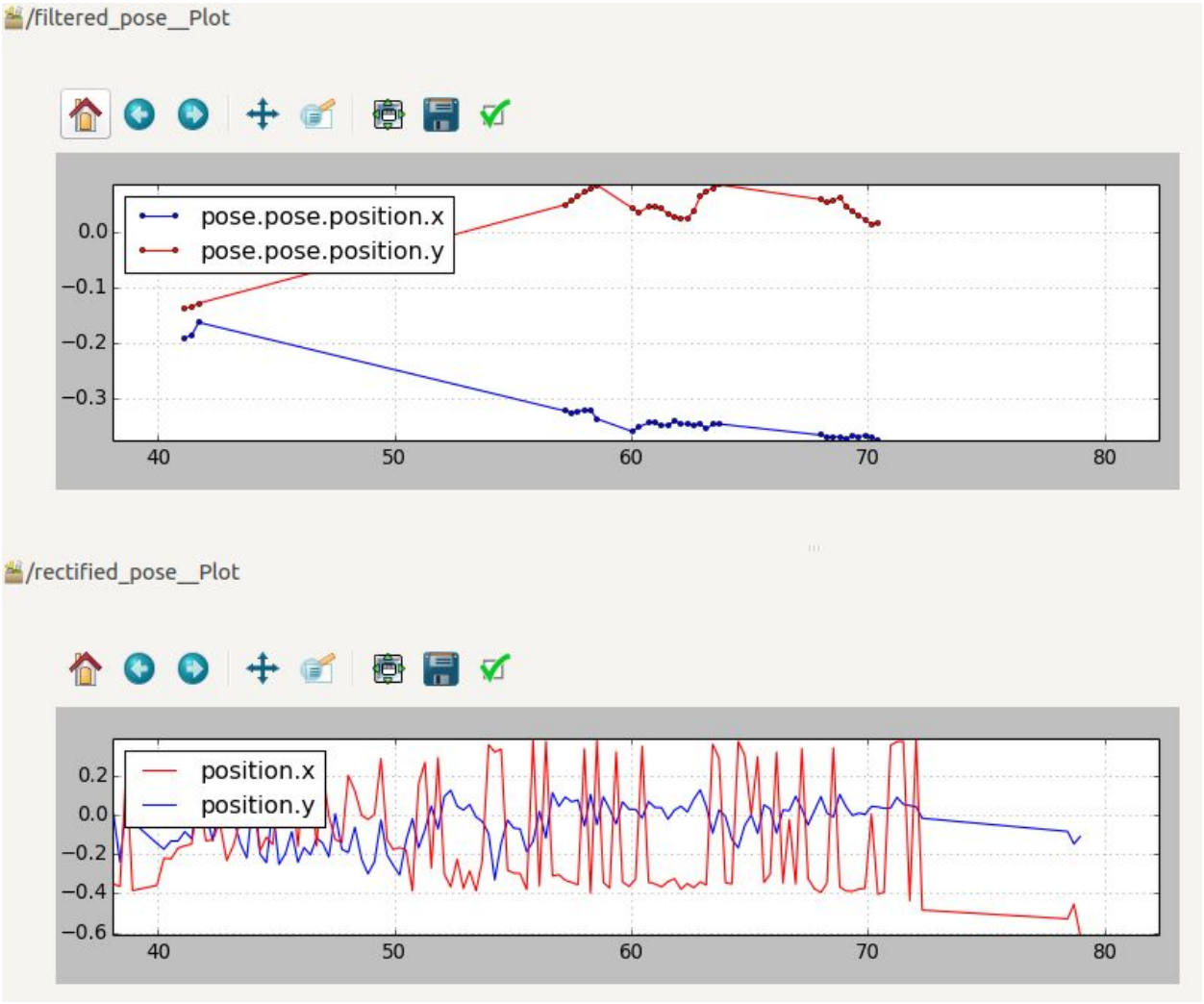

Figure 3) Performance Before and After Filtering on a Point

Unfortunately, the colors are reverse in the two plot.

The top plot shows the filtered readings with using RANSAC based on a point model. The large spaces without readings are due to the fact that this version of RANSAC is more stringent to what is considered an inlier.

As you can see from the image, there are no inliers taken.

The quadcopter is fairly robust to lack of readings. It can hover well enough on its own. This is the version of the filter that is shown in [5].

Rohan and I also worked on a simulator for an EKF. We implemented the code and tested it on a simple point mass model as shown in Figure 4.

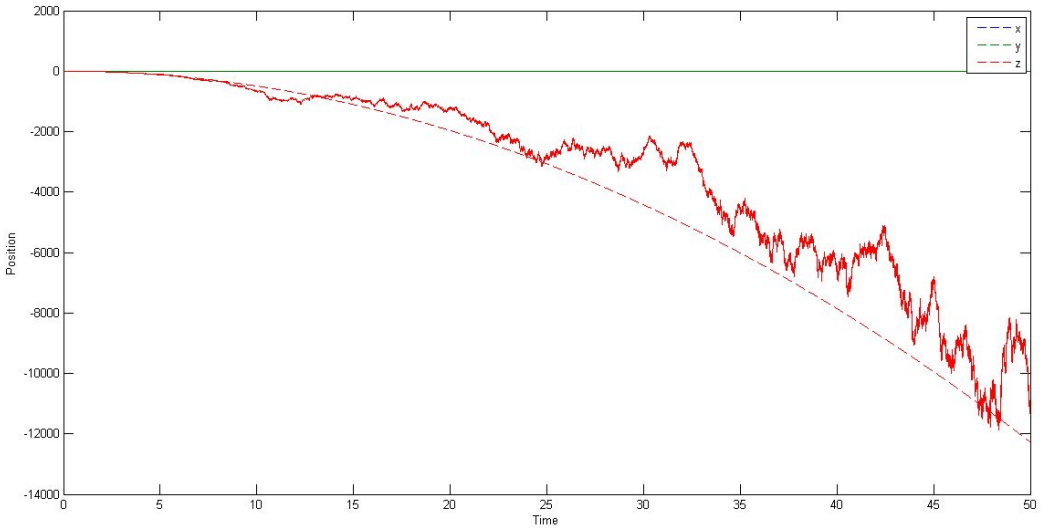

Figure 4) EKF with Point Mass Model

We used this simulation to characterise the kind of drift we can expect to see from our position estimates using the velocity estimates from the optical flow.

Future Work

For the next PR, we will have the fully integrated system working for the SVE dress rehearsal.

Erik will be working with the python scripts in order to ensure that the system integration works as planned. He will also be working on getting ROS communication with the SLAM system that he showed earlier.

Job, Rohan, and I will be working on closing the loop during landing. This is the last thing that we need to do in order to ensure a completed system. We will also be working on including a more realistic quadcopter model to our EKF and characterising the readings from a SLAM system into the EKF. This will be used later to improve our position estimates which will be gotten by Erik's SLAM system.

Resources

[1] BodyPoseFilter.cpp

[2] RANSAC Image Source

[3] rectified_april_pose.cpp

[4] world_modeling.cpp

[5] April Tag Hover Video