Vision-Guided Precision Landing of a Parrot Mambo Minidrone on a Mobile Line-Follower Platform Using Simulink

Anushka Satav G[1]
[1]Masters Student in Robotics and Autonomous Systems (AI), Arizona State University, Tempe, AZ, USA

Keywords:

Parrot Mambo
MATLAB
Simulink
HSV Color Segmentation
Blob Analysis
Color Detection
Real-time Image Processing
Autonomous UAV Navigation
Computer Vision
RGB Line Detection
Flight Control System
Drone Soft Landing

ABSTRACT

Unmanned Aerial Vehicles (UAVs) such as the Parrot Mambo Minidrone have gained traction as versatile platforms for real-time vision-guided navigation and landing. This project focuses on implementing an autonomous landing system where the drone identifies and lands on a **mobile ground vehicle**—a line-follower robot—using live video feed processed in MATLAB Simulink. The system employs HSV color segmentation and blob analysis to detect a colored landing platform in real-time. The drone computes positional error between the platform centroid and the camera frame center to guide motion corrections. A custom Stateflow logic governs flight behavior across hover, cruise, and descent phases. During the cruise state, the drone continuously adjusts its horizontal position while gradually lowering altitude as it aligns with the moving platform. Once the alignment and proximity conditions are satisfied, a controlled vertical descent is executed. The system is fully deployed onboard using Simulink's code generation tools, enabling autonomous operation without external supervision. This work demonstrates the feasibility of integrating real-time image processing with onboard path planning to achieve precision landings on moving ground targets.

Corresponding Author:

Anushka Gangadhar Satav
Arizona State University, Tempe, Arizona, USA
Email: anushka.satav@asu.edu

1. INTRODUCTION

In recent years, drones have emerged as powerful tools in robotics due to their versatility in aerial navigation, real-time sensing, and automation capabilities. Among them, the Parrot Mambo Minidrone offers an accessible and programmable platform for developing and testing autonomous flight systems. With its integrated downward-facing camera and compatibility with MATLAB and Simulink, the drone is particularly well-suited for computer vision and control applications.

This project builds upon foundational drone control concepts and extends them toward a more complex task: enabling the Parrot Mambo to autonomously detect and land on a **moving line-follower robot**. Unlike traditional static platform landing, this task introduces additional challenges in timing, visual tracking, and trajectory synchronization. The system must not only identify a designated landing platform via color segmentation, but also track and align with it while both the drone and the platform are in motion.

Using MATLAB and Simulink, the onboard video feed is processed in real time to segment a uniquely colored platform using HSV thresholding and blob analysis. The drone calculates the offset (centroid error) between the platform's position and the center of its visual frame, which is then used to generate flight corrections. A Stateflow-based control architecture governs the drone's flight behavior across discrete states—hovering, cruising, and landing—based on visual cues and alignment thresholds. During cruise, the drone actively follows the platform while gradually decreasing altitude to prepare for descent. This project demonstrates the potential of combining image processing with dynamic path planning for real-time, onboard decision-making in autonomous UAVs. The successful integration of perception and control within a Simulink-based environment also serves as a foundation for more advanced aerial-ground coordination tasks in multi-agent systems and mobile robotics.

2. METHOD

This section details the methodology used to implement an autonomous platform-following and precision landing system for the Parrot Mambo Minidrone using MATLAB and Simulink. The objective was to enable the drone to autonomously detect a colored landing platform mounted on a **moving line-follower robot**, track it in real time using onboard vision, and execute a controlled descent for soft landing.

The solution integrated Simulink-based image processing, blob detection, path planning, and state-based control logic into a unified model. A real-time video stream from the drone's downward-facing camera was processed using HSV color segmentation and blob analysis to detect the platform and determine its position relative to the drone's field of view. The system computed horizontal position errors (x and y) from the centroid of the detected blob, which served as feedback signals for controlling lateral movement.The flight control architecture was managed using a single Stateflow chart, which handled the drone's behavioral transitions across all phases of operation: take-off, cruising toward the moving platform, and vertical descent. During the cruise phase, the drone adjusted its horizontal position in proportion to the visual error while simultaneously reducing altitude as alignment improved. When both position error and altitude thresholds were satisfied, a descent flag was raised within the Stateflow logic, initiating a smooth, autonomous landing.

The development process followed a model-based design workflow. After validating the logic through simulation, the model was deployed to the Parrot Mambo drone using the Simulink hardware support package. The drone operated in a controlled indoor environment where the line-follower robot moved along a black path, and a colored platform served as the landing zone. Real-world flight tests confirmed the system's ability to autonomously follow the moving platform and execute soft landings based entirely on onboard visual input.

