Establishing Communication Link using Control systems

A Project Report

submitted by

ANIRUDH R

in partial fulfilment of requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

May 2018

THESIS CERTIFICATE

This is to certify that the thesis titled **Establishing Communication Link using Control systems**, submitted by **Anirudh R**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Harishankar Ramachandran
Research Guide
Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 19 June 2018

ACKNOWLEDGEMENTS

I would like to express my profound gratitude to my project guide, Dr. Harishankar Ramachandran, for giving me the opportunity to work under him, on this project. His vast knowledge, outlook towards research, patience and willingness to help, was instrumental in helping me complete my project.

I would also like to thank my friends, my parents and the faculty at IIT Madras for being a great source of motivation and encouragement.

Anirudh R
EE14B009
Student
Department of Electrical Engineering
IIT-Madras, 600 036

ABSTRACT

KEYWORDS: Snell's law , Brent's root finding algorithm , Control system

The aim of this project is to establish a communication link between a plane and a submarine (located underwater). There are many parameters related to both plane and submarine, that influence the link such as velocity , depth and equation of ocean wave. In order to provide a practical, real world solution, the system has to be modelled as a control system.

The contributions of this thesis include :

Task 1 : Given position of plane, ocean equation and beam direction, find the incident point , reflected and transmitted beams.

Task 2 : Given error in position of link from submarine, find appropriate change in angle of transmission of laser beam

The methods used and the results and graphs derived are shown in the forthcoming sections.

NOTATION

A	Amplitude of ocean wave
λ	Wavelength
T	Time period
K	Wave Number
ω	Angular frequency of wave
μ/n	Refractive index
r	relative refractive index
v	Velocity
θ	Various angles like incidence angle etc.

TABLE OF CONTENTS

CHAPTER 1

THEORY PREREQUISITES

During the course of solving the problem, a number of prerequisites(in theory) were required, which are listed here.

1.1 Snell's law

Snell's law is a formula used to describe the relationship between the angles of incidence and refraction, when referring to light or other waves passing through a boundary between two different isotropic media, such as water, glass, or air.

Snell's law states that the ratio of the sines of the angles of incidence and refraction is equivalent to the ratio of phase velocities in the two media, or equivalent to the reciprocal of the ratio of the indices of refraction:

$$sin\theta_2/sin\theta_1 = v_2/v_1 = n_1/n_2$$

1.2 Triangle law of vector addition

If 2 vectors acting simultaneously on a body are represented both in magnitude and direction by 2 sides of a triangle taken in an order then the resultant(both magnitude and direction) of these vectors is given by 3rd side of that triangle taken in opposite order.

1.3 Brent's method

Brent's method is a root-finding algorithm combining the bisection method, the secant method and inverse quadratic interpolation. It has the reliability of bisection but it can be as quick as some of the less-reliable methods. The algorithm tries to use the potentially fast-converging secant method or inverse quadratic interpolation if possible, but it falls back to the more robust bisection method if necessary.

Algorithm behind Brent's method is as follows :

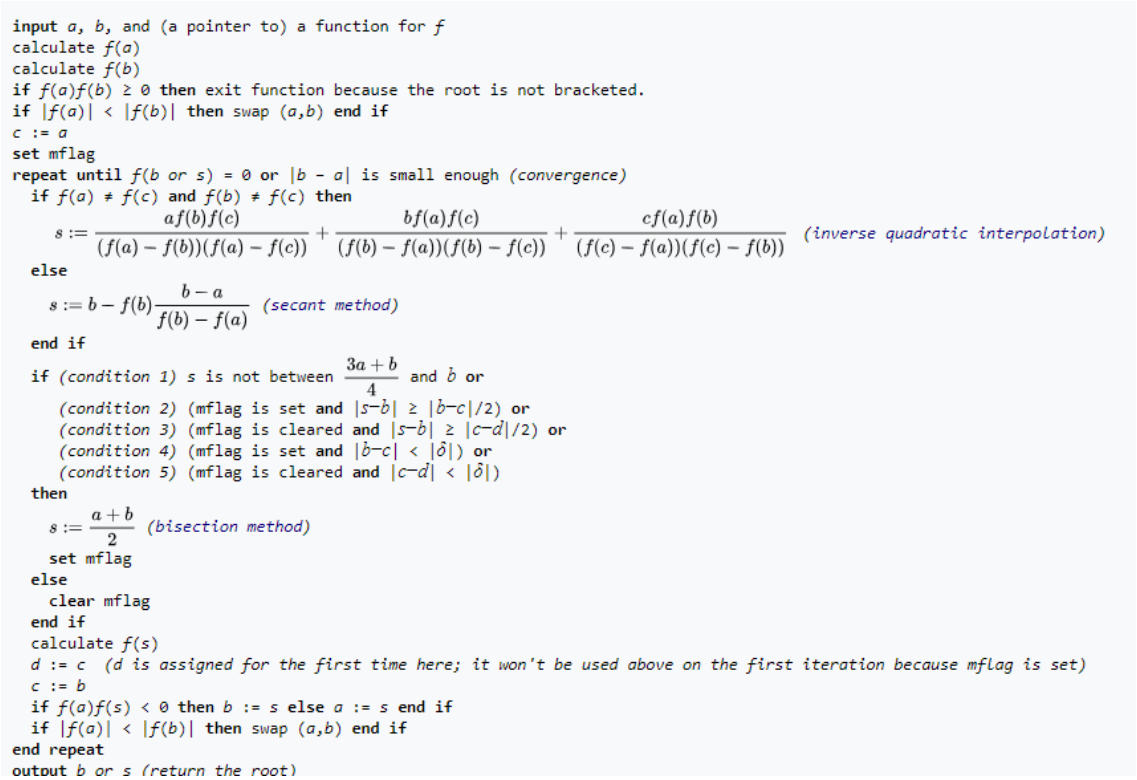


Figure 1.1: The above image is an illustration of the sequence Brent's algorithm uses to find the root of a function over many iterations. Ref : Wikipedia (Link added in Bibiliography)

1.4 PI controller

A PI controller is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PI controller continuously calculates an error value *e(t)* as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional and integral terms (denoted P, I respectively) which give the controller its name.

In this model,

- **P** is proportional to the current value of the error *e(t)*. For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor **K**. Using proportional control alone in a process with compensation such as temperature control, will result in an error between the setpoint and the actual process value, because it requires an error to generate the proportional response. If there is no error, there is no corrective response

- Term I accounts for past values of the error and integrates them over time to produce the I term. For example, if there is a residual error after the application of proportional control, the integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. When the error is eliminated, the integral term will cease to grow. This will result in the

proportional effect diminishing as the error decreases, but this is compensated for by the growing integral effect.

The balance of these effects is achieved by "loop tuning" (discussed in detail later) to produce the optimal control function. The tuning constants are shown below as "K" and must be derived for each control application, as they depend on the response characteristics of the complete loop external to the controller.

Defining *u(t)* as the controller output, the final form of the PID algorithm is :

$$U(t) = K_p e(t) + K_i \int_0^T e(t)dt + K_d \frac{de}{dt}$$

where K_p is the proportional gain, K_i is the integral gain and K_d is is the derivative gain. Equivalently, the transfer function in the Laplace domain of the PID controller is

$$L(s) = K_p + \frac{K_i}{s} + K_d s$$

where s is the complex frequency.

