ROB5

Algorithmic Approaches to Drone Interception in Dynamic Scenarios

PROJECT REPORT

YOUNES BAZI

Table Of Contents

1. Introduction

Interception is the process of capturing, neutralizing, or engaging a moving target, often under dynamic and complex conditions. This concept is widely applied in modern technology, particularly in fields like defense, security, and autonomous systems. From guided missiles intercepting fast-moving threats to drones neutralizing hostile aerial or ground targets, interception plays a critical role in ensuring safety and operational success. In recent years, the rapid development of robotics and autonomous systems has brought interception to the forefront, especially in military applications, where autonomous drones have become central to defense strategies. For instance, the ongoing conflict in Ukraine has demonstrated how drones are not only used for surveillance but also as effective tools for interception, highlighting the need for advanced path planning and decision-making algorithms.

This semester, I chose to work on the topic of interception as part of my robotics project to expand my knowledge and skills in high-level robotics applications. While my previous projects often focused on the electronic components and hardware aspects of building robotic systems, I wanted to take a different approach by delving into the software and algorithmic side of robotics. Specifically, I aimed to work on the high-level application of an already built drone to learn and implement advanced technologies, such as trajectory planning and interception strategies.

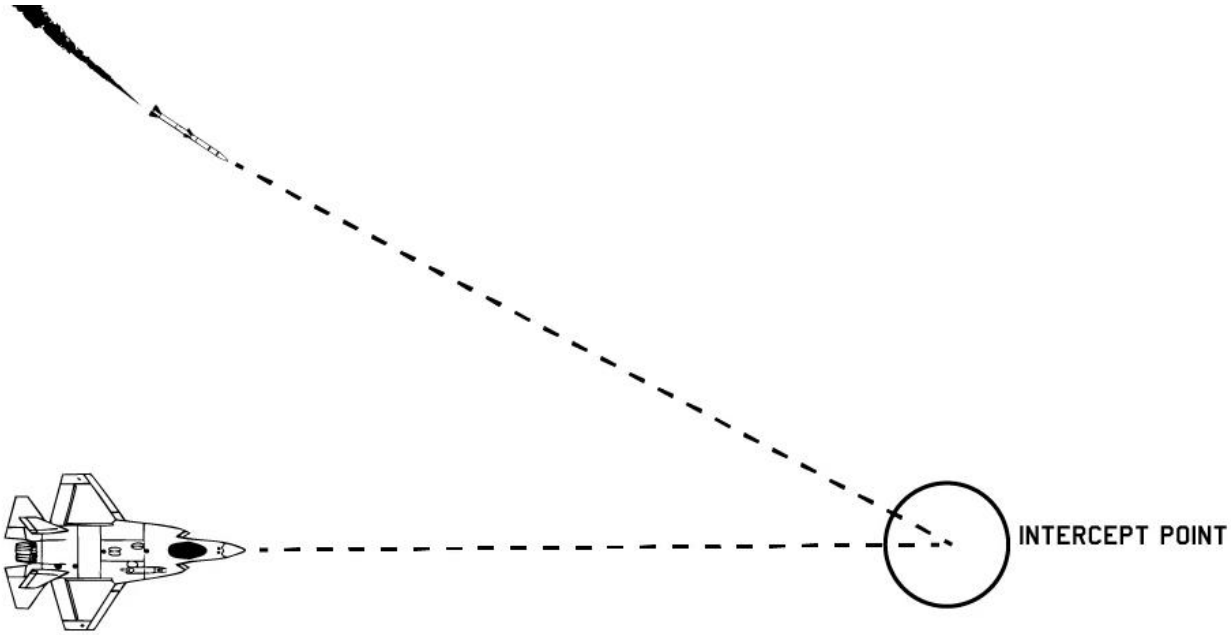

Figure 1: Illustration of the Interception Process and Predicted Intercept Point

During this semester, I focused on understanding and applying new methods to solve dynamic problems in real-time, particularly optimizing drone paths to intercept moving targets. I explored multiple algorithms, analyzing and comparing their performance in terms of efficiency, stability, and speed to identify the most suitable approach for interception scenarios. Building on this theoretical foundation, I implemented a 3D simulation of a drone intercepting a moving target, utilizing one of these algorithms to demonstrate its practical application and effectiveness in a realistic environment.

2. Pursuit Guidance (PG)

During this project, we chose to develop, implement, and test our algorithms in MATLAB. The first algorithm we implemented was **Pursuit Guidance (PG)**, a simple yet intuitive approach to interception.

In Pursuit Guidance, the drone continuously adjusts its heading to aim directly at the current position of the target. The algorithm calculates the line of sight (LOS) between the drone and the target and modifies the drone's velocity vector to minimize the distance along this LOS.

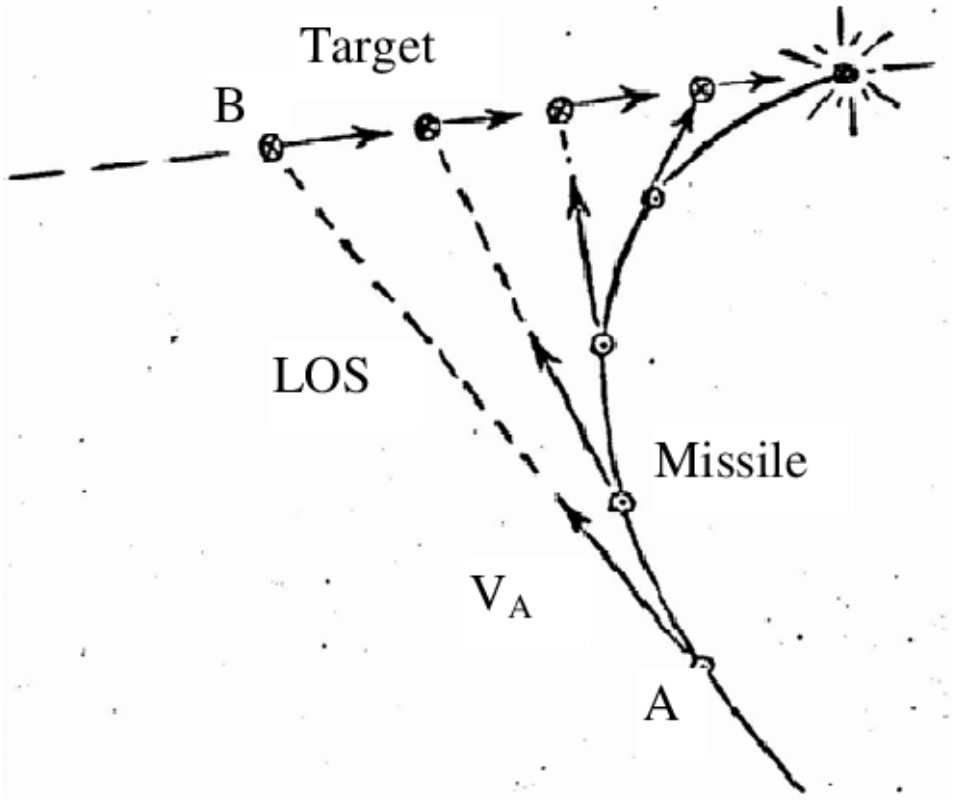

Figure 2: Illustration of Pursuit Guidance: Missile Trajectory Toward Moving Target

This method is straightforward to implement and works effectively when the target's motion is relatively slow or predictable. However, it has limitations when dealing with fast or highly maneuverable targets, as the drone may end up following a curved or inefficient trajectory.

To effectively compare the performance of different path planning algorithms, we tested them on three distinct pursuit scenarios: crossing, head-on, and tail chase. These scenarios represent common interception geometries and provide a standardized framework for evaluating the algorithms under consistent conditions.

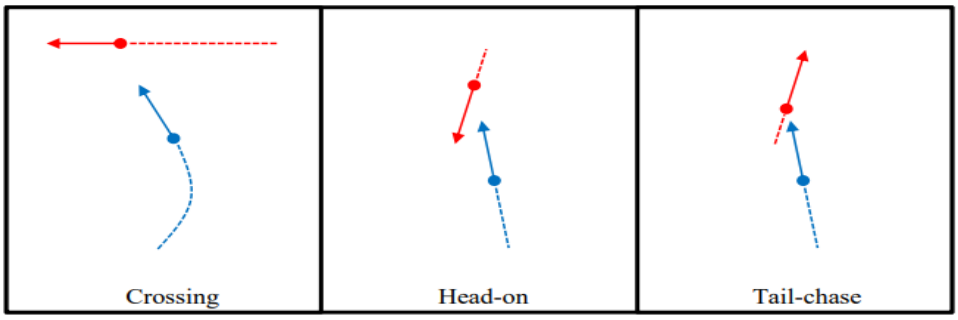

Figure 3: Illustration of Scenarios for Algorithm Testing: Crossing, Head-On, and Tail-Chase

Crossing Scenario: In this scenario, the target moves perpendicular to the drone's initial position, requiring the drone to intercept the target by predicting its path and adjusting its trajectory accordingly. This setup tests the algorithm's ability to handle lateral movements and calculate efficient interception points.

Head-On Scenario: Here, the target moves directly toward the drone's initial position. The drone must adjust its trajectory to intercept the target quickly, often resulting in a direct or nearly straight-line approach. This scenario evaluates the algorithm's efficiency and responsiveness in high-speed, head-on engagements.

Tail Chase Scenario: In this case, the target moves away from the drone, requiring the drone to pursue and catch up with the target. This scenario is particularly challenging, as the drone must optimize its trajectory to minimize the time and energy required for interception.

```
function [dronePath, targetPath, interceptPoint] = simplePursuitGuidance(p0_drone, p0_target, d

    interceptionThreshold = 5;  % Distance threshold for interception (m)
    interceptPoint = [];        % Interception point
    dronePos  = p0_drone;
    targetPos = p0_target;

    for step = 1:maxSteps
        % Update target position
        targetPos = targetPos + v_target * dt;
        LOS = targetPos - dronePos;
        dist   = norm(LOS);
        if dist <= interceptionThreshold
            interceptPoint = targetPos;
            break;
        end
        % Calculate unit direction toward the target
        unitDir = relPos / dist;
        % Update drone position
        dronePos = dronePos + droneSpeed * unitDir * dt;
        % Store positions
        dronePath  = [dronePath, dronePos];
        targetPath = [targetPath, targetPos];
    end
end
```

Figure 5: Code Snippet – Pursuit Guidance Algorithm Implementation

The code above implements the Pursuit Guidance algorithm, where the drone continuously updates its velocity based on the Line of Sight (LOS) to the target. At each time step, the algorithm calculates the LOS vector, which is the relative position from the drone to the target. The direction of the drone's velocity is determined by normalizing this LOS vector, ensuring the drone always moves directly toward the target's current position. The intensity of the drone's velocity remains constant, determined by its fixed speed. By combining this constant speed with the updated direction, the drone's velocity is recalculated at every step, ensuring it adjusts its trajectory dynamically to follow the target.

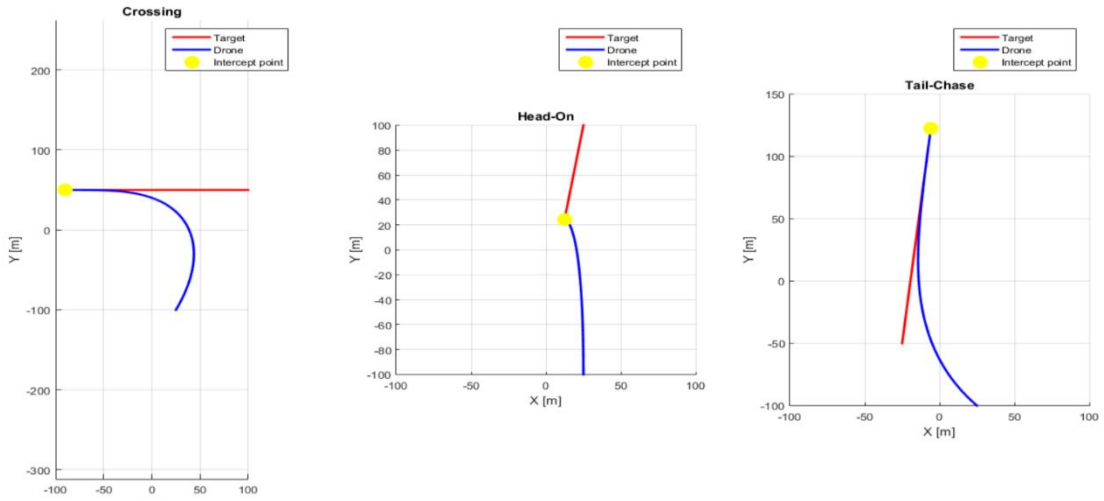

Figure 4: Trajectories Generated by Pursuit Guidance Algorithm in Crossing, Head-On, and Tail-Chase Scenarios

As you can see, the results above demonstrate the effectiveness of the Pursuit Guidance algorithm across three scenarios: Crossing, Head-On, and Tail-Chase. In the Crossing scenario, the drone takes a curved trajectory to intercept the horizontally moving target, successfully meeting at the marked interception point (yellow dot). In the Head-On scenario, the drone follows a near-straight path toward the target, with minimal lateral adjustment due to the slight offset, achieving a direct interception. In the Tail-Chase scenario, the drone dynamically adjusts its heading, following a curved path to catch up with the target moving upward with a horizontal drift. The interception points in all scenarios confirm the algorithm's ability to steer the drone effectively toward the target, ensuring successful interception within the predefined threshold of ±5 m.

3. Proportional Navigation (PN)

Proportional Navigation (PN) is a widely used interception algorithm that predicts the future position of a target and steers the interceptor toward the calculated interception point. Unlike Pursuit Guidance, which directly follows the target's current position, PN calculates the rate of change of the line of sight (LOS) angle between the interceptor and the target. It then applies acceleration proportional to this angular rate to nullify the relative motion along the LOS.

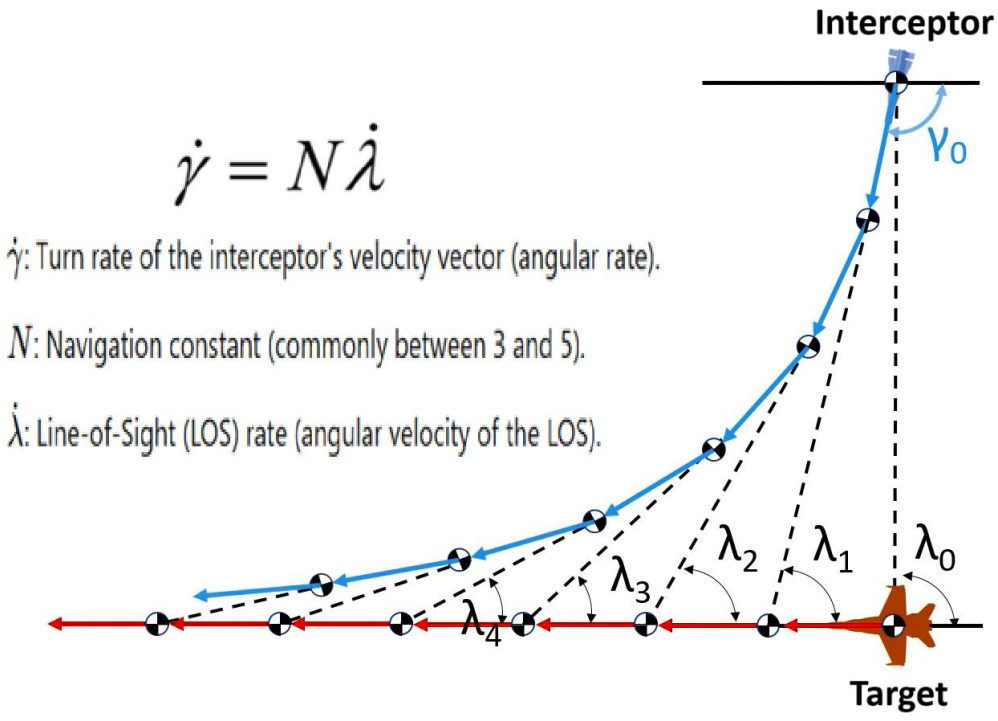

Figure 6: Illustration of Proportional Navigation (PN) Concept

This approach is highly effective for intercepting fast-moving targets with predictable trajectories and is commonly used in missile guidance systems. However, its performance can be affected by noisy measurements or abrupt changes in the target's motion.

```
function [dronePath, targetPath, interceptPoint] = simulateProportionalNavigation( ...
    p0_drone, p0_target, droneSpeed, v_target, dt, timeVec, N)

    interceptPoint = [];  % Interception point (if any)
    threshold = 5;        % Interception distance (m)
    dronePos = p0_drone;  % Initialize drone position
    targetPos = p0_target; % Initialize target position
    LOS_prev = atan2(targetPos(2) - dronePos(2), targetPos(1) - dronePos(1)); % Initial LOS angle

    for k = 2:length(timeVec)
        % Update target position
        targetPos = targetPos + v_target * dt;
        % Compute relative position and LOS
        relPos = targetPos - dronePos;
        dist = norm(relPos);
        LOS = atan2(relPos(2), relPos(1));      % Current LOS angle
        LOS_rate = (LOS - LOS_prev) / dt;       % LOS rate of change
        % Proportional Navigation Turn Rate
        turnRate = N * LOS_rate;
        % Update drone heading based on turn rate
        droneHeading = atan2(relPos(2), relPos(1)) + turnRate * dt;
        unitDir = [cos(droneHeading); sin(droneHeading)];
        % Update drone position
        dronePos = dronePos + droneSpeed * unitDir * dt;
        % Store positions and update LOS for the next step
        dronePath(:,k) = dronePos;
        targetPath(:,k) = targetPos;
        LOS_prev = LOS;
    end
    % Trim unused path entries
    dronePath = dronePath(:, ~isnan(dronePath(1,:)));
    targetPath = targetPath(:, ~isnan(targetPath(1,:)));
end
```

Figure 7: Code Snippet – Proportional Navigation Algorithm Implementation

The code above simulates the Proportional Navigation (PN) guidance algorithm, where the interceptor dynamically adjusts its heading to intercept the moving target. It starts by calculating the Line of Sight (LOS) angle between the drone and the target. At each time step, the LOS rate $d\lambda/dt$ is computed as the rate of change of the LOS angle, and the drone's turn rate $d\gamma/dt$ is updated using the proportional navigation formula: $d\gamma/dt=N*d\lambda/dr$, where N is the navigation constant (typically between 3-5). The drone's new heading is determined by adding the turn rate to its current heading, and its position is updated accordingly using the unit direction vector.

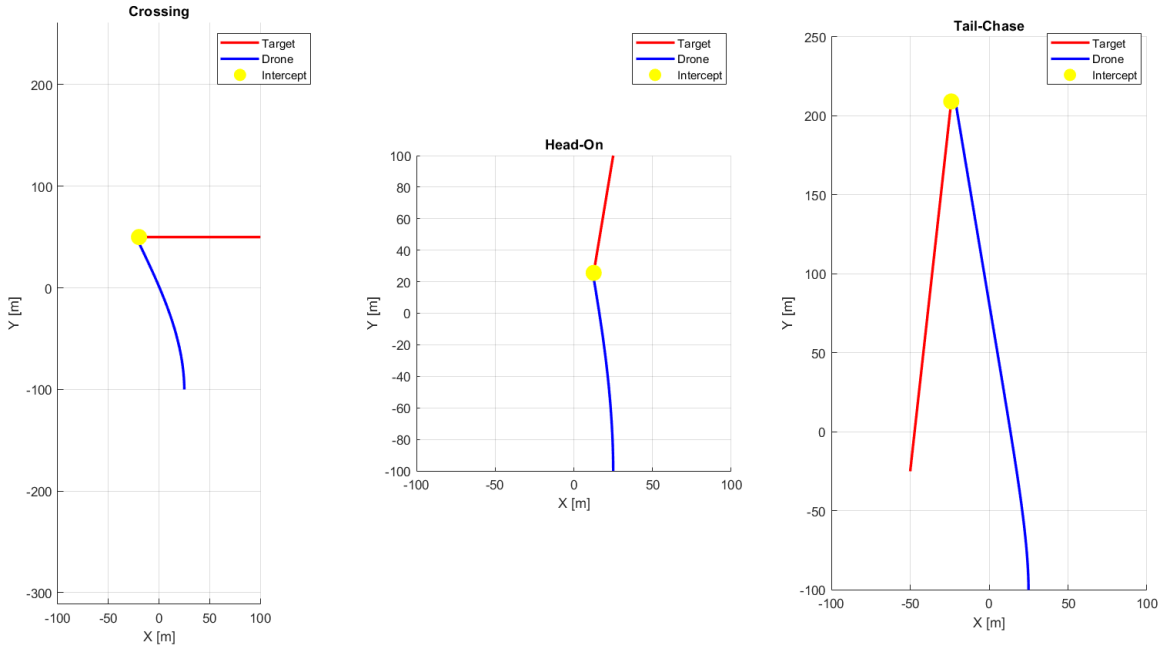

Figure 8: Trajectories Generated by Proportional Navigation Algorithm in Crossing, Head-On, and Tail-Chase Scenarios

As you can see in the results above, the trajectories generated by Proportional Navigation (PN) demonstrate significant differences compared to Pursuit Guidance (PG), especially in the crossing scenario. In PN, the interceptor dynamically adjusts its heading based on the Line of Sight (LOS) rate, leading to a smoother and more direct path toward the interception point. This allows the interceptor to intercept the target more efficiently, reducing the time to interception.

4. Linear Quadratic Tracker (LQT)

The Linear Quadratic Tracker (LQT) is a control method that generates optimal trajectories for a system by minimizing a predefined cost function. This cost function balances two objectives: minimizing the tracking error (the difference between the drone's position and the target's position) and minimizing the control effort (the energy or force required for the drone to adjust its trajectory). In the context of interception, LQT works by continuously comparing the drone's current state (position and velocity) with the target's state and calculating the best control inputs (acceleration or steering commands) to minimize the error. Using Linear Quadratic Regulator (LQR) feedback, the drone dynamically adjusts its velocity and direction to follow a smooth, efficient path toward the target, ensuring interception in the shortest time while avoiding excessive or abrupt movements.

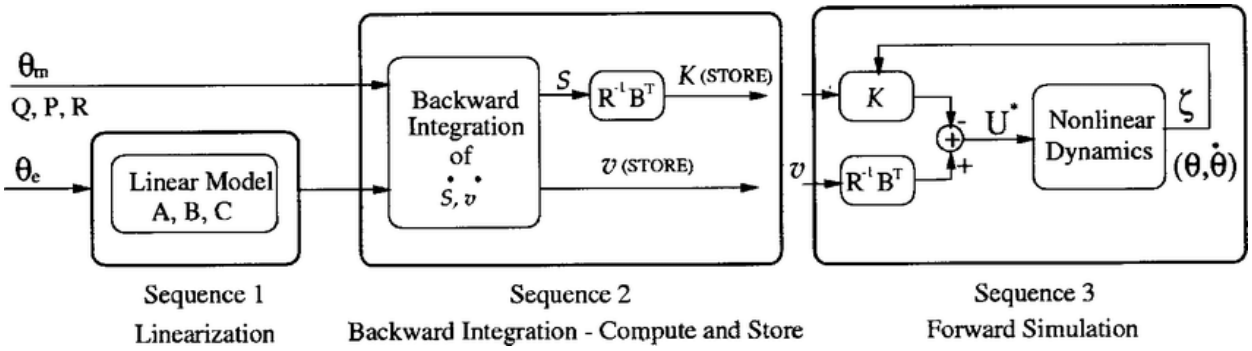

Figure 9: Linear Quadratic Tracker (LQT) Workflow

The system to be controlled is modeled as:

$$\dot{x} = Ax + Bu$$

where:

- x is the state vector, representing the system's current state (e.g., position and velocity),
- dx/dt is the time derivative of the state vector,
- u is the input vector (control inputs),
- A and B are constant matrices defining the system dynamics.

The LQT algorithm minimizes a cost function of the form:

$$J = \int_{t_0}^{t_f} \left[\frac{1}{2}(x(t) - r(t))^T Q(x(t) - r(t)) + \frac{1}{2}u(t)^T R u(t)\right] dt + \frac{1}{2}(x_f - r_f)^T H(x_f - r_f)$$

where:

- r(t) is the desired reference trajectory.
- Q, R, and H are positive-definite weighting matrices.
    - Q penalizes the tracking error between the state x(t) and the reference trajectory r(t).
    - R penalizes the magnitude of the control effort u(t).
    - H penalizes the terminal error at the final state xf.

The Hamiltonian for the cost function is defined as:

$$H(x, u, p, t) = \frac{1}{2}(x(t) - r(t))^T Q(x(t) - r(t)) + \frac{1}{2}u(t)^T R u(t) + p^T(Ax + Bu)$$

where p is the costate vector, representing the sensitivity of the cost to the state.

The necessary conditions for optimality are:

1. System dynamics:

$$\dot{x} = Ax + Bu$$

2. Costate dynamics:

$$\dot{p} = -\frac{\partial H}{\partial x} = -Q(x - r) - A^T p$$

3. Control law:

$$u^* = -R^{-1}B^T p$$

So, the optimal control input for the system is given by:

$$u^*(t) = -R^{-1}B^T p(t)$$

So, after implementing the LQT algorithm in MATLAB, we obtained the following results:

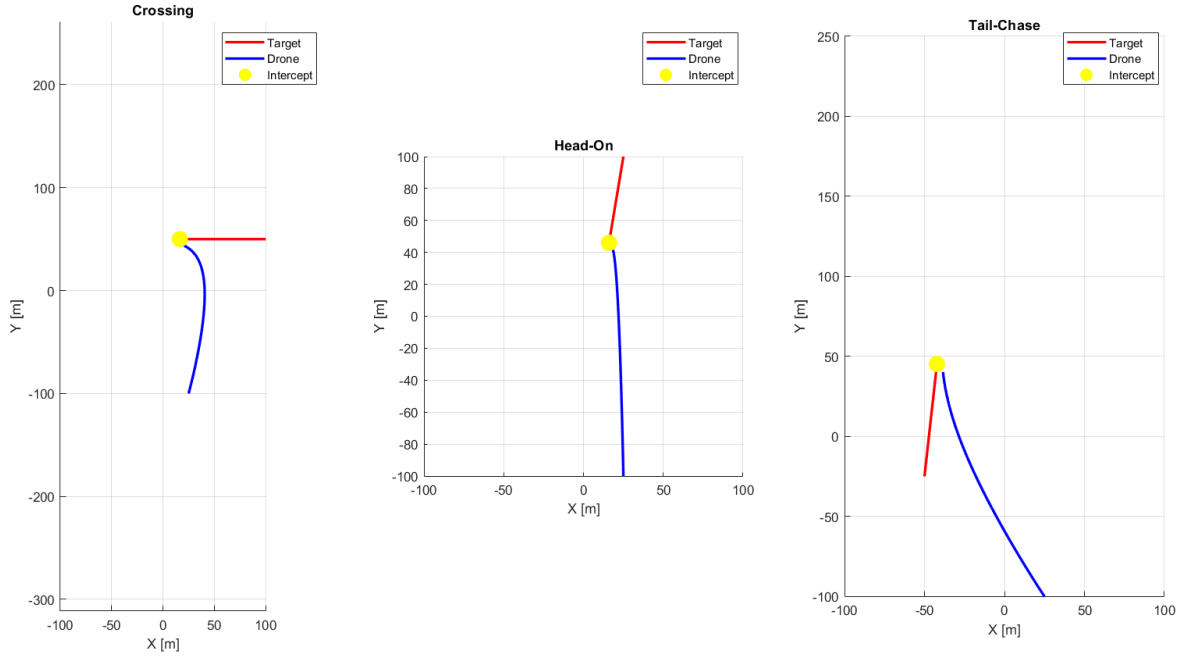

Figure 10: Trajectories Generated by Linear Quadratic Tracker (LQT) Algorithm in Crossing, Head-On, and Tail-Chase Scenarios

As you can see, the results above demonstrate that the Linear Quadratic Tracker (LQT) algorithm produces smoother, more optimal, and faster interception trajectories compared to Proportional Navigation (PN) and Pursuit Guidance (PG). LQT minimizes tracking error and control effort, ensuring efficient and

precise interception, particularly in predictable scenarios. Its key advantage lies in balancing trajectory accuracy and effort efficiency through the well-defined cost function J. However, LQT has a major drawback: it requires a known reference trajectory for the target, making it less effective in cases where the target's motion is unknown or highly unpredictable(even though we can use real-time state estimation, target motion prediction, and probabilistic planning to predict the target motion). This limitation shows the adaptability of PN and PG in dynamic scenarios despite their less optimal paths.

5. Algorithm Comparison and Final Selection

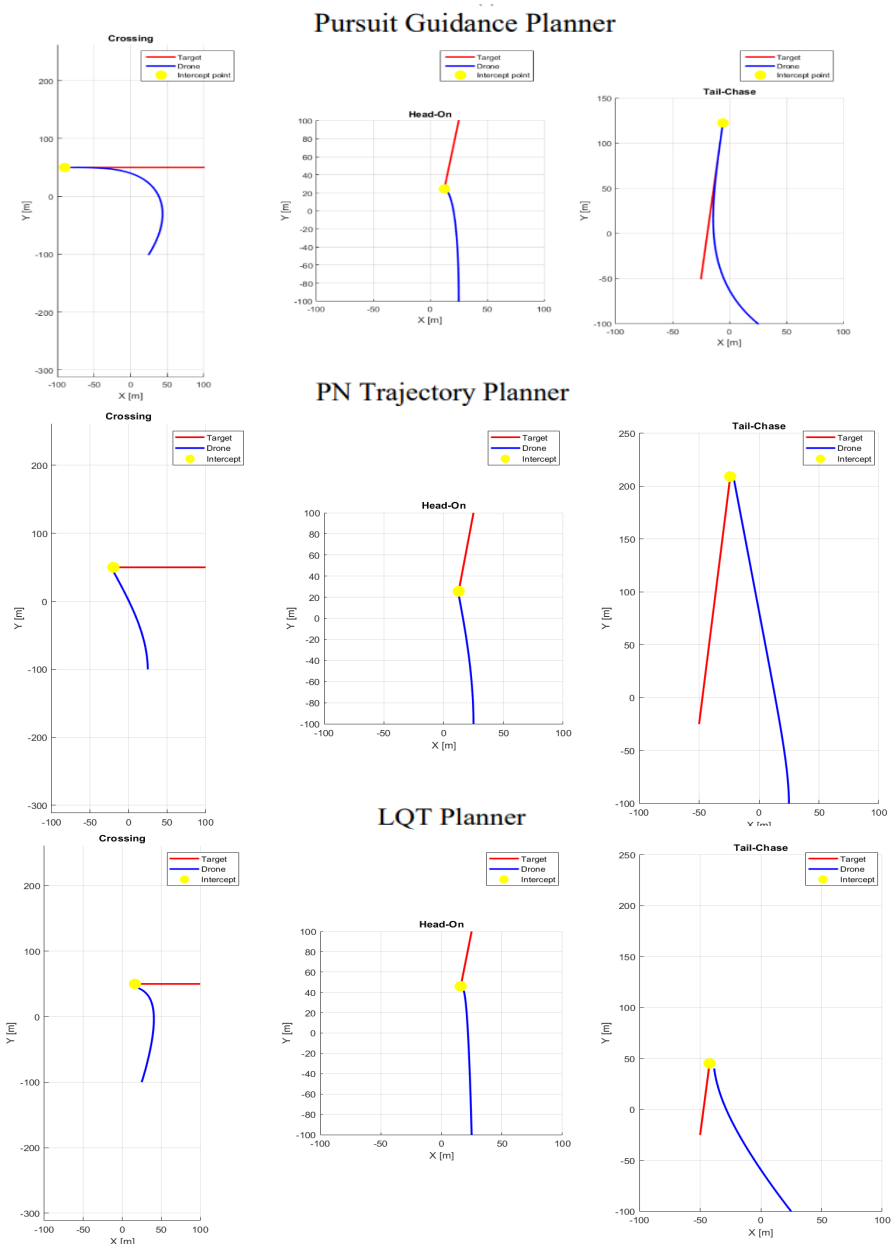

Figure 11: Trajectory Comparisons of Pursuit Guidance, Proportional Navigation, and LQT Planners

The time to intercept (seconds) for each planner and geometry type are shown in Table below.

	PG	PN	LQT
Crossing	10.00	6.30	4.40
Head-on	6.30	6.20	4.50
Tail-chase	9.60	13.00	3.90

Figure 12: Experimental Intercept Times

Below is a table that highlights the advantages and drawbacks of each algorithm:

Algorithm	Advantages	Drawbacks
PG	• Simple and easy to implement. • Robust against uncertainties in target motion. • Computationally inexpensive.	• Generates inefficient trajectories. • Struggles with rapidly maneuvering targets. • Delays in interception for high-speed targets.
PN	• Efficient, smooth, and fast trajectories. • Adapts to dynamic changes in target motion. • Widely used in real-world applications.	• Requires precise tuning of navigation constant (N). • Relies on accurate LOS rate measurements. • Limited predictive capability for highly maneuverable targets.
LQT	• Produces energy-efficient, optimal trajectories. • Predicts future target positions effectively. • Handles both position and velocity errors.	• Computationally demanding due to real-time matrix operations. • Requires accurate reference trajectory for optimal performance. • Struggles with unknown or highly unpredictable target motion.

Figure 13: Advantages and Drawbacks of the 3 Algorithms

As you can see in the table above, each algorithm has its advantages and drawbacks, but we chose to work with the Proportional Navigation (PN) algorithm in the 3D simulation due to its balanced performance and practicality. PN strikes a middle ground by being computationally efficient and adaptable to

various scenarios without requiring precise knowledge of the target's motion. While it is not the fastest algorithm, its simplicity and ability to handle dynamic environments make it a reliable choice for real-time applications. Keep in mind that each of these algorithms is designed for specific target behaviors. For example, Pursuit Guidance (PG) is more suited for slow-moving targets, PN is effective for fast-moving or maneuvering targets with unpredictable motion, and Linear Quadratic Tracker (LQT) excels when the target's motion is predictable and computational resources are not constrained. That said, it isn't entirely scientifically accurate to compare these algorithms directly since they are designed for different purposes and target behaviors. This comparison was simply my humble attempt to learn about interception and path-planning algorithms.

6. 3D Simulation Using Unreal Engine

To simulate the drone in a realistic 3D environment, we chose to use AirSim on Unreal Engine. Unreal Engine was selected due to its advanced rendering capabilities, real-time physics, and ability to create highly realistic and immersive environments for simulation. This provided a visually detailed and dynamic testing ground for our interception algorithms. Unreal Engine works by leveraging real-time graphics rendering and a physics engine to simulate environmental interactions, making it ideal for applications like drone simulations where environmental realism is crucial.

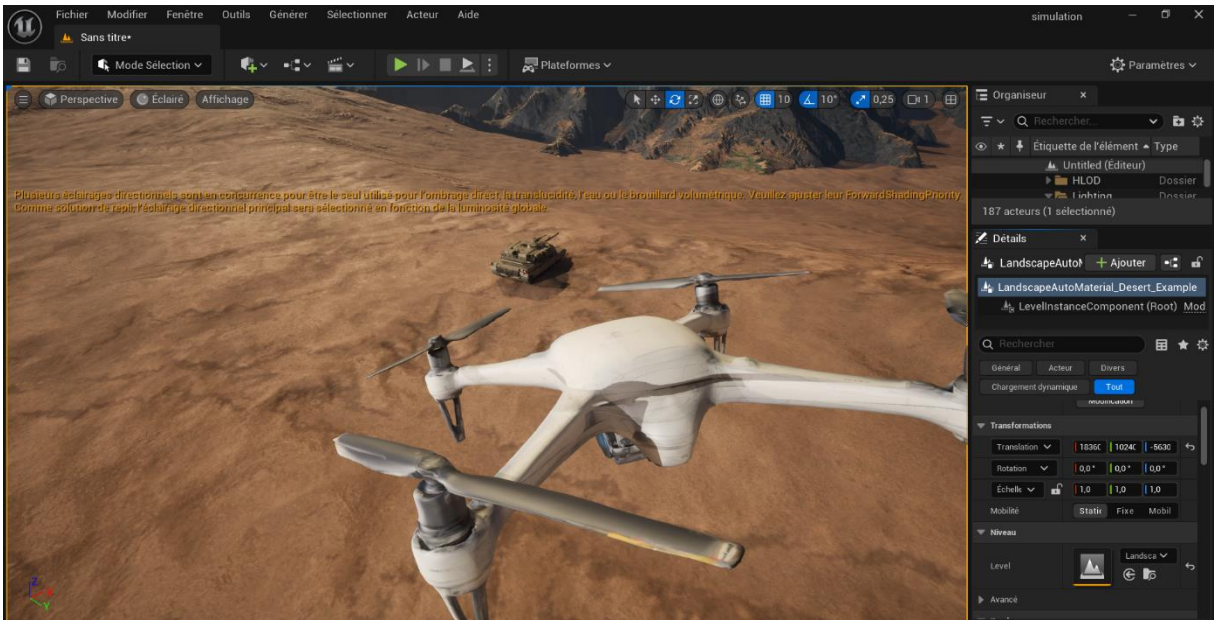

Figure 14: 3D Drone Simulation Environment in Unreal Engine

For controlling the drone, we used MATLAB and established a link with Unreal Engine via AirSim's TCP/IP-based API, where MATLAB sent commands to AirSim, and the simulation environment responded with telemetry data, such as the drone and target's position, velocity, and orientation.

During this project, we did not focus on estimating the target's position using onboard sensors or cameras, as it would require implementing complex sensor models, noise filtering, and estimation techniques such as Kalman filtering. While such methods are essential in real-world applications, they are time-consuming to develop and prone to errors. Instead, we directly accessed the target's position within the Unreal Engine environment using AirSim's API. This approach allowed us to focus solely on evaluating the performance of the interception algorithms without being affected by the complexities of sensor-based target tracking.

Regarding the results, they were satisfying enough given the complexity of the simulation. The drone successfully intercepted the target in approximately 70% of the scenarios. This performance was influenced by the threshold for the interception point, which was set much smaller in the 3D environment compared to the MATLAB simulation. The tighter threshold in AirSim added realism to the simulation but made achieving consistent interceptions more challenging. Despite this, the results demonstrated the effectiveness of the chosen algorithms and provided valuable insights for further improvements in real-world applications.

7. Conclusion

This project provided us with valuable insights into interception algorithms and their practical applications in robotics. By exploring Pursuit Guidance (PG), Proportional Navigation (PN), and Linear Quadratic Tracker (LQT), we gained a deep understanding of their strengths, limitations, and real-world applicability. PN emerged as the most balanced choice for 3D simulation due to its simplicity, efficiency, and adaptability to dynamic environments.

While the project focused on algorithm evaluation, future work could incorporate advanced techniques like Kalman filtering and probabilistic motion prediction to improve target tracking and system accuracy. Additionally, exploring machine learning-based path planning could enhance the adaptability of interception strategies.

It is crucial to emphasize that this project's military interception scenarios were imagined solely for scientific research and study. Our work does not reflect any stance on wars; we strongly wish for PEACE. The project aimed to expand our knowledge and contribute to advancements in robotics for applications such as disaster relief and environmental monitoring. If given the opportunity, future efforts would refine these algorithms and explore their broader applications.