

MTRX5700
EXPERIMENTAL ROBOTICS

Assignment 3

Author(s):

KAUSTHUB KRISHNAMURTHY

JAMES FERRIS

SACHITH GUNAWARDHANA

SID:

312086040

311220045

440623630

Due: May 6, 2015

1 Data Fusion

Our aim was to read the measured observation data from all four sources and fuse them appropriately in order to ascertain a real representation of the robot's path. This section of the report details the methods we employed to conduct this data fusion step. The Velocity observations would provide internal data of the robot's velocities (linear and turn rate) so that its position can be segmentally ascertained using the dead reckon approximation method. However in order to account for issues such as slip (on the wheels or motors) and observational error we need to incorporate compass and GPS readings and use corrective alterations to our position prediction. Using the retro-reflective beacon readings and positions would have added an extra level of position determination by allowing us to use a local triangulation & trilateration method using the angle & distance of the beacons with respect to the robot.

1.a Program Flow & Observation Scheduling

The program requires a structure that would allow us to access different courses of action based on the input received at any given time because the order of observations being made is critical to the flow of the program. We can consider this to be four different lists of inputs that need to be scheduled which can be done with a set of flag variables which was stored in an array where if a particular type of observational event was perceived to occur a corresponding array element would be driven high. On testing this element we can allow sections of the code to be run or block them from running so that only the necessary processes occur at any given time. We detect the "next" action by storing all data in a matrix which is ordered by time and an array storing the index (with respect to that matrix) of the next event and these values increment as we access each observation. Using these indices we put the timestamps of each next event into an array, detect the smallest timestamps and test each event time (for coincidental events) for procedures that need to occur.

Iters stores the index of the next even in each observational data set that we are yet to "perceive"

$$iters = [itersVel \quad itersGPS \quad itersComp \quad itersLasern]$$

time stores the timestamps for the next index in each observational data set. This is so that we can use the *min()* function in matlab as opposed to writing the min finding code ourselves (a more elegant solution for adapting to a system with more sensors).

$$time = [timestampVel \quad timestampGPS \quad timestampComp \quad timestampLaser]$$

RunFlags are each set to 1 when their corresponding data has been perceived next with respect to time and 0 when there is another PRECEDING observation. Important to note that it will be set to 1 if there is another coincidental observation.

$$runFlags = [runVel \quad runGPS \quad runComp \quad runLasern]$$

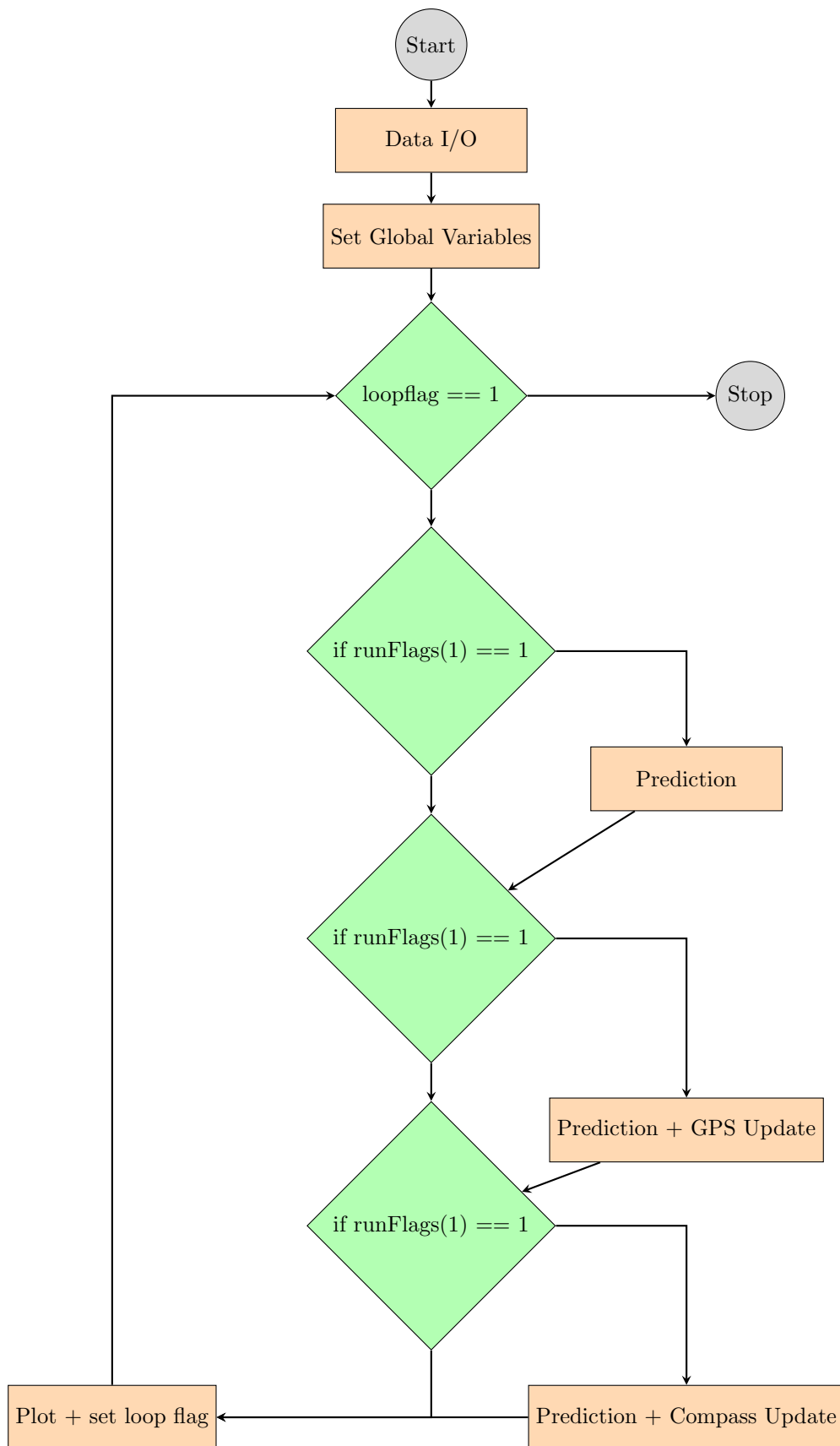


Figure 1: Overall Program Flow

1.a.i Data I/O

Having to store each set of data separately while still having easily accessible data (for ease of writing the code and minimization of code runtime) we needed to filter the data into a series of structural arrays with the following template:

ObservationDataStructure = [*timestampVel(s)* *timestampGPS* *timestampComp* *timestamp...* *measurementn*]

Process Descriptions:

- 1) Parse Input: Takes in the columns of data with space delimiting input saving the first column to "Seconds", second column to "Microseconds", and each subsequent observation column into a Measurements vector. In Figure 2 we see the structural flow for an 2-parameter data set such as Velocity Observations where Velocity would be Measurements 1 and Turn Rate would be Measurements 2.
- 2) Merge Time Stamps: An implementation of the following equation: $ts1 + ts2^{10^{-6}} - FirstTimeStamp$
This effectively "zeroes" the events so that the very first event occurs at $t = 0$

