Jack Wilkie

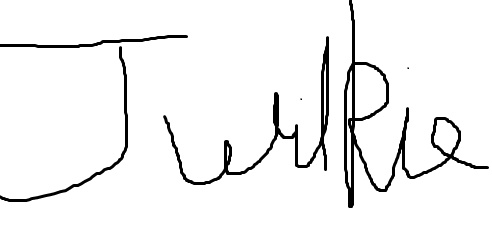
201733825

EE474 Robotics: Systems and Control

Electronics and Electrical Engineering

Semester 2 Assignment Report

**I certify that the material included in this report is my own and has not come from any other source unless otherwise stated.**

**Signed :** 

**Name: Jack Wilkie**

**Date: 07/04/21**

Table of Contents

[1. On Rails Raster Scan 1](#_Toc68735951)

[2. Adding Encoder Noise 2](#_Toc68735952)

[3. Beacon Position Estimates 3](#_Toc68735953)

[4. EKF Implementation 6](#_Toc68735954)

[5. Appendices 9](#_Toc68735955)

[5.1. Appendix 1: Code Listing 9](#_Toc68735956)

# On Rails Raster Scan

A differential drive robot is to perform a raster scan motion, with a length of 2000 mm and a width of 150 mm. It is to perform 7 sweeps of the scan. The wheels can move at a speed of 20 mms-1 in either direction.

The kinematic parameters of the differential drive robot are given by Equation (1).

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

From Equation (1), the rolling velocity of the robot is therefore given by Equation (2).

|  |  |  |
| --- | --- | --- |
|  |  | (2) |

From Equation (1). The angular velocity of the robot is therefore given by Equation (3).

|  |  |  |
| --- | --- | --- |
|  |  | (3) |

From Equation (2), the distance that the robot travels between measurements is therefore calculated as the product of the robot’s speed and its roll velocity as shown given by Equation (4).

|  |  |  |
| --- | --- | --- |
|  |  | (4) |

The sensors update at a rate of 10 Hz. When the robot was travelling to the left this value was added to its x position; or subtracted when the robot was travelling to the right. Similarly, it was either added or subtracted to the y position when the robot travelled vertically.

From Equation (3), the angle that the robot has turned can be calculated as the product of the time between measurements and its angular velocity as given by Equation (5).

|  |  |  |
| --- | --- | --- |
|  |  | (5) |

The robot then produced the required raster scan by updating its x position, y position, and its angle. The wheel speed at each measurement was recorded as a control variable which can be used to have the robot produce the path. The path is shown in Figure 1.

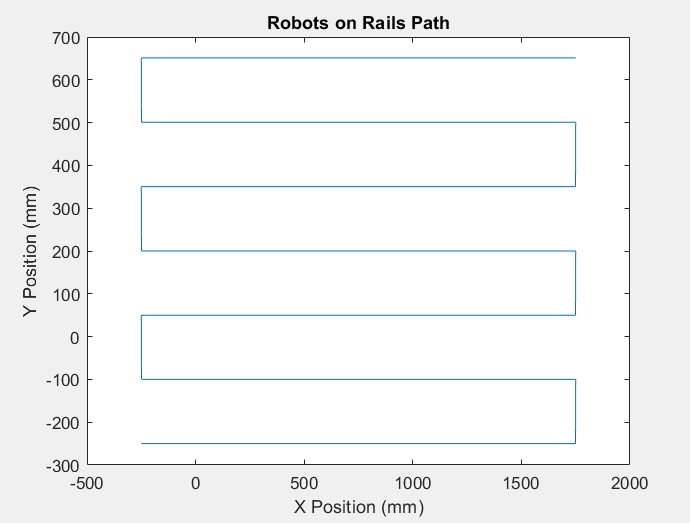


Figure Robots on Rails Path without Noise

# Adding Encoder Noise

In reality the encoders will have a measurement noise. A gaussian noise with a variance of 0.1 (mm s-1)2 was added to each of the wheel speeds recorded to produce the robot’s actual path. Repeating the calculations for the robots x and y positions using noisy wheel speeds gave the robots dead reckoning position estimate.