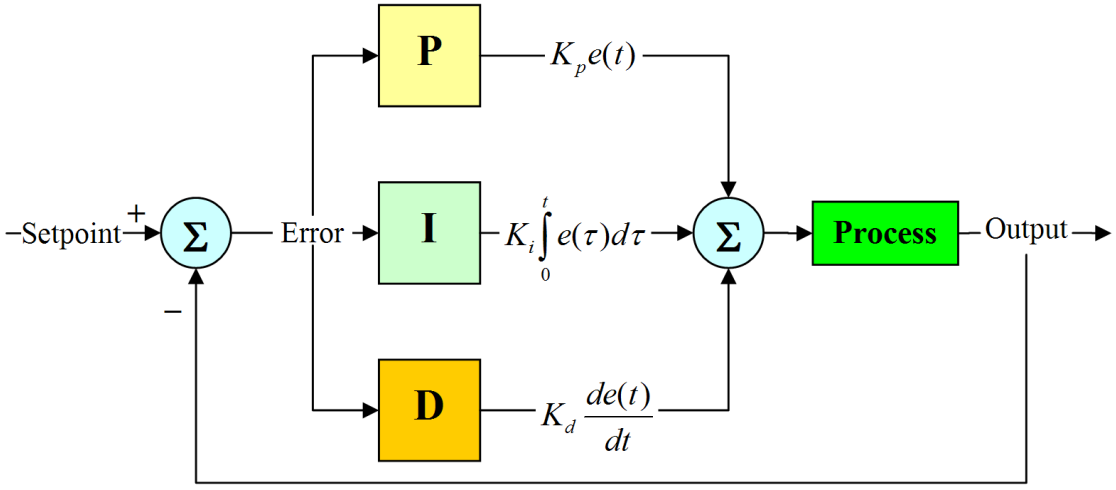


Figure 1.2: Block diagram of a PID controller

CHAPTER 2

TASK 1

2.1 Methodology

Problem statement for this task : Laser is launched from a plane. Given location of plane P = (P_x , P_z), unit vector in direction of laser $\hat{l}$ and the wave shape *Asin(ωt- Kx)*, find launch direction of laser on plane so that it reaches the submarine.

Assumptions :

1. Assume plane and submarine are stationary (no motion relative to each other).

2. This is a 1-D problem. Y Coordinate is assumed to be 0.

Let incident point be Q = (Q_x , Q_z) be the point where laser beam intersects the horizontal i.e. if the sea were a plane surface.

Also, let the z component of unit vector in direction of laser be l_z.

Using Triangle law of vectors, we can arrive at the following result :

$$Q = P - \frac{P_z}{l_z}\hat{l}$$

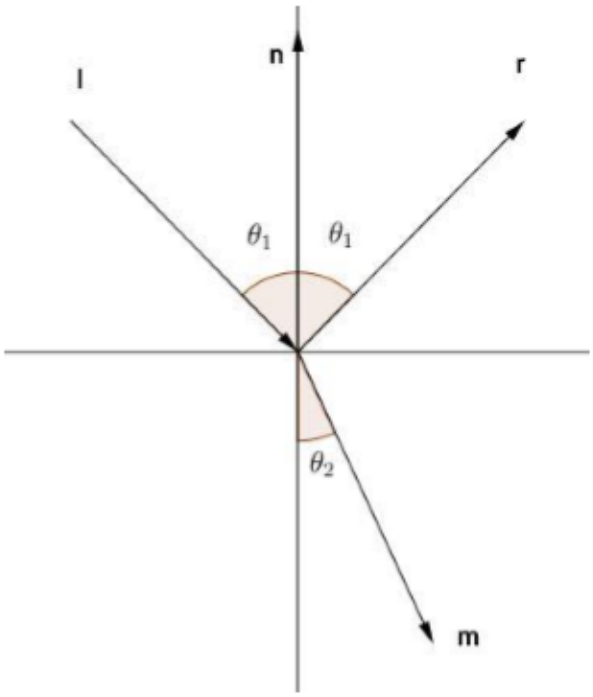


Figure 2.1: Diagram of problem statement where **l** is the direction of laser and **m** is direction of refracted wave

At a given instant of time *t* , we can Brent's method to find the incident point on the ocean wave. As mentioned earlier, Brent's method requires a bracketing interval to find the root of a function which will be provided by the amplitude of the wave **A**. Since the incident point resides on the wave, Q ∈ [-A,A]. Now we devise a function such that its root gives us the incident point. The function at a given instant of time t will be :

$$g(z) = z - A\sin(k(P_x + \frac{z - P_z)l_x}{l_z} - \omega t)$$

Applying this will get us the incident point.

$$Q = P + \frac{z - P_z}{l_z}\hat{l}$$

We now have the incident beam and point of incidence. In order to find the reflected and transmitted beams, we need to find the equation of normal at point of incidence.

$$N = \frac{-df}{dx} = -A\cos(kx - \omega t)$$

$$\hat{n} = \frac{N}{|N|}$$

The directions of the transmitted vector ($\hat{t}$) and reflected vector ($\hat{r}$) are found by :

$$\hat{t} = r_{12}\hat{l} + (r_{12}\cos(\theta_i) - \sqrt{1 - r_{12}(1 - \cos\theta_i)})$$

$$\hat{r} = 1 + 2\cos\theta_i\hat{n}$$

θ_{i} = incident angle = $\cos(-\hat{n}.\hat{l})$

2.2 Results

Running brent's algorithm for 10 iterations at each instant of time using ***brentq()*** function, we get the following outputs :

