**Abbreviated Lab Report**

*Lab #4: One State Extended Kalman Filter*

Prepared by: Sean Lantto

Team Members: Hussien AlAhmad and Conner Castle

03/29/2017

# Objective

* Learn how to implement an Extended Kalman Filter (EKF) for simple robot applications;
* Learn how to perform sensor fusion for a 1D navigation problem.

# Observation and Discussion

Initial attempts to turn 30⁰ left and back to 0 repeatedly for six times was done using only using the encoders. *Figure 1* in Appendix A shows the change in angle for this attempt.

Next the yaw rate gyro was used to perform the same task. Prior to implementing this however, the robot was left stationary to collect the gyro data to obtain a bias by finding the mean of the collected yaw rate data. With the bias of 0.0028 radians per epoch removed, the robot was placed to run the turning task using the yaw rate gyro. *Figure 2* shows the results of this run.

The SMART robot was configured with a LIDAR which was then used to estimate the yaw angle, referencing a wall 900 mm directly in front of it. The angle was estimated using *equation 1,* and the results from this run can be seen in *figure 3*. The LIDAR was about 155 mm from center of the robot which is where the angle was being measured from.

To implement the EKF for sensor fusion the variance of the yaw rate gyro and laser range finder were determined to be 2.6312e-5 (rad/sec)2 and 15.3413 mm2 respectively. The laser range finder was used for the measurement update and the yaw rate gyro dead reckoning was used for the prediction step. The following series of equations were used to implement the EKF. (The code can be found in Appendix B)

**Initial Values and State transition matrix**

**Prediction Update**

**Measurement Update**

*Figure 4* shows a comparison of the different measured values during the EKF run and the estimated sensor fusion result. The EKF spikes to several hundred degrees at the end of the six turns, the cause of which is not determined. It is obvious to see that the EKF favors the results of the LIDAR over the rate gyro.

Some potential error sources using sensor fusion could include, unmodeled dynamics of the system such as unexpected movement (wheels not traveling at the same rate). It could also include tuning errors, placing too much emphasis on one sensor over another. There could also be unknown environmental conditions that could disrupt the measurements of a particular sensor.

If translational movement along the x-axis was included, a second state would need to be added, such as distance to wall, or just x postion in general. This would require more sensors to be included in sensor fusion and new equations derived for the prediction and update steps.

# Problems and Solutions

The biggest issue was that SMART would move backwards while doing the turning task. This interfered with the LIDAR angle estimation and likely contributed to the others as well. The wheel speeds were set to be the same in the code, but it appeared that didn’t matter. Changing the wheel speeds some didn’t seem to make a difference either. The best solution was to hold a pencil on the back of the robot to try and stop the motion without interfering with the turning.

# Learning

I learned in this lab how to implement an EKF into a real-time physical system, unlike my previous experience with post processing using an EKF and with simulated systems only.

# Comments and Suggestions

It would be interesting to see the implementation of multiple states and a more complex task, however I don’t believe a single lab session would have been enough time to implement this. It also would have been interesting to try and include the ultrasonic rangefinders to estimate distance to the wall to help counter backwards travel.