2.1 Drone Setup and Configuration

The experiment began with the setup and configuration of the Parrot Mambo Minidrone to ensure stable flight and reliable communication with MATLAB and Simulink. The process started by connecting the Parrot Mambo Minidrone to MATLAB through the Parrot Minidrone Support Package, establishing a seamless interface for control and data exchange. Initial flight control tests were then conducted in a controlled indoor environment to verify the drone's basic functionality, including take-off, hovering, and landing. These tests confirmed that the drone operated as expected under manual commands before advancing to autonomous operations. Next, the Simulink template for the Parrot Minidrone was loaded, providing a foundational framework for integrating image processing and flight control algorithms. To prepare for vision-based navigation, the drone's downward-facing camera was calibrated to optimize image capture conditions, adjusting for factors such as lighting variations to ensure consistent performance during operation.

Figure 1. Parrot Mambo Minidrone used in this project

2.2 Image Processing for Platform Detection

The development of robust color-based navigation relied on real-time image processing of frames captured by the Parrot Mambo drone's downward-facing camera. MATLAB Simulink-Image Processing System block was used to extract the position and orientation of the target R/G/B path. A customized image processing pipeline was developed using the MATLAB Color Thresholder App and integrated into Simulink as a functional subsystem (see Figure. 2).

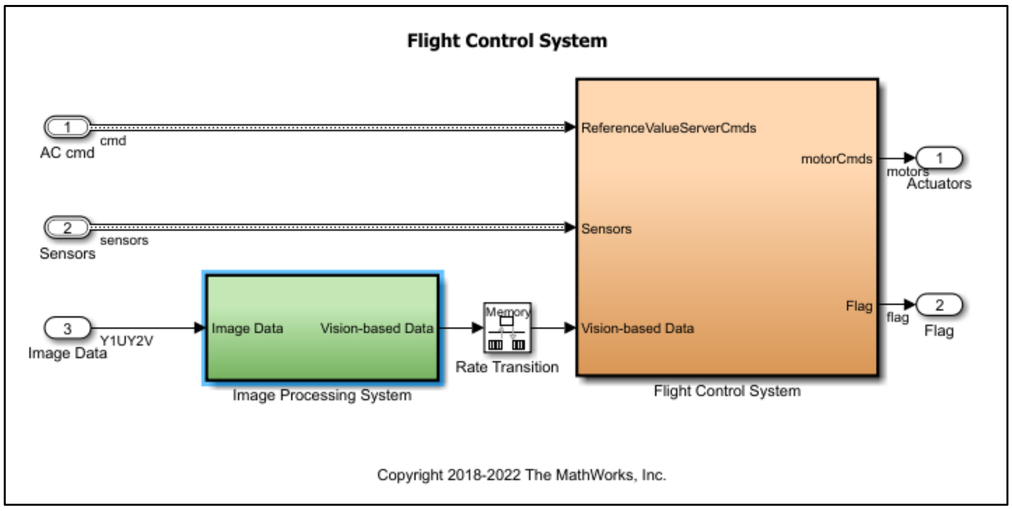


Figure 2. Simulink Top-Level Model and Image Processing System Block

To enable the drone to identify colored blocks (Red, Green, Blue, and Yellow), an image processing pipeline was developed in MATLAB. Real-time video frames from the drone's camera were captured and processed using the following steps:

- **Color Segmentation**: The MATLAB Color Thresholder App was used to define threshold ranges in the **HSV color space** for the colored platform (red or green). HSV space was selected over RGB due to its robustness under variable lighting conditions. Threshold values were manually tuned to ensure consistent detection of the platform in an indoor environment. Once optimized, the thresholding settings were exported as a MATLAB function and embedded into Simulink using a Function block for real-time detection during flight. *(see Figure 2 and Figure 3)*

