Chanheum Cho 20378843

Sean Po 20387579

ME597 Homework Assignment #1

10/26/2014

Question 1:

In order to derive the motion model for the Swedish wheel vehicle, we must first calculate the contribution of each wheel to the robot's x and y velocities. This was done by finding the magnitude contribution of each wheel, then the direction contribution of each wheel.

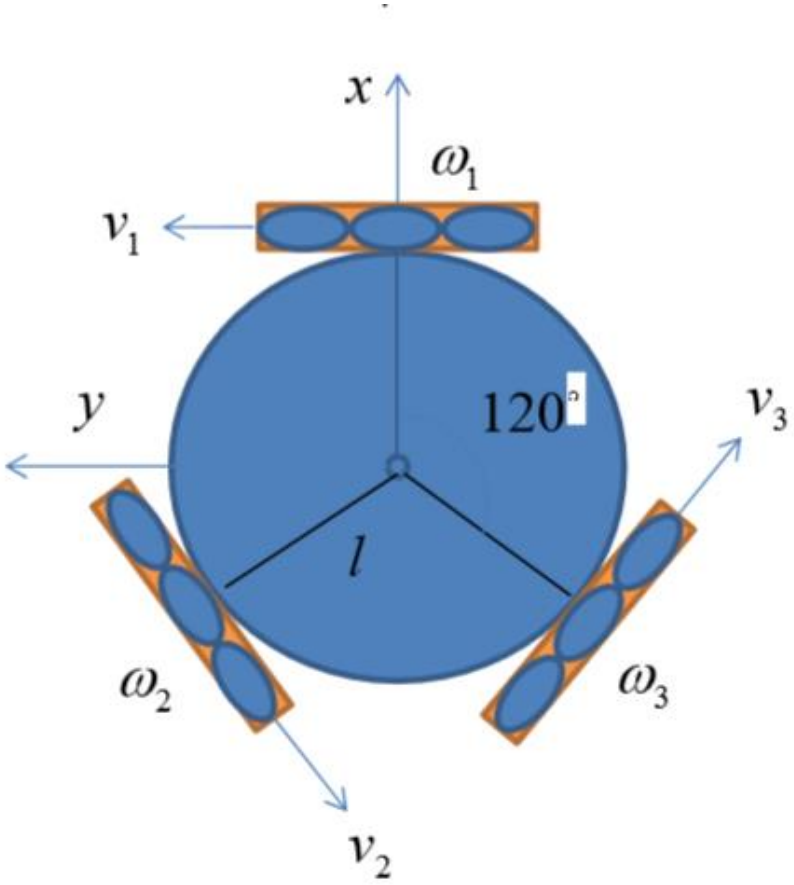


Figure 1: Robot in its local frame of reference.

Using figure 1, it can be seen that the velocity in the y direction is solely dependent on w1 when the instantaneous center of rotation is set on the cross section of the extended motions of w2 and w3 as shown in Figure 2 below. This is used to determine the contribution of each wheel to the speed of the entire robot.