# Appendix A: Figures and Table

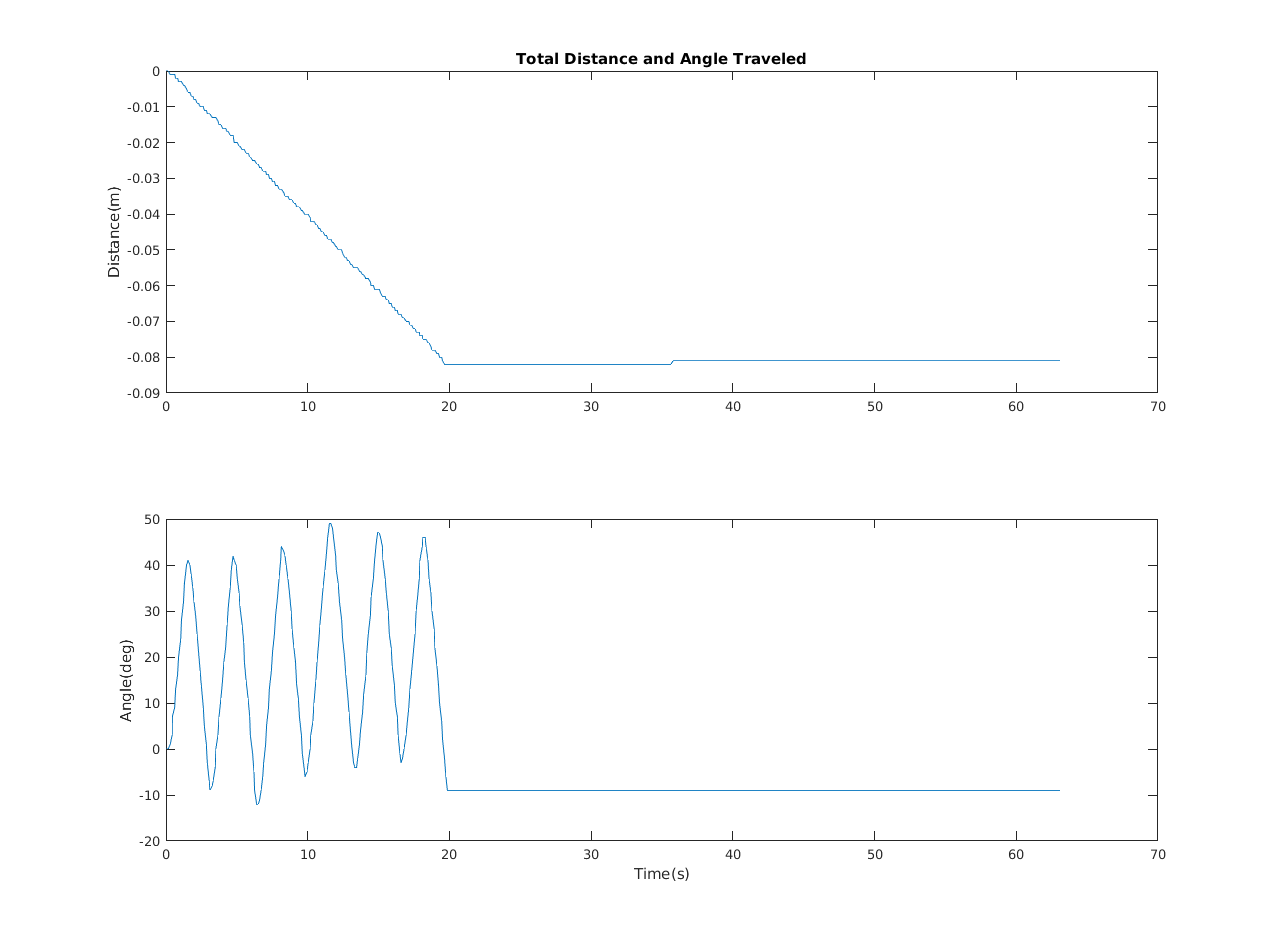


Figure : Angle and Distance according to SMART encoders

# 

Figure :Yaw angles according to yaw rate gyro, encoders and magnetometer.

