MTE 544 - Autonomous Mobile Robotics

Assignment 1

By: Lalit Lal (20572296), Michal Kaca (20564560)
Date: 03/11/2018

Parts 1 and 2: Motion Model Derivation and Simulation:
There are three parts to deriving the motion model in the form of $\dot{\zeta} = A\zeta + Bu,$ where $\zeta$is the robot state, and u is the control input. As defined in the problem, the robot state will be the robot pose, as defined by

$$\zeta = \begin{matrix} x\ [m] \\ y\ [m] \\ \theta\ [rads] \end{matrix}$$

and the control input, as defined by the problem, are the main wheel angular speeds,

$$u = \begin{matrix} \omega_1\ \left[\frac{rad}{s}\right] \\ \omega_2\ \left[\frac{rad}{s}\right] \\ \omega_3\ \left[\frac{rad}{s}\right] \end{matrix}$$

The three parts involve determining the coordinate transformation from the robot frame to the global frame, determining the Swedish wheel constraints, and finally, applying the wheel constraints to determine the robot kinematics constraints and model.

Coordinate Transformation:
As derived in class, the transformation from the global frame to the robot's local frame is defined by an orthogonal matrix,

$$R(\theta) = \begin{matrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix}$$

where the transpose yields the transformation from the robot frame to the global frame is the inverse of the rotation matrix. Since the rotation matrix is orthogonal, the inverse is its transpose,

$$R(\theta)^{-1} = \begin{matrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix}$$

Therefore, converting pose from the robot's frame ζ_r, to the global frame ζ_I, is done as by multiplying the inverse rotation matrix to the robot frame pose

$$\zeta_I = R(\theta)^{-1}\zeta_r$$

Wheel Constraints: Swedish Wheel
The robot moves using three Swedish wheels. The three Swedish wheels allow for omni-directional motion because of the inner rollers that are not along the same axis of rotation as the main wheels. The wheel constraints of the Swedish wheel are based from the wheel constraints of a standard fixed wheel, outlined in **figure 1** below obtained from a Carnegie Mellon paper (1). Namely, α is defined as the angle between the local frame and the wheel base, β is defined as the angle between the wheel plane and the chassis. It is important to note that these two are fixed since the local axis fixed, as it is always with respect to the robot, and the wheel is not steerable, so the wheel base angle cannot change. The parameter l, as defined in the assignment, is the length between the robot base and the wheel base.

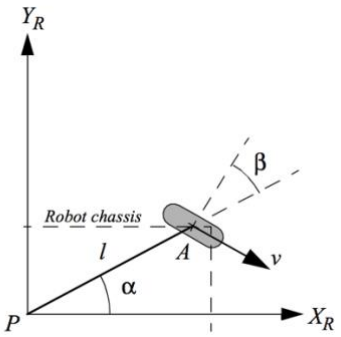

Figure 1: Standard Fixed Wheels

The wheel constraint for wheel relates the total motion along a wheel plane to the rotation from the spinning wheel. This is shown in equation form below.

$$[sin(\alpha+\beta) \quad -cos(\alpha+\beta) \quad (-l)cos(\beta)]R(\theta)\dot{\zeta}_I = r\omega_i$$

In the left-hand side is the total motion along the wheel plane, and the right-hand side is the motion from spinning the wheel. Now, to extend this to the Swedish wheel, **figure 2** is inspired from a Carnegie Mellon paper (1).

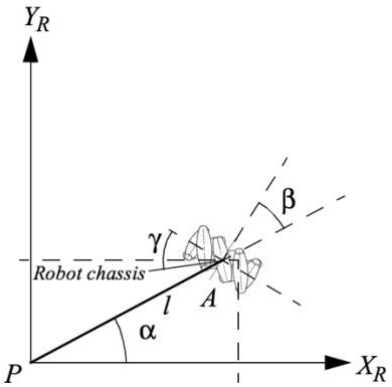

Figure 2: Swedish Wheel Setup

The only difference between the Swedish wheel and the fixed wheel is an additional parameter, γ, which is the angle between the main wheel plane and the axis of rotation of the small rollers. For visual demonstration, if $\gamma = \frac{\pi}{2}$, then the Swedish wheel behave exactly like the fixed wheel. With this acknowledgement, the Swedish wheel motion constraint becomes

$$[sin(\alpha+\beta+\gamma) \quad -cos(\alpha+\beta+\gamma) \quad (-l)cos(\beta)]R(\theta)\dot{\zeta}_I = r\omega_i \cos(\gamma)$$

which simplifies to,

$$J_1 R(\theta)\dot{\zeta}_I = r\omega_i \cos(\gamma)$$

where J_1 is a nx3 matrix, n being the number of wheels.

$$J_1 = [sin(\alpha+\beta+\gamma) \quad -cos(\alpha+\beta+\gamma) \quad (-l)cos(\beta)]$$

Specific to the assignment, there are three wheels. For all wheels, $\beta = 0$ and $\gamma = 0$ since all wheels are tangential to the circular robotic body and the rollers axis of rotation is perpendicular to the axis of rotation of the main wheel. Finally, from the assignment specifications, for wheel 1, $\alpha_1 = \pi/2, \alpha_2 = \frac{7}{6}\pi, \alpha_3 = -\frac{\pi}{6}$.

