Advanced Control 5 (ENG5009)

Lab Assignment

2127147b

March 22, 2019

Abstract

The following report outlines the development and testing of a waypoint following and obstacle avoidance system for the simulation of an autonomous robot in MATLAB. The system presented uses fuzzy logic controllers to generate desired turning commands and motor gains. A range of different input types were used alongside basic signal processing techniques to provide the fuzzy controllers with sufficient insight into the surrounding environment. The controller was found to produce successful results with the robot travelling to a specific coordinate within 0.05m. Further development and fine-tuning was carried out to optimise the controller performance for a set of different scenarios. All code can be found on GitHub at [1], relevant code is included in the appendices.

1 Methodology

1.1 Overview of System

Two cascaded fuzzy controllers are used, the first (path controller) determines a desired turn command based solely on the robot's current heading angle (ψ) and its angle relative to a desired waypoint (ψ_{ref}). The second controller takes the generated turn command with inputs relating to a nearby object (d_{wall}, $\bar{d}_{wall}$, Θ_{wall}, r) to determine appropriate gains for the left and right motors. A variable lowpass Finite Impulse Response (FIR) filter is used for each motor gain output to smooth the voltages, its cutoff frequency is controlled by one of the fuzzy motor controller outputs. A proportional drive voltage is applied to each motor that varies with the robot's distance from the waypoint, it remains constant until close to the waypoint. A block diagram of the system can be seen in figure 1.

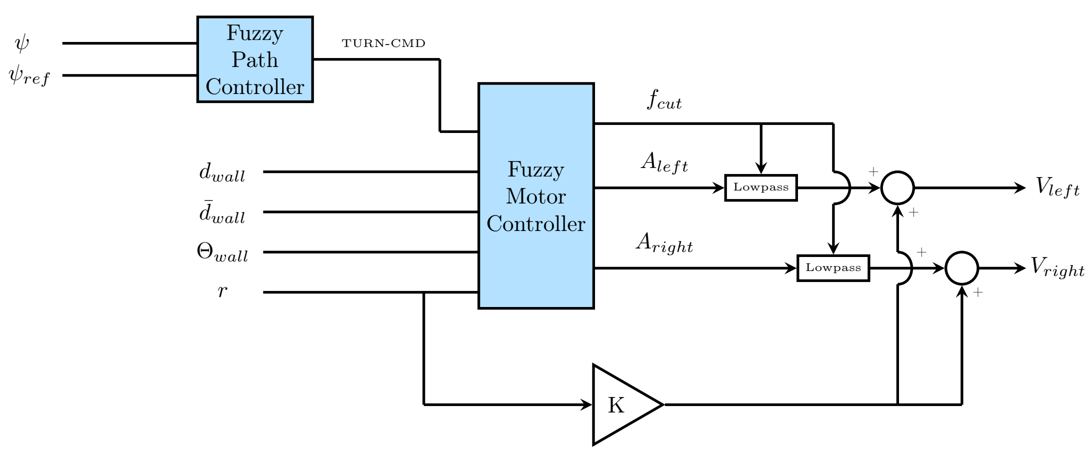


Figure 1: Block Diagram of Control System

1.2 Task 1: Waypoint Following

Overview

The aim of this task is to guide the robot to a set waypoint without obstacle avoidance. This was achieved using fuzzy logic to generate a desired turn command by comparing the robot's current heading angle (ψ) with its angle relative to the waypoint (ψ_{ref}). The fuzzy logic is implemented to drive the error between both angles to zero such that the robot travels in the direction of the waypoint.

A second fuzzy controller was created to control the gain applied to the motors, it takes the generated turn command and radius from the waypoint as inputs. As the robot approaches the waypoint, the rules are altered to enable coarser manoeuvres allowing it to stop within a 0.05m tolerance. A drive voltage is also proportionally reduced as the robot's position converges on the waypoint.

Fuzzy Sets

The input variables to the path controller are the heading and reference angles, they are measured from 0 rads (north), to either $-\pi$ or $+\pi$ rads (south) where a negative angle represents a counterclockwise angle and vice versa. Nine fuzzy input sets were derived as follows, $\{S_{\text{-ve}}$, SW, W, NW, N, NE, E, SE, $S_{+\text{ve}}\}$, where N is north, NE is north-east etc, these sets are identical for both the heading and reference angle inputs. $S_{\text{-ve}}$ and $S_{+\text{ve}}$ both represent a range around south as the angle jumps from -π to $+\pi$ and are therefore treated as the same set in the fuzzy rules. Trapezoidal membership functions were used for fuzzification, they can be seen for the heading angle input in figure 2, the sets for the reference angle input are identical.

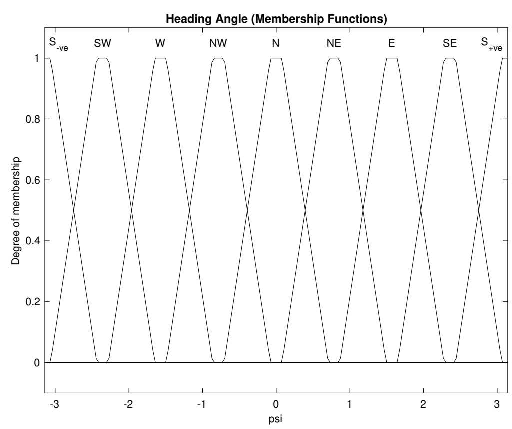


Figure 2: Membership Functions for Heading Angle Input

The path controller output variable is a turning command with the derived set, $\{L_{\text{rev}}$, L_{rot}, L_{hard}, L_{soft}, FWD, R_{soft}, R_{hard}, R_{rot}, $R_{\text{rev}}\}$, for reverse, rotate, hard, soft and forward manoeuvres respectively. These commands allow for a variety of coarse or fine turning adjustments to be made by the motor controller, trapezoidal membership functions were used for the fuzzification and can be seen in figure 3

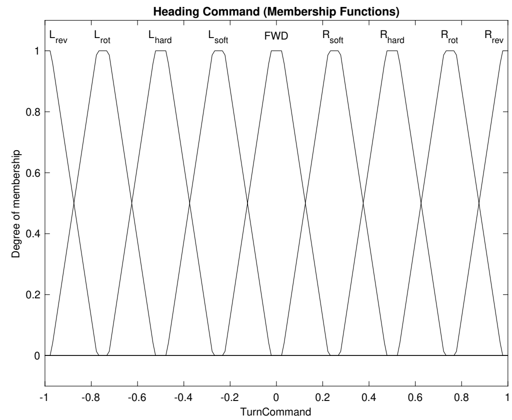


Figure 3: Membership Functions for Turn Command Output

The second input to the motor controller is the robot's radius to the waypoint with the following set, {VN, N, F, VF}, representing very-near, near, far and very-far respectively. This is used to execute tighter turning manoeuvres when close to the waypoint as when the robot is very-near to the waypoint, the turn command will begin to change much more rapidly. This therefore needs to be taken into account such that the robot can navigate to within 0.05m. The radius is also used in the main code to proportionally reduce the drive voltage on approach to the waypoint. Trapezoidal membership functions are used and can be seen in figure 4.

