PROPORTIONAL NAVIGATION

GROUP MEMBERS

- BHASWAR CHATTOPADHYAY – 2020CSB019
- KIRANMOY SAHA – 2020CSB023
- MANDIRA DAS – 2020CSB041
- RAUNAK KUMAR – 2020CSB071
- VIHAAN SABBANI – 2020CSB093
- DIVI SRIVASTAVA – 2020CSB101

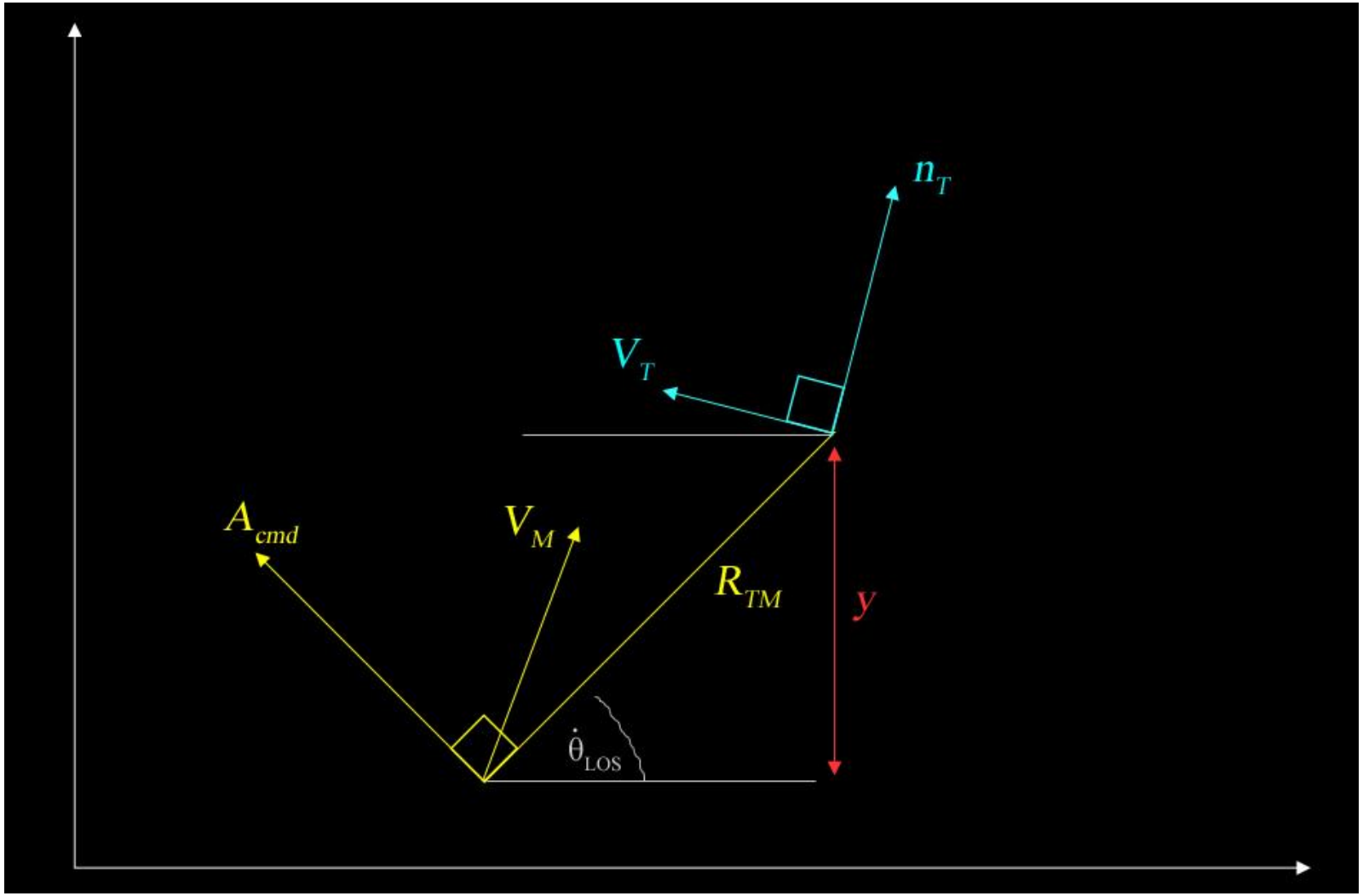
OBJECTIVES:

- To understand the basics of proportional navigation
- To implement our understanding in the form of coding and to simulate the results to get desired output

