Keyboard-Controlled UAV Navigation and Autonomous Landing Using Real-Time Color Detection

Chinmay Amrutkar[1]
[1]Arizona State University, Tempe, AZ

Keywords:

Parrot Mambo
Drone
MATLAB
Color Detection
Color Threshold
Keyboard Control

ABSTRACT

Unmanned Aerial Vehicles (UAVs) have revolutionized surveillance, automation, and environmental monitoring with their mobility and real-time data capture. This project integrated keyboard-based navigation with color-based object detection using the Parrot Mambo drone. The drone was manually controlled to hover over colored blocks (Red, Green, Blue, or Yellow) and execute a smooth landing upon detection. Its onboard camera captured live video, which was processed in MATLAB and Simulink using the RGB model, thresholding, and morphological operations. The detection system was refined to handle varying lighting conditions while ensuring optimal flight stability. Overall, the project demonstrates the successful fusion of computer vision with UAV control for autonomous navigation and smart surveillance. Future enhancements may include adaptive thresholding or machine learning for improved robustness.

Corresponding Author:

Chinmay Ravindra Amrutkar
Arizona State University, Tempe, AZ, USA
Email: chinmay.amrutkar@asu.edu

1. INTRODUCTION

Drones, or Unmanned Aerial Vehicles (UAVs), have become essential in modern technology for real-time data capture and autonomous navigation, serving key roles in surveillance, automation, and environmental monitoring. The Parrot Mambo drone, a compact and user-friendly platform equipped with a downward-facing camera and onboard sensors, provides an ideal basis for experiments in computer vision and robotics.

In this project, we extend the drone's functionality by integrating manual keyboard-based navigation with real-time, color-based object detection. The objective is to control the drone via keyboard inputs, maneuver it to hover over a specific colored block (red, green, blue, or yellow), and then trigger a controlled, slow landing once the target color is identified using MATLAB's image processing tools. This integration of manual control with automated visual response not only demonstrates the drone's potential for safe and precise navigation but also lays the groundwork for future applications in areas such as search-and-rescue, industrial inspections, and smart surveillance.

2. SETTING UP THE PARROT MAMBO DRONE

To successfully implement real-time color detection with the Parrot Mambo drone, it is essential to first establish a stable connection between the drone and the development environment. This setup involves installing the necessary drivers, configuring the communication interface, and verifying hardware functionality through initial deployment tests.

1. Installing Bluetooth Drivers and Establishing Connection:

The Parrot Mambo drone connects to MATLAB and Simulink via Bluetooth, requiring proper driver installation on a Windows system. Following the guidelines from MathWorks, the appropriate Bluetooth drivers were installed to ensure stable wireless communication. Once the drivers were set up, the connection was verified by pairing the drone with the computer and confirming its availability in the Simulink support package.

2. Deploying the Parrot Getting Started Model:

To validate the connection and test the drone's hardware, the parrot_gettingstarted model from the MathWorks documentation was deployed. This model executes a basic motor test by spinning opposite motors alternately, ensuring that all four motors function correctly. Successful execution of this test confirmed that the drone was properly configured and ready for further experimentation.

By completing this setup, the foundation was established for integrating the drone with MATLAB-based image processing and control systems, enabling further development in color-based object detection and UAV automation.

Figure 1. Parrot Mambo Drone

3. METHODLOGY

Following the successful setup of the Parrot Mambo drone, this experiment was conducted in a structured manner to achieve keyboard-based navigation integrated with real-time color detection and controlled landing. The methodology involved configuring the Simulink model by integrating real-time image processing with manual keyboard-based navigation and an automated landing mechanism on the Parrot Mambo drone. The following steps outline the process to enable the drone to hover over a target-colored block and perform a controlled landing upon detection.

3.1. Simulink Model Setup and Framework Analysis

- Launch and Template Selection:
  - Begin by launching Simulink and selecting the Code Generation Template for Image Processing and Control. This template establishes the basic framework required for integrating image processing with flight control.
- Framework Analysis:
  - Carefully analyze the pre-configured image processing block to understand its workflow and ensure that it can interface seamlessly with the drone's downward-facing camera. For additional context, execute the command:

    openExample('parrot/GettingStartedWithVisionOnPARROTExample')

  - This example provides insight into the integration of image processing and flight control functions.

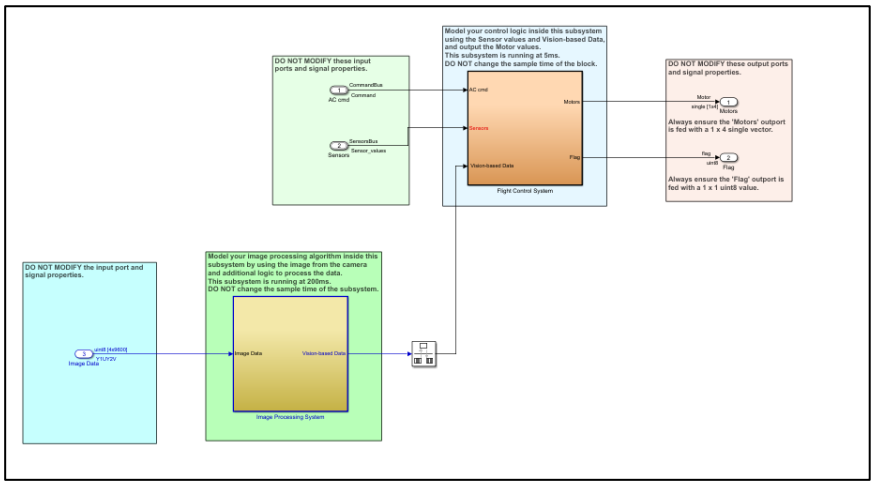

Figure 2. Code Generation Template For Image Processing and Control

3.2. Development of the Color Detection Algorithm

- Algorithm Design:
  - Develop a vision-based algorithm following MATLAB's Image Processing Guide. We utilized the MATLAB Color Thresholding App to generate detection functions for the target colors (red, green, blue, and yellow).

- Calibration and Validation:
  - Initially loading an image of the colored (red) block captured by the drone into the Thresholding App (see Figure 3) to fine-tune segmentation parameters (Figure 4). These calibrated settings were then exported as a function (Figure 5) and integrated into the image processing block, with the same process applied to all required colors (Figure 6).

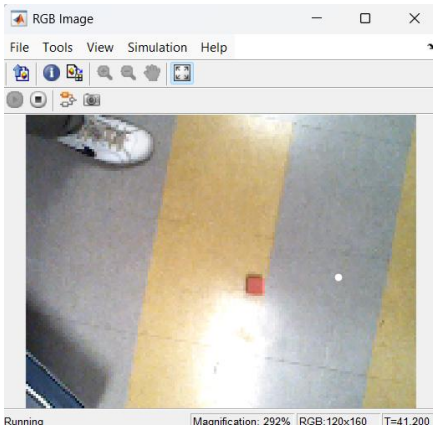

Fig. 3. Captured Image

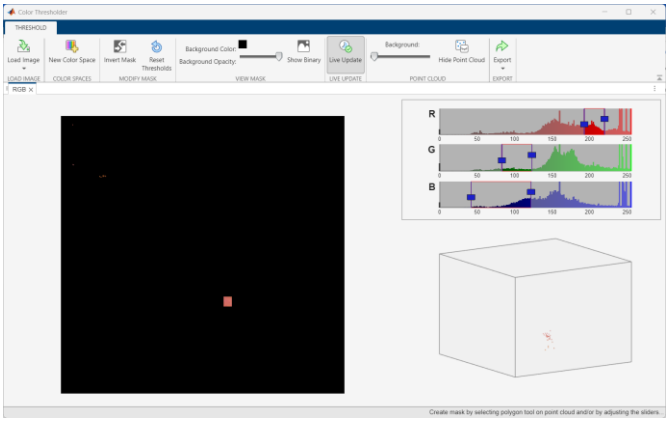

Fig. 4. Color Thresholding App (fine tuned for red color)

