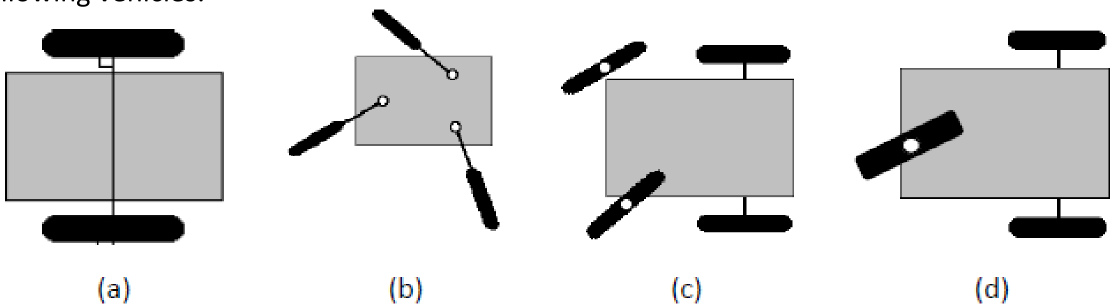
Shreyas Prabhakar
26th March 2018

# AUE 8930: AUTONOMOUS DRIVING TECHNOLOGIES

## HOMEWORK 4

**Problem 1**

(1) Give the degree of Maneuverability, degree of mobility and degree of steerability of the following vehicles.



a.
   i. Degree of Maneuverability - 2
   ii. Degree of mobility - 2
   iii. Degree of steerability - 0

b.
   i. Degree of Maneuverability - 3
   ii. Degree of mobility - 3
   iii. Degree of steerability - 0

c.
   i. Degree of Maneuverability - 2
   ii. Degree of mobility - 1
   iii. Degree of steerability - 1

d.
   i. Degree of Maneuverability - 2
   ii. Degree of mobility - 1
   iii. Degree of steerability - 1

(2) Why is dynamic control better but harder than kinematic control?

Dynamic Control is better because it more accurate than kinematic control, but it is hard because it is more responsive/sensitive than kinematic control.

(3) What are four major classes of machine learning? Why is deep neural network better than "fat" neural network?

The 4 major classes of machine learning are –

a. Classification or Categorzation
b. Clustering
c. Regression
d. Dimensionality Reduction

Deep neural network is better than fat neural network because –

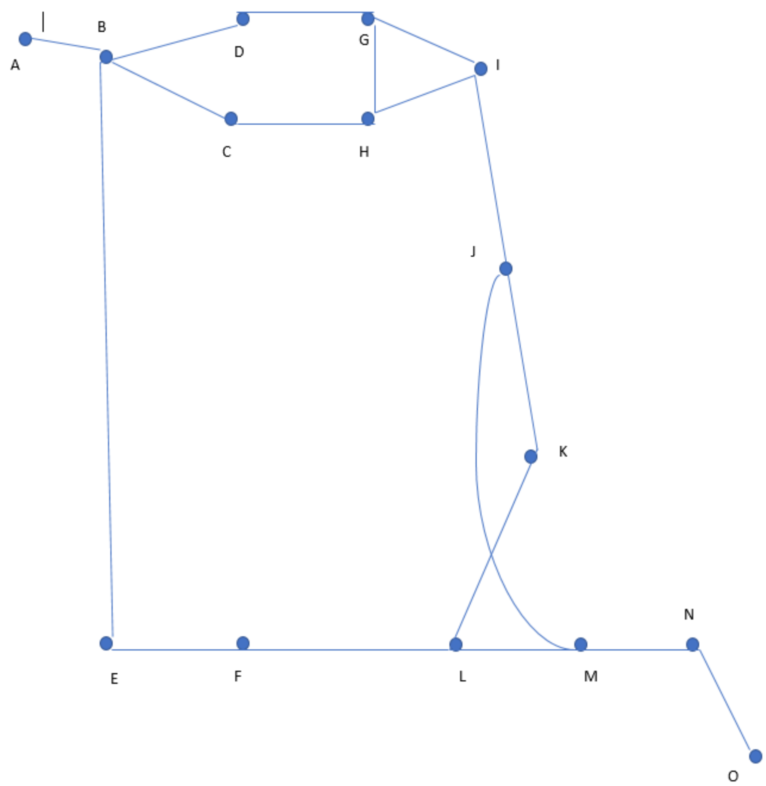
1. The error rate of deep neural netork is less after computing the same number of parameters ( like example – 7x2k network has less error compared to 1x16k network)
2. Fat neural network needs more data to train. Dep neural network can be trained by relatively little data
3. In deep neural network , modularization is automatically learned from the data.

(4) Give five application examples of machine learning in autonomous driving.

1. Object Recognition – Pedestrians, other cars etc.
2. Decisions – at stop signs, traffic lights etc.
3. Controls – Veclocity and steering angle
4. Route finding – Best route from A to B ( Ranking)
5. Vehicle Navigation in unknown environment ( Reinforcement Learning)

**Problem 2**

Build a graph to represent the map including Clemson (main campus) and CU-ICAR, where the vertices include Clemson, CU-ICAR, and intersections of roads (only considering roads: 85, 25 185/29, 123, 93, 178, 76) between Clemson and CU-ICAR, and the edges are road sections connecting each two vertices.



The vertices are

| **Vertex** | **Road Intersection** |
|---|---|
| A | Clemson University |
| B | Clemson University/76 |
| C | 76/123 |
| D | 76/93 |
| E | 76/85 |
| F | 85/178 |
| G | 178/123 |
| H | 178/93 |
| I | 93/123 |
| J | 123/25 |
| K | 123/(185/29) |
| L | 85/25 |
| M | 85/(185/29) |
| N | 85/CU-ICAR |
| O | CU-ICAR |

The edge weights are

| **Edge** | **Distance** | **Time** |
|---|---|---|
| AB | 0.7 mi | 1m |
| BC | 0.3 mi | 47s |
| BD | 1.0 mi | 2m |
| BE | 10.3 mi | 14m |
| CH | 11.3 mi | 20m |
| DG | 10.7 mi | 10 m |
| EF | 1.0 mi | 53s |
| FG | 15.6 mi | 20m |
| FL | 21.6 mi | 19m |
| GH | 1.5 mi | 3m |
| GI | 7.1 mi | 9m |
| HI | 8.5 mi | 15m |
| IJ | 6.8 mi | 11 m |
| JK | 3.9 mi | 9m |
| JM | 4 mi | 8m |
| KL | 5.6 mi | 10 m |
| LM | 1.3 mi | 1 m |
| MN | 4.7 mi | 5m |
| NO | 1.0 mi | 3m |

(1) Assign weights to each edge by the section road distance between two vertices, and give the detailed process (including every step and their cost functions) of using A* algorithm to find the route of shortest distance from CU-ICAR to Clemson.

The heuristic cost h(n) for each of the nodes are taken as the Euclidian distance between that point and the goal point (point A)

| Vertex | h(n) |
|---|---|
| A | 0 |
| B | 0.17 |
| C | 0.95 |
| D | 1.08 |
| E | 10.15 |
| F | 10.57 |
| G | 11.66 |
| H | 11.24 |
| I | 18.55 |
| J | 24.9 |
| K | 28.17 |
| L | 24.27 |
| M | 25.33 |
| N | 29.91 |
| O | 30.47 |

F(n) = g(n)+h(n)

Starting Node – O
O -> 0+30.47 = 30.47

Iteration 1

Consider O
O -> N = 1+29.91 = 30.91
Paths of Interest
ON

Iteration 2

Consider ON
ON -> M = 5.7+ 25.33 = 31.33
Paths of Interest
ONM

Iteration 3

Consider ONM
ONM -> J = 9.7 + 24.9 = 34.6
ONM -> K = 11.3 + 28.17 = 39.49
ONM -> L = 7.0 + 24.27 = 31.27
Paths of Interest
ONMJ
ONMK
ONML

Iteration 4

Consider ONML

ONML -> K = 12.6 + 28.17 = 40.77
ONML -> F = 28.6 + 10.57 = 39.17
Paths of Interest
ONMJ
ONMLK
ONMLF

Iteration 5
Consider ONMJ
ONMJ -> I = 16.5 + 18.55 = 35.05
Paths of Interest
ONMLK
ONMLF
ONMJI