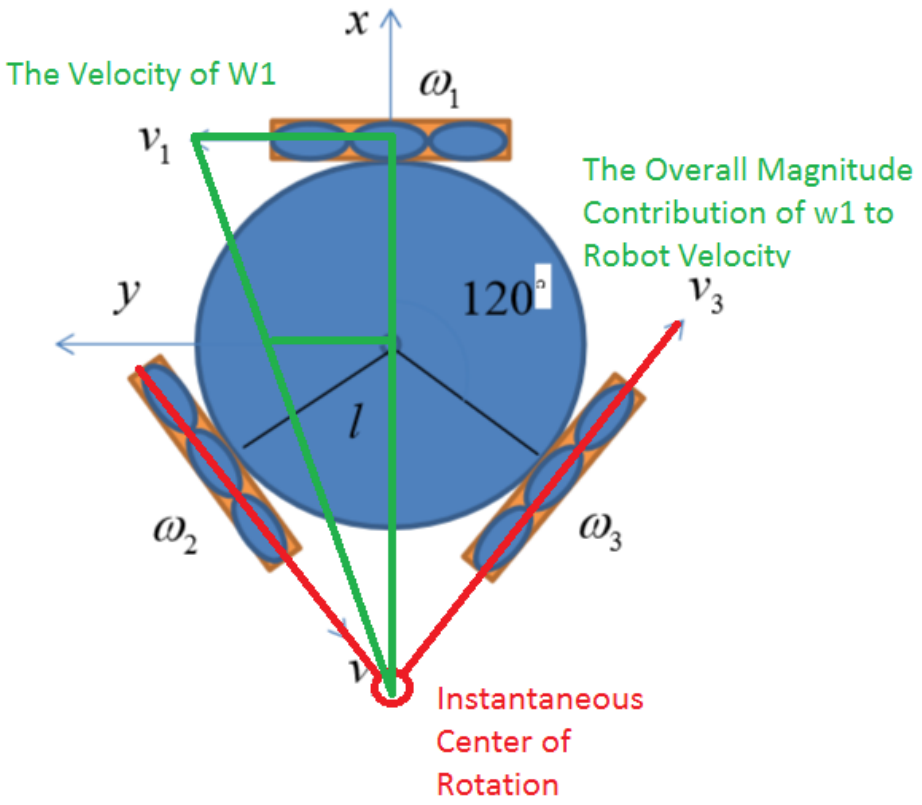


Figure 2: Robot in its local frame of reference with Instantaneous center or rotation located at the meeting point of motions w1 and w2

Using the above method, it was found that the magnitude contribution of each wheel is to be taken by a factor of 2/3.

Next, what was needed was to calculate the direction contribution of each wheel to the entire robot. It was found that w1 has no contribution to the x velocity of the robot, while w2 and w3 have a sin(30) contribution. It was also found that w1 has full contribution to the y velocity of the robot, while w2 and w3 have a cos(30) contribution.

Lastly, what was needed is to calculate the contribution of each wheel to the orientation of the robot. this was done by taking the angular velocity of each wheel, multiplying it by the radius of each wheel to find the linear velocity. You can translate the linear velocity to the angular velocity of the robot by dividing it with the radius of the entire robot. The orientation was the average of the previous calculations done on each wheel.

Given the above, the following vector of interesting states can be derived:

$$\begin{bmatrix} \dot{y} \\ \dot{x} \\ \dot{\theta} \end{bmatrix}$$

The final motion model of the robot incorporating each part becomes the **Equation 1** when a time change is multiplied to the above vector. Next, a rotation matrix moving from the body frame to the global frame is applied. Lastly, error is incorporated into the motion model by summing it with the motion disturbance model.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} cos\theta_{t-1} & -sin\theta_{t-1} & 0 \\ sin\theta_{t-1} & cos\theta_{t-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -\frac{2}{3}rcos30° & \frac{2}{3}rcos30° \\ \frac{2}{3}r & -\frac{2}{3}rsin30° & -\frac{2}{3}rsin30° \\ \frac{r}{3l} & \frac{r}{3l} & \frac{r}{3l} \end{bmatrix} \begin{bmatrix} \omega_{1,t} \\ \omega_{2,t} \\ \omega_{3,t} \end{bmatrix} dt + \varepsilon_t$$

$$\varepsilon_t = \mu + E\lambda^{\frac{1}{2}} randn(n, 1), where$$

$$\varepsilon_t = \text{motion disturbance } model$$

$$\mu = average$$

$$E = eigenvectors\ of\ motion\ \ disturbance\ covariance$$

$$\lambda = eigenvalues\ of\ motion\ disturbance\ covariance$$

$$motion\ disturbance\ covariance = \begin{bmatrix} 0.01^2 & & \\ & 0.01^2 & \\ & & 0.01^2 \end{bmatrix}$$

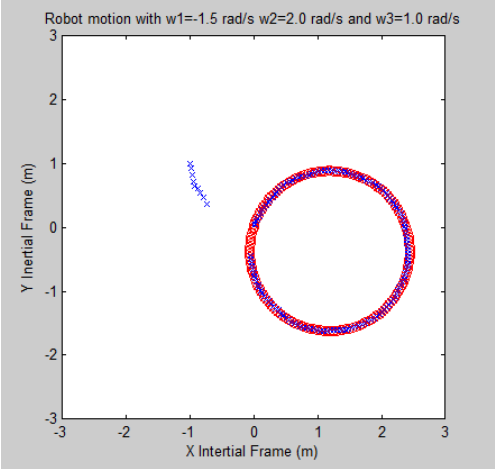
Equation 1: Motion model of the robot.