THEORY

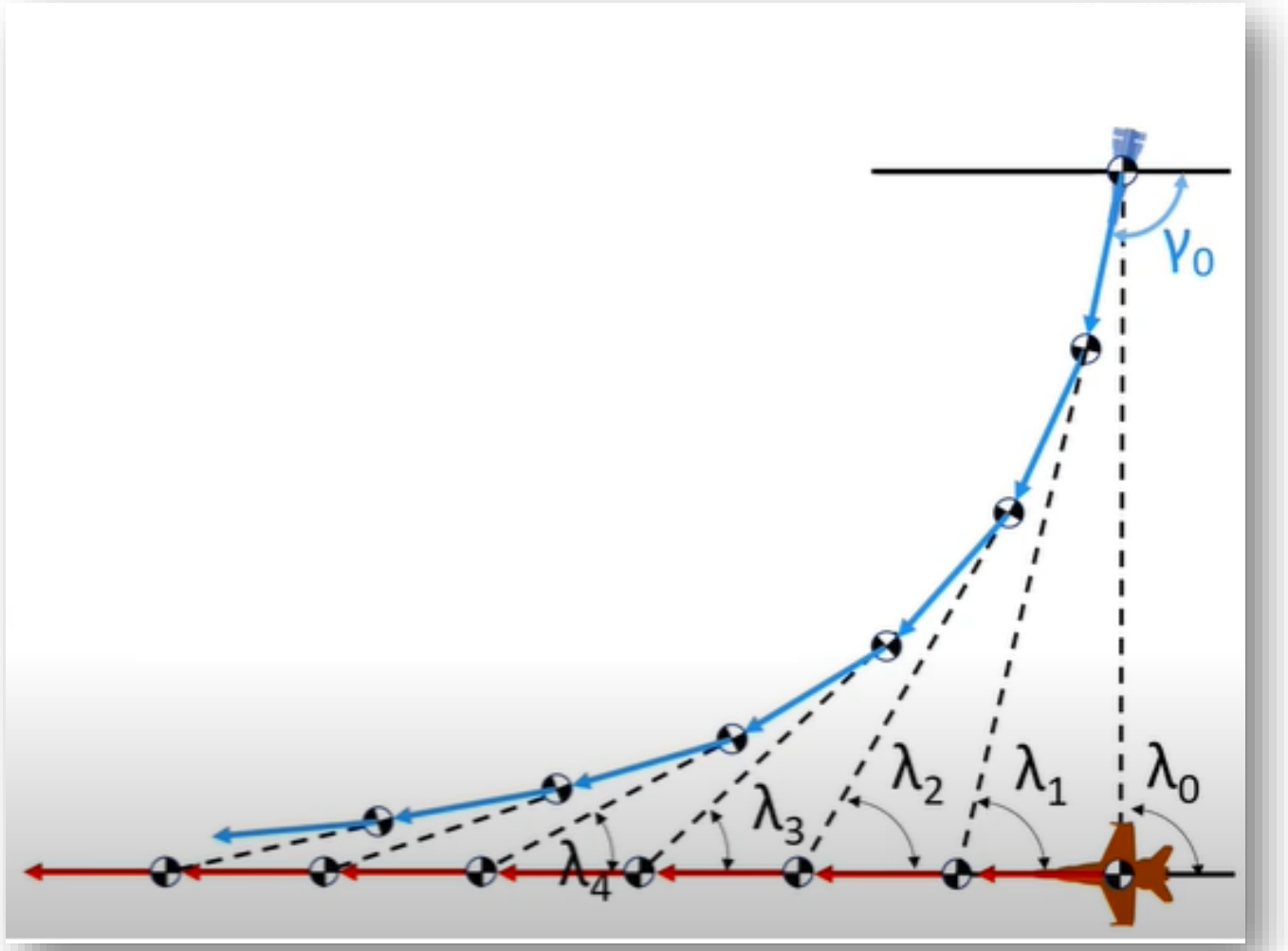
- PN works on the principle of "Constant Bearing Decreasing Range" (CBDR), where when two objects are heading in the same direction with no change in Line of Sight (bearing) angle, objects *will* collide. Line of Sight (LOS) is an imaginary sight-line between you and the target; when target is moving from left to right in your field of view, it is said that the LOS angle is changing from left to right.
- If however, you were to also run from left to right and accelerate appropriately to keep the target centered in your field of view, it is then said the rate at which LOS angle is changing (LOS rotation rate) is zero ("null"). Continuing to run with LOS rate maintained at zero will result in intercept and "lead to pursuit" collision with the object you're chasing.
- Commanded Acceleration = N * Vc * LOS_Rate N = Unitless navigation gain (constant) -- between 3 to 5 Vc = Closing Velocity (or "range closing rate") LOS_Rate = LOS Rotation Rate