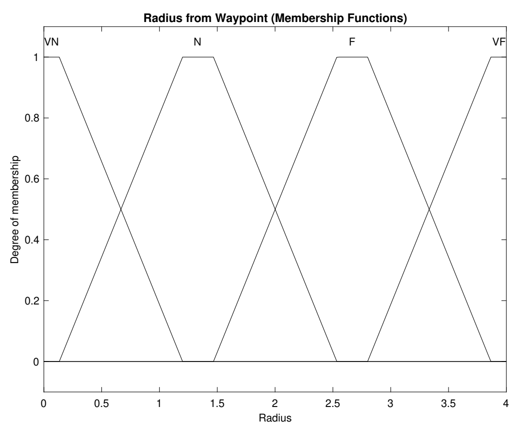


Figure 4: Membership Functions for Radius Input

The motor controller produces gains for the motors as outputs in the range of -1 to 1 with the derived set, {REV_{hard}, REV_{soft}, OFF, FWD_{soft}, FWD_{hard}}, representing hard-reverse, soft-reverse, off, soft-forward and hard-forward manoeuvres repectively. These gains are scaled in the main code to an appropriate range and limited to the maximum range of $\pm$7.4V. Triangular membership functions are used and can be seen in figure 5.

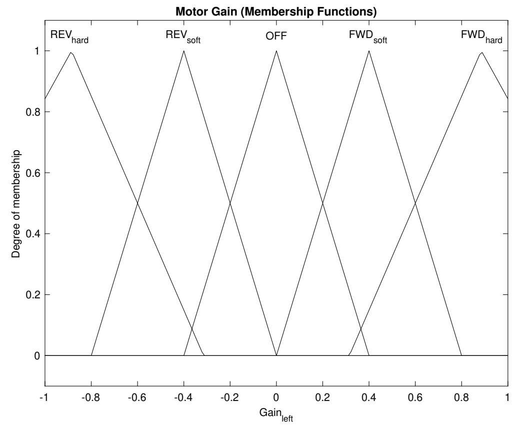


Figure 5: Membership Functions for Motor Gain Outputs

Rules

The path controller will produce a turning command that is appropriate for the robot's heading in relation to the its reference angle to the waypoint. For example: **IF** the robot is facing north (ψ=N) **AND** its bearing is north east (ψ_{ref}=NE), **THEN** turn soft right (R_{soft}). In this situation only a soft right turn is required as angles in north and north-east are likely to be close together, a harder manoeuvre is more likely to result in an undesired overshoot. Rules were derived for each combination of headings and reference angles, a sample of the 81 rules can be seen in table 1 for a heading angle of south-positive. The defuzzification method used was the bisector method. The entire set of rules are included in appendix ???????????????????.

Table 1: Sample of Fuzzy Logic Rules for Path Controller (outputs in yellow)

ψ_{ref}	ψ	**TURN CMD**
S_{-ve}	S_{-ve}	FWD
S_{-ve}	SW	L_{soft}
S_{-ve}	W	L_{hard}
S_{-ve}	NW	L_{rot}
S_{-ve}	N	L_{rev}
S_{-ve}	NE	R_{rot}
S_{-ve}	E	R_{hard}
S_{-ve}	SE	R_{soft}
S_{-ve}	S_{+ve}	FWD

The motor controller interprets these turn commands by applying appropriate gains to the left and right motors such that robot executes the requested turning manoeuvre. For example: **IF** the requested manoeuvre is a soft-right turn (TURN CMD = R_{soft}) **AND** the robot is not very-near to the waypoint (r != VN), **THEN** A_{left} is FWD_{soft} **AND** A_{left} is OFF. If the robot is very-near to the waypoint then it will only execute rotational manoeuvres, for example: **IF** the requested manoeuvre is a soft-right turn **AND** the robot is very-near (r = VN), **THEN** A_{left} is FWD_{soft} **AND** A_{left} is REV_{soft}.

These rules can be seen in table 2

Table 2: Truth table of motor controller rules (outputs in yellow)

TURN CMD	r	A_{left}	A_{right}
FWD	!VN	$\mathrm{FWD_{soft}}$	$\mathrm{FWD_{soft}}$
$\mathrm{L_{soft}}$	!VN	OFF	$\mathrm{FWD_{soft}}$
$\mathrm{L_{hard}}$	!VN	$\mathrm{REV_{soft}}$	$\mathrm{FWD_{hard}}$
$\mathrm{L_{rot}}$	!VN	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$
$\mathrm{L_{rev}}$	!VN	$\mathrm{REV_{hard}}$	$\mathrm{REV_{soft}}$
$\mathrm{R_{rev}}$	!VN	$\mathrm{REV_{soft}}$	$\mathrm{REV_{hard}}$
$\mathrm{R_{rot}}$	!VN	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$
$\mathrm{R_{hard}}$	!VN	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{soft}}$
$\mathrm{R_{soft}}$	!VN	$\mathrm{FWD_{soft}}$	OFF
FWD	VN	$\mathrm{FWD_{soft}}$	$\mathrm{FWD_{soft}}$
$\mathrm{L_{soft}}$	VN	$\mathrm{REV_{soft}}$	$\mathrm{FWD_{soft}}$
$\mathrm{L_{hard}}$	VN	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$
$\mathrm{L_{rot}}$	VN	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$
$\mathrm{L_{rev}}$	VN	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$
$\mathrm{R_{rev}}$	VN	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$
$\mathrm{R_{rot}}$	VN	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$
$\mathrm{R_{hard}}$	VN	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$
$\mathrm{R_{soft}}$	VN	$\mathrm{FWD_{soft}}$	$\mathrm{REV_{soft}}$

Verification

The path controller was tested using the waypoints specified in the assignment brief (task 1). The results shown in table 3 demonstrate that the testing was successful as the robot stopped at a point within 0.05m of the waypoint in each scenario. An example plot of the robot's path from the origin to (-0.2, 2) is shown in figure 6.

Table 3: Start and end coordinates for task 1

Start Coordinate	**End Coordinate**
(0, 3)	(-0.043, 3.0246)
(1, 2)	(0.9949, 2.0463)
(-1, 4)	(-1.0409, 4.0230)
(-1, 2)	(-1.0448, -1.9871)
(0, 3)	(-0.2391, 2.0056)

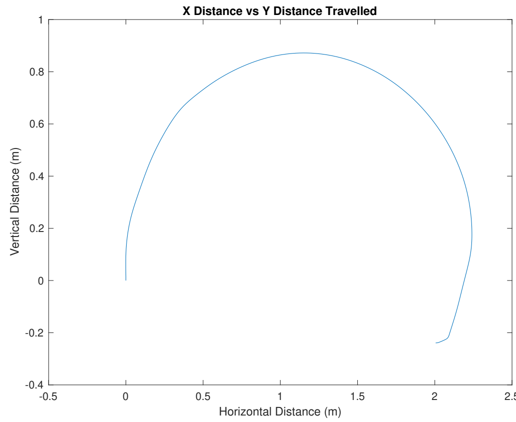


Figure 6: Path of robot from the origin to (-0.2, 2)

1.3 Task 2: Obstacle Avoidance

Overview

The aim of this task is to guide the robot to a set waypoint as before, however doing so while avoiding obstacles. This was achieved using additional inputs from distance sensors to the motor controller such that the robot could execute evasive manoeuvres when in the proximity of a wall. The fuzzy logic is implemented in such a way as to drive the error between the heading and reference angles to zero (as in task one) when outwith the proximity of a wall however, different evasion commands would be executed as required when near a wall.