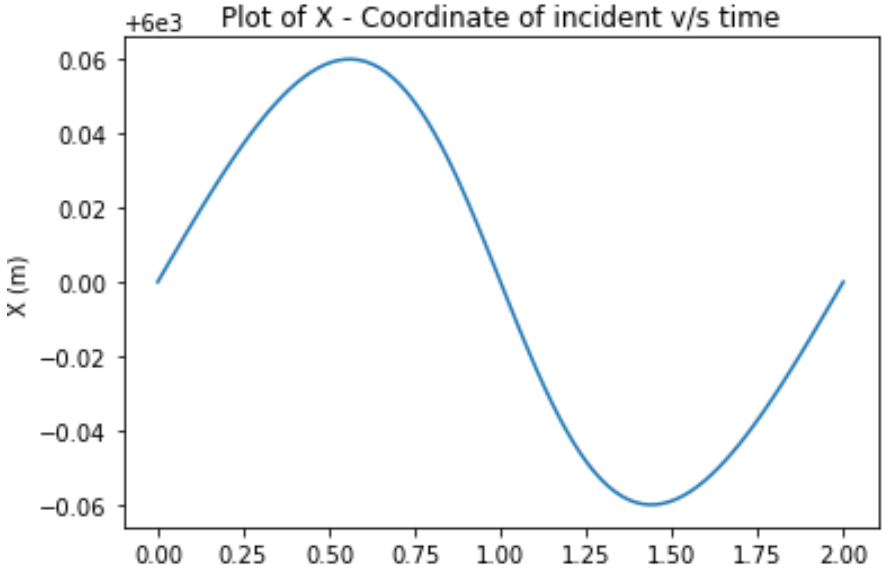


Figure 2.2: Plot of X - Coordinate of incident v/s time

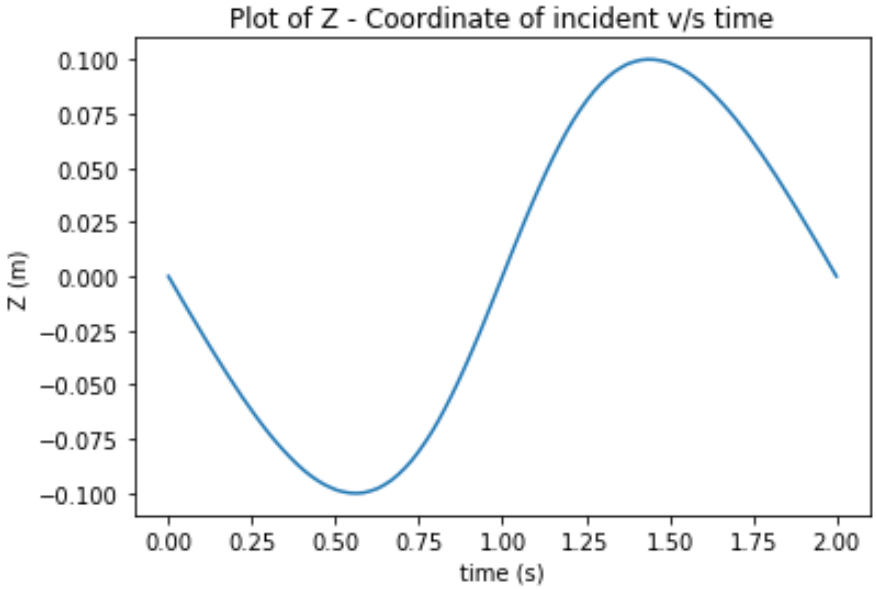


Figure 2.3: Plot of Z - Coordinate of incident v/s time

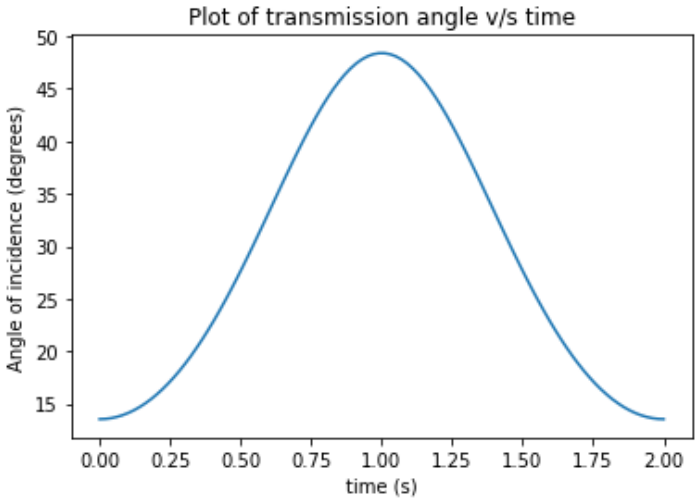


Figure 2.4: Variation of transmission angle with time

```
Unit normal Vector , Unit refracted Vector , Unit reflected vector
[-0.2997168   0.95402822] [ 0.46282672 -0.88644877] [-0.06831884  0.99766354]
[-0.29932677  0.95415066] [ 0.46273351 -0.88649743] [-0.06750311  0.99771906]
[-0.29816237  0.95451517] [ 0.46245521 -0.88664264] [-0.06506824  0.99788082]
[-0.29621603  0.95512097] [ 0.46198993 -0.88688517] [-0.06099947  0.9981378 ]
[-0.29348068  0.95596501] [ 0.4613358  -0.88722561] [-0.05528394  0.99847067]
[-0.28996357  0.95703768] [ 0.46049431 -0.88766265] [-0.04793966  0.99885023]
[-0.28566718  0.95832889] [ 0.45946572 -0.8881955 ] [-0.03897572  0.99924016]
[-0.28055647  0.95983752] [ 0.45824118 -0.88882789] [-0.02832434  0.99959879]
[-0.27466332  0.96154046] [ 0.4568277 -0.8895552] [-0.0160589   0.99987105]
[-0.26796887  0.96342757] [ 0.45521999 -0.890379  ] [-0.00214886  0.99999769]
[-0.26044502  0.96548868] [ 0.45341028 -0.89130192] [0.0134529  0.99990951]
[-0.25214489  0.96768949] [ 0.45141011 -0.8923166 ] [0.03062247 0.99953102]
[-0.24298213  0.97003076] [ 0.44919709 -0.89343269] [0.04952136 0.99877306]
[-0.23305301  0.97246403] [ 0.44679256 -0.89463758] [0.06993045 0.99755187]
[-0.22226146  0.9749871 ] [ 0.44417094 -0.89594206] [0.09202245 0.99575693]
[-0.2106861   0.97755377] [ 0.44134852 -0.89733577] [0.1156069  0.99329504]
[-0.19825906  0.98014965] [ 0.43830541 -0.89882611] [0.1407874  0.99003985]
[-0.18503334  0.98273224] [ 0.43505063 -0.90040599] [0.16741584 0.98588637]
[-0.17102978  0.98526586] [ 0.43158478 -0.90207238] [0.1954046  0.98072271]
[-0.15627532  0.98771353] [ 0.42790938 -0.90382164] [0.22464869 0.97443982]
[-0.1407522   0.99004486] [ 0.42401407 -0.9056556 ] [0.25512525 0.96690801]
[-0.12449549  0.99222017] [ 0.41990084 -0.90756999] [0.28670034 0.95802031]
[-0.10754856  0.99419983] [ 0.41557316 -0.90955976] [0.31921928 0.94768088]
[-0.0899634   0.99594507] [ 0.41103609 -0.91161907] [0.35250737 0.93580904]
[-0.0717436   0.99742311] [ 0.40628152 -0.91374795] [0.38647718 0.92229897]
[-0.0530722   0.99859068] [ 0.40134793 -0.91592567] [0.42070764 0.90719628]
[-0.03385368  0.9994268 ] [ 0.39620013 -0.91816418] [0.45529115 0.89034261]
[-0.01429018  0.99989789] [ 0.39088189 -0.92044084] [0.48978068 0.87184568]
```

Figure 2.5: Final output obtained for Normal vector $\hat{n}$,Refracted vector $\hat{t}$,Reflected vector

$\hat{r}$

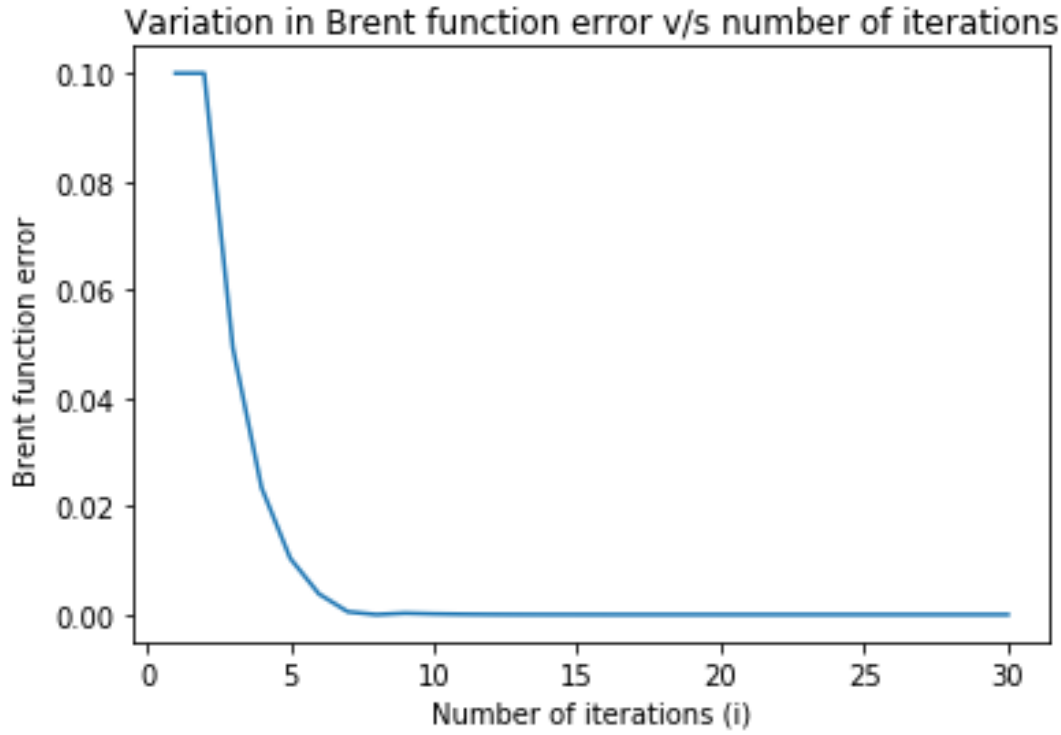


Figure 2.6: Variation of error in Brent function until it converges versus number of iterations