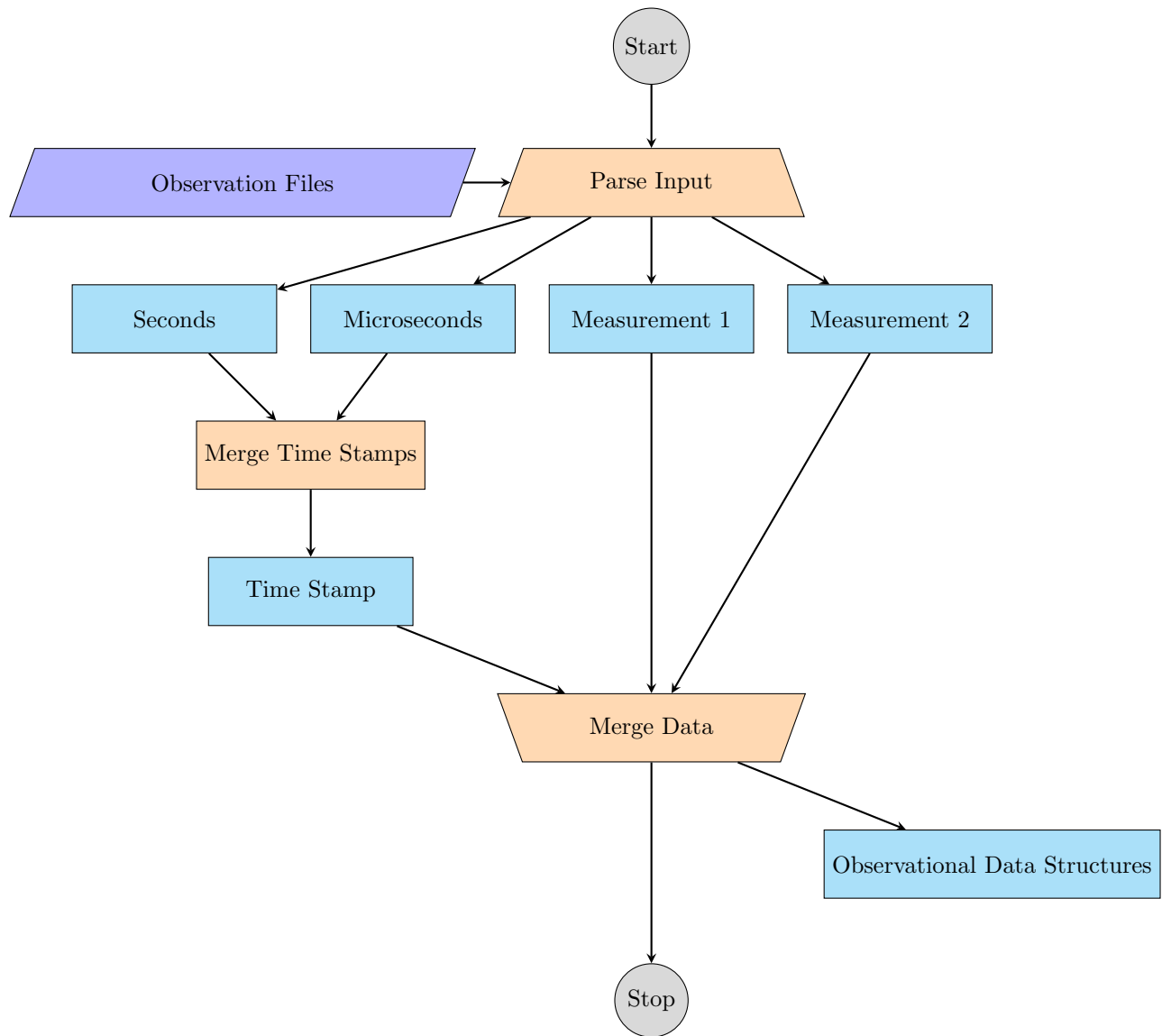


Figure 2: Data I/O Process

1.b Prediction Stage Implementation

```
1 function pr = predictionStage( XvPrev, YvPrev, THvPrev, deltaT, currTurnRate, currVel )
2 %Prediction Function given previous values
3     XvPred = XvPrev + deltaT*currVel*cos(THvPrev);
4     YvPred = YvPrev + deltaT*currVel*sin(THvPrev);
5     THvPred = THvPrev + deltaT*currTurnRate;
6     pr = [XvPred, YvPred, THvPred];
7 end
```

Steps for implementing Prediction Stage

- 1) find change in time and set new velocity observation values
- 2) run prediction stage function from above listing
- 3) set new ourX, ourY, ourHeading values
- 4) update index, time and reset flag values

1.c Prediction + Update Stage Implementation

Steps for implementing Prediction Stage

- 1) find change in time and set new velocity observation values
- 2) run prediction stage function from above listing
- 3) run the function for either GPS or Compass updating depending on which action is required.
- 4) set new ourX, ourY, ourHeading values
- 5) update index, time and reset flag values

1.c.i GPS

```
1 function gUpd = updateStageGPS(XvPred, YvPred, XvObs, YvObs, alphaP)
2
3 XvUpd = (1 - alphaP)*XvPred + alphaP*XvObs;
4 YvUpd = (1 - alphaP)*YvPred + alphaP*YvObs;
5 gUpd = [XvUpd, YvUpd];
6 end
```

1.c.ii Compass

```
1 function THvUpd = updateStageCompass(THvPred, THvObs, alphaTH)
2 a = abs(THvObs - THvPred);
3 while(a > pi)
4     if(THvObs > THvPred)
5         THvObs = THvObs - 2*pi;
6         %     THvPred = THvPred + 2*pi;
7     else
8         THvObs = THvObs + 2*pi;
9         %     THvPred = THvPred - 2*pi;
10    end
11    a = abs(THvObs - THvPred);
12 end
13
14 THvUpd = (1 - alphaTH)*THvPred + alphaTH*THvObs;
15 % THvUpd = THvPred - alphaTH*(THvObs - THvPred);
16
17 end
```

1.c.iii Laser

```
1 function mb = matchBeacons(ourX, ourY, ourHeading, lasFeat, thisrange, beaconCentreInds)
2 % Summary of this function goes here
3 % Detailed explanation goes here
4 betal = degtorad(beaconCentreInds(1)/2);
5 beta2 = degtorad(beaconCentreInds(2)/2);
6
7 if(beaconCentreInds(1) < pi/2)
8     thetal = pi/2 - betal;
9 else
10     thetal = betal - pi/2;
11 end
12
13
14 if(beaconCentreInds(1) < pi/2)
15     theta2 = pi/2 - beta2;
16 else
17     theta2 = beta2 - pi/2;
18 end
19
20 Tx1 = ourX + thisrange(1,beaconCentreInds(2))*cos(thetal+ourHeading);
21 Ty1 = ourY + thisrange(1,beaconCentreInds(2))*sin(thetal+ourHeading);
22 T1 = [Tx1 Ty1];
23 lf = lasFeat;
24 distances = sqrt(sum(bsxfun(@minus, lf, T1).^2,2));
25 T1 = lf(find(distances==min(distances)),:);
26
27
28 Tx2 = ourX + thisrange(1,beaconCentreInds(3))*cos(theta2+ourHeading);
29 Ty2 = ourY + thisrange(1,beaconCentreInds(3))*sin(theta2+ourHeading);
30 T2 = [Tx2 Ty2];
31 lf = lasFeat;
32 distances = sqrt(sum(bsxfun(@minus, lf, T2).^2,2));
33 T2 = lf(find(distances==min(distances)),:);
34
35 Tx1 = T1(1);
36 Ty1 = T1(2);
37 Tx2 = T2(1);
38 Ty2 = T2(2);
39
40 ourHeading = atan((Ty2-Ty1)/Tx2-Tx1) - atan(thisrange(beaconCentreInds(3))*sin(theta2+ourHeading
    ↪ )-thisrange(beaconCentreInds(2))*sin(thetal+ourHeading))/(thisrange(beaconCentreInds(3))*
    ↪ cos(theta2+ourHeading)-thisrange(beaconCentreInds(2))*cos(thetal+ourHeading));
41 ourX = Tx1 - thisrange(beaconCentreInds(2))*cos(thetal+ourHeading);
42 ourY = Ty1 - thisrange(beaconCentreInds(2))*sin(thetal+ourHeading);
43
44 mb = [ourX, ourY, ourHeading];
45 end
```

1.d Alpha Variation Results

What we noticed when varying alpha (assuming alpha for both compass and GPS was the same) is that the larger your reliability factor (closer to 1) the more erratic the data seemed to get as it included all error components.

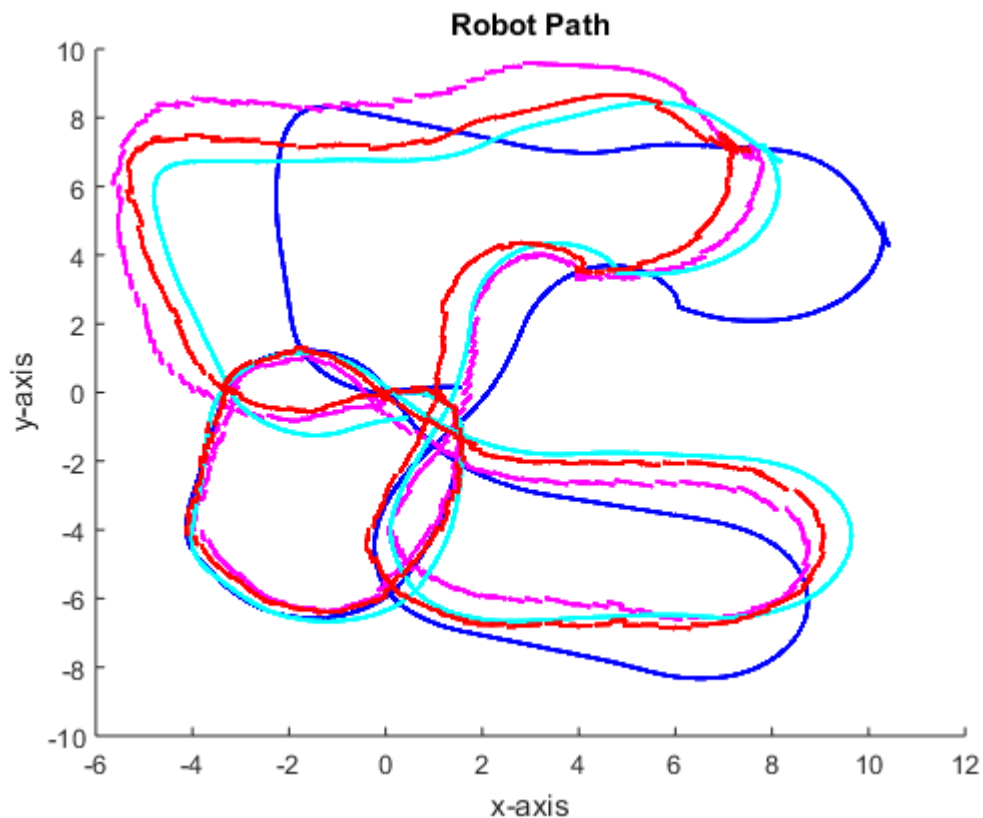


Figure 3: Robot Paths

- 1) Blue: Dead Reckoning ($\alpha_P = \alpha_{TH} = 0$)
- 2) Magenta: GPS data observed ($\alpha_P = 0.1$)
- 3) Cyan: Compass data observed ($\alpha_P = 0.1$)
- 4) Red: Compass + GPS data observed ($\alpha_P = \alpha_{TH} = 0.1$)