Fuzzy Sets

The angle at which the robot approaches an obstacle (Θ_{wall}) is an input to the motor controller and is calculated in the main code using the following equation: $\Theta = arctan\left(\frac{d_{left} - d_{right}}{\Delta x}\right)$, where d_{left}, d_{right} and Δx are the left sensor distance, right sensor distance and distance between the sensors respectively. As the maximum output from a sensor is 1 and the distance between the sensors is 0.2, the maximum detectable wall angle/slope is 1.3734 and therefore the range of the fuzzy input set is ± 1.3734. The derived set is as follows, {-ve_{hard}, -ve_{soft}, FLAT, +ve_{soft}, +ve_{hard}}, where each member represents a slope gradient.

For example, -ve_{hard} refers to a steep negative gradient relative to the direction that the robot is facing the wall. This set uses trapezoidal membership functions for all but the FLAT member, it was found that the controller performed best when limiting the FLAT member to a small range with a triangular function as otherwise it would become indecisive when approaching the wall at a slight angle in certain scenarios. These can be seen in figure 7.

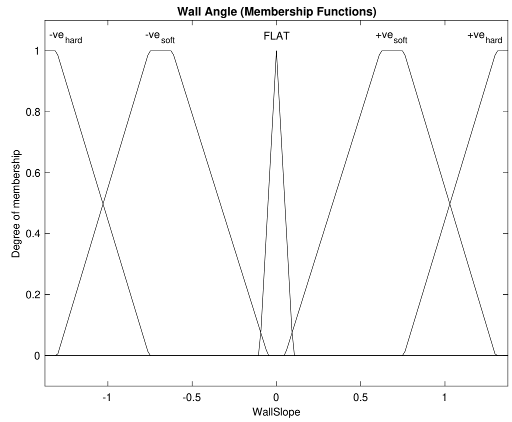


Figure 7: Membership Functions for Wall Angle Input

The proximity to the wall is determined as the smaller of either distances detected by the left or right sensors and is an input with the following set, {CLOSE, NEAR, FAR} with trapezoidal membership functions that can be seen in figure 8. Since the sensor returns one when an obstacle is not detected, any input greater than this is definitively a FAR value of proximity as this allows the controller logic to function reliably when detecting a wall. The controller is designed to guide the robot alongside the wall until it reaches its corner, at which point it can continue towards the waypoint as normal.

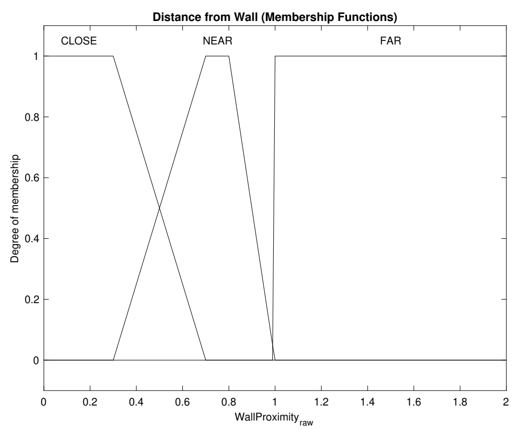


Figure 8: Membership Functions for Wall Proximity Input

In certain situations the robot will begin tracking along a wall with the waypoint initially directly on the other side. As it moves along the wall, the difference between the heading and reference angles begins to increase and therefore, the requested turning command will become more aggressive. This becomes a problem when the robot is in parallel with the wall as its sensors return a value of one and thus the robot acts as though it is FAR from the wall. In this situation, when the turning command is aggressive, the robot will jolt towards the wall as soon as it becomes parallel.

To rectify this, it was necessary to provide the controller with an input that could indicate whether or not the robot had cleared a wall while tracking along it. This was achieved by taking a 20 sample moving average (mean) of the wall proximity and providing this as an additional input ($\bar{d}_{wall}$), the robot could be assumed to have cleared a wall only when both the filtered and unfiltered proximity values are one. The fuzzy set and membership functions are identical to that of the unfiltered proximity seen in figure 8.

A variable, Finite Impulse Response (FIR) lowpass filter was applied to smooth the output voltages. This was achieved by writing and using a sample by sample FIR filter class, this can be seen in appendix ?????????.

Its cutoff frequency is an output of the motor controller such that the motor responsiveness could be adjusted dynamically in certain scenarios, its set is as follows, {SLOW, AVG, FAST}. Trapezoidal membership function are used and can be seen in figure 9, its total range is from 0 to 1 as the filter is normalised to the nyquist frequency.

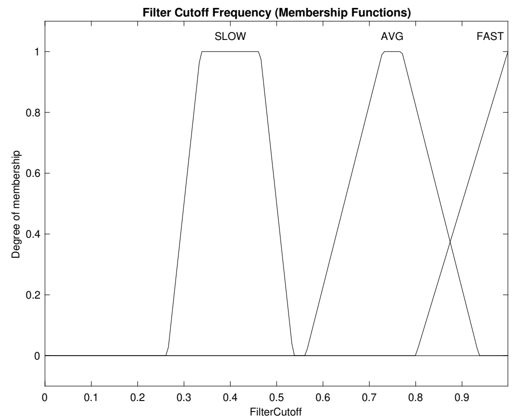


Figure 9: Membership Functions for Filter Cutoff Frequency Output

Rules

When the robot is far from a wall and has cleared it in the situation that it was tracking alongside one, then it applies the same turn command that of task one in table 1. When in proximity of a wall, an appropriate manoeuvre is executed to alter the robot such that it is in parallel with the wall depending on the wall angle relative to the robots heading. For example, **IF** the robot is facing the wall with a steep positive gradient (Θ_{wall} = +ve$_{\text{hard}}$) **AND** the robot is NEAR to the wall, **THEN** A_{left} is FWD$_{\text{hard}}$ **AND** A_{right} is FWD$_{\text{soft}}$. In this scenario, the robot is close to parallel to the wall so only a gentle right turn is required. In situations where the robot approaches the wall at a flat angle, the robot will turn left or right depending on the current turn command requested by the path controller, if the turn command requested is forward the robot will turn right by default. This can be seen in table 4.

