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"

$$iters = \begin{bmatrix} itersVel & itersGPS & itersComp & itersLasern \end{bmatrix}$$

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).

$$time = \begin{bmatrix} timestampVel & timestampGPS & timestampComp & timestampLaser \end{bmatrix}$$

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.

$$runFlags = \begin{bmatrix} runVel & runGPS & runComp & runLasern \end{bmatrix}$$

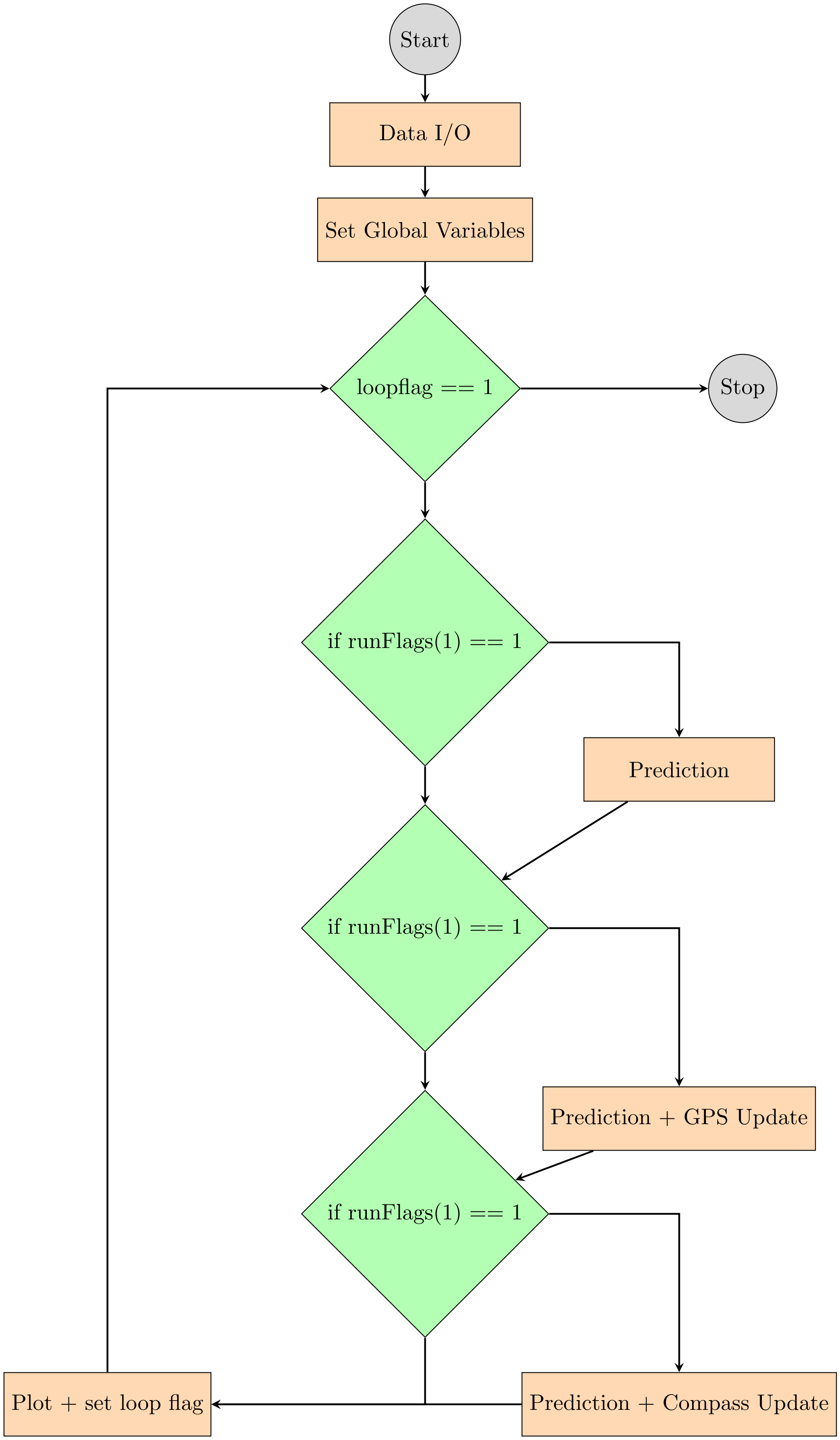

Figure 1: Overall Program Flow

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:

$$ObservationDataStructure = \begin{bmatrix} timestampVel(s) & timestampGPS & timestampComp & timestamp... & measurementn \end{bmatrix}$$

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 $t = 0$

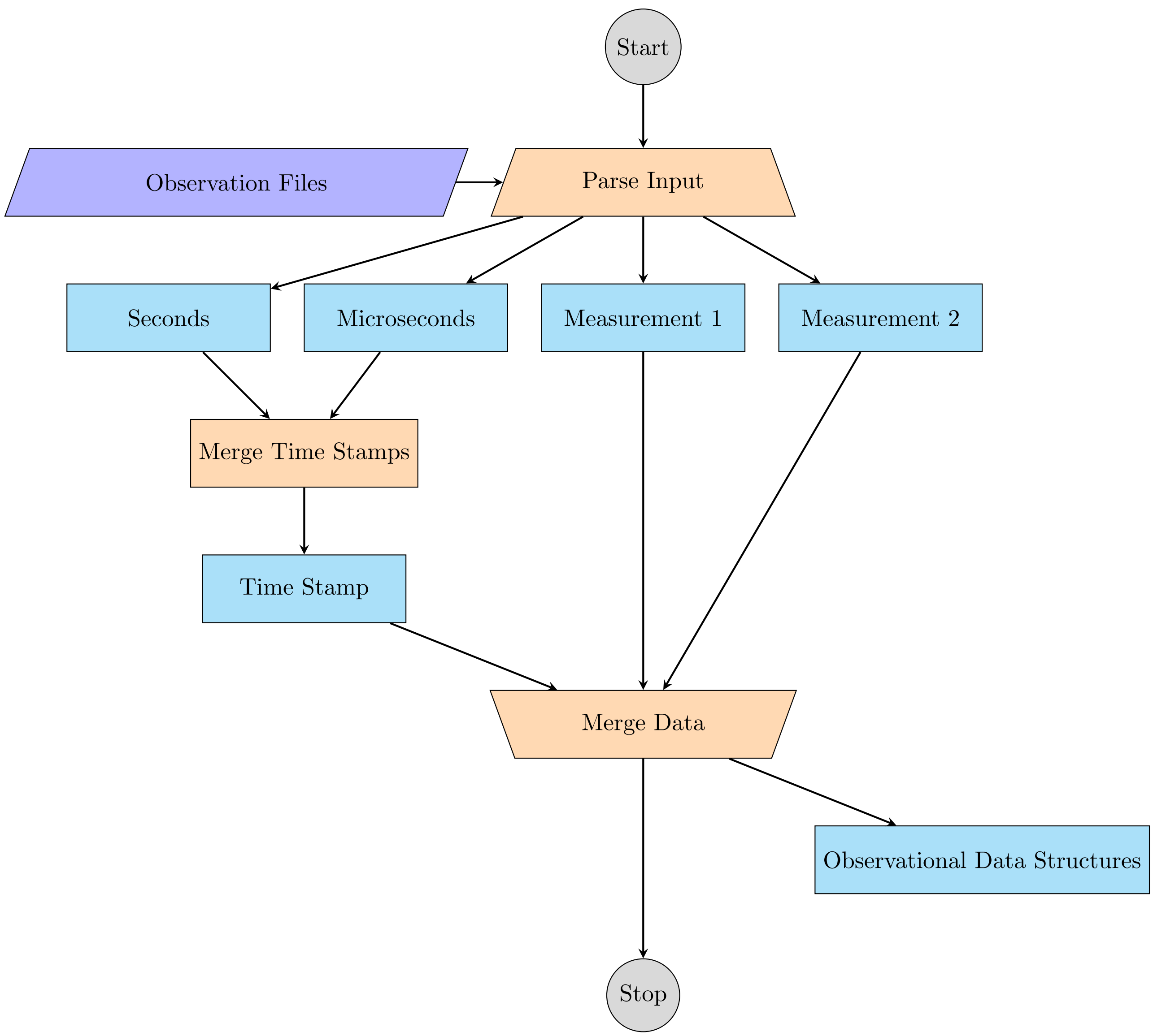

Figure 2: Data I/O Process

1.b Prediction Stage Implementation

```
function pr = predictionStage( XvPrev, YvPrev, THvPrev, deltaT, currTurnRate, currVel )
%Prediction Function given previous values
    XvPred = XvPrev + deltaT*currVel*cos(THvPrev);
    YvPred = YvPrev + deltaT*currVel*sin(THvPrev);
    THvPred = THvPrev + deltaT*currTurnRate;
    pr = [XvPred, YvPred, THvPred];
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