From the assignment, we know the length of the wheelbase, l, is 0.3 meters. This information simplifies our matrix J1, for three wheels, to the following:

$$J_1 = \begin{bmatrix} sin(\pi/2) & -cos(\pi/2) & (0.3) \\ sin\left(\frac{7}{6}\pi\right) & -cos\left(\frac{7}{6}\pi\right) & (0.3) \\ sin\left(-\frac{\pi}{6}\right) & -cos\left(-\frac{\pi}{6}\right) & (0.3) \end{bmatrix}$$

Therefore the motion constraint equation for three Swedish wheels becomes

$$J_1 R(\theta)\dot{\zeta}_I = J2\begin{bmatrix}\omega_1 \\ \omega_2 \\ \omega_3\end{bmatrix}$$

where J2 is the diagonal matrix with the diagonal elements as the radius for each wheel. In this case, all wheels are the same with a radius r = 0.25m. Finally, the wheel constraint becomes

$$\begin{bmatrix} sin(\pi/2) & -cos(\pi/2) & (0.3) \\ sin\left(\frac{7}{6}\pi\right) & -cos\left(\frac{7}{6}\pi\right) & (0.3) \\ sin\left(-\frac{\pi}{6}\right) & -cos\left(-\frac{\pi}{6}\right) & (0.3) \end{bmatrix} R(\theta)\dot{\zeta}_I = \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}\begin{bmatrix}\omega_1 \\ \omega_2 \\ \omega_3\end{bmatrix}$$

Robot Kinematics Constraints:
Uniquely, the Swedish heel imposes no kinematic constraints on the robot chassis since the wheels make the robot omni-directional.

Therefore, solving for the motion of the robot, the following equation is obtained,

$$\dot{\zeta}_I = R(\theta)^{-1}\begin{bmatrix} sin\left(\frac{\pi}{2}\right) & -cos\left(\frac{\pi}{2}\right) & (0.3) \\ sin\left(\frac{7}{6}\pi\right) & -cos\left(\frac{7}{6}\pi\right) & (0.3) \\ sin\left(-\frac{\pi}{6}\right) & -cos\left(-\frac{\pi}{6}\right) & (0.3) \end{bmatrix}^{-1} \begin{bmatrix} 0.25 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}\begin{bmatrix}\omega_1 \\ \omega_2 \\ \omega_3\end{bmatrix}$$

In software, updating the robot pose using the motion model at a rate of 10Hz (i.e., dt = 0.1), with a deviation of 0.01 meters in x,y and 0.1 degree in θ, the equation becomes

$$\zeta_{I,t} = \zeta_{I,t} + \dot{\zeta}_I dt + \varepsilon$$

where ε is the Gaussian noise defined by the deviations in the assignment. The Gaussian noise is defined, as from lectures, as follows:

$$\varepsilon = E_R \lambda_R^{\frac{1}{2}} randn(n, 1)$$

Where E_R, λ_R are the eigen vector and value pairs that are obtained from the covariance matrix R, which is a matrix defined by the diagonals being the squared value of each state deviation.

$$R = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & \left(\frac{0.1\pi}{180}\right)^2 \end{bmatrix}$$

$$E_R = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix},\ \lambda_R \begin{bmatrix} 3.0462e-06 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

Simulations:
The output from the motion model with inputs w = [-1.5 2.0 1.0] is shown below. The blue path is the ideal motion without noise, and the red path is the motion model with noise.

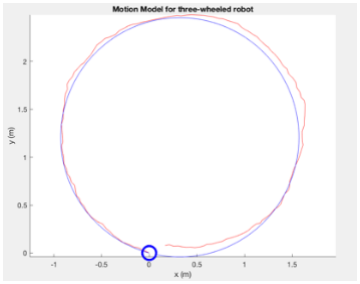

Figure 3: Ideal vs. Noise Injected Motion Model of 3 Wheeled Robot. Inputs are [-1.5 2.0 1.0].

