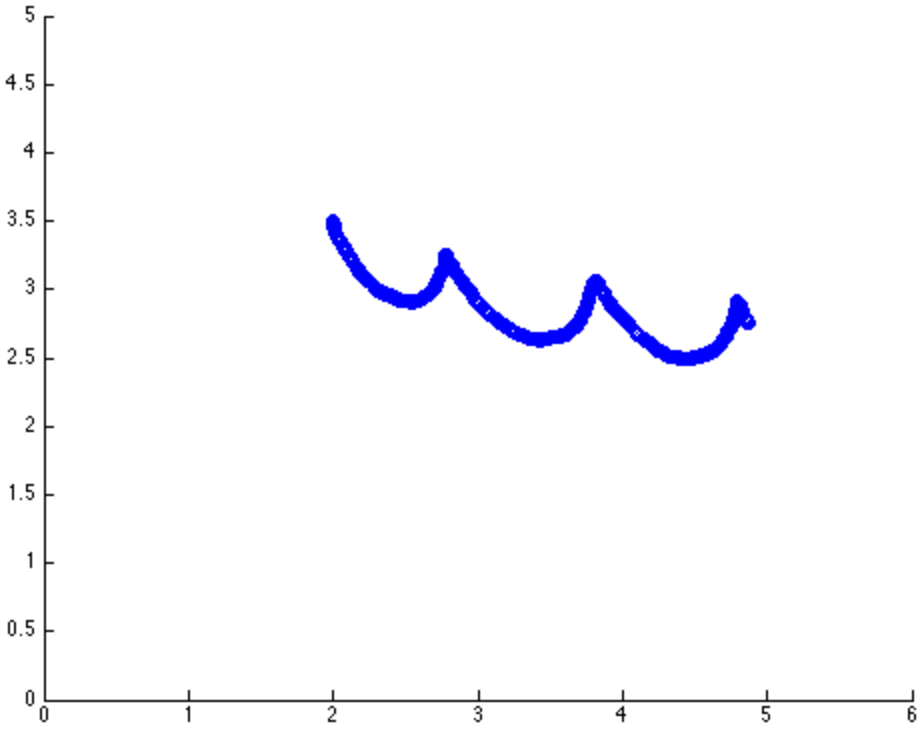
ME597 Final Exam

Leigh Pauls - 20339616
Dec 9 to Dec 10, 2013

Question 1

Question 1 - Part A

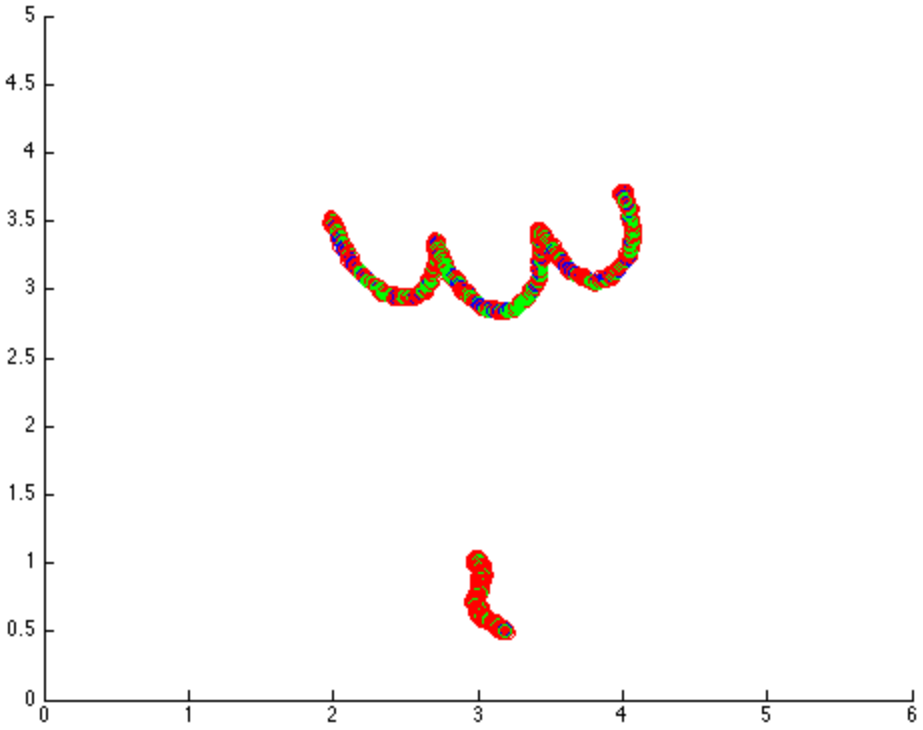
The linear motion model devised for the system is implemented in “/Q1/partA.m”. Using the provided input, below is the plot of one run of the motion model.



The actual position of the robot is shown in blue.

Question 1 - Part B

The same linear motion model was applied to the wand, but no input current was applied to it. This means that the wand is only subject to noise.
The estimator is implemented in “/Q1/partB.m”



The actual positions of the robot and wand are drawn in blue, the measurements in red, and the kalman filter estimates in green. There is a great deal of overlay, but the green estimates track very closely to the blue real positions.