```
function gUpd = updateStageGPS(XvPred, YvPred, XvObs, YvObs, alphaP)

XvUpd = (1 - alphaP)*XvPred + alphaP*XvObs;
YvUpd = (1 - alphaP)*YvPred + alphaP*YvObs;
gUpd = [XvUpd, YvUpd];
end
```

1.c.ii Compass

```
function THvUpd = updateStageCompass(THvPred, THvObs, alphaTH)
a = abs(THvObs - THvPred);
while(a > pi)
    if(THvObs > THvPred)
        THvObs = THvObs - 2*pi;
%          THvPred = THvPred + 2*pi;
    else
        THvObs = THvObs + 2*pi;
%          THvPred = THvPred - 2*pi;
    end
    a = abs(THvObs - THvPred);
end

THvUpd = (1 - alphaTH)*THvPred + alphaTH*THvObs;
% THvUpd = THvPred - alphaTH*(THvObs - THvPred);

end
```

1.c.iii Laser

```
function mb = matchBeacons(ourX, ourY, ourHeading, lasFeat, thisrange, beaconCentreInds)
% Summary of this function goes here
%   Detailed explanation goes here
    beta1 = degtorad(beaconCentreInds(1)/2);
    beta2 = degtorad(beaconCentreInds(2)/2);

    if(beaconCentreInds(1) < pi/2)
        theta1 = pi/2 - beta1;
    else
        theta1 = beta1 - pi/2;
    end


    if(beaconCentreInds(1) < pi/2)
        theta2 = pi/2 - beta2;
    else
        theta2 = beta2 - pi/2;
    end

    Tx1 = ourX + thisrange(1,beaconCentreInds(2))*cos(theta1+ourHeading);
    Ty1 = ourY + thisrange(1,beaconCentreInds(2))*sin(theta1+ourHeading);
    T1 = [Tx1 Ty1];
    lf = lasFeat;
    distances = sqrt(sum(bsxfun(@minus, lf, T1).^2,2));
    T1 = lf(find(distances==min(distances)),:);


    Tx2 = ourX + thisrange(1,beaconCentreInds(3))*cos(theta2+ourHeading);
    Ty2 = ourY + thisrange(1,beaconCentreInds(3))*sin(theta2+ourHeading);
    T2 = [Tx2 Ty2];
    lf = lasFeat;
    distances = sqrt(sum(bsxfun(@minus, lf, T2).^2,2));
    T2 = lf(find(distances==min(distances)),:);

    Tx1 = T1(1);
    Ty1 = T1(2);
    Tx2 = T2(1);
    Ty2 = T2(2);

    ourHeading = atan((Ty2-Ty1)/Tx2-Tx1) - atan(thisrange(beaconCentreInds(3))*sin(theta2+ourHeading)-thisrange(beaconCentreInds(2))*sin(theta1+ourHeading))/(thisrange(beaconCentreInds(3))*cos(theta2+ourHeading)-thisrange(beaconCentreInds(2))*cos(theta1+ourHeading));
    ourX = Tx1 - thisrange(beaconCentreInds(2))*cos(theta1+ourHeading);
    ourY = Ty1 - thisrange(beaconCentreInds(2))*sin(theta1+ourHeading);

    mb = [ourX, ourY, ourHeading];
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.

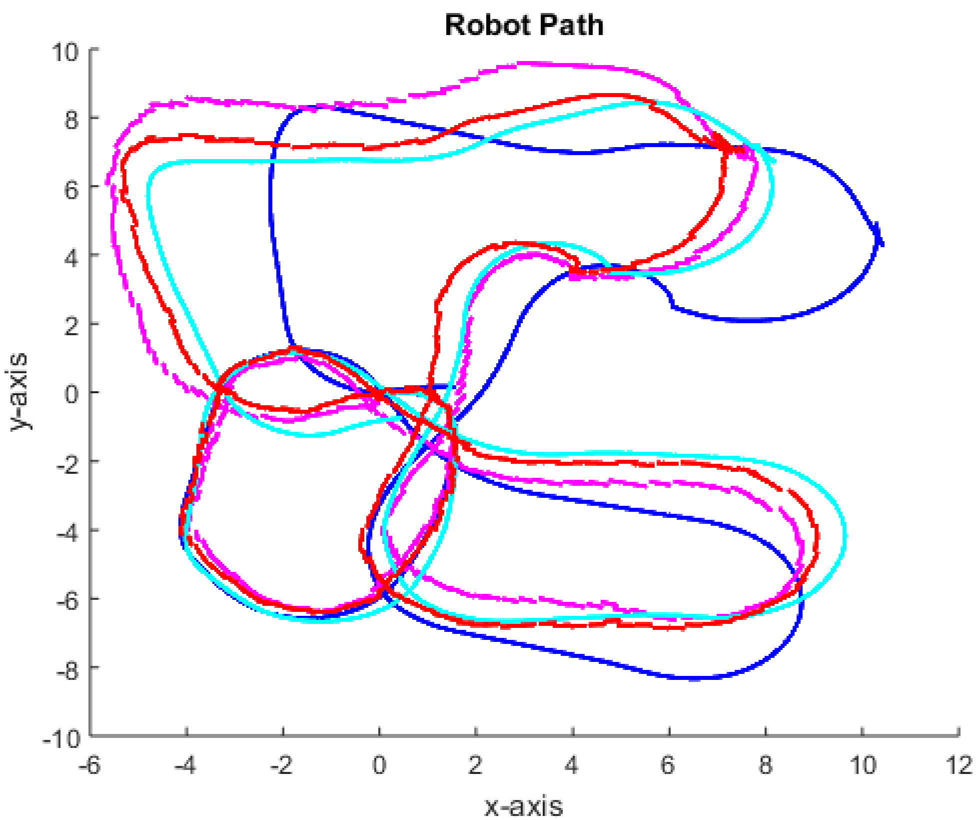

Figure 3: Robot Paths

1) Blue: Dead Reckoning (alphaP = alphaTH = 0)
2) Magenta: GPS data observed (alphaP = 0.1)
3) Cyan: Compass data observed (alphaP = 0.1)
4) Red: Compass + GPS data observed (alphaP = alphaTH = 0.1)

1.d.i alpha = 0.5

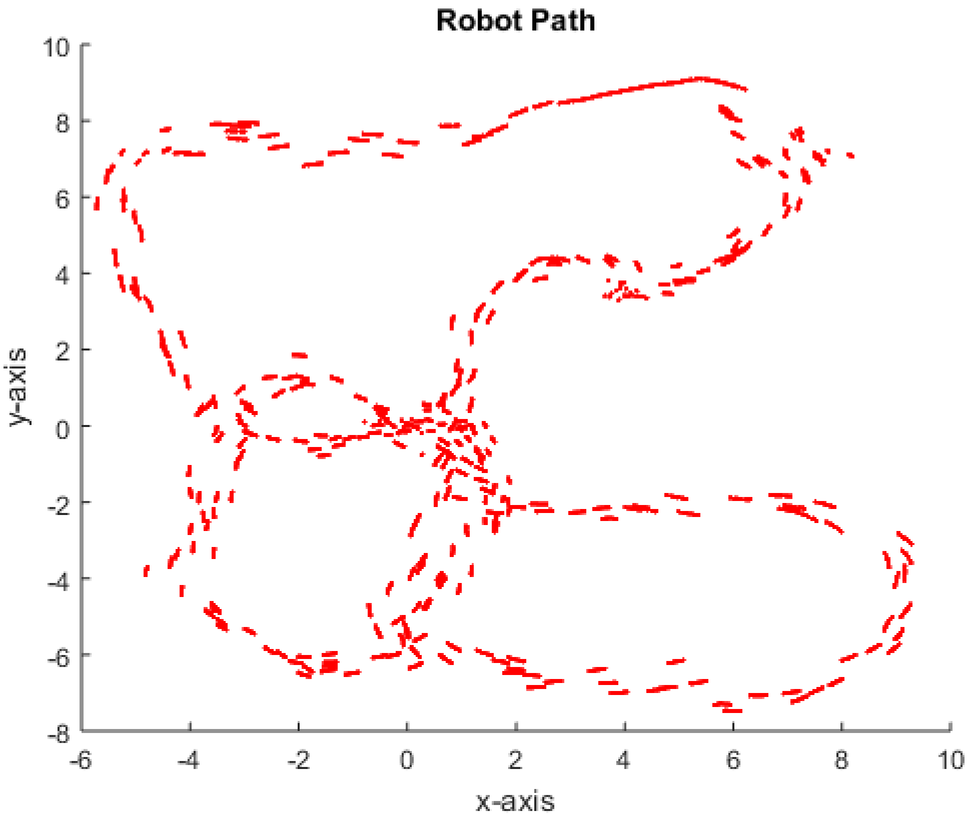

Figure 4: All Data (alpha = 0.5)

1.d.ii alpha = 0.9

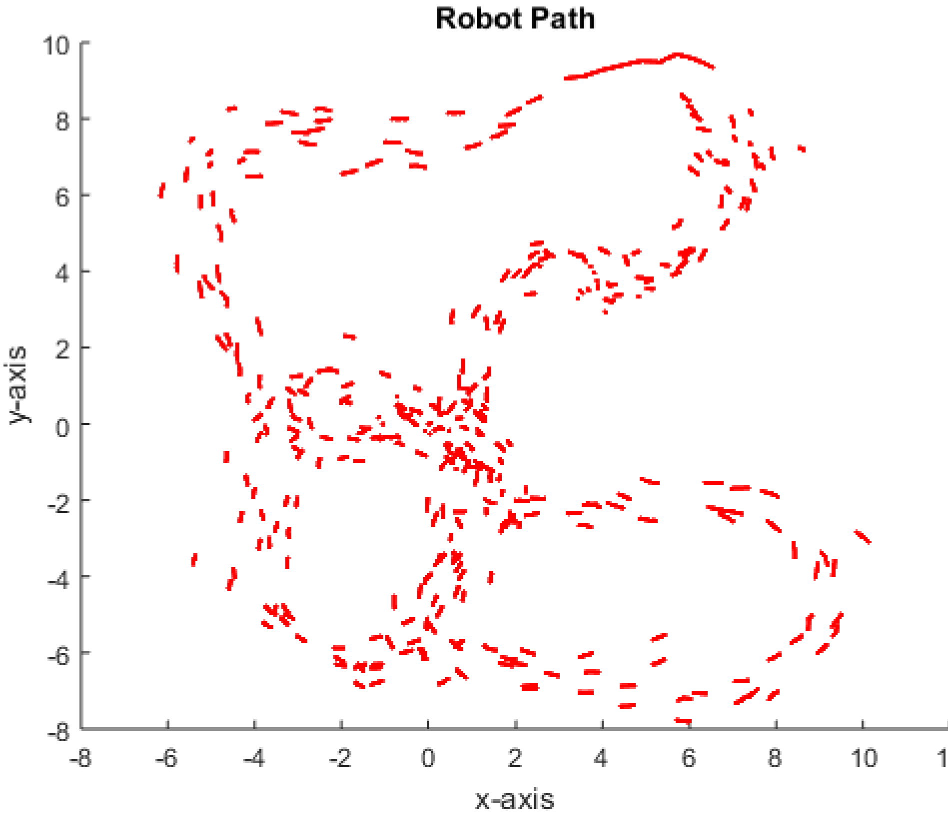

Figure 5: All Data (alpha = 0.9)

1.d.iii alpha = 1

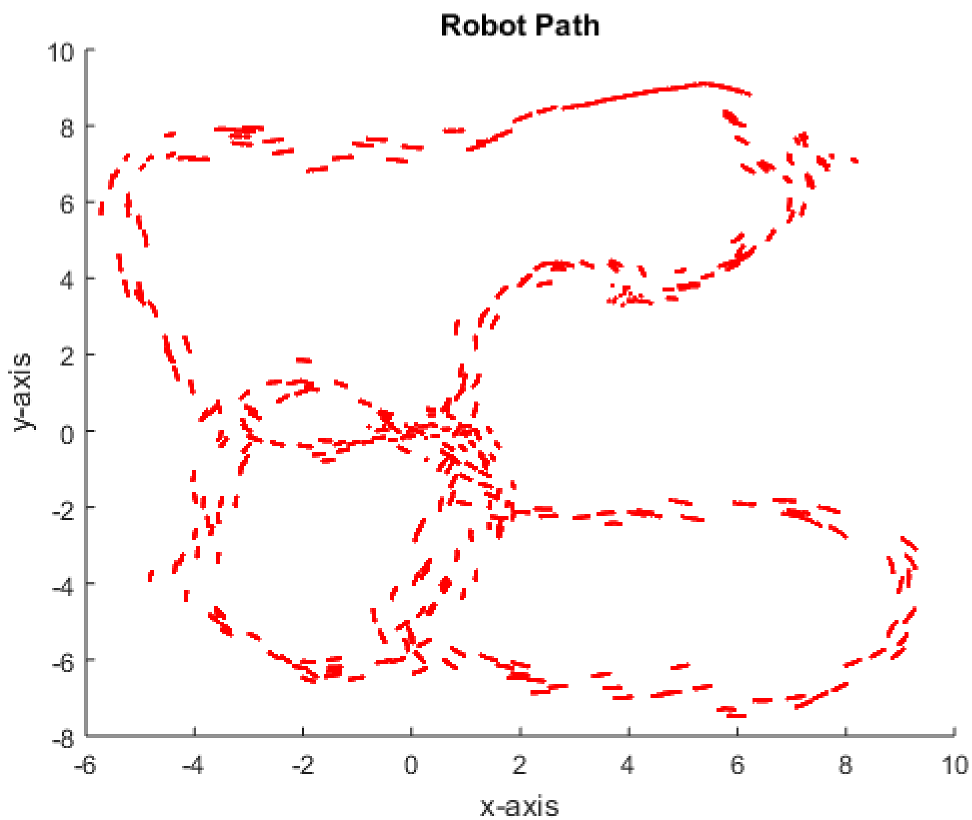

Figure 6: All Data (alpha = 1)

1.e Code Listing

1.e.i Demonstration Code

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

```
clear
% close all
clc

DEGREES = 180/pi;
RADIANS = pi/180;
SUN = 1.496*10^8;
% % Prediction Stage
% diary './q1output'
%Load observational data
velocityObs = load('velocityObs.txt');
positionObs = load('positionObs.txt');
compassObs = load('compassObs.txt');
laserObs = load('laserObs.txt');

%Get velocity data
time1 = velocityObs(:,1) + (velocityObs(:,2)*10^-6) - 1115116000;%get in microseconds
velocity = velocityObs(:,3);
turnRate = velocityObs(:,4);

velObs = [time1 velocity turnRate];

%Get GPS position data
time2 = positionObs(:,1) + (positionObs(:,2)*10^-6) - 1115116000;
xPos = positionObs(:,3);
yPos = positionObs(:,4);

posObs = [time2 xPos yPos];

%Get GPS compass data
time3 = compassObs(:,1) + (compassObs(:,2)*10^-6) - 1115116000;
heading = compassObs(:,3);

compObs = [time3 heading];

% % %get laser data
time4 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;
% % i = 1;
% % j = 4;
% % sizeLas = size(laserObs);
% % lasObs = zeros(sizeLas(1, 2));
% % while(i <= sizeLas(1))
% %     lasObs(i, 1) = [time4(i)];
% %     while(j<=sizeLas(2))
% %         lasObs(i, j) = laserObs(i, j);
% %         lasObs(i, j-1) = laserObs(i, j-1);
% %
% %         j = j + 2;
% %     end
% %
% %     i = i + 1;
% % end
% % size(time4)



%%postInput parsing

alphaP = 0.1;
alphaTH = 0.1;

lastTime = 0;
deltaT = 0;

```

```
latestVel = 0;
latestTurnRate = 0;

ourX = 0;
ourY = 0;
ourHeading = 0;

indLengths = [length(time1), length(time2), length(time3), length(time4)];
maxIters = max(indLengths);

% output = [lastTime, ourX, ourY, ourHeading];
output = zeros(maxIters, 6);

% deadreckonedpts = zeros(maxIters, 3);

%iters [velInd, posInd, compInd, lasInd];
iters = [2, 2, 2, 2];
runFlags = [0, 0, 0, 0];
loopFlag = 1;
loopCount = 2;
%%loop starts
while(loopFlag == 1)
    loopCount = loopCount + 1
    time = [time1(iters(1)), time2(iters(2)), time3(iters(3)), time4(iters(4)),];
    nextT = min(time);


    for i = 1:4
        if time(i) == nextT
            runFlags(i) = 1;
        else
            runFlags(i) = 0;
        end
    end

    %if velocityobs
    if(runFlags(1) == 1)
        deltaT = time1(iters(1)) - lastTime;
        latestVel = velObs(iters(1),2);%
        latestTurnRate = velObs(iters(1),3);%
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = pr(3);

%         deadreckonedpts(1) = ourX;
%         deadreckonedpts(2) = ourY;
%         deadreckonedpts(3) = ourHeading;
        lastTime = time1(iters(1));%
        runFlags(1) = 0;
        if iters(1) == length(time1)
            time1(iters(1)) = SUN;
        else
            iters(1) = iters(1) + 1;
        end
    end

% % if GPS
     if(runFlags(2) == 1)
        deltaT = time2(iters(2)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        gUpd = updateStageGPS(pr(1), pr(2), posObs(iters(2),2), posObs(iters(2),3), alphaP); %xvobs and yvobs need to come from the file
        ourX = gUpd(1);
        ourY = gUpd(2);
        ourHeading = pr(3);
        lastTime = time2(iters(2));%
        runFlags(2) = 0;
        if iters(2) == length(time2)
            time2(iters(2)) = SUN;
        else
```

```
            iters(2) = iters(2) + 1;
        end
     end
%
% % if compass
     if(runFlags(3) == 1)
        deltaT = time3(iters(3)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        cUpd = updateStageCompass(pr(3), compObs(iters(3),2), alphaTH);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = cUpd;
        lastTime = time3(iters(3));
        runFlags(3) = 0;
        if iters(3) == length(time3)
            time3(iters(3)) = SUN;
        else
            iters(3) = iters(3) + 1;
        end
    end
%
% % if laser data
    if(runFlags(4) == 1)
% %       fill this in
        runFlags(4) = 0;
        time4(iters(4)) = SUN;   %remove this line
    end

%check loop
    if(time1(iters(1)) == SUN)
        if(time2(iters(2)) == SUN)
            if(time3(iters(3)) == SUN)
                if(time4(iters(4)) == SUN)
                     loopFlag = 0;
                end
            end
        end
    end

%plot stuff
hold on
title('Robot Path');
xlabel('x-axis');
ylabel('y-axis');
% legend('Dead Reckoning');
% drawnow

plot(ourX, ourY, 'r.');
output(loopCount, 1) = lastTime;
output(loopCount, 2) = ourX;
output(loopCount, 3) = ourY;
output(loopCount, 4) = ourHeading;
output(loopCount, 5) = latestVel;
output(loopCount, 6) = latestTurnRate;
end

% diary ON
% output
% diary OFF

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.

```
clear
close all
clc

DEGREES = 180/pi;
RADIANS = pi/180;
SUN = 1.496*10^8;

%Load observational data
velocityObs = load('velocityObs.txt');
positionObs = load('positionObs.txt');
compassObs = load('compassObs.txt');
laserObs = load('laserObs.txt');
laserFeat = load('laserFeatures.txt');

%get laser features
lasFeat = [laserFeat(:,1) laserFeat(:,2)];

%Get velocity data
time1 = velocityObs(:,1) + (velocityObs(:,2)*10^-6) - 1115116000;%get in microseconds
velocity = velocityObs(:,3);
turnRate = velocityObs(:,4);

velObs = [time1 velocity turnRate];

%Get GPS position data
time2 = positionObs(:,1) + (positionObs(:,2)*10^-6) - 1115116000;
xPos = positionObs(:,3);
yPos = positionObs(:,4);

posObs = [time2 xPos yPos];

%Get GPS compass data
time3 = compassObs(:,1) + (compassObs(:,2)*10^-6) - 1115116000;
heading = compassObs(:,3);

compObs = [time3 heading];

% % %get laser data
time4 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;

f1=1;

range = zeros(length(laserObs), (size(laserObs,2)-2)/2);
intensity = zeros(length(laserObs), (size(laserObs,2)-2)/2);

%Extracting range & intensity data from LaserObs
for i=1:length(laserObs)
   for f2=3:2:size(laserObs,2)
    range(i,f1)=laserObs(i,f2);
    intensity(i,f1)=laserObs(i,f2+1);
    f1=f1+1;
   end
   f1=1;
end


% postInput parsing

alphaP = 0.1;
alphaTH = 0.1;

lastTime = 0;
deltaT = 0;
```

```
latestVel = 0;
latestTurnRate = 0;

ourX = 0;
ourY = 0;
ourHeading = 0;

indLengths = [length(time1), length(time2), length(time3), length(time4)];
maxIters = max(indLengths);

% output = [lastTime, ourX, ourY, ourHeading];
output = zeros(maxIters, 6);

% deadreckonedpts = zeros(maxIters, 3);

%iters [velInd, posInd, compInd, lasInd];
iters = [2, 2, 2, 2];
runFlags = [0, 0, 0, 0];
loopFlag = 1;
loopCount = 2;
%%loop starts
while(loopFlag == 1)
    loopCount = loopCount + 1
    time = [time1(iters(1)), time2(iters(2)), time3(iters(3)), time4(iters(4)),];
    nextT = min(time);


    for i = 1:4
        if time(i) == nextT
            runFlags(i) = 1;
        else
            runFlags(i) = 0;
        end
    end

    %if velocityobs
    if(runFlags(1) == 1)
        deltaT = time1(iters(1)) - lastTime;
        latestVel = velObs(iters(1),2);%
        latestTurnRate = velObs(iters(1),3);%
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = pr(3);

%          deadreckonedpts(1) = ourX;
%          deadreckonedpts(2) = ourY;
%          deadreckonedpts(3) = ourHeading;
        lastTime = time1(iters(1));%
        runFlags(1) = 0;
        if iters(1) == length(time1)
            time1(iters(1)) = SUN;
        else
            iters(1) = iters(1) + 1;
        end
    end

% % if GPS
     if(runFlags(2) == 1)
        deltaT = time2(iters(2)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        gUpd = updateStageGPS(pr(1), pr(2), posObs(iters(2),2), posObs(iters(2),3), alphaP); %xvobs and yvobs need to come from the file
        ourX = gUpd(1);
        ourY = gUpd(2);
        ourHeading = pr(3);
        lastTime = time2(iters(2));%
        runFlags(2) = 0;
        if iters(2) == length(time2)
            time2(iters(2)) = SUN;
```

```
        else
            iters(2) = iters(2) + 1;
        end
     end
%
% % if compass
     if(runFlags(3) == 1)
        deltaT = time3(iters(3)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        cUpd = updateStageCompass(pr(3), compObs(iters(3),2), alphaTH);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = cUpd;
        lastTime = time3(iters(3));
        runFlags(3) = 0;
        if iters(3) == length(time3)
            time3(iters(3)) = SUN;
        else
            iters(3) = iters(3) + 1;
        end
    end
%
% % if laser data
    if(runFlags(4) == 1)
%         find beacons
        deltaT = time4(iters(4)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = pr(3);

        intensityTrav = 1;
        beaconCentreInds = 0;
        %find beacon centres
        while(intensityTrav <= 361)
            if(intensity(intensityTrav) == 1)
                iStart = intensityTrav;
                intensityTrav = intensityTrav + 1;
                while(intensity(intensityTrav) == 1 && (abs(range(intensityTrav) - range(
                    ↪ intensityTrav + 1) < 1)))
                    intensityTrav = intensityTrav + 1;
                end
                iEnd = intensityTrav - 1;
                iMid = (iEnd - iStart)/2;
                beaconCentreInds = cat(2, beaconCentreInds, iMid);
            else
                intensityTrav = intensityTrav + 1;
            end
        end
%       project to real world
        mb = matchBeacons(ourX, ourY, ourHeading, lasFeat, range(iters(4),:), beaconCentreInds);
        ourX = mb(1);
        ourY = mb(2);
        ourHeading = mb(3);

        lastTime = time4(iters(4));
        runFlags(4) = 0;
        if iters(4) == length(time4)
            time3(iters(4)) = SUN;
        else
            iters(4) = iters(4) + 1;
        end
    end