To determine the required inputs a straight line, one must simply assign the same rotational value for two wheels, but in the opposite direction of one another, with no input to the third wheel. For example, set inputs as w = $[\omega \quad -\omega \quad 0]$. The output for w = [-4.0; 0; 4.0] is shown below.

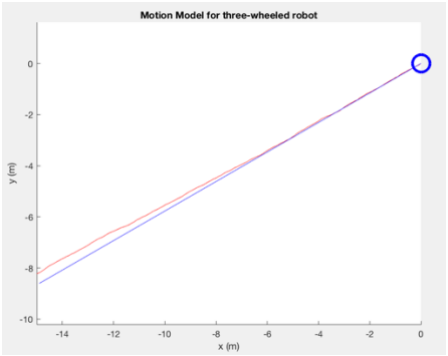

Figure 4: Ideal vs Noise Injected Straight Line Path. W = [-4 0 4]

For a circle of radius R, the path pose of the robot is

$$\zeta_I = \begin{bmatrix} Rcos(\theta) \\ Rsin(\theta) \\ \theta \end{bmatrix}$$

This shows that the motion of the robot in a circular path is defined by

$$\dot{\zeta}_I = \begin{bmatrix} -R\omega sin(\theta) \\ R\omega cos(\theta) \\ \omega \end{bmatrix}$$

where ω is an arbitrary rotational, and is the same value for each wheel. For the assignment, the radius of choice is 1 meter since the diameter is 2 meters. Therefore, working backwards from the ideal motion model, we can see that

$$u = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = (R(\theta)^{-1} J_1^{-1} J_2)^{-1} \begin{bmatrix} -R\omega sin(\theta) \\ R\omega cos(\theta) \\ \omega \end{bmatrix}$$

Solving this symbolically in MATLAB for a rotational speed of w = 0.5 rad/s, the inputs become: [-0.6000 1.1321 -2.3321]. This is plotted from a starting position of [0 0 0.0].

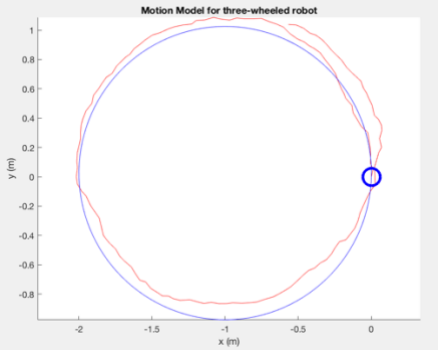

Figure 5: Ideal vs. Noise Injected 2M Circle Path. W = [-0.6 1.132 2.332]

To solve for which inputs will be required for a circular path, a similar approach is followed for the circular path, except the radius is now a function of the robot heading. Specifically, given a dynamic radius $r = a\theta$, a is a scalar value where the pose of the robot is below.

$$\zeta_I = \begin{bmatrix} a\theta cos(\theta) \\ a\theta sin(\theta) \\ \theta \end{bmatrix}$$

The motion model then becomes the equation below, where ω is a rotational rate of the whole robot.

$$\dot{\zeta}_I = \begin{bmatrix} a[\omega cos(\theta) + \theta\omega cos(\theta) - \theta \sin(\theta)] \\ a[\omega sin(\theta) + \theta\omega sin(\theta) + \theta \cos(\theta)] \\ \omega \end{bmatrix}$$

The equation to solve for the wheel speed inputs is seen below, and can be solved symbolically in MATLAB.

$$u = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = (R(\theta)^{-1} J_1^{-1} J_2)^{-1} \begin{bmatrix} a[\omega cos(\theta) + \theta\omega cos(\theta) - \theta \sin(\theta)] \\ a[\omega sin(\theta) + \theta\omega sin(\theta) + \theta \cos(\theta)] \\ \omega \end{bmatrix}$$

By setting $a = 1$, and the vehicle rotation rate to 1.0 rad/s, the following was inputs were obtained:

$$u = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \begin{bmatrix} 4\theta + \frac{14}{5} \\ 2\sqrt{3}\theta - 2\theta - \frac{16}{5} \\ -2\theta - 2\sqrt{3}\theta - \frac{16}{5} \end{bmatrix}$$

This robot motion with this input yielded the output below.