The x and y distances travelled are now calculated separately by splitting the roll velocity into 2 components as shown by Equations (6) and (7) respectively.

|  |  |  |
| --- | --- | --- |
|  |  | (6) |

|  |  |  |
| --- | --- | --- |
|  |  | (7) |

Where vn is the encoder measurement noise. The calculated value will be negative when the robot is travelling in the opposite direction, therefore, the value was then added to the previous position. The calculation for the robot’s angle remained unchanged.

A comparison plot between the robot’s on rails path and the robot’s dead reckoning location estimate is shown in Figure 2. The dead reckoning follows the robot’s actual path in the beginning, however, very quickly as the robot progressed the noise began to compound, and the position estimate deviates further and further from the actual path. The dead reckoning then varies very significantly from the robots actual path. This makes it unsuitable for accurate position estimates of the robot over anything other than very short distances.

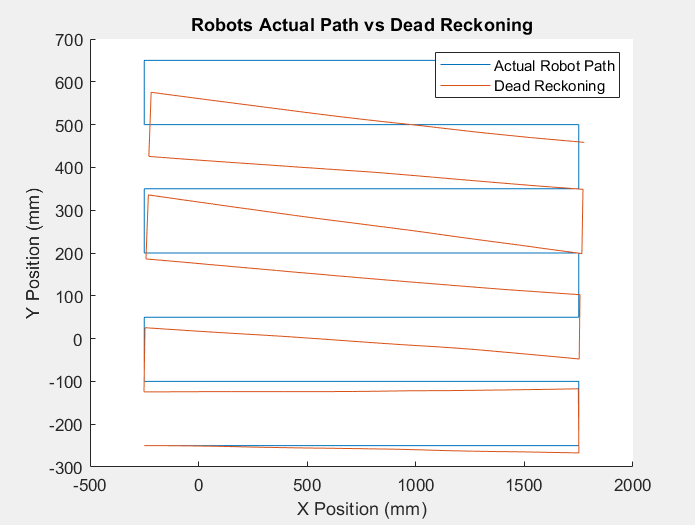


Figure Robots on Rails Path vs Dead Reckoning

# Beacon Position Estimates

The beacons can be used to estimates the robot’s location, without suffering from progressive drift as the robot continues along its path. This is because the beacon measurements do not rely on previous measurements and so error does not accumulate. The beacon measurements, however, are inherently noisier than the encoder measurements.

Each of the beacon measurements can be calculated as Euclidean distance between the robot and the beacon, rxyz. This is done as shown in Equation (8).

|  |  |  |
| --- | --- | --- |
|  |  | (8) |

Where vn is gaussian noise with a with a variance of 20 mm2. xR is the x position of the robot, and xb is the x position of the beacon etc.

Each of the three beacons measures the Euclidean distance between that and the robot. Since the differential drive robot always remains on the floor, assuming that the floor is perfectly flat, the position can be calculated in only 2-diemensions, x and y, and assigning z as 0; as opposed to producing a 3-diemensional, xyz, position estimate. To do this the xyz radius of the robot from the beacon must be converted to a radius on the xy plane. Since all 3 beacons have a z position of 1000 this is illustrated as shown in Figure 3.

