TDPS Individual Report

Computer Vision (Object and Color Detection) for Quadrocopter

Name: Jinming Ren

UESTC ID: 2022190908020

UofG ID: 2840216R

Team: Golden Snitch (#16)

Date: June 23, 2025

University: UoG-UESTC Joint School

Contents

Abstract

In this report, we successfully designed and implemented a Computer Vision (CV) module on a OrangePi for the quadrocopter project at UESTC. We utilized Excess Red Index and RDP algorithm together with OpenCV library to achieve two main tasks: landing area recognition and package search, with the goal of fast, robust and GPU-free real-time video stream processing.

1 Introduction

Computer vision (CV) is a crucial module for an agent (robots or drones) to have environmental perception. The position of CV is shown in the yellow box in our system diagram (Figure 1). In the following parts, we did a literature review on real-time GPU-free CV and introduce the basic theory of color space.

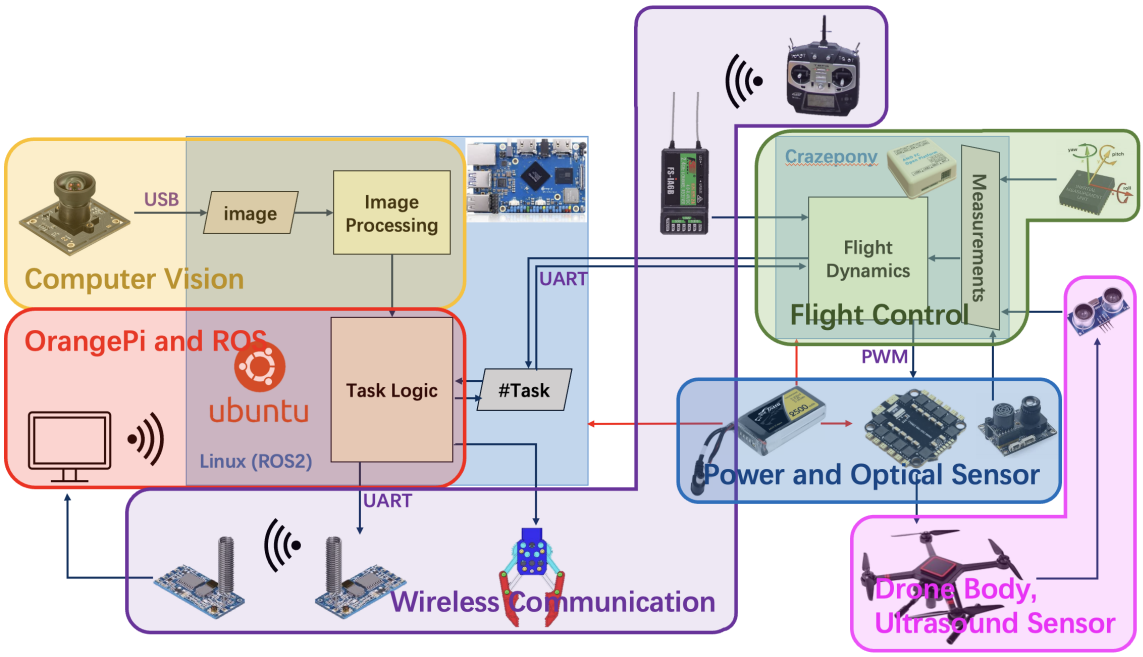


Figure 1: The position of CV in the system and their connections.

1.1 Review of real-time GPU-free CV

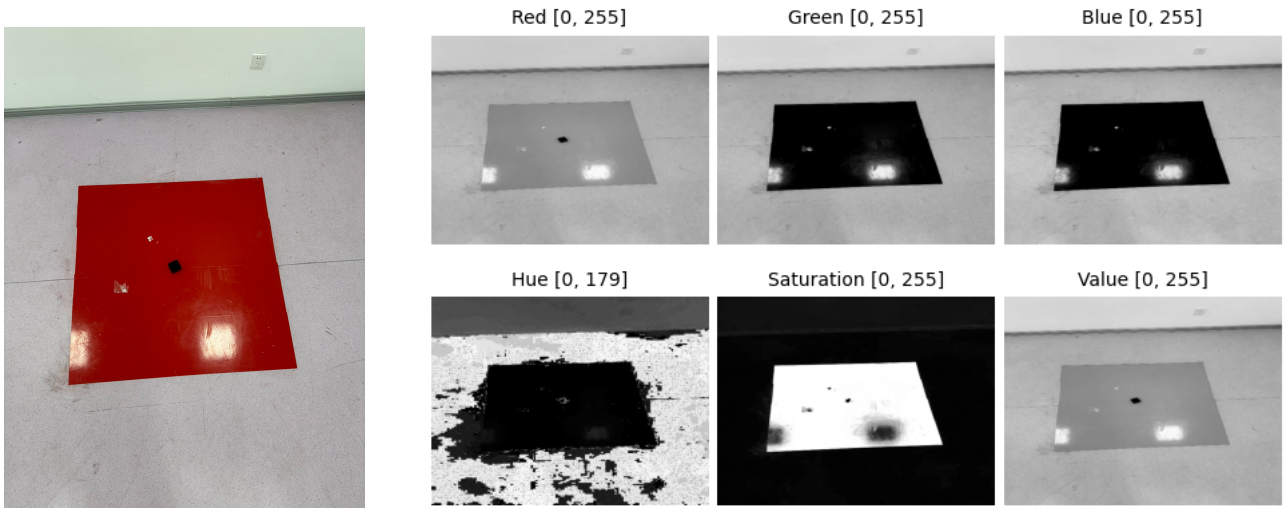
We only focus on object seperation in the scenario of quadrocopter, which is generally simple, real-time and GPU-free. So the algorithm should be fast and robust to noise and lighting variations. For color recognition, there are several classical methods [1] listed below. We used Excess color thresholding method in our project.

1. **R thresholding:** Set a hard threshold for R channel, ignoring G and B channels. The value below and above the threshold is then set to be either 255 or 0 (Binarization). Straightforward but sensitive to lighting conditions.
2. **Excess Red index:** Define index $\mathrm{Ex}R := R - f(G, B)$ and set a hard threshold for it.
3. **HSV thresholding:** Set a threshold for H, V (ignore S channel to tolerate glare [2]).
4. **Mixed thresholding:** Combination of the above methods.

1.2 Basic theory of color space

The set of all colors has the structure of a 3-dimensional vector space, called the **color space**. Different choice of basis leads to different color models. We will use the famous two models: **RGB** (Red, Green, Blue) and **HSV**[1] (Hue, Saturation, Value), demonstrated in Figure 2.

[1] **Hue** is the color type, which can be represented by an angle in the color wheel. Highly **saturated** colors appear vivid, low **saturation** makes them look grayish. **Value** is the brightness of the picture.



(a) Original image (b) Coordinate values in RGB and HSV color space

Figure 2: Any image can be decomposed into RGB and HSV color space.

2 Experimental Design

2.1 Software & Hardware

The CV module is composed of a camera and an algorithm running on the OrangePi. We choose IMX462-H264 as our camera module, it is connected to the OrangePi through USB port, which is a device `/dev/video0` on Linux. We also utilize `cv2`, a Python library which provides tons of common functions for image processing, to implement the algorithm.

2.2 Proposed CV Algorithm

The input of the CV module is a video stream. The output is a topic[2] in ROS, which contains 6 status information shown in Figure 3. It turns out that these 6 values is enough for the drone to sense the environment. These 6 values could be derived from the coordinates of the vertices of the square. Therefore, the problem reduces to finding vertices. We show these steps in the following sections: **Color recognition, edge detection** and **vertices detection.**

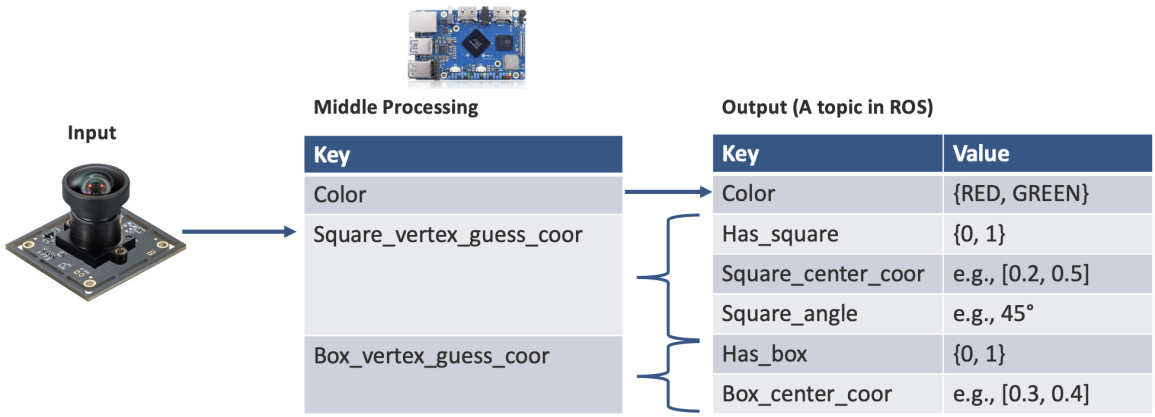


Figure 3: The objective of the CV module.

[2]We are using ROS2 operating system on OrangePi, where we use `Node` to encapsulate a module and `Topic` to share information between modules. The topic published by the CV Node is `CV_status`, which contains the 6 calculated values as shown in Figure 3.

2.2.1 Color Recognition

We demonstrate the recognition process mainly for the red square only since the green region is analogous. To find a more robust method between R thresholding and ExR (where $f(G, B) = (G + B)/2$), we set different hard threshold values for R channel and H, V channels to get a sense of the appropriate threshold intervals, within which the red square should be clearly distinguished from the floor. The wider the interval, the more robust the method is.

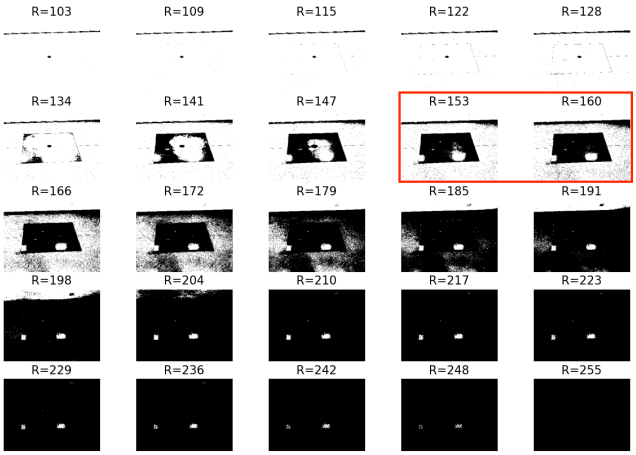


Figure 4: R channel thresholding

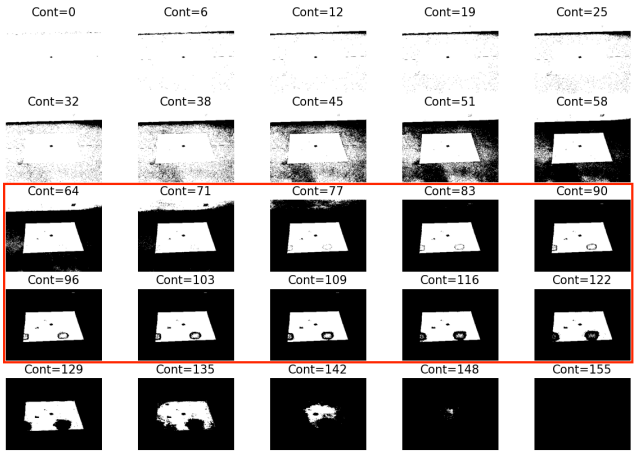


Figure 5: Excess red index (More robust)

As shown in Figure 4 and 5, the intervals are $[153, 160]$ and $[64, 122]$, respectively. Therefore, ExR is more robust to noise and lighting variations. We choose[3] $C_R = 77$ and $C_G = 87$ as the threshold of the excess index for the red and green square respectively.

2.2.2 Edge Detection

we use `cv2.findContours()` function to find the contours of the red square. The return value of `cv2.findContours()` is a list of points on the edge of the square, which is plotted on Figure 6 and 7. The falsely detected edges was resolved by `cv2.contourArea()` function. We use it to sort the contours by area, and keep the largest one as the target square. This acts as a filter to remove noise and small objects. The results are shown in Figure 8 and 9.