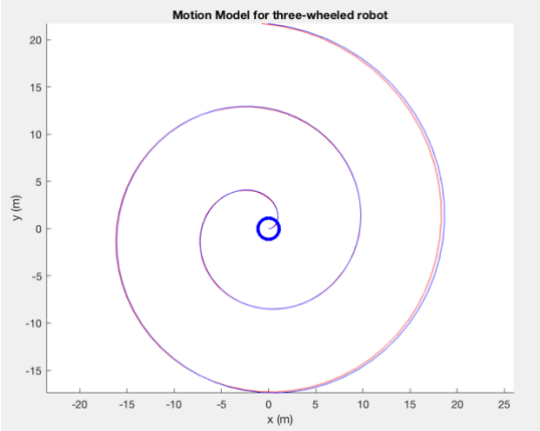

Figure 6: Ideal vs. Noise Injected Spiral Path

Part 3: Measurement Model:
Due to explicitness in the instructions, the measurement model was a simple derivation.

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{9.7\pi}{180} \end{bmatrix} + \delta$$

where δ is the Gaussian noise, depicted as

$$\delta = E_Q \lambda_Q^{\frac{1}{2}} randn(n, 1)$$

and E_Q, λ_Q are the eigen vector and value pairs that are obtained from the covariance matrix Q, as defined below. Q is a matrix defined by the diagonals being the squared value of each measurement deviation.

$$Q = \begin{bmatrix} 0.5^2 & 0 & 0 \\ 0 & 0.5^2 & 0 \\ 0 & 0 & \left(\frac{10\pi}{180}\right)^2 \end{bmatrix}$$

$$E_Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \ \lambda_Q \begin{bmatrix} 0.030462 & 0 & 0 \\ 0 & 0.25 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$$

Part 4: Extended Kalman Filter

Prediction Update:
Recalling from above, the ideal motion model of the robot is shown below.

$$\zeta_{I,t} = \zeta_{I,t} + \dot{\zeta}_I dt$$

Expanding this in MATLAB, the motion for the robot is defined below.

$$\zeta_{I,t} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} x - dt[\frac{w_2}{12}(\cos(\theta) + \sqrt{3}\sin(\theta)) + \frac{w_3}{12}(\cos(\theta) - \sqrt{3}\sin(\theta)) - \frac{w_1}{6}(\cos(\theta))] \\ y - dt[\frac{w_2}{12}(\sin(\theta) - \sqrt{3}\cos(\theta)) + \frac{w_3}{12}(\sin(\theta) + \sqrt{3}\cos(\theta)) - \frac{w_1}{6}(\sin(\theta))] \\ \theta - dt * \frac{5}{18}[w_1 + w_2 + w_3] \end{bmatrix}$$

Using MATLAB, obtaining the Jacobian of this nonlinear motion model is defined below, where θ is the mean from the previous time step. In other words, θ from below is substituted by $\mu_{3,t-1}$

$$G = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & \frac{\partial f_3}{\partial x_3} \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & dt[\frac{w_2}{12}(\sin(\theta) - \sqrt{3}\cos(\theta)) + \frac{w_3}{12}(\sin(\theta) + \sqrt{3}\cos(\theta)) - \frac{w_1}{6}(\sin(\theta))] \\ 0 & 1 & -dt[\frac{w_2}{12}(\cos(\theta) + \sqrt{3}\sin(\theta)) + \frac{w_3}{12}(\cos(\theta) - \sqrt{3}\sin(\theta)) - \frac{w_1}{6}(\cos(\theta))] \\ 0 & 0 & 1 \end{bmatrix}$$

The covariance matrix R, is the same one used as defined in part 1.

Measurement Update:
Since the measurement model from part 2 is linear, we can directly use this for our measurement update. Therefore, using the measurement model, from above, we obtain the H matrix below.

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \bar{\mu}_{1,t-1} \\ \bar{\mu}_{2,t-1} \\ \bar{\mu}_{3,t-1} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{-9.7\pi}{180} \end{bmatrix}$$

The rest of the calculations for the EKF are dependent on G, H, R, and Q and will be left out as it is an easy implementation on MATLAB.

Part 5: EKF Simulation
The 15 second simulation below is for an input of w = [-1.5, 2.0, 1.0], and starting point and prior of [3.0, 1.0, 0.0]. The estimate is shown in blue, and the true motion in shown in red.