This could be considered a good motion model for the wand, since the mass of the wand plus the user's arm could be similar to the 5 Kg mass of the robot, and the strength of the robot is similar to the amount of force that a typical user's arm would apply. The amount of noise makes sense, considering that the typical user's arm would become very tired and inaccurate after holding a 5 Kg wand for a few seconds.

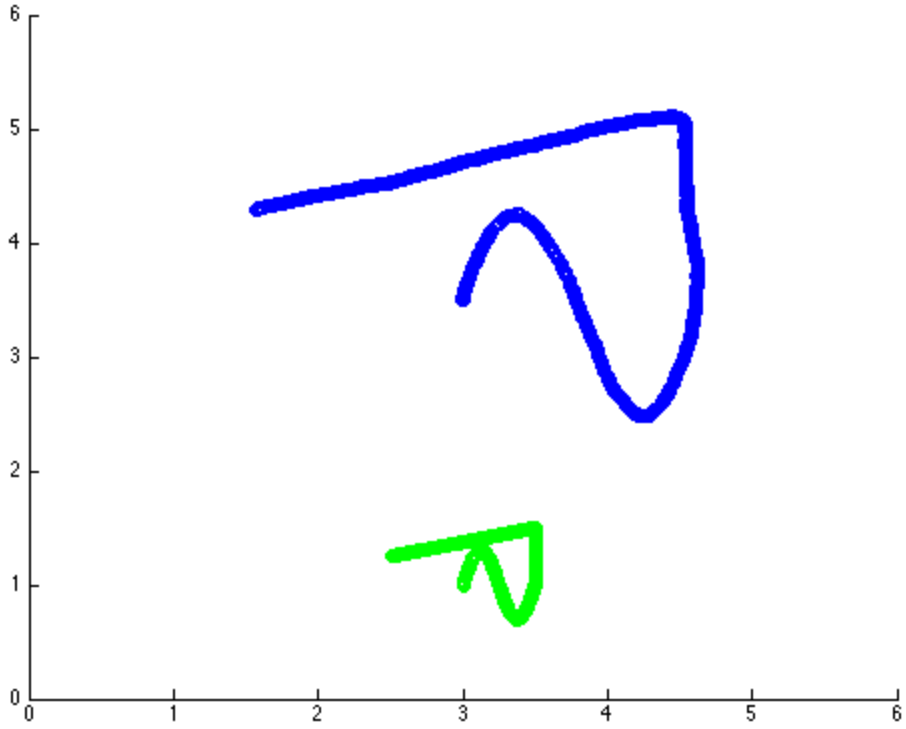
The input terms, however, make no sense since there is no relationship between the robot's coils and the user's arm.

Question 1 - Part C

An LQR was implemented, controlling to robot to track the motion of the wand. The implementation is shown in "/Q1/partCD.m".
The control parameters, LQR_Q and LQR_R were tuned iteratively, to find a tuning where the maximum current did not exceed the limit of 20 Amperes, while tracking the wand position accurately.

Question 1 - Part D

The LQR controller was run using the provided wand positions. The control was implemented in "/Q1/partCD.m".