Table 4: Truth table of motor controller rules when within proximity of a wall

TURN CMD	r	Θ_{wall}	d_{wall}	d_{wall}	A_{left}	A_{right}	ω_{cut}
X	X	+ve$_{\text{hard}}$	NEAR	X	FWD$_{\text{hard}}$	FWD$_{\text{soft}}$	AVG
X	X	+ve$_{\text{soft}}$	NEAR	X	FWD$_{\text{hard}}$	REV$_{\text{soft}}$	AVG
X	X	+ve$_{\text{hard}}$	CLOSE	X	FWD$_{\text{hard}}$	FWD$_{\text{soft}}$	AVG
X	X	+ve$_{\text{soft}}$	CLOSE	X	FWD$_{\text{hard}}$	REV$_{\text{hard}}$	AVG
X	X	-ve$_{\text{hard}}$	NEAR	X	FWD$_{\text{soft}}$	FWD$_{\text{hard}}$	AVG
X	X	-ve$_{\text{soft}}$	NEAR	X	REV$_{\text{soft}}$	FWD$_{\text{hard}}$	AVG
X	X	-ve$_{\text{hard}}$	CLOSE	X	FWD$_{\text{soft}}$	FWD$_{\text{hard}}$	AVG
X	X	-ve$_{\text{soft}}$	CLOSE	X	REV$_{\text{hard}}$	FWD$_{\text{hard}}$	AVG
FWD	X	FLAT	!FAR	X	FWD$_{\text{hard}}$	REV$_{\text{soft}}$	FAST
L$_{\text{any}}$	X	FLAT	!FAR	X	REV$_{\text{soft}}$	FWD$_{\text{hard}}$	FAST
R$_{\text{any}}$	X	FLAT	!FAR	X	FWD$_{\text{hard}}$	REV$_{\text{soft}}$	FAST

When tracking along a wall, if the robot is parallel and the waypoint resides an the other side of it, the motor controller will execute a soft turning command towards the wall. This prevents the the robot jolting towards the wall when an aggressive turning command is requested by the path controller as described previously. The logic is implemented such that these rules are enabled when the unfiltered wall proximity is FAR but the unfiltered proximity is not, i.e. the robot is parallel and therefore still tracking along the wall. This can be seen in figure 5.

Table 5: Truth table of motor controller rules when approximately parallel to wall

TURN CMD	r	Θ_{wall}	d_{wall}	d_{wall}	A_{left}	A_{right}	ω_{cut}
R_{rev}	!VN	X	FAR	!FAR	FWD_{hard}	FWD_{soft}	AVG
R_{rot}	!VN	X	FAR	!FAR	FWD_{hard}	FWD_{soft}	AVG
L_{rev}	!VN	X	FAR	!FAR	FWD_{soft}	FWD_{hard}	AVG
L_{rot}	!VN	X	FAR	!FAR	FWD_{soft}	FWD_{hard}	AVG

2 Results and Testing

Assigned Waypoint

The following plot (figure 10) shows the robot's path towards the waypoint specified in the assignment brief (3.5, 2.5). The robot was successful in reaching the waypoint within a 0.05m tolerance, finishing at the coordinate (3.5342, 2.5302). A subtle avoidance manoeuvre is executed as it reaches the wall at (1.2, -1), the robot tracks alongside the wall until it has cleared it at (1.2, 1) at which it point it turns left towards the waypoint. As its position converges on that of the waypoint, a left rotation is executed demonstrating the successful execution of rules to be enabled when very-near to the waypoint.

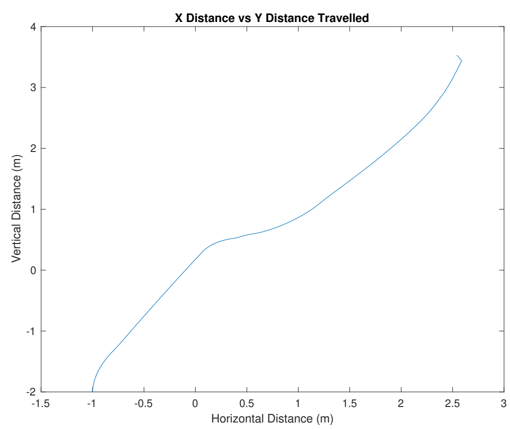
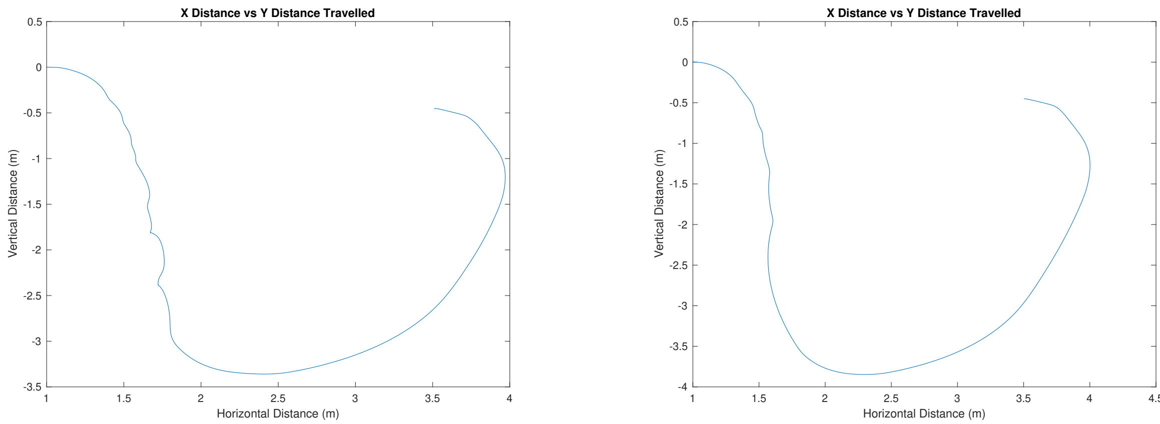


Figure 10: Robot path to waypoint (3.5, 2.5) from starting point (-2, -1)

Wall Tracking

In order to test the robot's ability to detect whether or not it was in parallel to a wall, the following test scenario was devised. The robot is initially directly facing a wall, the waypoint is placed directly opposite the robot on the other side of the wall. The controller will guide the robot around the wall to the right, when the filtered proximity input is enabled the robot should track along the wall smoothly and when disabled, it should track in a jagged fashion. This input was disabled by setting it as one, ensuring that the relevant rules would never fire. The results can be seen in figure 11. When the filtered proximity input is enabled (11b), the path as the robot tracks along the wall is much smoother than when disabled (11a).



(a) No filtered proximity input rules applied

(b) Filtered proximity input rules applied

Figure 11: Robot path with filtered proximity input

3 Further Work

More rules need to be developed for waypoints residing close to a wall, the requested turn command will be ignored in such a scenario. As the robot converges on the waypoint, it will come into proximity of the wall and the coarser turn commands, to be enabled when very-near to the waypoint, will not be fired resulting in the robot being unable to reach the point. This could be rectified by adding further logic to account for the robot being near a wall AND the waypoint.

Simplifications could have been made to the logic to reduce unused or rarely used variables and members. The path controller's logic could have been simplified by calculating the error between the heading and reference angles and using this as a single fuzzy set, this would have resulted in only 9 rules rather than 81. The lowpass filter cutoff frequency is also rarely altered with the exception of the robot approaching a wall perpendicularly in which which it is increased to allow for a faster turning response. More investigation should be done into how this output could be better utilised.

More fine-tuning to the motor controller rules could be carried out, particularly to those fired when close to the waypoint. As can be seen in figure 10, the robot stops abruptly next to the waypoint before rotating and approaching from a very small distance. Ideally the trajectory as the robot approaches would be smoother and more gradual. This could be achieved by adding more rules or members to the motor gain output set, increasing the potential resolution of the output voltages.