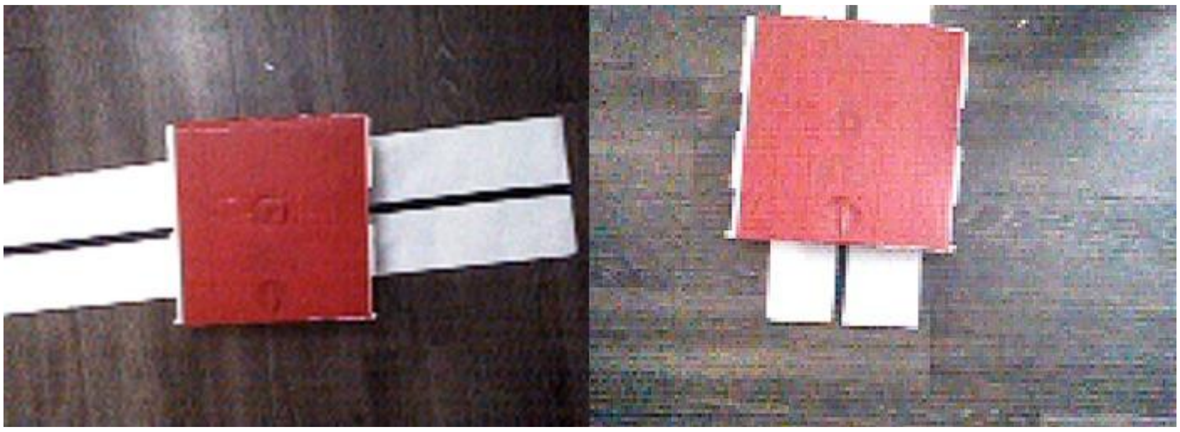


Figure 3. Test image for creating colour detecting functions

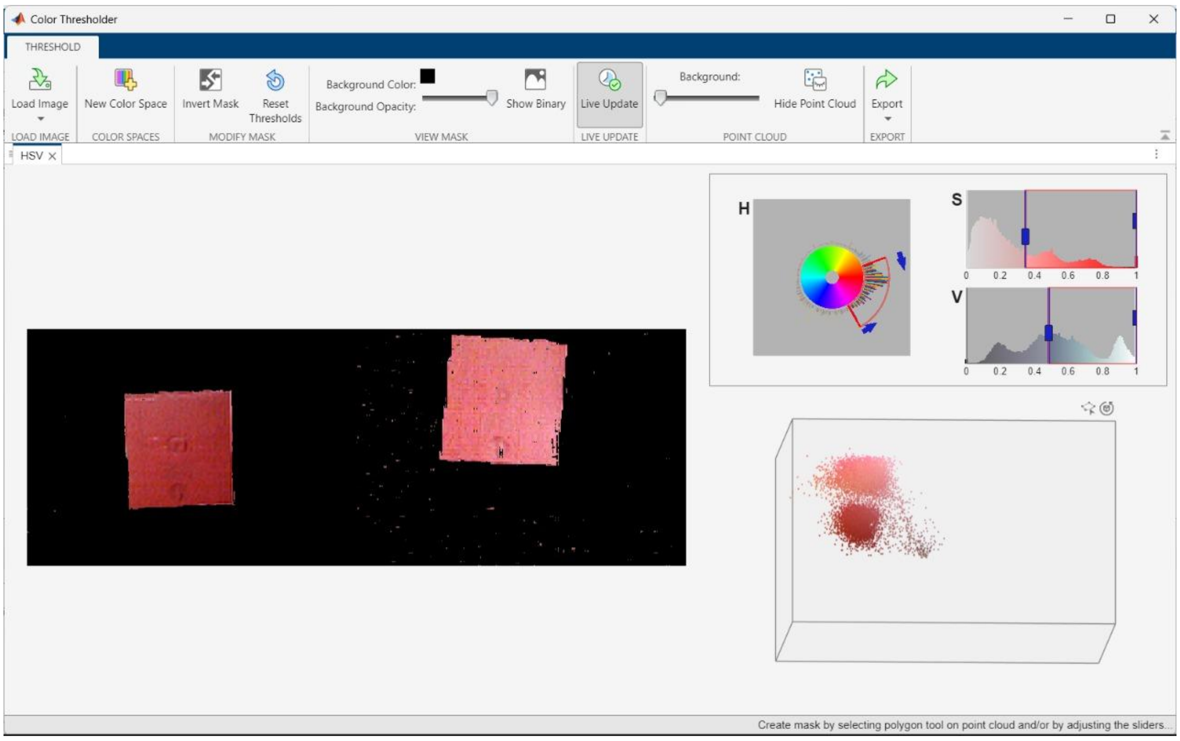


Figure 4. Color Thresholder App for Red Platform Detection

- **Mask Generation and Filtering**: The real-time image stream from the drone's camera was processed frame-by-frame to generate a **binary mask**, where pixels matching the HSV threshold appeared white and all other pixels were black. This binary output was passed through a **Median Filter** block to suppress noise such as isolated white or black pixels, improving the clarity of the segmented region. The filtered binary image preserved the shape of the platform while removing spurious detections. *(see Figure 5).*