ENGAGEMENT GEOMETRY



METHODOLOGY:

- A target T is taken that moves along the X axis and the interceptor I is vertically above.
- Now T moves along X axis with some velocity vt and I moves with some velocity vp.
- I must change the direction of its velocity vector else misses T.
- The angle of velocity vector is changed proportional to the angle of line of sight with t.
- Depending on vp, vt and constant of proportionality (n) various cases are implemented.



γ= flight path angle
λ = line of sight angle

Demonstration of discretization of proportional navigation

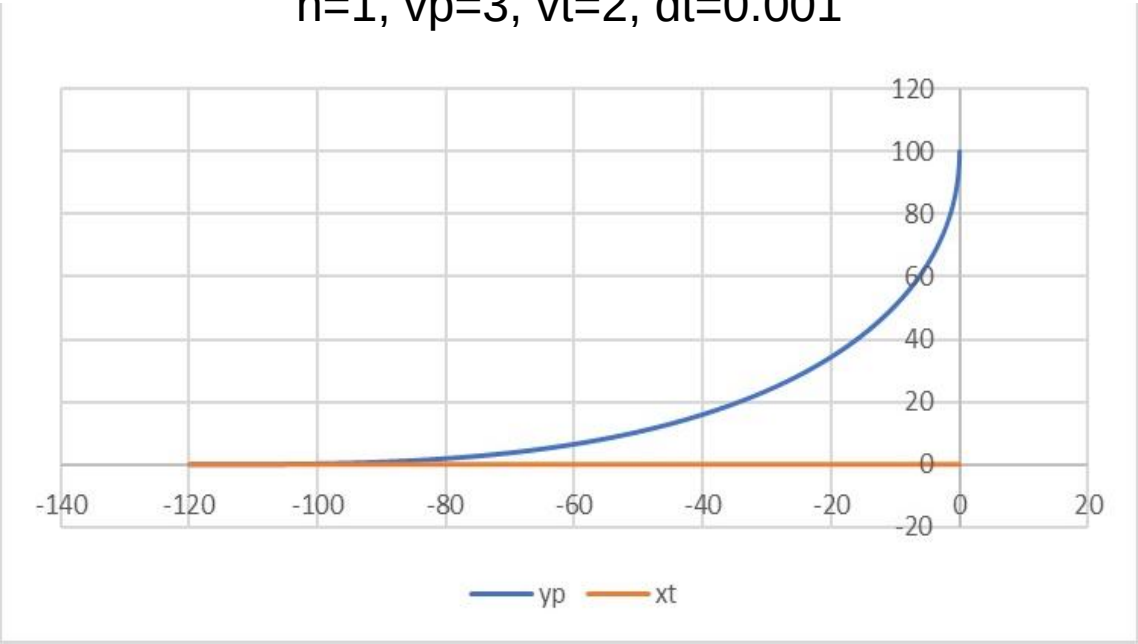
IMPLEMENTATION:

```c
#include<stdio.h>
#include<math.h>
#include<time.h>
float dist(float, float, float, float);
void delay(int number_of_seconds);

float Convert(double radian) //radiun to degree
{
    float pi = 3.14159;
    return(radian * (180/pi));
}

float Convertd2r(double radian) //deegree to radiun
{
    float pi = 3.14159;
    return(radian * (pi/180));
}
```

```
void main()
{
    FILE *fxp, *fyp,*fxt,*fyt;
    fxp=fopen("xp.txt","w+");
    fyp=fopen("yp.txt","w+");
    fxt=fopen("xt.txt","w+");
    fyt=fopen("yt.txt","w+");

    float xp,yp,xt,yt,lambda, gamma,vt,vp,dt;
    xp=yp=xt=yt=lambda=gamma=0;
    xp=0;yp=100; // taking pursure at 100 meter height
    lambda=gamma=90; // taking both inititally 90
    int n=5; int count=0;
    vt=2;vp=3;dt=0.001;
    while(dist(xp,yp,xt,yt)>0.001)
    {
        xp=xp-vp*dt*sin(Convertd2r(gamma-90));
        yp=yp-vp*dt*cos(Convertd2r(gamma-90));
        xt=xt-vt*dt;
        yt=0;
        gamma=gamma+(lambda-Convert(atan(yp/(xp-xt))))*n;
        lambda=Convert(atan((yp/(xp-xt))));
        printf("%f %f, %f %f the value of lambda = %f, the value of gamma = %f\n",xp,yp,xt,yt,lambda,gamma);
```