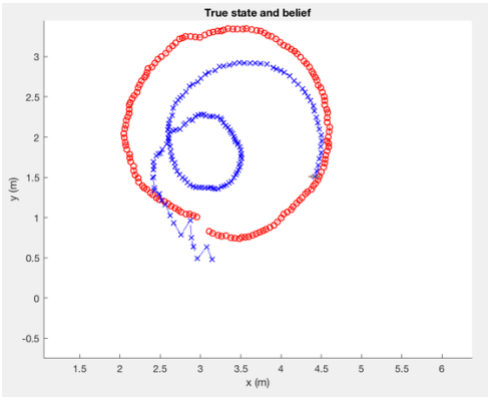

Figure 7: True vs Belief Motion with Perfect Prior

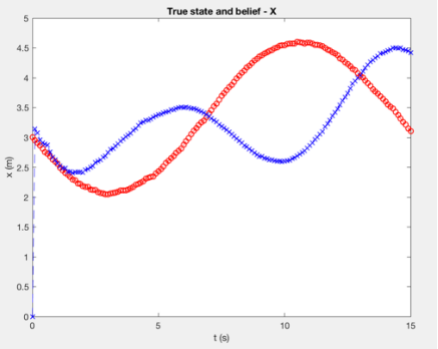

Figure 8: True vs Belief for X position (m)

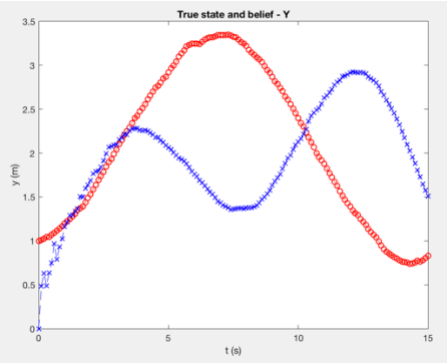

Figure 9: True vs Belief for Y Position (m)

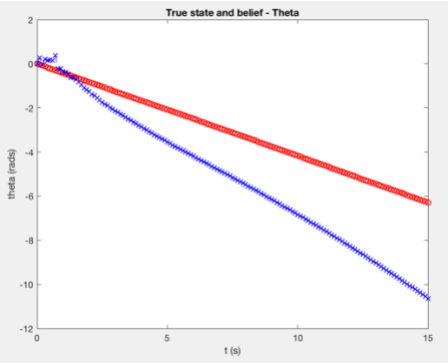

Figure 10: True vs Belief for Heading (radians)

Part 6: Multi-Rate Kalman Filter:

The new GPS corrections at a rate of 1Hz does not change rate of update, which is 10Hz, since that is the greatest common factor of 1 and 10. Consequently, at time steps 1 to 9, the measurement update looks the same as from part 2.

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{9.7\pi}{180} \end{bmatrix} + \delta$$

However, at a time step that is a multiple of 10, there's an improved GPS correction, which essentially replaces the original GPS measurement. The only change to the measurement update equation is the new noise, δ_2.

$$y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -\frac{9.7\pi}{180} \end{bmatrix} + \delta_2$$

Here, δ_2 is the new Gaussian noise, depicted as

$$\delta_2 = E_{Q_2} \lambda_{Q_2}^{\frac{1}{2}} randn(n, 1)$$

and E_{Q_2}, λ_{Q_2} are the eigen vector and value pairs that are obtained from the new covariance matrix Q_2, as defined below. Q_2 is a matrix defined by the diagonals being the squared value of each measurement deviation. The eigen vectors and eigen values are determined using MATLAB.

$$Q_2 = \begin{bmatrix} 0.01^2 & 0 & 0 \\ 0 & 0.01^2 & 0 \\ 0 & 0 & \left(\frac{10\pi}{180}\right)^2 \end{bmatrix}$$

$$E_{Q_2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \lambda_{Q_2} \begin{bmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.030462 \end{bmatrix}$$

The following are the updated plots for the Multi-EKF. Red is the true path, blue is the belief path.

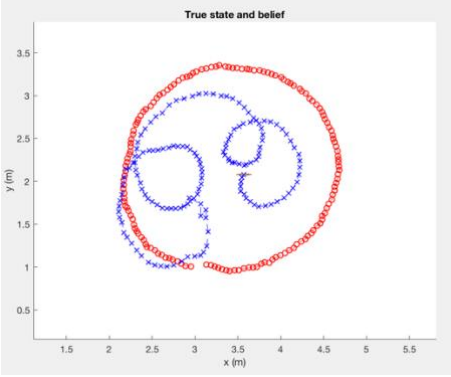

Figure 11: True vs Belief with Perfect Prior

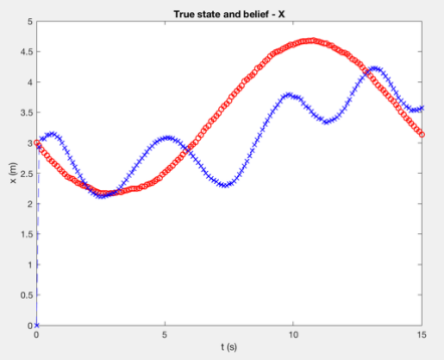

Figure 12: True vs Belief for X position (m)