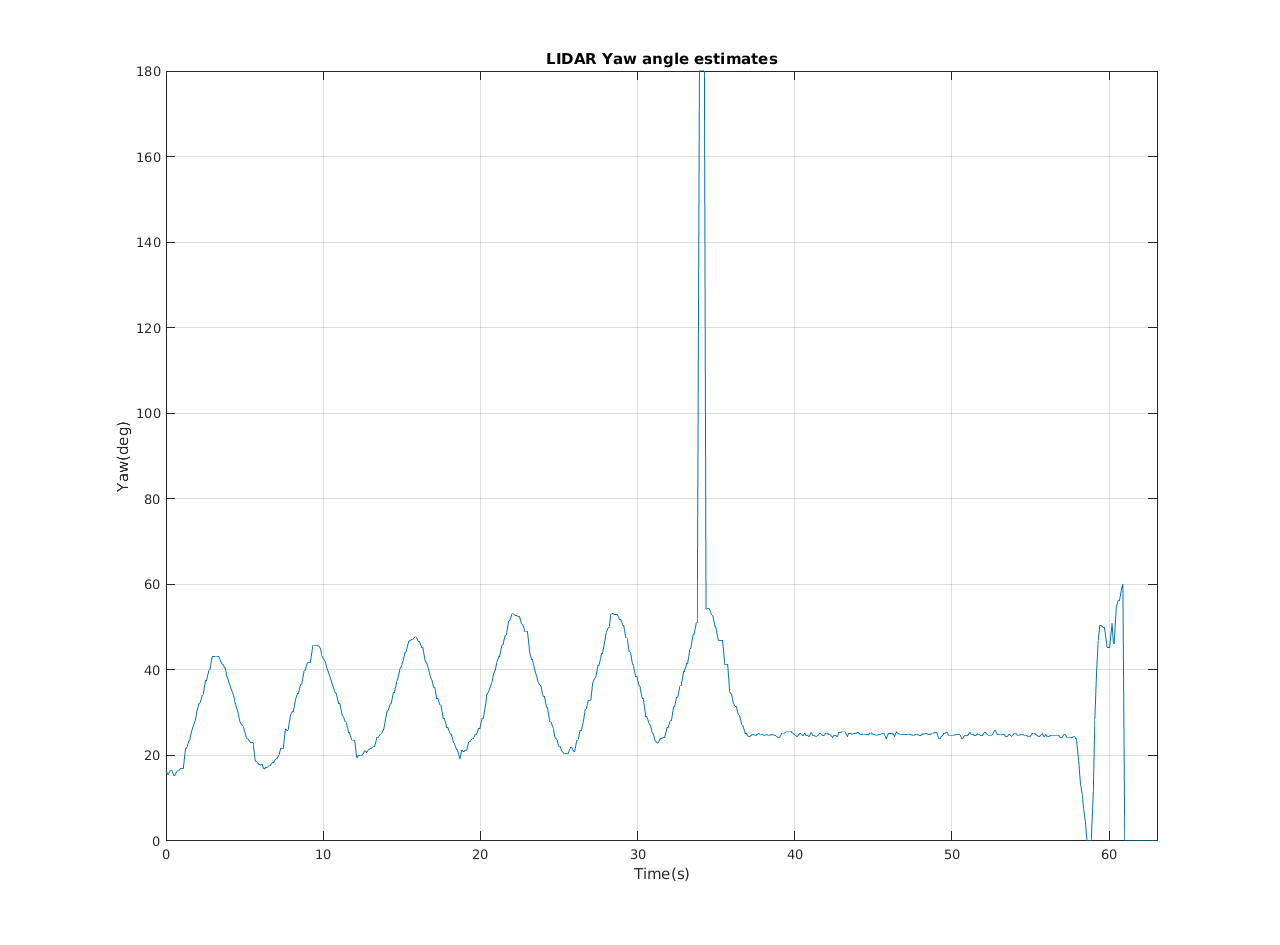


Figure : Yaw angle estimated from LIDAR data

# 

Figure :Comparison of EKF estimated yaw angle and measured yaw angles

# Appendix B: Source Code

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This is the main code for runing the SMART Robot %

% Author: Yu Gu %

% This is the version with Hukuyo lidar interface %

% The Kinect interface is removed %

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clear all

close all

% Definitions

Ts\_Desired=0.1; % Desired sampling time

Ts=0.1; % sampling time is 0.1 second. It can be reduced

% slightly to offset other overhead in the loop

Tend=60; % Was 60 seconds;

Total\_Steps=Tend/Ts\_Desired; % The total number of time steps;

Create\_Full\_Speed=0.2; % The highest speed the robot can travel. (Max is 0.5m/s)

% Magnetometer Calibration Data

Mag\_A=[ 2.3750 0.2485 -0.2296

0 2.6714 -0.1862

0 0 2.5061]; % estimated shape of the soft iron effect

Mag\_c=[ 0.0067; 0.2667; 0.0473]; % estimated center of the hard iron effect

% Rate Gyro Biases

P\_Bias=0;

Q\_Bias=0;

R\_Bias=0;

% P\_Bias=-0.0019;

% Q\_Bias= 0.0053;

% R\_Bias=-0.0149;

% Define the main Data Structure

SD= struct( 'Index',zeros(1,Total\_Steps),... % SD stands for SMART Data, which stores all the robot data

'Time', zeros(1,Total\_Steps),... % The actual time at each time step (s)

'Time\_Diff', zeros(1,Total\_Steps),... % The time difference between two step (s)

'Delay',zeros(1,Total\_Steps),... % Delay needed between each time step. It indicate how much time avaliable for other compuations.