Chanheum Cho 20378843

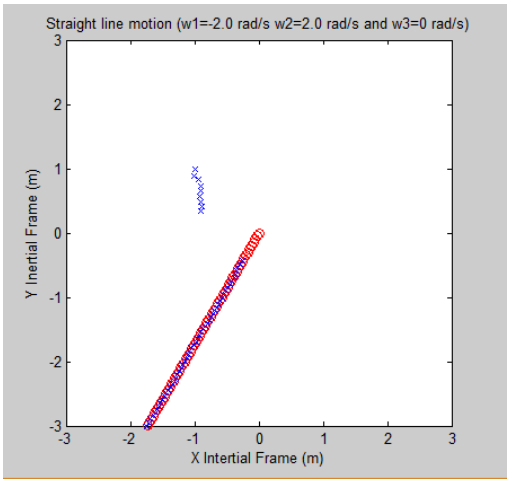
Sean Po 20387579

Question 2:

The following is an image of the simulated robot with the inputs shown in the title of the image. All images below will not include Gaussian noise in order to prove that the motion is in fact what the robot was programmed for. In **Question 5** and **Question 6**, graphs can be seen with Gaussian noise added.



In order to move the robot in a straight line, the only operation needed is to move two wheels in opposite directions. The following is the simulated model of such movement.



In order to find the velocity required to move in a full 2m radius in 15 seconds, the following was calculated:

$$\frac{2\pi r}{15} \cong 0.42 \; where \; r = 1m$$

With the speed information, the angular velocity about an arbitrary ICR can be found.

