Probabilistic Robotics coursework

Probabilistic Robotics Assignment

Student Name: Luca Ricagni

Student Number: 16010178

Module code: UFMFNF-15-3

Submission Date: 23/04/20

Department of Engineering, Design, and Mathematics
University of the West of England

Contents

Task 1: Robot model

Parameter to find	Answer	File found in
Robot chassis dimensions (m) (x, y, z)	0.2mx0.24mx0.2m	robot.inc
Pose of each range finder (robot coordinate frame (x, y, z, theta))	Range finder 1: (0.1, 0, -0.15, 0) Range finder 2: (0, 0.1, -0.15, 45) Range finder 3: (0, -0.1, -0.15, -45)	probrob.world
Max. range (m) of: a) range finders b) fiducial detector	a) 2 b) 2	Sensors.inc
Field of view (degrees) of: a) range finders b) fiducial detector	a) 30 b) 180	Sensors.inc
Robot motion error parameters: a) Noise measured in rotation caused by rotation b) Noise measured in rotation caused by translation c) Noise measured in translation caused by translation d) Noise measured in translation caused by rotation e) Additional noise measured in rotation caused by translation f) Additional noise measured in rotation caused by rotation	a) alpha1 = 0.1 b) alpha2 = 0.02 c) alpha3 = 0.2 d) alpha4 = 0.01 e) alpha5 = 0.001 f) alpha6 = 0.01	robot.inc
Variance in range and bearing measurements from fiducial detector a) Range b) Bearing	a) 0.1 b) 0.01	robot.inc
Size of warehouse (m) (x,y,z)	10x5x0.5	probrob.world
Number of fiducials in the warehouse	12	robot.inc
List of coordinates of each fiducial (world coordinate frame (x, y, z, theta))	Fiducial 1: (3, 2.2, 0.3, 0) Fiducial 2: (3, 0, 0.3, 0) Fiducial 3: (3, -2.2, 0.3, 0) Fiducial 4: (1, -2.2, 0.3, 0) Fiducial 5: (-1, -2.2, 0.3, 0) Fiducial 6: (-3, -2.2, 0.3, 0) Fiducial 7: (-4.5, -2.2, 0.3, 0) Fiducial 8: (-4.5, 0, 0.3, 0) Fiducial 9: (-4.5, 2.2, 0.3, 0)	robot.inc

	Fiducial 10: (-3, 2.2, 0.3, 0) Fiducial 11: (-1, 2.2, 0.3, 0) Fiducial 12: (1, 2.2, 0.3, 0)	
Initial pose of the robot (world coordinate frame (x, y, z, theta))	(4.5, 2.2, 0, 180)	probrob.world

Task 2: Mapping

Part 1

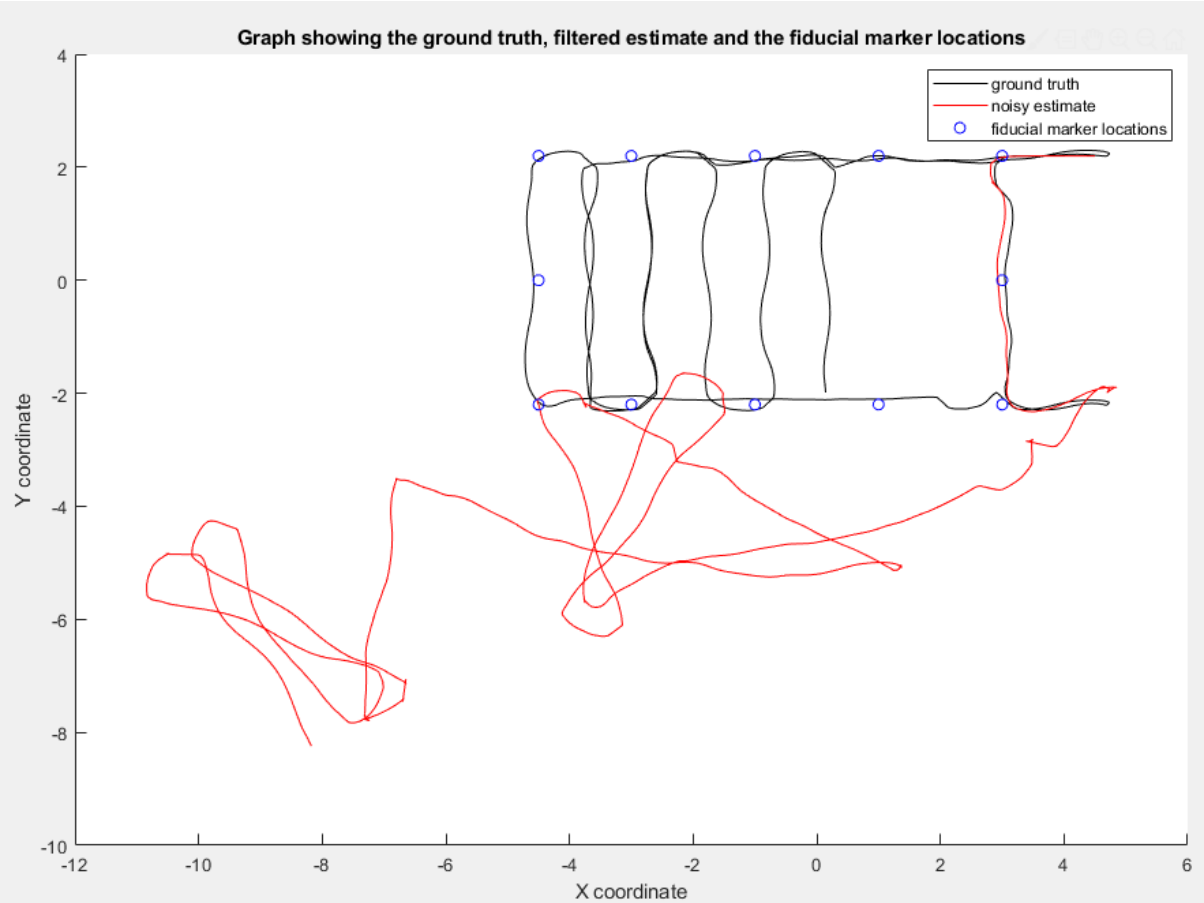

Figure 1

Figure one shows the ground truth trace in black, the noisy estimate trace in red and the fiducial marker locations in blue circles.

