Numerical Analysis

**Effector Controller**

**Table of contents**

1. Introduction

2. Approach

3. Software requirements

4. Implementations

5. Source code

6. Screenshots

7. Reference

**1.Introduction**

In this project we are numerically calculating the output of a controller code based on specific drive strategy. The following is the architecture of the controller and the equations we use



Controller Architecture

Equations:



BD=constant, 1025 Kg/m3

We have a proportional-integral-derivative (PID) controller which is a control loop feedback mechanism. A Kalman filter or Kalman state estimator given a state-space model of the plant and the process and measurement noise covariance data. The Kalman estimator provides the optimal solution to continuous or discrete estimation problems. An optimum control (drag compensation), physical system and a set point generation where the position set point is generated by searching the point of the path that is closest to the actual position. The velocity set point Vc is the corresponding point on the velocity profile.

Input: Path PhTarget (j), Velocity V(j), Environmental force Fenv(i), VCorridor(j).

Provided by the instructor: Example Code in Matlab (Project2.txt & Project2.m), PhTarget (j), GCorridor(j) Environmental force Fenv(i)

|  |  |  |
| --- | --- | --- |
| Name | Explanation | Data type |
| PhTarget | Target path | n x 3 matrix |
| GCorridor | Corridor (the radius of the path at location j) | n x 1 matrix |
| VTarget | Requested Speed | n x 3 matrix |
| Fenv | F Environment | n x 3 matrix |
| Kp, Ki, Kd | controller parameters | scalar |
| Grad | Gradient | n x 3 scalar matrix |
| dGrad | First derivative | n-1 x 3 matrix |
| d2Grad | Second Derivative | n-2 x 3 matrix |
| PhResp | The performed path | n x 3 matrix |
| PhError | Path error PhResp vs. PhTarget | nx1 vs n x 1 matrix |
| VResp | Velocity performed | n x 3 matrix |
| Acc | Acceleration | n x 3 matrix |



The above figure shows the overview of the code input/output. In dashed boxes are the two inputs that will be entered as files or produced by a custom function.

**2.Approach**

Operator Selected Parameters & calculations:

1. Scalar (be careful of the units!): Enter in the GUI entry Sphere radius Rs, Sphere magnetization Ms, the three controller parameters Kp, Ki, and Kd.
2. Velocity Profile V: From the GUI your code should offer the options to:
   * Enter a constant value
   * Load a file with pre-calculated values j, VTarget\_X(j), VTarget\_Y(j), VTarget\_Z(j)
   * Calculate them internally with a function you selected
3. Environmental Force Profile Fenv: From the GUI your code should offer the options to:
   * Enter a constant value
   * Load a file with pre-calculated values j, Fenv\_X(j), Fenv\_Y(j), Fenv\_Z(j)
   * Calculate them internally with a function you selected
4. Empty

Algorithm Output: gradient vector Grad(j) (i.e. Grad\_X(j), Grad\_Y(j), Grad\_Z(j)), first derivative of gradients dGrad(j) (i.e. dGrad\_X(j), dGrad\_Y(j), dGrad\_Z(i)), second derivative of gradient d2Grad(j) (i.e., d2Grad\_X(j), d2Grad\_Y(j), d2Grad\_Z(i)), object acceleration Acc(j) (i.e. Acc\_X(j), Acc\_Y(j), Acc\_Z(j)), performed PhResp(j) (i.e. PhResp\_X(j), PhResp\_Y(j), PhResp\_Z(j)), error of PhResp vs. PhTarget (i.e. PhError \_X(j), PhError\_Y(j), PhResp\_Z(j)), performed velocity VResp(j) (i.e. VResp\_X(j), VResp\_Y(j), VResp\_Z(j))

GUI: Generate a GUI to manage the operation of the software and see the results. The GUI should (1) have dedicated windows to present 2D and 3D graphs, (2) enable entering parameters, (2) loading files, (4) saving files

Algorithms: Numerical calculation of the functions, differentiation, interpolation and fitting with b-splines, error analysis.

While (sphere is navigating) do:

* Get the current position Pc and velocity Vc of the sphere
* Calculate the setpoints of the regulator (the position setpoint is the point of Ph(j) that is the closest to the actual position of the sphere, the velocity setpoint is the corresponding value in V(j)).
* Calculate the output of the PID regulator.
* Add the optimum control to the output of the PID regulator:

Calculate the motion of the sphere (from Newton’s second law).

Software Output:

1. A Study Report txt file that has the following separate line lists:
   * Comment line (entered into a line of a GUI)
   * Target Path filename
   * Velocity filename
   * Environmental Force filename
   * Controller parameters (5 lines)
   * Maximum gradient GradMax\_X, GradMax\_Y, GradMax\_Z
   * Gradient Output filename
   * Study Output filename
2. Graphs in ONE window present multiple graphs one under the other that are aligned horizontally:
3. Gradient Output file. This will have a four column spreadsheet: j, Grad\_X(j), Grad\_Y(j), Grad\_Z(j)
4. Study Output file. This will have a twenty-four column spreadsheet: j, time(j),PhTarget\_X(j),PhTarget\_Y(j),PhTarget\_Z(j),PhResp\_X(j),PhResp\_Y(j), PhResp\_Z(j), PhError\_X(j), PhError\_Y(j), PhError\_Z(j), VTarget\_X(j), VTarget\_Y(j), VTarget\_Z(j), VResp\_X(j), VResp\_Y(j), VResp\_Z(j), Grad\_X(j), Grad\_Y(j), Grad\_Z(j), dGrad\_X(j), dGrad\_Y(j), dGrad\_Z(i), d2Grad\_X(j), d2Grad\_Y(j), d2Grad\_Z(i).

**3.Software Requirements**

Software requirements:

1. Matlab

2. Python 2.7

3. Pandas

4. GUI: Tkinter

**4.Implementation**

We started implementing our code in project in C++ using concept of vectors to load and store data from csv files. However, we faced issues with data being large and we also had some storage issues. As we were looking out for other options, we observed that python is a better alternative to implement our project. Then we started implementing our project in python using pandas.

Python is more flexible for dealing with large datasets. We used pandas in python for our project. **Pandas** is a [Python](http://www.python.org/) package providing fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, **real world** data analysis in Python. Additionally, it has the broader goal of becoming **the most powerful and flexible open source data analysis / manipulation tool available in any language**. Our main ideology of this project is using the flexibility of the python language to perform our numerical calculations and analysis, but not to use the inbuilt features of it.

We are using GUI “Tkinter”. Tkinteris a Python binding to the Tk GUI toolkit. Tk is the original GUI library for the Tcl language. Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter.

We have converted the given matlab code in to python for our project. We have used pandas to read and store data from csv files. We used numpy for calculating the length of an array. We have implemented our own functions for all the other required calculations for our project.

 Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more. We have implemented our complete code for project in .ipny file in jupyter notebook. We have read data into data frames from given .csv files using pandas in python. Then we calculated velocity and other values by writing our own functions in python. Then we have implemented interpolate function and then “pchip” for extrapolate. Then we have plot the graphs and stored all the data in the program output files using data frames and concatenate function.

**5.Source code**

import pandas as pd

import pylab as P

import os,sys,csv

from decimal import \*

import numpy as np

np.seterr(divide='ignore', invalid='ignore')

import math

import matplotlib.pyplot as plt

import matplotlib as mpl

from mpl\_toolkits.mplot3d import Axes3D

pi=3.14

from tkinter import \*

class Values(Tk):

"""docstring for Values"""

def \_\_init\_\_(self, parent):

Tk.\_\_init\_\_(self,parent)

self.parent = parent

self.initialize()

def initialize(self):

self.grid()

stepOne = LabelFrame(self, text=" 1. PROJECT 2 ")

stepOne.grid(row=0, columnspan=7, sticky='W',padx=5, pady=5, ipadx=5, ipady=5)

self.Val1Lbl = Label(stepOne,text="Sphere\_Radius Rs")

self.Val1Lbl.grid(row=0, column=0, sticky='E', padx=5, pady=2)

self.Val1Txt = Entry(stepOne)

self.Val1Txt.grid(row=0, column=1, columnspan=3, pady=2, sticky='WE')

self.Val2Lbl = Label(stepOne,text="Sphere magnetization Ms")

self.Val2Lbl.grid(row=1, column=0, sticky='E', padx=5, pady=2)

self.Val2Txt = Entry(stepOne)

self.Val2Txt.grid(row=1, column=1, columnspan=3, pady=2, sticky='WE')

self.Val3Lbl = Label(stepOne,text="controller parameters Kp")

self.Val3Lbl.grid(row=2, column=0, sticky='E', padx=5, pady=2)

self.Val3Txt = Entry(stepOne)

self.Val3Txt.grid(row=2, column=1, columnspan=3, pady=2, sticky='WE')

self.Val4Lbl = Label(stepOne,text="controller parameters Ki")

self.Val4Lbl.grid(row=3, column=0, sticky='E', padx=5, pady=2)

self.Val4Txt = Entry(stepOne)

self.Val4Txt.grid(row=3, column=1, columnspan=3, pady=2, sticky='WE')

self.Val5Lbl = Label(stepOne,text="controller parameters Kd")

self.Val5Lbl.grid(row=4, column=0, sticky='E', padx=5, pady=2)

self.Val5Txt = Entry(stepOne)

self.Val5Txt.grid(row=4, column=1, columnspan=3, pady=2, sticky='WE')

'''

self.Val6Lbl = Label(stepOne,text="Velocity\_file name")

self.Val6Lbl.grid(row=5, column=0, sticky='E', padx=5, pady=2)

self.Val6Txt = Entry(stepOne)

self.Val6Txt.grid(row=5, column=1, columnspan=3, pady=2, sticky='WE')

'''

self.val1 = None

self.val2 = None

self.val3 = None

self.val4 = None

self.val5 = None

#self.val6 = None

SubmitBtn =Button(stepOne, text="Submit",command=self.submit)

SubmitBtn.grid(row=8, column=3, sticky='W', padx=5, pady=2)

def submit(self):

self.val1=self.Val1Txt.get()

if self.val1=="":

Win2=Tk()

Win2.withdraw()

self.val2=self.Val2Txt.get()

if self.val2=="":

Win2=Tk()

Win2.withdraw()

self.val3=self.Val3Txt.get()

if self.val1=="":

Win2=Tk()

Win2.withdraw()

self.val4=self.Val4Txt.get()

if self.val1=="":

Win2=Tk()

Win2.withdraw()

self.val5=self.Val5Txt.get()

if self.val1=="":

Win2=Tk()

Win2.withdraw()

'''

self.val6=self.Val6Txt.get()

if self.val1=="":

Win2=Tk()

Win2.withdraw()

'''

self.quit()

#FUNCTION DEFINITIONS

def root(x):

last\_guess= x/2.0

while True:

guess= (last\_guess + x/last\_guess)/2

if abs(guess - last\_guess) < 0.00000000000000000000000000000000000000000000000000000000000001: # example threshold

return guess

last\_guess= guess

def power(a,b):

result = 1

for \_ in range(b): # using \_ to indicate throwaway iteration variable

result \*= a

return result

hh=pd.DataFrame(columns=[0,1,2])

hh.loc[0]=0

def interp(X1,Y,xi):

type(Y)

X = [float(x) for x in X1]

X.sort(key=float)

l= len(X)

#print (l)

#print (Y[0])

i=0

# print (X[0])

for i in range(l):

#print (X[i],i)

if (X[i]==xi):

if(i==0):

return hh.loc[0]

break

else:

l1=i-1

l2=i+1

break

elif (X[i]>xi):

#print ('in elif loop')

#print (i)

l1=i-1

l2=i

#print(l1)

#print(l2)

break

i=i+1

m=(Y[l2]-Y[l1])/(X[l2]-X[l1])

yi=Y[l1]+(m\*(xi-X[l1]))

return yi

def pchip\_init(x,y):

'''

x = [float(x) for x in X1]

x.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

'''

n = len(x)

ss=list(np.array(x[1:]) - np.array(x[:-1]))

ss=np.array(ss)

yy=(y[1:]) - (y[:-1])

# Compute the slopes of the secant lines between successive points

i=0

for i in range(len(ss)):

delta=yy[i]/ss[i]

# Initialize the tangents at every points as the average of the secants

m = []

m =[0] \* n

# At the endpoints - use one-sided differences

m[0] = delta[0]

m[n-1] = delta[-1]

# In the middle - use the average of the secants

m[1:-1] = (delta[:-1] + delta[1:]) / 2.0

# Special case: intervals where y[k] == y[k+1]

# Setting these slopes to zero guarantees the spline connecting

# these points will be flat which preserves monotonicity

indices\_to\_fix = np.compress((delta == 0.0), range(n))

for ii in indices\_to\_fix:

m[ii] = 0.0

m[ii+1] = 0.0

if(delta.all()==0):

alpha=[]

beta=[]

dist=[]

dist.append(0)

tau=[]

else:

alpha = m[:-1]/delta

beta = m[1:]/delta

dist = alpha\*\*2 + beta\*\*2

b=len(dist)

if(b==1):

tau = 3.0 / root(dist[0])

else:

tau = 3.0 / root((power(dist[0],2))+(power(dist[1],2))+(power(dist[2],2)))

#over = list(dist > 9.0)

over=[i>9.0 for i in dist]

indices\_to\_fix = np.compress(over, range(n))

for ii in indices\_to\_fix:

m[ii] = tau[ii] \* alpha[ii] \* delta[ii]

m[ii+1] = tau[ii] \* beta[ii] \* delta[ii]

return m

def pchip\_eval(x,y, m, xvec):

'''

x = [float(x) for x in X1]

x.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

'''

n = len(x)

mm = 1

# Find the indices "k" such that x[k] < xvec < x[k+1]

# Create "copies" of "x" as rows in a mxn 2-dimensional vector

xx = np.resize(x,n).transpose()

xx1=xx.tolist()

#print (type(xx1))

xxx = [i for i in xx1 if i >xvec]

#print (xxx)

# Compute column by column differences

z = np.array(xxx[:-1])- np.array(xxx[1:])

#print (z)

l= len(x)

for i in range(l):

if (x[i]>xvec):

#print ('in elif loop')

#print (i)

k=i-1

break

# Collapse over rows...

# k = z.argmax(axis=0)

#print ('value')

#print (k)

#print('X')

#print (x)

# Create the Hermite coefficients

h = x[k+1] - x[k]

#print (h)

t = (xvec - x[k])/h

# Hermite basis functions

h00 = (2 \* t\*\*3) - (3 \* t\*\*2) + 1

h10 = t\*\*3 - (2 \* t\*\*2) + t

h01 = (-2\* t\*\*3) + (3 \* t\*\*2)

h11 = t\*\*3 - t\*\*2

# Compute the interpolated value of "y"

ynew = h00\*y[k] + h10\*h\*m[k] + h01\*y[k+1] + h11\*h\*m[k+1]

return ynew

def pchip\_interp(x,y,xi,args1,args2):

m = pchip\_init(x, y)

yi=pchip\_eval(x, y, m, xi)

return yi

def root(x):

last\_guess= x/2.0

while True:

guess= (last\_guess + x/last\_guess)/2

if abs(guess - last\_guess) < 0.00000000000000000000000000000000000000000000000000000000000001: # example threshold

return guess

last\_guess= guess

def power(a,b):

result = 1

for \_ in range(b): # using \_ to indicate throwaway iteration variable

result \*= a

return result

hh=pd.DataFrame(columns=[0,1,2])

hh.loc[0]=0

def interp(X1,Y,xi):

type(Y)

X = [float(x) for x in X1]

X.sort(key=float)

l= len(X)

#print (l)

#print (Y[0])

i=0

# print (X[0])

for i in range(l):

#print (X[i],i)

if (X[i]==xi):

if(i==0):

return hh.loc[0]

break

else:

l1=i-1

l2=i+1

break

elif (X[i]>xi):

#print ('in elif loop')

#print (i)

l1=i-1

l2=i

#print(l1)

#print(l2)

break

i=i+1

m=(Y[l2]-Y[l1])/(X[l2]-X[l1])

yi=Y[l1]+(m\*(xi-X[l1]))

return yi

def pchip\_init(x,y):

'''

x = [float(x) for x in X1]

x.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

'''

n = len(x)

ss=list(np.array(x[1:]) - np.array(x[:-1]))

ss=np.array(ss)

yy=(y[1:]) - (y[:-1])

# Compute the slopes of the secant lines between successive points

i=0

for i in range(len(ss)):

delta=yy[i]/ss[i]

# Initialize the tangents at every points as the average of the secants

m = []

m =[0] \* n

# At the endpoints - use one-sided differences

m[0] = delta[0]

m[n-1] = delta[-1]

# In the middle - use the average of the secants

m[1:-1] = (delta[:-1] + delta[1:]) / 2.0

# Special case: intervals where y[k] == y[k+1]

# Setting these slopes to zero guarantees the spline connecting

# these points will be flat which preserves monotonicity

indices\_to\_fix = np.compress((delta == 0.0), range(n))

for ii in indices\_to\_fix:

m[ii] = 0.0

m[ii+1] = 0.0

if(delta.all()==0):

alpha=[]

beta=[]

dist=[]

dist.append(0)

tau=[]

else:

alpha = m[:-1]/delta

beta = m[1:]/delta

dist = alpha\*\*2 + beta\*\*2

b=len(dist)

if(b==1):

tau = 3.0 / root(dist[0])

else:

tau = 3.0 / root((power(dist[0],2))+(power(dist[1],2))+(power(dist[2],2)))

#over = list(dist > 9.0)

over=[i>9.0 for i in dist]

indices\_to\_fix = np.compress(over, range(n))

for ii in indices\_to\_fix:

m[ii] = tau[ii] \* alpha[ii] \* delta[ii]

m[ii+1] = tau[ii] \* beta[ii] \* delta[ii]

return m

def pchip\_eval(x,y, m, xvec):

'''

x = [float(x) for x in X1]

x.sort(key=float)

y = [float(x) for x in Y1]

y.sort(key=float)

'''

n = len(x)

mm = 1

# Find the indices "k" such that x[k] < xvec < x[k+1]

# Create "copies" of "x" as rows in a mxn 2-dimensional vector

xx = np.resize(x,n).transpose()

xx1=xx.tolist()

#print (type(xx1))

xxx = [i for i in xx1 if i >xvec]

#print (xxx)

# Compute column by column differences

z = np.array(xxx[:-1])- np.array(xxx[1:])

#print (z)

l= len(x)

for i in range(l):

if (x[i]>xvec):

#print ('in elif loop')

#print (i)

k=i-1

break

# Collapse over rows...

# k = z.argmax(axis=0)

#print ('value')

#print (k)

#print('X')

#print (x)

# Create the Hermite coefficients

h = x[k+1] - x[k]

#print (h)

t = (xvec - x[k])/h

# Hermite basis functions

h00 = (2 \* t\*\*3) - (3 \* t\*\*2) + 1

h10 = t\*\*3 - (2 \* t\*\*2) + t

h01 = (-2\* t\*\*3) + (3 \* t\*\*2)

h11 = t\*\*3 - t\*\*2

# Compute the interpolated value of "y"

ynew = h00\*y[k] + h10\*h\*m[k] + h01\*y[k+1] + h11\*h\*m[k+1]

return ynew

def pchip\_interp(x,y,xi,args1,args2):

m = pchip\_init(x, y)

yi=pchip\_eval(x, y, m, xi)

return yi

if \_\_name\_\_ == '\_\_main\_\_':

app = Values(None)

app.title('USER VALUES:PROJECT2')

app.mainloop()

Sphere\_radius=float(app.val1)

Sphere\_density=7500

blood\_velocity=0.001; #[m/s] It is assumed to be constant for now but it is actually a function od the artery diameter. Enter a negative value to have the blood flow go against the sphere displacement.

base\_velocity=0.0002; #[m/s]

curv0=0.4; #curvature value that will divide the velocity by two

radii0=0.0015; #safety corridor radius value that will divide the velocity by two

Kp=float(app.val3)

Ki=float(app.val4)

Kd=float(app.val5)

kr=0.7

Magnetization\_sphere=float(app.val2)

Cd=0.47; #Drag coefficient of a sphere at Re=10e5.

Blood\_density=1025; #[Kg/m^3]

Q=pi\*power(Sphere\_radius,2); #Reference area for drag calculation

#Other parameters calculation

Volume\_Sphere=(4/3)\*pi\*power(Sphere\_radius,3); #m^3]

Mass\_sphere=Sphere\_density\*Volume\_Sphere; #Kg]

Moment\_sphere=Magnetization\_sphere\*Volume\_Sphere;

P1=pd.read\_csv('FileName.csv', sep=',',header=None)

P1=P1\*0.001

radii\_=pd.read\_csv('FileName2.csv', sep=',',header=None)

radii\_=radii\_\*0.001

curv1\_=pd.read\_csv('FileName3.csv', sep=',',header=None)

curv1\_.index = np.arange(1,len(curv1\_)+1)

curv2=pd.DataFrame()

curv2.loc[0,0]=0.282150

frames=[curv2,curv1\_]

curv\_ = pd.concat(frames)

curv\_.loc[1738,0]=0.01772

dP\_=pd.DataFrame(columns=['x','y','z'])

path\_distance=[]

path\_distance=[0]\*1

k=0

distance=[]

a=[]

b=[]

c=[]

i=0

for i in range(1738):

a.append(P1.loc[i+1,0]-P1.loc[i,0])

b.append(P1.loc[i+1,1]-P1.loc[i,1])

c.append(P1.loc[i+1,2]-P1.loc[i,2])

dP\_['x']=a

dP\_['y']=b

dP\_['z']=c

dP\_.columns = range(dP\_.shape[1])

P=P1.as\_matrix()

radii=radii\_.as\_matrix()

curv=curv\_.as\_matrix()

dP=dP\_.as\_matrix()

#DELETING DATAFRAMES TO SAVE MEMORY

del radii\_

del curv\_

del dP\_

del curv2

del curv1\_

#PLOTTING FIGURE 1

mpl.rcParams['legend.fontsize'] = 10

fig1 = plt.figure()

ax = fig1.gca(projection='3d')

fig1.set\_size\_inches(15,9,forward=True)

fig1.suptitle('FIGURE 1', fontsize=20)

ax.plot(P1[0], P1[1], P1[2])

ax.set\_xlabel('x (m)')

ax.set\_ylabel('y (m)')

ax.set\_zlabel('z (m)')

ax.legend()

ax.azim = 225

plt.show()

for k in list(range(1738)):

tempo=root(power(dP[k][0],2)+power(dP[k][1],2)+power(dP[k][2],2))

distance.append(tempo)

path\_distance.append(path\_distance[k]+distance[k])

V1=pd.DataFrame(columns=['x','y','z'])

V1.loc[0] = pd.Series({'x':0, 'y':0, 'z':0})

V1.columns = range(V1.shape[1])

k=0

for k in range(1738):

V1.loc[k+1,:]=base\_velocity\*(1/(1+(curv[k][0]/curv0)))\*((radii[k][0]-Sphere\_radius)/(radii0-Sphere\_radius))

V1.loc[1,:]=V1.loc[2,:]

m,n=V1.shape

V=V1.as\_matrix()

del V1

a = np.zeros(shape=(m-1,n))