'Lidar\_Angle', zeros(682,Total\_Steps),... % Hukuyo Lidar Scan Angles

'Lidar\_Range', zeros(682,Total\_Steps),... % Hukuyo Lidar Scan Range

'Logger\_Counter', zeros(1,Total\_Steps),... % The Counter number for the SMART data logger

'Ax',zeros(1,Total\_Steps),... % Acceleration along the x-Axis (g)

'Ay',zeros(1,Total\_Steps),... % Acceleration along the y-Axis (g)

'Az',zeros(1,Total\_Steps),... % Acceleration along the z-Axis (g)

'P',zeros(1,Total\_Steps),... % Roll Rate (deg/s)

'Q',zeros(1,Total\_Steps),... % Pitch Rate (deg/s)

'R',zeros(1,Total\_Steps),... % Yaw Rate (deg/s)

'Mx',zeros(1,Total\_Steps),... % Magnetic strength along the x-axis (G)

'My',zeros(1,Total\_Steps),... % Magnetic strength along the y-axis (G)

'Mz',zeros(1,Total\_Steps),... % Magnetic strength along the z-axis (G)

'IMU\_T',zeros(1,Total\_Steps),... % IMU internal temperacture (C)

'Roll',zeros(1,Total\_Steps),... % Roll Angle (rad)

'Pitch',zeros(1,Total\_Steps),... % Pitch Angle (rad)

'Yaw',zeros(1,Total\_Steps),... % Yaw Angle (rad)

'Mag\_Heading',zeros(1,Total\_Steps),... % Magnetic Heading (rad)

'X',zeros(1,Total\_Steps),... % Robot X Position (m)

'Y',zeros(1,Total\_Steps),... % Robot Y Position (m)

'RF\_F',zeros(1,Total\_Steps),... % Front Range Finder (mm)

'RF\_FL',zeros(1,Total\_Steps),... % Front-Left Range Finder (mm)

'RF\_L',zeros(1,Total\_Steps),... % Left Range Finder (mm)

'RF\_B',zeros(1,Total\_Steps),... % Back Range Finder (mm)

'RF\_R',zeros(1,Total\_Steps),... % Right Range Finder (mm)

'RF\_FR',zeros(1,Total\_Steps),... % Front Right Range Finder (mm)

'Laser\_RF', zeros(1,Total\_Steps),... % Laser Range Finder (mm)

'Wall', zeros(1,Total\_Steps),... % Wall sensor of Create (0/1)

'VirtWall', zeros(1,Total\_Steps),... % Detect the Virtual Wall (0/1)

'Dist', zeros(1,Total\_Steps),... % Distance Traveled Since Last Call (m)

'TotalDist', zeros(1,Total\_Steps),... % Total Distance Traveled(m)

'Angle', zeros(1,Total\_Steps),... % Angle Traveled Since Last Call (rad)

'TotalAngle', zeros(1,Total\_Steps),... % Total Angle Traveled (rad)

'CreateVolts', zeros(1,Total\_Steps),... % Voltage of the Create Robot (rad)

'CreateCurrent', zeros(1,Total\_Steps)); % Current of the Create Robot (rad)

% Initialize the Serial Port for the Data Logger

S\_Logger=Init\_Logger('2'); % May be different on different computer

% Initialize the Serial Port for iRobot Create

S\_Create=RoombaInit(1);

% Initialize the Serial Port for LightWare Laser Rangefinder

% S\_LightWare=Init\_LightWare('3');

% Initialize the Serial Port for Hukuyo Lidar

S\_Hokuyo=Init\_Hokuyo(4); % initalize the Lidar

pause(0.1);

fscanf(S\_Hokuyo)

rangescan=zeros(1, 682);

for i=1:Total\_Steps

SD.Lidar\_Angle(:,i)=(-120:240/682:120-240/682)\*pi/180; % an Array of all the scan angles

end

fprintf(S\_Hokuyo,'GD0044072500'); % request a scan

flushinput(S\_Logger); % Flush the data logger serial port

flushinput(S\_Create); % Flush the iRobot Create serial port

fwrite(S\_Create, [142 0]); % Request all sensor data from Create

BeepRoomba(S\_Create); % Make a Beeping Sound

pause(0.1);

turncount=0;

turnflag=0; %0 for left 1 for right

SD.thetar=zeros(1,600);

A=1;

X\_P=0;

P\_P=1;

% Starting the main loop

for i=1:Total\_Steps

tic

SD.Index(i)=i;