1.d.i $\alpha = 0.5$

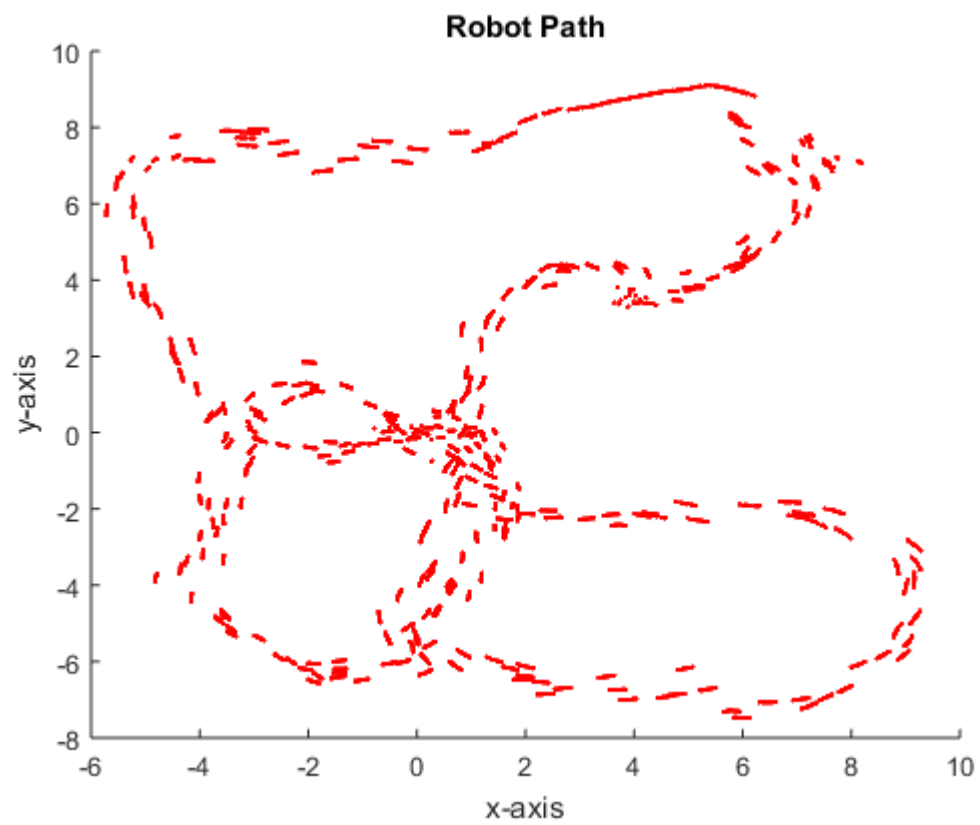


Figure 4: All Data ($\alpha = 0.5$)

1.d.ii $\alpha = 0.9$

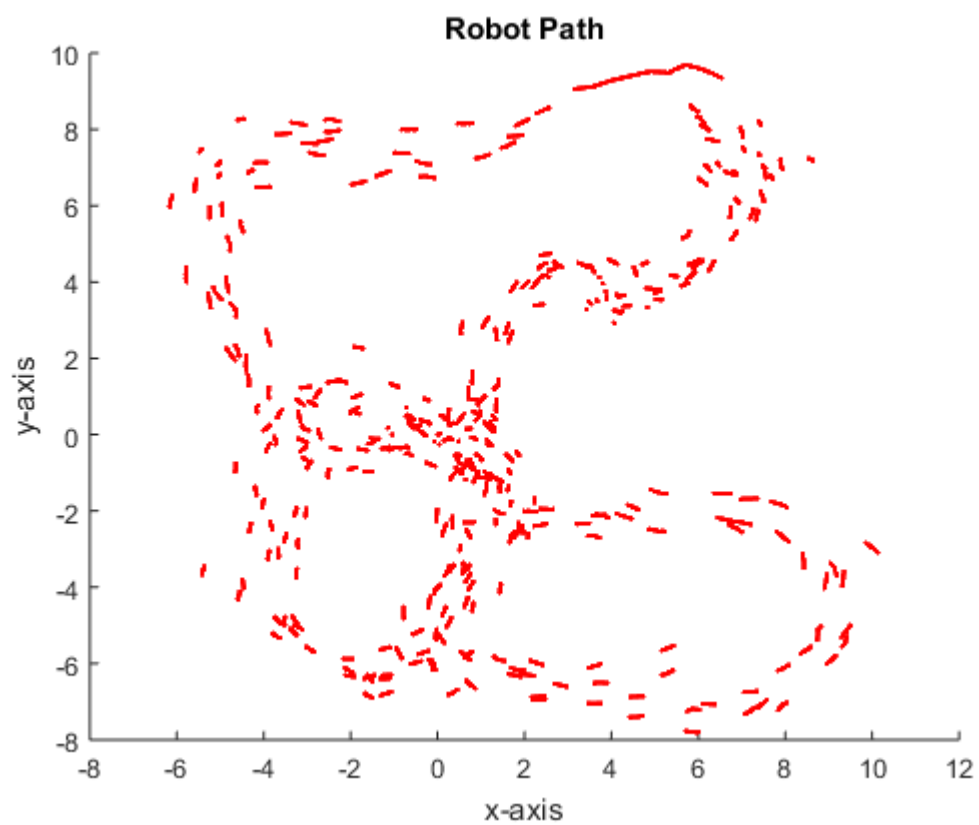


Figure 5: All Data ($\alpha = 0.9$)

1.d.iii $\alpha = 1$

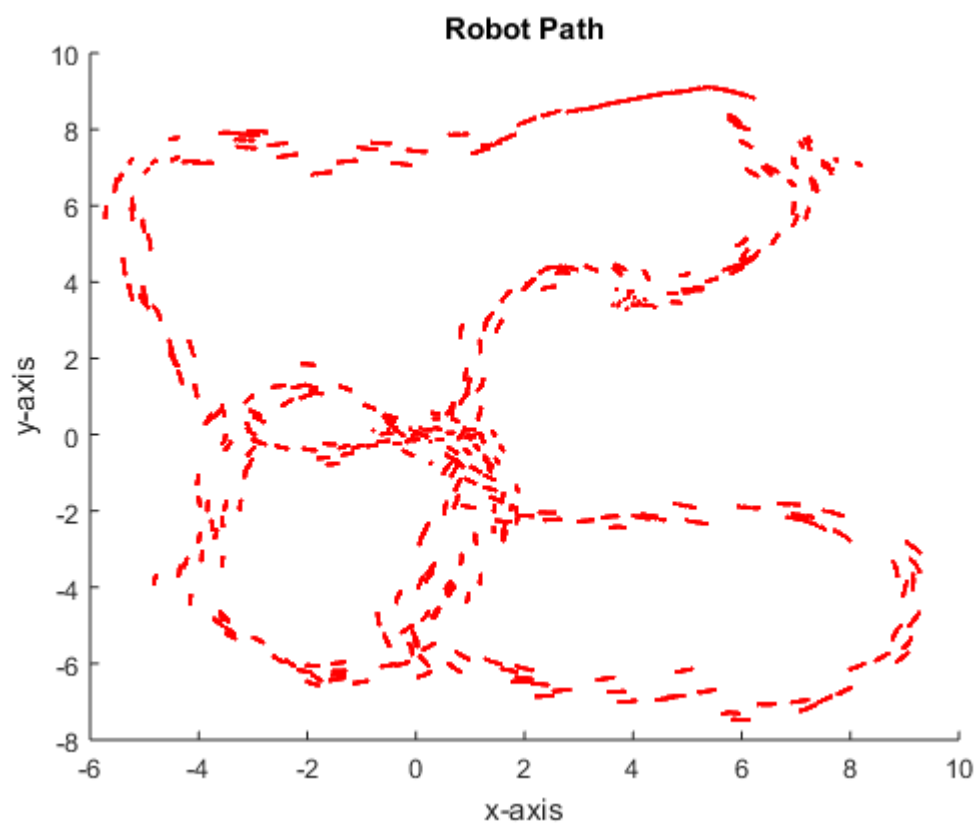


Figure 6: All Data ($\alpha = 1$)

1.e Code Listing

1.e.i Demonstration Code

This code listing is the code used at the time of demonstration.

```
1 clear
2 % close all
3 clc
4
5 DEGREES = 180/pi;
6 RADIANS = pi/180;
7 SUN = 1.496*10^8;
8 % % Prediction Stage
9 % diary './qloutput'
10 %Load observational data
11 velocityObs = load('velocityObs.txt');
12 positionObs = load('positionObs.txt');
13 compassObs = load('compassObs.txt');
14 laserObs = load('laserObs.txt');
15
16 %Get velocity data
17 time1 = velocityObs(:,1) + (velocityObs(:,2)*10^-6) - 1115116000;%get in microseconds
18 velocity = velocityObs(:,3);
19 turnRate = velocityObs(:,4);
20
21 velObs = [time1 velocity turnRate];
22
23 %Get GPS position data
24 time2 = positionObs(:,1) + (positionObs(:,2)*10^-6) - 1115116000;
25 xPos = positionObs(:,3);
26 yPos = positionObs(:,4);
27
28 posObs = [time2 xPos yPos];
29
30 %Get GPS compass data
31 time3 = compassObs(:,1) + (compassObs(:,2)*10^-6) - 1115116000;
32 heading = compassObs(:,3);
33
34 compObs = [time3 heading];
35
36 % % %get laser data
37 time4 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;
38 % % i = 1;
39 % % j = 4;
40 % % sizeLas = size(laserObs);
41 % % lasObs = zeros(sizeLas(1), 2));
42 % % while(i <= sizeLas(1))
43 % %     lasObs(i, 1) = [time4(i)];
44 % %     while(j<=sizeLas(2))
45 % %         lasObs(i, j) = laserObs(i, j);
46 % %         lasObs(i, j-1) = laserObs(i, j-1);
47 % %
48 % %         j = j + 2;
49 % %     end
50 % %
51 % %     i = i + 1;
52 % % end
53 % % size(time4)
54
55
56
57 %%postInput parsing
58
59 alphaP = 0.1;
60 alphaTH = 0.1;
61
62 lastTime = 0;
63 deltaT = 0;
64
```

```

65 latestVel = 0;
66 latestTurnRate = 0;
67
68 ourX = 0;
69 ourY = 0;
70 ourHeading = 0;
71
72 indLengths = [length(time1), length(time2), length(time3), length(time4)];
73 maxIters = max(indLengths);
74
75 % output = [lastTime, ourX, ourY, ourHeading];
76 output = zeros(maxIters, 6);
77
78 % deadreckonedpts = zeros(maxIters, 3);
79
80 %iters [velInd, posInd, compInd, lasInd];
81 iters = [2, 2, 2, 2];
82 runFlags = [0, 0, 0, 0];
83 loopFlag = 1;
84 loopCount = 2;
85 %%loop starts
86 while(loopFlag == 1)
87     loopCount = loopCount + 1
88     time = [time1(iters(1)), time2(iters(2)), time3(iters(3)), time4(iters(4)),];
89     nextT = min(time);
90
91
92     for i = 1:4
93         if time(i) == nextT
94             runFlags(i) = 1;
95         else
96             runFlags(i) = 0;
97         end
98     end
99
100     %if velocityobs
101     if(runFlags(1) == 1)
102         deltaT = time1(iters(1)) - lastTime;
103         latestVel = velObs(iters(1),2);%
104         latestTurnRate = velObs(iters(1),3);%
105         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
106         ourX = pr(1);
107         ourY = pr(2);
108         ourHeading = pr(3);
109
110         %         deadreckonedpts(1) = ourX;
111         %         deadreckonedpts(2) = ourY;
112         %         deadreckonedpts(3) = ourHeading;
113         lastTime = time1(iters(1));%
114         runFlags(1) = 0;
115         if iters(1) == length(time1)
116             time1(iters(1)) = SUN;
117         else
118             iters(1) = iters(1) + 1;
119         end
120     end
121
122     % % if GPS
123     if(runFlags(2) == 1)
124         deltaT = time2(iters(2)) - lastTime;
125         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
126         gUpd = updateStageGPS(pr(1), pr(2), posObs(iters(2),2), posObs(iters(2),3), alphaP); %xvobs
127         ↪ and yvobs need to come from the file
128         ourX = gUpd(1);
129         ourY = gUpd(2);
130         ourHeading = pr(3);
131         lastTime = time2(iters(2));%
132         runFlags(2) = 0;
133         if iters(2) == length(time2)
134             time2(iters(2)) = SUN;
135         else