Part 2

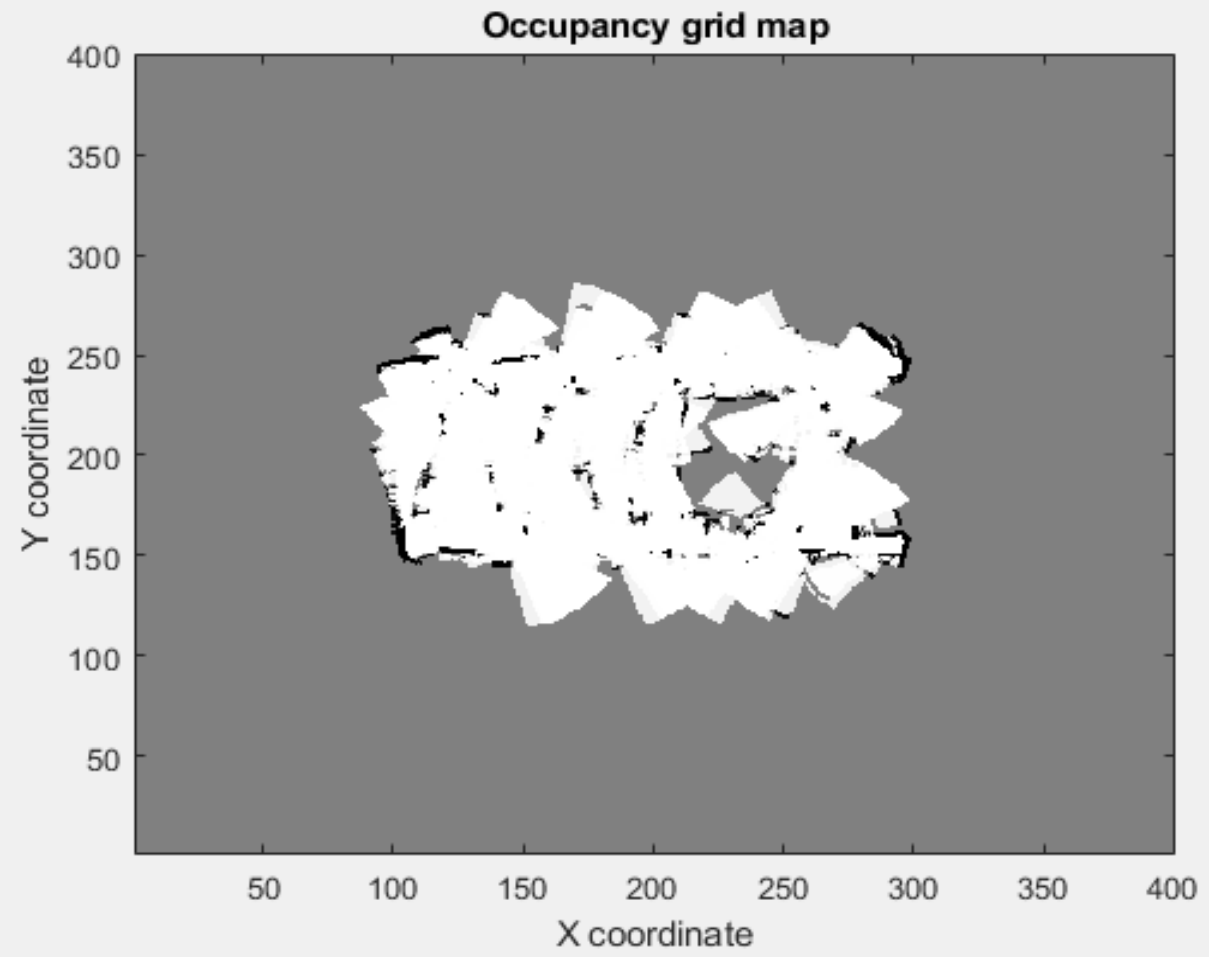

Figure 2

Figure 2 shows the occupancy grid map generated.

Task 3: Extended Kalman Filter based localisation

Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probabilistic Robotics Assignment
% Task 3: Implement an Extended Kalman Filter
% By Luca J. Ricagni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clear
close all   % Close all figures
clear       % Clear all variables in workspace
clc         % Clear command window

%% Import relevant data files

ground_truth_pose = load('ground_truth_pose.dat');      % Import ground
truth pose
noisy_pose_estimate = load('noisy_pose_estimate.dat');  % Import noisy pose
estimates
motor_commands = load('motor_commands.dat');  % Import motor commands
fiducials = load('fiducials.dat');  % Import fiducials
fiducials_noise = load('fiducials_noise.dat');  % Import noisy fiducials
ranges = load('ranges.dat');  % Import ranges

%% Declare variables
filtered_estimate = zeros(2000, 3); % Place to store filtered position
estimates

numOfIterations = 2000; % Number of times the loop will run

numOfFiducials = 12;

initial_estimate = ground_truth_pose(1,:);

% mew_t = mean(initial_estimate);
mew_t = initial_estimate;

if mew_t(3) > pi || mew_t(3) < -pi
    mew_t(3) = wrapToPi(mew_t(3));
end

sigma_t = zeros(3); % initialise value for sigma

 %% Declare map (m) which is list of fiducials
m = [3 2.2;         % Fuducial 1
     3 0;           % Fuducial 2
     3 -2.2;        % Fuducial 3
     1 -2.2;        % Fuducial 4
     -1 -2.2;       % Fuducial 5
     -3 -2.2;       % Fuducial 6
     -4.5 -2.2;     % Fuducial 7
     -4.5 0;        % Fuducial 8
     -4.5 2.2;      % Fuducial 9
     -3 2.2;        % Fuducial 10
     -1 2.2;        % Fuducial 11
     1 2.2];        % Fuducial 12
```

```
%% For each time t
mult = 0;
for i=1:numOfIterations
    observed_fiducials = zeros(numOfFiducials,3);
    c = zeros(numOfFiducials,1);

    % Extract the 12 fiducial values for a given time t
    for j=1:numOfFiducials
        observed_fiducials(j,:) = fiducials(mult*12 + j,:);
        c(j,1) = j;
    end

    mult = mult + 1; % increment mult

    if i==1
        [mew_t, sigma_t, p_zt] = extended_kalman_filter(mew_t, sigma_t,
motor_commands(i,:), observed_fiducials, c, m);
    else
        [mew_t, sigma_t, p_zt] = extended_kalman_filter(mew_t, sigma_t,
motor_commands(i,:), observed_fiducials, c, m);
    end

    % store the filtered estimate for plotting
    filtered_estimate(i,:) = mew_t;
end

%% Plot
figure(1)
hold on

plot(filtered_estimate(:,1), filtered_estimate(:,2), 'b')            % Plot
filtered estimate
plot(ground_truth_pose(:,1), ground_truth_pose(:,2), 'k')       % Plot
ground truth pose in black
plot(noisy_pose_estimate(:,1), noisy_pose_estimate(:,2), 'r')   % Plot
noisy pose estimate in red

title('Graph showing the filtered estimate, ground truth and the noisy
estimate') % Add title to graph
legend('filtered estimate', 'ground truth', 'noisy estimate') % Add key to
graph

xlabel('X coordinate') % Add label to x axis
ylabel('Y coordinate') % Add label to y axis


%% Calculate L2Norm distances
% Initialise arrays to store calculated values
l2Norm_ground_truth_noisy_estimate = zeros(numOfIterations,1);
l2Norm_ground_truth_filtered_estimate = zeros(numOfIterations,1);

% loop to iterate through and calculate the L2Norm values for each time
step
for p=1:numOfIterations
    l2Norm_ground_truth_noisy_estimate(p,1) =
sqrt((noisy_pose_estimate(p,1) - (ground_truth_pose(p,1)))^2 +
(noisy_pose_estimate(p,2) - (ground_truth_pose(p,2)))^2);
```

```
    l2Norm_ground_truth_filtered_estimate(p,1) =
sqrt((filtered_estimate(p,1) - (ground_truth_pose(p,1)))^2 +
(filtered_estimate(p,2) - (ground_truth_pose(p,2)))^2);
end

%% Plot L2Norm distances
figure(2)
hold on
plot(l2Norm_ground_truth_noisy_estimate, 'k')

plot(l2Norm_ground_truth_filtered_estimate, 'b')

title('Graph showing the L2Norm distance between ground truth and noisy
pose, with ground truth and filtered estimate') % Add title to graph
legend('L2Norm: ground truth - noisy estimate', 'L2Norm: ground truth -
filtered estimate') % Add key to graph

xlabel('Time step (t)') % Add label to x axis
ylabel('L2Norm distance') % Add label to y axis

////////////////////////////////////////////////////////////////////////////

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probabilistic Robotics Assignment
% Task 3: Extended Kalman Filter algorithm
% Page 204 Probabilistic Robotics By S. Thrun
% By Luca J. Ricagni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:
% mew_t_previous - mean guassian estimate of robot pose
% sigma_t_previous - co-variance of robot pose
% u_t - control/motor commands
% z_t - set of features visible, a 12x3 matrix as the number of fiducials
in the map is 12 therefor many will be 0, each row is a
(range,bearing,signature) of a fiducial
% c_t - correspondence variables
% m - map
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outputs:
% mew_t - new estimate of mean of robot pose
% sigma_t- new estimate of co-variance of robot pose
% p_zt - likelihood of the feature observation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [mew_t, sigma_t, p_zt] = extended_kalman_filter(mew_t_previous,
sigma_t_previous, u_t, z_t, c_t, m)
%% Declare and extract variables
delta_t = 0.1; % Declare time interval change

alpha1 = 0.1;       % noise measured in rotation caused by rotation
alpha2 = 0.02;      % noise measured in rotation caused by translation
alpha3 = 0.2;       % noise measured in translation caused by translation
alpha4 = 0.01;      % noise measured in translation caused by rotation

theta = mew_t_previous(3); % Extract the angle from the pose

v_t = u_t(1); % Extract linear velocity from motor command

w_t = u_t(2); % Extract angular velocity from motor command
```

```
if w_t == 0
    w_t = (rand-0.5)*0.0001; % add noise to w if w == 0
end

I = eye(3); % Declare identity matrix

%% Compute Jacobians needed for linearised motion model
% G_t is the derivative of the function g with respect to x_t-1 evaluated
at u_t and mew_t-1
G_t = [1 0 (-(v_t/w_t)*cos(theta) + (v_t/w_t)*cos(theta + w_t*delta_t))
       0 1 (-(v_t/w_t)*sin(theta) + (v_t/w_t)*sin(theta + w_t*delta_t))
       0 0 1];


% V_t is the derivative of the function g with respect to the motion
parameters evaluated at u_t and mew_t-1
V_t = [((-sin(theta) + sin(theta + w_t*delta_t))/w_t) (((v_t*(sin(theta) -
sin(theta + w_t*delta_t)))/w_t^2) + (v_t*(cos(theta +
w_t*delta_t))*delta_t)/w_t)
       ((cos(theta) - cos(theta + w_t*delta_t))/w_t) (-(v_t*(cos(theta) -
cos(theta + w_t*delta_t))/w_t^2) + ((v_t*(sin(theta +
w_t*delta_t))*delta_t)/w_t))
       0
delta_t];


%% Determine the motion noise covariance matrix
%M_t is the covariance matrix of the noise in control space
M_t = [(alpha1*(v_t^2) + alpha2*(w_t^2)) 0
       0 (alpha3*(v_t^2) + alpha4*(w_t^2))];

%% Calculate motion update
% Calculate new estimate of pose based off previous estimate and the new
motor commands v_t and w_t
mew_hat_t = mew_t_previous + [(-(v_t/w_t)*sin(theta) + (v_t/w_t)*sin(theta
+ w_t*delta_t)) ((v_t/w_t)*cos(theta) - (v_t/w_t)*cos(theta + w_t*delta_t))
w_t*delta_t];

if mew_hat_t(3) > pi || mew_hat_t(3) < -pi
    mew_hat_t(3) = wrapToPi(mew_hat_t(3));
end

% Calculate new estimate of covariance of position
% V_t*M_t*(V_t)^transpose provides the approximate mapping between motion
noise in comtrol space and motion noise in state space
sigma_hat_t = G_t*sigma_t_previous*G_t.' + V_t*M_t*(V_t');

%% Measurement update (correction step)
Q_t = [0.01^2 0 0
       0 0.1^2 0
       0 0 0];

% loop to iterate through all observed fiducials
for i=1:12
    % this if statement skips the loop iteration when no fiducial is
observed
    % this is because z_t is a 12x3 matrix, one row for each landmark in
the map
```

```
    if z_t(i,1) == 0
        continue;
    end

    % extract correspondence of observed fiducial
    j = c_t(i,1);

    % calculate difference between map coords of fiducial and the pose of
the robot
    q = (m(j,1) - mew_hat_t(1))^2 + (m(j,2) - mew_hat_t(2))^2;

    % z_hat_t is the predicted measurement
    z_hat_t = [sqrt(q)
               (atan2((m(j,2) - mew_hat_t(2)), (m(j,1) - mew_hat_t(1))) -
mew_hat_t(3))
               0];

    % H_t is the jacobian of the measurement model
    % the last row of H_t are all 0 because the correct correspondence of
each fiducial is known
    H_t = [-((m(j,1) - mew_hat_t(1))/sqrt(q)) -((m(j,2) -
mew_hat_t(2))/sqrt(q)) 0
           ((m(j,2) - mew_hat_t(2))/q) -((m(j,1) - mew_hat_t(1))/q) -1
           0                                                  0
0];

    % S_t is the overall measurement prediction uncertainty
    % the robot location uncertainty is mapped into observation uncertainty
by multiplying by H_t
    S_t = H_t*sigma_hat_t*H_t.' + Q_t;

    % compute the inverse of the matrix S_t
    % pinv is used to compute the pseudoinverse as the determinant is often
0 and so inverse does not exist
    S_t_inverse = pinv(S_t);

    % K_t is the kalman gain matrix
    % the offset between the predicted and observed measurement is mapped
into state space
    % used to move the location estimate in the direction that would reduce
the measurement innovation
    % the more certain the observation the higher the kalman gain
    K_t = sigma_hat_t*H_t.'*S_t_inverse;

    % mew_hat_t is the updated position estimate
    % the kalman gain K_t is used to map the difference between predicted
and observed measurement into state space
    mew_hat_t = mew_hat_t + (K_t*(z_t(i,:).' - z_hat_t)).';

    % sigma_hat_t is the uncertainty elipse of the location estimate
    sigma_hat_t = (I - K_t*H_t)*sigma_hat_t;
end

% mew_t is updated with the best estimate for output
mew_t = mew_hat_t;

% sigma_t is updated with the best estimate for output
sigma_t = sigma_hat_t;
```