$$\dot{\theta}r = \dot{x}, where\ r = 1m$$

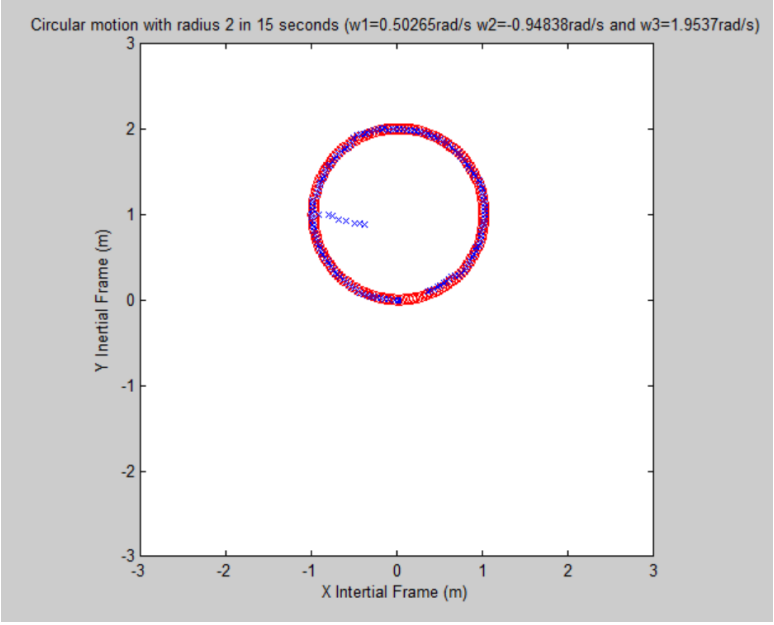
$$\dot{\theta} = \dot{x}$$

$$\frac{r}{3l}(w_1 + w_2 + w_3) = \frac{2}{3}(-\cos(30)\, w_2 + \cos(30) w_3)$$

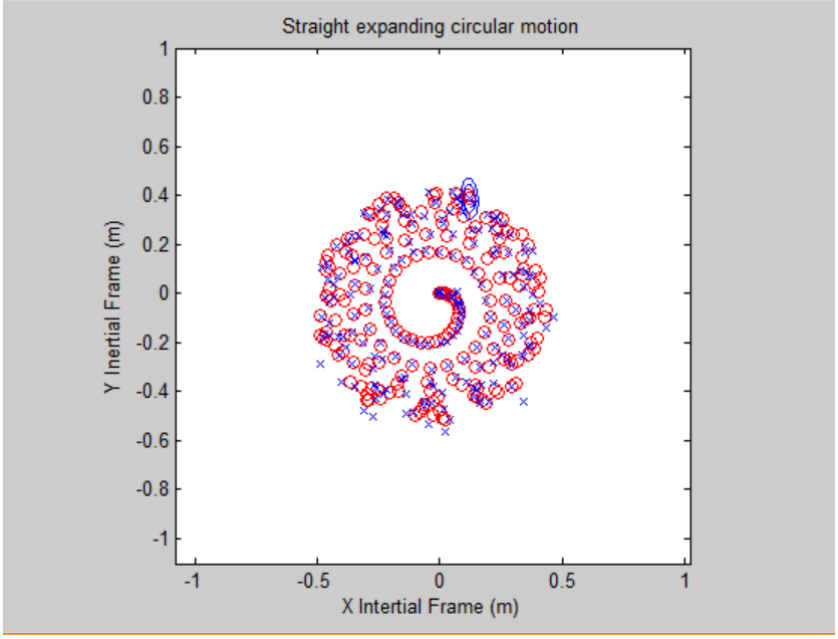
$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix} = \begin{bmatrix}\frac{2\pi}{15}\\ 0\\ \frac{2\pi}{15}\end{bmatrix} = \begin{bmatrix}0\\0\\0\end{bmatrix} + B\begin{bmatrix}\omega_{1,t}\\ \omega_{2,t}\\ \omega_{3,t}\end{bmatrix}$$

$$B = \begin{bmatrix} 0 & -\frac{2}{3}rcos30° & \frac{2}{3}rcos30° \\ \frac{2}{3}r & -\frac{2}{3}rsin30° & -\frac{2}{3}rsin30° \\ \frac{r}{3l} & \frac{r}{3l} & \frac{r}{3l} \end{bmatrix}$$

With the above, it is clear that the transpose of B can be taken, and left multiplied with $\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{\theta}\end{bmatrix}$ to get the appropriate input required to cause the robot to rotate in a circle with a diameter of 2m within 15 seconds. The following image is a result of the above calculations.



Lastly, to make the robot rotate in expanding spirals, the velocity of an arbitrary wheel is increased every sampling period. The following image is an example of the robot rotating in expanding spirals.


Straight expanding circular motion
Y Inertial Frame (m)
X Intertial Frame (m)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.5
0
0.5
1

Question 3

The measurement model can be seen as the actual location determined by the motion model with added disturbances from measurement noise, as well as the declination of 9.7 degrees West due to being in Waterloo. The measurement disturbances were calculated in a similar manner to how the motion disturbances were found in **Question 1**.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \theta_t - 9.7° \left(\frac{\pi}{180°}\right) \end{bmatrix} + \delta_t, where$$

$$\delta_t = \mu + E\lambda^{\frac{1}{2}} randn(n, 1), where$$

$$\delta_t = \text{measurement disturbance } model$$

$$\mu = average$$

$$E = eigenvectors\ of\ measurement\ disturbance\ covariance$$

$$\lambda = eigenvalues\ of\ measurement\ disturbance\ covariance$$

$$measurement\ disturbance\ covariance = \begin{bmatrix} 0.5^2 & & \\ & 0.5^2 & \\ & & 0.5^2 \end{bmatrix}$$

Chanheum Cho 20378843

Sean Po 20387579

Question 4

In order to implement an Extended Kalman filter, the predicted mean and covariance, the actual mean and covariance and the Kalman gain have to be identified for each state. For the prediction update of the Extended Kalman Filter, the following steps are taken to get the predicted mean and covariance.