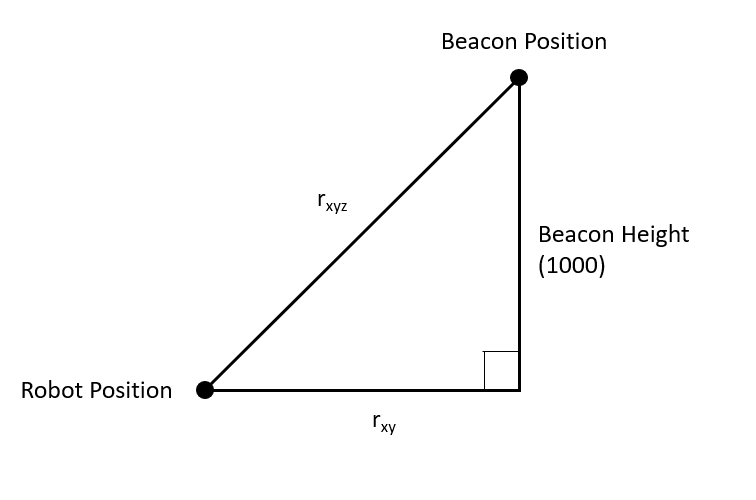


Figure Beacon radius measurements

rxyz is the 3-dimensional radius measured by the beacon. This can be converted to a radius on the xy plane, rxy as given by Equation (9).

|  |  |  |
| --- | --- | --- |
|  |  | (9) |

Considering the 2-d dimensional radius, rxy, for a single beacon, the robot can lie at any point in a circle around the beacon with radius, r, this is given by the Equation (10).

|  |  |  |
| --- | --- | --- |
|  |  | (10) |

Expanding the squares then assigning an equation for each beacon gives Equations (11.1) to (11.3).

|  |  |  |
| --- | --- | --- |
|  |  | (11.1) |
|  |  | (11.2) |
|  |  | (11.3) |

Subtracting the Equation (11.1) from Equation (11.2) gives Equation (12.1).

|  |  |  |
| --- | --- | --- |
|  |  | (12.1) |

Similarly subtracting Equation (11.3) from Equation (11.2) Equation (12.2).

|  |  |  |
| --- | --- | --- |
|  |  | (12.2) |

This is a system of 2 linear equations which can be represented by Equations (13.1) and (13.2).

|  |  |  |
| --- | --- | --- |
|  |  | (13.1) |
|  |  | (13.2) |

The intersection of the 2 lines (the robot’s position) occurs at the position given by Equations (14.1) and (14.2).

|  |  |  |
| --- | --- | --- |
|  |  | (14.1) |
|  |  | (14.2) |

Solving each time that the beacon takes a position measurement (at rate of 10 Hz) gives allow of the beacons position estimates of the robot. Figure 4 shows the beacon position estimates plotted alongside the robot’s actual path and the dead reckoning estimate.