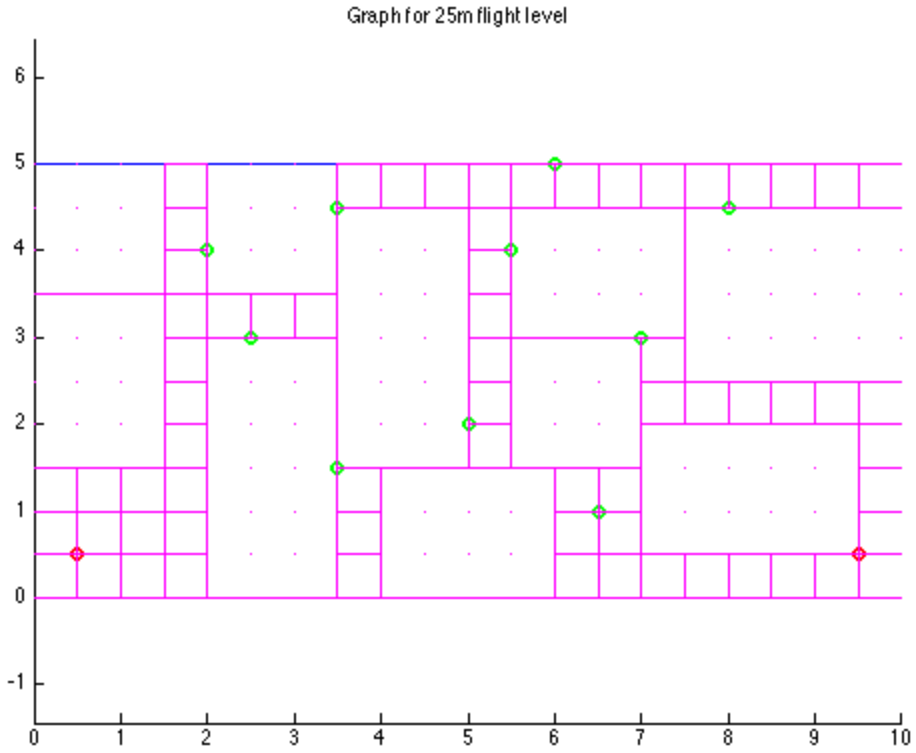


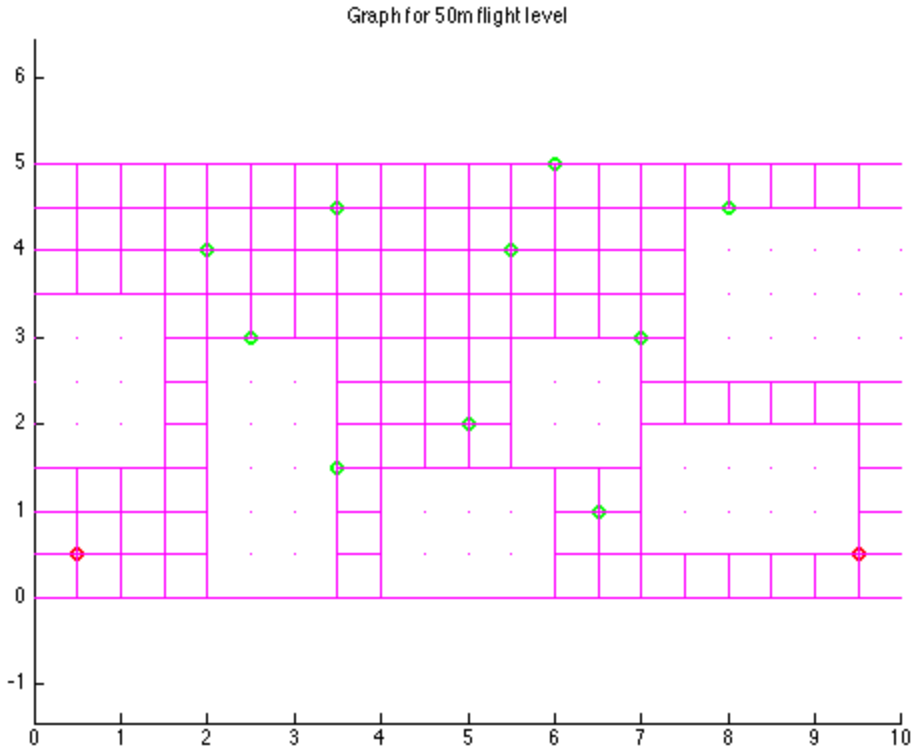
The actual positions of the wand (green) and the robot (blue) are shown in the figure above.

Question 2

Question 2 - Part A

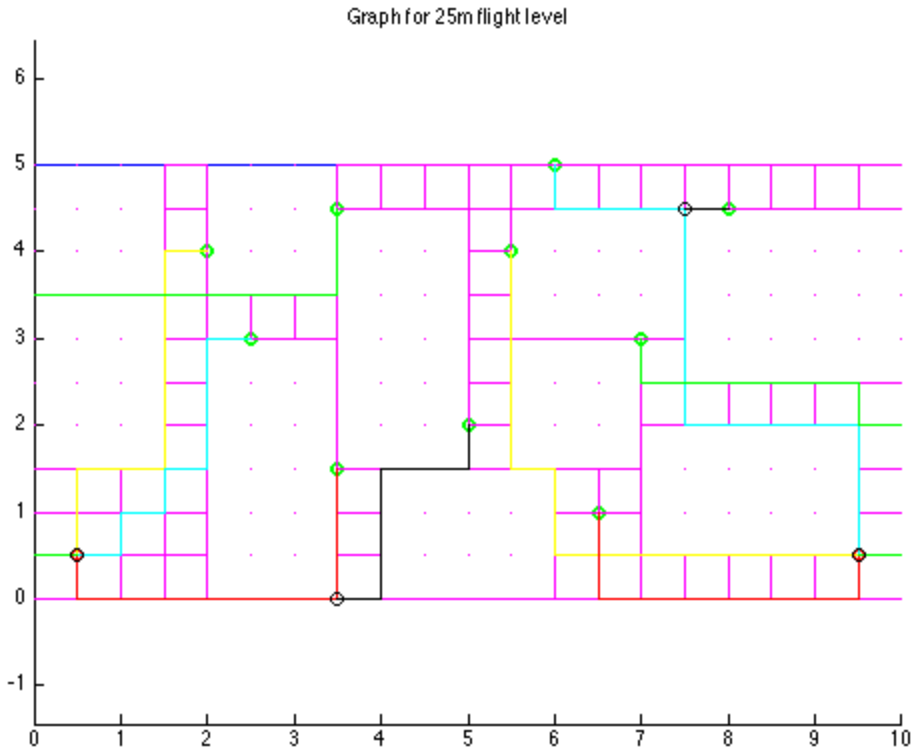
The upper and lower flight plane graphs are pictured below.
The connections are turned into a matrix of connections, implemented in "/Q2/makeConnectionMatrix.m".