```
% p_zt = det((2*pi*S_t)^-0.5) * exp(-0.5*(z_t - z_hat_t).'*(S_t^-1)*(z_t -
z_hat_t));
p_zt=0; % value is not necessary for filter to work, included for
completeness of algorithm
end
```

Figures

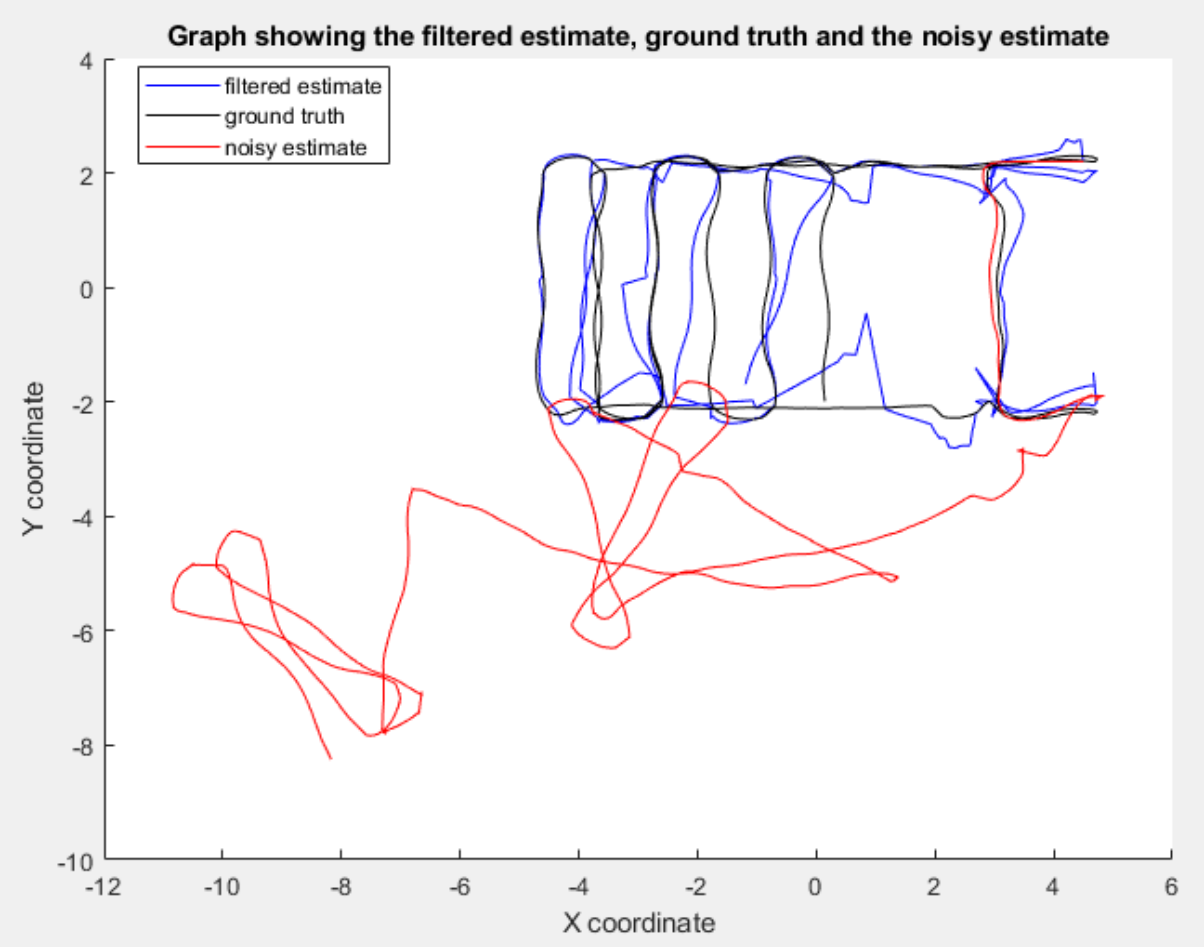

Figure 3

Figure 3 shows the pose estimate generated by the Kalman filter as the blue trace, the ground truth trace in black and the noisy estimate trace in red. As can be seen the filtered pose is very similar to the ground truth pose and as such the filter is accurate.

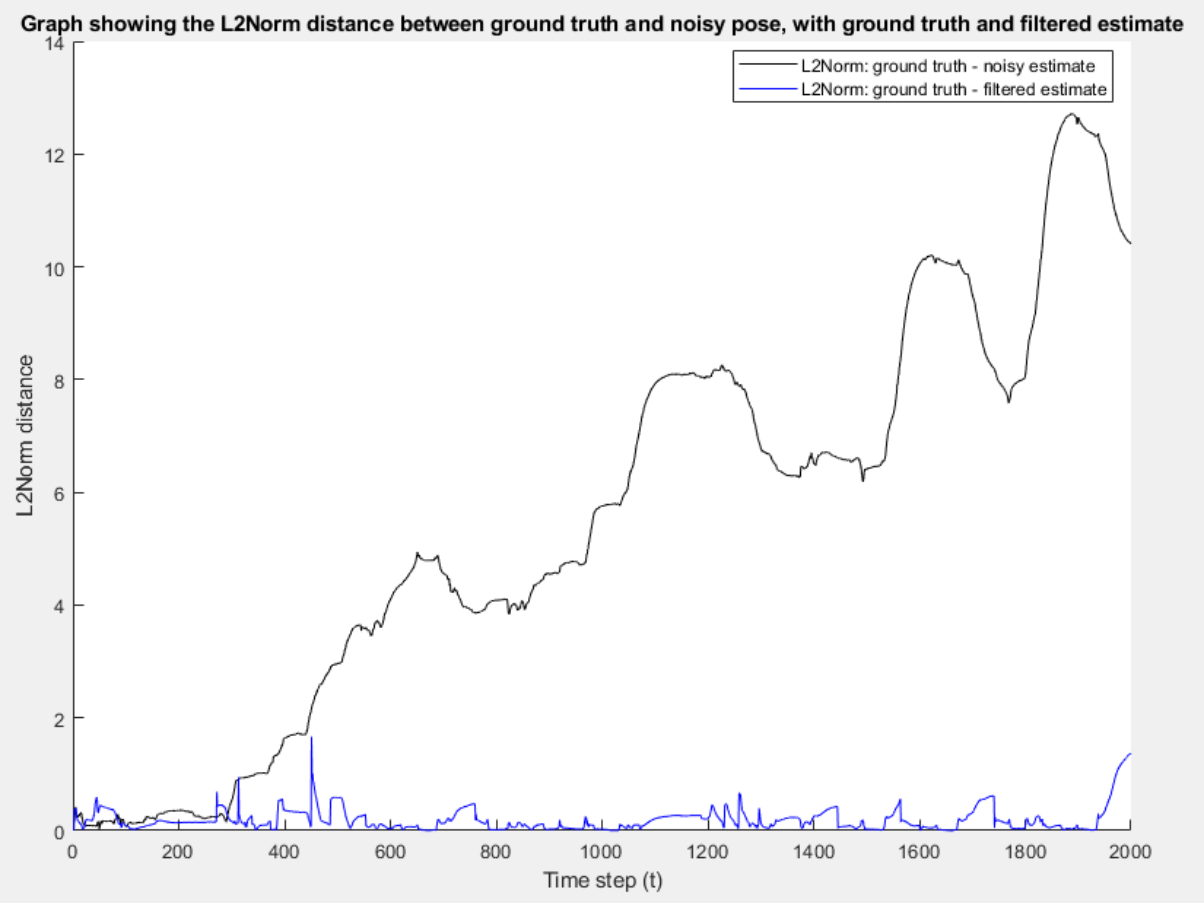

Figure 4

Figure 4 shows the Euclidean distance between the ground truth and the noisy estimate in black, and the Euclidean distance between the ground truth and the filtered estimate in blue. The closer the blue trace is to 0 the more accurate the filter is, as such this filter is very accurate.

Task 4: Particle Filter based localisation

Code

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probabilistic Robotics Assignment
% Task 4: Implement a Particle Filter
% By Luca J. Ricagni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Clear
close all   % Close all figures
clear       % Clear all variables in workspace
clc         % Clear command window

%% Import relevant data files

ground_truth_pose = load('ground_truth_pose.dat');      % Import ground
truth pose
noisy_pose_estimate = load('noisy_pose_estimate.dat');  % Import noisy pose
estimates
motor_commands = load('motor_commands.dat');  % Import motor commands
fiducials = load('fiducials.dat');  % Import fiducials
fiducials_noise = load('fiducials_noise.dat');  % Import noisy fiducials
ranges = load('ranges.dat');  % Import ranges

%% Declare variables
filtered_estimate = zeros(2000, 4); % Place to store filtered position
estimates

numOfIterations = 2000; % Number of times the loop will run

numOfFiducials = 12;

M = 1000; % Number of particles

X_t = zeros(M, 4); % Initialise particle storage

% Initialise particles to starting groung truth pose
for i=1:M
    X_t(i,1:3) = ground_truth_pose(1,:);
end

%% Declare map (m) which is list of fiducials
map = [3 2.2;         % Fuducial 1
    3 0;           % Fuducial 2
    3 -2.2;        % Fuducial 3
    1 -2.2;        % Fuducial 4
    -1 -2.2;       % Fuducial 5
    -3 -2.2;       % Fuducial 6
    -4.5 -2.2;     % Fuducial 7
    -4.5 0;        % Fuducial 8
    -4.5 2.2;      % Fuducial 9
    -3 2.2;        % Fuducial 10