Iteration 6
Consider ONMJI
ONMJI -> G = 23.6 + 11.66 = 35.28
ONMJI -> H = 25 + 11.24 = 36.24
Paths of Interest
ONMLK
ONMLF
ONMJIG
ONMJIH

Iteration 7
Consider ONMJIG
ONMJIG -> H = 25.1 + 11.24 = 36.34
ONMJIG -> D = 34.3 + 1.08 = 35.38
ONMJIG -> F = 39.2 + 10.57 = 49.77
Paths of Interest
ONMLK
ONMLF
ONMJIH
ONMJIGD

Iteration 8
Consider ONMJIGD
ONMJIGD -> B = 35.3 + 1.07 = 36.37
Paths of Interest
ONMLK
ONMLF
ONMJIH
ONMJIGDB

Iteration 9
Consider ONMJIH
ONMJIH -> C = 36.3 + 0.95 = 37.25

Paths of Interest
ONMLK
ONMLF
ONMJIGDB
ONMJIHC

Iteration 10
Consider ONMJIGDB
ONMJIGDB -> A = 36 + 1.07
Paths of Interest
ONMLK
ONMLF
ONMJIHC
ONMJIGDBA

**ONMJIGDBA is the shortest path reaching vertex A from vertex O**

(2) Assign weights to each edge by the driving time (section road distance/speed limit) between two vertices, and give the detailed process (including every step and their cost functions) of using Dijkstra's algorithm to find the route with shortest time from Clemson to CU-ICAR.

Starting Node = A
Goal Node = O

Iteration 1
Consider A
A -> B = 6s
Paths of Interest
AB

Iteration 2
Consider AB
AB -> C = 53s
AB -> D = 2m 6s
AB -> E = 14m 6s
Paths of Interest
ABC
ABD
ABE

Iteration 3
Consider ABC
ABC -> H = 20m 53s
Paths of Interest
ABD
ABE
ABCH

Iteration 4

Consider ABD
ABD -> G = 12m 6s
Paths of Interest
ABE
ABCH
ABDG

Iteration 5

Consider ABDG
ABDG -> H = 15m 6s
ABDG -> I = 21m 6s
ABDG -> F = 32m 6s
Paths of Interest
ABE
ABDGH
ABDGI
ABDGF

Iteration 6

Consider ABE
ABE -> F = 14m 59s
Paths of Interest
ABDGH
ABDGI
ABEF

Iteration 7

Consider ABEF
ABEF -> L = 33m 59s
Paths of Interest
ABDGH
ABDGI
ABEFL

Iteration 8

Consider ABDGH
ABDGH -> I = 30m 6s
Paths of Interest
ABDGI
ABEFL

Iteration 9

Consider ABDGI
ABDGI -> J = 32m 6s
Paths of Interest
ABEFL
ABDGIJ

Iteration 10

Consider ABDGIJ
ABDGIJ -> K = 41m 6s
ABDGIJ -> M = 40m 6s
Paths of Interest
ABEFL
ABDGIJK
ABDGIJM

Iteration 11

Consider ABEFL
ABEFL -> M = 34m 59s
ABEFL -> K = 43m 59s
Paths of Interest
ABDGIJK
ABEFLM

Iteration 12

Consider ABEFLM
ABEFLM -> N = 39m 59s
Paths of Interest
ABDGIJK
ABEFLMN

Iteration 13

Consider ABEFLMN
ABEFLMN -> O = 42m 59s
Paths of Interest
ABDGIJK
ABEFLMNO

Iteration 14

Consider ABDGIJK
No nodes that are already visited
Paths of Interest
ABEFLMNO

**The shortest path from vertex A to vertex O is ABEFLMNO**

**Problem 3: Motion Control**

A vehicle is driving along a straight lane which center line is represented by $y=1$. The vehicle needs to switch to an adjacent straight lane which center line is represented by $y=2$. The units are all meters. A simplified discrete vehicle model is given by

$$\begin{bmatrix} x(i+1) \\ y(i+1) \\ \theta(i+1) \end{bmatrix} = \begin{bmatrix} x(i) \\ y(i) \\ \theta(i) \end{bmatrix} + \begin{bmatrix} v * cos\theta(i) \\ v * sin\theta(i) \\ tan\varphi * \Delta t * v/L \end{bmatrix} \Delta t$$

where the vehicle baseline $L=1\ m$, the control sampling time is chosen as 0.01 second, and the vehicle speed is a constant $v=1\ m/s$.