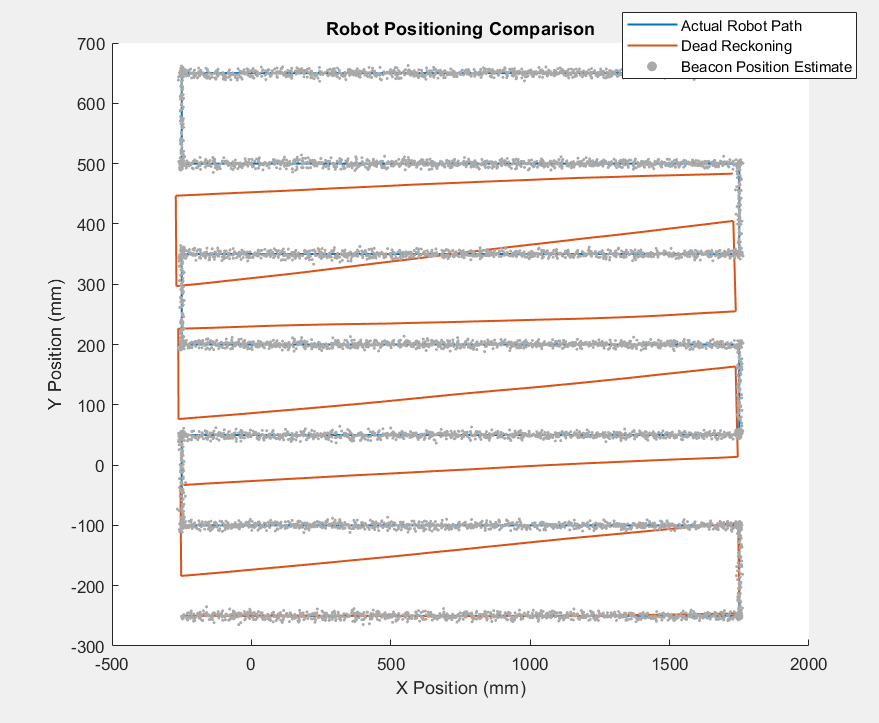


Figure Comparison of Beacon Estimates and Dead Reckoning

The beacon estimates do not drift from the actual path like the dead reckoning does, however, the beacon estimates suffer from noisy estimates, which are very close to the path, but few actually lie on it. Neither estimate is sufficient as a position estimate as dead-reckoning drifts and beacons give noisy estimates

# EKF Implementation

The beacon estimates and the dead reckon estimates were then combined using an EKF. This is done by taking the gaussian distribution of each measurements and multiplying them to find the resulting gaussian distribution and covariance matrix. This value is then used as a more accurate measurement. Since the filter assumes gaussian distribution, nonlinear functions must be linearised around the operating point. This is done by taking the Taylor expansion and then discarding all non-linear (non-first order) terms.

The filter is applied using a prediction step and an update step. In the prediction step a new state estimate, and a new covariance is found. In the update step they predicted values are combined with measured values to give a Kalman filtered state estimation.

In the prediction step, the new predicted is calculated in a similar manner as before. The predicted state matrix,, is given by Equation (15).

|  |  |  |
| --- | --- | --- |
|  |  | (15) |

Here the hat indicates a predicted value of a variable, whilst the + and – superscripts indicate the value is after updating or before updating, respectively. The variance matrix, , in this measurement is then found by linearising the previous variance and adding a linearised version of the measurement noise. This is done given by Equation (16).

|  |  |  |
| --- | --- | --- |
|  |  | (16) |

Since AAT gives the identity matrix, will be a linearised version of P. Here Ak is the Jacobian matrix containing first order derivatives of each equation which give the current state parameters with respect to x. It is given by Equation (17).

|  |  |  |
| --- | --- | --- |
|  |  | (17) |

Wk is the Jacobian matrix containing the first order derivatives of each equation which give the current state parameters with respect to the left and right wheel noise. It is given by Equation (18).

|  |  |  |
| --- | --- | --- |
|  |  | (18) |

Qk is the process noise covariance at k. It is given by Equation (19).

|  |  |  |
| --- | --- | --- |
|  |  | (19) |

Where is the variance in encoder measurements.

In the update step the predicted values are used to find the new state, and its corresponding covariance matrix for the robot.

The robot’s updated state estimate is calculated by finding the difference between the beacon measurements and the radius from each beacon using dead reckoning measurements This is multiplied by the Kalman gain to weight these according to there variance and then added to the previous value as shown in Equation (20).

|  |  |  |
| --- | --- | --- |
|  |  | (20) |

is the Kalman gain, which represents the ratio of the total linearised variance held by each measurement it is given shown in Equation (21).

|  |  |  |
| --- | --- | --- |
|  |  | (21) |

Here V is a 3x3 identity matrix. h is a matrix containing the distance of the robot from each beacon, using the robots current predicted position using the Kalman filter. It is given as shown in Equation (22).

|  |  |  |
| --- | --- | --- |
|  |  | (22) |

z is a matrix containing each of the 3 beacon measurements as given by Equation (23).

|  |  |  |
| --- | --- | --- |
|  |  | (23) |

H is the matrix containing first order partial derivatives of the measurement model with respect to each parameter of the predicted state. It specifies the ratio of each dimension in the radius from the robot to each beacon. It is given by Equation (24).

|  |  |  |
| --- | --- | --- |
|  |  | (24) |

R is a diagonal matrix containing the variance in each of the 3 beacons measurements given in Equation (25).

|  |  |  |
| --- | --- | --- |
|  |  | (25) |

The updated error covariance is then found using the formula shown in Equation (26).

|  |  |  |
| --- | --- | --- |
|  |  | (26) |

The Kalman filter was then implemented to provide a better location estimate than either the beacons or dead reckoning when used on their own. Figure 5 shows a comparison of all of the robot location estimates. The EKF path follows the robot’s actual path much more closely than the dead-reckoning and does not suffer from the same drift. The EKF also does not suffer from the same noise as the beacon position estimates, always following the robot’s actual path very closely. Overall, the EKF is a reliable way to track the robot’s position.