    -1 2.2;        % Fuducial 11
    1 2.2];        % Fuducial 12

mult = 0;
for i=1:numOfIterations
```

```
    % Initialise array to hold fiducials for time t
    observed_fiducials = zeros(numOfFiducials,3);

    % Extract the 12 fiducial values for a given time t
    for j=1:numOfFiducials
        observed_fiducials(j,:) = fiducials(mult*12 + j,:);
    end

    mult = mult + 1;

    % Call monte_carlo_localization (particle filter)
    X_t = monte_carlo_localization(X_t, motor_commands(i,:),
observed_fiducials, map, M);

    %% Take mean (x,y) of particles
    x_mean = mean(X_t(:,1));
    y_mean = mean(X_t(:,2));

    filtered_estimate(i,1) = x_mean;
    filtered_estimate(i,2) = y_mean;
end

%% Plot
figure(1)
hold on

plot(filtered_estimate(:,1), filtered_estimate(:,2), 'b')       % Plot
filtered estimate
plot(ground_truth_pose(:,1), ground_truth_pose(:,2), 'k')       % Plot
ground truth pose in black
plot(noisy_pose_estimate(:,1), noisy_pose_estimate(:,2), 'r')   % Plot
noisy pose estimate in red

title('Graph showing the filtered estimate, ground truth and the noisy
estimate') % Add title to graph
legend('filtered estimate', 'ground truth', 'noisy estimate') % Add key to
graph

xlabel('X coordinate') % Add label to x axis
ylabel('Y coordinate') % Add label to y axis


%% Calculate L2Norm distances
% Initialise arrays to store calculated values
l2Norm_ground_truth_noisy_estimate = zeros(numOfIterations,1);
l2Norm_ground_truth_filtered_estimate = zeros(numOfIterations,1);

% loop to iterate through and calculate the L2Norm values for each time
step
for p=1:numOfIterations
    l2Norm_ground_truth_noisy_estimate(p,1) =
sqrt((noisy_pose_estimate(p,1) - (ground_truth_pose(p,1)))^2 +
(noisy_pose_estimate(p,2) - (ground_truth_pose(p,2)))^2);

    l2Norm_ground_truth_filtered_estimate(p,1) =
sqrt((filtered_estimate(p,1) - (ground_truth_pose(p,1)))^2 +
(filtered_estimate(p,2) - (ground_truth_pose(p,2)))^2);
end
```

```
%% Plot L2Norm distances
figure(2)
hold on
plot(l2Norm_ground_truth_noisy_estimate, 'k')

plot(l2Norm_ground_truth_filtered_estimate, 'b')

title('Graph showing the L2Norm distance between ground truth and noisy
pose, with ground truth and filtered estimate') % Add title to graph
legend('L2Norm: ground truth - noisy estimate', 'L2Norm: ground truth -
filtered estimate') % Add key to graph

xlabel('Time step (t)') % Add label to x axis
ylabel('L2Norm distance') % Add label to y axis
```

