Advanced Control 5 (ENG5009)
Lab Assignment

2127147b

March 21, 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 with a 0.05m radius tolerance. 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 Introduction

2 Methodology

2.1 Overview of System

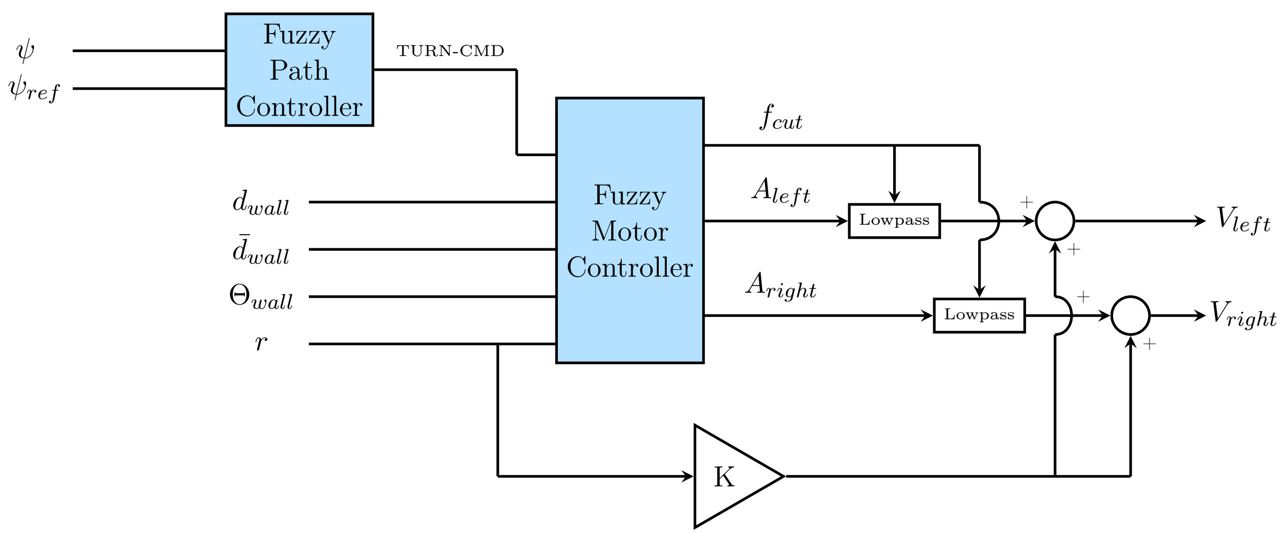

Figure 1: Block Diagram of Control System

2.2 Task 1: Waypoint Following

2.2.1 Overview

2.2.2 Fuzzy Sets

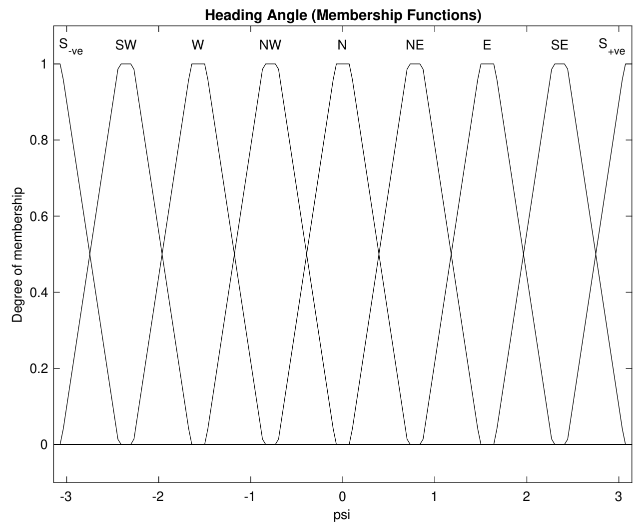

Figure 2: Membership Functions for Heading Angle Input

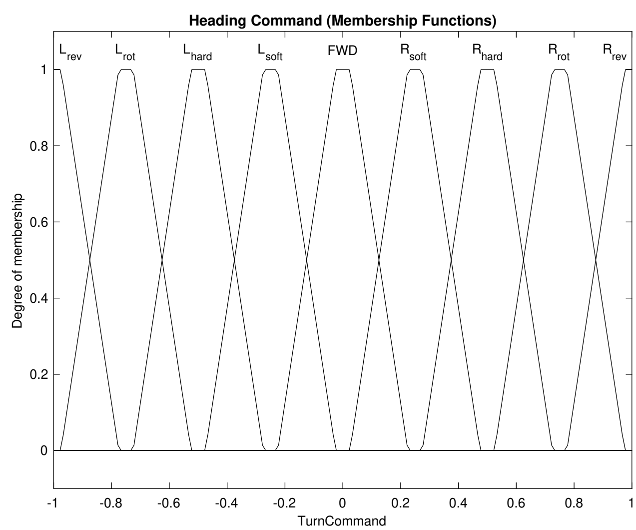

Figure 3: Membership Functions for Turn Command Output

2.2.3 Rules

Table 1: Sample of Fuzzy Logic Rules for Path Controller

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
SW	S_{-ve}	R_{soft}
SW	SW	FWD
SW	W	L_{soft}
SW	NW	L_{hard}
SW	N	L_{rot}
SW	NE	L_{rev}
SW	E	R_{rot}
SW	SE	R_{hard}
SW	S_{+ve}	R_{soft}
W	S_{-ve}	R_{hard}
W	SW	R_{soft}
W	W	FWD
W	NW	L_{soft}
W	N	L_{hard}
W	NE	L_{rot}
W	E	L_{rev}
W	SE	R_{rot}
W	S_{+ve}	R_{hard}