```

```

135         iters(2) = iters(2) + 1;
136     end
137 end
138 %
139 % % if compass
140 if(runFlags(3) == 1)
141     deltaT = time3(iters(3)) - lastTime;
142     pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
143     cUpd = updateStageCompass(pr(3), compObs(iters(3),2), alphaTH);
144     ourX = pr(1);
145     ourY = pr(2);
146     ourHeading = cUpd;
147     lastTime = time3(iters(3));
148     runFlags(3) = 0;
149     if iters(3) == length(time3)
150         time3(iters(3)) = SUN;
151     else
152         iters(3) = iters(3) + 1;
153     end
154 end
155 %
156 % % if laser data
157 if(runFlags(4) == 1)
158 % %         fill this in
159     runFlags(4) = 0;
160     time4(iters(4)) = SUN;    %remove this line
161 end
162
163 %check loop
164 if(time1(iters(1)) == SUN)
165     if(time2(iters(2)) == SUN)
166         if(time3(iters(3)) == SUN)
167             if(time4(iters(4)) == SUN)
168                 loopFlag = 0;
169             end
170         end
171     end
172 end
173
174 %plot stuff
175 hold on
176 title('Robot Path');
177 xlabel('x-axis');
178 ylabel('y-axis');
179 % legend('Dead Reckoning');
180 % drawnow
181
182 plot(ourX, ourY, 'r. ');
183 output(loopCount, 1) = lastTime;
184 output(loopCount, 2) = ourX;
185 output(loopCount, 3) = ourY;
186 output(loopCount, 4) = ourHeading;
187 output(loopCount, 5) = latestVel;
188 output(loopCount, 6) = latestTurnRate;
189 end
190
191 % diary ON
192 % output
193 % diary OFF
194
195 output(:,1) = output(:,1); %+ 1115116000;

```

1.e.ii Development Code

This code listing includes a partially complete implementation of fusing the data from the Laser Range Finder detection of retro reflective beacons. However it was not demonstrated as the incomplete code led to program flow issues that wouldn't allow us to run it properly.

```
1 clear
2 close all
3 clc
4
5 DEGREES = 180/pi;
6 RADIANS = pi/180;
7 SUN = 1.496*10^8;
8
9 %Load observational data
10 velocityObs = load('velocityObs.txt');
11 positionObs = load('positionObs.txt');
12 compassObs = load('compassObs.txt');
13 laserObs = load('laserObs.txt');
14 laserFeat = load('laserFeatures.txt');
15
16 %get laser features
17 lasFeat = [laserFeat(:,1) laserFeat(:,2)];
18
19 %Get velocity data
20 time1 = velocityObs(:,1) + (velocityObs(:,2)*10^-6) - 1115116000;%get in microseconds
21 velocity = velocityObs(:,3);
22 turnRate = velocityObs(:,4);
23
24 velObs = [time1 velocity turnRate];
25
26 %Get GPS position data
27 time2 = positionObs(:,1) + (positionObs(:,2)*10^-6) - 1115116000;
28 xPos = positionObs(:,3);
29 yPos = positionObs(:,4);
30
31 posObs = [time2 xPos yPos];
32
33 %Get GPS compass data
34 time3 = compassObs(:,1) + (compassObs(:,2)*10^-6) - 1115116000;
35 heading = compassObs(:,3);
36
37 compObs = [time3 heading];
38
39 % % %get laser data
40 time4 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;
41
42 f1=1;
43
44 range = zeros(length(laserObs), (size(laserObs,2)-2)/2);
45 intensity = zeros(length(laserObs), (size(laserObs,2)-2)/2);
46
47 %Extracting range & intensity data from LaserObs
48 for i=1:length(laserObs)
49     for f2=3:2:size(laserObs,2)
50         range(i,f1)=laserObs(i,f2);
51         intensity(i,f1)=laserObs(i,f2+1);
52         f1=f1+1;
53     end
54     f1=1;
55 end
56
57
58 % postInput parsing
59
60 alphaP = 0.1;
61 alphaTH = 0.1;
62
63 lastTime = 0;
64 deltaT = 0;
```

```

65
66 latestVel = 0;
67 latestTurnRate = 0;
68
69 ourX = 0;
70 ourY = 0;
71 ourHeading = 0;
72
73 indLengths = [length(time1), length(time2), length(time3), length(time4)];
74 maxIters = max(indLengths);
75
76 % output = [lastTime, ourX, ourY, ourHeading];
77 output = zeros(maxIters, 6);
78
79 % deadreckonedpts = zeros(maxIters, 3);
80
81 %iters [velInd, posInd, compInd, lasInd];
82 iters = [2, 2, 2, 2];
83 runFlags = [0, 0, 0, 0];
84 loopFlag = 1;
85 loopCount = 2;
86 %%loop starts
87 while(loopFlag == 1)
88     loopCount = loopCount + 1
89     time = [time1(iters(1)), time2(iters(2)), time3(iters(3)), time4(iters(4)),];
90     nextT = min(time);
91
92
93     for i = 1:4
94         if time(i) == nextT
95             runFlags(i) = 1;
96         else
97             runFlags(i) = 0;
98         end
99     end
100
101     %if velocityobs
102     if(runFlags(1) == 1)
103         deltaT = time1(iters(1)) - lastTime;
104         latestVel = velObs(iters(1),2);%
105         latestTurnRate = velObs(iters(1),3);%
106         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
107         ourX = pr(1);
108         ourY = pr(2);
109         ourHeading = pr(3);
110
111         %         deadreckonedpts(1) = ourX;
112         %         deadreckonedpts(2) = ourY;
113         %         deadreckonedpts(3) = ourHeading;
114         lastTime = time1(iters(1));%
115         runFlags(1) = 0;
116         if iters(1) == length(time1)
117             time1(iters(1)) = SUN;
118         else
119             iters(1) = iters(1) + 1;
120         end
121     end
122
123     % % if GPS
124     if(runFlags(2) == 1)
125         deltaT = time2(iters(2)) - lastTime;
126         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
127         gUpd = updateStageGPS(pr(1), pr(2), posObs(iters(2),2), posObs(iters(2),3), alphaP); %xvobs
128         ↪ and yvobs need to come from the file
129         ourX = gUpd(1);
130         ourY = gUpd(2);
131         ourHeading = pr(3);
132         lastTime = time2(iters(2));%
133         runFlags(2) = 0;
134         if iters(2) == length(time2)
135             time2(iters(2)) = SUN;

```

```

135         else
136             iters(2) = iters(2) + 1;
137         end
138     end
139 %
140 % % if compass
141 if(runFlags(3) == 1)
142     deltaT = time3(iters(3)) - lastTime;
143     pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
144     cUpd = updateStageCompass(pr(3), compObs(iters(3),2), alphaTH);
145     ourX = pr(1);
146     ourY = pr(2);
147     ourHeading = cUpd;
148     lastTime = time3(iters(3));
149     runFlags(3) = 0;
150     if iters(3) == length(time3)
151         time3(iters(3)) = SUN;
152     else
153         iters(3) = iters(3) + 1;
154     end
155 end
156 %
157 % % if laser data
158 if(runFlags(4) == 1)
159 %     find beacons
160     deltaT = time4(iters(4)) - lastTime;
161     pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
162     ourX = pr(1);
163     ourY = pr(2);
164     ourHeading = pr(3);
165
166     intensityTrav = 1;
167     beaconCentreInds = 0;
168     %find beacon centres
169     while(intensityTrav <= 361)
170         if(intensity(intensityTrav) == 1)
171             iStart = intensityTrav;
172             intensityTrav = intensityTrav + 1;
173             while(intensity(intensityTrav) == 1 && (abs(range(intensityTrav) - range(
174                 ↪ intensityTrav + 1) < 1)))
175                 intensityTrav = intensityTrav + 1;
176             end
177             iEnd = intensityTrav - 1;
178             iMid = (iEnd - iStart)/2;
179             beaconCentreInds = cat(2, beaconCentreInds, iMid);
180         else
181             intensityTrav = intensityTrav + 1;
182         end
183     end
184 %     project to real world
185     mb = matchBeacons(ourX, ourY, ourHeading, lasFeat, range(iters(4),:), beaconCentreInds);
186     ourX = mb(1);
187     ourY = mb(2);
188     ourHeading = mb(3);
189
190     lastTime = time4(iters(4));
191     runFlags(4) = 0;
192     if iters(4) == length(time4)
193         time3(iters(4)) = SUN;
194     else
195         iters(4) = iters(4) + 1;
196     end
197 end
198 %check loop
199 if(time1(iters(1)) == SUN)
200     if(time2(iters(2)) == SUN)
201         if(time3(iters(3)) == SUN)
202             if(time4(iters(4)) == SUN)
203                 loopFlag = 0;
204             end

```



```
205         end
206     end
207 end
208
209 %plot stuff
210 hold on
211 title('Robot Path');
212 xlabel('x-axis');
213 ylabel('y-axis');
214 % legend('')
215 % drawnow
216
217 plot(ourX, ourY, 'r.');
218 output(loopCount, 1) = lastTime;
219 output(loopCount, 2) = ourX;
220 output(loopCount, 3) = ourY;
221 output(loopCount, 4) = ourHeading;
222 output(loopCount, 5) = latestVel;
223 output(loopCount, 6) = latestTurnRate;
224 end
```

2 Question 2

2.a Obtaining Obstacle Location from Laser Data

First we start by taking the data output from question one, which consisted of the robot $x - y$ coordinates in the world coordinate system, as well as the velocity and turn rate for that particular timestamp, for each timestamp that occurs in the Velocity observation data, the Compass data and the GPS data.

Similar to the way question one works, we combine this data with the Laser observation data by comparing timestamps, using the Prediction Stage equation to estimate the robots $x - y$ coordinates at the time that the laser data was generated. Combining the robots $x - y$ world coordinates with the relative position of the obstacles obtained from the Laser data.

For initial tests to attempt to identify obstacles, any range reading from the Laser data that was less than eight (the maximum range of the sensor) was considered an obstacle. It was intended to apply filters to this data once the Occupancy Grid had been generated. Until then, raw data would be used.

The matlab code is shown below:

```

1 clear
2 clc
3 close all
4
5 DEGREES = 180/pi;
6 RADIANS = pi/180;
7
8 %Load data generated from Q1
9 positionData = load('qloutput1.txt');
10
11 %Load laser observation data
12 laserObs = load('laserObs.txt');
13
14 %Get output data
15 time1 = positionData(:,1);
16 Xpos = positionData(:,2);
17 Ypos = positionData(:,3);
18 heading = positionData(:,4);
19 velocity = positionData(:,5);
20 turnRate = positionData(:,6);
21
22 %Setup output recording
23 diary './q2Output4'
24
25 %Get laser data
26 time2 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;%get in microseconds
27
28
29 %Extracting range from LaserObs
30
31 f1=1;
32 range = zeros(length(laserObs), (size(laserObs,2)-2)/2);
33
34 %Extracting range & intensity data from LaserObs
35 for i=1:length(laserObs)
36     for f2=3:2:size(laserObs,2)
37         range(i,f1)=laserObs(i,f2);
38         f1=f1+1;
39     end
40     f1=1;
41 end
42
43 lasersX = 0;
44 lasersY = 0;
45
46 % alphaP = 0.5;
47 % alphaTH = 0.5;
48
49 lastTime = 0;
50 deltaT = 0;
51
52 latestVel = 0;
53 latestTurnRate = 0;
54
55 ourX = 0;
56 ourY = 0;
57 ourHeading = 0;
58
59 indLengths = [length(time1), length(time2)];
60 maxIters = max(indLengths);
61
62 interval = 20;
63
64 %iters [velInd, posInd, compInd, lasInd];iters = [2, 2];
65 runFlags = [0, 0];
66 loopFlag = 1;
67 loopCount = 2;
68
69 %%loop starts
70

```

```

71
72 while(loopFlag == 1)
73     loopCount = loopCount + 1;
74     time = [time1(iters(1)), time2(iters(2))];
75     nextT = min(time);
76
77
78     for i = 1:2
79         if time(i) == nextT
80             runFlags(i) = 1;
81         else
82             runFlags(i) = 0;
83         end
84     end
85
86     %if Positiondata
87     if(runFlags(1) == 1)
88         deltaT = time1(iters(1)) - lastTime;
89         latestVel = velocity(iters(1));
90         latestTurnRate = turnRate(iters(1));
91         ourX = Xpos(iters(1));
92         ourY = Ypos(iters(1));
93         ourHeading = heading(iters(1));
94
95         lastTime = time1(iters(1));%
96         runFlags(1) = 0;
97         if iters(1) >= length(time1)-interval
98             time1(iters(1)) = 1.496*10^8;
99         else
100             iters(1) = iters(1) + interval;
101         end
102     end
103
104     % % if Laser Observation Data
105     if(runFlags(2) == 1)
106         deltaT = time2(iters(2)) - lastTime;
107         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
108         ourX = pr(1);
109         ourY = pr(2);
110         ourHeading = pr(3);
111         lastTime = time2(iters(2));%
112         runFlags(2) = 0;
113
114
115         % Get X,Y coordinates for laser ob data
116
117         for i = 1:size(range,2)
118             if (range(iters(2),i) < 8.0 && range(iters(2),i) > 0.0001)
119                 lasersX = ourX + range(iters(2),i)*cos(((i-1)*0.5)*RADIANS+ourHeading);
120                 lasersY = ourY + range(iters(2),i)*sin(((i-1)*0.5)*RADIANS+ourHeading);
121                 diary ON
122                 fprintf('%d\t%d\n', lasersX, lasersY);
123                 diary OFF
124             end
125         end
126
127         %increment
128         if iters(2) >= length(time2)-interval
129             time2(iters(2)) = 1.496*10^8;
130         else
131             iters(2) = iters(2) + interval;
132         end
133
134     end
135
136
137     %check loop
138     if(time1(iters(1)) == 1.496*10^8)
139         if(time2(iters(2)) == 1.496*10^8)
140             loopFlag = 0;
141         end

```

```

142     end
143
144     %plot stuff
145     % hold on
146     % title('Obstacles');
147     % xlabel('x-axis');
148     % ylabel('y-axis');
149     % legend('')
150     % drawnow
151 end

```

2.b Generate Occupancy Grid

To generate the Occupancy Grid, we took the Data set of the X-Y coordinates of all 'Obstacles' detected, and determined the difference between the minimum and maximum x and y values detected. Given a user defined size for the occupancy grid, we could then determine the $x - y$ range that corresponded to a grid location. Then for every Obstacle $x - y$ coordinate we could determine the grid location it corresponded to, incrementing the grid value to increase the weighting, which would indicate the likelihood of an obstacle being in that region. Some experimenting with the grid size using the data for the Robot path generated in question one indicated that a grid size of 200×200 would be best, as this resulted in a grid map that, while not extremely sharp, was also not extremely blurred. Ideally, the grid would be vague enough to generalise a position for the obstacles, but not so clear that it simply resulted in a plot of every possible obstacle coordinates. See the figures below:

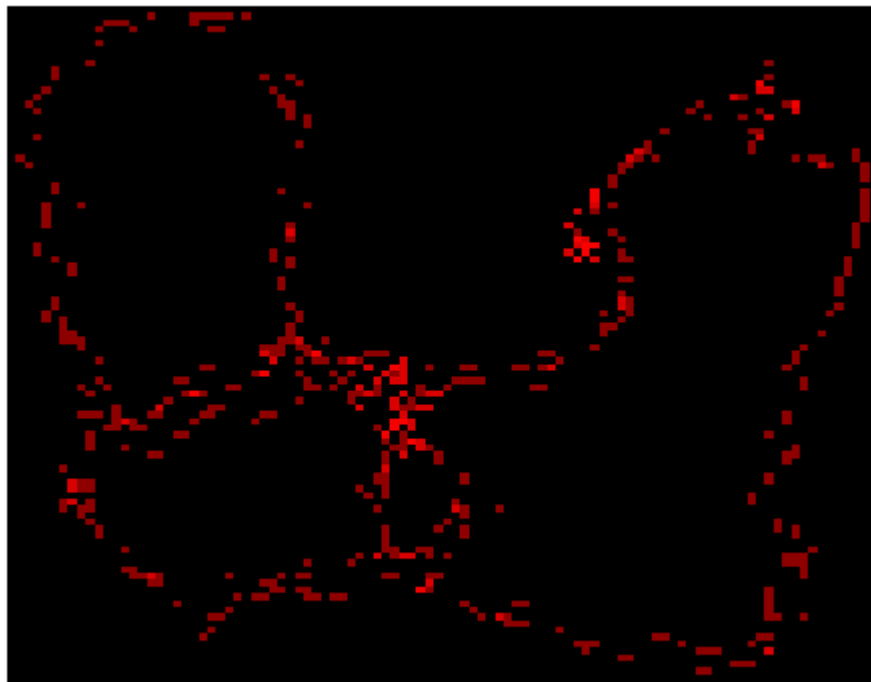


Figure 7: Grid size = 100×100

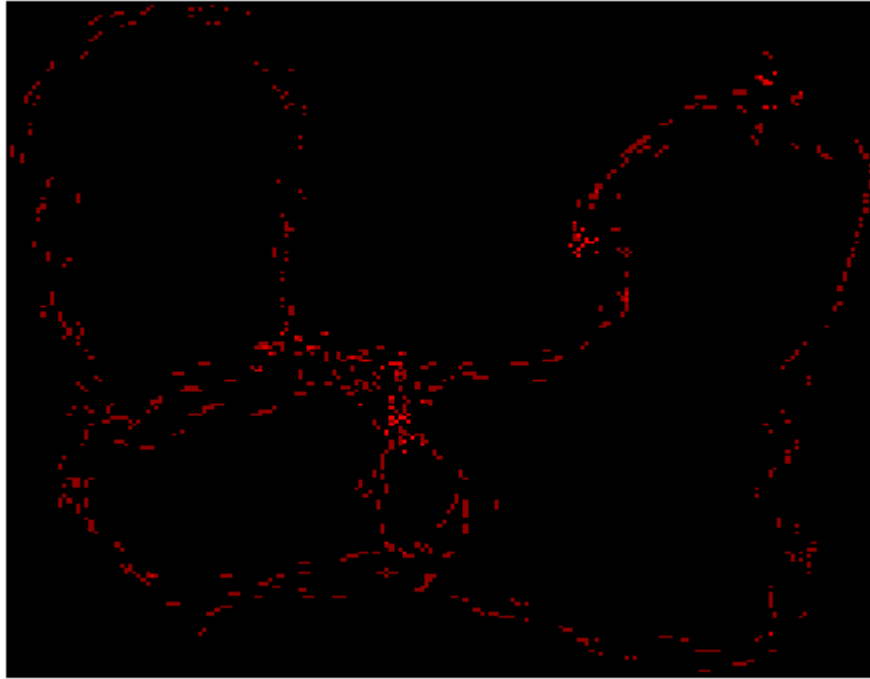


Figure 8: Grid size = 200×200

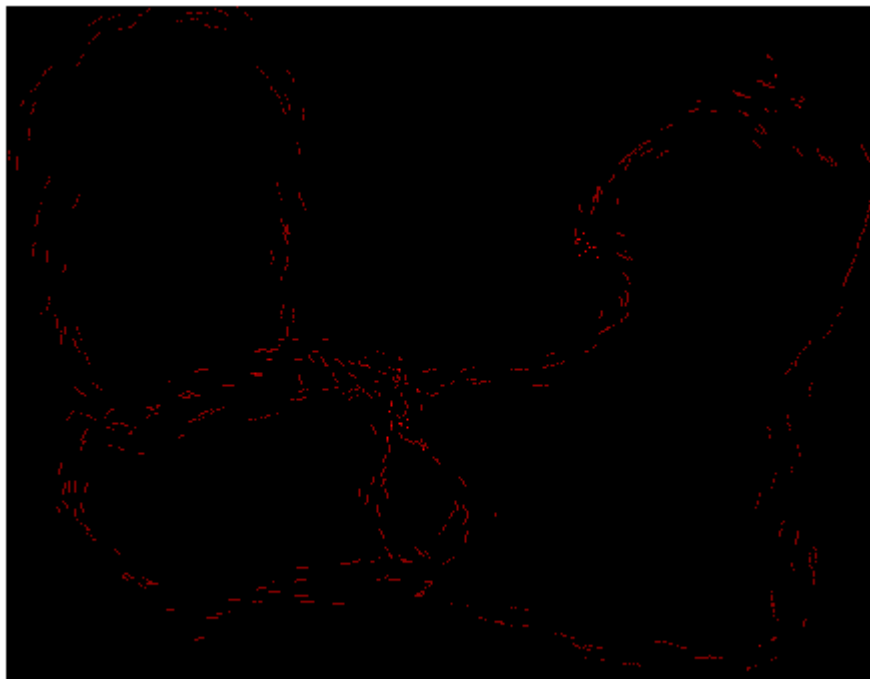


Figure 9: Grid size = 500×500

The matlab code for the occupancy grid is shown below:

```
1 clear
2 close all
3 clc
4
5 % load output from the obtain_obstacles code
6 positionData = load('laserPosiitons.txt');
7
8 xPos = positionData(:,1);
9 yPos = positionData(:,2);
10
11 %Get maximum and minimum x-y values
12 xMin = min(xPos);
13 xMax = max(xPos);
14
15 yMin = min(yPos);
16 yMax = max(yPos);
17
18 %define a square grid size
19 gridSize = 100;
20
21 grid = zeros(gridSize);
22
23 %Determine the difference between each grid location
24 yDiff = (yMax - yMin)/(gridSize - 2);
25 xDiff = (xMax - xMin)/(gridSize - 2);
26
27 %Determine which grid loaction each coordinate corresponds to.
28 for i = 1:length(xPos)
29     tmpX = xPos(i);
30     tmpY = yPos(i);
31     j = 1;
32     while (tmpX > xMin)
33         tmpX = tmpX - xDiff;
34         j = j + 1;
35     end
36     k = 1;
37     while (tmpY > yMin)
38         tmpY = tmpY - yDiff;
39         k = k + 1;
40     end
41     grid(j,k) = grid(j,k) + 1;
42 end
43
44 %Generate occupancy grid
45 HeatMap(grid);
```

2.c Results

The main problem with generating the obstacle data is that the program used took far too long to run. In order to find the obstacles detected for each timestamp, the program took nearly forty minutes to run. The text file containing the laser observation data was nearly eight megabytes in size, so a long run time was expected, however the actual time taken was excessive. In order to get results in an appropriate time, the code was modified to iterate through the timestamps in jumps of twenty, resulting in a run time of a few minutes. Although this would reduce the accuracy of the robot location slightly, the tradeoff was considered acceptable to get an output in a decent timeframe.

The second problem came when the obstacle location data was compiled into a heatmap. See the images below:

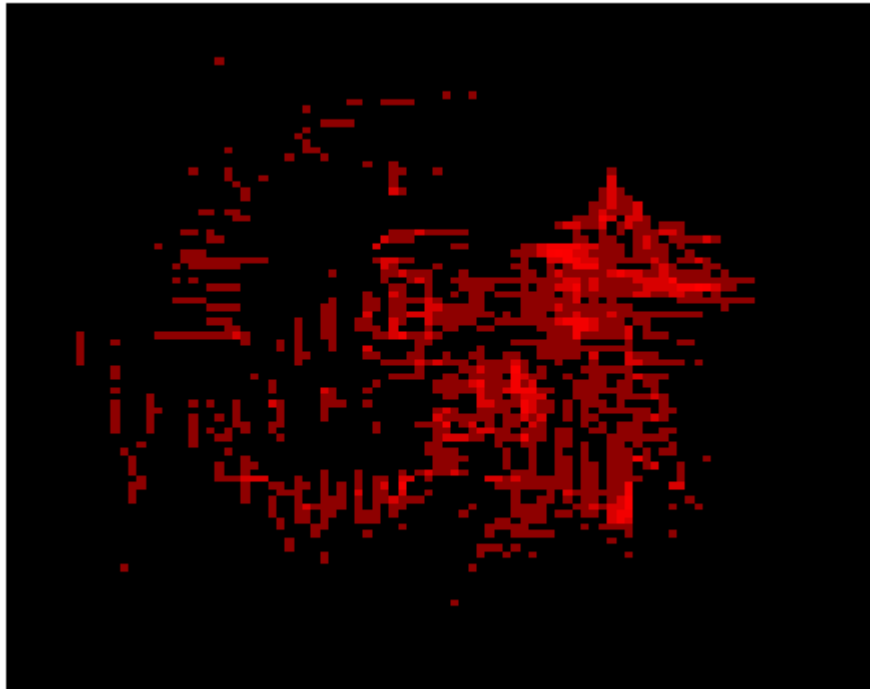


Figure 10: Grid size = 200×200

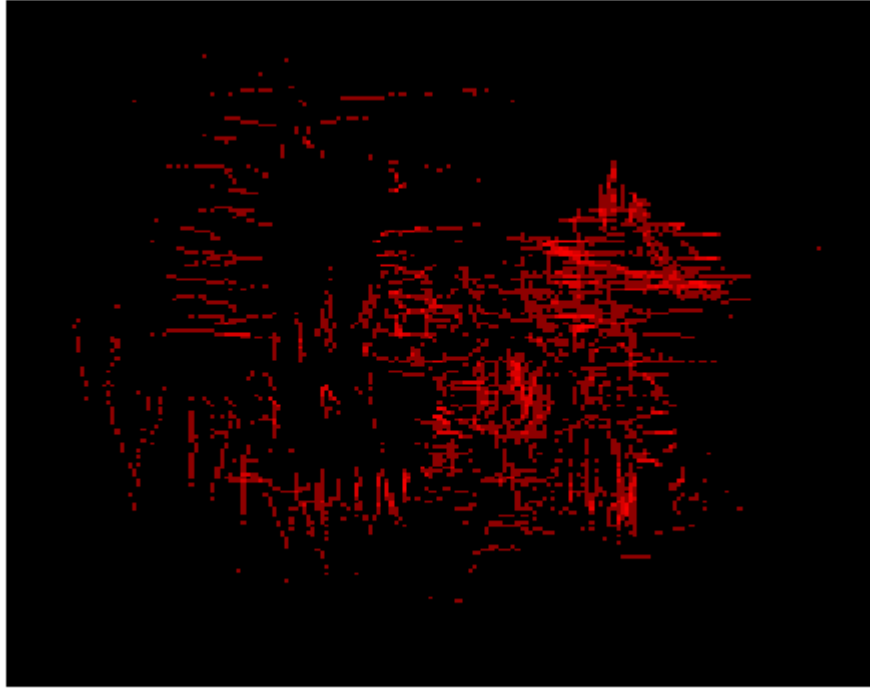


Figure 11: Grid size = 500×500

As can be seen, the obstacles are not accurately generated. It appears as if the obstacles/walls detected are in fact rotating. This suggests a flaw in the conversion between the robots coordinate system and the world coordinate system. The code lines that governed this are shown below:

```
1 for i = 1:size(range,2)
2     if (range(iters(2),i) < 8.0 && range(iters(2),i) > 0.0001)
3         lasersX = ourX + range(iters(2),i)*cos(((i-1)*0.5)*RADIANS+ourHeading);
4         lasersY = ourY + range(iters(2),i)*sin(((i-1)*0.5)*RADIANS+ourHeading);
5         diary ON
6         fprintf('%d\t%d\n', lasersX, lasersY);
7         diary OFF
8     end
9 end
```