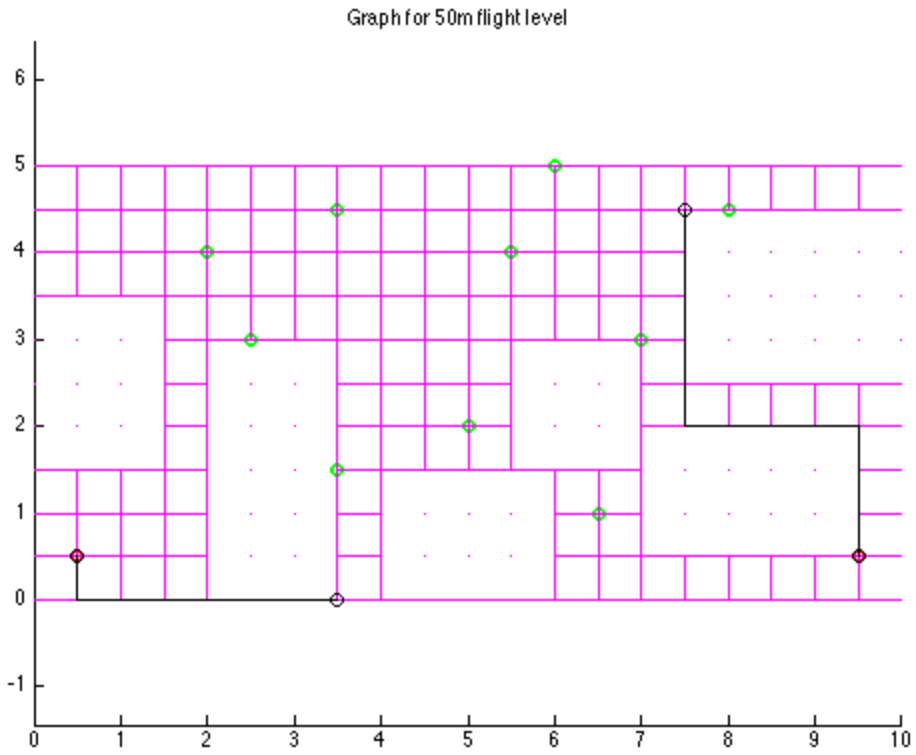

Graph for 25m flight level
6
5
4
3
2
1
0
-1
0
1
2
3
4
5
6
7
8
9
10


Graph for 50m flight level
6
5
4
3
2
1
0
-1
0
1
2
3
4
5
6
7
8
9
10

Question 2 - Part B

A simple breadth-first search was implemented, treating each transition across a grid cell or transition up or down between planes as a 1 minute action. The 5 drop locations on the left side of the map were assigned to the left side depot, and the 5 on the right side were assigned to the right side depot. Sequentially, paths are found one at a time, and each connection used is removed from the connection matrix. This is implemented in “/Q2/partC.m”, and the paths are shown below.