% Acquire data from the sensor interface board

if i==1

[SD.Logger\_Counter(i), SD.Ax(i), SD.Ay(i), SD.Az(i), Raw\_P, Raw\_Q, Raw\_R, Raw\_Mx, Raw\_My, Raw\_Mz, SD.IMU\_T(i), A2D\_Ch1, A2D\_Ch2, SD.RF\_F(i), SD.RF\_FL(i), SD.RF\_L(i), SD.RF\_B(i), SD.RF\_R(i), SD.RF\_FR(i)] = Read\_Logger\_2(S\_Logger,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0);

else

[SD.Logger\_Counter(i), SD.Ax(i), SD.Ay(i), SD.Az(i), Raw\_P, Raw\_Q, Raw\_R, Raw\_Mx, Raw\_My, Raw\_Mz, SD.IMU\_T(i), A2D\_Ch1, A2D\_Ch2, SD.RF\_F(i), SD.RF\_FL(i), SD.RF\_L(i), SD.RF\_B(i), SD.RF\_R(i), SD.RF\_FR(i)] = Read\_Logger\_2(S\_Logger,SD.Logger\_Counter(i-1),SD.Ax(i-1),SD.Ay(i-1),SD.Az(i-1),Raw\_P, Raw\_Q, Raw\_R, Raw\_Mx, Raw\_My, Raw\_Mz, SD.IMU\_T(i-1), A2D\_Ch1, A2D\_Ch2, SD.RF\_F(i-1), SD.RF\_FL(i-1), SD.RF\_L(i-1), SD.RF\_B(i-1), SD.RF\_R(i-1), SD.RF\_FR(i-1));

end

SD.P(i)=Raw\_P-P\_Bias; SD.Q(i)=Raw\_Q-Q\_Bias; SD.R(i)=Raw\_R-R\_Bias; % Calibrate the gyro data

temp=Mag\_A\*[Raw\_Mx-Mag\_c(1); Raw\_My-Mag\_c(2); Raw\_Mz-Mag\_c(3)]; % Magnetometer Raw Data Correction

SD.Mx(i)=temp(1); SD.My(i)=temp(2); SD.Mz(i)=temp(3);

flushinput(S\_Logger); % Flush the data logger serial port

% Acquire data from the LightWare Laser Rangefinder

% if i==1

% SD.Laser\_RF(i) = Read\_LightWare(S\_LightWare,0);

% else

% SD.Laser\_RF(i) = Read\_LightWare(S\_LightWare,SD.Laser\_RF(i-1));

% end

% flushinput(S\_LightWare); % Flush the LightWare Laser Rangefinder serial port

% Acquire the data from the iRobot Create

[BumpRight,BumpLeft,BumpFront,SD.Wall(i),SD.VirtWall(i),CliffLeft,CliffRight,CliffFrontLeft,CliffFrontRight,LeftCurrOver,RightCurrOver,DirtL,DirtR,ButtonPlay,ButtonAv,SD.Dist(i),SD.Angle(i),SD.CreateVolts(i),SD.CreateCurrent(i),Temp,Charge,Capacity,pCharge]=Read\_Create\_2(S\_Create);

flushinput(S\_Create); % Flush the iRobot Create serial port

fwrite(S\_Create, [142 0]); % Request all sensor data from Create

% Acquire data from Hukuyo Lidar

rangescan = Read\_Hokuyo(S\_Hokuyo, rangescan);

SD.Lidar\_Range(:,i)=rangescan;

SD.Laser\_RF(i) = rangescan(341); % simulated laser rangefinder using the LIDAR center spot

% plot(rangescan);

% drawnow

flushinput(S\_Hokuyo); % Flush the Lidar serial port

fprintf(S\_Hokuyo,'GD0044072500'); % request a new Lidar scan

Time=clock; % Mark the current time;

SD.Time(i)=Time(6); % Store the seconds;

if i==1

SD.Roll(i)=atan2(SD.Ay(i), SD.Az(i)); % Calculate the Roll angle based on the gravity vector

SD.Pitch(i)=atan(-SD.Ax(i)/(SD.Ay(i)\*sin(SD.Roll(i))+SD.Az(i)\*cos(SD.Roll(i)))); % Calculate the Roll angle based on the pitch vector

SD.Mag\_Heading(i)=atan2(-SD.My(i), SD.Mx(i)); % The initial 2D magnetic heading of the robot

SD.Yaw(i)=0; % Set the current yaw angle as zero

Attitude\_P=zeros(3,3); % Initialization the error covariance matrix for attitude estimation

SD.X(i)=0; % Initial Robot X Position

SD.Y(i)=0; % Inital Robot Y Position

else % If i>1

SD.Time\_Diff(i)=SD.Time(i)-SD.Time(i-1); % Calculate the time difference between steps

if SD.Time\_Diff(i)<0

SD.Time\_Diff(i)=SD.Time\_Diff(i)+60; % Compensate for the minute change

end

SD.TotalDist(i)=SD.TotalDist(i-1)+SD.Dist(i); % Calculate the total traveled distance based on the encoder reading

SD.TotalAngle(i)=SD.TotalAngle(i-1)+SD.Angle(i); % Calculate the total traveled angle based on the encoder reading