V\_blood\_dir1 = pd.DataFrame(a)

b = np.zeros(shape=(m-1,1))

norm\_drag1 = pd.DataFrame(b)

drag=pd.DataFrame(columns=[0,1,2])

k=0

for k in range(1737):

V\_blood\_dir1.loc[k,:]=V[k+1]/root(power(V[k+1][0],2)+power(V[k+1][1],2)+power(V[k+1][2],2))

nn=power((root(power(V[k+1][0],2)+power(V[k+1][1],2)+power(V[k+1][2],2))),2)

norm\_drag1.loc[k,0]=-0.5\*Cd\*Blood\_density\*nn\*Q\*power((nn-blood\_velocity),2)

drag.loc[k,:]=norm\_drag1.loc[k,0]\*V\_blood\_dir1.loc[k,:]

V\_blood\_dir=V\_blood\_dir1.as\_matrix()

del V\_blood\_dir1

norm\_drag=norm\_drag1.as\_matrix()

norm\_drag1

m1=len(P)

n1=3

current\_G=pd.DataFrame(columns=[0,1,2])

current\_G.loc[0] = pd.Series({0:0, 1:0, 2:0})

current\_pos=pd.DataFrame(columns=[0,1,2])

current\_pos.loc[0]=P[0]

current\_pos

current\_vel1=pd.DataFrame(columns=[0,1,2])

current\_vel1.loc[0] = pd.Series({0:0, 1:0, 2:0})

current\_vel=current\_vel1.as\_matrix()

del current\_vel1

delta\_t=base\_velocity\*100

current\_time=0#initial time

PI=0 #initial value of the integral component of the PID regulator

plot\_var=0

error\_vel\_pos=0 #error on velocity

error\_pos\_history=[]

error\_pos\_history.append(0)#error on position

time\_history=[]

time\_history.append(0)

#time

safe\_radii\_history=[]

safe\_radii\_history.append(radii[0][0])

vel\_history=[]; #velocity of the sphere

vel\_history.append(0)

curv\_history=[]

curv\_history.append(0)

pos\_history=current\_pos

G\_history=pd.DataFrame(columns=[0,1,2])

G\_history.loc[0] = pd.Series({0:0, 1:0, 2:0})

ip=0

plot\_var=0

while ip<=20:

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# 1)Get the current positon of the sphere

#the position and velocity of the sphere is already stored in current\_pos and

#current\_vel

# 2)Calculate the setpoint of the controller

#%Find the closest point on the path

dist=[]

d=[]

d\_temp = 0

i=0

while (d\_temp<path\_distance[len(path\_distance)-1]):

yi=interp(path\_distance,P,d\_temp)

temporary=current\_pos-yi

ik = root((power(temporary.loc[0,0],2))+(power(temporary.loc[0,1],2))+(power(temporary.loc[0,2],2)))

dist.append(ik)

d\_temp = d\_temp+0.0001

d.append(d\_temp)

i=i+1

temporary.drop(0)

d\_min= 0.0

index =1

Position\_on\_path=d[0]

#Find the velocity vector setpoint

vel\_setpoint=interp(path\_distance,V,Position\_on\_path+0.001)

pos\_setpoint=interp(path\_distance,P,Position\_on\_path+0.001)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_-

# 3) Calculate the output of the controller

#calculation of the error on the position and velocity

old\_error=error\_vel\_pos

temporary2=current\_pos-pos\_setpoint

error\_pos=root(power(temporary2.loc[0,0],2)+power(temporary2.loc[0,1],2)+power(temporary2.loc[0,2],2))

error\_vel\_pos=(current\_vel-vel\_setpoint)+(kr\*(current\_pos-pos\_setpoint))

d\_error\_over\_dt=(error\_vel\_pos-old\_error)/delta\_t

#P component

PF=-Kp\*error\_vel\_pos;

#I component

PI=PI-error\_vel\_pos\*delta\_t\*Ki;

#D component

PD=-Kd\*d\_error\_over\_dt;

#4) add the optimum control

#optimum control (drag)

FF=0.5\*Cd\*Blood\_density\*(root(power(vel\_setpoint[0],2)+power(vel\_setpoint[1],2)+power(vel\_setpoint[2],2)))\*Q\*vel\_setpoint

#no - sign because the force must be applied in the velocity direction

current\_G=(1/Moment\_sphere)\*(PF+PI+FF+PD)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#5) This part simulates the movements of the sphere and calculate its position

#and velocity on the next time step

current\_pos=current\_pos+current\_vel\*delta\_t

yy1=pchip\_interp(path\_distance[1:len(path\_distance)],V\_blood\_dir,Position\_on\_path,'PCHIP','extrap')

yy2=pchip\_interp(path\_distance[1:len(path\_distance)],norm\_drag,Position\_on\_path,'PCHIP','extrap')

blood\_flow\_drag=yy1\*yy2

#t1= root(power(current\_vel[0][0],2)+power(current\_vel[0][1],2)+power(current\_vel[0][2],2))

t1=0

current\_drag=0.5\*Cd\*Blood\_density\*t1\*Q\*current\_vel+blood\_flow\_drag

current\_a=current\_G\*Moment\_sphere+current\_drag #acceleration of the sphere

current\_vel=current\_vel+current\_a\*delta\_t #velocity of the sphere

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#store results

safe\_radii=pchip\_interp(path\_distance,radii,Position\_on\_path,'PCHIP','extrap')

pos\_history=pos\_history.append(current\_pos,ignore\_index=True)

safe\_radii\_history.append(safe\_radii[0])

G\_history=G\_history.append(current\_G,ignore\_index=True)

val1=time\_history[(len(time\_history))-1]+delta\_t

time\_history.append(val1)

error\_pos\_history.append(error\_pos)

vel\_history.append(root(power(vel\_setpoint[0],2)+power(vel\_setpoint[1],2)+power(vel\_setpoint[2],2)))

yim=pchip\_interp(path\_distance,curv,Position\_on\_path,'PCHIP','extrap')

curv\_history.append(yim)