References

[1] Jamie Brown. Git repository for advanced control 5 assignment. `https://github.com/jamieb133/AdvancedControl5`.

A Main Simulation Code

```
%
% Main simulation with control system
%
% Author: Jamie Brown
% File: run_model.m
%
% Created: 25/02/19
%
% Changes
%
%
%
%
%-----------------------------------------------%
close all;
clear all;
clc;
%-----------------------------------------------%

%-----------------------------------------------%
%simulation config
sim_time = 15;
fs = 20; %sampling rate
fn = fs / 2; %nyquist
dT = 1 / fs;
xi = zeros(1,24); % intial state for x
xi(19) = -1; %starting x coordinate
xi(20) = -4; %starting y coordinate
LeftS = 0;
RightS = 0;
%-----------------------------------------------%

%-----------------------------------------------%
% Create Environment

max_x = 10;
max_y = 10;

Obs_Matrix = zeros(max_x/0.01,max_y/0.01);

wall = WallGeneration(-1, 1,1.2,1.2,'h');
wall2 = WallGeneration(-3, -3, -2, 2,'v');
wall3 = WallGeneration(2, 2, -3, 1,'v');
wall4 = WallGeneration(-3, -1, 4, 4,'h');

for x=1:length(wall)
    xpos = int16(wall(x,1)/0.01)+((max_x/2)/0.01);
    ypos = int16(wall(x,2)/0.01)+((max_y/2)/0.01);
    Obs_Matrix(ypos,xpos) = 1;
end

for x=1:length(wall2)
    xpos = int16(wall2(x,1)/0.01)+((max_x/2)/0.01);
    ypos = int16(wall2(x,2)/0.01)+((max_y/2)/0.01);
    Obs_Matrix(ypos,xpos) = 1;
end

for x=1:length(wall3)
    xpos = int16( (wall3(x,1)/0.01)+((max_x/2)/0.01) );
    ypos = int16( (wall3(x,2)/0.01)+((max_y/2)/0.01) );
    Obs_Matrix(ypos,xpos) = 1;
end
```

```

for x=1:length(wall4)
    xpos = int16( (wall4(x,1)/0.01)+((max_x/2)/0.01) );
    ypos = int16( (wall4(x,2)/0.01)+((max_y/2)/0.01) );
    Obs_Matrix(ypos,xpos) = 1;
end

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

%----------------------------------------------%
%setup filters
n = 2;
fCut = fn/1.5; %filter cutoff
wn = fCut / (fs / 2) %normalise cutoff frequency to nyquist
filtType = 'low';
firCoeffs = fir1(n, wn, filtType);
leftFilter = FIRFilter(firCoeffs); %filter for right motor
rightFilter = FIRFilter(firCoeffs); %filter for left motor

sensorDelay = zeros(1, fs*2); %simple moving average buffer for wall proximity
%----------------------------------------------%

%----------------------------------------------%
ObjectAvoider = readfis('ObjectAvoider.fis');
HeadingController = readfis('HeadingsToTurnCmd.fis');
MotorController = readfis('TurnCommand.fis');

targetX = 3.5;
targetY = 2.5

%change these for different scenarios

xi(19) = 0
xi(20) = -0;
%xi(24) = pi/2;
targetX = -1;
targetY = -2;


targetWaypoint = [targetX, targetY];
simpleGain = 10/pi;
Vd = 2.5; %drive voltage
motorGain = 15;

time = zeros(1, sim_time/dT);
%----------------------------------------------%


%----------------------------------------------%
% MAIN SIMULATION LOOP

for outer_loop = 1:(sim_time/dT)

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

    %obtain current reference and heading angles
    [atWaypoint, refAngle] = los_auto(xi(19), xi(20), targetWaypoint);
    headingAngle = xi(24);

    %calculate radius to target waypoint
    deltaX = xi(19) - targetX;
    deltaY = xi(20) - targetY;
    radius = sqrt(deltaX^2 + deltaY^2);

    if radius < 0.05
        %we are within tolerance of 5cm so stop
```

```
        Vl = 0;
        Vr = 0;
    else
        %obtain current distance to obstacle
        sensorOut = ObsSensor1(xi(19), xi(20), [0.2 0], xi(24), Obs_Matrix);

        %calculate wall angle and proximity
        wallAngle = atan( (sensorOut(:,2) - sensorOut(:,1)) / 0.2);
        if sensorOut(:,1) < sensorOut(:,2)
            wallProximity = sensorOut(:,1);
        else
            wallProximity = sensorOut(:,2);
        end;

        %this controller determines a desired turn command (headingCmd)
        %   based solely on reference and heading angle fuzzy input sets
        headingCmd = evalfis([refAngle, headingAngle], HeadingController);

        %take moving average value of wall proximity
        %   (allows the fuzzy motor controller to estimate whether
        %   or not it is parallel to a wall while the robot "snakes" alongside it)
        sensorDelay = circshift(sensorDelay, 1);
        sensorDelay(1) = wallProximity;
        wallProximityFiltered = mean(sensorDelay);

        wallProximityFiltered = 1;
        wallProximity = 1;

        %this controller takes a turn command from the heading controller
        % and determines the output motor voltages depending on whether or
        %   not a wall is detected or assumed to be parallel
        fuzzyOut = evalfis([headingCmd, radius, wallAngle, wallProximity, wallProximityFiltered
    ], MotorController);

        %generate coefficients for new filter cutoff frequency
        newCoeffs = fir1(n, fuzzyOut(:,3), 'low');
        leftFilter.coeffs = newCoeffs;
        rightFilter.coeffs = newCoeffs;

        %apply lowpass filter to fuzzy motor gains to smoothen
        gainLeft = leftFilter.filter(fuzzyOut(:,1));
        gainRight = rightFilter.filter(fuzzyOut(:,2));

        %apply individual voltages calculated from fuzzy controller
        if radius > 1
            %apply an additional constant drive voltage when far from waypoint
            %   and not in viscinity of a wall
            Vl = Vd + (motorGain * gainLeft);
            Vr = Vd + (motorGain * gainRight);
        else
            if wallProximity < 1
                %while in viscinity of wall, reduce drive voltage proprtionally
                Vl = (Vd * wallProximity) + (motorGain * gainLeft);
                Vr = (Vd * wallProximity) + (motorGain * gainRight);
            else
                %when close to waypoint, reduce drive voltage proportionally
                Vd * radius;
                Vl = (Vd * radius ) + (motorGain * gainLeft);
                Vr = (Vd * radius ) + (motorGain * gainRight);
            end;
        end;

        %limit the outputs to max voltage range (+- 7.4V)
        if Vl > 14.8
            Vl = 14.8;
        elseif Vl < -14.8
```

```
            Vl = -14.8;
        end;

        if Vr > 14.8
            Vr = 14.8;
        elseif Vl < -14.8
            Vl = -14.8;
        end;

    end;

    %apply calculated output voltages to motors
    Va = [Vl/2; Vl/2; Vr/2; Vr/2];
    [xdot, xi] = full_mdl_motors(Va,xi,0,0,0,0,dT);

    %euler integration
    xi = xi + (xdot*dT);

    %store variables
    xdo(outer_loop,:) = xdot;
    xio(outer_loop,:) = xi;
    VlResults(outer_loop,:) = Vl;
    VrResults(outer_loop) = Vr;
    %----------------------------------------------%


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


    %----------------------------------------------%
    %draw robot on graph for each timestep
    figure(1);
    clf; hold on; grid on; axis([-5,5,-5,5]);
    drawrobot(0.2,xi(20),xi(19),xi(24),'b');
    xlabel('y, m'); ylabel('x, m');
    plot(wall(:,1),wall(:,2),'k-');
    plot(wall2(:,1),wall2(:,2),'k-');
    plot(wall3(:,1),wall3(:,2),'k-');
    plot(wall4(:,1),wall4(:,2),'k-');
    pause(0.001);

    time(outer_loop) = outer_loop*dT;
    %----------------------------------------------%

end
%----------------------------------------------%
disp(xi(19));
disp(xi(20));
%----------------------------------------------%
%PLOTS

figure(2);
plot(xio(:,19));
title('Y Distance Travelled');
xlabel('Timesteps');
ylabel('Distance (m)');

figure(3);
plot(xio(:,20));
title('X Distance Travelled');
xlabel('Timesteps');
ylabel('Distance (m)');

figure(4);
plot(xio(:,24));
title('PSI Angle');
```