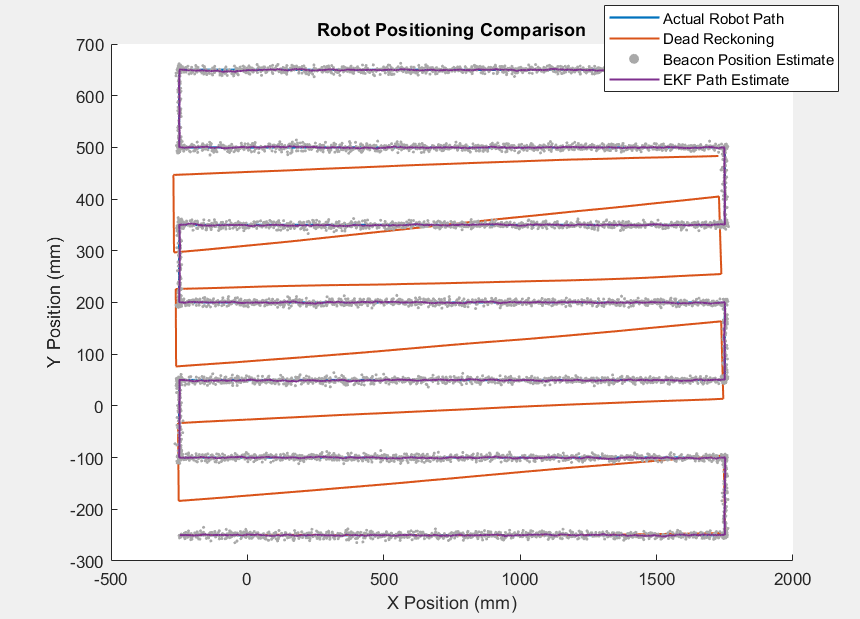


Figure Robot Position Estimate using EKF

# Appendices

## Appendix 1: Code Listing

%% Robotics Assignment Sem 2

%Jack Wilkie

%13/03/21

%Robotics Kalman Filter Assignment

close all;

clear all;

clc

cd 'C:\Users\jackw\Desktop\Robotics\_sem2'

%% Specifiy Robot Parameters

fs= 10; %frequecny of enocder measurements

ts = 1/fs; %enocder period

r = 10; %robot wheel radius

d = 2\*r; %robot wheel diameter

c = pi \* d;

base = 120; %robot wheel base

l = base/2; %length from centre of gravity to wheel

vmax = 20; %maximum robot velocity

varianceR = 0.1; %variance in encoder measurement

%robot starting position

x0 = -250; %robot starting position

y0 = -250;

z0 = 0;

theta0 = 0; %robot starting rotation

%beacon locations

B1 = [2500;2500;1000];

B2 = [-2500;2500;1000];

B3 = [2500;-2500;1000];

varianceB = 20; %variance of beacon distance

%initalise robot parameteres

x = x0;

y = y0;

z = z0;

%left and right wheel speed

vl = vmax;

vr = vmax;

theta = theta0;

t=0;

%% Define Kinematic Eqns as Functions

v\_roll =@(vl,vr) (vr+vl)/2 ; %roll velocity

v\_theta =@(vl,vr) (vr-vl)/(2\*l); %angular velocity

%% Q1 Plot Rails

%track robot position

count = 1;

x\_loc(count,:) = x;

y\_loc(count,:) = y;

theta\_value(count,:) = 0;

wheel\_array(count,:) = [vl,vr];

