Autonomous Aerial Engagement Systems: Technical Proof of Concept and Architectural Analysis

1. Strategic Context and Executive Overview

1.1 The Imperative for Autonomous Maneuver

The operational landscape of modern aerial warfare is undergoing a fundamental paradigm shift, transitioning from remotely piloted systems to fully autonomous collaborative combat aircraft (CCA). This evolution is driven by the increasing contestability of the electromagnetic spectrum, where adversary electronic warfare (EW) capabilities threaten to sever the critical datalinks required for human-in-the-loop (HITL) teleoperation.[1] In such denied environments, the ability of an unmanned system to independently execute the Observe-Orient-Decide-Act (OODA) loop - specifically the terminal engagement phase known as "dogfighting" - is not merely a technical novelty but a strategic necessity.

Recent initiatives, such as DARPA's Air Combat Evolution (ACE) program and the AlphaDogfight Trials, have demonstrated that artificial intelligence (AI) agents can outperform human pilots in simulated within-visual-range (WVR) combat.[3] These developments underscore a critical capability gap: while high-end AI is being tested on modified F-16 platforms like the X-62A VISTA, there is an immediate requirement for scalable, low-cost autonomous interceptors capable of neutralizing hostile drones.[3] This report presents a Proof of Concept (PoC) for such a system, utilizing Commercial Off-The-Shelf (COTS) hardware and deterministic algorithmic logic to demonstrate "minimum viable autonomy" for aerial engagement.

1.2 Objectives of the Proof of Concept

The primary objective of this technical report is to provide a Squadron Leader and associated engineering staff with a comprehensive architectural blueprint for an autonomous "Hunter-Killer" drone. The proposed system is designed to detect, track, and intercept a maneuvering aerial target without human intervention. The core philosophy of this design is "Bounded Autonomy," utilizing a deterministic Finite State Machine (FSM) rather than opaque "black box" neural networks for decision-making.[1] This approach ensures that the system's behavior is predictable, explainable, and certifiable under current military Rules of Engagement (ROE).[6]

The system architecture decouples the high-level mission computing (the "Brain") from the low-level flight stabilization (the "Reflexes"). By integrating a companion computer, such as the NVIDIA Jetson series, with a robust flight controller like the Pixhawk, we create a platform capable of processing computer vision (CV) algorithms at the edge while maintaining flight safety through established aviation-grade firmware.[7]

This report details the algorithmic flow, mathematical guidance laws, hardware integration, and software protocols required to realize this capability.

1.3 Operational Relevance and Use Case

The proposed logic is tailored for a specific defensive counter-air (DCA) scenario: "Autonomous Island Defense" or base protection.[1] In this scenario, a defensive drone is launched to intercept an incoming bandit that has breached the outer defensive layer. The operational flow dictates that the drone must navigate to the threat sector, visually acquire the target using onboard sensors, and execute a kinetic or electronic attack. The relevance of this PoC extends beyond simple 1v1 engagements; it serves as the foundational "atomic unit" for future swarming behaviors, where multiple autonomous agents coordinate to suppress enemy air defenses.[10]

2. Operational Concept: The Autonomous OODA Loop

2.1 Automating the Engagement Cycle

The framework for the proposed algorithm is the OODA loop, a concept originally developed by Colonel John Boyd. In a manned aircraft, the pilot observes the enemy, orients themselves spatially, decides on a maneuver, and acts on the controls. In an autonomous system, these cognitive processes are mapped to distinct software modules running in real-time. The speed at which this loop cycles - the system's frequency - determines its lethality. For this PoC, the target loop rate is 20-30Hz, sufficient to counter the dynamics of Group 1 and Group 2 UAS.[12]

The transition from human control to machine autonomy requires the digitization of intuition. A human pilot instinctively knows to "lead" a target - to aim where the enemy *will be*, not where they are. The autonomous system achieves this through mathematical Guidance Laws, specifically Proportional Navigation (PN), which converts visual error measurements into acceleration commands.[14] The "Observe" phase utilizes computer vision to extract target state data; "Orient" involves coordinate frame transformations to map 2D pixel data to 3D world space; "Decide" selects the appropriate state from the FSM; and "Act" transmits MAVLink velocity vectors to the flight controller.[16]

2.2 Phase 1: Acquisition and Search

Before an engagement can begin, the system must locate the threat. Unlike a human pilot who can scan a vast horizon, a drone is limited by the Field of View (FOV) of its camera. The "Search" state incorporates a systematic scan pattern - typically a continuous yaw rotation or a "lawnmower" flight path - to cover the expected threat volume.[1] Upon detection, defined by the CV algorithm returning a confidence score above a set threshold (e.g., >0.7), the system transitions to the "Track" phase.

This mimics the radar lock mechanism in modern fighters, where the sensor creates a track file for the fire control computer.

2.3 Phase 2: Terminal Guidance and Intercept

Once locked, the drone abandons its search trajectory and enters the intercept geometry. This is the most critical phase, where the algorithm must solve the maneuvering target problem. The system does not merely chase the target's tail (Pure Pursuit), as this would result in a stern chase that makes interception impossible against a faster or equally fast opponent.[17] Instead, it calculates a collision triangle, creating a lead angle that places the drone on a collision course. This phase continues until the range to target drops below the "Kill Radius" (e.g., 5 meters), triggering the terminal effect.[1]

3. Sensing and Perception Architecture

3.1 Visual Target Acquisition

The "eyes" of the system are provided by a monocular camera setup. In the context of a rudimentary PoC, relying on a single camera reduces weight and complexity compared to LiDAR or stereo-vision systems, though it introduces challenges in depth estimation.[18] The primary sensor data consists of RGB frames processed to identify the target's bounding box within the image plane.

The detection pipeline utilizes established object detection networks such as YOLO (You Only Look Once) or MobileNet SSD, which are optimized for edge computing devices like the Jetson Nano.[12] These networks are trained on datasets of aerial targets to recognize drones against complex backgrounds (sky, clouds, or ground clutter). The output of this module is the target's centroid coordinates (u, v) in pixels and the bounding box dimensions (w, h).

3.1.1 Bounding Box to State Estimation

While pixel coordinates provide the target's bearing, they do not directly provide range. For a monocular system, range is estimated using the "Pinhole Camera Model" and the known physical size of the target. If the enemy drone type is known (e.g., a DJI Phantom with a width of 350mm), the range Z can be approximated by:

$$Z = \frac{f \cdot W_{real}}{W_{pixel}}$$

Where f is the camera focal length in pixels, W_{real} is the physical width of the target, and W_{pixel} is the width of the bounding box.[12]

While this method suffers from noise, it provides sufficient accuracy for the guidance law to function, as Proportional Navigation relies primarily on the rate of change of the Line of Sight (LOS) angle rather than absolute range.[15]

3.2 Coordinate Frame Transformations

A significant engineering challenge in autonomous flight is the management of coordinate frames. The camera "sees" in the 2D Image Frame, but the drone flies in the 3D Body or Inertial Frame.[20]

Frame	Notation	Description	Orientation
Inertial Frame	F_I	Fixed Earth Frame (NED)	North (x), East (y), Down (z)
Body Frame	F_B	Attached to Drone CoG	Nose (x), Right Wing (y), Down (z)
Camera Frame	F_C	Attached to Optical Center	Right (x), Down (y), Forward (z)
Image Frame	F_P	2D Pixel Plane	Right (u), Down (v)

The transformation from the visual error in the image frame to a velocity command in the body frame requires a rotation matrix. If the camera is mounted rigidly to the body, this is a constant static transform. However, the system must account for the fact that a "right" command in the image frame corresponds to a "Yaw Right" command for the drone. Furthermore, because quadcopters must roll to move laterally, the coupling between the camera's roll and the image horizon must be compensated for, or the drone may enter an unstable oscillation.[21]

3.3 State Estimation and Filtering

Raw detection data from computer vision is inherently noisy. A target bounding box may jitter by several pixels frame-to-frame due to sensor noise or lighting changes. Feeding this raw noise directly into the flight controller would result in erratic motor behavior and rapid battery depletion. To mitigate this, a Kalman Filter or an Alpha-Beta filter is employed.[23]

The filter predicts the target's next position based on its current velocity and corrects that prediction with the new measurement. This provides a smooth "Track State" that represents the best estimate of the enemy's position and velocity. In the event of a momentary visual loss (e.g., the target passes behind a cloud or sun glare), the filter allows the drone to "coast" on the predicted trajectory for a short duration (1-3 seconds) before reverting to search mode.[24] This persistence is vital for maintaining the lock during high-dynamic maneuvering.

4. Guidance, Navigation, and Control (GNC) Logic

4.1 Guidance Law Selection

The Guidance Law is the mathematical algorithm that dictates the drone's path. For air-to-air engagement, two primary families of guidance laws are considered: Pursuit Guidance and Proportional Navigation.

4.1.1 Pure Pursuit (PP)

Pure Pursuit is the simplest form of guidance. The algorithm commands the velocity vector of the pursuer to align directly with the Line of Sight (LOS) vector to the target.[25]

$$\dot{\psi} = K \cdot (\psi_{target} - \psi_{drone})$$

While easy to implement, PP is kinematically inefficient. Against a crossing target, PP results in a curved flight path that settles into a tail chase. If the target is faster than the pursuer, interception is impossible. Consequently, PP is generally rejected for high-performance dogfighting, though it serves as a useful fallback mode if LOS rate data becomes unreliable.

4.1.2 Proportional Navigation (PN)

Proportional Navigation is the industry standard for missile guidance and is the chosen algorithm for this PoC. The fundamental principle of PN is "Constant Bearing, Decreasing Range" (CBDR). If the LOS angle to a target remains constant while the range decreases, a collision is guaranteed.[14]

The acceleration command a_c generated by PN is proportional to the rotation rate of the LOS vector $\dot{\lambda}$ and the closing velocity V_c:

$$a_c = N \cdot V_c \cdot \dot{\lambda}$$

Where N is the navigation constant (gain). For UAV applications, an N value between 3 and 5 is typical.[14]

The intuition is that if the target drifts to the right at 5 degrees per second, the drone should turn to the right at $N \times 5$ degrees per second. This effectively "leads" the target, flying toward the intercept point rather than the current position.

4.2 Implementation of Image-Based Visual Servoing (IBVS)

Visual servoing bridges the gap between the camera image and the guidance law. In an IBVS scheme, the "error" signal is defined directly in the 2D image plane, avoiding the complex and error-prone reconstruction of 3D space.[21]

The control objective is to drive the distance between the target centroid (u, v) and the image center (u_c, v_c) to zero.

- **Yaw Control (V_ψ):** The horizontal error $(u - u_c)$ drives the yaw rate. If the target is on the right of the image, the drone yaws right.
- **Altitude Control (V_z):** The vertical error $(v - v_c)$ drives the vertical velocity. If the target is above center, the drone climbs.
- **Forward Velocity (V_x):** The size of the bounding box (or estimated range) determines forward speed. If the target is far (small box), velocity increases. If close, velocity decreases to prevent overshoot, or increases to ensure kinetic impact, depending on the phase.[22]

The interaction matrix (Jacobian) L_s relates feature velocity to camera velocity:

$$\dot{s} = L_s \cdot V_c$$

For a simplified quadcopter model in a dogfight, we can approximate and decouple the axes, applying independent PID controllers to each axis. This "Look-and-Move" architecture is robust and computationally inexpensive.[29]

4.3 PID Controller Tuning

To convert the guidance commands into smooth motor outputs, a Proportional-Integral-Derivative (PID) controller is utilized.

- **Proportional (P):** Reacts to the current error. A high P gain makes the drone snap quickly to the target but may cause oscillation.
- **Integral (I):** Reacts to accumulated past error. This corrects for steady-state errors, such as wind pushing the drone off course.
- **Derivative (D):** Reacts to the rate of change of error. This dampens the motion,

preventing overshoot.[13]

The mathematical representation for the control signal $u(t)$ is:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}$$

In the context of the PoC, tuning these gains is critical. An overly aggressive D-term can amplify the noise from the vision system, causing the motors to jitter. A "Soft" tune is preferred for the outer guidance loop, while the inner flight stabilization loop (handled by the Pixhawk) utilizes a "Hard" tune for stability.[31]

5. Algorithmic Architecture: Finite State Machine (FSM)

5.1 The Case for Deterministic Logic

While modern research explores end-to-end Deep Reinforcement Learning (DRL) where a neural network maps pixels directly to motor speeds, such "Black Box" systems are currently difficult to validate for safety-critical operations.[1] For a Proof of Concept presented to military leadership, predictability is paramount. A Finite State Machine (FSM) provides a transparent decision-making framework where every behavior can be traced to a specific logic gate.[5]

5.2 State Definitions

The operational logic is encapsulated in five discrete states. The system transitions between these states based on sensor inputs and timer events.[5]

1. **STATE_IDLE (Pre-Flight):**
    - **Behavior:** Motors disarmed. Companion computer performs handshake with Flight Controller. Camera warms up and adjusts exposure.
    - **Transition:** Manual "Arm" command or mission start signal -> STATE_TAKEOFF.
2. **STATE_TAKEOFF:**
    - **Behavior:** Drone arms motors and ascends vertically to a pre-defined tactical altitude (e.g., 20m AGL).
    - **Transition:** Altitude error < 0.5m -> STATE_SEARCH.
3. **STATE_SEARCH (Acquisition):**
    - **Behavior:** The drone executes a search pattern. For this PoC, a constant Yaw Rate (e.g., 20 deg/s) provides a 360-degree scan.
    - **Transition:** Visual Detection (Confidence > Threshold) -> STATE_TRACK.

4. **STATE_TRACK (Pursuit):**
    - **Behavior:** The FSM activates the Guidance Law (PN). Velocity commands are sent to keep the target centered while closing distance. The "safety off" switch is virtually engaged.
    - **Transition:**
        - Target Lost (Timeout > 2s) -> STATE_SEARCH.
        - Range < Intercept_Radius (e.g., 5m) -> STATE_ENGAGE.
        - Battery Low or Geofence Breach -> STATE_RTL (Return to Launch).
5. **STATE_ENGAGE (Terminal):**
    - **Behavior:** The drone executes the terminal maneuver. In a kinetic kill vehicle, this might be maximum acceleration. In a simulated dogfight, this triggers a "Virtual Shot" log entry and a breakaway maneuver to avoid mid-air collision.
    - **Transition:** Maneuver Complete -> STATE_IDLE or STATE_SEARCH.

5.3 FSM Logic Flow Diagram

The following represents the decision logic in a structured format suitable for technical review.

Current State	Triggering Event	Condition	Next State	System Action
IDLE	Operator Command	System Ready	**TAKEOFF**	Arm Motors, Set Target Alt
TAKEOFF	Altimeter	Alt > Target_Alt	**SEARCH**	Initiate Yaw Spin
SEARCH	Computer Vision	Detection > 70%	**TRACK**	Lock Gimbal / Body Yaw
TRACK	Computer Vision	Detection < 20%	**SEARCH**	Resume Yaw Spin (Last Known Dir)
TRACK	Range Estimation	Range < 5m	**ENGAGE**	Max Throttle / Virtual Fire

TRACK	Geofence	Dist > Max_Radius	**RTL**	Return to Home
ENGAGE	Timer	Time > 3s	**RTL**	Return to Home

This deterministic structure ensures that the drone never enters an undefined state. For example, the "Target Lost" transition prevents the drone from flying off into the distance if the enemy maneuvers out of the FOV; it immediately reverts to a search behavior to reacquire.[34]

6. Hardware and Systems Integration

6.1 The "Companion Computer" Architecture

A critical design choice for this system is the bifurcation of processing duties. The "Reflexes" (stabilization) are handled by a microcontroller, while the "Brain" (vision/logic) is handled by a microprocessor. This separation ensures that a software crash in the high-level Python script does not cause the drone to fall out of the sky.[36]

6.1.1 Compute Module: NVIDIA Jetson Nano / Orin

The NVIDIA Jetson platform is selected for its integrated GPU, which is essential for running the YOLO object detection network at high frame rates (30+ FPS). A standard Raspberry Pi CPU would struggle to maintain the framerate required for high-speed tracking.[8]

- **Role:** Image Processing, FSM Logic, Guidance Calculation, MAVLink message generation.
- **I/O:** GPIO UART (Serial) for communication with the flight controller; CSI/USB for camera input.

6.1.2 Flight Controller: Pixhawk (The Cube)

The Pixhawk is the industry standard for open-source heavy-duty flight control. It runs the ArduPilot firmware, which handles the complex physics of multi-rotor flight, sensor fusion (EKF), and fail-safes.[7]

- **Role:** Motor mixing, attitude stabilization (IMU), global navigation (GPS), battery monitoring.
- **I/O:** TELEM2 port for connection to Companion Computer; PWM outputs for ESCs.

6.2 Interface and Wiring Diagram

The physical link between the Jetson and the Pixhawk is a UART (Universal Asynchronous Receiver-Transmitter) serial connection. The following wiring schema must be strictly adhered to for successful communication.[7]

Wiring Table: Jetson Nano to Pixhawk (TELEM2)

Signal Function	Jetson Nano Pin (GPIO Header)	Pixhawk TELEM2 Pin	Note
Ground	Pin 6 (GND)	Pin 6 (GND)	Common Ground reference essential
Transmit Data	Pin 8 (UART_TX)	Pin 3 (UART_RX)	TX must go to RX
Receive Data	Pin 10 (UART_RX)	Pin 2 (UART_TX)	RX must go to TX
Power	N/A	N/A	**DO NOT** connect 5V lines between units

Engineering Note: The Jetson Nano operates at 3.3V logic levels, which is compatible with modern Pixhawk ports. However, power must be supplied separately. The Pixhawk cannot supply sufficient current to power the Jetson. A dedicated 5V/4A BEC (Battery Elimination Circuit) is required to power the Jetson directly from the LiPo battery.[39]

6.3 Sensor Suite

- **Camera:** A Global Shutter camera (e.g., Raspberry Pi Global Shutter or equivalent) is highly recommended over a Rolling Shutter camera. Rolling shutter artifacts ("jello effect") caused by high-vibration drone flight can severely distort the image, causing the CV algorithm to lose track or calculate incorrect error vectors.
- **Rangefinder:** While monocular depth estimation is used for the rudimentary PoC, the addition of a lightweight LiDAR (e.g., TF-Luna) improves the "Kill" trigger accuracy by providing precise distance measurements in the terminal phase.[1]

7. Software Implementation and Communication Protocols

7.1 The Software Stack

The software architecture is built on open-source standards to ensure modularity and ease of development.

- **Operating System:** Ubuntu Linux (running on Jetson).
- **Middleware:** ROS (Robot Operating System) Noetic or Humble. ROS allows the vision system and control system to run as separate "nodes" that communicate via topics. Alternatively, a monolithic Python script using DroneKit is sufficient for a simple PoC.[40]
- **Protocol:** MAVLink (Micro Air Vehicle Link). This is the binary messaging protocol used to send commands to the ArduPilot firmware.[42]

7.2 MAVLink Command Implementation

To control the drone's movement, the companion computer does not send "Stick Positions" (like a virtual remote control). Instead, it sends high-level **Velocity Vectors**. The specific MAVLink message used is SET_POSITION_TARGET_LOCAL_NED.[16]

This message is powerful because it allows the developer to use a **Type Mask** to ignore position constraints and strictly command velocity and yaw rate. This is essential for the "Visual Servoing" approach, where we don't know the GPS coordinate of the target, but we know we need to move "Forward at 5 m/s" and "Yaw Right at 10 deg/s."

7.2.1 Bitmask Configuration

The type_mask field tells the flight controller which data fields to listen to. For velocity control, we mask out Position (bits 0, 1, 2) and Acceleration (bits 6, 7, 8).

- **Target Mask:** 0b0000111111000111 (Binary)
  - Ignore Position (x, y, z)
  - Ignore Acceleration (ax, ay, az)
  - **Enable Velocity** (vx, vy, vz)
  - **Enable Yaw Rate**.[40]

7.3 Detailed Algorithm Pseudo-Code

The following pseudo-code illustrates the logic loop running on the Jetson. This demonstrates the integration of the CV detection with the MAVLink command generation.

```
# Import necessary libraries
from dronekit import connect, VehicleMode
from pymavlink import mavutil
import time
import cv2

# Connect to the Flight Controller via UART
vehicle = connect('/dev/ttyTHS1', baud=57600, wait_ready=True)

def send_velocity_command(vx, vy, vz, yaw_rate):
    """
    Sends a MAVLink SET_POSITION_TARGET_LOCAL_NED message.
    vx, vy, vz: Velocity in m/s (North, East, Down)
    yaw_rate: Yaw rate in rad/s
    """
    msg = vehicle.message_factory.set_position_target_local_ned_encode(
        0, 0, 0,
        mavutil.mavlink.MAV_FRAME_BODY_NED, # Frame relative to drone body
        0b0000011111000111, # Bitmask: Enable Vel + YawRate
        0, 0, 0, # Ignored Positions
        vx, vy, vz, # Velocity Inputs
        0, 0, 0, # Ignored Accel
        0, yaw_rate) # Yaw Inputs
    vehicle.send_mavlink(msg)
```

```
def autonomous_dogfight_loop():
    # Main Control Loop
    while True:
        # 1. ACQUIRE IMAGE
        frame = camera.capture()

        # 2. RUN DETECTION (YOLO / Color Threshold)
        target = detector.detect(frame)

if target.is_found:
            # 3. CALCULATE ERRORS
            # Error ranges from -1.0 (Left/Top) to 1.0 (Right/Bottom)
            err_x = target.x_offset
            err_y = target.y_offset

            # 4. COMPUTE CONTROL OUTPUTS (PID Logic)
            # P-Controller for Yaw (Turn to target)
            cmd_yaw_rate = K_yaw * err_x

            # P-Controller for Altitude (Climb/Descend to target)
            # Note: Negative Z velocity is UP in NED frame
            cmd_vz = K_alt * err_y

            # Constant Forward Pursuit Velocity
            cmd_vx = 5.0 # Fly forward at 5 m/s

            # 5. SEND COMMAND
            send_velocity_command(cmd_vx, 0, cmd_vz, cmd_yaw_rate)

            print("Tracking: YawRate={}, Vz={}".format(cmd_yaw_rate, cmd_vz))

        else:
            # 6. TARGET LOST -> SEARCH MODE
            # Spin slowly to find target
            send_velocity_command(0, 0, 0, 0.3) # 0.3 rad/s Yaw
            print("Searching...")

        # Maintain Loop Rate (e.g., 20Hz)
        time.sleep(0.05)
```

```
# Start the mission
autonomous_dogfight_loop()
```

This code snippet highlights the simplicity of the interface. The complexity lies in the detector.detect() function (the CV pipeline), while the control logic remains clean and understandable.[40]

8. Simulation, Validation, and Risk Management

8.1 Simulation-In-The-Loop (SITL)

Before any hardware leaves the ground, the entire code stack is validated in a virtual environment. ArduPilot's SITL (Software In The Loop) allows the Jetson code to communicate with a virtual flight controller running the exact same firmware as the physical hardware.

- **Physics Engine:** Gazebo or AirSim is used to render the 3D world and simulate the aerodynamics of the drone.[40]
- **Scenario Testing:** We create a "Red Team" drone in the simulator that flies a fixed figure-8 pattern. The "Blue Team" (our autonomous code) is launched to intercept.
- **Validation Metrics:**
  - **Intercept Success Rate:** Percentage of runs where minimum distance < 5m.
  - **Control Stability:** Measuring oscillation in the flight path (to tune PID gains).
  - **Latency Analysis:** Measuring the time delta between "Pixel Change" in the simulator and "Motor Output" change.[26]

8.2 Operational Risks and Mitigations

Deploying autonomous systems carries inherent risks, particularly regarding kinetic safety and loss of control.

- **Geofencing:** A "Hard Fence" is programmed into the Pixhawk. If the drone breaches a GPS radius (e.g., 200m) or a maximum altitude (e.g., 50m), the firmware overrides the Jetson and executes an immediate Return-to-Land (RTL) or Land Now failsafe. This "Hard Deck" cannot be overridden by the Python script.[30]
- **Communication Timeout:** A "Heartbeat" failsafe is implemented. If the Pixhawk stops receiving velocity commands from the Jetson for more than 1 second (indicating a code crash or cable failure), it automatically switches to LOITER mode (hover in place).[46]
- **Visual False Positives:** To prevent the drone from attacking a bird or a friendly aircraft, the detection threshold is set high (Confidence > 0.7). Additionally, a "Persistence Filter" requires the target to be detected for 3 consecutive frames before entering TRACK mode, reducing reaction to transient visual noise.

9. Future Horizons: Reinforcement Learning and Swarms

9.1 Beyond FSM: Deep Reinforcement Learning (DRL)

While the FSM approach is sufficient for a PoC, it is limited by the foresight of the programmer. It cannot "invent" tactics. The next generation of this system will utilize Deep Reinforcement Learning (DRL), specifically Proximal Policy Optimization (PPO).47
In a DRL framework, the agent is not told how to fly; it is given a reward function (e.g., "+100 for Intercept", "-10 for Overshoot"). Through millions of simulated iterations, the neural network learns optimal control policies that may include complex maneuvers like "High Yo-Yos" or "Scissors" that are difficult to code explicitly.25 The "Sim-to-Real" gap remains the primary challenge, but transfer learning techniques are rapidly closing this divide.3

9.2 Collaborative Swarms

The architecture presented here serves as the individual node in a larger network. Future iterations will implement "Collaborative Combat," where multiple drones share their target state via a mesh network. This enables tactics such as the "Pincer Maneuver," where one drone flushes a target into the path of another, or "Sensor-Shooter" separation, where a high-flying drone provides coordinates to low-flying interceptors.[2]

10. Conclusion

This report has outlined a comprehensive technical path to demonstrating an Autonomous Aerial Engagement System. By leveraging the **Proportional Navigation** guidance law, **Image-Based Visual Servoing**, and a decoupled **Companion Computer architecture**, the system achieves a robust balance between performance and safety.

The proposed "Automatic Dogfight" capability is not a distant sci-fi concept but a tangible engineering reality achievable with current COTS technology. The integration of the NVIDIA Jetson for edge-AI processing with the reliability of the Pixhawk flight controller creates a platform that is both smart enough to hunt and stable enough to fly safely. This Proof of Concept serves as the critical first step - the "Crawl" phase - in the development of trusted autonomous wingmen that will define the future of air dominance.

The recommendation to the Squadron Leader is to authorize the immediate development of the hardware testbed described herein to move from theoretical simulation to physical validation, securing a tactical advantage in the evolving domain of autonomous warfare.

11. About the Author

Dhruva Shaw is a Robotics and Automation Engineer with experience in embedded systems, cybersecurity, OSINT, and SIGINT-related applications. His work lies at the intersection of automation, signal processing, and intelligent systems.

He was awarded the IEEE Signal Processing Society (SPS) Scholarship in 2024 in recognition of his contributions to the signal processing domain.

https://dhruvashaw.in

me@dhruvashaw.in

Works cited

1. Autonomous Island Defense (AID) algorithm for autonomous attack drones - Reddit, accessed January 6, 2026, https://www.reddit.com/r/CredibleDefense/comments/1itdr24/autonomous_island_defense_aid_algorithm_for/
2. Autonomous Drones Will Not Replace Fighter Pilots, They Will Be Their Wingmen - Belfer Center, accessed January 6, 2026, https://www.belfercenter.org/sites/default/files/2025-06/AutonomousDrones%2C%20Moscioni%2C%20DETS.pdf
3. In a 'world first,' DARPA project demonstrates AI dogfighting in real jet - Breaking Defense, accessed January 6, 2026, https://breakingdefense.com/2024/04/in-a-world-first-darpa-project-demonstrates-ai-dogfighting-in-real-jet/
4. Pentagon takes AI dogfighting to next level in real-world flight tests against human F-16 pilot, accessed January 6, 2026, https://defensescoop.com/2024/04/17/darpa-ace-ai-dogfighting-flight-tests-f16/
5. A Parallel Hierarchical Finite State Machine Approach to UAV Control for Search and Rescue Tasks - SciTePress, accessed January 6, 2026, https://www.scitepress.org/papers/2014/51211/51211.pdf
6. The Autonomous Arsenal in Defense of Taiwan: Technology, Law, and Policy of the Replicator Initiative | The Belfer Center for Science and International Affairs, accessed January 6, 2026, https://www.belfercenter.org/replicator-autonomous-weapons-taiwan
7. Pixhawk 4 Wiring Quick Start | PX4 Guide (main), accessed January 6, 2026, https://docs.px4.io/main/en/assembly/quick_start_pixhawk4

8. Px4 Jetson Nano Connection - NVIDIA Developer Forums, accessed January 6, 2026, https://forums.developer.nvidia.com/t/px4-jetson-nano-connection/271119
9. Target-Defense Games with One or Two Unmanned Surface Vehicles Defending an Island: A Geometric Analytical Approach - MDPI, accessed January 6, 2026, https://www.mdpi.com/2077-1312/13/2/365
10. Autonomous Drone Combat: A Mutli-Agent Reinforcement Learning Approach - ORBi, accessed January 6, 2026, https://orbi.uliege.be/bitstream/2268/335358/1/_article_quad_combat.pdf
11. Autonomous Maneuver Decision Making of Dual-UAV Cooperative Air Combat Based on Deep Reinforcement Learning - MDPI, accessed January 6, 2026, https://www.mdpi.com/2079-9292/11/3/467
12. Visual Servoing of a Moving Target by an Unmanned Aerial Vehicle - PMC - NIH, accessed January 6, 2026, https://pmc.ncbi.nlm.nih.gov/articles/PMC8434602/
13. QuadSim: A Quadcopter Rotational Dynamics Simulation For Reinforcement Learning Algorithms - GitHub, accessed January 6, 2026, https://github.com/BurakDmb/quadsim
14. Terminal attack trajectories of peregrine falcons are described by the proportional navigation guidance law of missiles | PNAS, accessed January 6, 2026, https://www.pnas.org/doi/10.1073/pnas.1714532114
15. Proportional navigation - Wikipedia, accessed January 6, 2026, https://en.wikipedia.org/wiki/Proportional_navigation
16. Copter Commands in Guided Mode — Dev documentation - ArduPilot, accessed January 6, 2026, https://ardupilot.org/dev/docs/copter-commands-in-guided-mode.html
17. Lock, Pursue, Destroy: Understanding Proportional Navigation for Kamikaze Drones | by ROHAN MORE | Medium, accessed January 6, 2026, https://medium.com/@rohanmore90/lock-pursue-destroy-understanding-proportional-navigation-for-kamikaze-drones-90c6407c88cc
18. Toward Increased Airspace Safety: Quadrotor Guidance for Targeting Aerial Objects - Carnegie Mellon University Robotics Institute, accessed January 6, 2026, https://www.ri.cmu.edu/app/uploads/2020/08/MSR_Thesis_AnishBhattacharya.pdf
19. The flowchart of nearby drone detection and tracking - ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/The-flowchart-of-nearby-drone-detection-and-tracking_fig2_324773702
20. Visual Servoing for Aerial Vegetation Sampling Systems - MDPI, accessed January 6, 2026, https://www.mdpi.com/2504-446X/8/11/605
21. Online Predictive Visual Servo Control for Constrained Target Tracking of Fixed-Wing Unmanned Aerial Vehicles - MDPI, accessed January 6, 2026, https://www.mdpi.com/2504-446X/8/4/136
22. Quadrotor UAV Dynamic Visual Servoing Based on Differential Flatness Theory - MDPI, accessed January 6, 2026, https://www.mdpi.com/2076-3417/13/12/7005
23. Design of an Image-Based Visual Servoing System for Autonomous Quadcopter Interception, accessed January 6, 2026, https://repository.tudelft.nl/file/File_bfe89460-33ee-4842-9f04-181bf6accde1
24. Flowchart of the target tracking system | Download Scientific Diagram -

ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/Flowchart-of-the-target-tracking-system_fig3_318168103
25. Pursuit-Evasion - Google Sites, accessed January 6, 2026, https://sites.google.com/view/pursuit-evasion-rl
26. The flowchart of the proposed interception strategy. | Download Scientific Diagram - ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/The-flowchart-of-the-proposed-interception-strategy_fig8_342650159
27. Application of Image-Based Visual Servoing on Autonomous Drones - IEEE Xplore, accessed January 6, 2026, https://ieeexplore.ieee.org/document/9248119/
28. Translational Visual Servoing Control of Quadrotor Helicopters - UBC Library Open Collections, accessed January 6, 2026, https://open.library.ubc.ca/media/stream/pdf/24/1.0072149/1
29. Target tracking using visual servoing: the Parrot AR Drone 2.0 case study - Universidade de Lisboa, accessed January 6, 2026, https://fenix.tecnico.ulisboa.pt/downloadFile/1126295043837959/Resumo_Alargado_Alexandra_Pereira_81072.pdf
30. How a Drone Hovers at a Position Hold (GPS) — Explained with Python Simulation! | by ROHAN MORE | Medium, accessed January 6, 2026, https://medium.com/@rohanmore90/how-a-drone-hovers-at-a-position-hold-gps-explained-with-python-simulation-2a82424eed7a
31. UAV Power Line Tracking Control Based on a Type-2 Fuzzy-PID Approach - MDPI, accessed January 6, 2026, https://www.mdpi.com/2218-6581/12/2/60
32. Python PID Controller Example: A Complete Guide | by UATeam | Medium, accessed January 6, 2026, https://medium.com/@aleksej.gudkov/python-pid-controller-example-a-complete-guide-5f35589eec86
33. Finite state diagram that determines the behavior of the UAV. - ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/Finite-state-diagram-that-determines-the-behavior-of-the-UAV_fig2_334134829
34. Flow chart of our finite state machine (the state of the vehicle is... - ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/Flow-chart-of-our-finite-state-machine-the-state-of-the-vehicle-is-marked-with-blue_fig7_349369607
35. Finite State Machines, Flow Charts, and State Tables - Tutorials - RobotShop Community, accessed January 6, 2026, https://community.robotshop.com/forum/t/finite-state-machines-flow-charts-and-state-tables/13169
36. How To Connect PixHawk to Raspberry Pi and NVIDIA Jetson - YouTube, accessed January 6, 2026, https://www.youtube.com/watch?v=nIuoCYauW3s
37. Copter not responding to set_position_target_local_ned mavlink commands #1205 - GitHub, accessed January 6, 2026, https://github.com/dronekit/dronekit-python/issues/1205
38. Holybro Pixhawk Jetson Baseboard - PX4-user_guide - GitHub, accessed January 6, 2026, https://github.com/PX4/PX4-user_guide/blob/main/tr/companion_computer/holybro_pixhawk_jetson_baseboard.md

39. Connecting PixHawk to Raspberry Pi and NVIDIA Jetson - Hackster.io, accessed January 6, 2026, https://www.hackster.io/Matchstic/connecting-pixhawk-to-raspberry-pi-and-nvidia-jetson-b263a7
40. Example: Guided Mode Movement and Commands (Copter) - DroneKit Python, accessed January 6, 2026, https://dronekit-python.readthedocs.io/en/latest/examples/guided-set-speed-yaw-demo.html
41. Guiding and Controlling Copter - DroneKit-Python's documentation!, accessed January 6, 2026, https://dronekit.netlify.app/guide/copter/guided_mode
42. MAVLINK Common Message Set (common.xml), accessed January 6, 2026, https://mavlink.io/en/messages/common.html
43. How to do «go to left/right/forward/backward» with python dronekit? - Stack Overflow, accessed January 6, 2026, https://stackoverflow.com/questions/39695181/how-to-do-go-to-left-right-forward-backward-with-python-dronekit
44. MPC-based Visual Servo Control for UAVs - kth .diva, accessed January 6, 2026, https://kth.diva-portal.org/smash/get/diva2:1479331/FULLTEXT01.pdf
45. The flowchart of our architecture. | Download Scientific Diagram - ResearchGate, accessed January 6, 2026, https://www.researchgate.net/figure/The-flowchart-of-our-architecture_fig3_342444320
46. Guided Mode — Copter documentation - ArduPilot, accessed January 6, 2026, https://ardupilot.org/copter/docs/ac2_guidedmode.html
47. Autonomous Dogfight Decision-Making for Air Combat Based on Reinforcement Learning with Automatic Opponent Sampling - MDPI, accessed January 6, 2026, https://www.mdpi.com/2226-4310/12/3/265
48. A DRL Framework for Autonomous Pursuit-Evasion: From Multi-Spacecraft to Multi-Drone Scenarios - MDPI, accessed January 6, 2026, https://www.mdpi.com/2504-446X/9/9/636
49. Inroads into Autonomous Network Defence using Explained Reinforcement Learning - CEUR-WS.org, accessed January 6, 2026, https://ceur-ws.org/Vol-3391/paper1.pdf
50. Learning Multi-Pursuit Evasion for Safe Targeted Navigation of Drones - arXiv, accessed January 6, 2026, https://arxiv.org/html/2304.03443v2