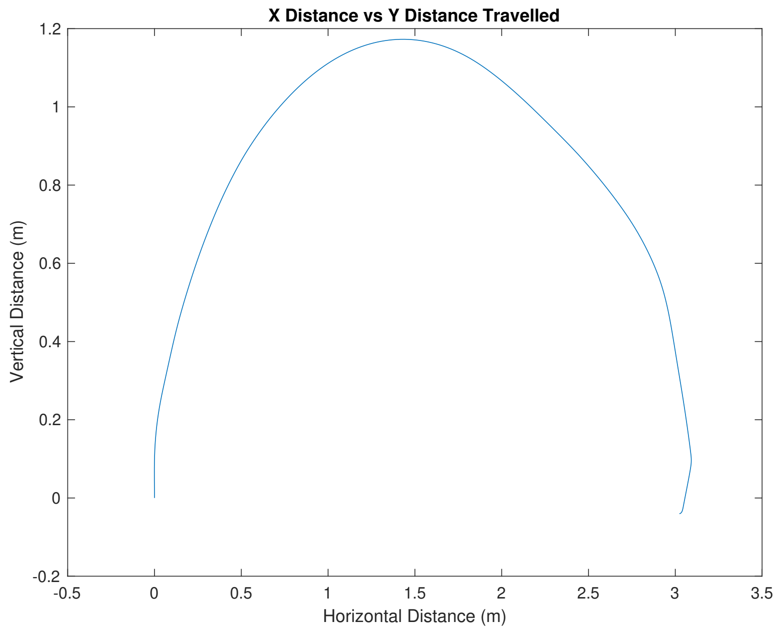
```
xlabel('Angle (rads)');
ylabel('Time (s)');

figure(5);
plot(xio(:,20),xio(:,19));
title('X Distance vs Y Distance Travelled');
xlabel('Horizontal Distance (m)');
ylabel('Vertical Distance (m)');

figure(6);
plot(VlResults(:,1));
title('Right Motor Voltage');
xlabel('Time (s)');
ylabel('Voltage (V)');

figure(7);
plot(time, xio(:,18));
title('Rotational Velocity');
xlabel('Time (s)');
ylabel('Rotional Velocity (rads/s)');

%----------------------------------------------%
```

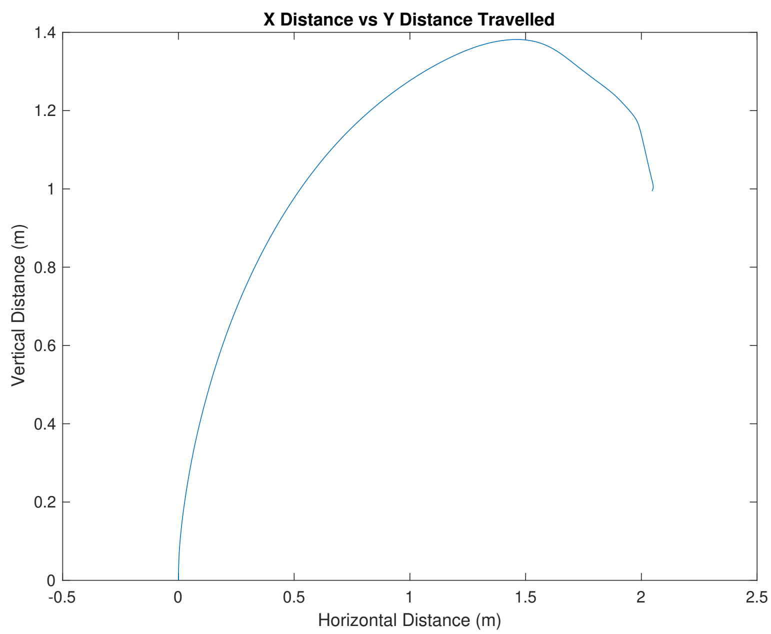
B FIRFilter Class

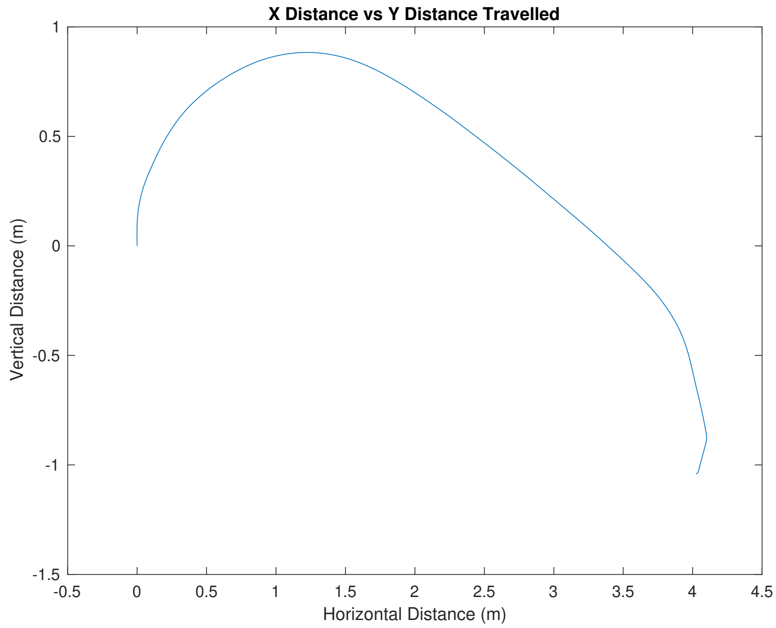
```
%
% Basic sample by sample FIR filter class
% File: FIRFilter.m
%
% Author: Jamie Brown
%
% Created: 25/02/19
%
% Changes
%
%
%
%
classdef (ConstructOnLoad = true) FIRFilter  < handle

    properties
        taps %number of filter coefficients
        coeffs %impulse response (array of coefficients)
        buffer %buffer containing previous samples
    end

    methods

        %constuctor
        function self = FIRFilter(coeffs)
            tapSize = size(coeffs)
            self.taps = tapSize(2)
            self.coeffs = coeffs
            self.buffer = ones(1, self.taps - 1)
        end

        %filters samples via convolution
        function outSample = filter(self, inSample)
            outSample = 0;

            %shift data along buffer by one sample
            self.buffer;
            self.taps;
            for count = self.taps:-1:2
                self.buffer(count) = self.buffer(count-1);
            end;

            %insert new sample
            self.buffer(1) = inSample;

            %convolve
            for count = 1 : (self.taps - 1)
                outSample = outSample + self.buffer(count) * self.coeffs(count);
            end
        end
    end
end
```

