AS5580 - Course Project

Pursuit Evasion

Abhigyan Roy AE21B002
Hrishav Das AE21B023
Reva Dhillon AE21B108
Agni Ravi Deepa AE21B101

October 27, 2024

Contents

Chapter 1

Optimal Control Problem Set Up

1.1 Introduction: Pursuit Evasion Game:

Pursuit - evasion (variants of which are called cops and robbers and graph searching) is a family of problems in mathematics and computer science in which one group attempts to track down members of another group in an environment. The pursuer seeks to minimize its distance from the evader and stay "pinned" to the evader.

Here, we apply this problem to a pair of Pursuer and Target aircraft. This pursuit-evasion model simulates the dynamics of a pursuer attempting to close in on an evader. For the purpose of this course, dynamics of a 3-DOF model are considered. The following document will present details on the dynamical system model of the aircraft, followed by a discussion on formulating the scenario as an optimal control problem, the 2 different methods that have been used, and visuals of the optimal trajectories obtained.

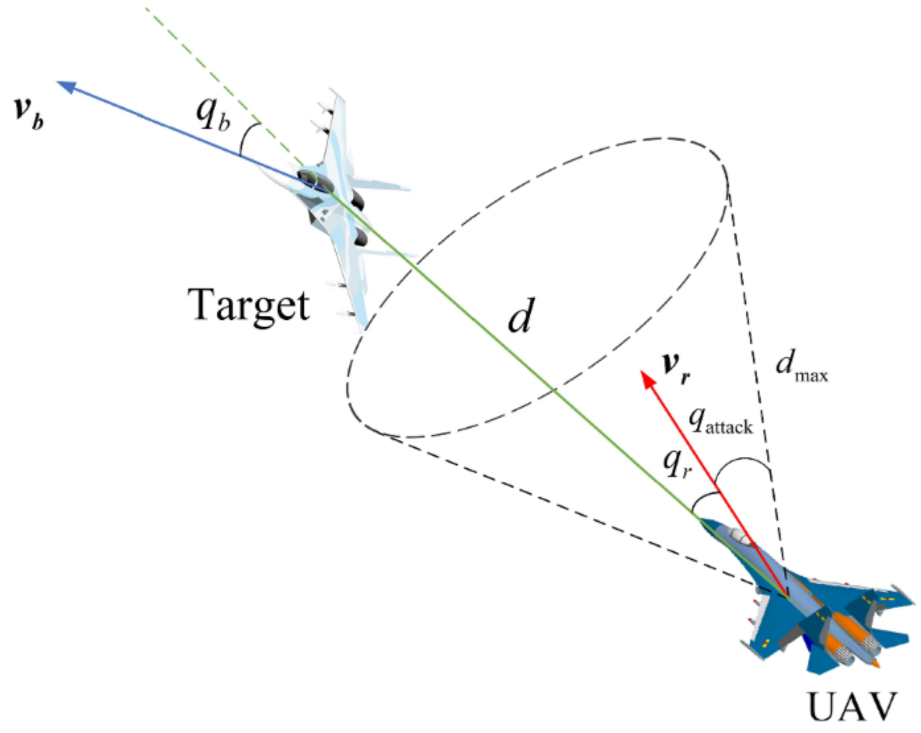


Figure 1.1: Pursuit Evasion

1.2 Solution Formulation

1.2.1 Aircraft 3-DOF Model

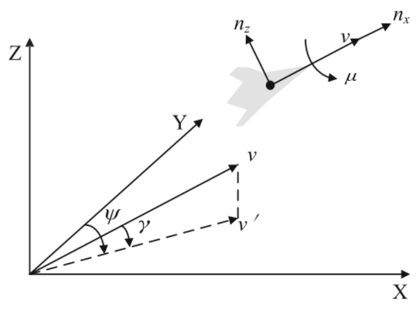
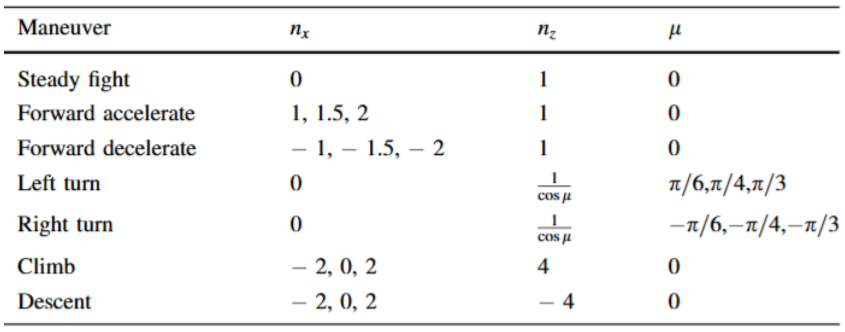


Figure 1.2: 3 degrees of freedom model

In this model, the aircraft is assumed to be a point mass in 3D space where x,y, and z are the position coordinates of the aircraft, v is the flight speed, γ, ψ, and μ are the flight path angle, heading angle, and roll angle respectively. The state vector is constructed as $\vec{X} = [x, y, z, v, \gamma, \psi]^T$. The control vector is $\vec{U} = [n_x, n_z, \mu]^T$ where n_x is a non-dimensional scaling of the tangential acceleration along the speed direction of the aircraft, and n_z is a non-dimensional scaling of the load factor. The equations governing the states are given below:

$$\begin{aligned}
\frac{dx}{dt} &= v\cos\gamma\cos\psi \\
\frac{dy}{dt} &= v\cos\gamma\sin\psi \\
\frac{dz}{dt} &= v\sin\gamma \\
\frac{dv}{dt} &= g\left(n_x - \sin\gamma\right) \\
\frac{d\gamma}{dt} &= \frac{g}{v}\left(n_z\cos\mu - \cos\gamma\right) \\
\frac{d\psi}{dt} &= \frac{gn_z\sin\mu}{v\cos\gamma}
\end{aligned} \tag{1.1}$$

Some sample values of control inputs and their resultant trajectory are listed below for an intuitive feel of the model:



Maneuver	n_x	n_z	μ
Steady fight	0	1	0
Forward accelerate	1, 1.5, 2	1	0
Forward decelerate	− 1, − 1.5, − 2	1	0
Left turn	0	$\frac{1}{\cos\mu}$	$\pi/6,\pi/4,\pi/3$
Right turn	0	$\frac{1}{\cos\mu}$	$-\pi/6,-\pi/4,-\pi/3$
Climb	− 2, 0, 2	4	0
Descent	− 2, 0, 2	− 4	0

Figure 1.3: 3 degrees of freedom model

1.2.2 The Runge-Kutta Fourth Order Method

The Runge-Kutta fourth order method employs the following formula to obtain the solution at the location $(k+1)$ to the equation $\dfrac{d\vec{X}}{dx} = f(\vec{X}, x)$ given the solution at location k.

$$\vec{X}_{k+1} = \vec{X}_k + \frac{\Delta x}{6}(f_1 + 2f_2 + 2f_3 + f_4) \tag{1.2}$$

where,

$$\begin{aligned} f_1 &= f(\vec{X}_k, x_k) \\ f_2 &= f(\vec{X}_k + \frac{\Delta x}{2} f_1, x_k + \frac{\Delta x}{2}) \\ f_3 &= f(\vec{X}_k + \frac{\Delta x}{2} f_2, x_k + \frac{\Delta x}{2}) \\ f_4 &= f(\vec{X}_k + \Delta x f_3, x_k + \Delta x) \\ \Delta x &= \text{Step size} \end{aligned}$$

The function f returns the derivative and may be viewed as a quantity analogous to the direction in a vector field.