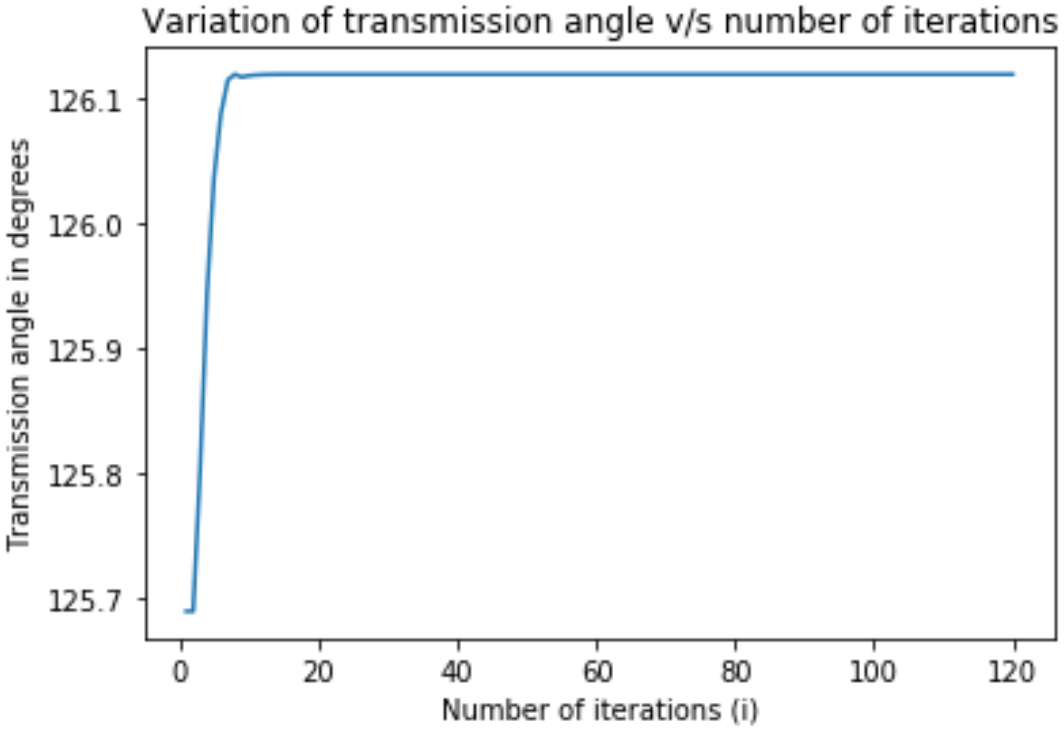


Figure 2.7: Variation of transmission angle until it converges versus number of iterations

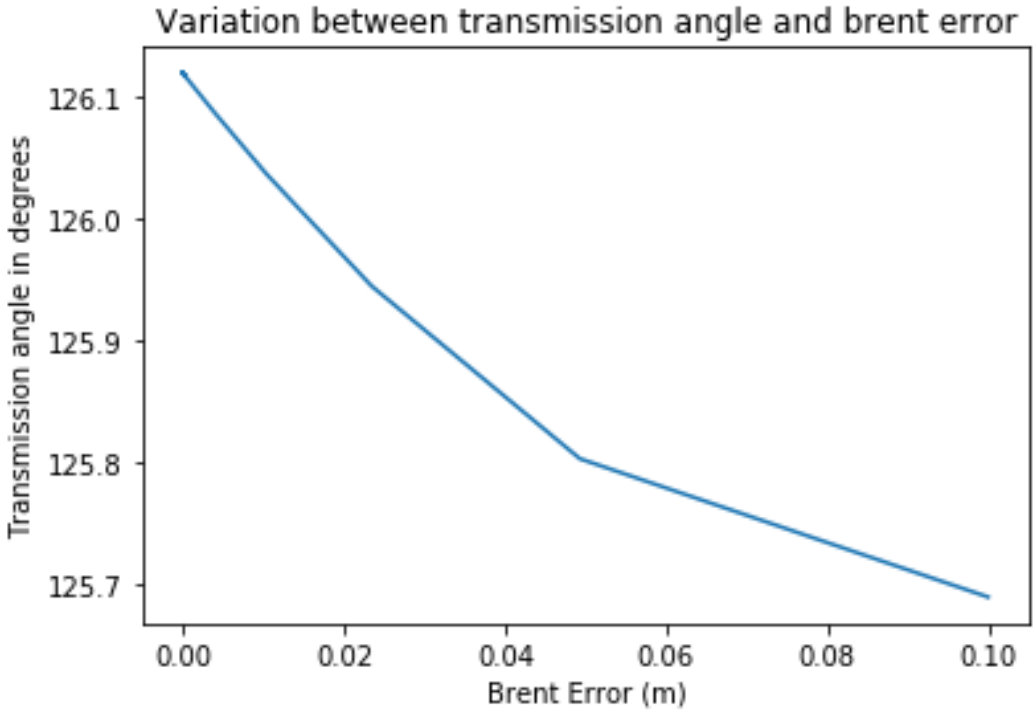


Figure 2.8: Variation of transmission angle with Brent Error

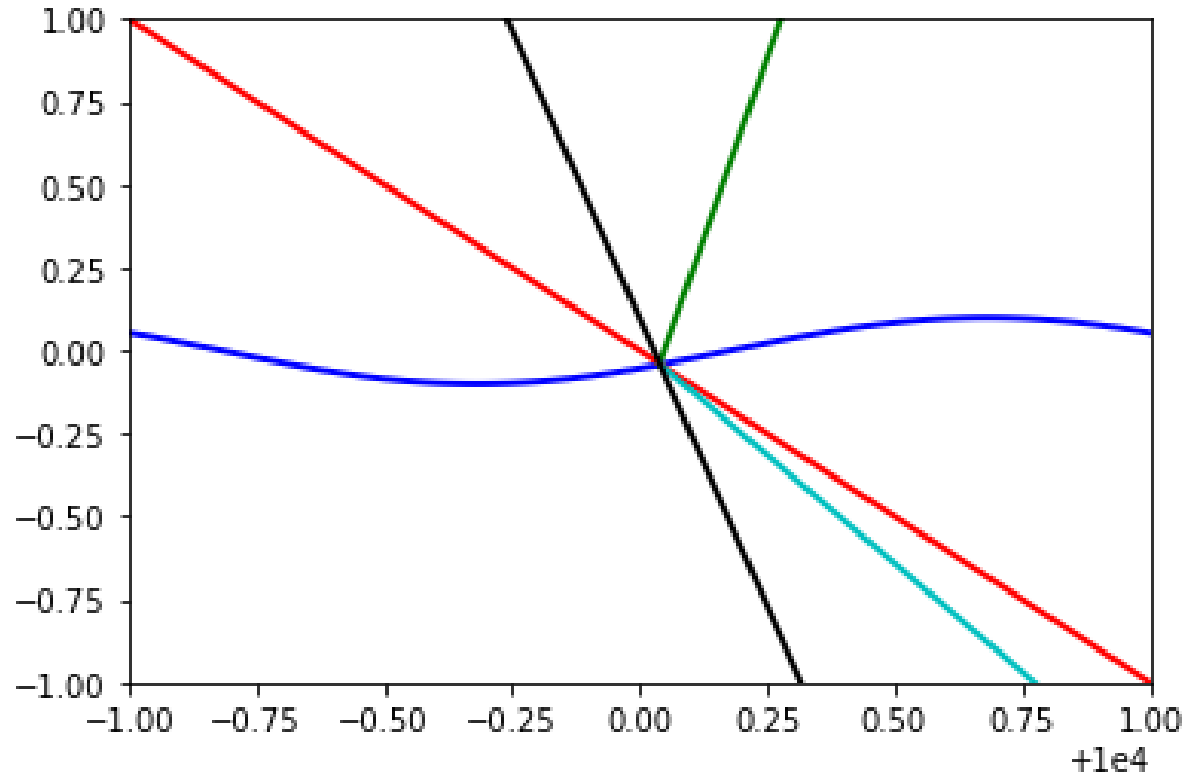


Figure 2.9: Pictorial representation of how the incident wave reaches ocean wave and then is split into reflected wave and refracted wave

Color Key for the above pic :

- Red : Incident laser wave
- Blue : Ocean Wave
- Black : Normal at point of incidence
- Indigo : Refracted wave
- Green : Reflected wave

CHAPTER 3

TASK 2

3.1 Methodology

Problem statement for this task : Finding relation between laser transmission angle and error in submarine position.

Assumptions :

- Assume this problem is solved in 1 D only. This means Y coordinate can be assumed to be 0
- Assume that one way link has already been extablished. We have to main it now
- Plane is moving while ocean and submarine are at rest
- Submarine communicates error in position to plane
- Depth of submarine is known

3.1.1 Implementation using Brentq function

We know that the plane is travelling at a velocity V_x. Once the plane receives error from plane, it will have to re calibrate the transmission angle and will resend the laser beam based on new information.

Let the position of plane be $P = (P_x + V_x t , P_y)$.