//

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probabilistic Robotics Assignment
% Task 4: Implement a Particle Filter
% Page 252 Probabilistic Robotics By S. Thrun
% By Luca J. Ricagni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:
% X_t_previous - set of previous particles
% u_t - control/motor commands
% z_t - set of features visible, a 12x3 matrix as the number of fiducials
in the map is 12 therefor many will be 0, each row is a
(range,bearing,signature) of a fiducial% m - map
% M - number of particles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outputs:
% X_t - new set of particles
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [X_t] = monte_carlo_localization(X_t_previous, u_t, z_t, map, M)

delta_t = 0.1; % Declare time step

% Inilialise variables
X_t = zeros(M, 4);
X_t_hat = X_t;

w_t = zeros(M, 1);

%% Update particles using motion and landmark models
for m=1:M
    % Use particles from current belief and sample from the motion model
    X_t(m,1:3) = sample_velocity_motion_model(u_t, X_t_previous(m,:),
delta_t);

    % Determine importance weight of particle
    w_t(m) = landmark_model_known_correspondence(z_t, X_t(m,:), map);

    % Store particle pose and weight
    X_t_hat(m,:) = X_t_hat(m,:) + X_t(m,:);
    X_t_hat(m,4) = w_t(m);
```

```
end

%% Perform roulette wheel selection
for m = 1:1:M

    % Randomly generate selection point which is not 0
    selection_point = 0;
    while selection_point == 0
        selection_point = sum(X_t_hat(:,4)) * rand(1);
    end

    % Iterate through until sum of weights so far > selection point
    running_total = 0;
    roulette_counter = 1;
    while (running_total < selection_point)
        running_total = running_total + X_t_hat(roulette_counter,4);
        roulette_counter = roulette_counter + 1;
    end
    % Store particle
    X_t(m,:) = X_t_hat(roulette_counter-1,:);
end

end
```

//

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Algorithm sample motion model velocity, P.124 Thrun
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Martin J. Pearson, Probabilistic Robotics, UWE
%%%%%
function [x] =  sample_velocity_motion_model(u, x_1, delta_t)
%%% Arguments:
% u is a 1x2 vector of most recent motor command velocity (linear angular)
% x_1 is the previous pose estimate [x y theta]
% delta_t is the update period
%%% Return:
% x is the current pose estimate [x y theta]

alpha1 = 0.1;        % noise measured in rotation caused by rotation
alpha2 = 0.02;       % noise measured in rotation caused by translation
alpha3 = 0.2;        % noise measured in translation caused by translation
alpha4 = 0.01;       % noise measured in translation caused by rotation

alpha5 = 0.001;      % additional noise measured in rotation caused by
translation
alpha6 = 0.01;       % additional noise measured in rotation caused by
rotation

x       = zeros(3,1);        % initialise output vector
v       = u(1);              % extract linear velocity
w       = u(2);              % extract angular velocity
theta   = x_1(3);            % extract direction of robot

% Add noise taken from distribution that represents p(x|u,x_1)
v_hat       = v  + sample_norm((alpha1*v^2) + (alpha2*w^2));
w_hat       = w  + sample_norm((alpha3*v^2) + (alpha4*w^2));
gamma_hat   = sample_norm((alpha5*v^2) + (alpha6*w^2));

if(w == 0)
```

```
    % forward model of motion to find sample estimate of pose
    x(1) = x_1(1) + v_hat*delta_t*cos(theta);
    x(2) = x_1(2) + v_hat*delta_t*sin(theta);
    x(3) = wrapToPi(theta + gamma_hat*delta_t);
else
    % forward model of motion to find sample estimate of pose
    x(1) = x_1(1) - (v_hat/w_hat)*sin(theta) +
(v_hat/w_hat)*sin(theta+w_hat*delta_t);
    x(2) = x_1(2) + (v_hat/w_hat)*cos(theta) -
(v_hat/w_hat)*cos(theta+w_hat*delta_t);
    x(3) = wrapToPi(theta + w_hat*delta_t + gamma_hat*delta_t);
end

% Sample from a Gaussian (p.124 Thrun)
function [x] = sample_norm(b)

b=sqrt(b);

x = 0.5* sum((rand(1,12)*(b*2))-b);
```

//

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Probabilistic Robotics Assignment
% Task 4: Implement a Particle Filter
% Page 252 Probabilistic Robotics By S. Thrun
% By Luca J. Ricagni
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Arguments:
% X_t - set of particles
% z_t - set of features visible
% map - map of fiducials
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outputs:
% q - likelihood of feature observation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [q] = landmark_model_known_correspondence(z_t, X_t, map)

% Variables which dictate the height of the peak and the width of the
gaussian
% Best ones so far
mu = 0.608;
sigma = 0.31;

% mu = 0.15;
% sigma = 0.015;

% Generate probability distribution
pd = makedist('Normal','mu',mu,'sigma',sigma);

visibleFiducials = 0;

% For each fiducial
for c=1:12
    if z_t(c,1) == 0
        % this if statement skips the loop iteration when no fiducial is
observed
```

```
        % this is because z_t is a 12x3 matrix, one row for each landmark
in the map
        continue;
    end

    % Calculate true range of landmark
    r_hat = sqrt(((map(c,1)-X_t(1))^2) + ((map(c,2)-X_t(2))^2));

    % Calculate true bearing of landmark
    phi_hat = atan2((map(c,1)-X_t(1)), (map(c,2)-X_t(2))); % - X_t(3)

    % Probability of range and bearing, assuming independence of noise
    q = pdf(pd, (z_t(c,1) - r_hat)) * pdf(pd, (z_t(c,2) - phi_hat));

    visibleFiducials = visibleFiducials + 1;
end

% If no fiducials where visible input some noise otherwise will get stuck
% in loop in monte_carlo_localization line 45
if visibleFiducials == 0
    q = abs((rand-0.5)*0.01);
end