%check loop
    if(time1(iters(1)) == SUN)
        if(time2(iters(2)) == SUN)
            if(time3(iters(3)) == SUN)
                if(time4(iters(4)) == SUN)
                     loopFlag = 0;
                end
```

```
            end
        end
    end

%plot stuff
hold on
title('Robot Path');
xlabel('x-axis');
ylabel('y-axis');
% legend('')
% drawnow

plot(ourX, ourY, 'r.');
output(loopCount, 1) = lastTime;
output(loopCount, 2) = ourX;
output(loopCount, 3) = ourY;
output(loopCount, 4) = ourHeading;
output(loopCount, 5) = latestVel;
output(loopCount, 6) = latestTurnRate;
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:

```
clear
clc
close all

DEGREES = 180/pi;
RADIANS = pi/180;

%Load data generated from Q1
positionData = load('q1output1.txt');

%Load laser observation data
laserObs = load('laserObs.txt');

%Get output data
time1 = positionData(:,1);
Xpos = positionData(:,2);
Ypos = positionData(:,3);
heading = positionData(:,4);
velocity = positionData(:,5);
turnRate = positionData(:,6);

%Setup output recording
diary './q2Output4'

%Get laser data
time2 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;%get in microseconds


%Extracting range from LaserObs

f1=1;
range = zeros(length(laserObs), (size(laserObs,2)-2)/2);

%Extracting range & intensity data from LaserObs
for i=1:length(laserObs)
   for f2=3:2:size(laserObs,2)
            range(i,f1)=laserObs(i,f2);
            f1=f1+1;
   end
   f1=1;
end

lasersX = 0;
lasersY = 0;

% alphaP = 0.5;
% alphaTH = 0.5;

lastTime = 0;
deltaT = 0;

latestVel = 0;
latestTurnRate = 0;

ourX = 0;
ourY = 0;
ourHeading = 0;

indLengths = [length(time1), length(time2)];
maxIters = max(indLengths);

interval = 20;

%iters [velInd, posInd, compInd, lasInd];iters = [2, 2];
runFlags = [0, 0];
loopFlag = 1;
loopCount = 2;

%%loop starts

```

```

while(loopFlag == 1)
    loopCount = loopCount + 1;
    time = [time1(iters(1)), time2(iters(2))];
    nextT = min(time);


    for i = 1:2
        if time(i) == nextT
            runFlags(i) = 1;
        else
            runFlags(i) = 0;
        end
    end

    %if Positiondata
    if(runFlags(1) == 1)
        deltaT = time1(iters(1)) - lastTime;
        latestVel = velocity(iters(1));
        latestTurnRate = turnRate(iters(1));
        ourX = Xpos(iters(1));
        ourY = Ypos(iters(1));
        ourHeading = heading(iters(1));

        lastTime = time1(iters(1));%
        runFlags(1) = 0;
        if iters(1) >= length(time1)-interval
            time1(iters(1)) = 1.496*10^8;
        else
            iters(1) = iters(1) + interval;
        end
    end

% % if Laser Observation Data
     if(runFlags(2) == 1)
        deltaT = time2(iters(2)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = pr(3);
        lastTime = time2(iters(2));%
        runFlags(2) = 0;


        % Get X,Y coordinates for laser ob data

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

        %increment
        if iters(2) >= length(time2)-interval
            time2(iters(2)) = 1.496*10^8;
        else
            iters(2) = iters(2) + interval;
        end


     end

%check loop
    if(time1(iters(1)) == 1.496*10^8)
        if(time2(iters(2)) == 1.496*10^8)
                     loopFlag = 0;
        end
```

```
    end

%plot stuff
% hold on
% title('Obstacles');
% xlabel('x-axis');
% ylabel('y-axis');
% legend('')
% drawnow
end
```

2.b Generate Occupancy Grid

To generate the Occupance 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 deteced. 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 $200x200$ 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:

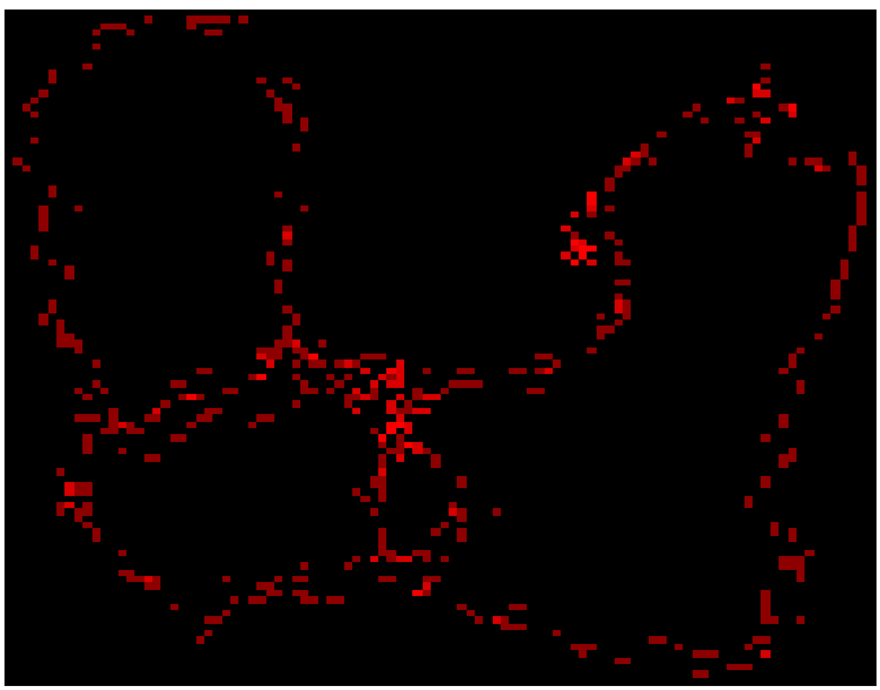

Figure 7: Grid size = 100×100

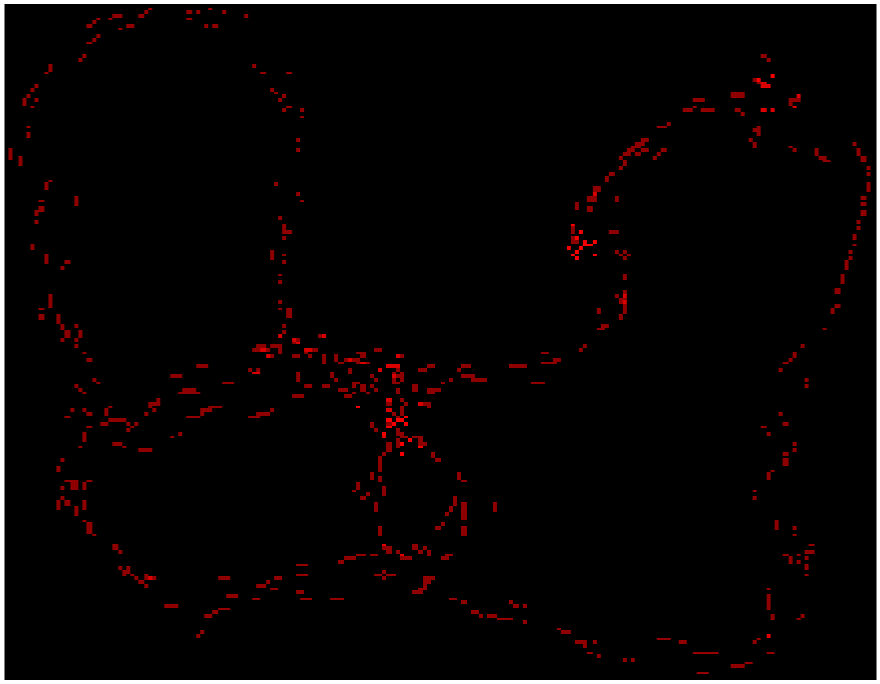

Figure 8: Grid size = 200×200

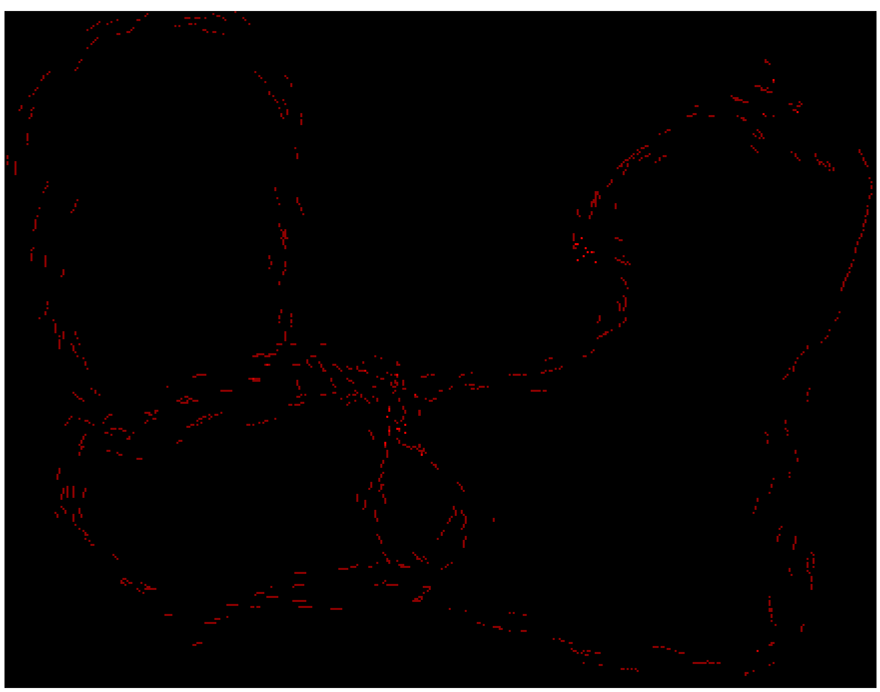

Figure 9: Grid size = 500×500

The matlab code for the occupancy grid is shown below:

```
clear
close all
clc

% load output from the obtain_obstacles code
positionData = load('laserPosiitons.txt');

xPos = positionData(:,1);
yPos = positionData(:,2);

%Get maximum and minimum x-y values
xMin = min(xPos);
xMax = max(xPos);

yMin = min(yPos);
yMax = max(yPos);

%define a square grid size
gridSize = 100;

grid = zeros(gridSize);

%Determine the difference between each grid location
yDiff = (yMax - yMin)/(gridSize - 2);
xDiff = (xMax - xMin)/(gridSize - 2);

%Determine which grid loaction each coordinate corresponds to.
for i = 1:length(xPos)
    tmpX = xPos(i);
    tmpY = yPos(i);
    j = 1;
    while (tmpX > xMin)
        tmpX = tmpX - xDiff;
        j = j + 1;
    end
    k = 1;
    while (tmpY > yMin)
        tmpY = tmpY - yDiff;
        k = k + 1;
    end

    grid(j,k) = grid(j,k) + 1;
end

%Generate occupancy grid
HeatMap(grid);
```

2.c Results

The main problem with generating the obstacle data is that the program used took far to 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 a 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:

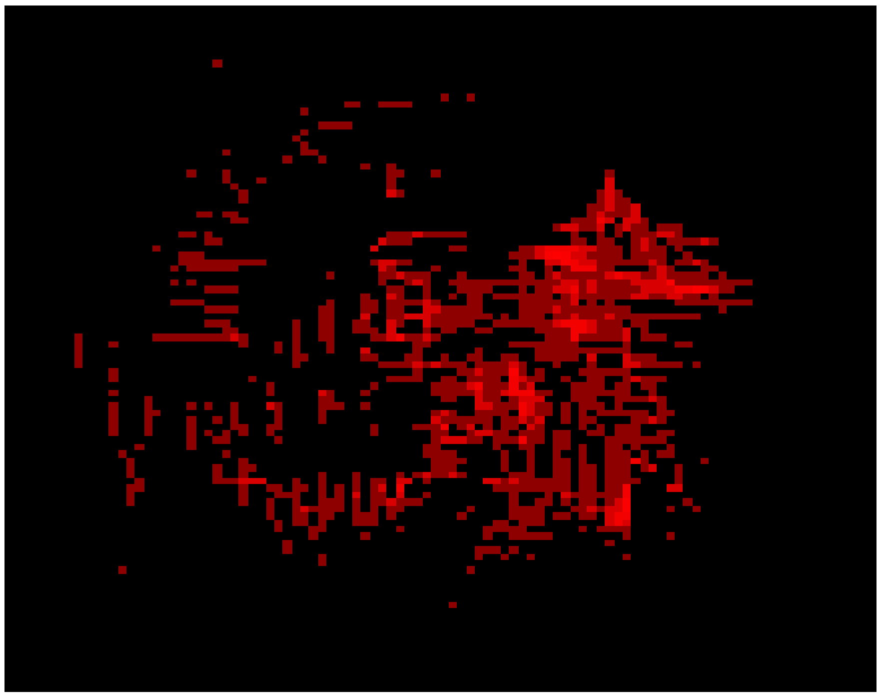

Figure 10: Grid size = 200×200

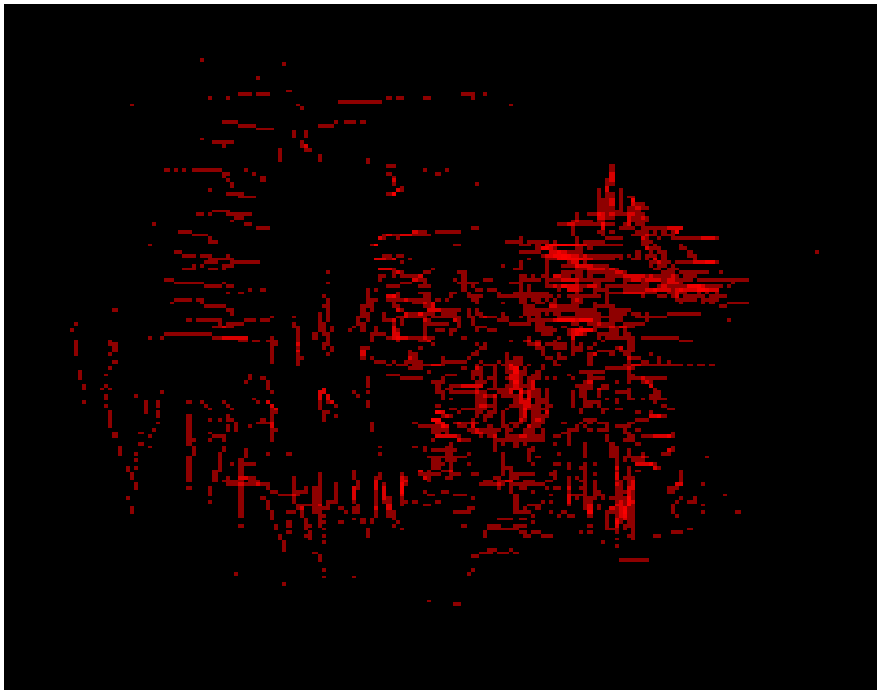

Figure 11: Grid size = 500 × 500

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:

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

```
clear
clc
close all

DEGREES = 180/pi;
RADIANS = pi/180;


%Load data generated from Q1
positionData = load('q1output1.txt');

%Load laser observation data
laserObs = load('laserObs.txt');

%Get output data
time1 = positionData(:,1);
Xpos = positionData(:,2);
Ypos = positionData(:,3);
heading = positionData(:,4);
velocity = positionData(:,5);
turnRate = positionData(:,6);

%Get laser data
time2 = laserObs(:,1) + (laserObs(:,2)*10^-6) - 1115116000;%get in microseconds

lasersX = 0;
lasersY = 0;

% alphaP = 0.5;
% alphaTH = 0.5;

lastTime = 0;
deltaT = 0;

latestVel = 0;
latestTurnRate = 0;

ourX = 0;
ourY = 0;
ourHeading = 0;

indLengths = [length(time1), length(time2)];
maxIters = max(indLengths);

interval = 20;

iters = [2, 2];
runFlags = [0, 0];
loopFlag = 1;
loopCount = 2;

xPos = zeros(1);
yPos = zeros(1);

%%loop starts


while(loopFlag == 1)
    loopCount = loopCount + 1;
    time = [time1(iters(1)), time2(iters(2))];
    nextT = min(time);


    for i = 1:2
```

```
        if time(i) == nextT
            runFlags(i) = 1;
        else
            runFlags(i) = 0;
        end
    end

    %if Positiondata
    if(runFlags(1) == 1)
        deltaT = time1(iters(1)) - lastTime;
        latestVel = velocity(iters(1));
        latestTurnRate = turnRate(iters(1));
        ourX = Xpos(iters(1));
        ourY = Ypos(iters(1));
        ourHeading = heading(iters(1));

        lastTime = time1(iters(1));%
        runFlags(1) = 0;
        if iters(1) >= length(time1)-interval
            time1(iters(1)) = 1.496*10^8;
        else
            iters(1) = iters(1) + interval;
        end
    end

% % if laser observation data
     if(runFlags(2) == 1)
        deltaT = time2(iters(2)) - lastTime;
        pr = predictionStage(ourX, ourY, ourHeading, deltaT, latestTurnRate, latestVel);
        ourX = pr(1);