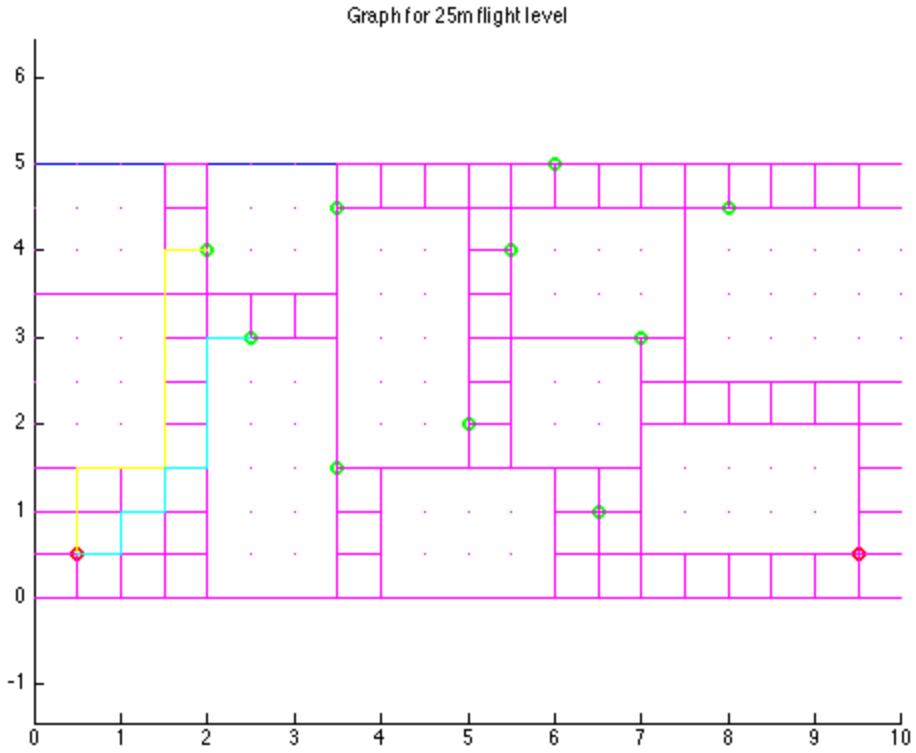


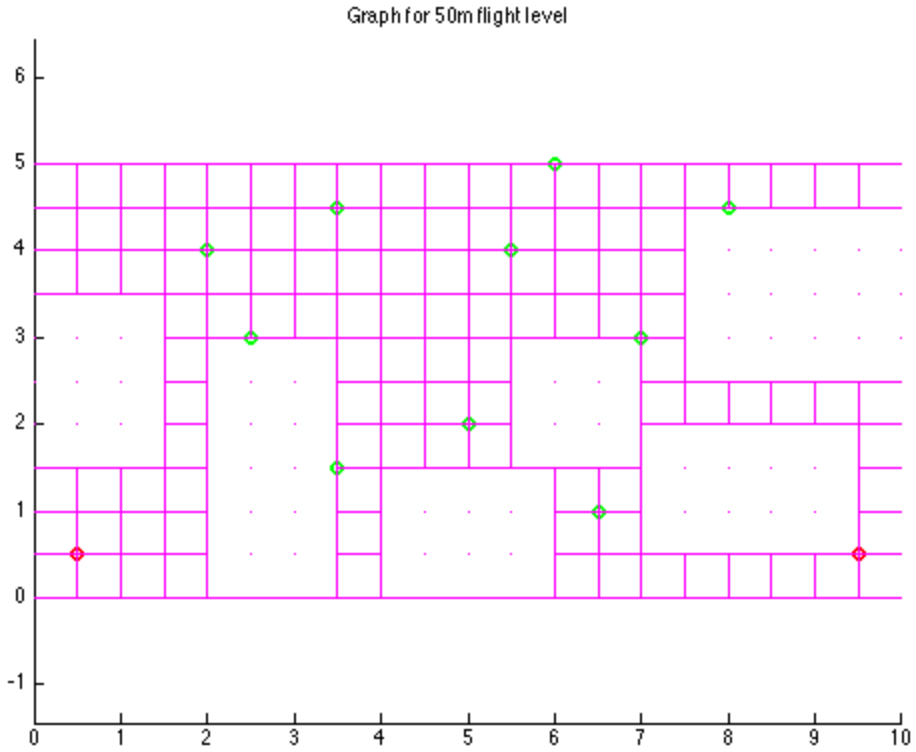
Question 2 - Part C

Using the same breadth-first search, each time that a new order for a delivery becomes available a path is planned for a copter to fly from a depot to the drop spot. The depots are chosen in the same fashion as with the offline planner.

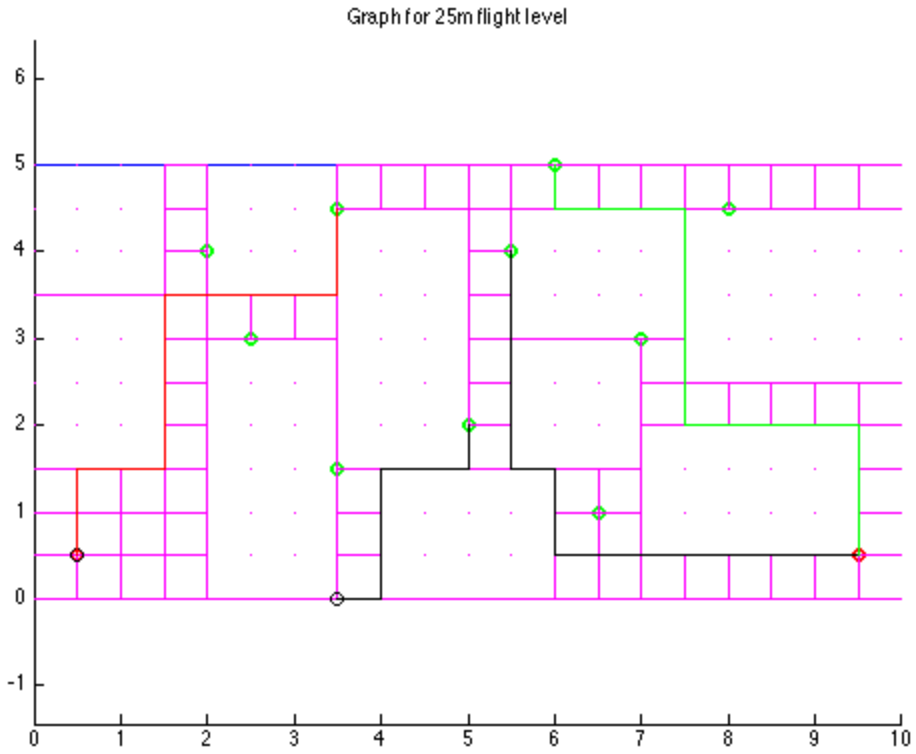
Once a copter has made it to it's drop point, the connections that is was acquiring from the connections matrix are re-enabled in the matrix. A plan for that copter to return back to its origin depot.