end
```

Figures

It should be noted that the function `sample_velocity_motion_model` was not written and is not owned by the author of this report and is only included for completeness sake as it is required for the particle filter to run.

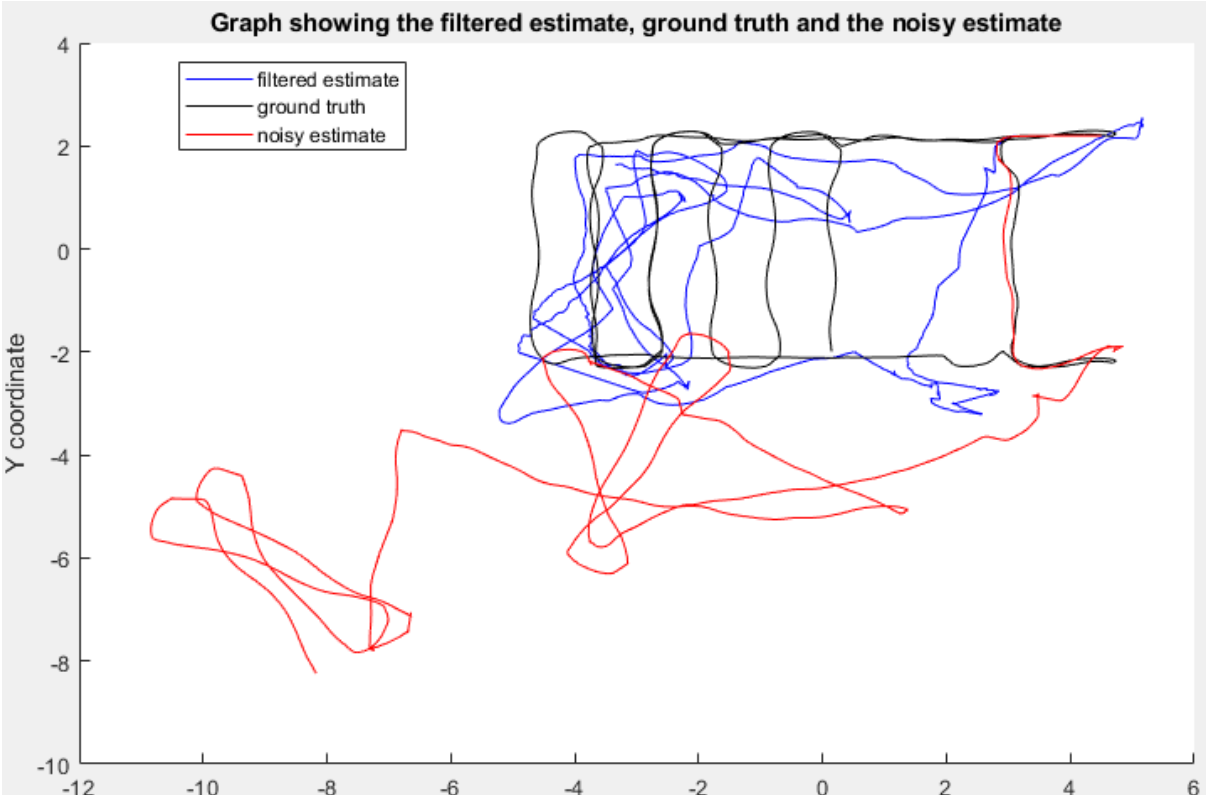

Figure 5

Figure 5 shows the pose estimate generated by the particle filter as the blue trace, the ground truth trace in black and the noisy estimate trace in red. As can be seen the filtered pose is very similar to the ground truth pose and as such the filter is accurate.

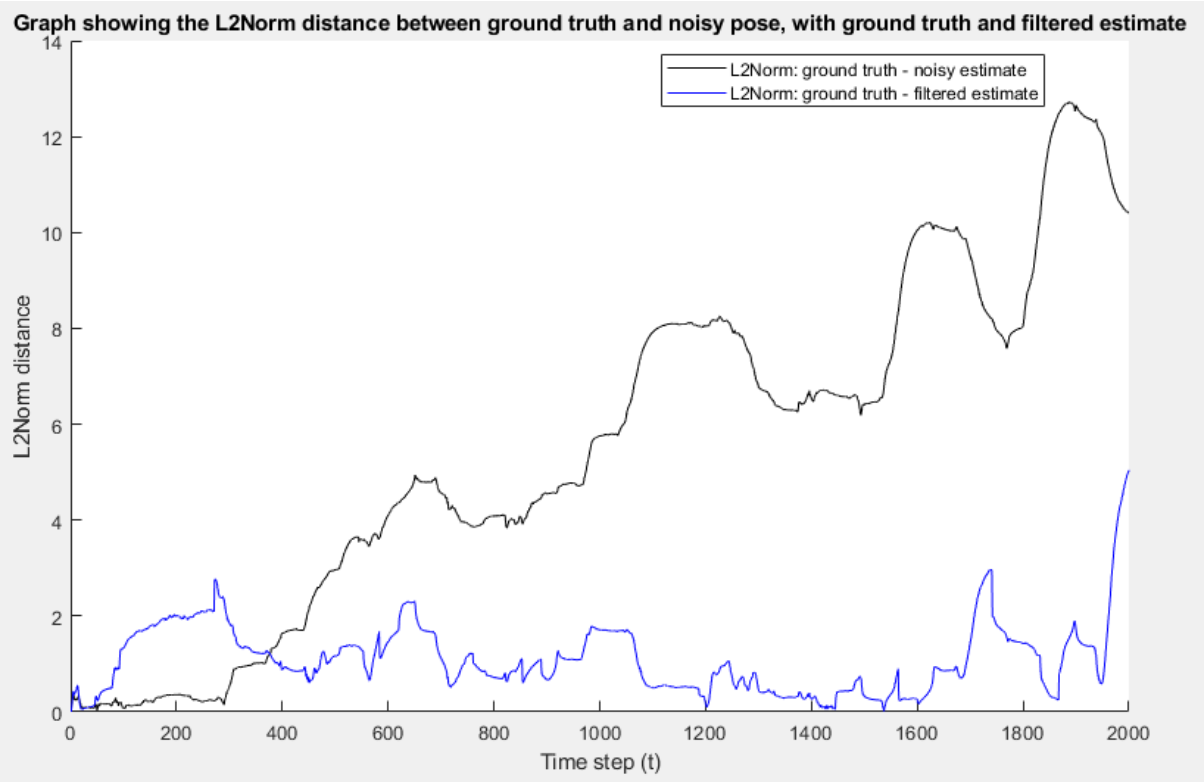

Figure 6

Figure 4 shows the Euclidean distance between the ground truth and the noisy estimate in black, and the Euclidean distance between the ground truth and the filtered estimate in blue. The closer the blue trace is to 0 the more accurate the filter is, as such this filter is reasonably accurate.