(1) Use Pure Pursuit Method to design a lane switching controller to calculate the vehicle steering angle $\varphi$ and implement it using Matlab.

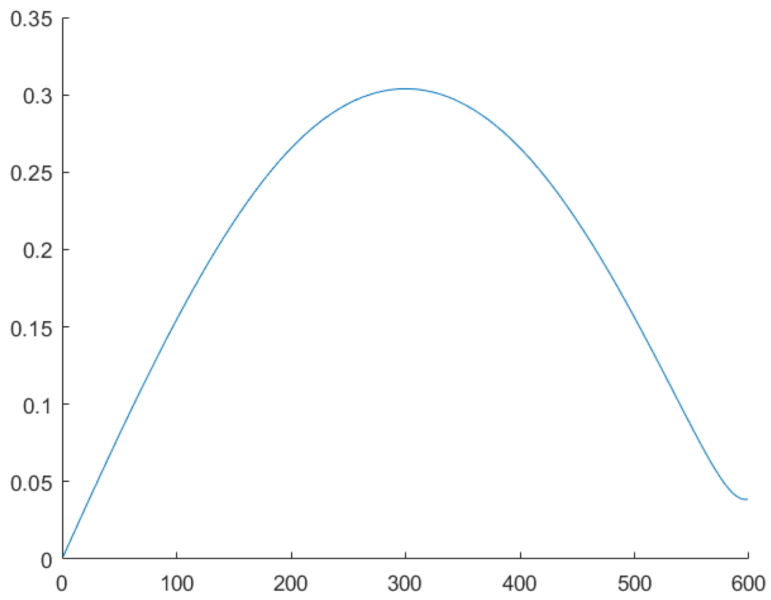
(2) Tune the control parameter $k$ to make the lane changing finished (lane tracking error < 0.01 m) within 6 seconds and the overshoot is within 0.05 m. Give the parameter and plot the desired-lane tracking errors, steering angles and vehicle orientations during the entire process.

The k was varied in the range 0 to 1 in steps of 0.01.

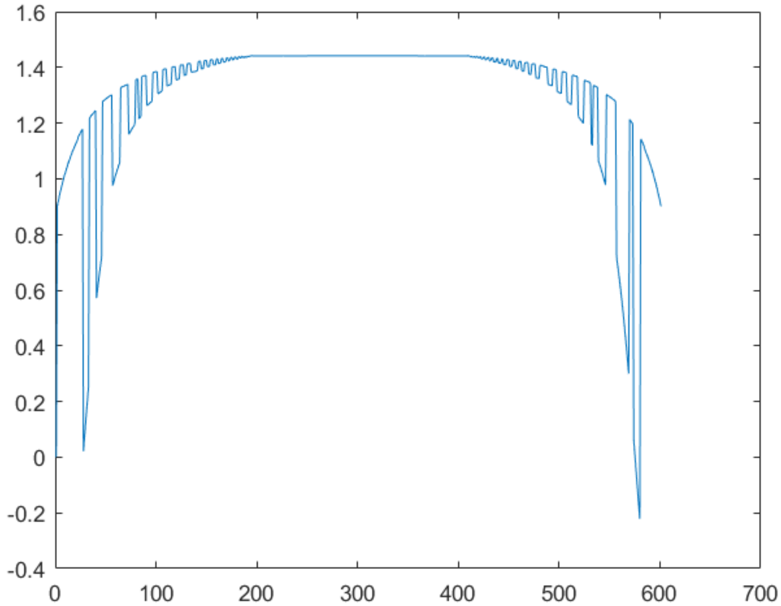
The k's with overshoot < 0.05 at time 6s were k = 0.05, 0.13, 0.26.

I have considered **k =0.26** as it had the minimum overshoot (0.0388 at 6s).