Therefore, given a solution $\vec{X}_k$ at a location x_k we obtain the solution $\vec{X}_{k+1}$ at location x_{k+1} by taking a step Δx in averaged direction $(f_1 + 2f_2 + 2f_3 + f_4)/6$.

f_1 can be viewed as the direction of the vector field at the base point. f_2 is the vector field evaluated after step $\Delta x/2$ in the f_1 direction. f_3 is the vector field evaluated after step $\Delta x/2$ in the f_2 direction. f_4 is the vector field evaluated after step Δx in the f_3 direction.

From the Taylor series of the solution, it can be observed that the method has local truncation error of the order $O(\Delta x^5)$ and total accumulated error of the order $O(\Delta x^4)$. This is what makes the method highly accurate and incredibly useful.

1.2.3 Method 1 - Time Stepping using RK4

The first method employed to solve the pursuit-evasion problem utilizes the RK4 algorithm to generate the aircraft trajectory based on the 3-DOF model. The steps followed are given below.

- **Step 1**: Given the initial conditions, generate the pursuer's trajectory using the RK4 algorithm (Python - solve_ivp, MATLAB - ode45). This gives the states of and control inputs applied to the aircraft at discrete time steps.
- **Step 2**: Using the states of the aircraft, compute the running cost and terminal cost.
- **Step 3**: Give this as input to the solver and obtain the optimal values of the control input.

- The input to the solver is the initial guess of the control variables at selected node points. Based on these values, a polynomial for them is computed using Lagrange interpolation. The RK4 method is applied within the cost function to generate the trajectory based on this polynomial. The values of the control variables at the node points are optimized.
- **Advantage of the method**: Each solver requires an initial guess for the input. The solver in most cases is very sensitive to this initial guess. The utilisation of the RK4 method allows us to reduce the number of variables to estimate initially.

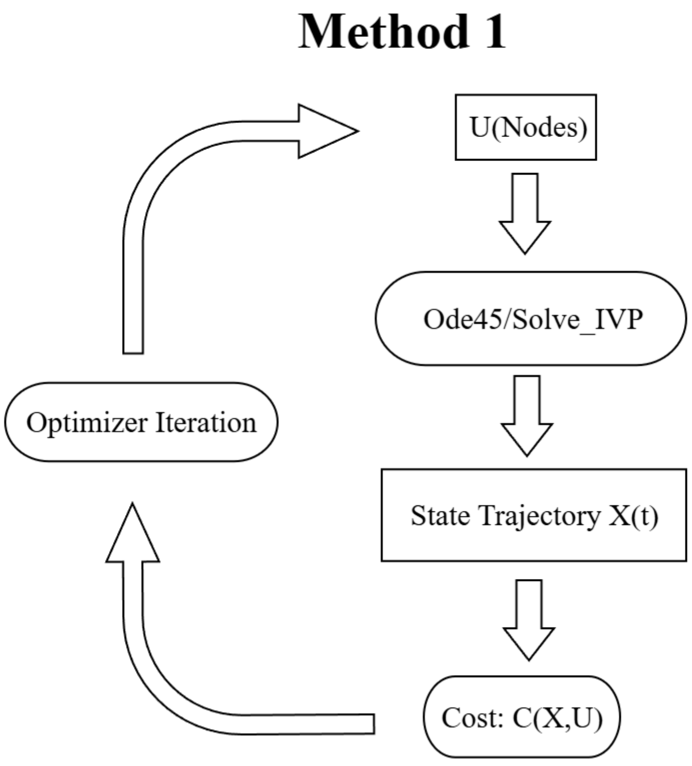


Figure 1.4: Method 1 Information Flow

1.2.4 Method 2 - Nonlinear Constraints

In the second method, we apply the governing equations as a set of non-linear constraints. The time span is discretized and we no longer have any time stepping.

- **Step 1**: For the time range considered, a set of points at which the trajectory will be evaluated is chosen. The governing equations are discretized for each point and these equations form the nonlinear constraints.
- **Step 2**: The cost function is formulated using the states estimated at every time step.
- **Step 3**: The solver generates the optimized trajectory along with the control inputs using this method.

- The input to the solver is a guess for the states as well as the control inputs at all the sampled time steps. The RK4 method is used to generate a feasible initial guess. Though the number of variables to be guessed increases significantly, the speed of the solution increases.
- **Advantage of the method**: Both **ode45 and solve_ivp** suffer from numerical instabilities depending upon the initial conditions. The RK4 method functions are also a blackbox for the solver. In contrast, in this formulation, no instabilities of the same nature are encountered.
- **Points to consider**:
  - We use the forward difference, central difference and backward difference operators to compute the non-linear constraints. The accuracy of this is not as high as that possessed by the RK4 functions.
  - Another method we employed for the constraints' computation is the differentiation matrix. However, this is only accurate if the solution is a polynomial of the order considered.
  - This method is sensitive to the initial guess and the number of variables to guess is significant. This may potentially be a source of error in more complex cases.

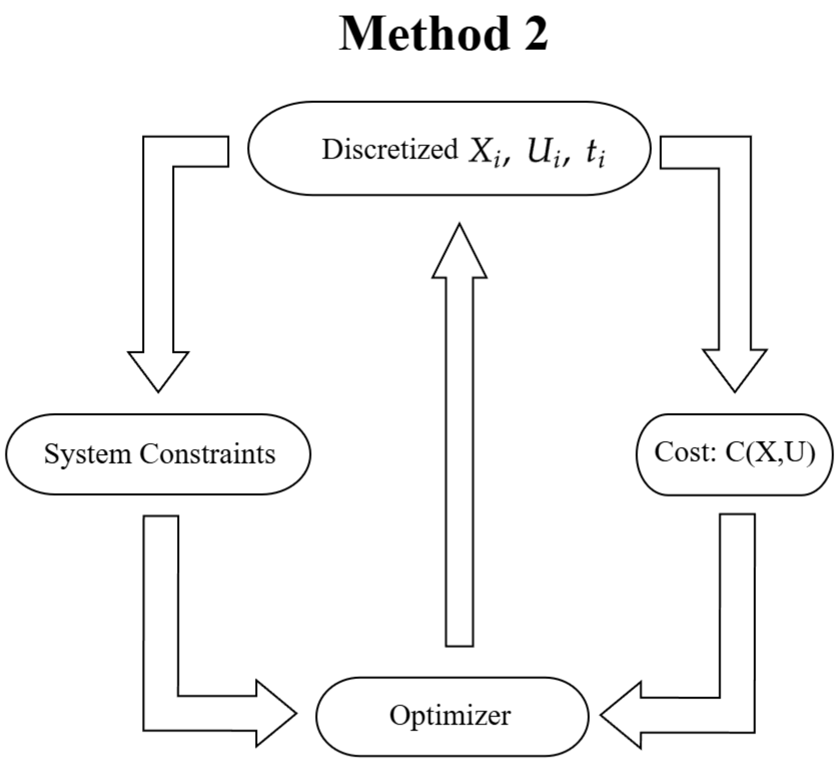


Figure 1.5: Method 2 Information Flow

1.3 Cost Function

The cost function is the function that we wish to minimize. Physically, a reduction in the value of the cost function should relate to a desirable change. The next question is, naturally, what are the desirables?