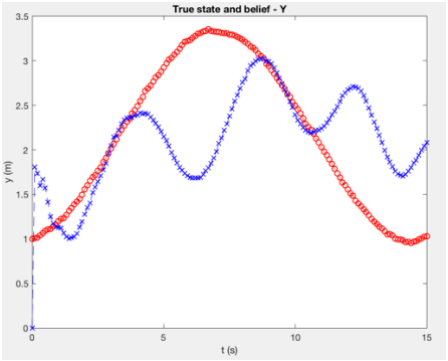

Figure 13: True vs Belief for Y Position (m)

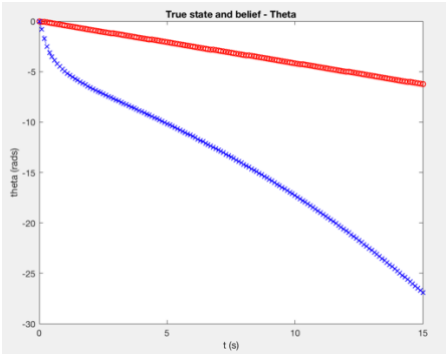

Figure 14: True vs Belief for Theta Position (radians)

Appendix:

Code for Parts 1, 2 (Simulation):

```
clc
close all
clear all
%% Two-wheeled robot motion model
r = 0.25;
l = 0.3;

al1 = pi/2;
al2 = 7/6 * pi;
al3 = -pi / 6;
J1 = [sin(al1) -cos(al1) -l; sin(al2) -cos(al2) -l; sin(al3) -cos(al3) -l];
J1 = inv(J1);
diag_vals = [r r r];
J2 = diag(diag_vals);
J1J2 = J1*J2;

dt = 0.1; % Timestep
%x0 = [0 0 0.1]';
x = [0.0 0.0 0.0]'; % Initial State
x1 = [0.0 0.0 0.0]';
w1 = -0.6;
w2 = 1.1341;
w3 = -2.3321;

w_initial = [-1.5; 2.0; 1.0];
w_circle2m = [-0.6000; 1.1321; -2.3321];
w_straight = [-4.0; 0; 4.0];
%w_spiral = [2/5; 3^(1/2)*theta - 17848520; - 3^(1/2)*theta - 17848520];

w = w_circle2m;

R = diag([0.01^2 0.01^2 (0.1*pi/180)^2])
[RE, Re] = eig(R);

n=150; % Samples
for i=2:n
    % Disturbance
    % Dynamics
    rot_mat = [cos(x(3,i-1)) -sin(x(3,i-1)) 0; sin(x(3,i-1)) cos(x(3,i-1)) 0; 0 0 1];
    w_spiral = [4*x(3,i-1) + 14/5;
                2*3^(1/2)*x(3,i-1) - 2*x(3,i-1) - 16/5;
                -2*x(3,i-1) - 2*3^(1/2)*x(3,i-1) - 16/5];
    RJ1J2W = rot_mat * J1 * J2 * w_spiral; %or w_spiral
    x(:,i) = x(:,i-1) + RJ1J2W * dt + RE*sqrt(Re)*randn(3,1);

    rot_mat1 = [cos(x1(3,i-1)) -sin(x1(3,i-1)) 0; sin(x1(3,i-1)) cos(x1(3,i-1)) 0; 0 0 1];
   w_spiral = [4*x1(3,i-1) + 14/5;
                2*3^(1/2)*x1(3,i-1) - 2*x1(3,i-1) - 16/5;
                -2*x1(3,i-1) - 2*3^(1/2)*x1(3,i-1) - 16/5];
    RJ1J2W1 = rot_mat1 * J1 * J2 * w_spiral; % or w_spiral
    x1(:,i) = x1(:,i-1) + RJ1J2W1 * dt;
    %x(:,i) = x0 + [ dt*(v+E(1))*cos(x0(3)+E(2)) ;  dt*(v+E(1))*sin(x0(3)+E(2)); E(2)+ dt*w];
end
% Disturbance free dynamics
%x1 = x0 + RJ1J2W * dt;

% Plot
figure(1); clf; hold on;
plot(x(1,:), x(2,:), 'Color', 'r');
plot (x1(1,:), x1(2,:), 'Color', 'b');
plot( x(1,1), x(2,1), 'bo', 'MarkerSize',20, 'LineWidth', 3)
%plot( x1(1), x1(2), 'bo', 'MarkerSize',20, 'LineWidth', 3)
%plot( [x0(1) x1(1)], [x0(2) x1(2)],'b')
%plot( x(1,:),x(2,:), 'm.', 'MarkerSize', 3)
title('Motion Model for three-wheeled robot')
xlabel('x (m)');
ylabel('y (m)');
axis equal
```