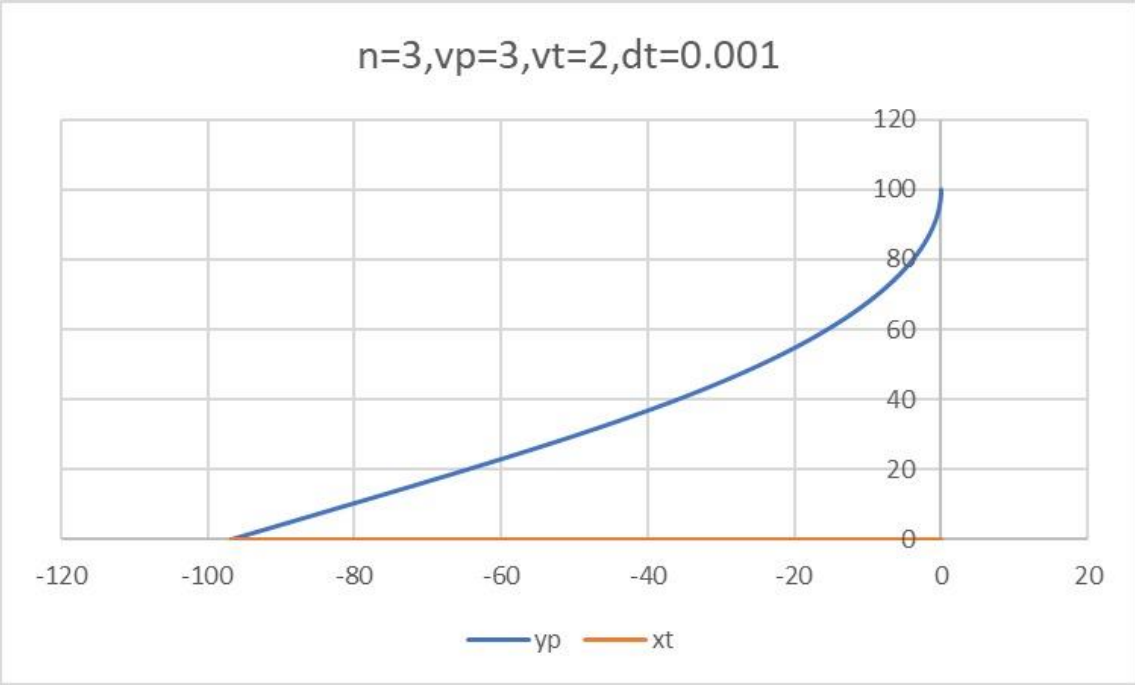
```
        printf("%f %f, %f %f the value of lambda = %f, the value of gamma = %f\n",xp,yp,xt,yt,lambda,gamma);
        fprintf(fxp,"%f \n",xp);
        fprintf(fyp,"%f \n",yp);
        fprintf(fxt,"%f \n",xt);
        fprintf(fyt,"%f \n",yt);

        //delay(1);

    }
    fclose(fxp);
    fclose(fyp);
    fclose(fxt);
    fclose(fyt);
}

float dist(float x1, float y1, float x2, float y2)
{
    float ans;
    ans=sqrt(pow((x1-x2),2)+pow((y1-y2),2));
    return ans;
}
```

RESULT AND SIMULATIONS:



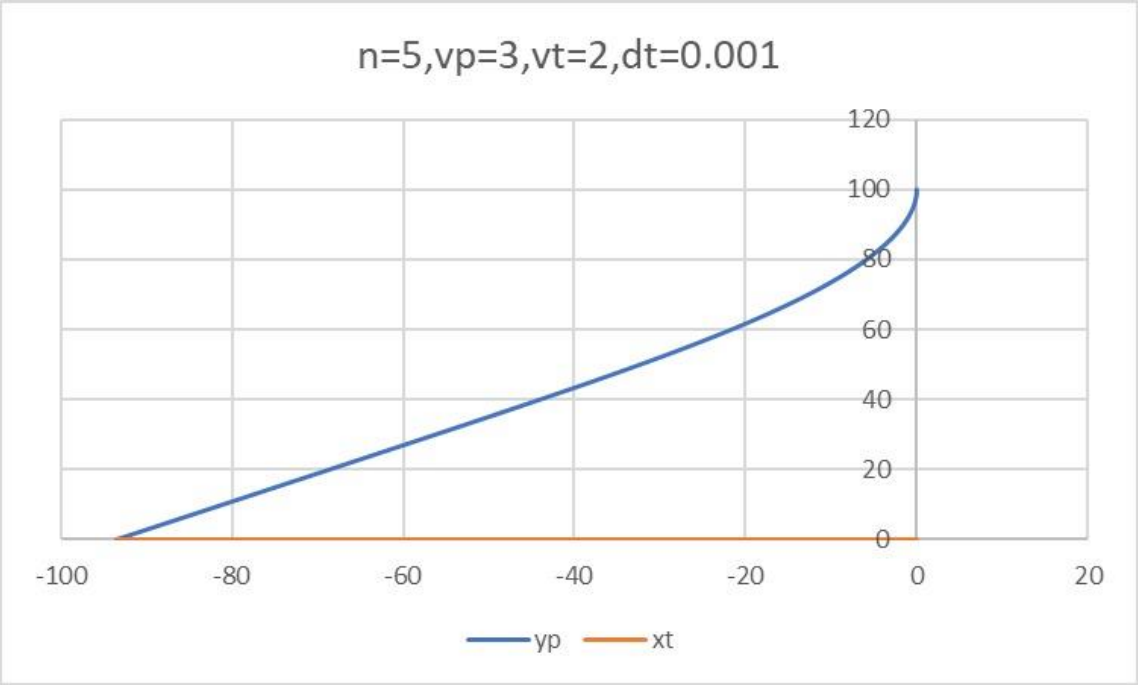
- For first implementation the navigation gain (n) is taken 1, but in real life scenario this is taken from 3-5. For n=1 we can see no collision triangle is formed and the simulation ended with a tail chase.

RESULT AND SIMULATIONS:



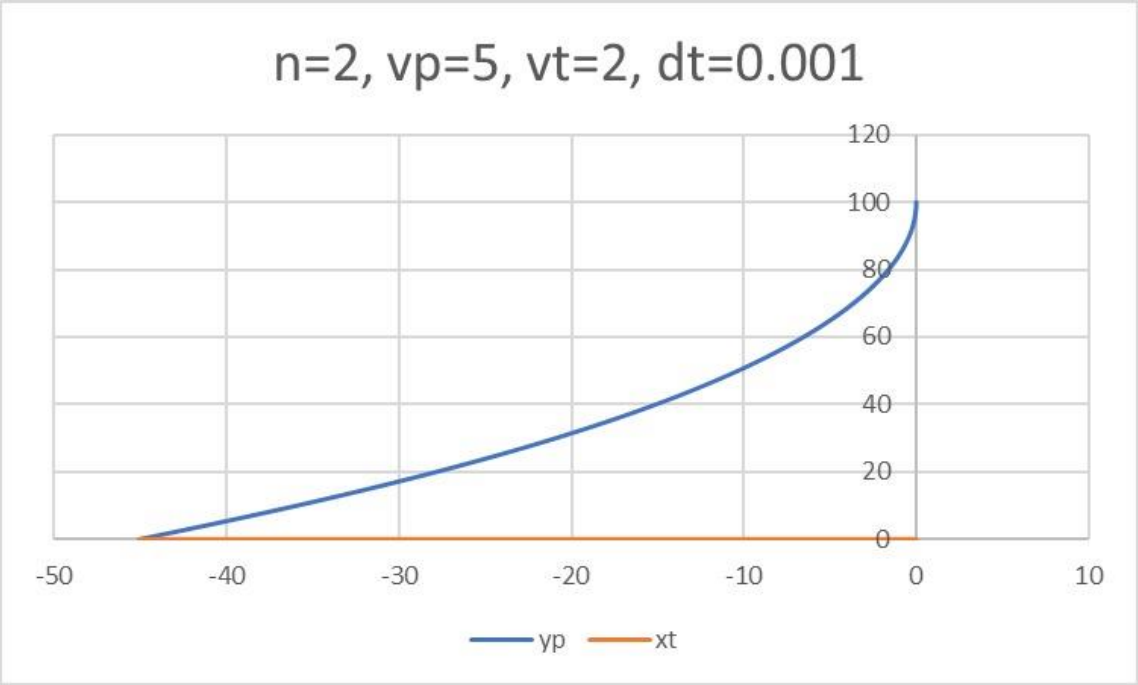
- Here the value of navigation gain (n) is taken 3. So we can see a collision triangle has been formed so the rate of change of lambda and gamma became zero after formation of the collision triangle.