        ourY = pr(2);
        ourHeading = pr(3);
        lastTime = time2(iters(2));%
        runFlags(2) = 0;

        %Begin here modified code extract from laserShowAcfr.m from Assignment 2
         xpoint = zeros(1);
         ypoint = zeros(1);
         for j = 4:2:size(laserObs,2)
             range = laserObs(iters(2),j-1);
             bearing = ((j)/2 - 90)*pi/180;
             if (range < 8.0)
                 xpoint = [xpoint  ourX+range*cos(bearing + ourHeading)];
                 ypoint = [ypoint  ourY+range*sin(bearing + ourHeading)];
             end

         end

         %End extract

         xPos = [xPos xpoint];
         yPos = [yPos ypoint];

        plot(xpoint(:), ypoint(:), '.');

        %increment
        if iters(2) >= length(time2)-interval
            time2(iters(2)) = 1.496*10^8;
        else
            iters(2) = iters(2) + interval;
        end


     end

%check loop
    if(time1(iters(1)) == 1.496*10^8)
        if(time2(iters(2)) == 1.496*10^8)
                    loopFlag = 0;
        end
    end
```

```
%plot stuff
% hold on
% title('Obstacles');
% xlabel('x-axis');
% ylabel('y-axis');
% legend('')
drawnow
% pause
end

xpoint(1) = [];
ypoint(1) = [];

xPos(1) = [];
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, severly 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:

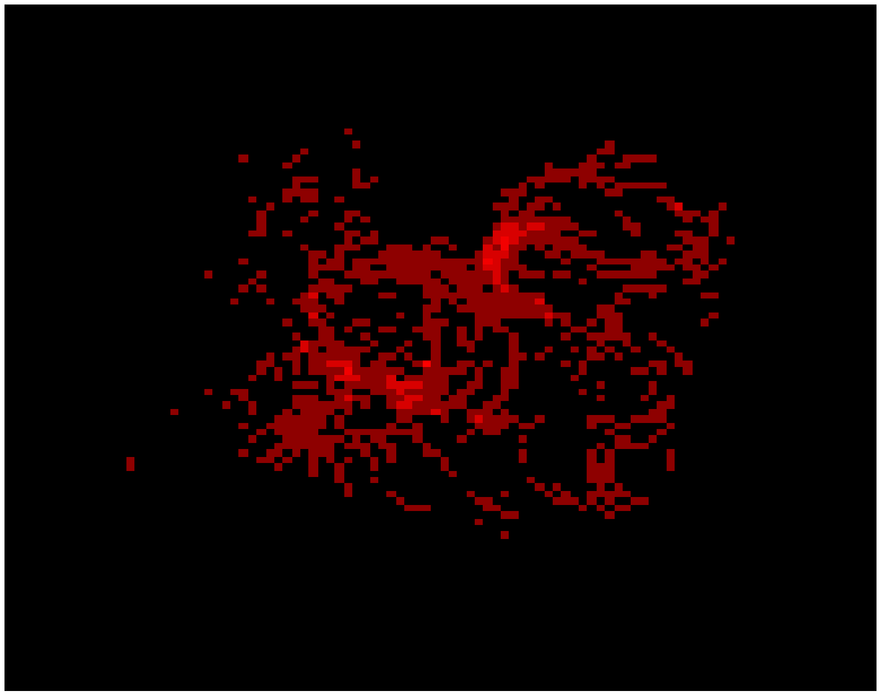

Figure 12: Grid size = 200×200

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

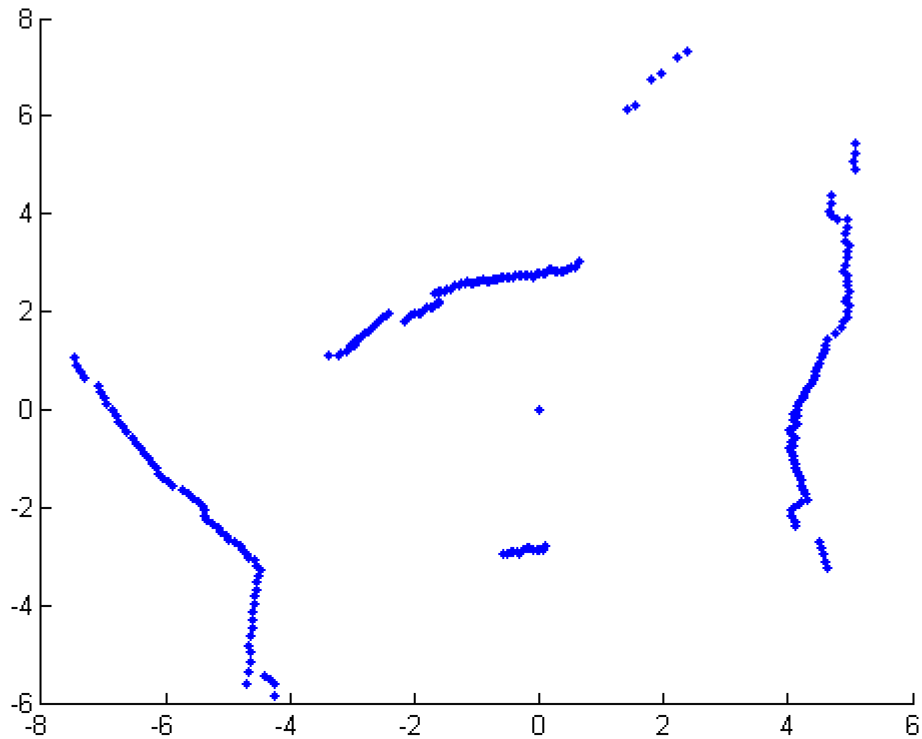

Figure 13:

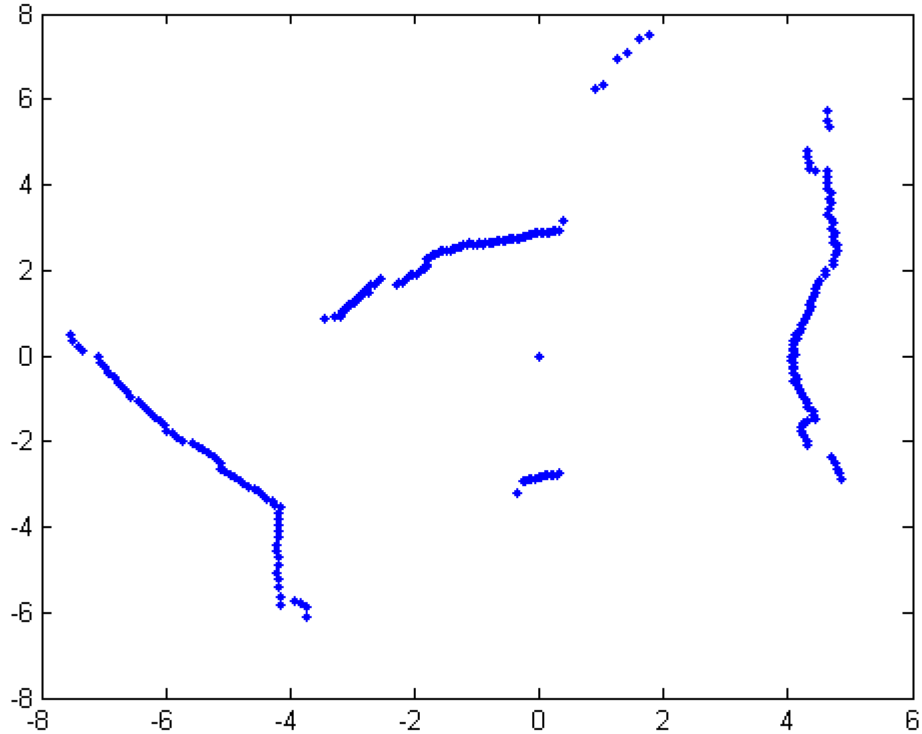

Figure 14:

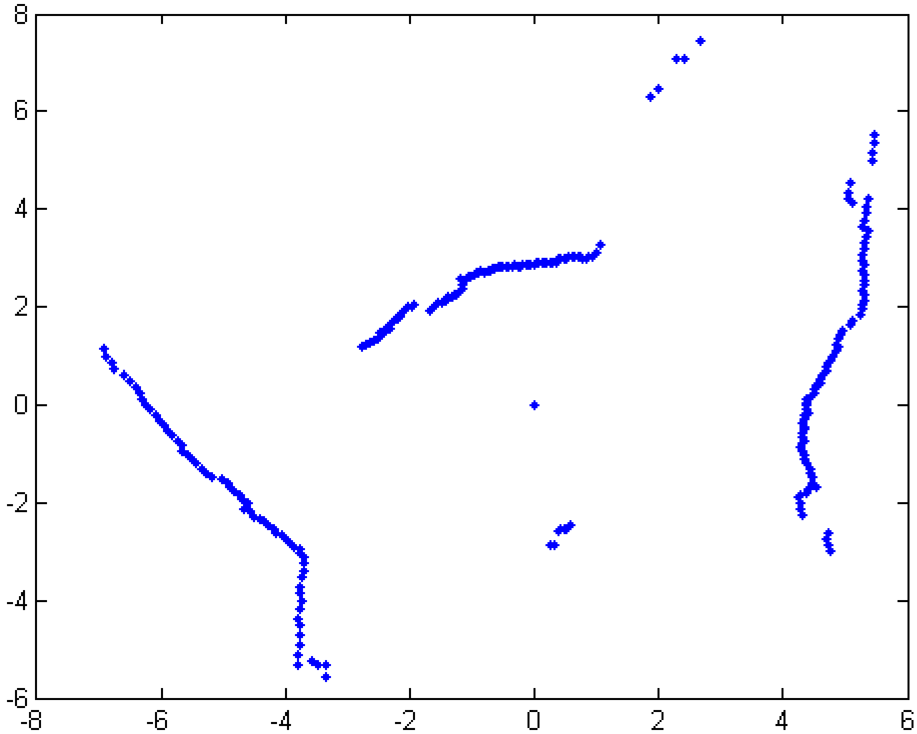

Figure 15:

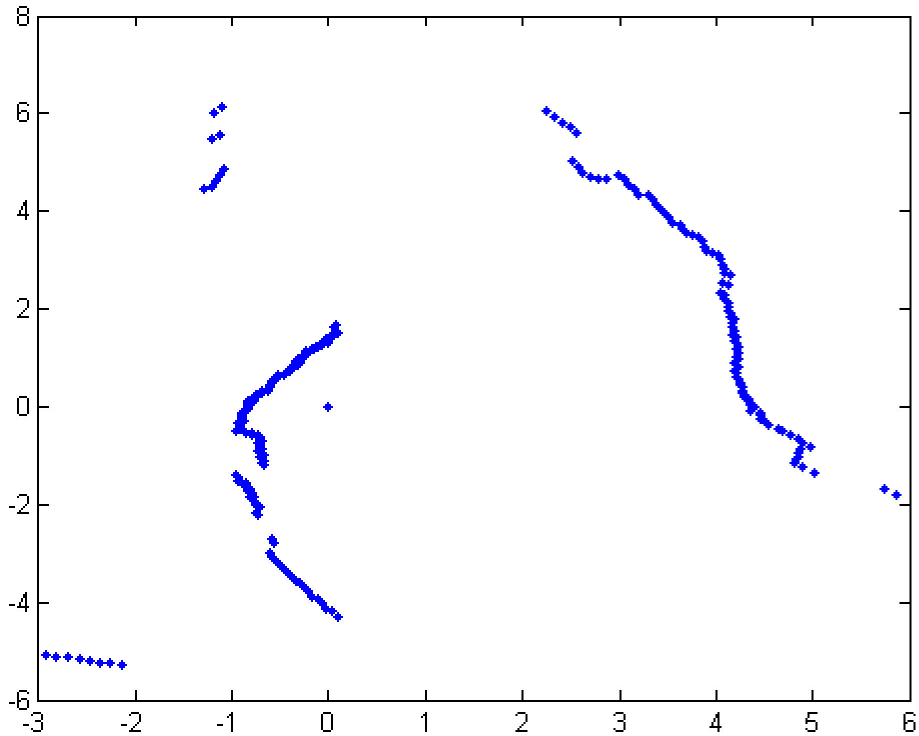

Figure 16:

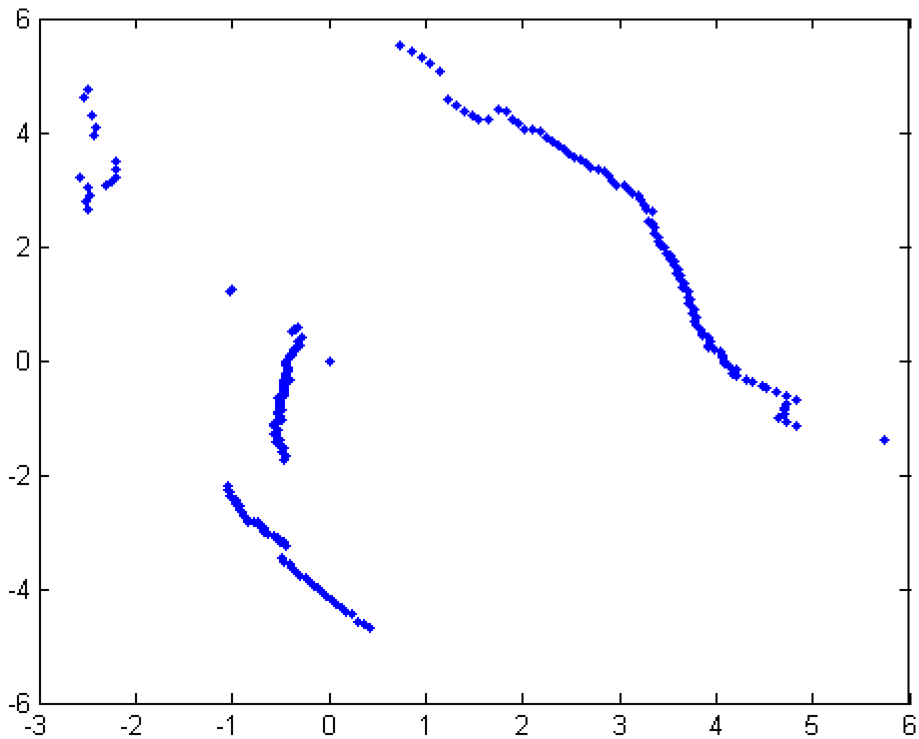

Figure 17:

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:

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

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.