2.2.4 Verification

2.3 Task 2: Obstacle Avoidance

2.3.1 Overview

2.3.2 Fuzzy Sets

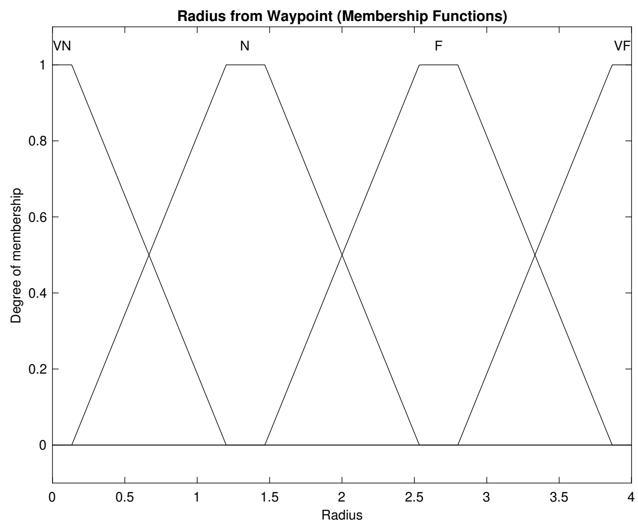

Figure 4: Membership Functions for Radius Input

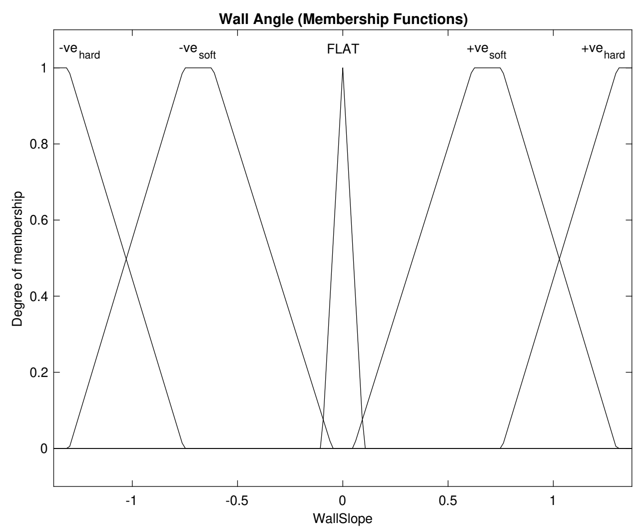

Figure 5: Membership Functions for Wall Angle Input

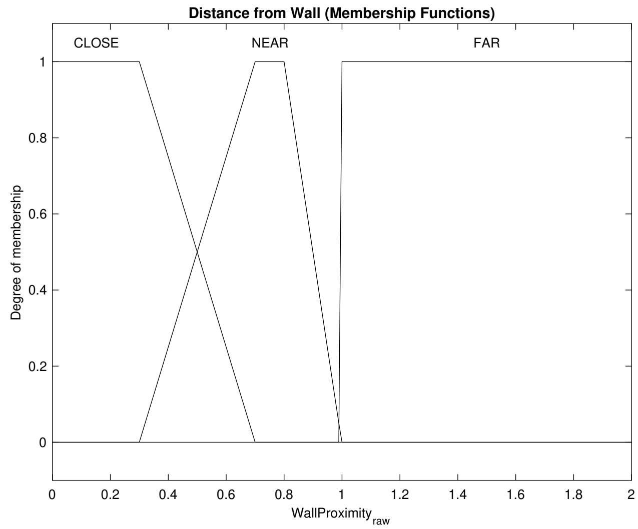

Figure 6: Membership Functions for Wall Proximity Input

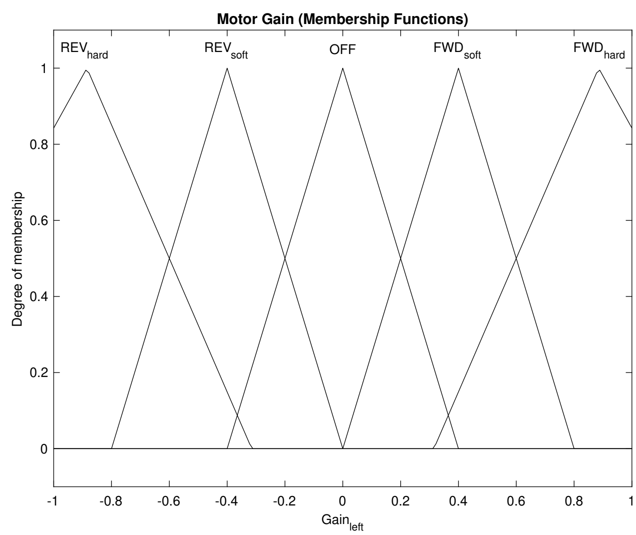

Figure 7: Membership Functions for Motor Gain Outputs

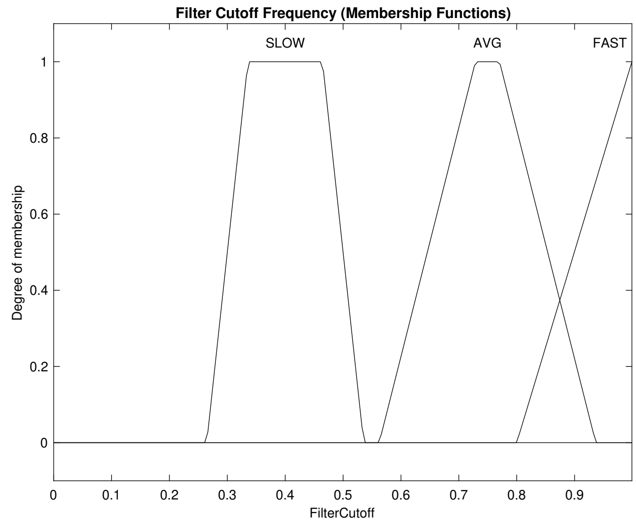

Figure 8: Membership Functions for Filter Cutoff Frequency Output

2.3.3 Rules

Table 2: Truth table of motor controller rules when outwidth the proximity of a wall

TURN CMD	r	Θ_{wall}	d_{wall}	$\bar{d}_{wall}$	A_{left}	A_{right}	ω_{cut}
FWD	!VN	X	FAR	FAR	$\mathrm{FWD_{soft}}$	$\mathrm{FWD_{soft}}$	AVG
$\mathrm{L_{soft}}$	!VN	X	FAR	FAR	OFF	$\mathrm{FWD_{soft}}$	AVG
$\mathrm{L_{hard}}$	!VN	X	FAR	FAR	$\mathrm{REV_{soft}}$	$\mathrm{FWD_{hard}}$	AVG
$\mathrm{L_{rot}}$	!VN	X	FAR	FAR	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$	AVG
$\mathrm{L_{rev}}$	!VN	X	FAR	FAR	$\mathrm{REV_{hard}}$	$\mathrm{REV_{soft}}$	AVG
$\mathrm{R_{rev}}$	!VN	X	FAR	FAR	$\mathrm{REV_{soft}}$	$\mathrm{REV_{hard}}$	AVG
$\mathrm{R_{rot}}$	!VN	X	FAR	FAR	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$	AVG
$\mathrm{R_{hard}}$	!VN	X	FAR	FAR	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{soft}}$	AVG
$\mathrm{R_{soft}}$	!VN	X	FAR	FAR	$\mathrm{FWD_{soft}}$	OFF	AVG
FWD	VN	X	FAR	FAR	$\mathrm{FWD_{soft}}$	$\mathrm{FWD_{soft}}$	AVG
$\mathrm{L_{soft}}$	VN	X	FAR	FAR	$\mathrm{REV_{soft}}$	$\mathrm{FWD_{soft}}$	AVG