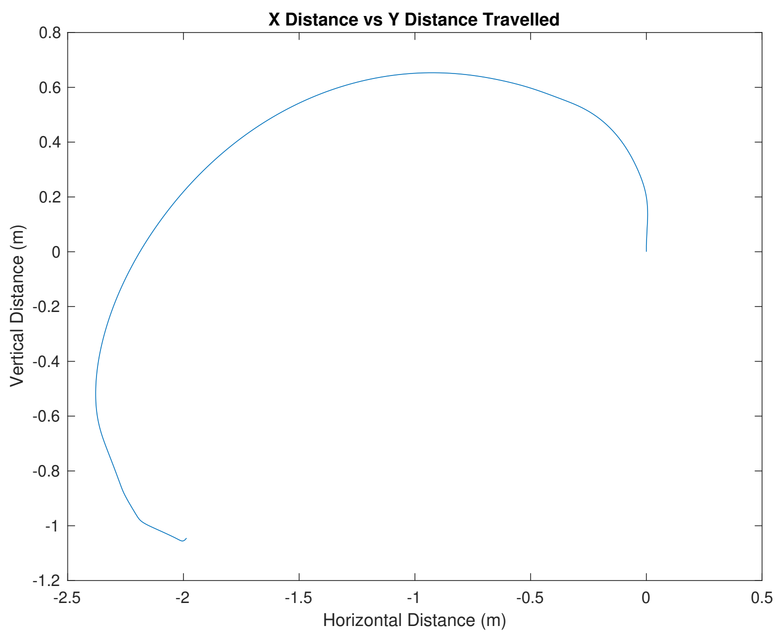
C Lab 1 Answer Sheet

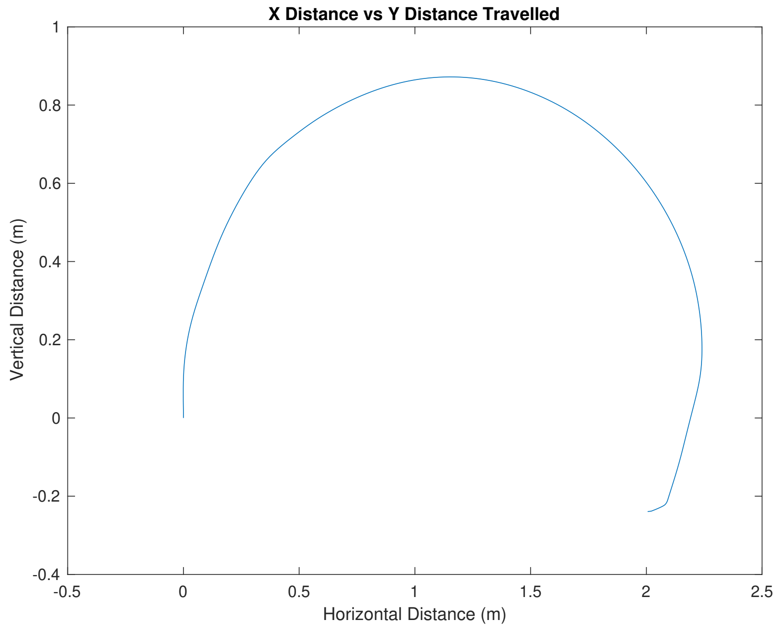
D Task 1 Plots






X Distance vs Y Distance Travelled
Vertical Distance (m)
Horizontal Distance (m)
1
0.5
0
-0.5
-1
-1.5
-0.5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5


X Distance vs Y Distance Travelled
Vertical Distance (m)
Horizontal Distance (m)
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1.2
-2.5
-2
-1.5
-1
-0.5
0
0.5