%rastor scan size

rastor\_l = 2000; %rastor length

rastor\_w = 150; %rastor spacing

rastor\_p = 7; %rastor passes

for sweep = 1:rastor\_p

%reset starting positon after sweep

x0 = x;

y0=y;

theta0 = theta;

if sweep < rastor\_p && mod(sweep,2) == 1 %odd pass travel x turn clock wise travel y then turn again

%travel in x direction

while abs(x-x0) < rastor\_l

count = count + 1; %increment counter

x = x + (v\_roll(vl,vr)\*ts); %find new x position

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, vr]; %store wheel speeds

end

%turn to face y axis

while theta - theta0 < pi/2

count = count + 1; %increment counter

theta = theta + (v\_theta(-vl,vr)\*ts); %turn anticlockwise (with respect to robot)

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [-vl, vr]; %store wheel speeds

end

%travel in y axis

while abs(y-y0) < rastor\_w

count = count + 1; %increment counter

y = y + (v\_roll(vl,vr)\*ts); %find new x position

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, vr]; %store wheel speeds

end

%turn back to x axis

while theta - theta0 < pi

count = count + 1; %increment counter

theta = theta + (v\_theta(-vl,vr)\*ts); %turn anticlockwise (with respect to robot)

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [-vl, vr]; %store wheel speeds

end

elseif sweep < rastor\_p && mod(sweep,2) == 0 %odd pass travel x turn clock wise travel y then turn again

%travel in x direction

while abs(x-x0) < rastor\_l

count = count + 1; %increment counter

x = x - (v\_roll(vl,vr)\*ts); %find new x position

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, vr]; %store wheel speeds

end

%turn to face y axis

while theta - theta0 > -pi/2

count = count + 1; %increment counter

theta = theta + (v\_theta(vl,-vr)\*ts); %turn anticlockwise (with respect to robot)

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, -vr]; %store wheel speeds

end

%travel in y axis

while abs(y-y0) < rastor\_w

count = count + 1; %increment counter

y = y + (v\_roll(vl,vr)\*ts); %find new x position

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, vr]; %store wheel speeds

end

%turn back to x axis

while theta - theta0 > -pi

count = count + 1; %increment counter

theta = theta + (v\_theta(vl,-vr)\*ts); %turn anticlockwise (with respect to robot)

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, -vr]; %store wheel speeds

end

else %final pass- travel length but dont turn

while abs(x-x0) < rastor\_l %travel length of rastor

count = count +1;

x = x + (v\_roll(vl,vr)\*ts); %find new x position

x\_loc(count,:) = x; %store new x location

y\_loc(count,:) =y; %store new y location

theta\_value(count,:) = theta; %store new angle

wheel\_array(count,:) = [vl, vr]; %store wheel speeds

end

end

end

% plot robots rails

rail = [x\_loc y\_loc zeros(length(x\_loc),1) theta\_value];

figure;

plot(x\_loc, y\_loc);

title('Robots on Rails Path');

xlabel('X Position (mm)');

ylabel('Y Position (mm)');

%% Q2 Add Noise

%kinematic eqns estimation with noise

x\_noise\_speed =@(vl,vr,theta) ((vr+vl)/2)\*cos(theta) ; %roll velocity with x noise

y\_noise\_speed =@(vl,vr,theta) ((vr+vl)/2)\*sin(theta) ; %roll velocity with y noise

%generate noise

noise\_left\_wheel = sqrt(varianceR).\*randn(length(wheel\_array),1);

noise\_right\_wheel = sqrt(varianceR).\*randn(length(wheel\_array),1);

%add noise to wheel speed measurements

wheels\_noisy(:,1) = wheel\_array(:,1) + noise\_left\_wheel; %left wheel noise

wheels\_noisy(:,2) = wheel\_array(:,2) + noise\_right\_wheel; %right wheel noise

%generate noisey dead reckoning path measurement

%initalise path

x = -250;

y = -250;

theta = 0;