$g(x_{t-1}, u_t)$ being the motion model, the predicted mean is

$$\bar{\mu} = \begin{bmatrix} \mu_{1,t-1} \\ \mu_{2,t-1} \\ \mu_{3,t-1} \end{bmatrix} + \begin{bmatrix} cos\theta_{t-1} & -sin\theta_{t-1} & 0 \\ sin\theta_{t-1} & cos\theta_{t-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -\frac{2}{3}rcos30° & \frac{2}{3}rcos30° \\ \frac{2}{3}r & -\frac{2}{3}rsin30° & -\frac{2}{3}rsin30° \\ \frac{r}{3l} & \frac{r}{3l} & \frac{r}{3l} \end{bmatrix} \begin{bmatrix} \omega_{1,t} \\ \omega_{2,t} \\ \omega_{3,t} \end{bmatrix} dt$$

And the linearizing factor G_t is identified by taking partial derivatives of $g(x_{t-1}, u_t)$ with respect to each of the state components x, y and θ.

$$G_t = \begin{bmatrix} 1 & 0 & \left(-B_{2,1}u_1 - B_{2,2}u_2 - B_{2,3}u_3\right)cos\theta_{t-1} + \left(-B_{1,2}u_2 - B_{1,3}u_3\right)sin\theta_{t-1} \\ 0 & 1 & \left(B_{1,2}u_2 + B_{1,3}u_3\right)cos\theta_{t-1} + \left(-B_{2,1}u_1 - B_{2,2}u_2 - B_{2,3}u_3\right)sin\theta_{t-1} \\ 0 & 0 & 1 \end{bmatrix}$$

Once G_t is identified, the predicted covariance is calculated using the formula $\bar{\Sigma}_t = G_t \Sigma_{t-1} {G_t}^T + R$.

The actual mean and covariance as well as the Kalman gain is obtained by solving the following equations:

$$K_t = \bar{\Sigma}_t {H_t}^T (H_t \bar{\Sigma}_t {H_t}^T + Q_t)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t(y_t - h(\bar{\mu}_t))$$

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

Where

$$H_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

After taking the partial derivative of the measurement model $h(x_t)$.

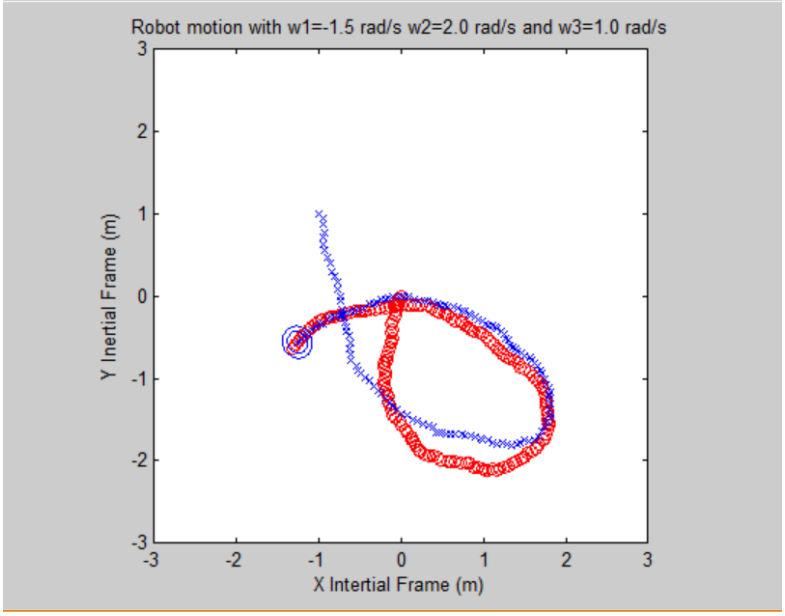
These calculations are repeated every iteration.

Chanheum Cho 20378843

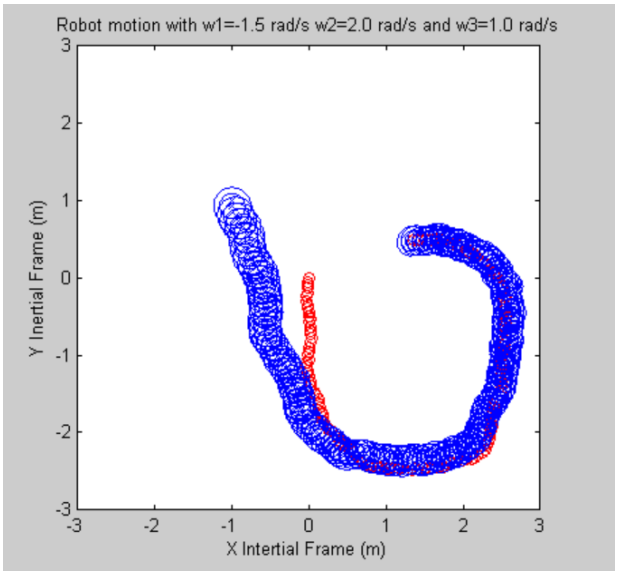
Sean Po 20387579

Question 5

The following graph is the 15 second simulation where blue crosses are the state estimate, and red is the true state.

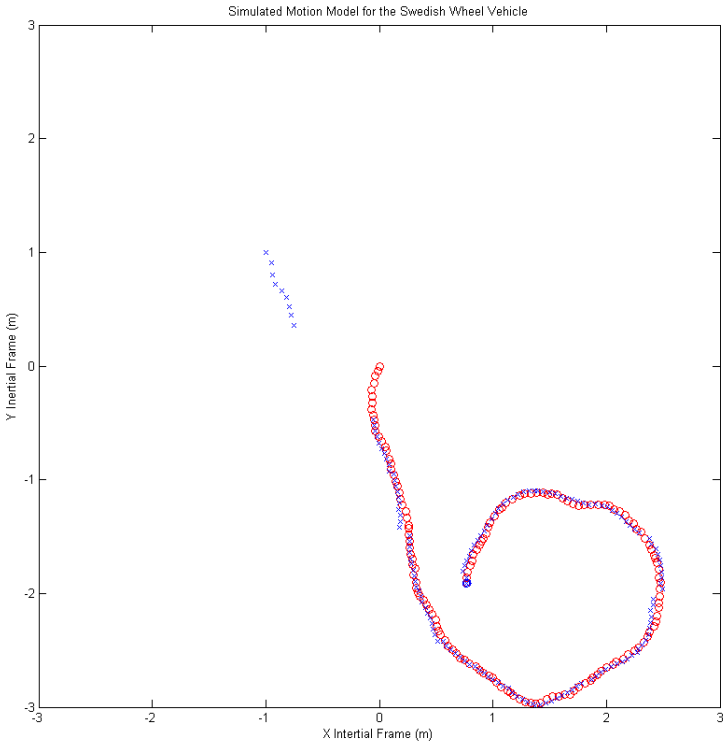


The following is the same image as above but with error ellipses more clearly defined for every state.



Question 6

The corrected GPS measurements arrive every second. This means that the measurement model includes smaller disturbances every 10 iteration. Therefore, the improved disturbance and the improved measurement model is used every second on top of updating the estimates using the regular disturbance and the regular measurement model every 0.1 seconds. The error ellipse get bigger until the robot receives the corrected GPS value which then shrinks due to the improved certainty of the robot's estimated location. The following figure shows that the improved GPS measurements correct the estimates.



