Part A

This part required an implementation of a "dead-reckoning" process, based on the kinematic model of the platform and on the measurements provided by certain onboard sensors (speed encoder and gyroscope). The kinematic model used for this part is shown below:

$$\frac{dx}{dt} = v(t)\cos\big(\theta(t)\big)$$

$$\frac{dy}{dt} = v(t)\sin(\theta(t))$$

$$\frac{d\theta}{dt} = \omega(t),$$

where $v(t)$ is the speed of the vehicle at time t, $\omega(t)$ is the yaw rate at time t, and $x(t)$, $y(t)$ and $\theta(t)$ describe the vehicle's pose and heading component.

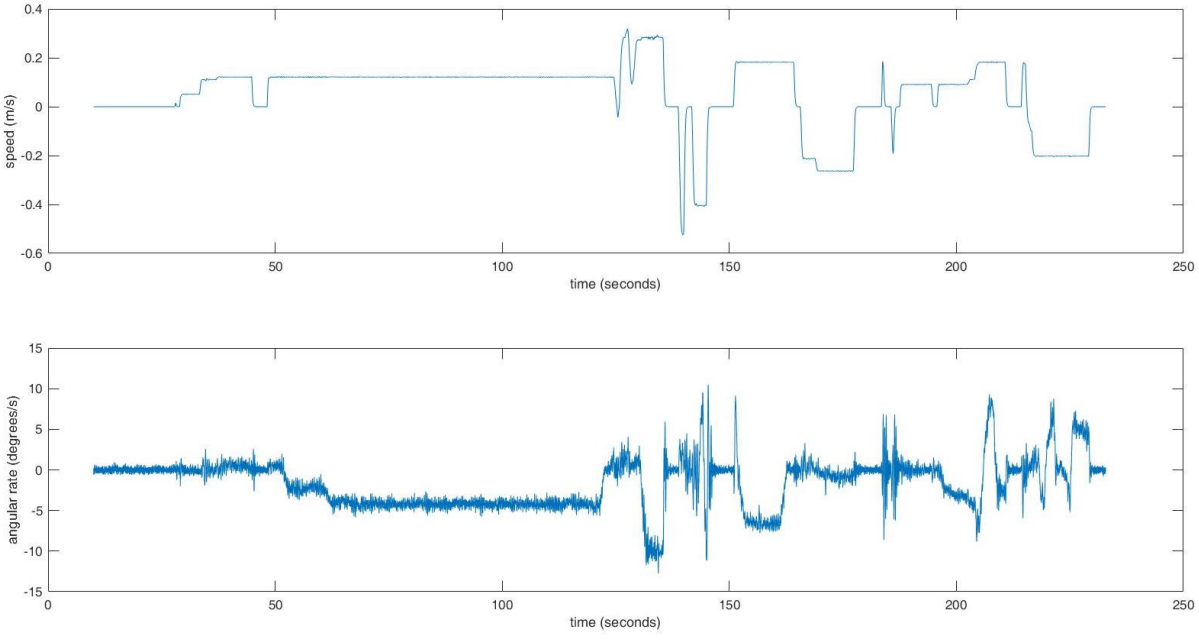
Using the equations above, Euler's Approximation is used to estimate the position and heading of the vehicle given the velocity and angular velocity. Before using Euler's Approximation, however, it should be noted that there is a certain gyro bias within the measurements of angular velocity. As the vehicle is stationary in the first few seconds, there should be no measurements recorded for both velocity and angular velocity. Thus, the angular velocity measurements recorded at these stationary time intervals can be averaged and subtracted from all data values.

Thus, using the code below, the velocity and the angular velocity can be found:

```
v = double(Data.Z(1,:))/1000;
w = double(Data.Z(2,:))/100;
gb = mean(w(1:4256)); % Gyro Bias
wgb = zeros(length(w),1);

% Gyro bias
for i=1:length(w)
    wgb(i) = w(i)-gb;
end
```

The velocity and angular velocity after accounting for gyro bias is shown in Figure 1 below:



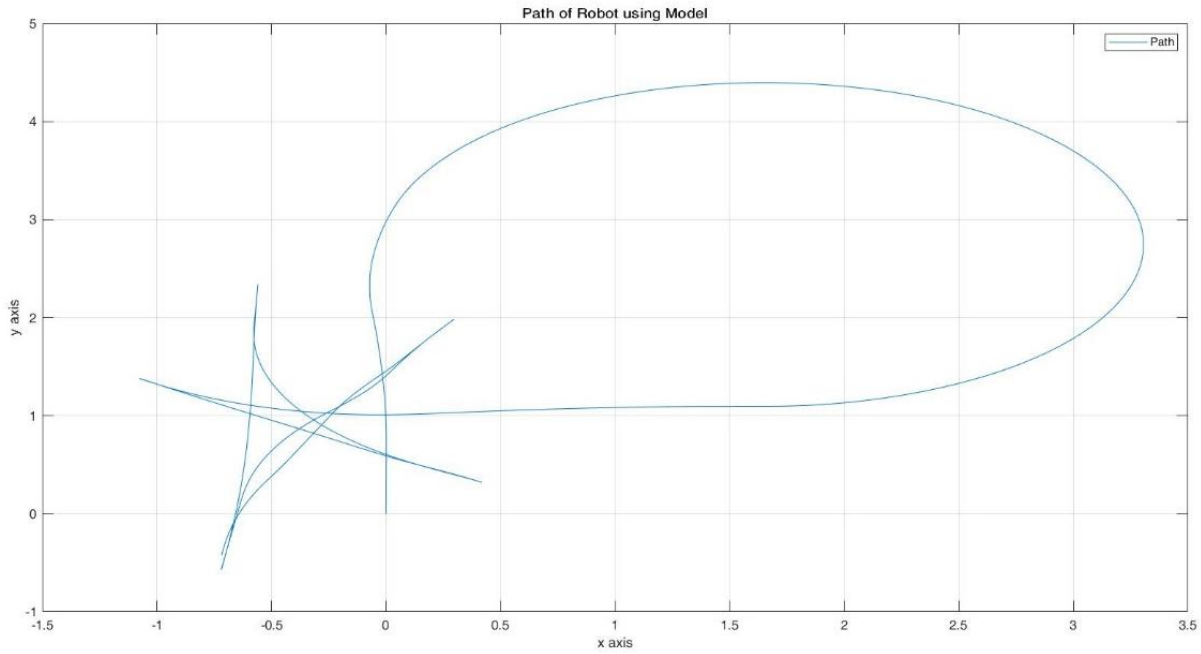
Euler's Approximation is then done using the following code:

```
X = [0;0;pi/2];  % initial pose

for k = 1:length(v)-1
    dt = t(k+1)-t(k);
    dX = [v(k)*cos(X(3,k));v(k)*sin(X(3,k));deg2rad(wgb(k))];

    % Euler's Approximation
    X(:,k+1)=X(:,k)+dt*dX;
end
```

Thus, the approximated path of the robot can be plotted, as shown in Figure 2 below:



Part B

This task covers the processing of the LiDAR native raw scans. As the LiDAR scans are given as a 13-bit number containing the polar coordinates of the scan as well as the intensity, the following function code is first used to separate the the two data sets:

```
function [r,I]=GetRangeAndIntensityFromRawScan(scan)
    r = 0.01*single(bitand(scan,8191));
    I= bitshift(scan,-13);
    % bits [0:12] are range, bits [13:15] intensity
end
```

Then, the polar coordinates had to be converted into Cartesian coordinates using the following formula:

$$x = r\cos(\theta),$$

where r is the distance from the LiDaR to the point

This is done in the following code:

```
% Obtaining Cartesian coordinates from polar
    angleScan = ((0:360)/2)';
    xcoor = r.*cos(deg2rad(angleScan));
    ycoor = r.*sin(deg2rad(angleScan));
```

Lastly, the brilliant pixels had to be located and shown in a different colour to the rest of the pixels. This was done by checking to see if the Intensity (I) value in the LiDAR scan was greater than 0, using the following code:

```
ii=find(I>0); set(hL2,'xdata',xcoor(ii),'ydata',ycoor(ii));
```

Part C

In this part, the Objects of Interest (OOIs) feature is implemented. As the brilliant pixels arise from five different reflective poles no less than 1 metre apart, the pixels need to be grouped together using a segmentation process to identify the 5 poles in the scans.

This was done by first looping through all the points and identifying the reflective points. When a reflective point is found, it is put into a cell array. The next point is then observed and if it is also reflective, then it is also put into the cell. This occurs until there is no longer a reflective point, or the next reflective point is found to be further than 1 metre away.

The following code was used to group the brilliant pixels together into a set:

```
    % Segmenting OOIs
    i=1;
    k=1;
    segment = {};
    xseg = [];
    yseg = [];

    % Finding and isolating points on same poles
    while i<length(I)
        if I(i)>0
            j=1;
            while i<length(I) && I(i)>0 && sqrt((X(1,i+1)-
X(1,i))^2+(X(2,i)-X(2,i+1))^2)<1
                temp(j) = i;
                i = i+1;
                j = j+1;
            end
            segment{k} = temp;
            temp = 0;
            k = k+1;
        end
        i = i+1;
    end
```

Then, this code was used to take the average of all the points so as to find the output, which is the centres of geometry of clusters of brilliant pixels:

```
    % Taking the average of the points
    for i=1:length(segment)
        xseg(i) = mean(xcoor(segment{i}));
        yseg(i) = mean(ycoor(segment{i}));
    end
```

The above code exploits the fact that the OOIs are relatively isolated, that it is separated from other OOIs and other opaque objects.