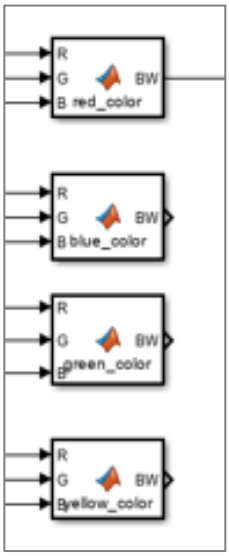

Fig. 5. Function Blocks generated

```
function BW  = red_color(R,G,B)
%createMask  Threshold RGB image using auto-generated code from colorThresholder app.
%  [BW,MASKEDRGBIMAGE] = createMask(RGB) thresholds image RGB using
%  auto-generated code from the colorThresholder app. The colorspace and
%  range for each channel of the colorspace were set within the app. The
%  segmentation mask is returned in BW, and a composite of the mask and
%  original RGB images is returned in maskedRGBImage.

% Auto-generated by colorThresholder app on 05-Feb-2025
%------------------------------------------------------


% Convert RGB image to chosen color space
I = R,G,B;

% Define thresholds for channel 1 based on histogram settings
channel1Min = 171.000;
channel1Max = 255.000;

% Define thresholds for channel 2 based on histogram settings
channel2Min = 83.000;
channel2Max = 127.000;

% Define thresholds for channel 3 based on histogram settings
channel3Min = 77.000;
channel3Max = 153.000;

% Create mask based on chosen histogram thresholds
sliderBW = (R >= channel1Min ) & (R <= channel1Max) & ...
    (G >= channel2Min ) & (G <= channel2Max) & ...
    (B >= channel3Min ) & (B <= channel3Max);
BW = sliderBW;

end
```

Fig.6. Code inside the function block auto-generated code from colorThresholder app for red color

- A video viewer block was integrated into the Simulink model to visualize the processed frames in real time, resulting in the final configuration of the Image Processing System Block (Figure 7).

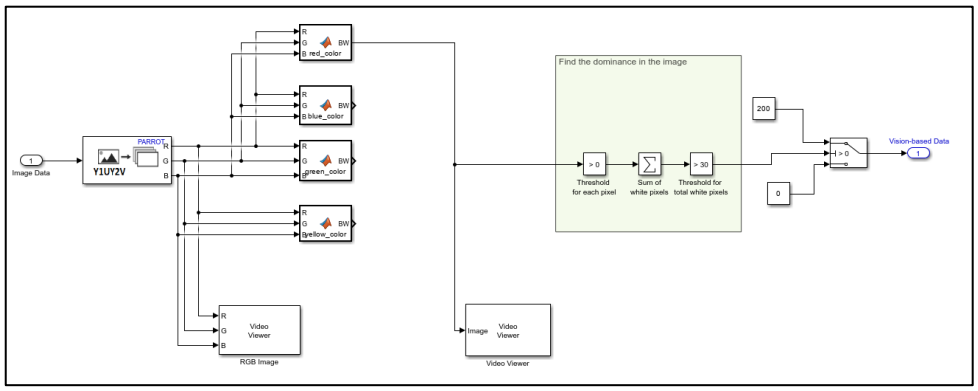

Figure 7. Image Processing System Block

3.3. Implementation of Keyboard-Based Flight Control

- Project Initialization:
  - Open the keyboard control project by executing: openExample('parrot/PathPlanningUsingParrotMinidroneExample')
- Configuring Control Logic:
  - In the Simulink model, navigate to the Flight Control System > Path Planning subsystem. You can view (Figure 8) the Keyboard Control Logic subsystem already defined in the example.
  - X-Axis: ‘w’ to move forward and ‘s’ to move backward
  - Y-Axis: ‘a’ for left and ‘d’ for right
  - Z-Axis (Altitude): ‘v’ to ascend and ‘b’ to descend
  - Yaw: ‘g’ for clockwise and ‘h’ for anticlockwise rotations

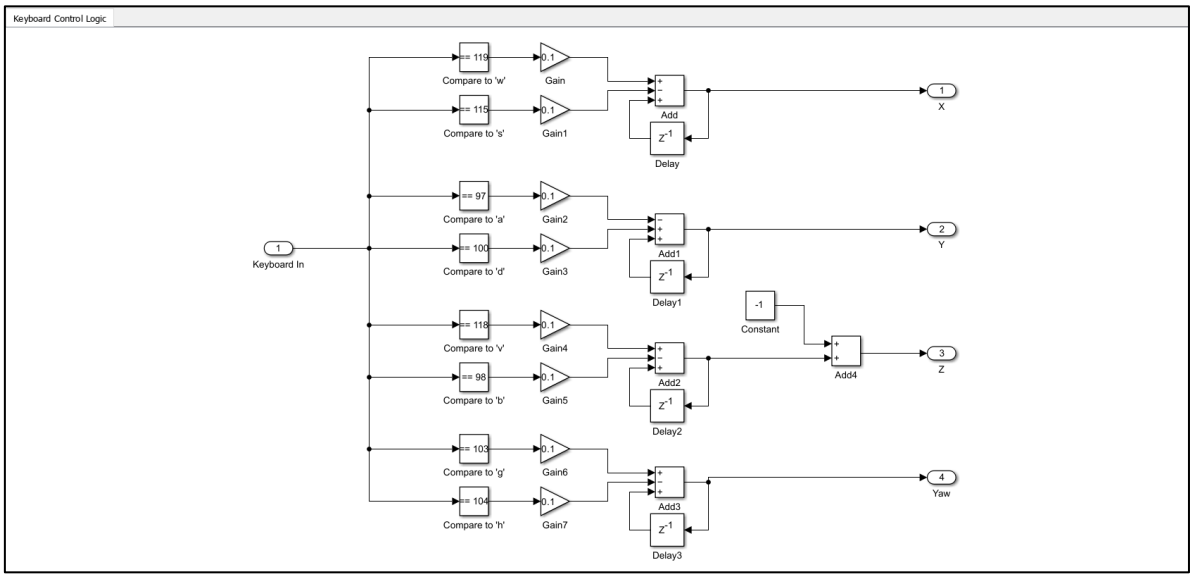

Figure 8. Keyboard Control Logic Block

3.4. Integration of Image Processing with Keyboard Control and Landing Logic

- Combining Systems:
  - Integrate the image processing module with the keyboard control system so that while manual navigation is in effect, the drone’s camera continuously processes live video to detect the target-colored block.

- Landing Enable Logic:
  - Modify the Landing Enable block (Figure 9) within the Path Planning subsystem under Flight Control Logic. Incorporate the image detection output as an input signal. When the specified color is detected, this signal triggers a gradual reduction in motor speed, enabling a smooth and controlled landing without abrupt stops.

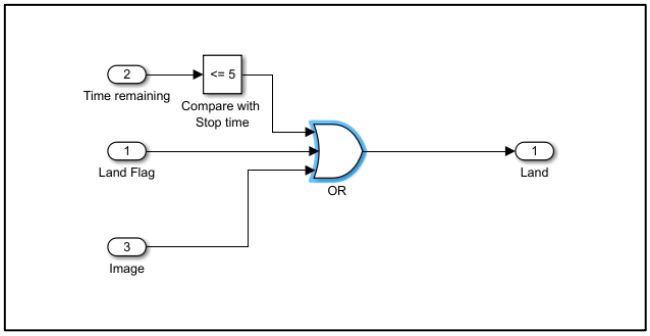

Fig. 9. Landing Enable Logic Block

3.5. Deployment of the Integrated System on the Parrot Mambo Drone

- Hardware Configuration:
  - Open the Flight Control System Model as a Top Model in Simulink and select the appropriate hardware board as Parrot Mambo under the Hardware Implementation pane in Hardware Settings.
- Building and Deploying:
  - Click Build, Deploy & Start to download and run the model on the drone. Once deployed, the drone takes off to the designated height.
- Operational Testing:
  - Use the configured keyboard controls to navigate the drone over a target colored block. Verify that upon detection, the landing logic initiates a slow descent, resulting in a controlled landing near the block.

4. RESULTS AND DISCUSSION

The implementation and testing of the integrated system—combining keyboard-based navigation with real-time color detection on the Parrot Mambo drone—provided valuable insights across multiple development stages. The evaluation focused on two key aspects: the accuracy of the image processing model in detecting the target color and the responsiveness of the drone's flight control system to keyboard commands. The results highlight both the strengths and limitations of merging manual control with an autonomous landing mechanism on a compact UAV.

Testing with the Parrot Mambo drone yielded the following observations:

- The system accurately detected the specified colored block under consistent lighting conditions.
- Real-time image processing operated efficiently, with minimal lag between color detection and the initiation of the landing sequence.
- Keyboard-based navigation allowed precise control, enabling the drone to hover effectively over the target block.
- Upon color detection, the drone executed a smooth, controlled descent, ensuring a gradual landing without abrupt stops.

Overall, the integrated approach effectively met the project's objectives by combining manual navigation with autonomous color-based landing. However, sensitivity to lighting variations and the reliance on single-color detection remain challenges that could be addressed in future enhancements through adaptive thresholding or machine learning-based approaches.