#PLOTTING

if plot\_var==20:

#figure2

mpl.rcParams['legend.fontsize'] = 10

fig2 = plt.figure()

ax = fig2.gca(projection='3d')

fig2.set\_size\_inches(15, 10, forward=True)

fig2.suptitle('FIGURE 2', fontsize=20)

ax.plot(P1.loc[:,0], P1.loc[:,1], P1.loc[:,2])

ax.set\_xlabel('x (m)')

ax.set\_ylabel('y (m)')

ax.set\_zlabel('z (m)')

ax.legend()

ax.azim = 225

plt.show()

#FIGURE 3

fig3 = plt.figure()

fig3.set\_size\_inches(15, 10, forward=True)

fig3.suptitle('FIGURE 3', fontsize=20)

plt.subplot(6,1,1) # the first subplot in the first figure

plt.plot(time\_history,list(G\_history.loc[:,0]))

plt.ylabel('Gx (T/m)')

plt.grid()

plt.subplot(6,1,2) # the second subplot in the first figure

plt.plot(time\_history,list(G\_history.loc[:,1]))

plt.ylabel('Gy (T/m)')

plt.grid()

plt.subplot(6,1,3) # the fourth subplot in the first figure

plt.plot(time\_history,list(G\_history.loc[:,2]))

plt.ylabel('Gz (T/m)')

plt.grid()

if Sphere\_radius in safe\_radii\_history:

safe\_radii\_history=safe\_radii\_history.remove(Sphere\_radius)

plt.subplot(6,1,6) # the sixth subplot in the first figure

plt.plot(time\_history,safe\_radii\_history,time\_history,error\_pos\_history)

plt.ylabel('Error (m)')

plt.xlabel('Time(s)')

plt.grid()

plt.subplot(6,1,5) # the fifth subplot in the first figure

plt.plot(time\_history,vel\_history)

plt.ylabel('Velocity (m/s)')

plt.grid()

plt.subplot(6,1,4)

plt.plot(time\_history,curv\_history)

plt.ylabel('Curvature (m^{-1})')

plt.grid()

plt.show()

else:

plot\_var=plot\_var+1

ip=ip+1

G\_history.to\_csv('project2:gradient\_output()')

G\_history.to\_csv('project3:gradient\_output')

with open('project2:gradient\_output()', 'w') as outcsv:

writer = csv.writer(outcsv)

writer.writerow(["j","Grad\_X(j)", "Grad\_Y(j)", "Grad\_Y(j)"])

for i in range(len(G\_history)):

writer.writerow([i,G\_history.loc[i,0],G\_history.loc[i,1],G\_history.loc[i,2]])

df1=pd.DataFrame()

df2=pd.DataFrame()

df3=pd.DataFrame()

df1['j']=list(range(1739))

df2['time']=time\_history

df1['PhTarget\_X(j)']=P1.loc[:,0]

df1['PhTarget\_Y(j)']=P1.loc[:,1]

df1['PhTarget\_Z(j)']=P1.loc[:,2]

df2['PhResp\_X(j)'] =pos\_history.loc[:,0]

df2['PhResp\_Y(j)'] =pos\_history.loc[:,1]

df2['PhResp\_Z(j)'] =pos\_history.loc[:,2]

df2['PhError\_X(j)'] =error\_pos\_history

df1['VTarget\_X(j)'] =V[:,0]

df1['VTarget\_Y(j)'] =V[:,1]

df1['VTarget\_Z(j)'] =V[:,2]

df3['VResp\_X(j)'] =V\_blood\_dir[:,0]

df3['VResp\_Y(j)'] =V\_blood\_dir[:,1]

df3['VResp\_Z(j)'] =V\_blood\_dir[:,2]

df2['Grad\_X(j)'] =G\_history.loc[:,0]

df2['Grad\_Y(j)'] =G\_history.loc[:,1]

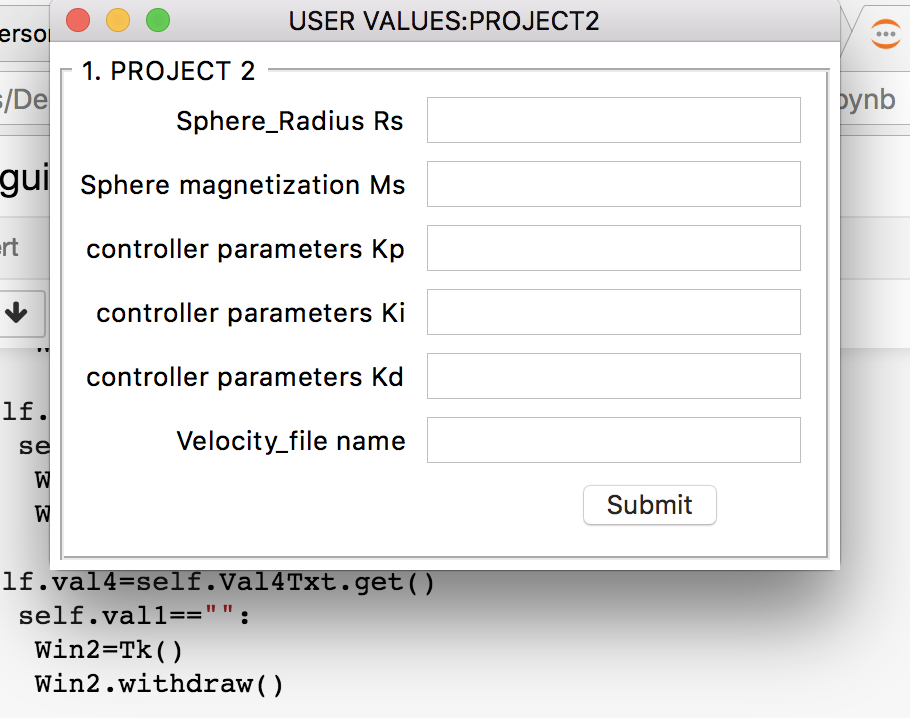
df2['Grad\_Z(j)'] =G\_history.loc[:,2]

df4=pd.concat([df1,df2,df3], axis=1)

df4.to\_csv('software\_ouput')