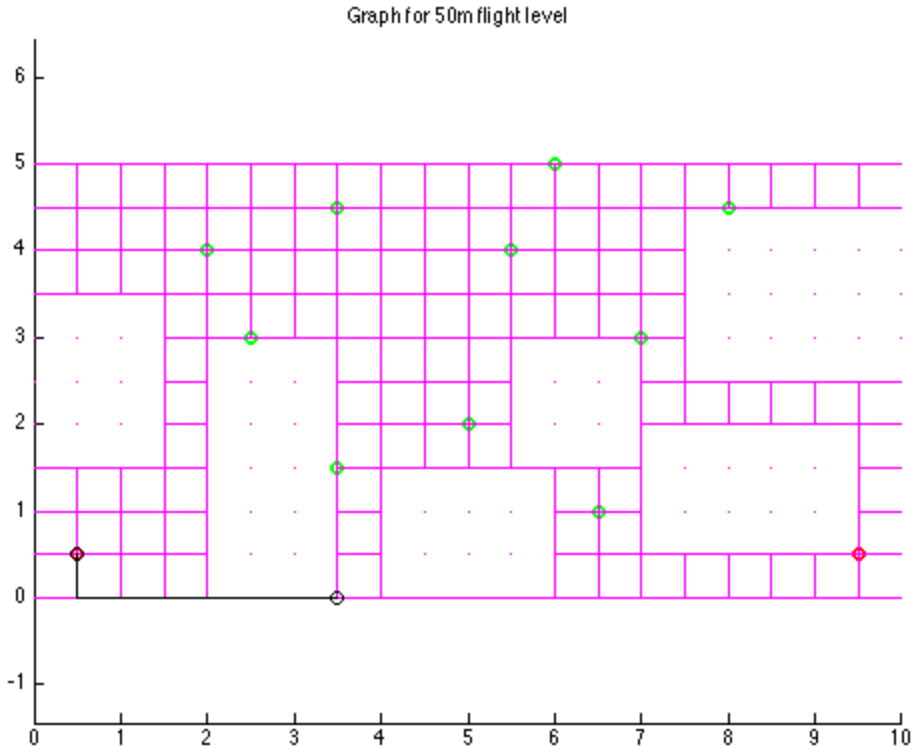
The simulation and planning is implemented in "/Q2/partC.m".


Graph for 25m flight level
6
5
4
3
2
1
0
-1
0
1
2
3
4
5
6
7
8
9
10


Graph for 50m flight level
6
5
4
3
2
1
0
-1
0
1
2
3
4
5
6
7
8
9
10

The above plots show the current ownership of graph edges at the 14 minute time step. Note that the yellow path is already on the return path by this point, and its happened to take the same path back as it took to the drop point.





The above plots show the graph connection ownership at the 39 minute time step. Notice that the red path has used the same connections that the yellow one did in in the 14 minute time step plot, which were re-allocated when the copter flying the yellow path returned to the depot.

Question 2 - Part D

Both planning schemes were run, and the total time in flight was recorded for each. The time of flight for each flight is defined as the total number of movements in minutes, plus 2 minutes, to take takeoff and landing into consideration.

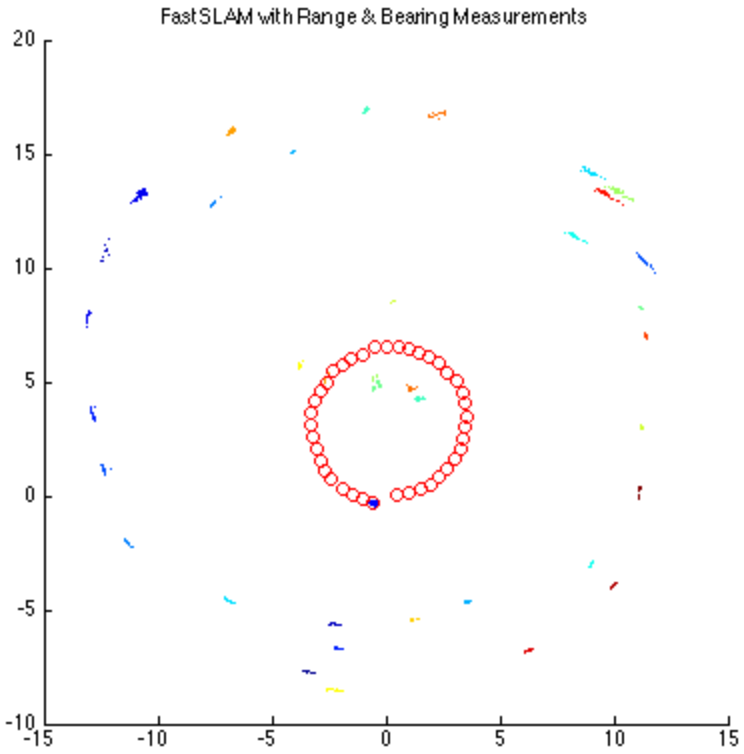
The online planner resulted in using 272 minutes of flight time, while the offline planner used 276 minutes of flight time.

The online planner is able to make a slightly lesser flight time because flights are being schedules with fewer consumed connections on the map, meaning that the bread-first search algorithm can find slightly faster paths.