$\mathrm{L_{hard}}$	VN	X	FAR	FAR	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$	AVG
$\mathrm{L_{rot}}$	VN	X	FAR	FAR	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$	AVG
$\mathrm{L_{rev}}$	VN	X	FAR	FAR	$\mathrm{REV_{hard}}$	$\mathrm{FWD_{hard}}$	AVG
$\mathrm{R_{rev}}$	VN	X	FAR	FAR	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$	AVG
$\mathrm{R_{rot}}$	VN	X	FAR	FAR	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$	AVG
$\mathrm{R_{hard}}$	VN	X	FAR	FAR	$\mathrm{FWD_{hard}}$	$\mathrm{REV_{hard}}$	AVG
$\mathrm{R_{soft}}$	VN	X	FAR	FAR	$\mathrm{FWD_{soft}}$	$\mathrm{REV_{soft}}$	AVG

2.3.4 Verification

3 Results and Testing

3.0.1 Assigned Waypoint

3.0.2 Wall Tracking

3.0.3 Perpendicular Approach

4 Discussion

4.1 Evaluation

4.2 Further Work

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 = 25;
fs = 20; %sampling rate
fn = fs / 2; %nyquist
dT = 1 / fs;
xi = zeros(1,24); % intial state for x
xi(19) = -2; %starting x coordinate
xi(20) = -1; %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) = 1;
xi(24) = pi/2;
targetX = -0.5;
targetY = 3.5;
%}

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


%----------------------------------------------%
% 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;

        %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);
    %----------------------------------------------%

end
%----------------------------------------------%

%----------------------------------------------%
%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)');

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

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