%add noise to measurements

for i = 1:length(wheel\_array)

dead\_reckoning(i, 1:4) = [x, y, 0 ,theta]; %store robot parameters

x = dead\_reckoning(i,1)+(x\_noise\_speed(wheels\_noisy(i,1), wheels\_noisy(i,2), dead\_reckoning(i,4))\*ts);

y = dead\_reckoning(i,2)+(y\_noise\_speed(wheels\_noisy(i,1), wheels\_noisy(i,2), dead\_reckoning(i,4))\*ts);

theta = dead\_reckoning(i,4)+ (v\_theta(wheels\_noisy(i,1),wheels\_noisy(i,2))\*ts); %calculate robot angle

end

%plot dead reckoning vs ideal path

figure;

hold on

plot(x\_loc, y\_loc);

plot(dead\_reckoning(:,1), dead\_reckoning(:,2))

hold off

title('Robots Actual Path vs Dead Reckoning');

xlabel('X Position (mm)');

ylabel('Y Position (mm)');

legend('Actual Robot Path','Dead Reckoning');

%% Q3 Implement Beacon Estimates

rad\_dist =@(x,y,z,beacon) sqrt((x-beacon(1)).^2 + (y-beacon(2)).^2 + (z-beacon(3)).^2); %find radial distance of robot from beacon with variance

r\_b1 = rad\_dist(rail(:,1),rail(:,2),rail(:,3),B1);

r\_b2 = rad\_dist(rail(:,1),rail(:,2),rail(:,3),B2);

r\_b3 = rad\_dist(rail(:,1),rail(:,2),rail(:,3),B3);

%generate beacon noise

r\_b1\_noise = sqrt(varianceB).\*randn(length(r\_b1),1);

r\_b2\_noise = sqrt(varianceB).\*randn(length(r\_b2),1);

r\_b3\_noise = sqrt(varianceB).\*randn(length(r\_b3),1);

%add noise to beacon position estimate

r\_b1\_noisy = r\_b1 + r\_b1\_noise;

r\_b2\_noisy = r\_b2 + r\_b2\_noise;

r\_b3\_noisy = r\_b3 + r\_b3\_noise;

%convert to 2d radius measurement

r\_b1\_2d = sqrt((r\_b1\_noisy.^2)-(1000^2));

r\_b2\_2d = sqrt((r\_b2\_noisy.^2)-(1000^2));

r\_b3\_2d = sqrt((r\_b3\_noisy.^2)-(1000^2));

%coefficencts of linear eqns

A = (-2\*B1(1))+(2\*B2(1));

B = (-2\*B1(2))+(2\*B2(2));

C = (r\_b1\_2d.^2) - (r\_b2\_2d.^2) -(B1(1)^2) + (B2(2)^2) - (B1(2)^2) + (B2(2)^2);

D = (-2\*B2(1))+(2\*B3(1));

E =(-2\*B2(2))+(2\*B3(2));

F = (r\_b2\_2d.^2) - (r\_b3\_2d.^2) - (B2(1)^2) + (B3(1)^2) - (B2(2)^2) + (B3(2)^2);

%find linear intersections (location estimate)

x\_est = ((C.\*E)-(F.\*B))./((A.\*E)-(B.\*D));

y\_est = ((C.\*D)-(A.\*F))./((B.\*D)-(A.\*E));

%Plot beacon estimates

figure;

hold on

plot(x\_loc, y\_loc,'LineWidth',1.2);

plot(dead\_reckoning(:,1), dead\_reckoning(:,2),'LineWidth',1.2);

%scatter(x\_est,y\_est,1,[154/255 205/255 50/255],'filled');

scatter(x\_est,y\_est,1,[169/255 169/255 169/255],'filled');

hold off

title('Robot Positioning Comparison');

xlabel('X Position (mm)');

ylabel('Y Position (mm)');

legend('Actual Robot Path','Dead Reckoning', 'Beacon Position Estimate');

%% Q4 Implement Kalman Filter

%measurement noise

R = [varianceB,0,0;0,varianceB,0;0,0,varianceB];