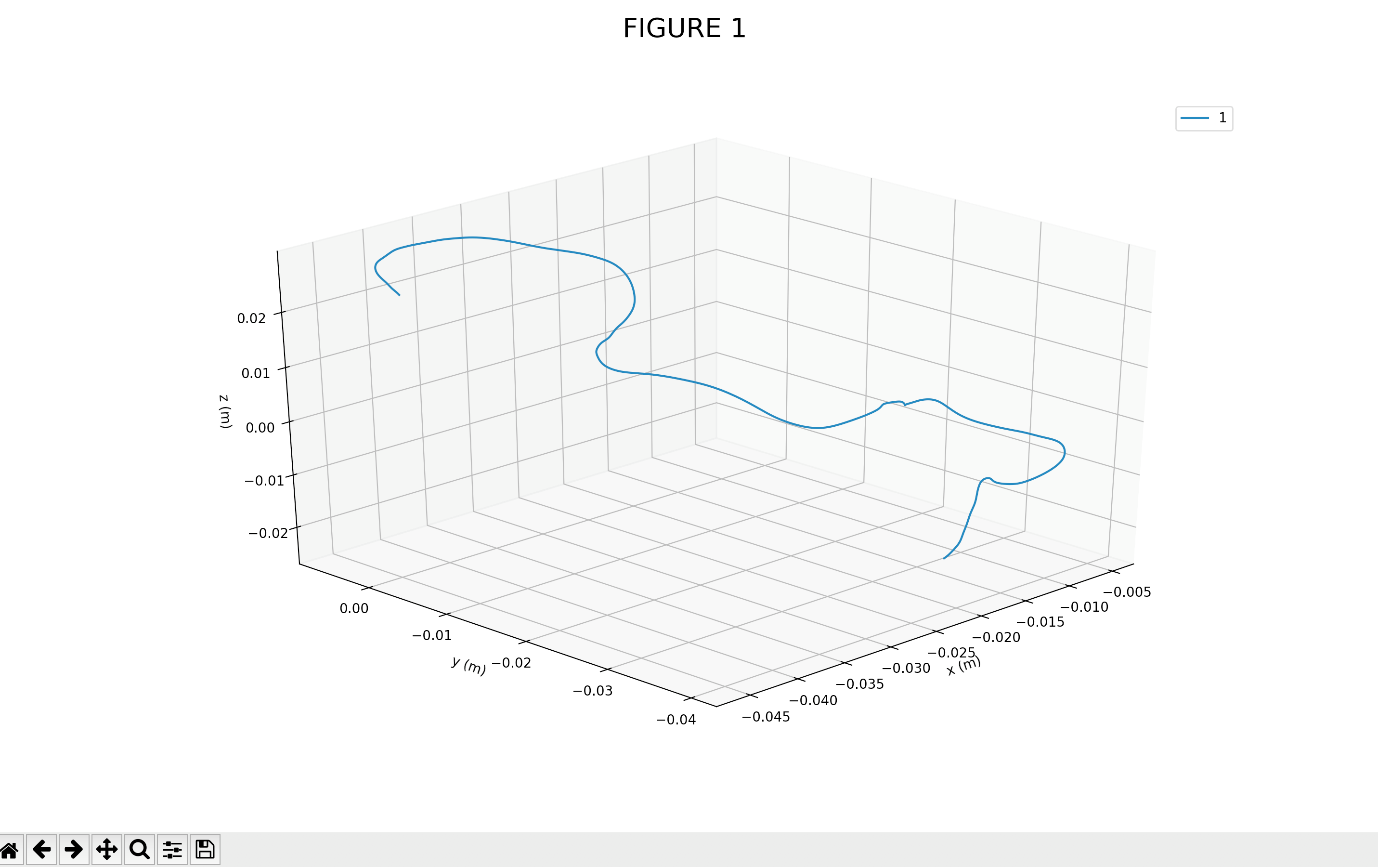
**6.Screenshots**

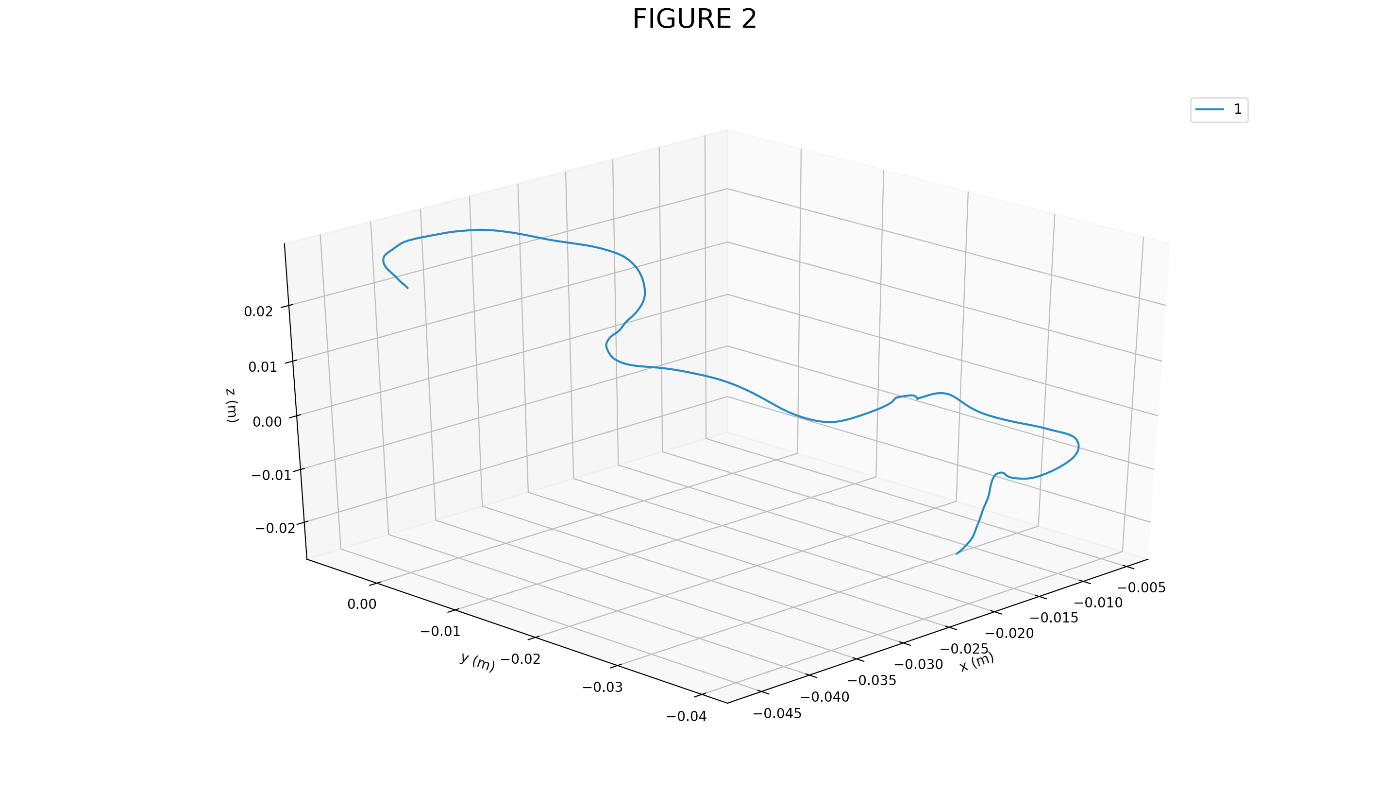
GUI snippets

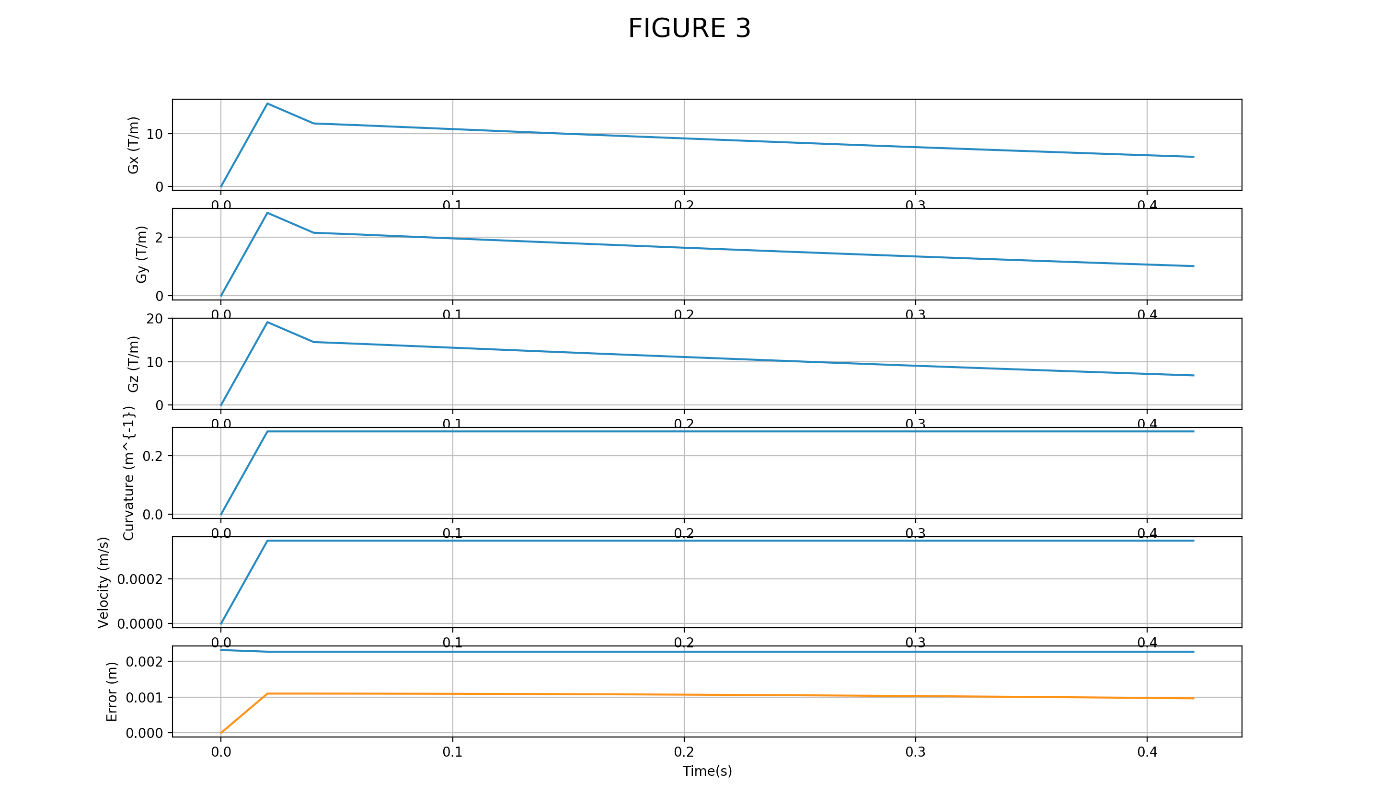


The basic snippet where we are taking the input values

Output Figures







Graph for time slots vs particular inputs such as Gx, Gy, Gz, Curvature, Velocity and Error.

**7.References**

1.[http://stackoverflow.com/questions/29595357/reading-mat-file-using-c-how- to-read-cell-structure-properly](http://stackoverflow.com/questions/29595357/reading-mat-file-using-c-how-%20%20%20to-read-cell-structure-properly)

2.<https://www.mathworks.com/downloads/web_downloads/download_release?release=R2017a>

3.http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read\_csv.html