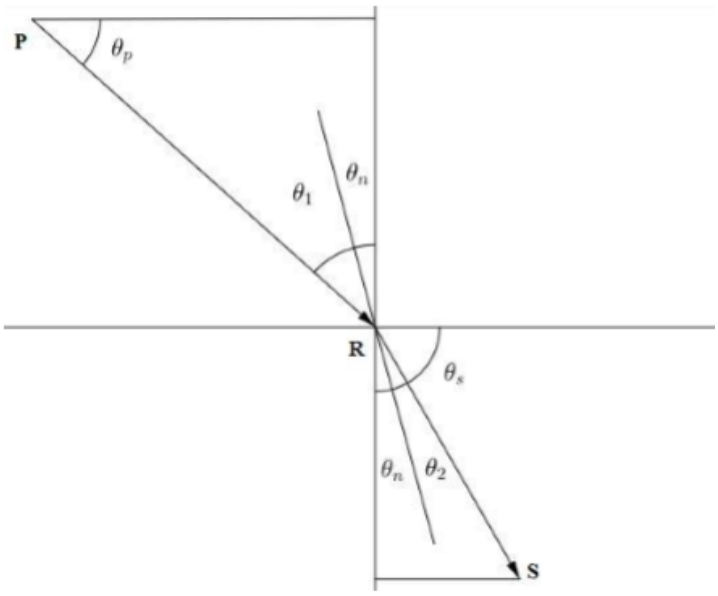


Figure 3.1: Pictorial representation of this problem

According to snell's law,

$$sin\theta_1/sin\theta_1 = n_2/n_1 = r_{21}$$
$$\cos\theta_1 d\theta_1 = r_{21}\cos\theta_1 d\theta_1$$
$$d\theta_1 = -d\theta_p - d\theta_n$$

From the geometry of the above diagram, we can also derive certain other conclusions :

$$d\theta_s = -d\theta_2 - d\theta_n$$
$$dS_x = V_x dt + \frac{S_y d\theta_s}{\sin^2\theta_s}$$

Substituting some of our earlier results into the equation for dS_x, we get the following relation :

$$dS_x = V_x dt + \frac{S_y d\theta_n(\sin(\theta_p+\theta_n) - r_{21}cos\theta_2)}{\sin^2\theta_s r_{21}\cos\theta_2} + (\frac{S_y(\sin(\theta_n+\theta_p) - r_{21}cos\theta_2 P_y\sin^2\theta_p)}{\sin^2\theta_s r_{21}\cos\theta_2})d\theta_p$$

In the above equation, $V_x dt$ (Distance travelled by the plane in that infinitesimal amount of time) shows the influence of time on the transmission angle. **t** includes $t_{travel} = 12.5\mu s$ (one way trip) and also $t_{computation}$ for brent function. We can use the second part of the equation to negate the effect of the wave and bring back the error to zero.

From this we can formulate our control equation :

$$d\theta_p = \frac{error}{\frac{S_y(\sin(\theta_n+\theta_p) - r_{21}cos\theta_2 P_y\sin^2\theta_p)}{\sin^2\theta_s r_{21}\cos\theta_2}}$$

This is taken as ultimate gain $\mathbf{K_u}$ for our control system.

We need to find θ_n. However, this can be done using the methodology described in task 1, since we know position of plane and direction of incident wave. Therefore, we can find incident point and transmitted angle also.

Thus, new error can be found and then, that will be relayed to plane.

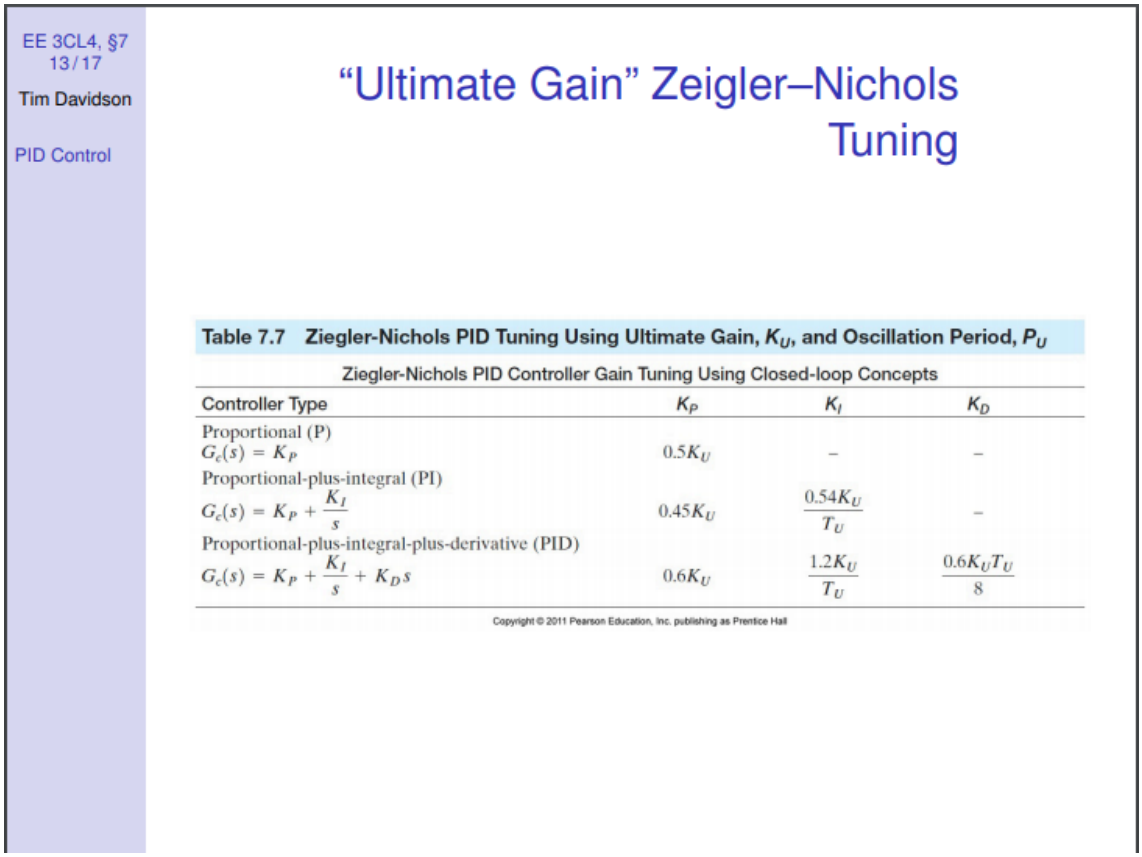
Modelling the control system

We know the overall equation of this control system in terms of its ultimate gain.

However, inorder to solve it we need to model it as a PI controller [Only then will the system be stable and steady state error will be less than given threshold].

To model it as a PI controller, we need to know $\mathbf{K_p}$ and $\mathbf{K_i}$.

We should use **Ziegler-Nichols method** to tune the system and hence find the relation between the above parameters and ultimate gain.



Therefore, once we have found these parameters, we can succesfully find the solution.

3.2 Results

Variations of Submarine error is plotted against time for different plane velocities.

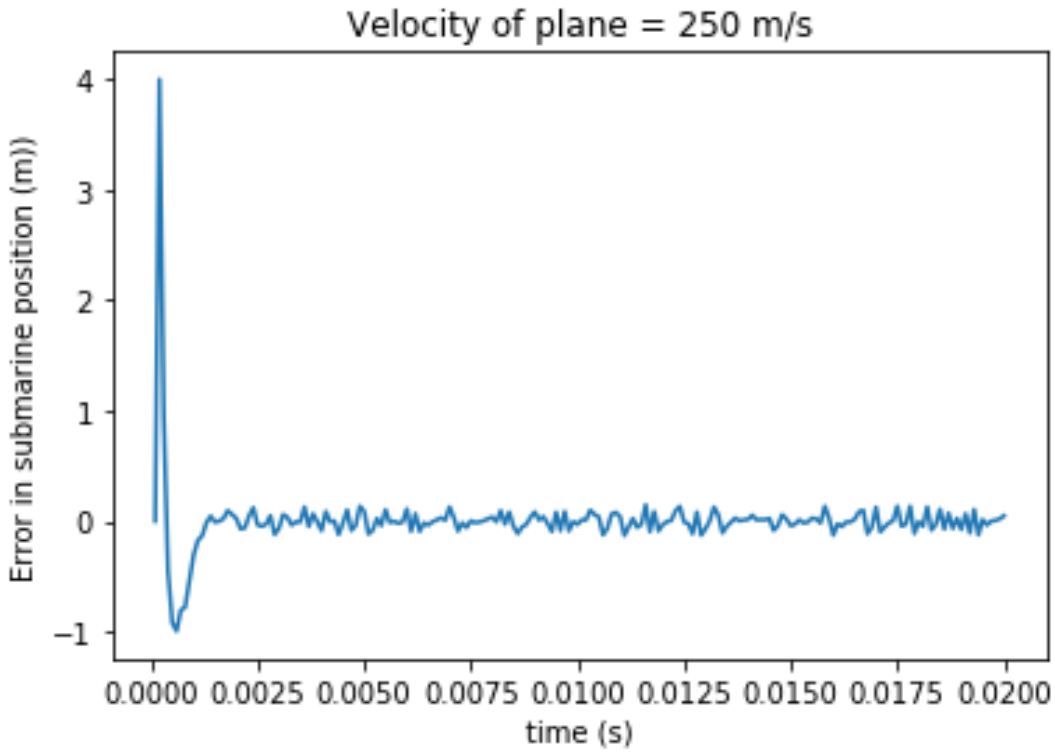


Figure 3.2: Variation of Error in submarine position S_x when velocity of plane V_x is 250