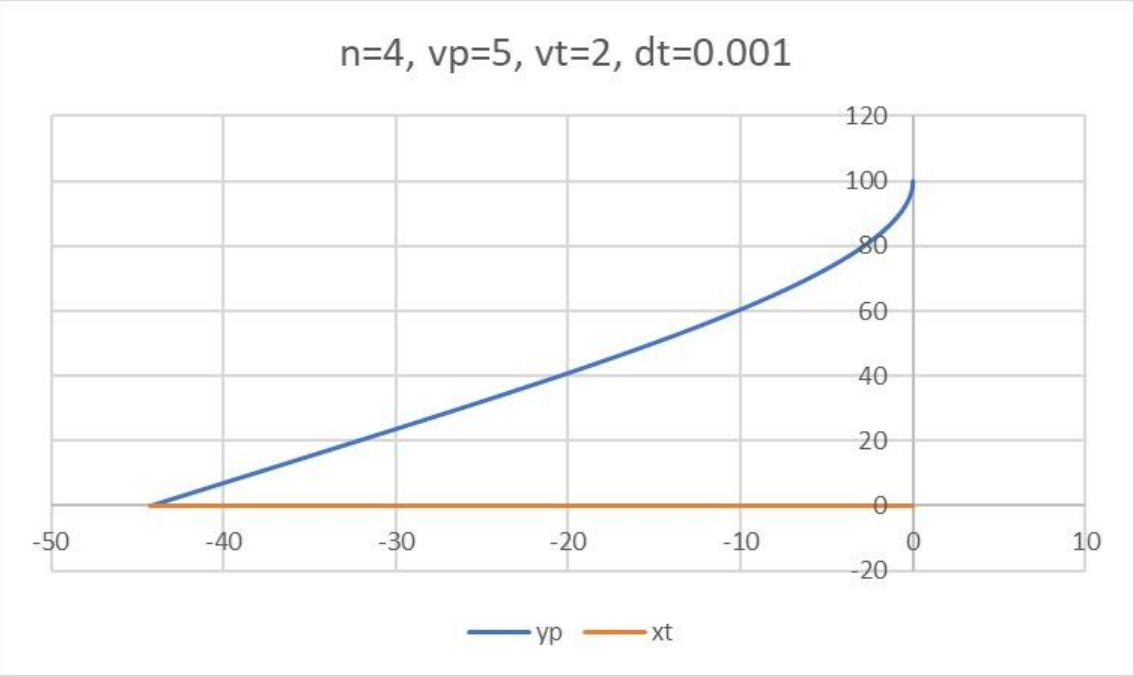
RESULT AND SIMULATIONS:



- For our last case we have taken the value of navigation gain(n) as 5. The interceptor now is very sensitive to change of LOS. So we can see the collision triangle formed even faster than the earlier case where we took the value of n as 3. Although higher value of n are restricted in real life by mechanical limitation of the pursuer.

RESULT AND SIMULATIONS:





In all our last cases we took the vp/vt ratio as 1.5. This is another demonstration where we altered the ratio of velocity of pursuer and target as 2.5 and two cases for n=2 and n=4 has been taken.

LIMITATIONS OF THE PROJECT:

- Initially the interceptor is vertically above the target
- The target moves along negative x axis
- Only 2D coordinates are considered
- Velocities are assumed to be constants

FUTURE SCOPE:

- Instead of taking assumption based special case, implementation of any arbitrary case can be done.
- Acceleration may be given to both target and pursuer.
- 3-D implementations can be done.
- The project can be extended to other dimensions like missile building, game theory etc.

ACKNOWLEDGEMENT:

We would like to express our special thanks of gratitude to our mentor Dr. Abhik Mukherjee , who gave us the golden opportunity to work on this innovative project on the topic of Proportional Navigation, which helped us in doing a lot of research, and we got insights about this domain.

BIBLIOGRAPHY:

- Journal: Fundamentals of proportional navigation by Stephen A. Murtaugh, Harry E. Criel
- Wikipedia: https://en.wikipedia.org/wiki/Proportional_navigation
- https://www.youtube.com/channel/UCzhBARe9GjHHGD_V72xallA
- https://www.moddb.com/members/blahdy/blogs/gamedev-introduction-to-proportional-navigation-part-i

THANK YOU