Code for Parts 3-6 (EKF):

```
% Extended Kalman filter example
clear;clc;

%System Variables
r = 0.25;
l = 0.3;

al1 = pi/2;
al2 = 7/6 * pi;
al3 = -pi / 6;
J1 = [sin(al1) -cos(al1) -l; sin(al2) -cos(al2) -l; sin(al3) -cos(al3) -l];
J1 = inv(J1);
diag_vals = [r r r];
J2 = diag(diag_vals);
J1J2 = J1*J2;

%  Video recording
%videoobj=VideoWriter('ekf_cur.mp4','MPEG-4');
%truefps = 1;
%videoobj.FrameRate = 10; %Anything less than 10 fps fails.
%open(videoobj);


% Discrete time step
dt = 0.1;

% Initial State
x0 = [3.0 1.0 0.0]';

% Prior
mu = [3.0 1.0 0.0]'; % mean (mu)
S = 1*eye(3);% covariance (Sigma)

% Discrete motion model
%Ad = [ 1 dt 0 ; 0 1 0; 0 0 1];

R = [0.01^2 0 0; 0 0.01^2 0 ; 0 0 (0.1*pi/180)^2];
[RE, Re] = eig(R);

% Measurement model defined below
Q = [0.5^2 0 0; 0 0.5^2 0; 0 0 (10*pi/180)^2];
[QE, Qe] = eig(Q);

Q2 = [0.01^2 0 0; 0 0.01^2 0; 0 0 (10*pi/180)^2];
[QE2, Qe2] = eig(Q2);

% Simulation Initializations
Tf = 15;
T = 0:dt:Tf;
n = 3; %length(Ad(1,:));
x = zeros(n,length(T));
x(:,1) = x0;
m = length(Q(:,1));
y = zeros(m,length(T));
mup_S = zeros(n,length(T));
mu_S = zeros(n,length(T));
w = [-1.5; 2.0; 1.0];

GPS_correct = 0;
%% Main loop
for t=2:length(T)

    if(mod(1,10) == 0)
        GPS_correct = 1;
    else
        GPS_correct = 0;
    end
    %% Simulation
    % Select a motion disturbance
    e = RE*sqrt(Re)*randn(n,1);
    % Update state
    rot_mat = [cos(x(3,t-1)) -sin(x(3,t-1)) 0; sin(x(3,t-1)) cos(x(3,t-1)) 0; 0 0 1];
    RJ1J2W = rot_mat * J1 * J2 * w;
```

```
    x(:,t) = x(:,t-1) + RJ1J2W * dt + e;
    %x(:,t) = Ad*x(:,t-1) + e;

    % Take measurement
    % Select a motion disturbance
    if(GPS_correct)
        d = QE2*sqrt(Qe2)*randn(m,1);
    else
        d = QE*sqrt(Qe)*randn(m,1);
    end

    % Determine measurement
    DEC = [0; 0; -9.7*pi/180];
    B = [1 0 0; 0 -1 0; 0 0 1];
    y(:,t) = B*x(:,t) + DEC + d;
    %y(:,t) = sqrt(x(1,t)^2 + x(3,t)^2) + d;

    %% Extended Kalman Filter Estimation
    % Prediction update
    %mup = Ad*mu;
    %Sp = Ad*S*Ad' + R;

    % Linearization
    Ad = [1 0 dt*(w(2)*(sin(mu(3))/12 - (3^(1/2)*cos(mu(3)))/12) + w(3)*(sin(mu(3))/12 +
(3^(1/2)*cos(mu(3)))/12) - (w(1)*sin(mu(3)))/6);
          0 1 -dt*(w(2)*(cos(mu(3))/12 + (3^(1/2)*sin(mu(3)))/12) + w(3)*(cos(mu(3))/12 -
(3^(1/2)*sin(mu(3)))/12) - (w(1)*cos(mu(3)))/6);
          0 0 1]; % THIS IS OUR G

    %mup = Ad*mu; mup = g(mu_t-1)
    mup = [mu(1) - dt*(w(2)*(cos(mu(3))/12 + (3^(1/2)*sin(mu(3)))/12) + w(3)*(cos(mu(3))/12 -
(3^(1/2)*sin(mu(3)))/12) - (w(1)*cos(mu(3)))/6);
           mu(2) - dt*(w(2)*(sin(mu(3))/12 - (3^(1/2)*cos(mu(3)))/12) + w(3)*(sin(mu(3))/12 +
(3^(1/2)*cos(mu(3)))/12) - (w(1)*sin(mu(3)))/6);
           mu(3) - dt*((5*w(1))/18 + (5*w(2))/18 + (5*w(3))/18)];

    Sp = Ad*S*Ad' + R;

    Ht = [mup(1) 0 0; 0 -mup(2) 0; 0 0 (mup(3) - (9.7*pi/180))];
    %Ht = [(mup(1))/(sqrt(mup(1)^2 + mup(3)^2)) 0 (mup(3))/(sqrt(mup(1)^2 + mup(3)^2))];
    if(GPS_correct)
        Q_Use = Q2;
    else
        Q_Use = Q;
    end

    % Measurement update
    K = Sp*Ht'*inv(Ht*Sp*Ht'+Q_Use);
    mu = mup + K*(y(:,t)-(B*mup + DEC));
    %mu = mup + K*(y(:,t)-sqrt(mup(1)^2 + mup(3)^2));
    S = (eye(n)-K*Ht)*Sp;

    % Store results
    mup_S(:,t) = mup;
    mu_S(:,t) = mu;
    %K_S(:,t) = K;


    %% Plot results
    figure(1);clf; hold on;
    %plot(0,0,'bx', 'MarkerSize', 6, 'LineWidth', 2)
    %plot([20 -1],[0 0],'b--')
    plot(x(1,2:t),x(2,2:t), 'ro--')
    plot(mu_S(1,2:t),mu_S(2,2:t), 'bx--')
    mu_pos = [mu(1) mu(2)];
    S_pos = [S(1,1) S(1,3); S(3,1) S(3,3)];
    error_ellipse(S_pos,mu_pos,0.75);
    error_ellipse(S_pos,mu_pos,0.95);
    title('True state and belief')
    xlabel('x (m)');
    ylabel('y (m)');
    axis([-1 20 -1 10])
    axis equal

    %F = getframe;
    % Dumb hack to get desired framerate
```

```
    %for dumb=1:floor(10/truefps)
     %   writeVideo(videoobj, F);
    %end
end
    hold off
    figure(2);
    %plot(0,0,'bx', 'MarkerSize', 6, 'LineWidth', 2)
    %plot([20 -1],[0 0],'b--')
    plot(T,x(1,1:t), 'ro--')
    hold on;
    plot(T,mu_S(1,1:t), 'bx--')
    title('True state and belief - X')
    xlabel('t (s)');
    ylabel('x (m)');
    hold off

    figure(3);
    %plot(0,0,'bx', 'MarkerSize', 6, 'LineWidth', 2)
    %plot([20 -1],[0 0],'b--')
    plot(T,x(2,1:t), 'ro--')
    hold on;
    plot(T,mu_S(2,1:t), 'bx--')
    title('True state and belief - Y')
    xlabel('t (s)');
    ylabel('y (m)');
    hold off

    figure(4);
    %plot(0,0,'bx', 'MarkerSize', 6, 'LineWidth', 2)
    %plot([20 -1],[0 0],'b--')
    plot(T,x(3,1:t), 'ro--')
    hold on;
    plot(T,mu_S(3,1:t), 'bx--')
    title('True state and belief - Theta')
    xlabel('t (s)');
    ylabel('theta (rads)');
%close(videoobj);
```

Code to For Motion Model Simplification, Determining Inputs for Simulation:

```
clc
close all
clear all

r = 0.25;
l = 0.3;

al1 = pi/2;
al2 = 7/6 * pi;
al3 = -pi / 6;
J1 = [sin(al1) 0 -l; sin(al2) -cos(al2) -l; sin(al3) -cos(al3) -l];
J1 = inv(J1);
J1(2,1) = 0;
diag_vals = [r r r];
J2 = diag(diag_vals);
B = J1*J2;


w_fix = 1.0;

syms x y theta w1 w2 w3 dt

R = theta;
a = 1

xdot_circle = [-R*w_fix*sin(theta); R*w_fix*cos(theta); w_fix];
xdot_spiral = [a*(w_fix*cos(theta) + theta * w_fix * cos(theta) - theta*sin(theta));
a*(w_fix*sin(theta) + theta*w_fix*sin(theta) + theta*cos(theta)); w_fix];
Rot = [cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0; 0 0 1];
omega = [w1; w2; w3];
X_prev = [1 0 0; 0 1 0; 0 0 1] * [x; y; theta];

Motion_Model = X_prev + (Rot*B*omega)*dt;


u = inv(Rot*B)*xdot_spiral;
u = eval(simplify(u));

%format shortg;
 Motion_Model = eval(simplify(Motion_Model));
```

References

(1) http://www.cs.cmu.edu/~rasc/Download/AMRobots3.pdf