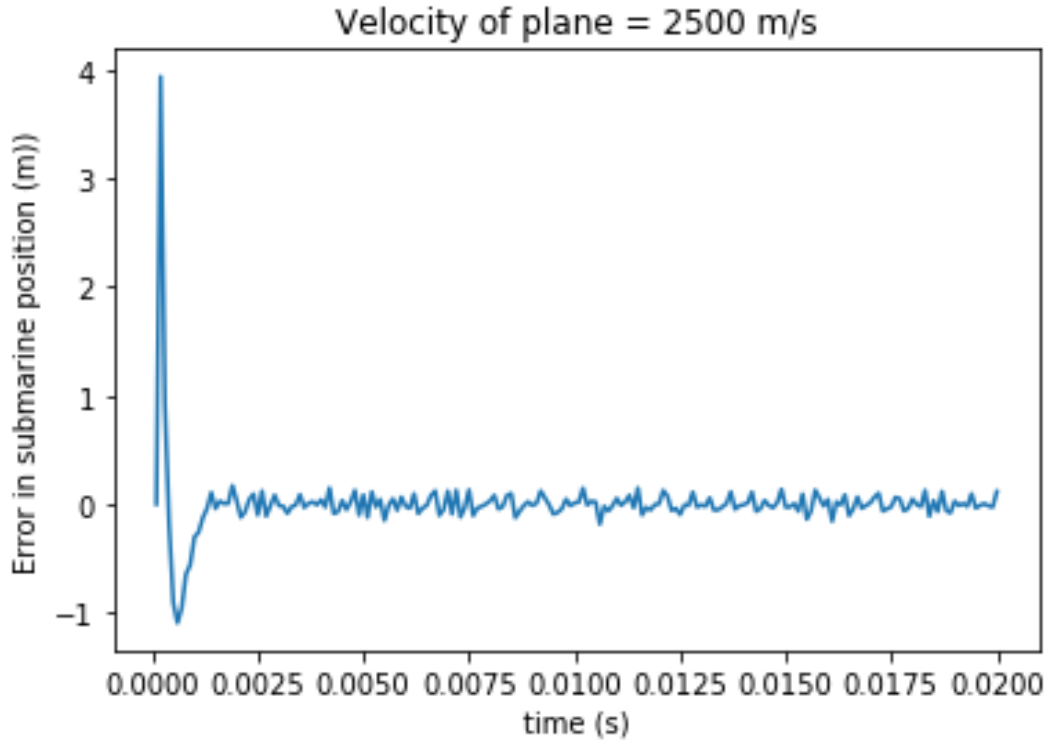


Figure 3.3: Variation of Error in submarine position S_x when velocity of plane V_x is 2500

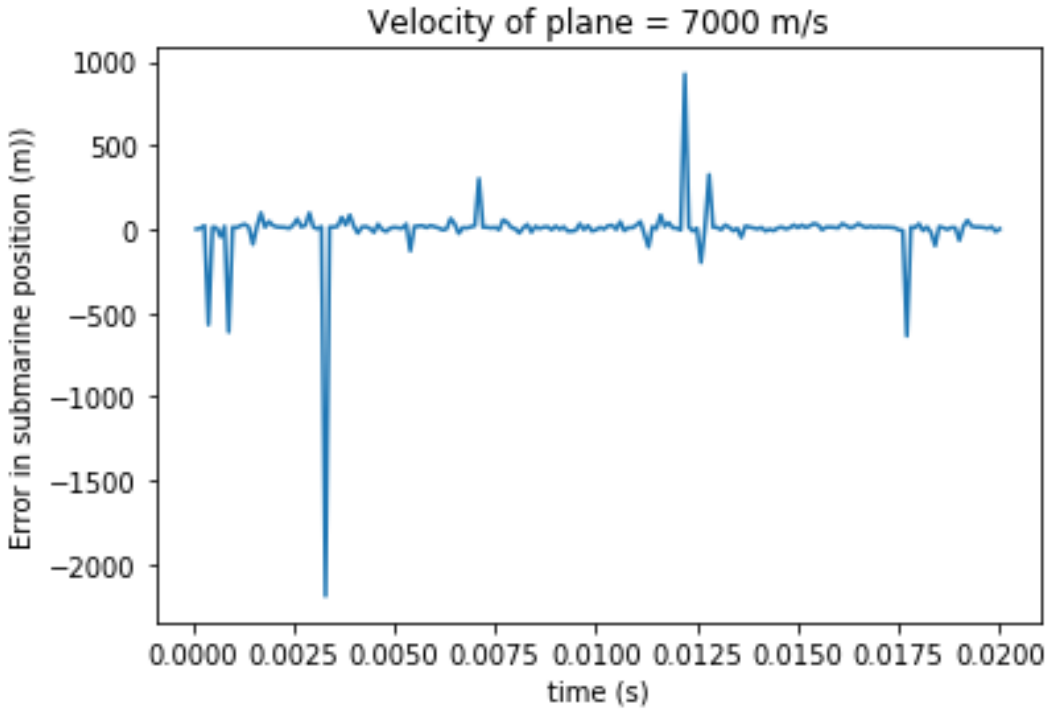


Figure 3.4: Variation of Error in submarine position S_x when velocity of plane V_x is 7500

3.3 Implementation using self defined Brent's function

3.3.1 Need for a self defined function

A common thread across solving the tasks listed in the earlier sections, was finding incident point of the laser beam on the ocean wave. This was done, in those sections, by using **brentq**(), located in the scipy package.

Brentq() function works by running the function over a fixed number of iterations (indicated by the argument *maxiter*) and returns the solution, once the error of function value goes below a fixed threshold (indicated by the argument *xtol,rtol*. As a result, we do not get to find the error of the function in other non-convergent cases using the brentq() function (as it gives a RunttimeError). Therefore, in order to find the error in those cases, it was essential to implement a self defined Brent function.

At a fixed point in time, We vary the number of iterations that brent takes each time (from 1 to 120), thus ensuring that there will be some cases where the function will be non-convergent and will return significant error. For each value of iteration, an exact brent is used to find and return a solution

Based on the error in incident point, other factors such as error in transmission angle and error in submarine position are calculated. All these factors are plotted as a function of iteration and illustrated in the graphs below.