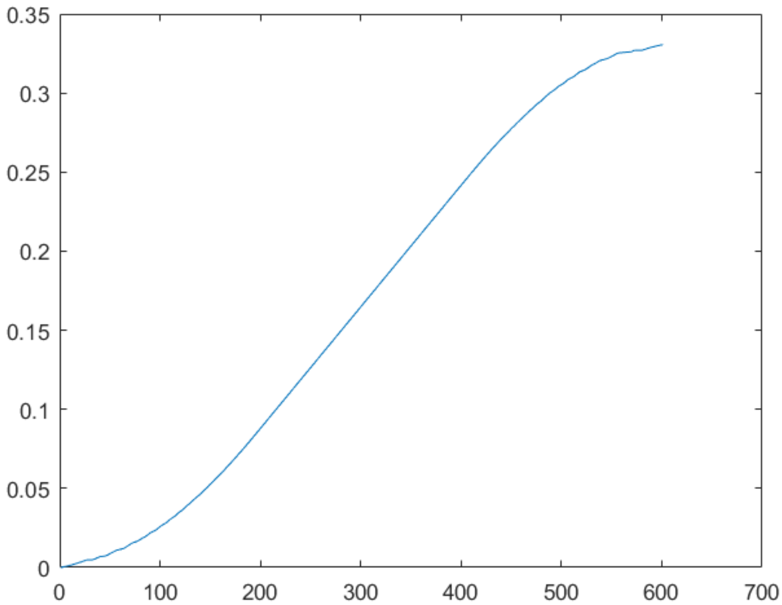
Lane tracking error at k =0.26



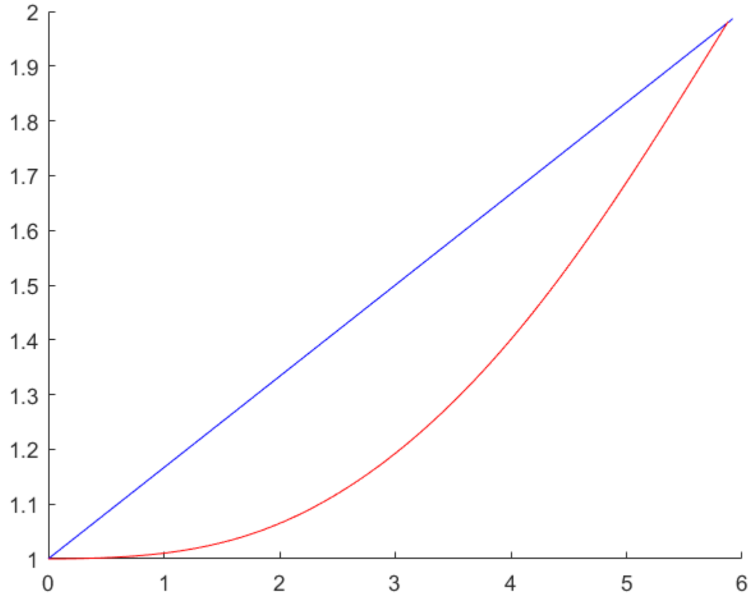
Steering Angle, phi, at k =0.26




Vehicle Orientation, theta, with k = 0.26



Vehicle actual path taken with k = 0.26 (red) and ideal path (blue)