Given the issues with the original code in regards to runtime, as well as the issues converting to the correct coordinate system, we decided to re-write the code to generate obstacle data. To do this, we utilised the laserShowAcfr.m file given for Assignment 2 and adapt it slightly to fit in our timestamp matching system. The resulting code is shown below:

```
1 clear
2 clc
3 close all
4
5 DEGREES = 180/pi;
6 RADIANS = pi/180;
7
8
9 %Load data generated from Q1
10 positionData = load('qloutput1.txt');
11
12 %Load laser observation data
13 laserObs = load('laserObs.txt');
14
15 %Get output data
16 time1 = positionData(:,1);
17 Xpos = positionData(:,2);
18 Ypos = positionData(:,3);
19 heading = positionData(:,4);
20 velocity = positionData(:,5);
21 turnRate = positionData(:,6);
22
23 %Get laser data
24 time2 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;%get in microseconds
25
26 lasersX = 0;
27 lasersY = 0;
28
29 % alphaP = 0.5;
30 % alphaTH = 0.5;
31
32 lastTime = 0;
33 deltaT = 0;
34
35 latestVel = 0;
36 latestTurnRate = 0;
37
38 ourX = 0;
39 ourY = 0;
40 ourHeading = 0;
41
42 indLengths = [length(time1), length(time2)];
43 maxIters = max(indLengths);
44
45 interval = 20;
46
47 iters = [2, 2];
48 runFlags = [0, 0];
49 loopFlag = 1;
50 loopCount = 2;
51
52 xPos = zeros(1);
53 yPos = zeros(1);
54
55 %%loop starts
56
57
58 while(loopFlag == 1)
59     loopCount = loopCount + 1;
60     time = [time1(iters(1)), time2(iters(2))];
61     nextT = min(time);
62
63
64     for i = 1:2
```

```

65         if time(i) == nextT
66             runFlags(i) = 1;
67         else
68             runFlags(i) = 0;
69         end
70     end
71
72     %if Positiondata
73     if(runFlags(1) == 1)
74         deltaT = time1(iters(1)) - lastTime;
75         latestVel = velocity(iters(1));
76         latestTurnRate = turnRate(iters(1));
77         ourX = Xpos(iters(1));
78         ourY = Ypos(iters(1));
79         ourHeading = heading(iters(1));
80
81         lastTime = time1(iters(1));%
82         runFlags(1) = 0;
83         if iters(1) >= length(time1)-interval
84             time1(iters(1)) = 1.496*10^8;
85         else
86             iters(1) = iters(1) + interval;
87         end
88     end
89
90     % % if laser observation data
91     if(runFlags(2) == 1)
92         deltaT = time2(iters(2)) - lastTime;
93         pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
94         ourX = pr(1);
95         ourY = pr(2);
96         ourHeading = pr(3);
97         lastTime = time2(iters(2));%
98         runFlags(2) = 0;
99
100         %Begin here modified code extract from laserShowAcfr.m from Assignment 2
101         xpoint = zeros(1);
102         ypoint = zeros(1);
103         for j = 4:2:size(laserObs,2)
104             range = laserObs(iters(2),j-1);
105             bearing = ((j)/2 - 90)*pi/180;
106             if (range < 8.0)
107                 xpoint = [xpoint ourX+range*cos(bearing + ourHeading)];
108                 ypoint = [ypoint ourY+range*sin(bearing + ourHeading)];
109             end
110
111         end
112
113         %End extract
114
115         xPos = [xPos xpoint];
116         yPos = [yPos ypoint];
117
118         plot(xpoint(:), ypoint(:), '.');
119
120         %increment
121         if iters(2) >= length(time2)-interval
122             time2(iters(2)) = 1.496*10^8;
123         else
124             iters(2) = iters(2) + interval;
125         end
126
127     end
128
129
130     %check loop
131     if(time1(iters(1)) == 1.496*10^8)
132         if(time2(iters(2)) == 1.496*10^8)
133             loopFlag = 0;
134         end
135     end

```

```
136
137 %plot stuff
138 % hold on
139 % title('Obstacles');
140 % xlabel('x-axis');
141 % ylabel('y-axis');
142 % legend('')
143 drawnow
144 % pause
145 end
146
147 xpoint(1) = [];
148 ypoint(1) = [];
149
150 xPos(1) = [];
151 yPos(1) = [];
```

This code was able to run for every timestamp in less than a minute. It was deduced that the reason for this is that, for the original code, we were reading the entire range reading from the laserObs.txt file into a matrix at the beginning of the program. This resulted in a matrix of size 2836, filled with doubles. This took up a large amount of memory, and caused accessing it to take huge amounts of time. The laserShowAcfr.m code from Assignment 2 instead only read a single line from the laserObs.txt at the time at which operations occurred on it, severely reducing the memory used, allowing the program to execute much faster.

Despite this breakthrough, we still had issues. The resulting occupancy grid can be found below:

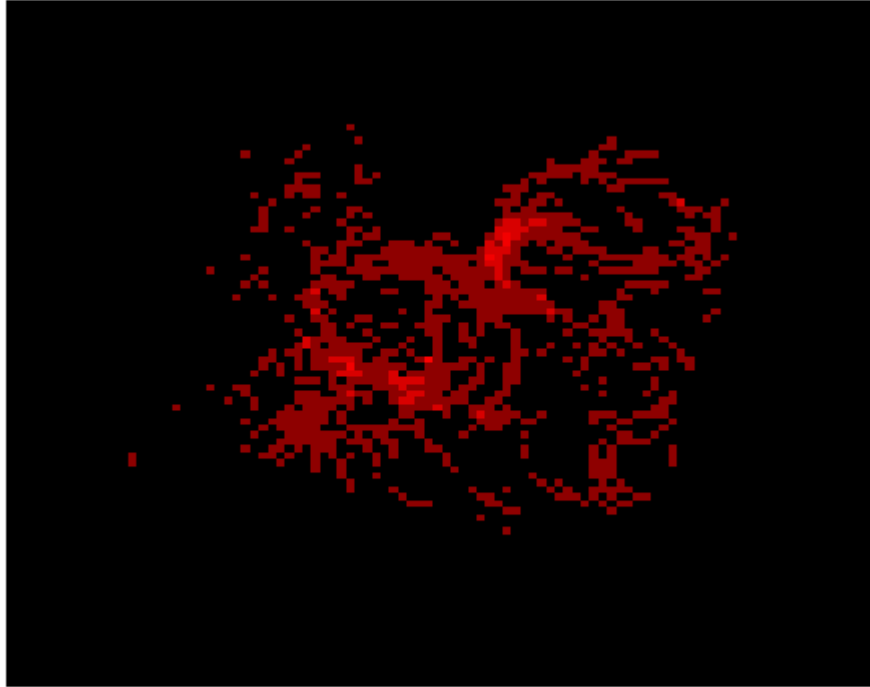


Figure 12: Grid size = 200×200

Obviously this is still not correct. Going through frame-by-frame, we can see the reason why:

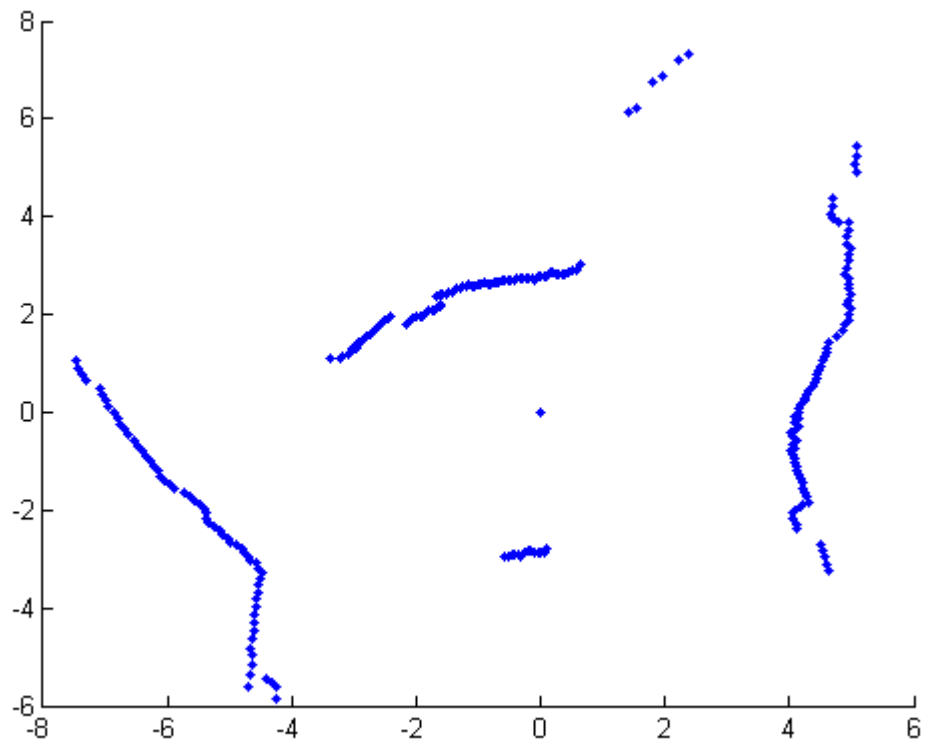


Figure 13:

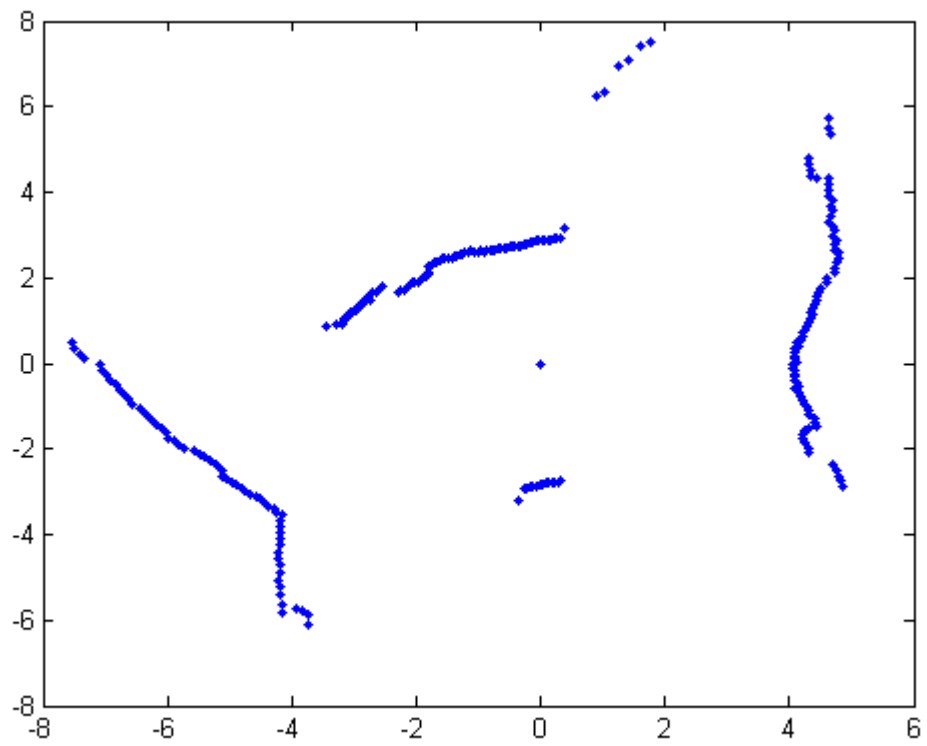


Figure 14:

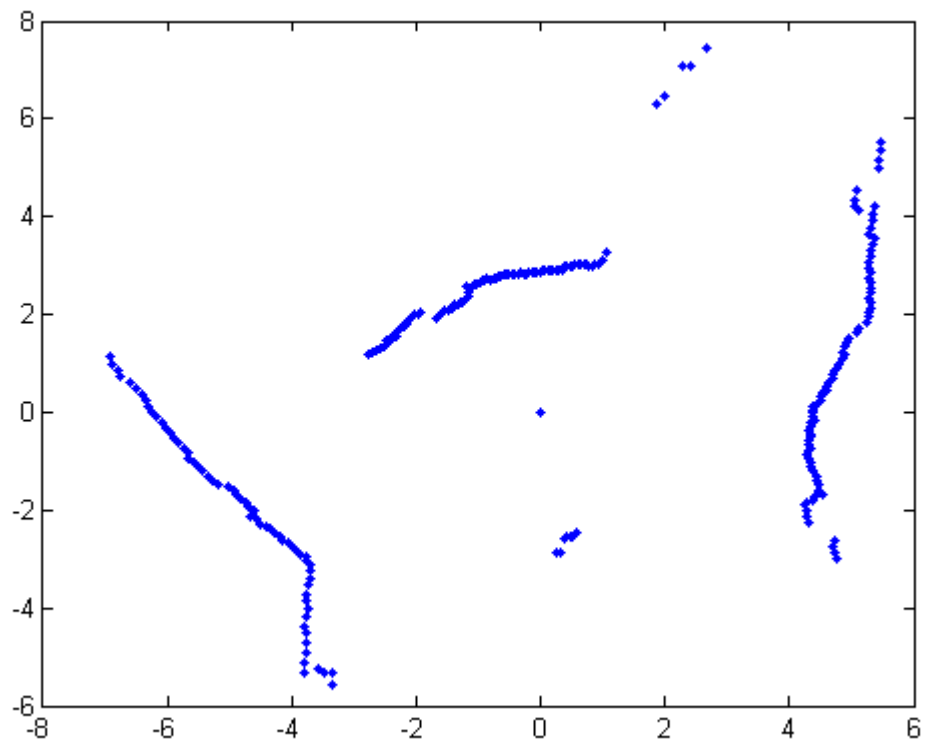


Figure 15:

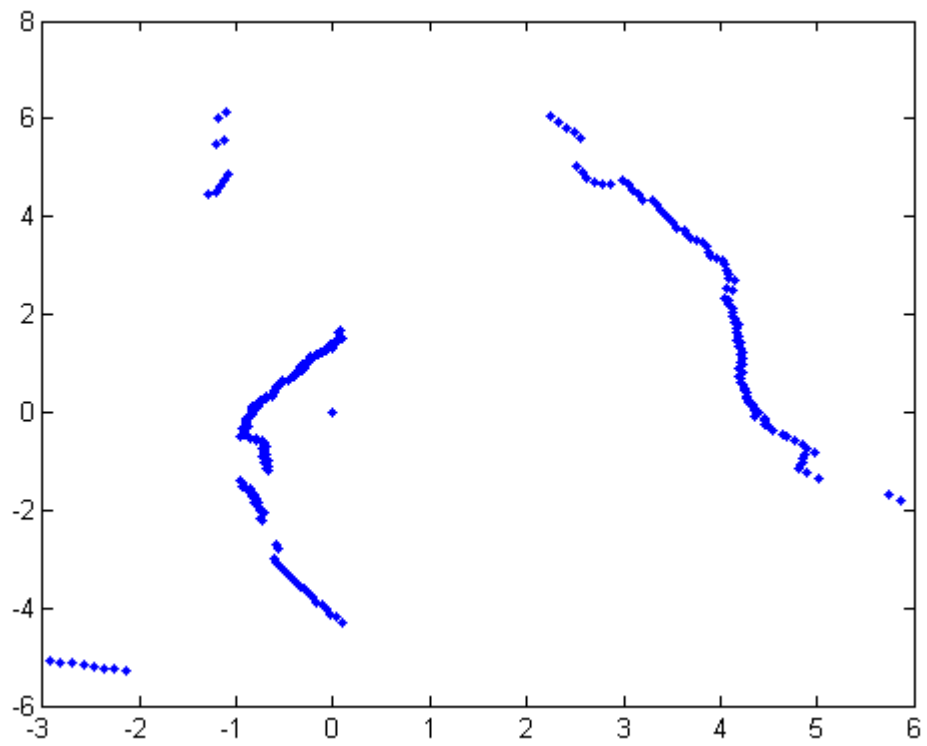


Figure 16:

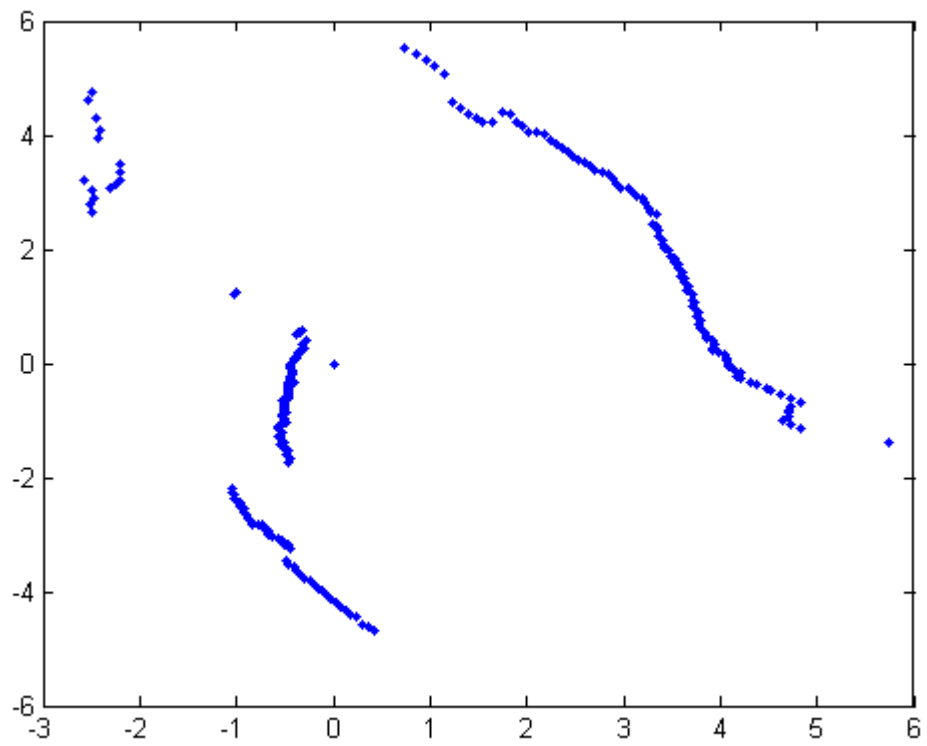


Figure 17:

The obstacles detected can be seen to be rotating. Obviously, once again, there is an issue with converting from the robots coordinate system to real-world coordinates.

The lines in the new code governing this are shown below:

```
1  for j = 4:2:size(laserObs,2)
2      range = laserObs(iters(2),j-1);
3      bearing = ((j)/2 - 90)*pi/180;
4      if (range < 8.0)
5          xpoint = [xpoint  ourX+range*cos(bearing + ourHeading)];
6          ypoint = [ypoint  ourY+range*sin(bearing + ourHeading)];
7      end
8
9  end
```

Upon closer analysis, it was determined that the flaw is that we are attempting to go straight from range and bearing to the real world coordinate system. Because of this, we are missing out on several crucial steps. This particularly effects the rotation. The method that should have been used is shown below:

```
1  %Calculate x and y coordinate from the laser bearing and range value,
2  %within the robots coordinate system.
3  x_robot = range*cos(bearing);
4  y_robot = range*sin(bearing);
5
6  %Rotate the coordinates by the robots current heading
7  %Places the coordinates partially in the world coordinate system
8  x_rotated = x_robot*cos(robot.heading) - y_robot*sin(robot.heading);
9  y_rotated = x_robot*sin(robot.heading) + y_robot*cos(robot.heading);
10
11 %Offset the rotated coordinates by the robots position
12 %Places the coordinates fully into the world coordinate system
13 x = x_rotated + robot.x_coordinate;
14 y = y_rotated + robot.y_coordinate;
```

The crucial part that was missing from our code is at lines 8 and 9 in the above code snippet. When rotating the x and y values from the robots frame of reference, we are not taking into account the sin offset for our x values, and the cos offset for our y values. This results in the final $x - y$ coordinates still being partially in the robots reference frame, giving rise to the observed rotation.