SD.Mag\_Heading(i)=atan2(-SD.My(i), SD.Mx(i)); % The 2D magnetic heading of the robot

[SD.Roll(i), SD.Pitch(i), SD.Yaw(i), Attitude\_P] = Attitude\_Estimation(SD.Mag\_Heading(1), SD.Roll(i-1), SD.Pitch(i-1), SD.Yaw(i-1), SD.Time\_Diff(i), Attitude\_P, SD.Ax(i), SD.Ay(i), SD.Az(i), SD.P(i), SD.Q(i), SD.R(i), SD.Mx(i), SD.My(i), SD.Mz(i)); % Perform the attitude estimation

SD.X(i)=SD.X(i-1)+SD.Dist(i)\*sin(SD.Yaw(i)-SD.Yaw(1)); % Dead Reconing for X position

SD.Y(i)=SD.Y(i-1)+SD.Dist(i)\*cos(SD.Yaw(i)-SD.Yaw(1)); % Dead Reconing for X position

end

%% Put your custom control functions here

%----------------------------------------------------------

% Use the following function for the robot wheel control:

% SetDriveWheelsSMART(S\_Create, rightWheelVel, leftWheelVel, SD.CliffLeft(i),SD.CliffRight(i),SD.CliffFrontLeft(i),SD.CliffFrontRight(i));

SD.thetaL(i)=acosd(900/(SD.Laser\_RF(i)-155));

if i==1

SD.thetar(i)=0+(SD.R(i)+0.0028)\*SD.Time\_Diff(i);

else

SD.thetar(i)=SD.thetar(i-1)+(SD.R(i)+0.0028)\*SD.Time\_Diff(i);

end

Q=(SD.Time\_Diff(i)^2)\*2.6312e-5;

R=10.3413;

%Prediction

X\_A=X\_P-SD.Time\_Diff(i)\*(SD.R(i)-0.0028)+Q;

P\_A=A\*P\_P\*A'+Q;

%Measurement update

H=900\*sec(X\_A)\*tan(X\_A);

y=SD.Laser\_RF(i)+R;

r=y-(900\*sec(X\_A)+155);

S=H\*P\_A\*H'+R;

K=P\_A\*H'\*inv(S);

X\_P=X\_A+K\*r;

P\_P=1-K\*H\*P\_A;

SD.ThetaEKF(i)=X\_P;

if turncount<6

if turnflag==0

SetDriveWheelsSMART(S\_Create, Create\_Full\_Speed\*.2, Create\_Full\_Speed\*-.2, CliffLeft,CliffRight,CliffFrontLeft,CliffFrontRight,BumpRight,BumpLeft,BumpFront);

if abs(SD.thetar(i)) >= deg2rad(30)

turnflag=1;

end

else

SetDriveWheelsSMART(S\_Create, Create\_Full\_Speed\*-.2, Create\_Full\_Speed\*.2, CliffLeft,CliffRight,CliffFrontLeft,CliffFrontRight,BumpRight,BumpLeft,BumpFront);

if -SD.thetar(i) <= 0

turnflag=0;

turncount=turncount+1;

end

end

else

SetDriveWheelsSMART(S\_Create, Create\_Full\_Speed\*0, Create\_Full\_Speed\*0, CliffLeft,CliffRight,CliffFrontLeft,CliffFrontRight,BumpRight,BumpLeft,BumpFront);

end

%----------------------------------------------------------

%% End of the Custom Control Code

% Calculate the time left in the constant interval

SD.Delay(i)=Ts-toc;

if SD.Delay(i)>0

pause(SD.Delay(i)); % Kill the remaining time

end

end

Total\_Elapse\_Time=SD.Time(Total\_Steps)-SD.Time(1) % Calcualte the total elapse time, not counting the minutes

SetDriveWheelsSMART(S\_Create, 0, 0, CliffLeft,CliffRight,CliffFrontLeft,CliffFrontRight,BumpRight,BumpLeft,BumpFront); % Stop the wheels

BeepRoomba(S\_Create); % Make a Beeping Sound

% Properly close the serial ports

delete(S\_Logger)

clear S\_Logger

delete(S\_Create)

clear S\_Create

%delete(S\_LightWare)

%clear S\_LightWare

fprintf(S\_Hokuyo,'QT');

fclose(S\_Hokuyo);

save('SMART\_DATA.mat', 'SD'); % Save all the collected data to a .mat file

SMART\_PLOT; % Plot all the robot data