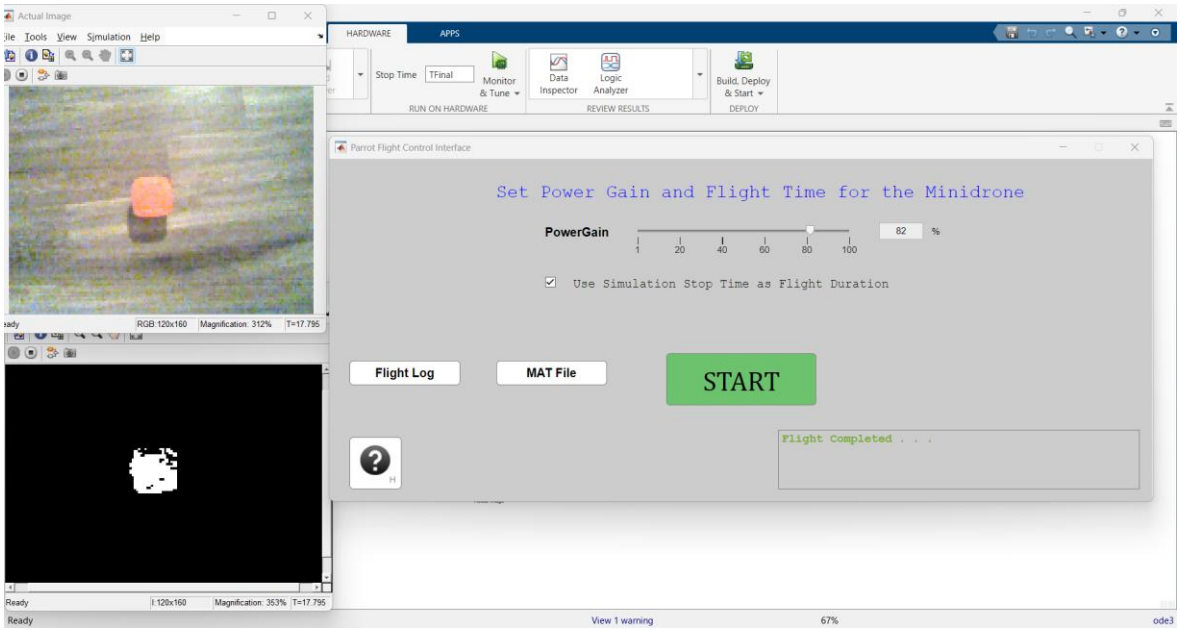

Fig. 10. Real time Testing with the Parrot Mambo drone for red color

5. CONCLUSION

This project successfully demonstrated the integration of keyboard-based navigation with real-time color detection on the Parrot Mambo drone. The primary objective—manually guiding the drone over a specified color block and initiating a controlled, gradual landing upon detecting the target color—was achieved through the seamless fusion of image processing algorithms and manual flight control via Simulink. The onboard camera effectively captured live images, which were processed using MATLAB-based techniques to reliably detect the designated color under consistent lighting conditions. Once detected, the drone executed a smooth landing, confirming the system's ability to merge autonomous decision-making with user-directed control. Although the RGB-based detection approach proved effective, its sensitivity to lighting variations highlights the need for further enhancements, such as adaptive thresholding or machine learning-based recognition, to improve robustness in diverse environments. Overall, this work lays a solid foundation for future advancements in UAV-based automation, smart surveillance, and autonomous navigation.

6. REFERENCES

1. MathWorks. (n.d.). *Install Windows Bluetooth Drivers*. Retrieved from https://www.mathworks.com/help/simulink/supportpkg/parrot_ug/install-windows-bluetooth-drivers.html

2. MathWorks. (n.d.). *Getting Started with Simulink Support Package for Parrot Minidrones*. Retrieved from https://www.mathworks.com/help/supportpkg/parrot/ref/getting-started-with-simulink-support-package-for-parrot-minidrones.html

3. MathWorks. (n.d.). *External Mode for Parrot Minidrones*. Retrieved from https://www.mathworks.com/help/supportpkg/parrot/ref/external-mode-for-parrot-minidrones.html

4. MathWorks. (n.d.). *Communicating with Parrot Minidrone Using TCP/IP and UDP*. Retrieved from https://www.mathworks.com/help/supportpkg/parrot/ref/communicating-with-parrot-minidrone-using-tcpip-and-udp.html

5. MATLAB example om Path Planning Using Keyboard Control for Parrot Minidrone https://www.mathworks.com/help/simulink/supportpkg/parrot_ref/path-planning-keyboard-example.html

7. BIOGRAPHIES OF AUTHOR

	**Chinmay Amrutkar** is a master's student at Arizona State University pursuing Robotics and Autonomous Systems (AI). He has completed his Bachelors of Technology in Robotics and Automation at MIT World Peace University, Pune, India. He can be contacted via email at camrutka@asu.edu