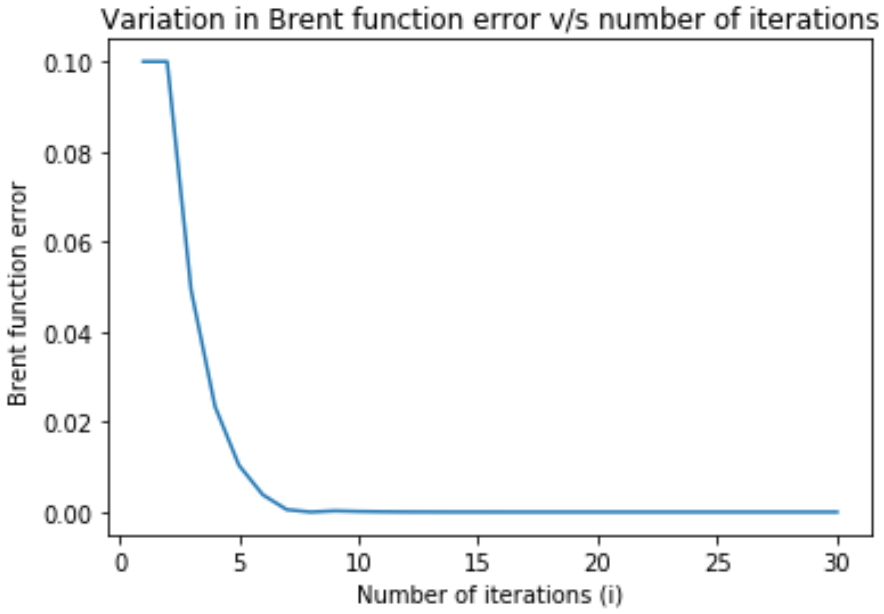


Figure 3.5: Variation of error in Brent function until it converges versus number of iterations

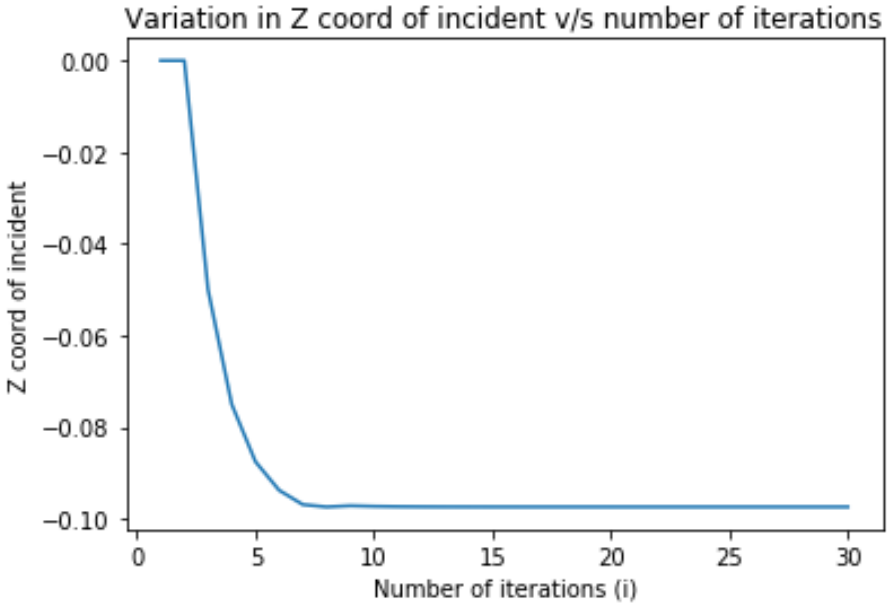


Figure 3.6: Variation in incident point with number of iterations

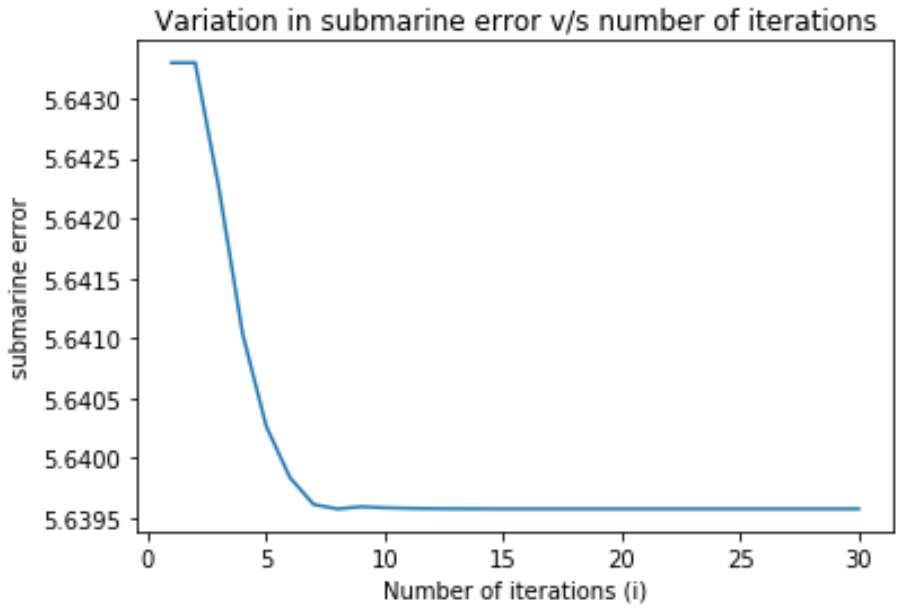


Figure 3.7: Variation of error in submarine position with iterations

From the graph, we see that there is a convergence in the Y axis values after a certain number of iterations. This means stability in the system is achieved after a given number of iterations (i.e) error goes below a threshold value.

Minimum number of iterations required for stabiity = 8

CHAPTER 4

CONCLUSION and FUTURE WORK

The following tasks have been successfully completed :

- Given location of plane and a direction of incident wave, we have found the required transmission angle, incident point and the directions of refracted wave that reaches submarine. Appropriate graphs have been plotted showing the variation of these parameters with time.
- Given a moving plane , stationary ocean and error in our submarine position, we were able to find required transmission angle to establish. We also found out accuracy of incident point location and laser direction.
- We also found out the minimum number of iterations for brent function so that the system was stable and subsequently, variations of various parameters with number of iterations. (Check previous sections for specifics)

Results Conclusion :

$$Incident\ point : (-6000, 0)$$

$$Number\ of\ iterations\ required\ for\ stability : 8$$

Regarding the future scope of this project, the following tasks can be performed :

- Given location of plane and location of submarine, find ideal lauch transmission angle for laser beam when ocean wave equation is not known (it, however, varies with time)
- Determine the minimum bandwidth so that link can be established and maintained.
- What are the stability boundaries wrt bits of resolution and frequency of correction info?

REFERENCES

[1] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.742.1321rep=rep1type=pdf

[2] https://math.berkeley.edu/ mgu/Seminar/Fall2011/7sept2011zerofinder.pdf

[3] http://www.ece.mcmaster.ca/ davidson/EE3CL4/slides/PID.pdf

[4] https://en.wikipedia.org/wiki/BrentAlgorithm

APPENDIX A

Appendix

A.1 Code for finding laser launch angle (By finding incident point)

```
import numpy as np, scipy as sp
import scipy.optimize as opti
import matplotlib.pyplot as plt
from matplotlib.axes import Axes
from mpl_toolkits.mplot3d import Axes3D
from pylab import *
import math
import datetime

A = 0.1
lamda = 2.0
T = 2.0
n1 = 1.000271800
n2 = 1.33
k = 2*np.pi/lamda
w = 2*np.pi/T
r = n1/n2
N = 100

P = np.array([0., 10000.]) #location of plane
S = np.array([6750., -2000.])#location of submarine
d = np.array([3.,-5.]) #incident vector
di = d/np.linalg.norm(d)#incident unit vector
#Q = P - d*P[2]/d[2] #incident point on x-y plane
xArray = np.zeros(N)
zArray = np.zeros(N)
linkArray = np.zeros(N)
incr = 0

def myfunc(z, t):
    return z-A*np.sin(k*(P[0]+(z-P[1])*di[0]/di[1]) - w*t)

for t in np.linspace(0, T, 100):
    z = opti.brentq(myfunc, -A, A, args=(t,) , maxiter = 10)
    x = P[0]+(z-P[1])*di[0]/di[1]
    z = round(z,4)
    xArray[incr] = x
    zArray[incr] = z
```

```
        R = P + di*(z-P[1])/di[1] #point of incidence on water
        n = np.array([-A*k*np.cos(k*R[0] - w*t), 1.])
        n = n/np.linalg.norm(n) #normal at incidence
        cos = -np.dot(n,di)
        dr = di + 2*cos*n
        dt = r*di + (r*cos - np.sqrt(1-(1-cos**2)*r**2))*n
        Sx = x + (S[1] - z)*dt[0]/dt[1]
        refr = -np.dot(n,dt)
        refractedAngle = arccos(refr)



        linkA = arccos(n[0]) * 180 / math.pi
        incidentAngle = arccos(cos)* 180/math.pi

        linkArray[incr] = incidentAngle#(180.0 - incidentAngle - linkA)
        #print(incidentAngle)
        #print(2*t1 + 2*t2)#running time
        #print(x,z)#printing incident point
        #print(180.0 - incidentAngle - linkA)#print incident angle
        print(n, dt, dr)
        incr = incr +1

plt.title("Plot of transmission angle v/s time")
plt.xlabel("time (s)")
plt.ylabel("Angle of incidence (degrees)")
plt.plot(np.linspace(0, T, 100) , linkArray)
plt.show()
```