Question 3

Question 3 - Part A

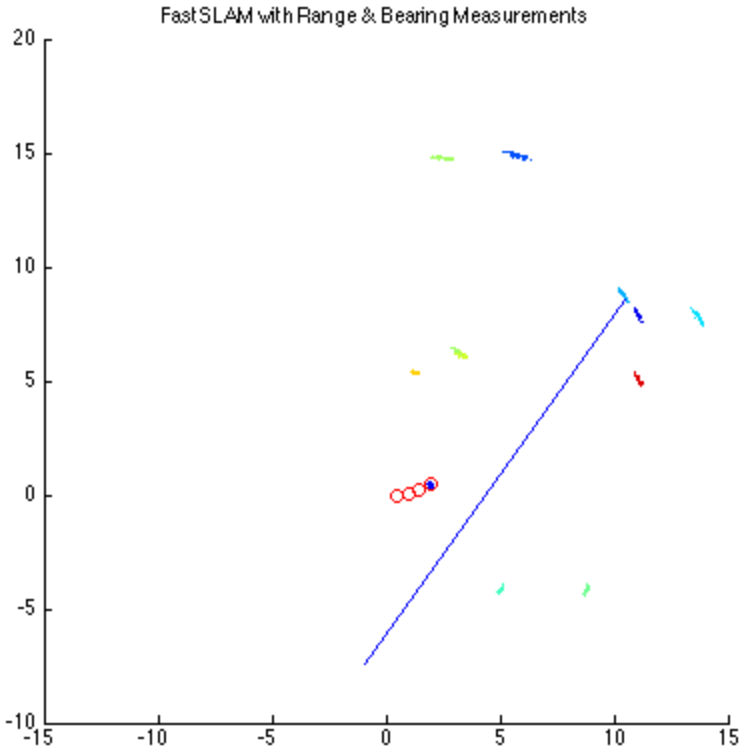
The modifications requested to the FastSLAM example were implemented in "/Q3/partA/fast_slam.m". The following is the result of a 20 second run.



The red circles show the mean particle filter robot position estimate at each timestep. The small colored blobs are the feature particles. The blue blob at the end of the robot position estimates is the particle filter's last state.

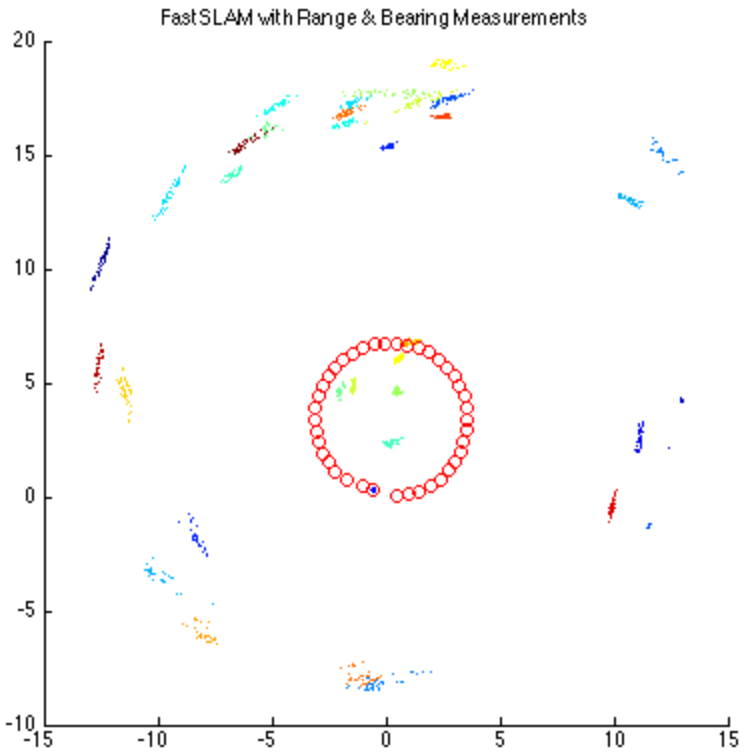
Question 3 - Part B

The intermittent error in feature identification was implemented in "/Q3/partB/fast_slam.m". The following is an instance of a particle being mis-identified.



Above a feature that is actually located near (0, -7) is being mis-identified near (10, 10).

The following is the result of a 20 second run with the error.



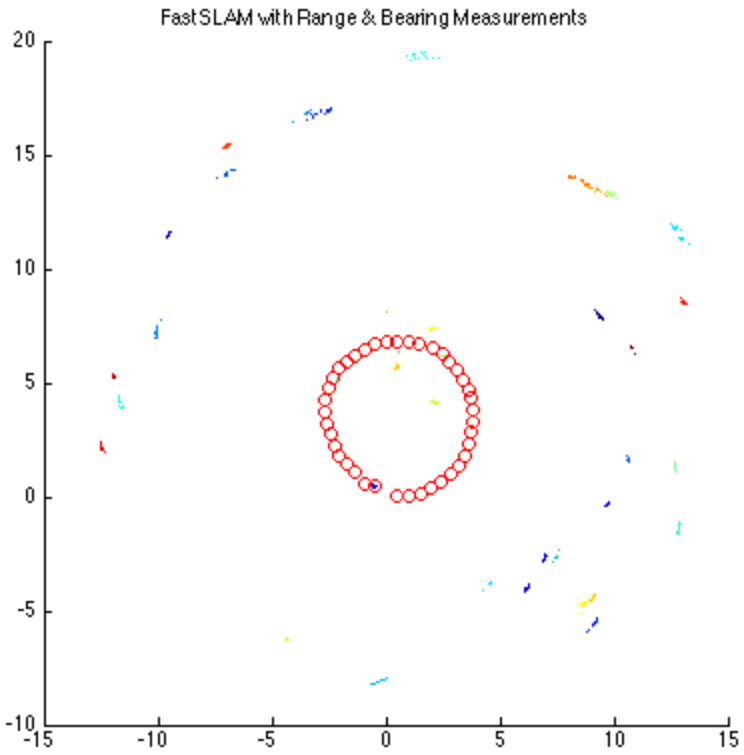
Comparing with the figure in Part A, notice that while the tracking error is not significantly increased while the feature particles are significantly more distributed. The addition of error particularly expanded the particle clouds in the direction perpendicular to the direction from which they were correctly observed.

Question 3 - Part C

Using the mean robot position estimate and the mean of the feature position particles, the outlier rejection algorithm was implemented in “/Q3/PartC/fast_slam.m”.

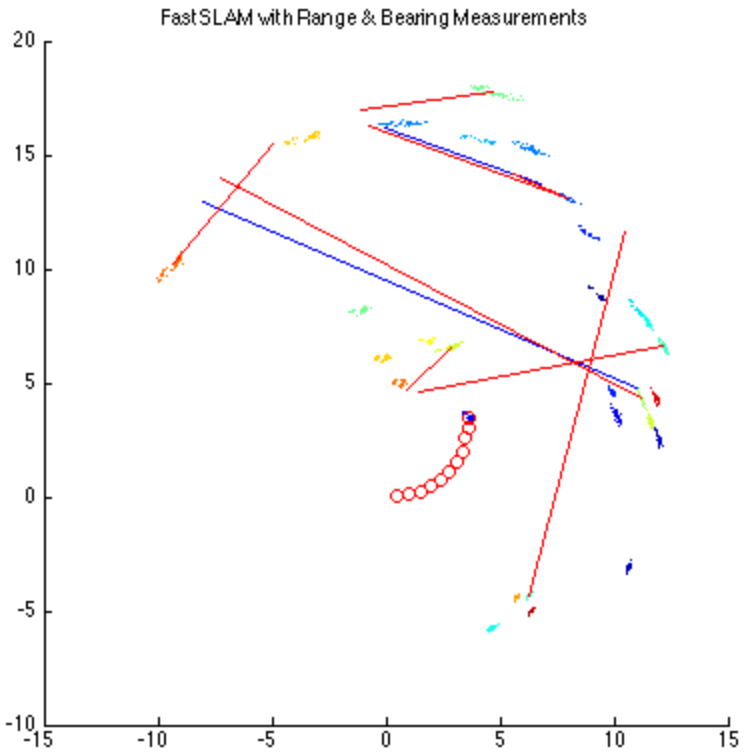
The logic used is to refuse any measurement of a particle which indicates that the particle is more than 2 meters away from where the prediction currently estimates the particle to be. This method assumes that most of the time, the first time that the robot measures a feature, it is measuring it within about 1 meter of the correct location.

Shown below are the robot position estimates and feature particle positions of a 20 second run.



Notice that there are 2 distinct types of feature particle clouds, very tightly packed clouds and fairly loosely packed clouds. The tightly packed clouds are particles first measured in the correct locations, and their positions converge as tightly as the correspondence-error-free version in Part A. The loosely packed clouds are particles which were first seen in error, and the real measurements were later removed as outliers. Since there are so many features, these lost features do not greatly impact the final position estimates.

Shown below is the algorithm rejecting outlier measurements as correspondence errors.



The blue lines show the true mis-identifications, and the red lines show the rejection of measurements based on their measurements not corresponding to their estimated positions.

Note that while many nodes are being incorrectly considered outliers, both of the actual outliers are all detected as outliers. The incorrect rejections are likely caused by initially detecting a feature as an outlier, but because they are being rejected, they do not distort the position estimates, and simply become effectively non-existent features.

Question 3 - Part D

The primary way to improve correspondence for this algorithm is to find a way to remove initial detections of features that were actually mis-identified outliers. One possible way is to look for the lack of a feature that should be in the robot's line of sight. If a particular feature has a high variance estimate with a mean within the line of sight, but it is not detected, then that estimate can be removed and the feature can be considered undiscovered.
Another way to prevent outliers from being treated as real measurements in the first place is to reject measurements to a new feature until it has been detected for some period of time/number of cycles. This way, time-variant noise (such as the noise applied to feature detection in this experiment) can not cause features to be incorrectly identified.