Appendix: Code

Main.m

```
%% Assignment 1
clear; clc; close all;

%% Q2
    % Given inputs
    RunRobot( ...
        'Robot motion with w1=-1.5 rad/s w2=2.0 rad/s and w3=1.0 rad/s', ...
        -1.5, 2.0, 1.0, 0 ...
    )
    % Run in a straight line
    RunRobot( ...
        'Straight line motion (w1=-2.0 rad/s w2=2.0 rad/s and w3=0
rad/s)', ...
        -2.0, 2.0, 0, 0 ...
    )
    % Rotate in a perfect circle
    r = 0.25;
    l = 0.3;
    circular_motion = inv( ...
                           r*[0         -2/3*cos(pi/6)      2/3*cos(pi/6);
                           2/3        -2/3*sin(pi/6)      -2/3*sin(pi/6);
                           1/(3*l)    1/(3*l)             1/(3*l)] ...
                       ) * [2*pi/15; 0; 2*pi/15];

    RunRobot( ...
        strcat( ...
            'Circular motion with radius 2 in 15 seconds (w1=', ...
            num2str(circular_motion(1)), ...
            'rad/s w2=', num2str(circular_motion(2)), 'rad/s and w3=', ...
            num2str(circular_motion(3)), 'rad/s)' ...
        ), ...
        circular_motion(1), circular_motion(2), circular_motion(3), 0 ...
    )
    % Rotate in expanding circle
    RunRobot( ...
        'Straight expanding circular motion', ...
        -2.0, -2.0, -2.0, 0, 0, 0.1, 0, [0; 0; 0], 0.1, 20 ...
    )

%% Q5
    % Given inputs
    RunRobot( ...
        'Robot motion with w1=-1.5 rad/s w2=2.0 rad/s and w3=1.0 rad/s', ...
        -1.5, 2.0, 1.0, 1 ...
    )
```