**APPENDIX**
**MATLAB CODE –**
**Hw4.m**

```
close all
clear all

L = 1;
dt = 0.01;
v = 1;

T = 6;

tt= 0;

t = 0:dt:T+2*tt;

for i = 1:length(t)
    if t(i)<=tt
        gphi(i) = 0;
        gx(i) = t(i);
        gy(i) = 1;
    elseif t(i)>T+tt
        gphi(i) = 0;
        gx(i) = t(i);
        gy(i) = 2;
    else
        gphi(i) = atan2(1,T);
        gx(i) = gx(i-1) + cos(gphi(i))*dt;
        gy(i) = gy(i-1) + sin(gphi(i))*dt;
    end
end

x(1) = 0;
y(1) = 1;
theta(1) = 0;
```

```
% k = 0.05;
k = 0.0:0.01:1;
for j = 1:length(k)
    gp = [0;1];
    clear x y theta
    x(1) = 0;
    y(1) = 1;
    theta(1) = 0;
    for i = 2:length(t)
        alpha = atan2(gy(i)-y(i-1),gx(i)-x(i-1)) - theta(i-1);
        [phi(i), ggp] = pure_pursuit(k(j), L ,v, [gx;gy], [x(i-1),y(i-
1),theta(i-1)]);
        gp = [gp, ggp];
        x(i) = x(i-1) + (v*cos(theta(i-1))*dt);
        y(i) = y(i-1) + (v*sin(theta(i-1))*dt);
        theta(i) = theta(i-1) + (tan(phi(i))*dt*(v/L)*dt);
    end

    % figure;
    % for i = 1:length(t)
    % hold on
    % plot(gx,gy,'b');
    % plot(x(i),y(i),'k*');
    % plot(gx(i+1),gy(i+1),'r*');
    % plot(gp(1,i),gp(2,i),'g*');

    % pause(0.05);
    % hold off
    % clf
    % end

    e = sqrt((gx(t>tt&t<T+tt)-x(t>tt&t<T+tt)).^2+(gy(t>tt&t<T+tt)-
y(t>tt&t<T+tt)).^2);
    err(j) = sqrt((gx(end)-x(end)).^2 + (gy(end)-y(end)).^2);
    error(j) = max(e);
    % hold on
    % plot(e);
    % pause(1);
end

clear x y theta
x(1) = 0;
y(1) = 1;
theta(1) = 0;
k2 = k(err<=0.05);
err(err<=0.05)
for j = 1:length(k2)
    gp = [0;1];
    clear x y theta
    x(1) = 0;
    y(1) = 1;
    theta(1) = 0;
    for i = 2:length(t)
        alpha = atan2(gy(i)-y(i-1),gx(i)-x(i-1)) - theta(i-1);
        [phi(i), ggp] = pure_pursuit(k2(j), L ,v, [gx;gy], [x(i-1),y(i-
1),theta(i-1)]);
        gp = [gp, ggp];
        x(i) = x(i-1) + (v*cos(theta(i-1))*dt);
        y(i) = y(i-1) + (v*sin(theta(i-1))*dt);
```

```
        theta(i) = theta(i-1) + (tan(phi(i))*dt*(v/L)*dt);
    end

%     figure;
%     for i = 1:length(t)
%         hold on
%         plot(gx,gy,'b');
%         plot(x(i),y(i),'k*');
%         plot(gx(i),gy(i),'r*');
%         plot(gp(1,i),gp(2,i),'g*');
%         pause(0.01);
%         hold off
%         clf
%     end

    e = sqrt((gx(t>tt&t<T+tt)-x(t>tt&t<T+tt)).^2+(gy(t>tt&t<T+tt)-
y(t>tt&t<T+tt)).^2);
    err(j) = sqrt((gx(end)-x(end)).^2 + (gy(end)-y(end)).^2);
    error(j) = max(e);
%     hold on
%     plot(e);
%     pause(1);
end

k = min(k2);
clear x y theta
x(1) = 0;
y(1) = 1;
theta(1) = 0;
for i = 2:length(t)
    alpha = atan2(gy(i)-y(i-1),gx(i)-x(i-1)) - theta(i-1);
    [phi(i), ggp] = pure_pursuit(k2(j), L ,v, [gx;gy], [x(i-1),y(i-
1),theta(i-1)]);
    gp = [gp, ggp];
    x(i) = x(i-1) + (v*cos(theta(i-1))*dt);
    y(i) = y(i-1) + (v*sin(theta(i-1))*dt);
    theta(i) = theta(i-1) + (tan(phi(i))*dt*(v/L)*dt);
end
figure;plot(phi);
figure;plot(theta);
figure;
hold on
plot(gx,gy,'b');
plot(x,y,'r');
hold off
```

**pure_pursuit.m**

```
function [phi, gp] = pure_pursuit(k, L, v, g, pos)
    ld = k*v;

    d = abs(sqrt((g(1,:)-pos(1)).^2+(g(2,:)-pos(2)).^2));
    d = abs(d-ld);
    d = fliplr(d);
    [~,i] = min(d);
    i = size(g,2) -i + 1;
    alpha = atan2(g(2,i)-pos(2),g(1,i)-pos(1)) - pos(3);
%     alpha = atan2(gy(i)-y(i-1),gx(i)-x(i-1)) - theta(i-1);
    phi = atan2(2*L*sin(alpha),ld);
    gp = g(:,i);
end
```