V = [1,0,0;0,1,0;0,0,1]; %3x3 identity matrix

%process noise

Q = [varianceR,0;0,varianceR];

b\_est = []; %beacon position estimates

p0 = zeros(4,4); %starting postition

P = p0; %postition

xkal = zeros(length(x\_est),4); %kalman filtered

prev\_state = rail(1,:);

for i = 1:length(x\_est)

%update state using wheel speeds

current\_state = [prev\_state(1) + (x\_noise\_speed(wheels\_noisy(i,1),wheels\_noisy(i,2),prev\_state(4))\*ts)

prev\_state(2) + (y\_noise\_speed(wheels\_noisy(i,1),wheels\_noisy(i,2),prev\_state(4))\*ts)

prev\_state(3)

prev\_state(4) + (v\_theta(wheels\_noisy(i,1),wheels\_noisy(i,2))\*ts) ];

%calculate jacobians

%parial derivatives of state parameters

A = [1,0,0, (-1/2)\*(wheels\_noisy(i,1) + wheels\_noisy(i,2))\*sin(prev\_state(4)); 0,1,0, (1/2)\*(wheels\_noisy(i,1) + wheels\_noisy(i,2))\*cos(prev\_state(4)); 0,0,1,0; 0,0,0,1];

%partial derivatives of left and right wheel noise

W = [cos(prev\_state(4))\*(ts/2) sin(prev\_state(4))\*(ts/2)

(cos(prev\_state(4))\*ts)/2 (sin(prev\_state(4))\*ts)/2

0 0

ts/l ts/l];

%measurement jacobian

L1 = sqrt(((current\_state(1)-B1(1))^2) + ((current\_state(2)-B1(2))^2) + ((current\_state(3)-B1(3))^2));

L2 = sqrt(((current\_state(1)-B2(1))^2) + ((current\_state(2)-B2(2))^2) + ((current\_state(3)-B2(3))^2));

L3 = sqrt(((current\_state(1)-B3(1))^2) + ((current\_state(2)-B3(2))^2) + ((current\_state(3)-B3(3))^2));

H = [(current\_state(1)-B1(1))/L1 (current\_state(2)-B1(2))/L1 (current\_state(3)-B1(3))/L1 0

(current\_state(1)-B2(1))/L2 (current\_state(2)-B2(2))/L2 (current\_state(3)-B2(3))/L2 0

(current\_state(1)-B3(1))/L3 (current\_state(2)-B3(2))/L1 (current\_state(3)-B3(3))/L3 0];

%beacon measurements

z = [r\_b1\_noisy(i);r\_b2\_noisy(i);r\_b3\_noisy(i)];

%beacon distance estimates using filtered state estimate

h = [rad\_dist(current\_state(1),current\_state(2),current\_state(3),B1)

rad\_dist(current\_state(1),current\_state(2),current\_state(3),B2)

rad\_dist(current\_state(1),current\_state(2),current\_state(3),B3)];

%kalman filter

P = A\*P\*A' + W\*Q\*W';

K = P\*H'\*inv(H\*P\*H' + V\*R\*V');

%state correction

state = current\_state + K\*(z-h);

xkal(i,1:4) = state(1:4)';

P = (eye(4)-K\*H)\*P;

prev\_state = state;

end

figure;

hold on

plot(x\_loc, y\_loc,'LineWidth',1.4);

plot(dead\_reckoning(:,1), dead\_reckoning(:,2),'LineWidth',1.2);

%scatter(x\_est,y\_est,1,[154/255 205/255 50/255],'filled');

scatter(x\_est,y\_est,1,[169/255 169/255 169/255],'filled');

plot(xkal(:,1),xkal(:,2),'LineWidth',1.2);

hold off

title('Robot Positioning Comparison');

xlabel('X Position (mm)');

ylabel('Y Position (mm)');

legend('Actual Robot Path','Dead Reckoning', 'Beacon Position Estimate', 'EKF Path Estimate');