1.3.1 Running Cost

The running cost is a cost function dependent on all time instances in the system considered. It has the form given below:

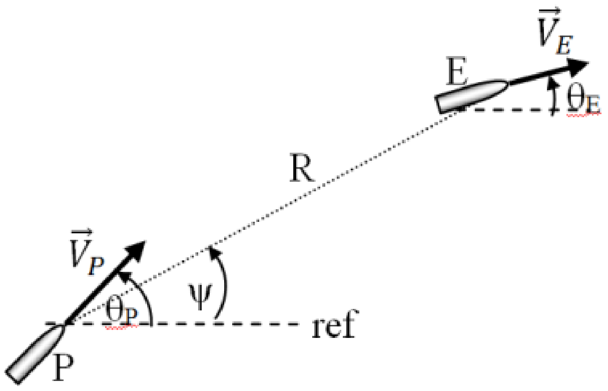
$$J = \int_{t_0}^{t_f} L(t, x(t), u(t))dt$$

- **Distance of the Pursuer from the Target:** To pin the target, the pursuer must get close to it. This can be computed as a running shown below.

  $$J = \int_{t=0}^{t=t_f} \left((x_p - x_e)^2 + (y_p - y_e)^2\right) dt$$

  For the purposes of simulation, this is estimated as a sum of the individual terms at discrete time steps/node points.

  $$J = \sum_{i=0}^{N} \left((x_p(i) - x_e(i))^2 + (y_p(i) - y_e(i))^2\right)$$



- **Control Effort:** It is desirable to ensure the control inputs for an optimal solution are also minimized. This is typically formulated as shown below.

  $$J = \sum_{i=0}^{N} U(i)^T Q U(i)$$

  where U is the control vector defined in 1.2.1, and Q is a diagonal matrix with positive entries that weigh each element of U.

- **Angle Deviation between the Pursuer and Target Penalty:** A penalty term is introduced to discourage the pursuer from entering the attack envelope cone of the target during any time in its trajectory.

  If the seperation distance is less than a defined danger range r_{danger}, we evaluate the angle θ_{pe} between the evaders heading and the vector to the pursuer. The angle θ_{pe} between the evader's heading direction and the relative position vector is given by:

  $$\theta_{pe} = \cos^{-1}\left(\mathbf{r}_{pe} \cdot \mathbf{h}_e\right)$$

  where r_{pe} is the unit vector from the evader to the pursuer defined as

  $$\mathbf{r}_{pe} = \frac{[x_p - x_e, y_p - y_e]}{\|\mathbf{r}_{pe}\|}$$

  and the targets heading direction vector $\mathbf{h}_e$ is calculated as

  $$\mathbf{h}_e = [\cos(\gamma_e)\cos(\psi_e), \cos(\gamma_e)\sin(\psi_e), \sin(\gamma_e)]$$

  where γ_e and ψ_e represent the targets pitch and yaw angles, respectively.

1.3.2 Terminal Cost

The terminal cost is a cost function defined with respect to the final states and final time of the system. It has the form given below:

$$J = K(t_f, x_f)$$

- **Final Distance between the Pursuer and the Target:** To ensure that the pursuer pins the target by the final time t_f, we add a terminal separation term:

  $$J = (x_{p_f} - x_{e_f})^2 + (y_{p_f} - y_{e_f})^2$$

- **Final states:** To ensure that upon capture, the pursuer continues to pin the evader, all the states must match at the final time. Therefore, we also impose a terminal cost of the form:

  $$J = ||x_{p_f} - x_{e_f}||^2$$

A favorable cost function captures all the above desirables. Another key factor that needs to be taken care of is to ensure that the cost function is as convex as possible. Notions of convexity ensure and aid optimizers to reach the optimal point faster. Here, we have taken most of the terms in quadratic form.

The cumulative cost function is written as a weighted sum of each of the individual terms presented above. One can play around with the weights to give relative importance to one aspect of the cost compared to the others depending upon the problem requirements.

The **terminal cost** and **angle penalty** have higher weights to make certain that the pinning occurs within the conditions we want that is,
- Pursuer hits target by final time.
- Pursuer never enters the attack envelope of target.

1.4 Time Discretization and Optimization Nodes

In this section, we discuss the manner in which time is discretized and the control inputs are computed.

Method 1:

- Here, given the final time, we discretize the time span into a number of points at which we choose to sample the states. Let the number of points be equal to N.
- To compute the control inputs, we select a smaller number of node points $= n$.
- Given the control inputs at these n points, we use Lagrange interpolation to generate a continuous $(n-1)^{th}$ order polynomial approximation for the control inputs.
- The solver optimises the values of the control inputs at the node points and RK4 is used to generate the states of the system.
- In this method, we have the luxury to estimate the control inputs $U(t)$ with a sparser node distribution while ensuring that the cost is calculated over a much finer time step.

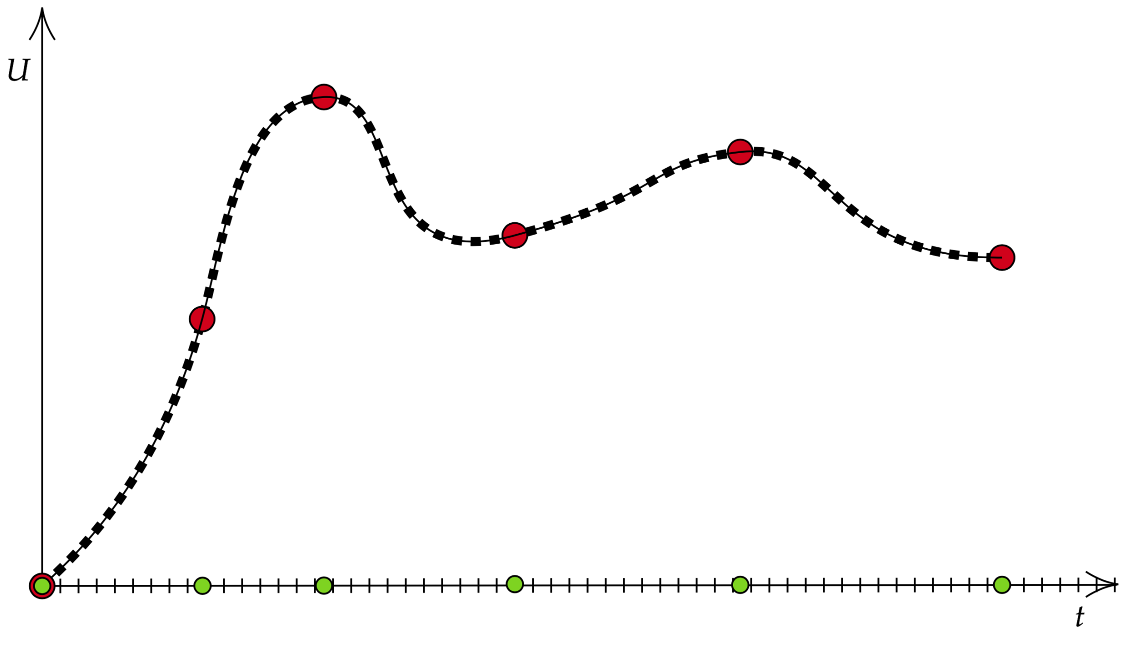


Figure 1.6: Variation of control inputs U at the node points considered. The green circles denote the constant initial guess given while the red circles denote the optimised values.

Method 2:

- Similar to method 1, given the final time, we discretize simulation time into N points (nodes).
- All the states and control inputs are estimated as a Lagrange polynomial interpolated at these time nodes. This included the 6 states in X and 3 control inputs in U.

- The system dynamics are enforced as a set of equality constraints given to the optimizer.
- The sampling points of the control inputs and the states are the same.
- The solver optimizes for the nodal values of the states and the control inputs subject to the system dynamics' equality constraints.
- In order to accurately capture the system dynamics, one needs a high number of nodes.

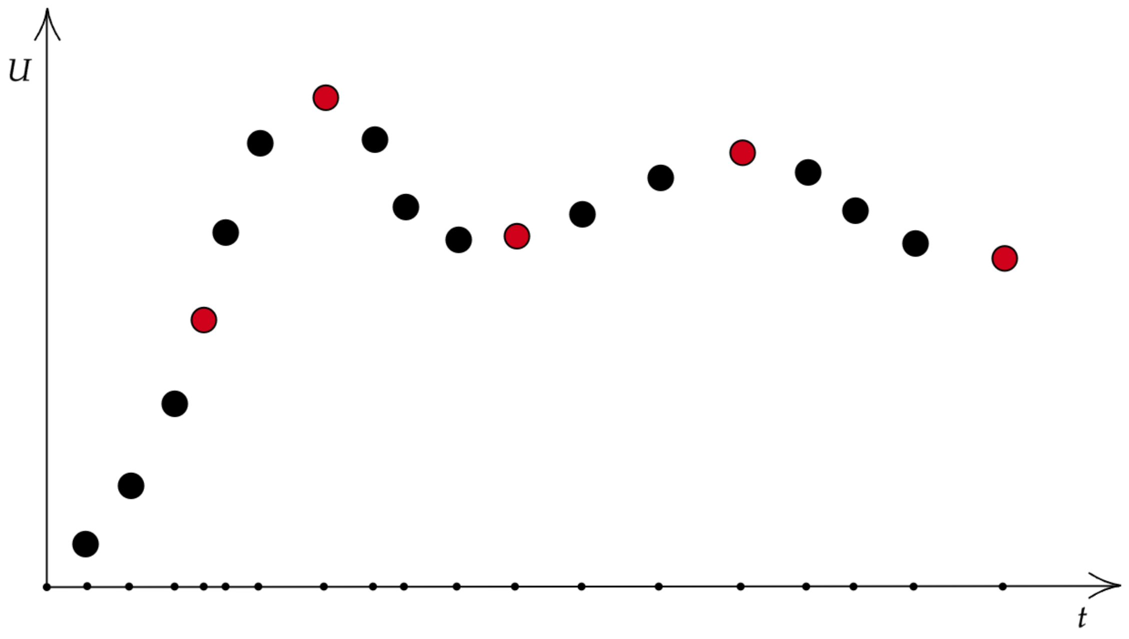


Figure 1.7: Method 2 Discretization

1.5 Constraints

There are broadly two types of constraints enforced:

- **Magnitude Constraints**: Enforced as upper and lower bounds on the control inputs.

$$\begin{bmatrix} n_{x_L} \\ n_{z_L} \\ \mu_L \end{bmatrix} \leq \begin{bmatrix} n_{x_i} \\ n_{z_i} \\ \mu_i \end{bmatrix} \leq \begin{bmatrix} n_{x_U} \\ n_{z_U} \\ \mu_U \end{bmatrix} \quad ;\forall i \text{ node points}$$

- **Rate Constraints**: Enforced as upper and lower bounds on the derivatives of the control inputs. The differentiation matrix is used to compute the derivative of the control inputs at the node points.

$$\begin{bmatrix} \dot{n}_{x_L} \\ \dot{n}_{z_L} \\ \dot{\mu}_L \end{bmatrix} \leq \begin{bmatrix} \dot{n}_{x_i} \\ \dot{n}_{z_i} \\ \dot{\mu}_i \end{bmatrix} \leq \begin{bmatrix} \dot{n}_{x_U} \\ \dot{n}_{z_U} \\ \dot{\mu}_U \end{bmatrix} \quad ;\forall i \text{ node points}$$

Chapter 2

Numerical Simulations

2.1 Solvers

We implemented the following solvers and compared the results from each of them:

- Python:
  - **cyipopt.Problem** and **cyipopt.minimize_ipopt**: cyipopt is a Python wrapper for the Ipopt optimization package, written in Cython. cyipopt.Problem and cyipopt.minimize_ipopt are two ways to implement the solver. We find that while accurate, this solver requires a significantly higher amount of time to complete the optimisation. This motivated us to explore the optimization functions avaliable in the scipy library.
  - **scipy.optimize.minimize(method = 'L-BFGS-B')**: It stands for Limited-memory Broyden Fletcher Goldfarb Shanno with Box constraints. It is a quasi Newton method that works very fast due to the fact that it does not calculate the complete hessian matrix required during the optimisation process. This method achieves speeds close to fmincon in matlab.
- MATLAB:
  - **fmincon**: fmincon is the nonlinear programming solver available in MATLAB that returns the optimal values for the desired variables such that the corresponding cost function is minimized subject to constraints.

2.2 Python

The table below summarizes all the cases considered.

Table 2.1: Simulation Parameters

Case	t_0	t_f	U_{target}	X_0 (Pursuer)	X_0 (Target)	N1	n1	N2
1	0	20	[0.0, 0.0, 0.0]	[0, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
2	0	20	[0.0, 0.0, 0.3]	[0, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
3	0	20	[0.0, 0.0, 0.3]	[400, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
4	0	30	[0.0, 0.0, -0.3]	[-300, 200, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
5	0	50	[0.0, 0.0, -0.7]	[0, 0, 0, 50, 0, 0.75π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
6	0	50	[0.0, 0.0, 0.0]	[0, 0, 0, 50, 0, 0.25π]	[250, 0, 0, 50, 0, 0.5π]	3	50	-

Where N1 is nodes of Method 1, n1 is number of Simulation Points and N2 is Nodes of Method 2.

Cost Function

```
cost = (X[0][:]-Xt[0][:len_time])**2 + (X[1][:]-Xt[1][:len_time])**2
        + (X[2][:]-Xt[2][:len_time])**2
eps = 0.1
cost = (cost - min(cost))/(max(cost) - min(cost) + eps)

terminal_cost = (X[0][-1]-Xt[0][-1])**2 + (X[1][-1]-Xt[1][-1])**2 +
                (X[2][-1]-Xt[2][-1])**2 + (X[3][-1]-Xt[3][-1])**2 +
                (X[4][-1]-Xt[4][-1])**2 + (X[5][-1]-Xt[5][-1])**2

cost = np.mean(cost)  + terminal_cost
```

Case 1:

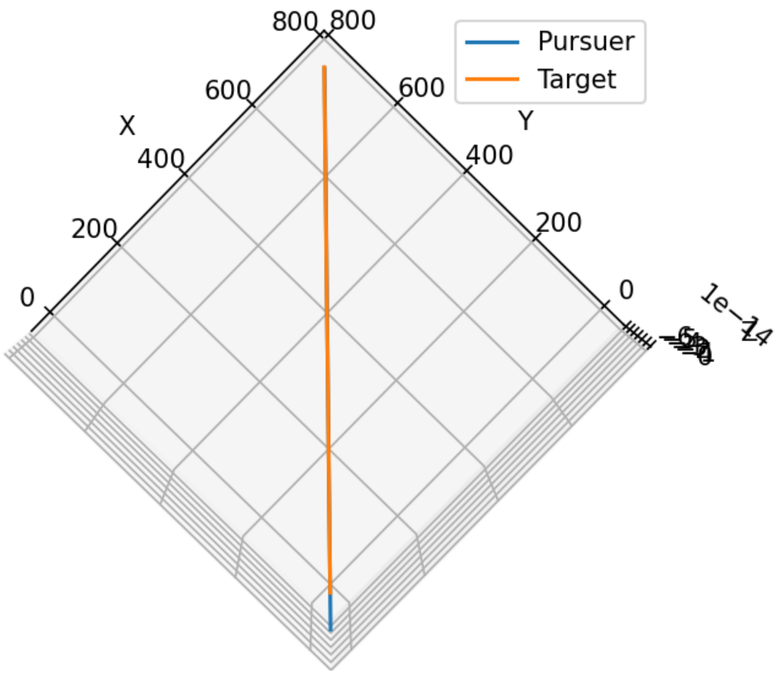


Figure 2.1: Method 1

Case 2:

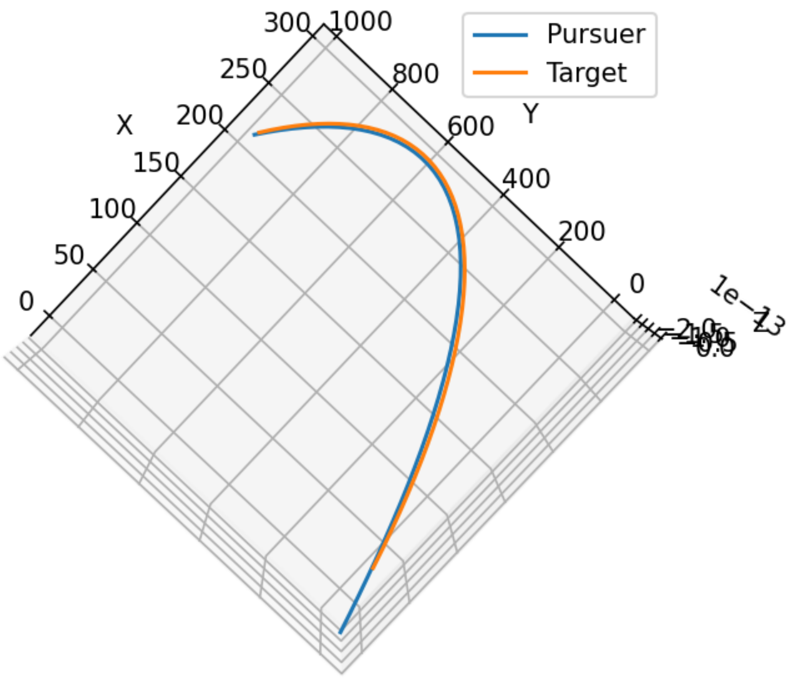


Figure 2.2: Method 1

Case 3:

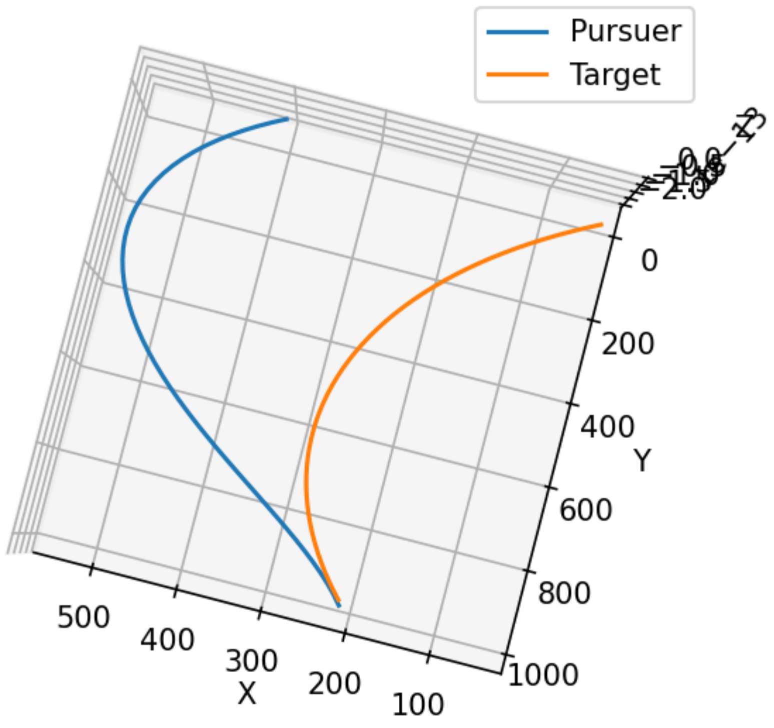


Figure 2.3: Method 1

Case 4:

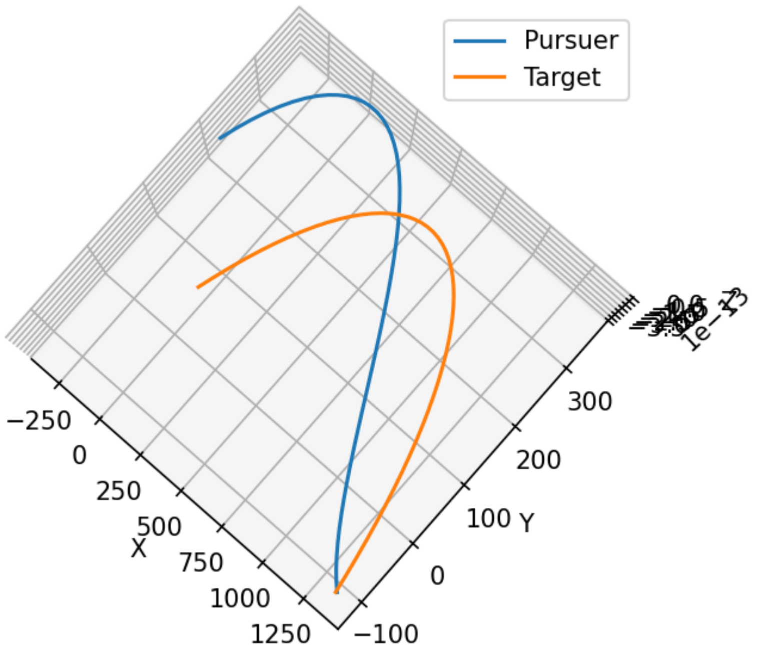


Figure 2.4: Method 1

Case 5:

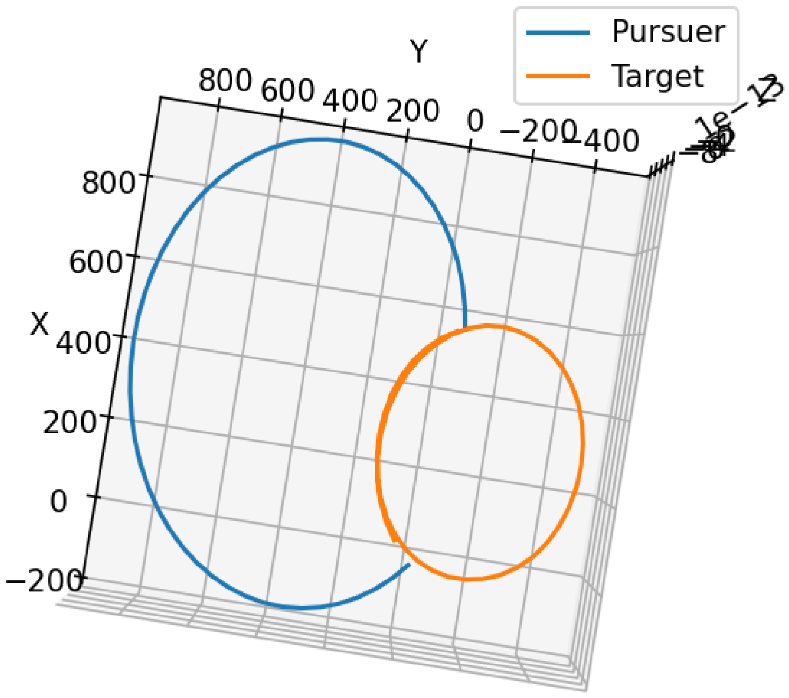


Figure 2.5: Method 1

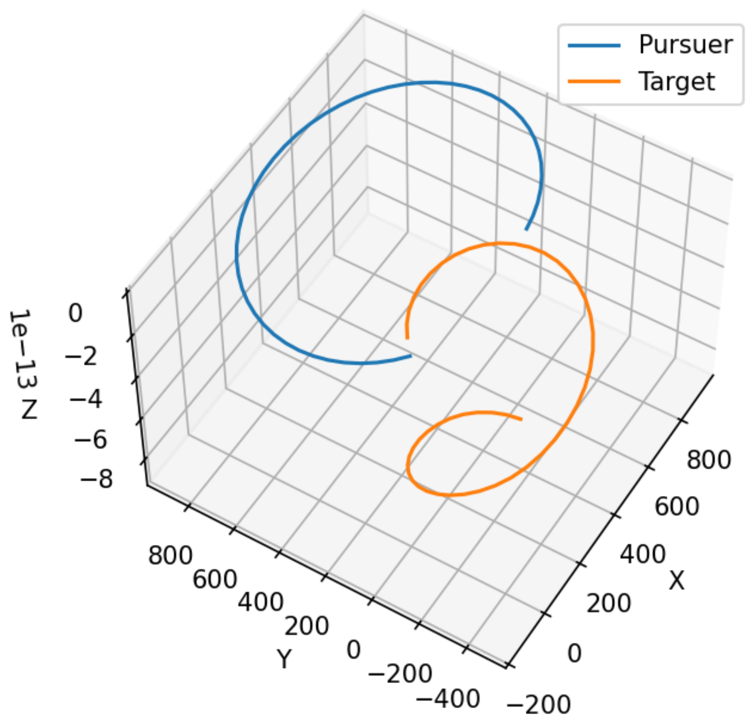


Figure 2.6: Method 1

Case 6:

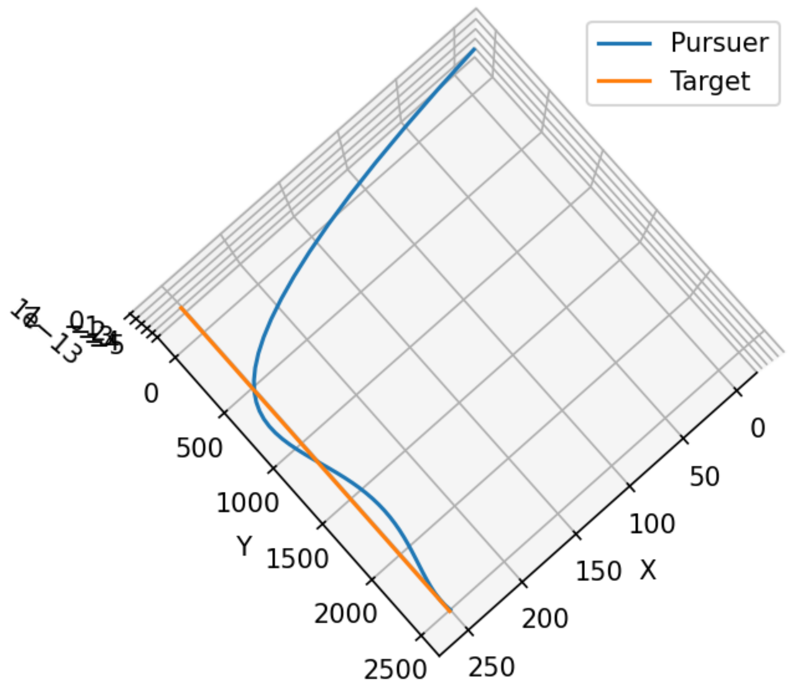


Figure 2.7: Method 1

2.3 MATLAB

The table below summarizes all the cases considered.

Table 2.2: Simulation Parameters

Case	t_0	t_f	U_{target}	X_0 (Pursuer)	X_0 (Target)	N1	n1	N2
1	0	20	[0.0, 0.0, 0.0]	[0, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	40
2	0	20	[0.0, 0.0, 0.3]	[0, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	40
3	0	20	[0.0, 0.0, 0.3]	[400, 0, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	40
4	0	30	[0.0, 0.0, -0.3]	[-300, 200, 0, 50, 0, 0.25π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
5	0	50	[0.0, 0.0, -0.7]	[0, 0, 0, 50, 0, 0.75π]	[50, 50, 0, 50, 0, 0.25π]	3	50	-
6	0	50	[0.0, 0.0, 0.0]	[0, 0, 0, 50, 0, 0.25π]	[250, 0, 0, 50, 0, 0.5π]	3	50	-

where N1 is nodes of Method 1, n1 is number of Simulation Points and N2 is Nodes of Method 2.

Cost Function

```
% Running cost (distance between pursuer and evader over time)
running_cost = sum((xp - xe).^2 + (yp - ye).^2)/ length(t);

% Terminal cost (penalty on final separation)
terminal_cost = ((xp(end) - xe(end))^2 + (yp(end) - ye(end))^2 ;

% Attack Angle Cost: Penalize Pursuer entering Evader's Attack Envelope
Angle_Penalty = {};  % Evader's attack envelope

% Combine the running cost, terminal cost, angle penalty and control effort
y = running_cost/r + 50*terminal_cost + 0.01*CE/length(t) + 100*Angle_Penalty;
```

Case 1:

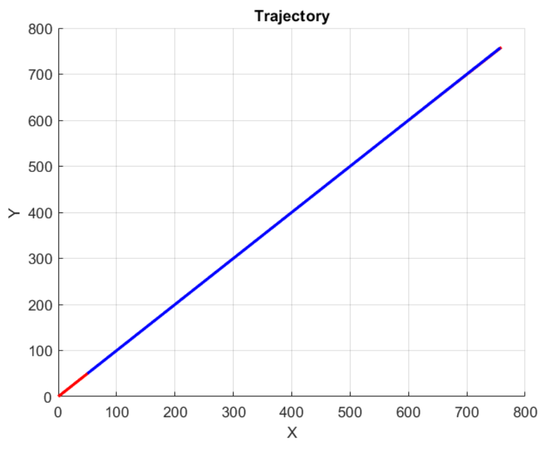


Figure 2.8: Method 1

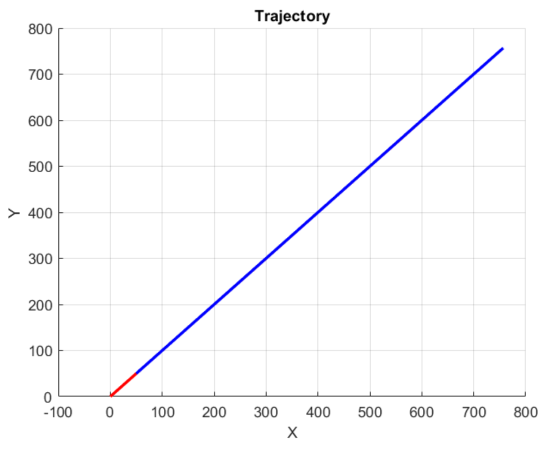


Figure 2.9: Method 2

Case 2:

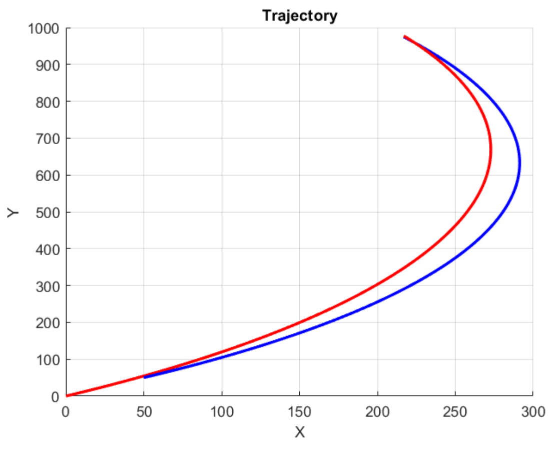


Figure 2.10: Method 1

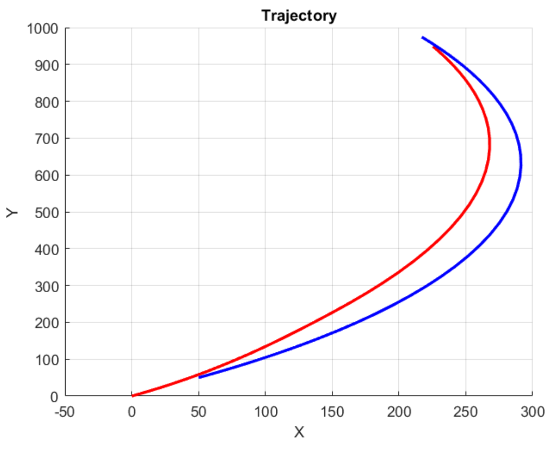


Figure 2.11: Method 2

Case 3:

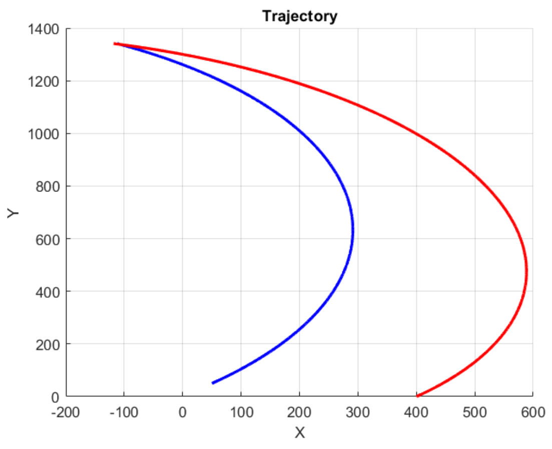


Figure 2.12: Method 1

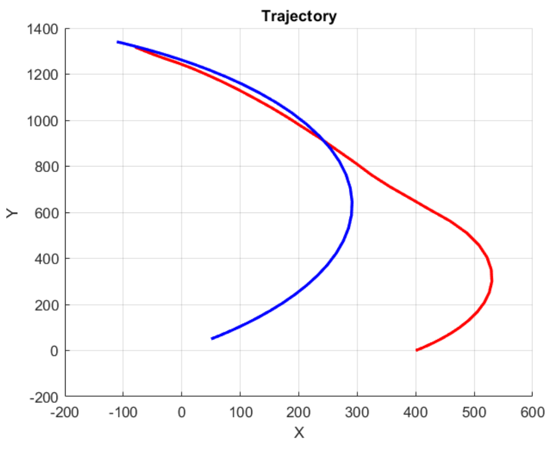


Figure 2.13: Method 2

Case 4:

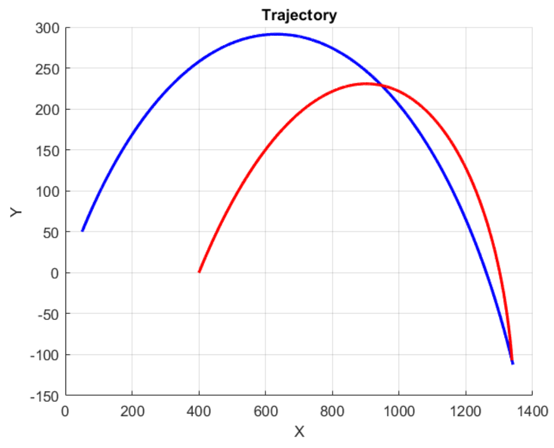


Figure 2.14: Method 1

Case 5:

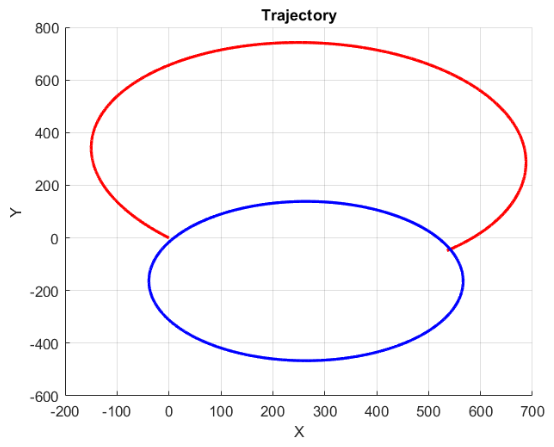


Figure 2.15: Method 1

Case 6:

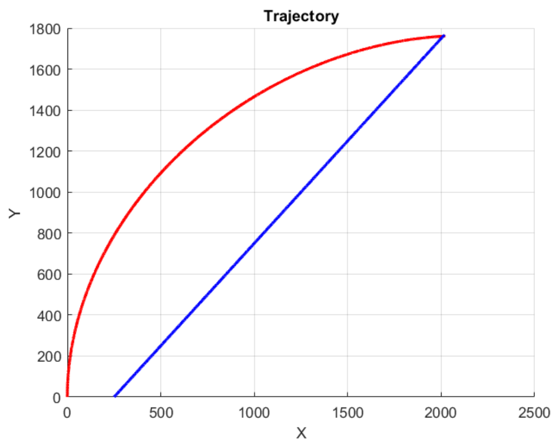


Figure 2.16: Method 1

Chapter 3

Conclusion and Future Work

We show that the optimized trajectory of the pursuer shows a successful approach toward the target while respecting the angle constraint imposed by the attack cone. The angle penalty successfully prevents a direct approach within the critical distance threshold, while control effort penalties maintain smooth maneuvering. Thus, the chosen cost function structure effectively balances distance minimization, control regularity, and angle constraints, offering a feasible solution to the optimal control problem.

We aim to first extend the current formulation to 3-D cases. We also intend to work on introducing active evasion maneuvers for the target, enabling a more dynamic and realistic 2 player model.

Literature Review

- Sani, M., Robu, B. and Hably, A., 2021, November. Pursuit-evasion games based on game-theoretic and model predictive control algorithms. In 2021 International Conference on Control, Automation and Diagnosis (ICCAD) (pp. 1-6). IEEE.
- Cognetti, M., De Simone, D., Patota, F., Scianca, N., Lanari, L. and Oriolo, G., 2017, May. Real-time pursuit-evasion with humanoid robots. In 2017 IEEE International Conference on Robotics and Automation (ICRA) (pp. 4090-4095). IEEE.
- De Simone, D., Scianca, N., Ferrari, P., Lanari, L. and Oriolo, G., 2017, September. MPC-based humanoid pursuit-evasion in the presence of obstacles. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 5245-5250). IEEE.
- Eklund, J.M., Sprinkle, J. and Sastry, S.S., 2011. Switched and symmetric pursuit/evasion games using online model predictive control with application to autonomous aircraft. IEEE Transactions on Control Systems Technology, 20(3), pp.604-620.
- Sprinkle, J., Eklund, J.M., Kim, H.J. and Sastry, S., 2004, December. Encoding aerial pursuit/evasion games with fixed wing aircraft into a nonlinear model predictive tracking controller. In 2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601) (Vol. 3, pp. 2609-2614). IEEE.
- Tzannetos, G., Marantos, P. and Kyriakopoulos, K.J., 2016, June. A competitive differential game between an unmanned aerial and a ground vehicle using model predictive control. In 2016 24th Mediterranean Conference on Control and Automation (MED) (pp. 1053-1058). IEEE.
- Sani, M., Robu, B. and Hably, A., 2021, December. Limited information model predictive control for pursuit-evasion games. In 2021 60th IEEE Conference on Decision and Control (CDC) (pp. 265-270). IEEE.
- Richards, A. and How, J.P., 2003, June. Model predictive control of vehicle maneuvers with guaranteed completion time and robust feasibility. In Proceedings of the 2003 American Control Conference, 2003. (Vol. 5, pp. 4034-4040). IEEE.
- Eklund, J.M., Sprinkle, J. and Sastry, S., 2005, June. Implementing and testing a nonlinear model predictive tracking controller for aerial pursuit/evasion games on a fixed wing aircraft. In Proceedings of the 2005, American Control Conference, 2005. (pp. 1509-1514). IEEE.

- Tian, B., Li, P., Lu, H., Zong, Q. and He, L., 2021. Distributed pursuit of an evader with collision and obstacle avoidance. IEEE Transactions on Cybernetics, 52(12), pp.13512-13520.
- Manoharan, A., Singh, M., Alessandretti, A., Manathara, J.G., Prusty, S.C., Mohanty, N., Kumar, I.S., Sahoo, A. and Sujit, P.B., 2019, July. Nmpc based approach for cooperative target defence. In 2019 American Control Conference (ACC) (pp. 5292-5297). IEEE.
- Kim, H.J., Shim, D.H. and Sastry, S., 2002, May. Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles. In Proceedings of the 2002 American control conference (IEEE Cat. No. CH37301) (Vol. 5, pp. 3576-3581). IEEE.
- Richards, A. and How, J.P., 2002, May. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301) (Vol. 3, pp. 1936-1941). IEEE.
- Shim, D.H., Kim, H.J. and Sastry, S., 2003, December. Decentralized nonlinear model predictive control of multiple flying robots. In 42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475) (Vol. 4, pp. 3621-3626). IEEE.