RunRobot.m

```
function [] = RunRobot( plot_title, w1, w2, w3,
motion_disturbance_coefficient, del_w1, del_w2, del_w3, starting_robot_state,
sampling_period, sim_duration )
% Runs a robot for given wheel velocities, sampling frequency
% simulation duration, and deltas for each wheel for each iteration
%
% Usage: RunRobot( plot_title, w1, w2, w3, [motion_disturbance_coefficient,
del_w1,
% del_w2, del_w3, starting_robot_state, sampling_period, sim_duration])
%
% plot_title: Title of the plot
% w1: Wheel one input
% w2: Wheel two input
% w3: Wheel three input
% motion_disturbance_coefficient: Determines the effect of the disturbance(1)
% del_w1: Additional wheel one input per iteration (0)
% del_w2: Additional wheel two input per iteration (0)
% del_w3: Additional wheel three input per iteration (0)
% starting_robot_state: Determines the starting state of the robot ([-1; 1;
0])
% sampling_period: Determines the simulated sampling period (0.1)
% sim_duration: Determines the length of the simulation in sim time seconds.
(15)

%% Default variables
    r = 0.25;      % radius of each wheel = 0.25 m
    l = 0.3;       % radius of the robot = 0.3 m

%% Input verification
    if nargin < 4 || isempty(w1) || isempty(w2) || isempty(w3)
       error('Wheel speed parameters are required and cannot be empty.')
    end

    if nargin < 5 || isempty(motion_disturbance_coefficient)
       motion_disturbance_coefficient = 1;
    end

    if nargin < 6 || isempty(del_w1)
       del_w1 = 0;
    end

    if nargin < 7 || isempty(del_w2)
       del_w2 = 0;
    end

    if nargin < 8 || isempty(del_w3)
       del_w3 = 0;
    end

    if nargin < 9 || isempty(starting_robot_state)
       starting_robot_state = [-1; 1; 0];
    end
```

```
    if nargin < 10 || isempty(sampling_period)
       sampling_period = 0.1;
    end

    if nargin < 11 || isempty(sim_duration)
       sim_duration = 15;
    end

%% Disturbance Models

    % Motion Disturbance Model
    R = [0.01^2 0 0;
        0 0.01^2 0;
        0 0 0.1^2];
    [RE, Re] = eig(R);

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

    % Accurate Measurement Disturbance Model
    Qnew = [0.01^2 0 0;
        0 0.01^2 0;
        0 0 (10*pi/180)^2];
    [QnewE, Qnewe] = eig(Qnew);

%% Motion Model (Ax+Bu+e)
    A = eye(3);
    B = r * [0          -2/3*cos(pi/6)      2/3*cos(pi/6);
             2/3         -2/3*sin(pi/6)      -2/3*sin(pi/6);
             1/(3*l)     1/(3*l)             1/(3*l)] * sampling_period;

%% Measurement Model Matrix (Cx + d)
    C = eye(1);

%% Initialize Simulation
    t = 0:sampling_period:sim_duration;
    input = [w1; w2; w3];
    del_input = [del_w1;del_w2;del_w3];

    vehicle_state = zeros(3,length(t));   % initial vehicle state
    measurements = zeros(3,length(t));   % initial measurements

    predicted_vehicle_state = starting_robot_state;
    sigma_covariance = 0.01*eye(3);

    mu_S = zeros(3,length(t));  % Belief
    mu_S(:,1) = predicted_vehicle_state;
```

```
    figure;
%% Simulate model
    for i=2:length(t)
        clf;
        % Motion Disturbance
        motion_disturbance = Disturbance(RE, Re);
        % Transformation Matrix (from Body Frame to Inertial Frame)
        theta = vehicle_state(3,i-1);
        rotation = [cos(theta)     -sin(theta)     0;
                    sin(theta)      cos(theta)      0;
                    0               0               1];

        input = input - del_input;

        % Motion Model Xt = Xt-1 + BUt + e
        vehicle_state(:,i) = A*vehicle_state(:,i-1) + rotation*B*input +
motion_disturbance*motion_disturbance_coefficient;

        % Measurement Disturbance
        disturbance_model = Q;

        %Q6 Correction - improved Q for correction every 10 iteration
        if mod(i,10) ~= 0
            measurement_disturbance = Disturbance(QE, Qe);
        else
            measurement_disturbance = Disturbance(QnewE, Qnewe);
            disturbance_model = Qnew;
        end

        % Measurement Model Yt = CXt + d
        % the magnetometer angle has 9.7 deg offset
        measurements(:,i) = C*[vehicle_state(1,i); vehicle_state(2,i);
vehicle_state(3,i)-(9.7*pi/180)] + measurement_disturbance;

        % Extended Kalman Filter
        % Prediction update
        predicted_vehicle_state(1:3) = A*predicted_vehicle_state +
rotation*B*input;

        predicted_rotation = [-sin(theta)   -cos(theta)     0;
                               cos(theta)     -sin(theta)     0;
                               0              0               0];

        dB = (predicted_rotation*B*input);

        Gt = [1 0 dB(1); 0 1 dB(2); 0 0 1+dB(3)];

        sigma_covariance(1:3,1:3) = Gt*sigma_covariance(1:3,1:3)*Gt' + R;

        % Measurement update
        Ht = C;
```

```
        Kt = sigma_covariance*Ht'*inv((Ht*sigma_covariance*Ht' +
disturbance_model));

        predicted_vehicle_state = predicted_vehicle_state +
Kt*(measurements(:,i)-Ht*predicted_vehicle_state);
        sigma_covariance = (eye(3)-Kt*Ht)*sigma_covariance;

        % Store results
        mu_S(:,i) = predicted_vehicle_state;

        % Plot
        plot(vehicle_state(1,1:i),vehicle_state(2,1:i),'ro'); hold on;
        plot(mu_S(1,1:i),mu_S(2,1:i),'bx');
        mu_pos = [predicted_vehicle_state(1) predicted_vehicle_state(2)];
        S_pos = sigma_covariance(1:2, 1:2);
        error_ellipse(S_pos,mu_pos,0.75);
        error_ellipse(S_pos,mu_pos,0.95);
        axis equal;
        grid=3; axis([-grid grid -grid grid]);
        xlabel('X Intertial Frame (m)'); ylabel('Y Inertial Frame (m)');
        title(plot_title);
    end
end
```

Disturbance.m

```
function disturbance = Disturbance( vector, value )
    disturbance = vector*sqrt(value)*randn(3,1);
end
```