Part D

This part involves translating all of the data points in the local frame into the global coordinate frame. This is done by using the estimated pose and heading angle that was done in Part A. The rotation matrix is given by:

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

and the global coordinates are related to the local coordinates by the following formula:

$$R(\theta) \cdot (P_G - \boldsymbol{X}) = P_L,$$

where P_G is the contains the vector coordinates of the global frame $\begin{bmatrix} x_G \\ y_G \\ \theta_G \end{bmatrix}$, P_L contains the vector coordinates of the local frame $\begin{bmatrix} x_L \\ y_L \\ \theta_L \end{bmatrix}$ and $\boldsymbol{X}$ is the predicted path coordinates $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ in of the robot using Euler's Approximation in Part A. As the θ component does not change due to the rotation matrix, it can be reduced to

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{bmatrix}.$$

Thus, to find the global coordinates,

$$P_G = R^{-1}(\theta) \cdot P_L + \boldsymbol{X},$$

where

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

This is shown in the code as:

```
Rthetainv = [cos(theta(pt)) -sin(theta(pt));sin(theta(pt))
cos(theta(pt))];
```

Thus, the following code is then used to find the global coordinates:

```
% Global Frame
xgf = zeros(length(I),1);
ygf = zeros(length(I),1);


% Global Coordinates
for i=1:length(I)
    Matrix = Rthetainv*[xcoor(i);ycoor(i)]+[X(1,pt);X(2,pt)];

    xgf(i) = Matrix(1);
    ygf(i) = Matrix(2);
end
```

Similarly, for the objects of interest, the following code was used:

```
% Global OOIs
xgfseg = [];
ygfseg = [];

for i=1:length(segment)
    Matrix = Rthetainv*[xseg(i);yseg(i)]+[X(1,pt);X(2,pt)];

    xgfseg(i) = Matrix(1);
    ygfseg(i) = Matrix(2);
end
```

Thus, the coordinates were changes from the local frame, where the vehicle stays stationary while the objects move, to a global frame, where the vehicle moves while the objects stay stationary.

Part E

This part contains the Data Association part of the project. A map of landmarks is provided and put into a table, and the identity of the detected OOIs need to be inferred each time a LiDAR scan is processed. This is done by expressing in the global coordinate frame the positions of the detected OOIs, and then matching those OOIs and map's landmarks, based on the distance between them.

Thus, for each pole, its length from the given map of landmarks is measure and if it is less than 1 metre, a line should be drawn, associating the detected object to the marked object. This is done using the code below, which is done five times for each pole on the map:

```
% Plotting the mapped pole points
    for i=1:length(segment)
        if sqrt((Mapx(1)-xgfseg(i))^2+(Mapy(1)-ygfseg(i))^2)<1
            linx(1) = Mapx(1);
            linx(2) = xgfseg(i);
            liny(1) = Mapy(1);
            liny(2) = ygfseg(i);
        end
    end
```

This method seems to perform the data association in 99% of the cases on average.

In order to combine all of the data points from all parts to plot all the graphs, the following code is used:

```
        % if we want to show Local Frame, ...
        set(hL,'xdata',xcoor,'ydata',ycoor);
        % which points do have intensity>0??
        ii=find(I>0); set(hL2,'xdata',xcoor(ii),'ydata',ycoor(ii));
        % Grouping brilliant points
        set(hL3,'xdata',xseg,'ydata',yseg);

        % if we want to show Global Frame, ...
        set(hG,'xdata',xgf,'ydata',ygf);
        % which points do have intensity>0??
        set(hG2,'xdata',xgf(ii),'ydata',ygf(ii));
        % Grouping brilliant points
        set(hG3,'xdata',xgfseg,'ydata',ygfseg);
        % Showing Robot moving
```

```
set(hG4,'xdata',xrob,'ydata',yrob);
% Showing EKF
set(hE,'xdata',linx,'ydata',liny);
set(hE2,'xdata',linx2,'ydata',liny2);
set(hE3,'xdata',linx3,'ydata',liny3);
set(hE4,'xdata',linx4,'ydata',liny4);
set(hE5,'xdata',linx5,'ydata',liny5);
```

The following Figure 3 is then obtained: