Pursuit Evasion Strategies for Drones in Airsim and Unreal Engine

Dissertation submitted in partial fulfillment of the requirements
for the award of the degree of

Dual Degree (B.Tech + M.Tech.)

by

Siddharth Anand

(Roll No. 20D070076)

Under the Supervision of

Prof. Debraj Chakraborty

Department of Electrical Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

Mumbai - 400076, India

June, 2024

Dissertation Approval

This dissertation entitled **Pursuit Evasion Strategies for Drones in Airsim and Unreal Engine** by **Siddharth Anand**, Roll No. 20D070076, is approved for the degree of **Dual Degree (B.Tech + M.Tech.)** from the Indian Institute of Technology Bombay.

.....
Prof. Dwaipayan Mukherjee
(Examiner 1)

.....
Prof. Harish K. Pillai
(Examiner 2)

.....
Prof. Debraj Chakraborty
(Supervisor)

Defence Date: 23 June 2025

Place: IIT Bombay

Certificate

This is to certify that the dissertation entitled **"Pursuit Evasion Strategies for Drones in Airsim and Unreal Engine"**, submitted by **Siddharth Anand** to the Indian Institute of Technology Bombay, for the award of the degree of **Dual Degree (B.Tech + M.Tech.)** in Electrical Engineering, is a record of the original, bona fide research work carried out by him under our supervision and guidance. The dissertation has reached the standards fulfilling the requirements of the regulations related to the award of the degree.

The results contained in this dissertation have not been submitted in part or in full to any other University or Institute for the award of any degree or diploma to the best of our knowledge.

.....

Prof. Debraj Chakraborty
Department of Electrical Engineering,
Indian Institute of Technology Bombay.

Declaration

I declare that this written submission represents my ideas in my own words. Where others' ideas and words have been included, I have adequately cited and referenced the original source. I declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated, or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will cause disciplinary action by the Institute and can also evoke penal action from the source which has thus not been properly cited or from whom proper permission has not been taken when needed.

. . . . . .

Siddharth Anand
Roll No.: 20D070076
Date: 22 June 2025
Place: IIT Bombay

Abstract

In this work, we develop a real-time vision-based pursuit framework for unmanned aerial vehicles (UAVs) using AirSim and Unreal Engine. The objective is to enable a pursuer drone to track and chase an evader drone using only visual input from an onboard monocular camera. To this end, we propose the Adaptive Camera Guided Control (ACGC) algorithm, which combines heading vector estimation and PID-based yaw and pitch control to keep the evader within the camera's field of view.

We further introduce an adaptive power strategy that dynamically modulates the pursuer's velocity based on depth cues, enabling smoother and more stable chases. Two distinct strategies for estimating depth from camera feed are explored: a scaling-factor-based approach using bounding box size, and a depth image-based method using AirSim's Depth Perspective camera.

Extensive simulations validate the effectiveness of ACGC under different motion profiles. The proposed adaptive power model outperforms static velocity scaling in terms of both chase completion and tracking accuracy, while successfully handling edge cases such as gimbal locking. The full control loop and perception pipeline are implemented within the AirSim simulator, with real-time performance plots and trajectory visualizations.

Index Terms — UAV pursuit, monocular vision, heading estimation, PID control, AirSim, Unreal Engine, adaptive velocity, object tracking, real-time control, gimbal locking

Contents

List of Figures

List of Tables

Abbreviations

UAV	**U**nmanned **A**erial **V**ehicle
ACGC	**A**daptive **C**amera-**G**uided **C**ontrol
GLC	**G**imbal **L**ocking **C**ompensation
PID	**P**roportional-**I**ntegral-**D**erivative (Controller)
FoV	**F**ield **o**f **V**iew
API	**A**pplication **P**rogramming **I**nterface
LOS	**L**ine **o**f **S**ight
FVC	**F**arthest **V**oronoi **C**orner
GLT	**G**imbal **L**ock **T**hreshold
GUT	**G**imbal **U**nlock **T**hreshold
RGB	**R**ed-**G**reen-**B**lue (Image Mode)
NED	**N**orth-**E**ast-**D**own (Coordinate Frame)
UE5	**U**nreal **E**ngine **5**

Chapter 1

Introduction

The increasing accessibility of unmanned aerial vehicles (UAVs) has led to emerging security concerns, especially around sensitive infrastructure and public spaces. Developing autonomous defense systems capable of detecting, tracking, and intercepting unauthorized drones is therefore of critical importance. This thesis explores simulation-based techniques for implementing real-time vision-guided pursuit strategies in such anti-drone systems, with a focus on camera-based tracking, adaptive control, and geometric coordination among multiple agents.

To ensure realism and flexibility in prototyping, we employ Unreal Engine 5.4 for environment rendering and Cosys-Airsim, a drone simulation platform built on top of Microsoft's AirSim, for high-fidelity physics-based UAV control. A prebuilt "City Park Environment" is used as the simulation setting, featuring diverse terrain, occlusions, and realistic urban scenery — ideal for simulating infrastructure defense scenarios.

The central problem addressed in this work is the real-time interception of an evading UAV using only the onboard camera feed of the pursuer drone. We tackle this using a modular vision-control system termed Adaptive Camera-Guided Control (ACGC), which performs continuous tracking of the evader in the image frame, while modulating pursuit

aggressiveness based on depth cues. This is complemented by a gimbal-locking compensation strategy to handle instability during steep vertical maneuvers.