A.2 Code for finding relation between laser transmission angle and error in submarine position

```
import numpy as np
import scipy as sp
import scipy.optimize as opti
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import matplotlib.animation as animation
import math

A = 0.1
lamda = 2.0
T = 2.0
n1 = 1.000271800
n2 = 1.33
k = 2*np.pi/lamda
w = 2*np.pi/T
r21 = n2/n1
r12 = n1/n2
```

```
t = 0.
Vx = 250.

P = np.array([0., 10000.])
Sy = -100.

d = np.array([3., 4.]) #incident vector
di = d/np.linalg.norm(d)#initial incident unit vector


#Moving the sine wave
Ki = 1.
Kd = 1.
errorSxSum = 0
deltaXo = 4000.0
So = 0.
N = 200 #Number of iterations
#totalTime = 10
#timePerIteration = 1e-3
#N = int(totalTime/timePerIteration )
errorSxArray = np.zeros(N)
Rarray = np.zeros(N)
incr = 0
integral = 0
derivative = 0
priorErrorSx = 0
totalTime = 0.02
timePerIteration = totalTime/N
thetaParray = np.zeros(N)
thetaNarray = np.zeros(N)
diArray = np.empty((0,2))
dtArray = np.empty((0,2))
nArray = np.empty((0,2))

def myfunc3(z, t):
    P = np.array([Vx*t    , 10000.])
    #print(P)
    R = P + di*(z-P[1])/di[1]
    return z-A*np.sin(k*R[0])

for t in np.linspace(timePerIteration, totalTime, num = N):

    P = np.array([Vx *t , 10000.])
    #print(P)
    z = opti.brentq(myfunc3, -A, A, args=(t,)  )
    print(z)
    R = P + di*(z-P[1])/di[1] #point of incidence on water
    Rarray[incr] = R[0]
    n = np.array([-A*k*np.cos(k*R[0]), 1.])
    n = n/np.linalg.norm(n) #normal at incidence
```

```
    cos = -np.dot(n,di)
    dr = di + 2*cos*n
    dt = r12*di + (r12*cos - np.sqrt(1-(1-cos**2)*r12**2))*n
    diArray = np.append(diArray, [di], axis=0)
    dtArray = np.append(dtArray, [dt], axis=0)
    nArray = np.append(nArray, [n], axis=0)
    thetaP = math.atan2(-di[1],di[0])
    thetaParray[incr] = np.rad2deg(thetaP)

    thetaS = math.atan2(-dt[1],dt[0])
    thetaN = math.atan2(-n[0],n[1])
    thetaR = (math.pi/2)-thetaN-thetaS
    #print(thetaR)
    #print(incr)
    Slevel = R + dt*(Sy-R[1])/dt[1]

    if(incr == 0) :
      So = Slevel[0]
    else :
      errorSx = Slevel[0] - So + deltaXo
      errorSxArray[incr] = errorSx
      #print(So)

      Kp = (0.5)/(Sy*math.sin(thetaP+thetaN)/
      (r21*math.cos(thetaR)*(math.sin(thetaS))**2)-P[1]/
      (math.sin(thetaP))**2)
      Ki = 1.2*Kp/(4*timePerIteration)
      integral = integral + errorSx*timePerIteration
      derivative = (errorSx - priorErrorSx)/t
      dthetaP = Kp*errorSx + Ki*integral + math.atan2(Sy,(Vx*t))
      priorErrorSx = errorSx
      thetaP = thetaP-dthetaP
      di[0] = math.cos(thetaP)
      di[1] = -math.sin(thetaP)
      #print(errorSx , R[0])
    incr = incr + 1

plt.title("Velocity of plane = 7000 m/s")
plt.xlabel('time (s)')
plt.ylabel('Error in submarine position (m))')
plt.plot(np.linspace(timePerIteration,totalTime,num =N),errorSxArray
plt.show()
```

A.3 Code for finding variation of various parameters with number of iterations in Brent's function

```
import numpy as np
import scipy as sp
import scipy.optimize as opti
```

```
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import matplotlib.animation as animation
import math
import datetime
import timeit
from pylab import *

A = 0.1
lamda = 2.0
T = 2.0
n1 = 1.000271800
n2 = 1.33
k = 2*np.pi/lamda
w = 2*np.pi/T
r21 = n2/n1
r12 = n1/n2
#t = 0.
Vx = 250

P = np.array([9000., 10000.])
Sy = -100.
d = np.array([3., 4.]) #incident vector
di = d/np.linalg.norm(d)#initial incident unit vector

saade = math.pi/2
So = 0.
deltaXo = 4000.0
N = 120
zArray = np.zeros(N)
brentArray = np.zeros(N)
subArray = np.zeros(N)
incrArray = np.zeros(N)
def swap(t1, t2):
  return t2, t1

def func(z):
  R = P + di*(z-P[1])/di[1]
  return z-A*np.sin(k*R[0] - saade)

def brent(func,lower,upper,tol,maxiter) :

  a = lower
  b = upper
  fa = func(lower)
  fb = func(upper)
  fs = 0

  if (abs(fa) < abs(b)) :
    swap(a,b)
    swap(fa,fb)

  c = lower
```

```
fc = func(lower)
mflag = True
s = 0 #Our Root that will be returned
d = 0 #Only used if mflag is unset (mflag == false)

for i in range(1,maxiter) :

  if abs(upper - lower) < tol:
    print("After" , i, "iterations the root is: " , s )
    return s

  if fa != fc and fb != fc :
    s = ( a * fb * fc / ((fa - fb) * (fa - fc)) )
    + (b*fa*fc/((fb - fa) * (fb - fc)))
    + ( c * fa * fb / ((fc - fa) * (fc - fb)) )

  else :
    #secant method
    s = b - fb * (b - a) / (fb - fa);


  if (((s<(3*a+b)*0.25) or (s>b)
  or (mflag and (abs(s-b)>=(abs(b-c)*0.5)))
  or ((not mflag) and (abs(s-b) >= (abs(c-d) * 0.5)))
  or ( mflag and      (abs(b-c) < tol) )
  or ( (not mflag) and (abs(c-d) < tol)))) :
    #bisection method
    s = (a+b)*0.5;
    mflag = True;

  else :
    mflag = False;


  fs = func(s)  # calculate fs
  d = c    # first time d is being used
  c = b    # set c equal to upper bound
  fc = fb  # set f(c) = f(b)

  if ( fa * fs < 0) :  # fa and fs have opposite signs

    b = s
    fb = fs  # set f(b) = f(s)

  else :

    a = s
    fa = fs  # set f(a) = f(s)


  if (abs(fa) < abs(fb)) :

    swap(a,b)   #swap a and b
```

```
        swap(fa,fb) #make sure f(a) and f(b) are correct after swap

    #End of for

    return(s)

incr = 0
for i in range(1,N+1):
    P = np.array([9000. , 10000.])
    #print(i)
    begin = datetime.datetime.now()
    z = brent(func, -A , A, tol = 10e-8, maxiter = i)
    end = datetime.datetime.now()
    print(i , z,func(z))
    R = P + di*(z-P[1])/di[1] #point of incidence on water

    n = np.array([-A*k*np.cos(k*R[0]), 1.])
    n = n/np.linalg.norm(n) #normal at incidence
    cos = -np.dot(n,di)
    dr = di + 2*cos*n
    dt = r12*di + (r12*cos - np.sqrt(1-(1-cos**2)*r12**2))*n
    linkA = np.arccos(n[0]) * 180 / math.pi
    incidentAngle = arccos(cos)* 180/math.pi


    Slevel = R + dt*(Sy-R[1])/dt[1]
    errorSx = Slevel[0] - So + deltaXo
    #print(i,func(z) , errorSx/1000)

    zArray[incr] = z
    brentArray[incr] = func(z)
    subArray[incr] = errorSx/1000
    incrArray[incr] = incidentAngle
    incr = incr + 1

plt.title("Variation between transmission angle and brent error")
plt.xlabel('Brent Error (m)')
plt.ylabel('Transmission angle in degrees')
plt.plot(brentArray , incrArray)
plt.show()
}
```