E Path Controller Rules

```
’1. If (psi_{ref} is S_{-ve}) and (psi is S_{-ve}) then (TurnCommand is FWD) (1) ’
    ’2. If (psi_{ref} is S_{-ve}) and (psi is SW) then (TurnCommand is L_{soft}) (1) ’
    ’3. If (psi_{ref} is S_{-ve}) and (psi is W) then (TurnCommand is L_{hard}) (1)  ’
    ’4. If (psi_{ref} is S_{-ve}) and (psi is NW) then (TurnCommand is L_{rot}) (1)  ’
    ’5. If (psi_{ref} is S_{-ve}) and (psi is N) then (TurnCommand is L_{rev}) (1)   ’
    ’6. If (psi_{ref} is S_{-ve}) and (psi is NE) then (TurnCommand is R_{rot}) (1)  ’
    ’7. If (psi_{ref} is S_{-ve}) and (psi is E) then (TurnCommand is R_{hard}) (1)  ’
    ’8. If (psi_{ref} is S_{-ve}) and (psi is SE) then (TurnCommand is R_{soft}) (1) ’
    ’9. If (psi_{ref} is S_{-ve}) and (psi is S_{+ve}) then (TurnCommand is FWD) (1) ’
    ’10. If (psi_{ref} is SW) and (psi is S_{-ve}) then (TurnCommand is R_{soft}) (1)’
    ’11. If (psi_{ref} is SW) and (psi is SW) then (TurnCommand is FWD) (1)          ’
    ’12. If (psi_{ref} is SW) and (psi is W) then (TurnCommand is L_{soft}) (1)      ’
    ’13. If (psi_{ref} is SW) and (psi is NW) then (TurnCommand is L_{hard}) (1)     ’
    ’14. If (psi_{ref} is SW) and (psi is N) then (TurnCommand is L_{rot}) (1)       ’
    ’15. If (psi_{ref} is SW) and (psi is NE) then (TurnCommand is L_{rev}) (1)      ’
    ’16. If (psi_{ref} is SW) and (psi is E) then (TurnCommand is R_{rot}) (1)       ’
    ’17. If (psi_{ref} is SW) and (psi is SE) then (TurnCommand is R_{hard}) (1)     ’
    ’18. If (psi_{ref} is SW) and (psi is S_{+ve}) then (TurnCommand is R_{soft}) (1)’
    ’19. If (psi_{ref} is W) and (psi is S_{-ve}) then (TurnCommand is R_{hard}) (1) ’
    ’20. If (psi_{ref} is W) and (psi is SW) then (TurnCommand is R_{soft}) (1)      ’
    ’21. If (psi_{ref} is W) and (psi is W) then (TurnCommand is FWD) (1)            ’
    ’22. If (psi_{ref} is W) and (psi is NW) then (TurnCommand is L_{soft}) (1)      ’
    ’23. If (psi_{ref} is W) and (psi is N) then (TurnCommand is L_{hard}) (1)       ’
    ’24. If (psi_{ref} is W) and (psi is NE) then (TurnCommand is L_{rot}) (1)       ’
    ’25. If (psi_{ref} is W) and (psi is E) then (TurnCommand is L_{rev}) (1)        ’
    ’26. If (psi_{ref} is W) and (psi is SE) then (TurnCommand is R_{rot}) (1)       ’
    ’27. If (psi_{ref} is W) and (psi is S_{+ve}) then (TurnCommand is R_{hard}) (1) ’
    ’28. If (psi_{ref} is NW) and (psi is S_{-ve}) then (TurnCommand is R_{rot}) (1) ’
    ’29. If (psi_{ref} is NW) and (psi is SW) then (TurnCommand is R_{hard}) (1)     ’
    ’30. If (psi_{ref} is NW) and (psi is W) then (TurnCommand is R_{soft}) (1)      ’
```

```
'31. If (psi_{ref} is NW) and (psi is NW) then (TurnCommand is FWD) (1)          '
'32. If (psi_{ref} is NW) and (psi is N) then (TurnCommand is L_{soft}) (1)      '
'33. If (psi_{ref} is NW) and (psi is NE) then (TurnCommand is L_{hard}) (1)     '
'34. If (psi_{ref} is NW) and (psi is E) then (TurnCommand is L_{rot}) (1)       '
'35. If (psi_{ref} is NW) and (psi is SE) then (TurnCommand is L_{rev}) (1)      '
'36. If (psi_{ref} is NW) and (psi is S_{+ve}) then (TurnCommand is R_{rot}) (1) '
'37. If (psi_{ref} is N) and (psi is S_{-ve}) then (TurnCommand is L_{rev}) (1)  '
'38. If (psi_{ref} is N) and (psi is SW) then (TurnCommand is R_{rot}) (1)       '
'39. If (psi_{ref} is N) and (psi is W) then (TurnCommand is R_{hard}) (1)       '
'40. If (psi_{ref} is N) and (psi is NW) then (TurnCommand is R_{soft}) (1)      '
'41. If (psi_{ref} is N) and (psi is N) then (TurnCommand is FWD) (1)            '
'42. If (psi_{ref} is N) and (psi is NE) then (TurnCommand is L_{soft}) (1)      '
'43. If (psi_{ref} is N) and (psi is E) then (TurnCommand is L_{hard}) (1)       '
'44. If (psi_{ref} is N) and (psi is SE) then (TurnCommand is L_{rot}) (1)       '
'45. If (psi_{ref} is N) and (psi is S_{+ve}) then (TurnCommand is L_{rev}) (1)  '
'46. If (psi_{ref} is NE) and (psi is S_{-ve}) then (TurnCommand is L_{rot}) (1) '
'47. If (psi_{ref} is NE) and (psi is SW) then (TurnCommand is L_{rev}) (1)      '
'48. If (psi_{ref} is NE) and (psi is W) then (TurnCommand is R_{rot}) (1)       '
'49. If (psi_{ref} is NE) and (psi is NW) then (TurnCommand is R_{hard}) (1)     '
'50. If (psi_{ref} is NE) and (psi is N) then (TurnCommand is R_{soft}) (1)      '
'51. If (psi_{ref} is NE) and (psi is NE) then (TurnCommand is FWD) (1)          '
'52. If (psi_{ref} is NE) and (psi is E) then (TurnCommand is L_{soft}) (1)      '
'53. If (psi_{ref} is NE) and (psi is SE) then (TurnCommand is L_{hard}) (1)     '
'54. If (psi_{ref} is NE) and (psi is S_{+ve}) then (TurnCommand is L_{rot}) (1) '
'55. If (psi_{ref} is E) and (psi is S_{-ve}) then (TurnCommand is L_{hard}) (1) '
'56. If (psi_{ref} is E) and (psi is SW) then (TurnCommand is L_{rot}) (1)       '
'57. If (psi_{ref} is E) and (psi is W) then (TurnCommand is L_{rev}) (1)        '
'58. If (psi_{ref} is E) and (psi is NW) then (TurnCommand is R_{rot}) (1)       '
'59. If (psi_{ref} is E) and (psi is N) then (TurnCommand is R_{hard}) (1)       '
'60. If (psi_{ref} is E) and (psi is NE) then (TurnCommand is R_{soft}) (1)      '
'61. If (psi_{ref} is E) and (psi is E) then (TurnCommand is FWD) (1)            '
'62. If (psi_{ref} is E) and (psi is SE) then (TurnCommand is L_{soft}) (1)      '
'63. If (psi_{ref} is E) and (psi is S_{+ve}) then (TurnCommand is L_{hard}) (1) '
'64. If (psi_{ref} is SE) and (psi is S_{-ve}) then (TurnCommand is L_{soft}) (1)'
'65. If (psi_{ref} is SE) and (psi is SW) then (TurnCommand is L_{hard}) (1)     '
'66. If (psi_{ref} is SE) and (psi is W) then (TurnCommand is L_{rot}) (1)       '
'67. If (psi_{ref} is SE) and (psi is NW) then (TurnCommand is L_{rev}) (1)      '
'68. If (psi_{ref} is SE) and (psi is N) then (TurnCommand is R_{rot}) (1)       '
'69. If (psi_{ref} is SE) and (psi is NE) then (TurnCommand is R_{hard}) (1)     '
'70. If (psi_{ref} is SE) and (psi is E) then (TurnCommand is R_{soft}) (1)      '
'71. If (psi_{ref} is SE) and (psi is SE) then (TurnCommand is FWD) (1)          '
'72. If (psi_{ref} is SE) and (psi is S_{+ve}) then (TurnCommand is L_{soft}) (1)'
'73. If (psi_{ref} is S_{+ve}) and (psi is S_{-ve}) then (TurnCommand is FWD) (1)'
'74. If (psi_{ref} is S_{+ve}) and (psi is SW) then (TurnCommand is L_{soft}) (1)'
'75. If (psi_{ref} is S_{+ve}) and (psi is W) then (TurnCommand is L_{hard}) (1) '
'76. If (psi_{ref} is S_{+ve}) and (psi is NW) then (TurnCommand is L_{rot}) (1) '
'77. If (psi_{ref} is S_{+ve}) and (psi is N) then (TurnCommand is L_{rev}) (1)  '
'78. If (psi_{ref} is S_{+ve}) and (psi is NE) then (TurnCommand is R_{rot}) (1) '
'79. If (psi_{ref} is S_{+ve}) and (psi is E) then (TurnCommand is R_{hard}) (1) '
'80. If (psi_{ref} is S_{+ve}) and (psi is SE) then (TurnCommand is R_{soft}) (1)'
'81. If (psi_{ref} is S_{+ve}) and (psi is S_{+ve}) then (TurnCommand is FWD) (1)'
```