In addition to single-drone pursuit, we also implement a 3D Voronoi-based pursuit strategy where multiple autonomous pursuer drones defend a static asset (such as a fountain) against a manually controlled evader. The evader, operated using an Xbox controller, mimics an adversarial UAV attempting to attack a fixed target. The pursuers dynamically coordinate based on real-time geometrical calculations to intercept the evader before it breaches the defense zone.

This work thus presents a vision-guided autonomous interception pipeline — from system architecture and sensor integration to visual feedback control.

1.1 Problem Statement

Design and implement a camera-based autonomous interception system in a simulated environment, where one or more pursuer drones detect, track, and chase an evading drone using real-time vision and adaptive control logic.

1.2 Need of the Study

Conventional tracking methods often rely on GPS or external positioning systems, which are impractical in many real-world scenarios involving drone threats. A robust anti-drone system must instead rely on onboard perception and control. This study addresses this gap by using only the visual input from onboard cameras and evaluates how depth, geometry, and control coupling affect interception quality in 3D environments.

1.3 Study Objective

This project aims to:

1. Build a realistic simulation platform for UAV pursuit-evasion using Unreal Engine 5.4 and Cosys-Airsim
2. Develop a real-time vision-based tracking system using horizontal and vertical image offsets
3. Implement adaptive power control strategies based on depth estimation and introduce gimbal-locking compensation for robust vertical tracking
4. Extend pursuit strategies to multi-agent coordination using 3D Voronoi geometry and simulate an infrastructure defense scenario, where pursuers autonomously intercept a manually operated evader before it reaches a target

1.4 Dissertation Organization

This dissertation is organized into two parts, each focusing on a distinct project within the broader domain of autonomous drone-based anti-intrusion systems:

- **Part I: Adaptive Camera Guided Control (ACGC)**

  This part presents a real-time, vision-based interception strategy where a pursuer drone uses its onboard camera to track and chase an evading drone. The system includes horizontal (yaw) and vertical (gimbal tilt) control loops, depth estimation using both geometric and sensor-based methods, an adaptive power modulation strategy, and gimbal-locking compensation for stability during steep vertical maneuvers.

- **Part II: Voronoi-Based 3D Pursuit Strategy**

  This part focuses on multi-agent coordination using 3D Voronoi geometry. A team of autonomous pursuers defends a fixed infrastructure (e.g., a fountain) against a manually controlled evader. The evader simulates a hostile UAV, and the pursuers dynamically assign roles and compute interception paths to prevent the evader from reaching its target.

Part I

Adaptive Camera Guided Control (ACGC)

Chapter 2

System Architecture

2.1 Software Stack

The simulation architecture is built using Unreal Engine 5.4 and Cosys-Airsim, creating a robust and high-fidelity platform for real-time UAV testing. Unreal Engine provides a photorealistic rendering environment with modular assets, dynamic lighting, and collision-aware environments. The City Park Environment is used to emulate semi-urban spaces with obstacles like trees, benches, pathways, and a fountain — ideal for testing pursuit-evasion strategies in complex environments.

To simulate drone physics, we employ Cosys-Airsim, an extended version of Microsoft's AirSim. Cosys-Airsim introduces better support for multi-drone control, sensor fidelity, and compatibility with Unreal Engine 5+, making it suitable for running real-time simulations involving perception and control.

For all experiments, we use AirSim's built-in simple_flight drone. This is a lightweight quadrotor simulation model equipped with a default cascaded PID control structure. It supports multiple control modes — such as velocity, angle, and position — and exposes internal sensor data and kinematics via a Python API. One major advantage of

simple_flight is its simulation-agnostic behavior: since it doesn't rely on a dedicated simulation mode, the same logic can be ported directly to physical UAVs.

FIGURE 2.1: Simulation setup in Unreal Engine 5.4 using the City Park Environment

2.2 Onboard Camera Specifications

Each simple_flight drone in AirSim is equipped with a front-facing monochrome RGB camera, referred to as front_center. This camera captures images at a default resolution of 256 × 144 pixels, though this can be configured via settings.json. The camera operates in multiple modes — including RGB, segmentation, and depth — which can be toggled using AirSim APIs.

2.3 AirSim APIs and Object Detection

AirSim offers a robust Python API for interacting with drones, capturing sensor data, and issuing control commands. A particularly useful feature for simulation is AirSim's Object Detection API, which interacts with Unreal Engine's internal Object IDs.

FIGURE 2.2: Raw camera feed from the pursuer drone's front_center camera

In Unreal Engine, every placed actor (like a drone, building, or tree) has an associated mesh name or object ID. AirSim's detection system scans the rendered scene and matches visible objects with the given IDs, allowing the simulation to detect specific entities like an evader drone.

In our setup:

- The evader drone (Drone0) is manually tagged with a mesh name like "Drone0"
- The pursuer drone adds this ID to its detection filter using simAddDetectionFilterMeshName
- It uses the simGetDetections API to return a bounding box and object ID if the evader is visible

This allows reliable visual detection in simulation — especially for testing LOS-based and camera-guided control methods.

2.4 Depth Estimation Using DepthPerspective API

To estimate the distance to the evader, we use AirSim's Depth Perspective mode. When the camera is switched to this mode, instead of RGB colors, the output is a grayscale

FIGURE 2.3: Evader drone detection using AirSim's object detection API

image where the intensity of each pixel corresponds to the depth (distance) from the camera to the first object hit by a ray.

Internally, Unreal Engine performs raycasting for each pixel — emitting a ray from the camera origin through the image plane and recording the distance to the first obstacle encountered. This produces a dense depth map that can be processed in real-time.

Chapter 3

Visual Pursuit Strategy Using Monocular Camera

3.1 Objective and Task Breakdown

The core objective of the **Adaptive Camera-Guided Control (ACGC)** system is to enable a pursuer drone to visually track and chase an evader drone using only the onboard camera feed.

The key requirement is that the evader must remain within the **field of view (FoV)** of the pursuer drone throughout the chase. This ensures continuous perception, enabling the pursuer to update its heading and adapt its motion in real time.

To achieve this, the tracking task is broken down into the following three modules:

1. Estimating the Heading Direction Vector

The first step is to estimate the relative heading direction from the pursuer to the evader using its position in the image frame. Given the evader's pixel location, the system

computes a direction vector $\hat{d}_g$ in the global frame that points from the pursuer to the evader.

This involves a series of coordinate transformations:

- Convert pixel offsets (e_x, e_y) into a unit direction vector in the **camera frame**.
- Transform the direction vector from the camera frame to the **drone body frame** using camera extrinsic parameters.
- Convert the body frame vector to the **global frame** using the drone's orientation quaternion obtained via AirSim API.

This transformed heading vector $\hat{d}_g$ serves as the direction along which the pursuer is commanded to move during the chase.

2. Yaw Control for Horizontal Alignment

To ensure that the drone body always faces the evader, a yaw control loop is employed. The horizontal offset of the evader in the image plane (e_x) is used as the input to a PID controller that adjusts the drone's yaw rate.

This alignment ensures that the forward direction of the pursuer is continuously reoriented to face the evader, allowing effective forward pursuit.

3. Camera Tilt Control for Vertical Alignment

To maintain vertical alignment, the onboard gimbal-mounted camera is controlled independently using the vertical image error e_y. A second PID controller adjusts the camera's pitch angle to keep the evader vertically centered in the image.

This decoupled control ensures that vertical tracking does not require the drone to tilt its entire body, making the system smoother and more stable during 3D maneuvers.

Together, these three components form the basis of the ACGC pipeline. The heading direction governs where to go, while yaw and camera tilt ensure the evader remains visually centered.

3.2 Full Visual Tracking using Yaw and Tilt PID Control

To ensure continuous tracking of the evader in the camera frame, the ACGC system employs two independent PID controllers—one for yaw (horizontal alignment) and another for camera tilt (vertical alignment). These controllers use pixel offset errors computed from the image feed to adjust the drone body and gimbal angles in real time, ensuring the evader remains centered in the frame throughout the chase.

Yaw Control for Horizontal Alignment

Yaw control addresses the horizontal drift of the evader in the image plane. The horizontal image error, denoted as $e_x = x_e - x_c$, is defined as the difference between the x-coordinate of the evader's bounding box center and the horizontal center of the image.

This error is passed to a PID controller that commands a corrective yaw rate to the drone body:

$$\dot{\psi}_B[t] = K_{p,\text{yaw}} \cdot e_x[t] + K_{i,\text{yaw}} \cdot \sum_{\tau=0}^{t} e_x[\tau] \cdot \Delta t + K_{d,\text{yaw}} \cdot \frac{e_x[t] - e_x[t-1]}{\Delta t}$$

This feedback loop continuously reorients the drone so that its forward-facing direction aligns with the evader's position in the horizontal axis.

Camera Tilt Control for Vertical Alignment

The vertical offset of the evader in the image, denoted as $e_y = y_e - y_c$, is handled by a separate PID controller responsible for adjusting the gimbal-mounted camera's pitch.

This pitch correction ensures that the evader remains vertically centered even if it moves significantly above or below the pursuer. The pitch control equation follows:

$$\dot{\theta}_C[t] = K_{p,\text{cam}} \cdot e_y[t] + K_{i,\text{cam}} \cdot \sum_{\tau=0}^{t} e_y[\tau] \cdot \Delta t + K_{d,\text{cam}} \cdot \frac{e_y[t] - e_y[t-1]}{\Delta t}$$

Since AirSim does not allow direct rate commands for camera pitch, we integrate:

$$\theta_C[t] = \theta_C[t-1] + \dot{\theta}_C[t] \cdot \Delta t$$

This decoupling of vertical tracking from body motion enhances stability and allows fine-grained control of the visual axis.

Combined Effect: Full Visual Lock

By coupling the yaw and tilt PID controllers, the ACGC system achieves full visual lock on the evader, maintaining it near the center of the camera frame. This ensures that the heading direction vector $\hat{d}_g$ is always computed reliably from centered image coordinates, making downstream velocity control more stable and accurate.

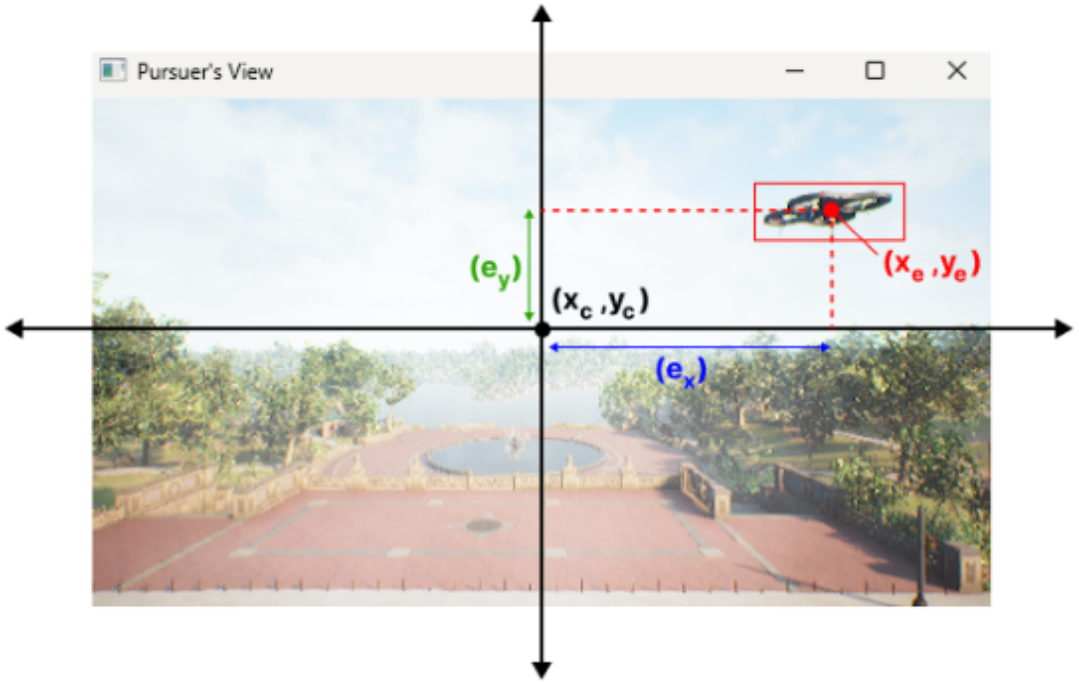

FIGURE 3.1: Illustration of evader detection and pixel error computation. The evader's bounding box is shown in red, and its centroid (x_e, y_e) is compared against the image center (x_c, y_c) to compute the horizontal (e_x) and vertical (e_y) pixel offsets. These offsets are used as inputs to the yaw and tilt PID controllers for real-time alignment

3.3 Heading Vector Estimation

Once the pixel coordinates of the evader have been detected in the camera image, the next task is to compute the 3D direction vector from the pursuer to the evader, expressed in the **global coordinate frame**. This heading vector, denoted $\hat{d}_g$, is essential for issuing velocity commands to the pursuer drone.

The process involves three main stages: computing the direction vector in the camera frame, transforming it to the drone body frame, and finally rotating it into the global frame.

Direction Vector in the Camera Frame

Let (x_e, y_e) be the pixel coordinates of the evader and (x_c, y_c) be the center of the image. The horizontal and vertical offsets are denoted by $e_x = x_e - x_c$ and $e_y = y_e - y_c$, respectively. Let D be the depth of the evader at that pixel location, obtained via depth estimation.

Using the standard pinhole camera model, the relative direction vector in the **camera frame** is given by:

$$\hat{d}_c = \begin{bmatrix} D \\ \frac{D \cdot e_x}{f_x} \\ \frac{-D \cdot e_y}{f_y} \end{bmatrix}$$

where f_x and f_y are the focal lengths of the camera in pixel units.

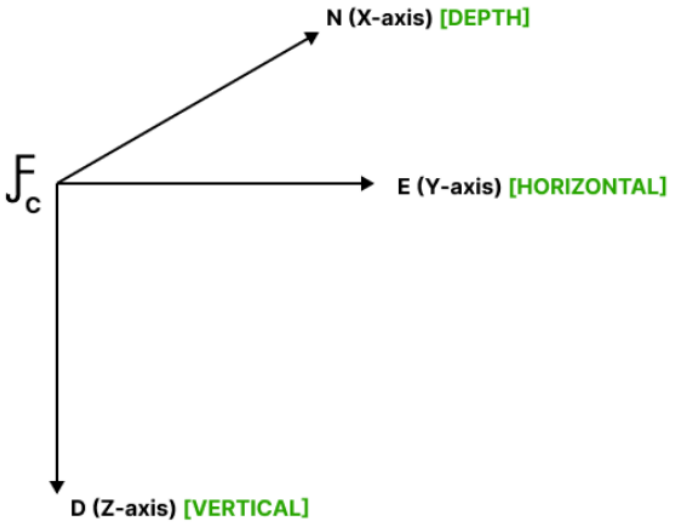

FIGURE 3.2: Camera frame showing the depth axis (N), horizontal (E), and vertical (D) directions. The estimated direction vector $\hat{d}_c$ originates from the image-space offsets

Transformation to the Body Frame

The camera is gimbal-mounted and tilted by a known pitch angle θ_c. To rotate the direction vector into the drone's body frame, we apply a rotation matrix $R_y(\theta_c)$ about the body-y axis:

$$\hat{d}_b = R_y(\theta_c) \cdot \hat{d}_c$$

$$R_y(\theta_c) = \begin{bmatrix} \cos\theta_c & 0 & \sin\theta_c \\ 0 & 1 & 0 \\ -\sin\theta_c & 0 & \cos\theta_c \end{bmatrix}$$

This accounts for the tilt between the camera and the body frame.

Transformation to the Global Frame

The final step is to convert the body-frame vector to the global frame using the drone's orientation quaternion $q = (w, x, y, z)$. The quaternion is converted into a rotation matrix $R(q)$, and the transformation is:

$$\hat{d}_g = R_{\text{body}}^{\text{global}} \cdot \hat{d}_b$$

$$R(q) = \begin{bmatrix} 1 - 2(y^2 + z^2) & 2(xy - zw) & 2(xz + yw) \\ 2(xy + zw) & 1 - 2(x^2 + z^2) & 2(yz - xw) \\ 2(xz - yw) & 2(yz + xw) & 1 - 2(x^2 + y^2) \end{bmatrix}$$

This yields the heading vector $\hat{d}_g$ in global coordinates, which is used to compute chase velocity commands.

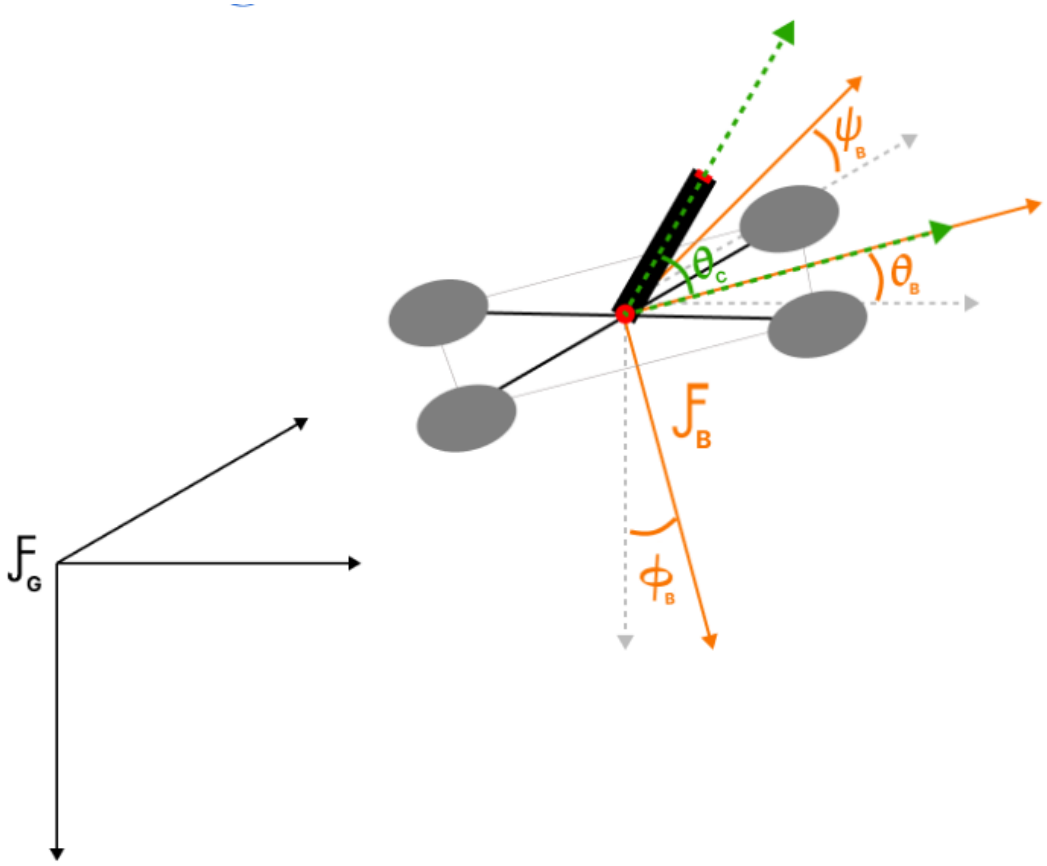

FIGURE 3.3: Transformation of the direction vector from the camera frame to the body frame and finally to the global frame, using known tilt and quaternion orientation

This complete transformation pipeline ensures that the pursuer always computes its motion direction in the correct global reference frame, based only on the image-space location of the evader.

3.3.1 Tracking Metrics

To quantitatively evaluate the performance of the Adaptive Camera-Guided Control (ACGC) system, we use two key tracking metrics: **Live Angle Error** and **True Distance to Evader**. These metrics help assess how accurately the pursuer is estimating the evader's direction and maintaining proximity.

Live Angle Error

This metric measures the angular deviation between the estimated heading vector $\hat{d}_g$, obtained from the image-space computations, and the true heading vector $\vec{d}_g$, derived from the ground-truth global coordinates.

It is defined as:

$$\text{Angle Error} = \cos^{-1}\left(\frac{\hat{d}_g \cdot \vec{d}_g}{\|\hat{d}_g\| \cdot \|\vec{d}_g\|}\right)$$

The result is reported in degrees and reflects the accuracy of the visual estimation pipeline.

True Distance to Evader

This metric captures the Euclidean distance between the pursuer and the evader in the global coordinate frame. It is computed using:

$$\|\vec{d}_g\| = \sqrt{(x_e - x_p)^2 + (y_e - y_p)^2 + (z_e - z_p)^2}$$

where (x_e, y_e, z_e) and (x_p, y_p, z_p) represent the positions of the evader and the pursuer, respectively.

3.4 Adaptive Power Modulation via Depth Estimation

While pixel-based tracking ensures that the evader remains centered in the field of view, the aggressiveness of the pursuer's motion must also be tuned based on how far the evader is. In the earlier control strategy, the pursuer chased the evader along a unit direction vector, without modulating speed based on distance. This static behavior caused either overshooting or sluggish responses, depending on how far the evader was.

To address this, the **Adaptive Camera-Guided Control (ACGC)** framework incorporates a *power modulation strategy* that adjusts the magnitude of the chase velocity based on estimated depth. The idea is simple: the farther the evader, the more aggressively we should pursue it; the closer it is, the more carefully we should approach.

Logarithmic Power Scaling Function

To modulate the aggressiveness of pursuit based on depth, we define a logarithmic power scaling function that varies smoothly between a minimum and maximum power level:

$$p = p_{\min} + \left(\frac{\log(1 + \|\hat{d}_g\|)}{\log(1 + \|\hat{d}_g\|) + 1} \right) \cdot (p_{\max} - p_{\min})$$

In our application:

- $p_{\min} = 1.0$ (set equal to evader's speed)
- $p_{\max} = 2.5$

Thus, the final expression becomes:

$$p = 1 + \left(\frac{\log(1 + \|\hat{d}_g\|)}{\log(1 + \|\hat{d}_g\|) + 1} \right) \cdot 1.5$$

Here, $\|\hat{d}_g\|$ is the estimated depth from the pursuer to the evader (obtained using one of the strategies described earlier). The output power p is then multiplied with the unit direction vector $\hat{d}_g$ to obtain the final velocity command:

$$\vec{v}_{\text{cmd}} = p \cdot \hat{d}_g$$

3.5 Depth Estimation

Depth Estimation: Strategy 1 – Scaling-Based Approximation

In the first approach, we assume the physical width w_{real} of the evader is known. The width of its bounding box in pixels w_{px} can then be used to estimate depth:

$$D_{\text{scale}} = \frac{f \cdot w_{\text{real}}}{w_{\text{px}}}$$

where f is the focal length of the camera (in pixels). While this method is simple and lightweight, it is sensitive to bounding box fluctuations and assumes known geometry and alignment.

Depth Estimation: Strategy 2 – DepthPerspective API

The second approach uses AirSim's `DepthPerspective` camera mode. A grayscale depth image is captured, and the pixel at the center of the bounding box is sampled for depth:

$$D_{\text{depth}} = \texttt{depth\_image}(x_e, y_e)$$

This method is more accurate and less sensitive to bounding box noise but relies on clean segmentation and line-of-sight.

Simulation Results and Comparison

A series of chase simulations were conducted using static power, adaptive power with scaling-based depth, and adaptive power with depth camera. The results clearly show faster interception and smoother behavior under adaptive strategies.

TABLE 3.1: Comparison of Chase Strategies

Strategy	**Avg. Chase Time (s)**	**Notes**
Static Power ($p = 1.5$)	28.5	No depth modulation
Static Power ($p_{mean} = 2$)	17.2	No depth modulation
Adaptive Power	**15.1**	Most stable tracking

Simulation performance table

Also, a simulation with static power $p = 2.5$ shows significant overshoot in the depth axis.

Simulation Demonstration: Adaptive Power

The full video simulations comparing all strategies are available at: https://drive.google.com/drive/folders/1vgD6-SRPwKMvw_B8PcCDiElMA-xTkrXO?usp=sharing

3.6 Gimbal Locking Compensation (GLC)

3.6.1 Cause of Gimbal Locking

Gimbal locking in the pursuer drone's camera system occurs when the evader moves vertically and the camera attempts to keep it centered by applying a large tilt (pitch) correction. The vertical pixel error e_y serves as the input to the pitch PID controller, which responds by commanding the gimbal to tilt upward or downward.

If the evader moves high into the top of the camera's field of view (i.e., large e_y), the tilt controller may command an excessive correction. When this pitch angle exceeds the mechanical limits of the gimbal (e.g., $\theta_c > 90°$), it can lead to unstable or undefined camera behavior — a condition known as **gimbal locking**.

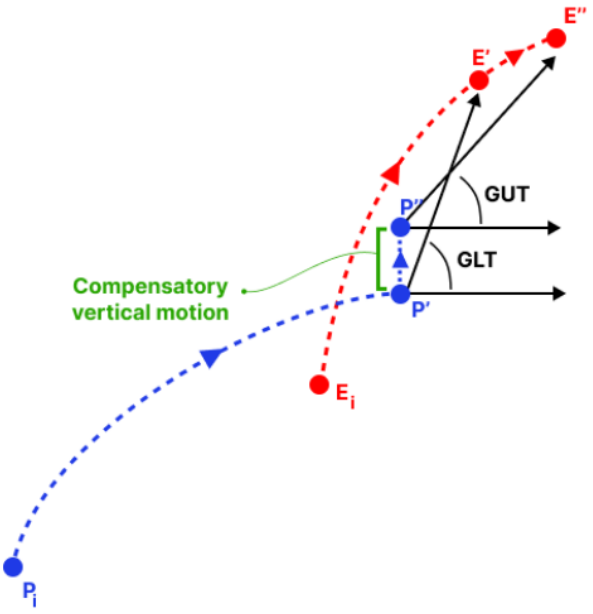

FIGURE 3.4: Gimbal locking arises when vertical tracking requires pitch angles that exceed the gimbal's physical limits, often due to high vertical displacement of the evader

3.6.2 Gimbal Locking Compensation Strategy

To mitigate this issue, the system includes a compensatory mechanism that modifies the drone's own vertical motion to relieve excessive tilt. Instead of relying purely on the gimbal, the pursuer temporarily redirects its velocity along the vertical (z) axis to reduce the commanded pitch.

We define two experimentally determined thresholds:

- **GLT (Gimbal Lock Threshold):** The critical tilt angle above which locking is likely. If $\theta_{\text{tilt}} \geq \text{GLT}$, compensatory vertical motion is triggered.
- **GUT (Gimbal Unlock Threshold):** The safer tilt angle below which tilt is resumed. The drone climbs or descends until $\theta_{\text{tilt}} \leq \text{GUT}$.

For typical aggressiveness levels (i.e., Power $\in (1, 2.5)$), the values are set as:

$$\text{GLT} = 70^\circ, \quad \text{GUT} = 60^\circ$$

Note: This logic assumes the evader is above the pursuer. If the pursuer is above the evader, vertical descent is triggered instead.

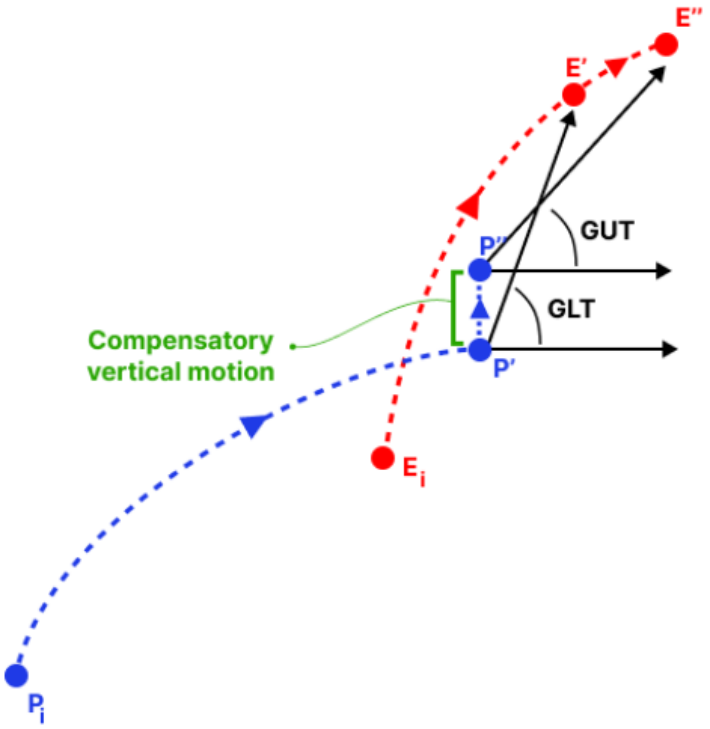

FIGURE 3.5: Gimbal locking compensation logic: the drone climbs to reduce camera tilt when pitch exceeds GLT, and resumes tracking once tilt falls below GUT.

Simulation Demonstration: Gimbal Locking Compensation (GLC)

The full video simulations comparing all strategies are available at: https://drive.google.com/drive/folders/1tLV9xlbz8-AYGCpOn-BDob6NQro1LIBr?usp=sharing

Part II

Voronoi-Based Pursuit Strategy

Chapter 4

Voronoi-Based Pursuit Strategy

In this part of the project, we focus on a multi-agent pursuit-evasion strategy based on geometric principles in 3D space. The goal is to enable multiple autonomous pursuer drones to collaboratively intercept an evader drone using Voronoi region computations. This strategy is later extended to an infrastructure defense scenario, where the pursuers must prevent the evader from reaching a predefined target location.

4.1 Theory of 3D Voronoi Pursuit Strategy

The core idea of the Voronoi-based pursuit is to partition the space such that each pursuer occupies a unique Voronoi region, and the evader is influenced by the geometry of these regions. In 3D, the Voronoi region of the evader becomes bounded only when it lies inside the convex hull of at least four non-coplanar pursuers.

4.1.1 Voronoi Region Geometry in 3D

The Voronoi region of a point in 3D space is bounded if it lies within the convex hull of its neighbors. For the evader to be captured, it must lie inside the tetrahedron formed by

any four pursuers.

- **Capture Condition:** If the evader lies inside the convex hull of the pursuers, it will eventually be captured.
- **Escape Condition:** If the evader lies outside the convex hull, it has at least one escape direction.

$$E \notin \text{ConvHull}(\{P_1, P_2, ..., P_n\}) \quad \Rightarrow \quad \text{Evader can escape}$$

4.1.2 Voronoi-Based Pursuit Strategy

The evader chooses to move toward its farthest Voronoi corner, attempting to escape. The pursuers follow two distinct roles:

- **Primary Pursuer (PPP):** Moves directly toward the evader using a pure pursuit approach.
- **Active Pursuers (APs):** Move to reduce the Voronoi region by heading toward specific Voronoi vertices.

Let V_{ijkE} denote the Voronoi vertex formed by three pursuers P_i, P_j, P_k and the evader E. Each Active Pursuer P_s moves according to:

$$\vec{u}_s = -v_{\max} \cdot \frac{P_s - V_{ijkE}}{\|P_s - V_{ijkE}\|}$$

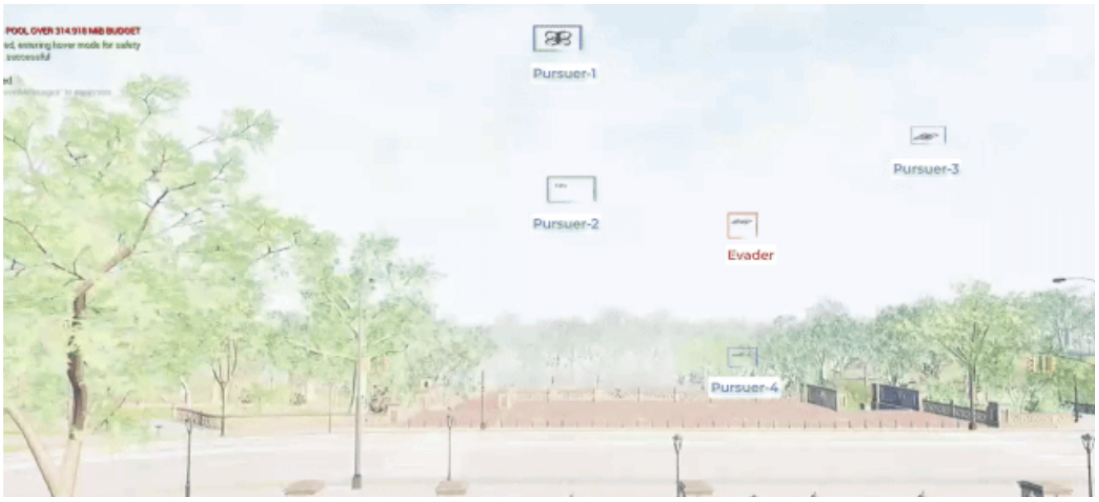

FIGURE 4.1: Each active pursuer moves toward a Voronoi vertex formed by other agents to reduce the evader's free space

4.2 Application: Infrastructure Defense Scenario

This strategy is applied to an infrastructure defense scenario where a team of pursuer drones protects a static asset (a fountain) from an incoming evader drone. The evader simulates a hostile UAV controlled via an Xbox controller, while the pursuers operate autonomously.

- The evader attempts to attack the fountain placed at a known coordinate.
- The pursuers are initialized at random defensive positions and coordinate in real time using the Voronoi strategy.
- The goal is to intercept the evader before it breaches a minimum distance to the asset.

4.3 Simulation Results and Demonstration

The simulation was conducted in the City Park Environment using Cosys-Airsim and Unreal Engine 5.4. The evader is manually controlled and attempts various entry paths

FIGURE 4.2: Infrastructure defense scenario: pursuers use Voronoi-based coordination to intercept the manually operated evader before it reaches the target (fountain)

toward the target. The pursuers adapt their behavior and often succeed in interception, validating the effectiveness of the Voronoi-based strategy in 3D.

The full set of simulation videos is available at the following link:

https://drive.google.com/drive/folders/1vgD6-SRPwKMvw_B8PcCDiElMA-xTkrXO?usp=drive_link

References

References

1. Microsoft AirSim Documentation: https://github.com/microsoft/AirSim

2. Unreal Engine Documentation: https://docs.unrealengine.com

3. Scaramuzza, D., Fraundorfer, F. (2011). *Visual Odometry [Tutorial].* IEEE Robotics Automation Magazine, 18(4), 80–92

4. Engel, J., Schöps, T., Cremers, D. (2014). *LSD-SLAM: Large-scale direct monocular SLAM.* European Conference on Computer Vision (ECCV), 834–849

Acknowledgements

I would like to express my deepest gratitude to my supervisor, **Professor Debraj Chakraborty**, for his invaluable guidance, consistent support, and insightful discussions throughout the course of this project. His mentorship played a crucial role in shaping the direction and depth of my work.

I am especially thankful to **Adityaya Dhande** and **Deep Boliya** for their help in developing the theoretical framework of the 3D Voronoi-based pursuit strategy, and for their inputs during implementation phases when I encountered challenges in AirSim. Their collaborative spirit and shared understanding greatly enriched the quality of the work.

I also wish to thank **Anjaneya Damle** for his contributions toward the formulation and development of the Adaptive Camera-Guided Control (ACGC) system. His ideas were instrumental in shaping the core control logic of the real-time visual tracking pipeline.

A special thanks to **Sanskriti Sharma** for her assistance with the visual illustrations and presentation slides. Her help significantly improved the clarity and communication of the results.

Finally, I would like to thank my friends and family for their constant encouragement, and the developers of *Unreal Engine* and *Microsoft AirSim*, whose tools provided the foundation for this simulation-based research.

Siddharth Anand