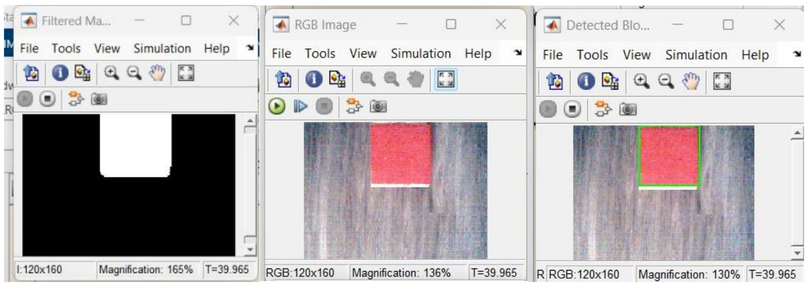


Figure 5. Original RGB Image, Median-Filtered Binary Mask and Blob detection

- **Blob Analysis and Centroid Extraction:** The filtered binary mask was fed into a **Blob Analysis** block in Simulink to extract key shape features. The block computed the **centroid (x, y)** of the largest detected blob—representing the visual center of the colored platform—as well as its bounding box and area. These centroid coordinates were then compared to the center of the image frame to compute x_error and y_error, representing the horizontal and vertical deviation of the drone relative to the platform. *(refer to outputs in Figure 5)*
- **Simulink Integration**: The complete image processing workflow—including HSV color segmentation, mask generation, noise filtering, blob analysis, and centroid tracking—was encapsulated into a custom **Image Processing System** block in Simulink. This modular design ensured that each step in the visual detection pipeline was clearly organized and easily modifiable for parameter tuning.
  The input to the block was the real-time video stream from the drone's downward-facing camera, while the outputs included:

- x_error and y_error: Positional errors representing the offset between the platform's centroid and the image center.

All logic blocks—such as the HSV thresholding function, Median Filter, Blob Analysis, and the landing condition logic—were connected within a structured Simulink subsystem.

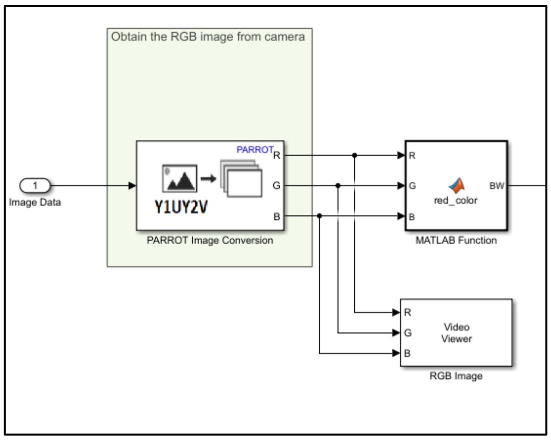


Figure 6. Color Detection Logic Blocks in Simulink

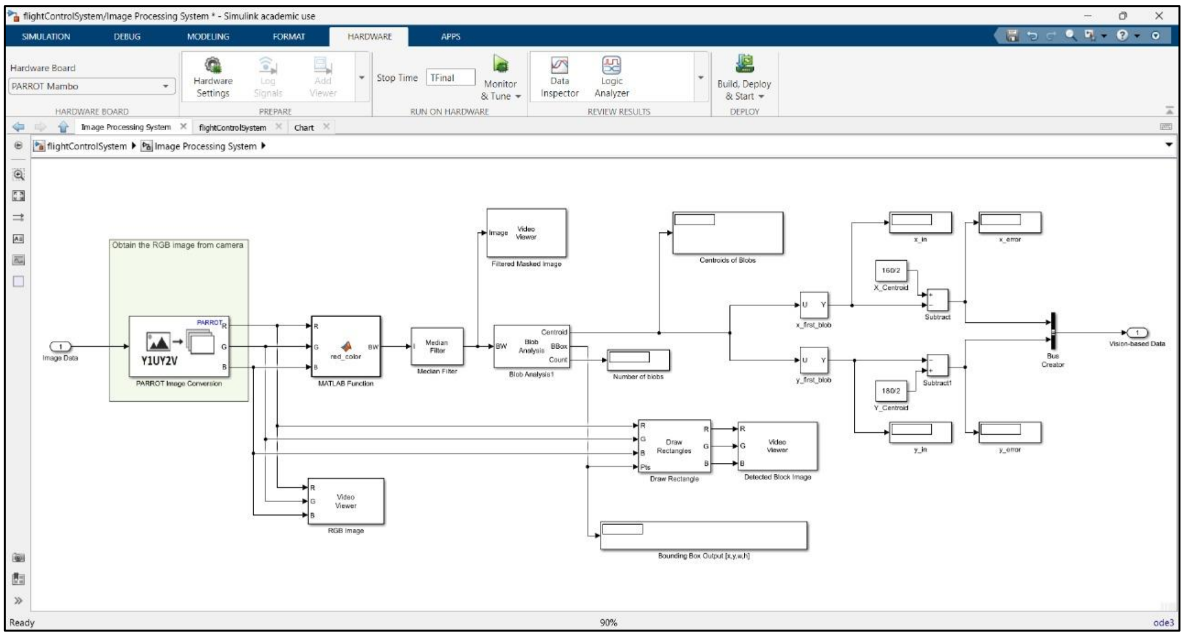
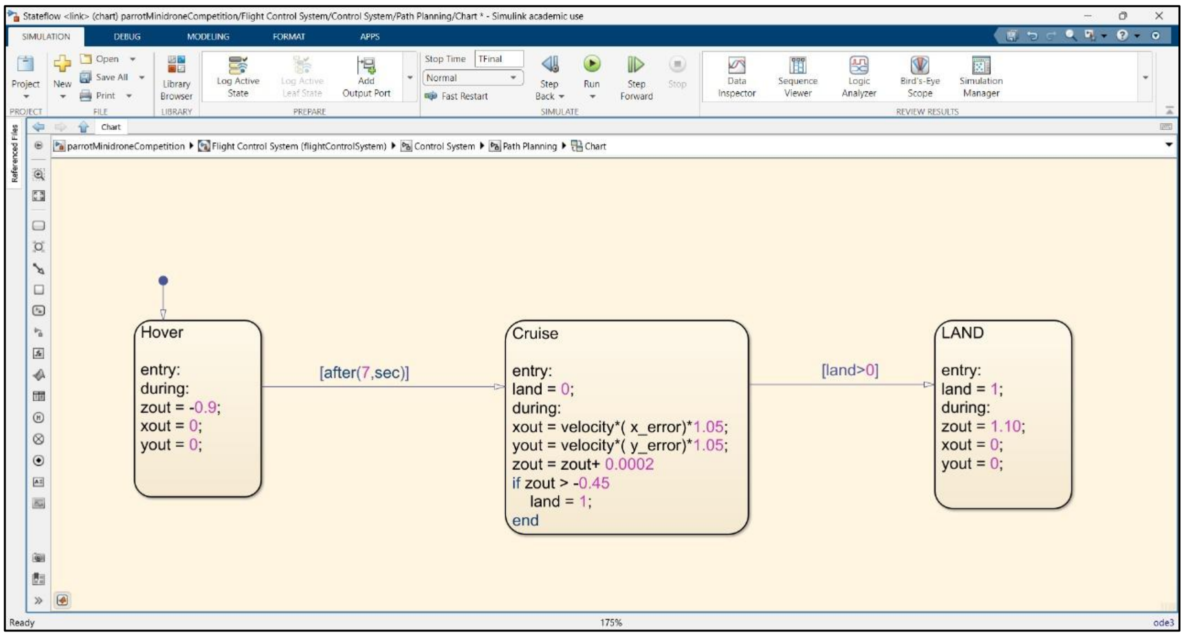


Figure 7. Image Processing Subsystem with Color Segmentation and Logic

2.3 Autonomous Platform Tracking and Descent Landing Logic

After the image processing subsystem extracted the centroid of the colored platform and computed the corresponding positional errors, the next stage involved enabling the drone to track the **moving line-follower platform** and autonomously initiate a precision landing. This behavior was governed by a single unified Stateflow chart, which controlled all flight phases, including takeoff, cruising, descent, and landing. The x_error and y_error signals—representing the offset between the platform's centroid and the center of the drone's camera view—were used to compute lateral velocity references (xout, yout). Within the **Cruise** state, these error signals were multiplied by a constant gain representing the desired horizontal velocity. The resulting commands adjusted the drone's trajectory in real time to follow the moving platform. This proportional control strategy allowed for continuous alignment corrections during flight. In addition to horizontal tracking, the same Cruise state gradually reduced the drone's **altitude (zout)** to initiate a smooth descent as the platform came into visual alignment. This was implemented by monitoring whether both x_error and y_error remained within a predefined threshold window. If alignment was maintained for a consistent duration, the Stateflow chart began incrementally increasing zout toward ground level. This approach allowed the drone to descend progressively while continuing to track the platform laterally. Once the altitude crossed a lower bound and visual alignment remained within acceptable limits, the system internally flagged that landing conditions were satisfied. The control logic then transitioned to the land state, where horizontal movement was halted (xout = 0, yout = 0) and vertical descent (zout) continued until touchdown. This separation of cruise and landing logic prevented lateral drift and ensured a stable descent over the moving platform.



```
Hover

entry:
during:
zout = -0.9;
xout = 0;
yout = 0;
```

[after(7,sec)]

```
Cruise

entry:
land = 0;
during:
xout = velocity*( x_error)*1.05;
yout = velocity*( y_error)*1.05;
zout = zout+ 0.0002
if zout > -0.45
   land = 1;
end
```

[land>0]

```
LAND

entry:
land = 1;
during:
zout = 1.10;
xout = 0;
yout = 0;
```

Figure 8. Stateflow chart for motion planning, tracking, and descent control based on real-time visual feedback.

The control architecture managing this behavior was implemented entirely within a **single Stateflow chart**, shown in Figure 8. The chart featured three principal states:

- **Hover**: An initial state allowing stabilization immediately after takeoff.
- **Cruise**: Enabled real-time horizontal tracking and altitude reduction based on visual error.
- **Land**: Triggered automatically when alignment and altitude conditions were satisfied.

Transitions between these states were governed by internal flags and timing conditions to avoid false triggers due to transient noise or brief visual misalignment. This unified control logic enabled the drone to autonomously follow the dynamic movement of the platform and execute a smooth landing, entirely driven by onboard visual feedback.

Simulink Integration:

The landing logic was fully embedded within the **Path Planning subsystem** using a unified Stateflow chart, which received x_error and y_error signals directly from the image processing block. As shown in Figure 10, these error signals were continuously monitored to guide horizontal motion during the cruise phase and evaluate platform alignment during descent. Unlike previous implementations that relied on an external landing trigger, the descent decision was made internally by the Stateflow chart. When both x_error and y_error remained within a defined threshold for a sustained duration, and the altitude (zout) had gradually decreased past a target value, the chart transitioned the drone into the land state. In this state, horizontal movement was frozen, and vertical descent continued until landing was complete. This integrated approach ensured that all navigation and landing transitions were handled within a single control structure, simplifying synchronization between vision data and motion commands. The design led to reliable and consistent landings during testing, with minimal overshoot or instability, highlighting the robustness of vision-guided autonomy using Simulink.

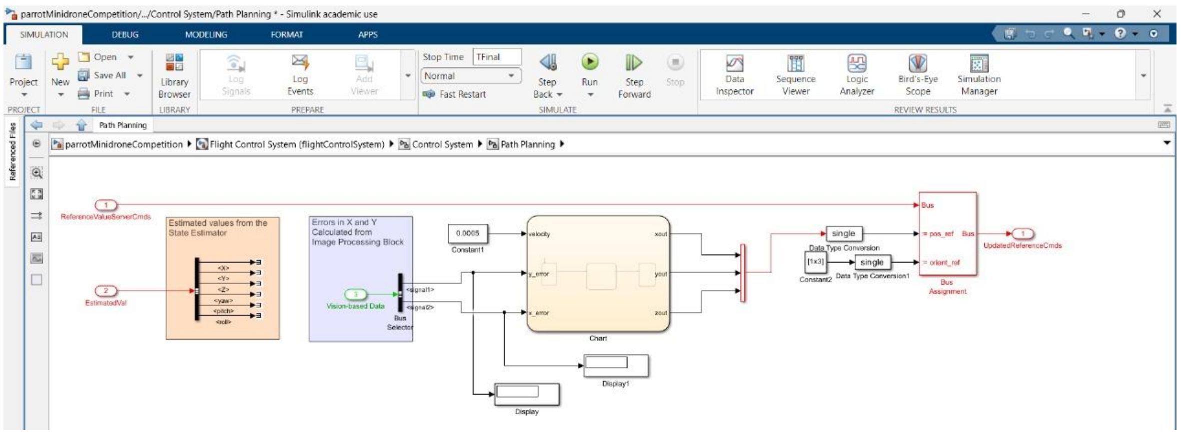


Figure 10. Path Planning Subsystem

2.5 Simulink Model Integration and Testing

After developing the image processing and control logic, all functional components were integrated into a unified Simulink model. This included the HSV-based segmentation block, blob detection and centroid tracking, path planning using position error feedback, and a single Stateflow chart responsible for real-time decision-making across all flight phases. The modularity of the model ensured seamless data flow between perception and control subsystems, while maintaining a clean architecture for debugging and tuning. Each subsystem was connected through structured data interfaces, such as bus signals carrying visual error outputs (x_error, y_error) from the image processing block to the motion control block. The control output, in the form of reference positions (xout, yout, zout), was directed to the drone's onboard motor control block, ensuring a real-time closed-loop response to visual cues.

Before hardware deployment, extensive simulations were conducted in the **Simulink 3D Animation environment**, which allowed virtual testing of tracking performance and descent timing. Simulated scenarios included varying platform speeds, lateral drifts, and alignment delays. These tests helped validate the proportional gain values used in the cruise phase, descent thresholds, and state transitions within the Stateflow chart. Following successful simulations, the complete model was deployed on the Parrot Mambo Minidrone using Simulink’s **Build, Deploy & Start** feature. Testing was conducted in an indoor setup where a line-follower robot followed a predefined black line with a red platform affixed on top. The drone autonomously detected the colored platform, corrected its trajectory in real time using visual error signals, and initiated a controlled descent once alignment and altitude conditions were met.

Throughout multiple test runs, the drone demonstrated consistent flight stability, responsive tracking of the moving platform, and soft landings. The descent phase occurred smoothly, with minimal lateral drift, confirming the robustness of the vision-guided path planning strategy and the effectiveness of Simulink’s real-time deployment workflow.

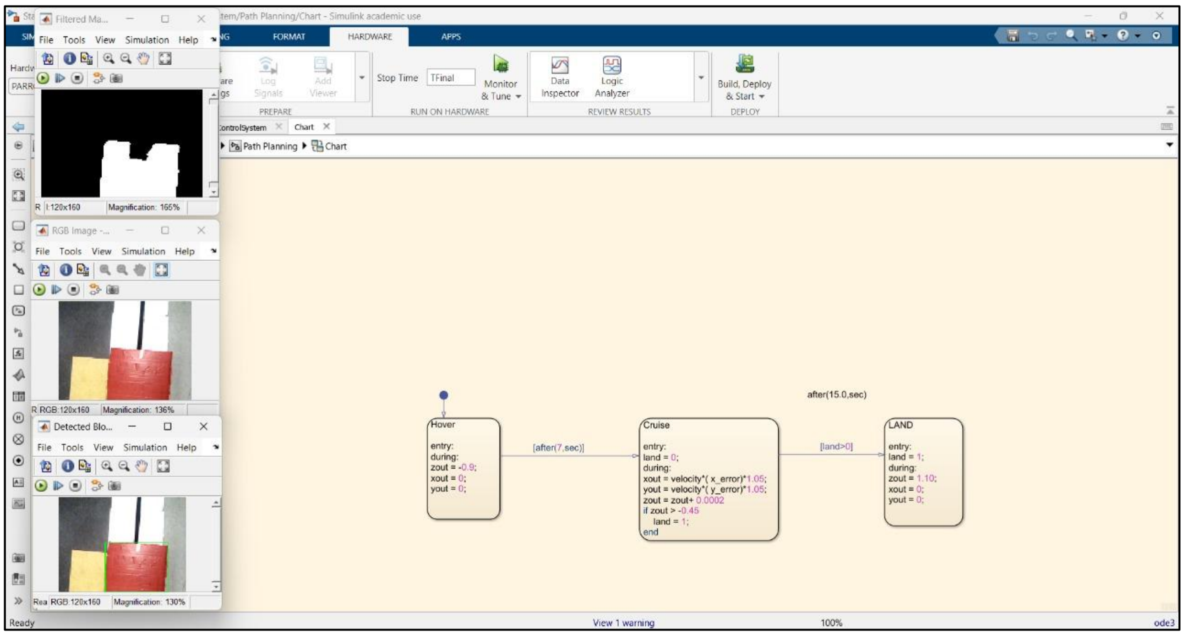
3. RESULTS AND DISCUSSION

This section presents the experimental results and system behavior observed during simulation and real-world deployment of the autonomous landing system. The focus was on evaluating the effectiveness of the image processing pipeline, the accuracy of the visual tracking system, and the stability of the drone during alignment and descent onto the moving platform.

3.1. Platform Detection and Tracking Performance

The HSV-based color segmentation method successfully identified the colored landing platform in all controlled indoor conditions. Blob analysis reliably extracted the centroid of the detected region, and the system generated clean binary masks even under mild lighting variations. The real-time tracking performance was evaluated by observing the drone's lateral response to the centroid position; the proportional controller consistently adjusted the drone’s horizontal trajectory based on x_error and y_error without oscillations or overcorrections.

Simulink visual outputs confirmed that the blob centroid remained well-tracked during the motion of the line-follower robot. Display scopes showed consistent positional errors converging toward zero as the drone approached alignment with the platform. Occasional fluctuations caused by shadows or reflections were minimized by median filtering and robust state transition logic in the controller.



```
Hover
entry:
during:
zout = -0.9;
xout = 0;
yout = 0;
```

```
Cruise
entry:
land = 0;
during:
xout = velocity*( x_error)*1.05;
yout = velocity*( y_error)*1.05;
zout = zout+ 0.0002
if zout > -0.45
    land = 1;
end
```

```
LAND
entry:
land = 1;
during:
zout = 1.10;
xout = 0;
yout = 0;
```

Figure 12. Binary mask and centroid tracking output from the HSV segmentation pipeline.

3.2. Flight Stability and Motion Control

During the **cruise phase**, the proportional control logic based on centroid error enabled the drone to continuously adjust its trajectory. The drone responded promptly to lateral deviations of the platform, maintaining visual lock even as the platform changed orientation or speed slightly. The gradual adjustment of xout and yout allowed for stable tracking without abrupt turns or oscillation.

The **Stateflow chart** managed all control transitions reliably, maintaining clean phase switching between hover, cruise, and descent. Timing logic ensured that the drone remained in cruise mode long enough to correct for alignment before initiating the landing phase.

3.3. Descent Timing and Landing Accuracy

As the drone achieved stable alignment (when both x_error and y_error fell within the predefined threshold), the Stateflow logic initiated a gradual descent. The reduction in zout was smooth and proportional, avoiding sudden drops. Final touchdown occurred with horizontal velocity set to zero, ensuring soft and stable landings with minimal drift or rebound.

Across multiple test flights, the drone consistently landed on or near the center of the moving platform, validating both the visual alignment accuracy and the altitude management logic. The onboard-only execution without external host support confirmed the viability of real-time embedded vision-based control.

4. CONCLUSION

This project successfully demonstrated the design, development, and deployment of a vision-guided autonomous landing system for the Parrot Mambo Minidrone, with the added complexity of tracking and descending onto a **moving platform**. By integrating HSV-based color segmentation, real-time blob analysis, centroid error computation, and a unified Stateflow-based control logic, the drone was able to autonomously align with a colored platform mounted on a mobile line-follower robot and execute a smooth, controlled descent. The image processing subsystem reliably identified the target platform using onboard camera input, even under varying lighting conditions. Positional errors derived from the centroid were used as feedback to continuously adjust the drone's trajectory during the cruise phase. The Stateflow chart successfully governed all flight phases—from hover and cruise to descent and landing—based entirely on real-time visual cues and internal altitude monitoring, eliminating the need for manual triggers or external positioning systems.

Simulation-based validation followed by hardware deployment confirmed the robustness of the system. The drone was able to detect the moving platform, remain centered during lateral movement, and initiate a stable vertical descent with high consistency. The system's modular structure in Simulink allowed for efficient tuning and seamless integration of visual perception and flight control. This work demonstrates the effectiveness of combining image-based feedback with state-based motion planning for precision landing tasks. It provides a strong foundation for more advanced UAV-ground coordination tasks, including multi-agent tracking, dynamic obstacle avoidance, or reinforcement learning-based navigation strategies.

REFERENCES

[1] https://www.mathworks.com/help/simulink/setup-and-configuration-parrot.html

[2] https://www.mathworks.com/help/simulink/supportpkg/parrot_ref/color-detection-and-landing-parrot-example.html

[3] https://www.mathworks.com/help/simulink/supportpkg/parrot_ref/getting-started-with-parrot-minidrone-vision.html

[4] https://youtu.be/UlDknqrtAHM?feature=shared

[5] https://www.mathworks.com/help/simulink/supportpkg/parrot_ref/path-planning-keyboard-example.html

[6] https://www.mathworks.com/academia/students/competitions/minidrones/global-drone-student-challenge.html

[7] https://www.mathworks.com/help/images/ref/colorthresholder- app.html

[8] https://www.mathworks.com/products/3d-animation.html

[9] https://www.mathworks.com/academia/student- competitions/tutorials-videos.html

BIOGRAPHIES OF AUTHORS

	Anushka Satav is a master's student at Arizona State University Pursuing Robotics and Autonomous Systems (AI). She has completed her Bachelor of Technology in Robotics and Automation at MIT World Peace University, Pune, India. She can be contacted at email: anushka.satav@asu.edu