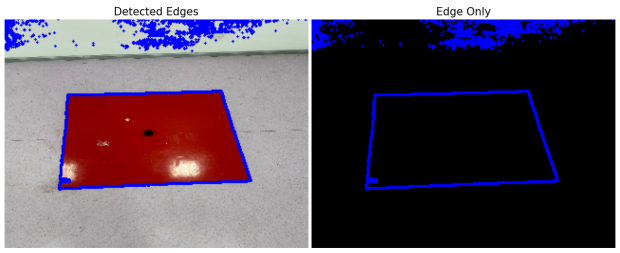


Figure 6: Edge detection for red square

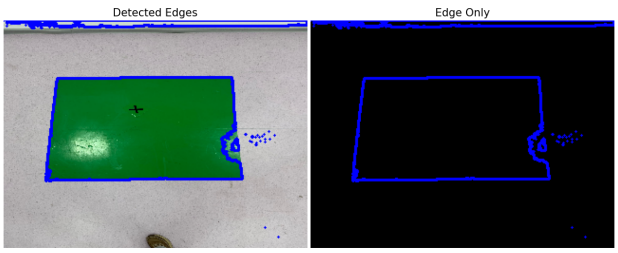


Figure 7: Edge detection for green square

2.2.3 Vertices Detection

We apply the `cv2.approxPolyDP()` function on the filtered contours from section 2.2.2 to approximate the polygon of the square. The coordinates of the polygon are the vertices. This function uses the **Ramer-Douglas-Peucker (RDP) algorithm** to reduce the number of points in a curve approximated by a series of points. The results are shown in Figure 8 and 9.

[3]Since we are not dealing with black object picking task, it is not a problem for any pattern within the detected region as long as the edge is clear and distinguishable.

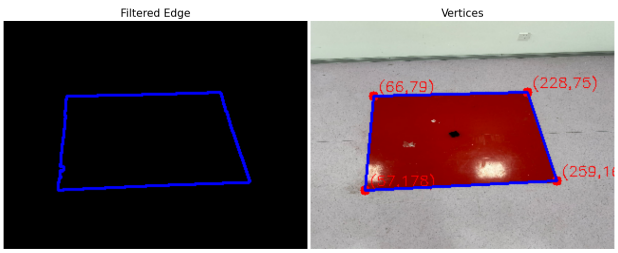


Figure 8: Vertices for red square

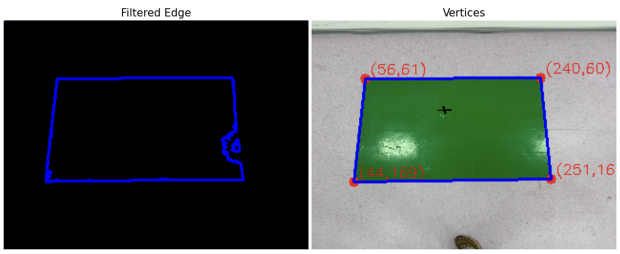


Figure 9: Vertices for green square

2.3 Convert middle processing information to output

Suppose the 4 vertices are $\{V_i = (x_i, y_i)\}_{i=1}^4$, we derive each key in Figure 3 is as follows:

1. **`Color:`** Depends on the threshold value we applied, which is controled by the program logic and state information in the OrangePi.

2. **`Has_square/Has_box:`** If the 4 vertices are detected, then it is set to 1, otherwise 0.

3. **`Square_center_coor/Box_center_coor:`** Compute by

$$(x_c, y_c) = \left(\frac{x_1 + x_2 + x_3 + x_4}{4}, \frac{y_1 + y_2 + y_3 + y_4}{4} \right).$$

4. **`Square_angle:`** Sort the y coordinates of all vertices and pick two smallest y_1, y_2 (without loss of generality) and their corresponding x coordinates x_1, x_2. Then the angle is computed by

$$\theta = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1} \right).$$

3 Results and Analysis

We implemented the computer vision module and rigorously tested its performance under both daytime and nighttime lighting conditions to evaluate its robustness and reliability in real-world scenarios. The experimental results demonstrate that in approximately 90% of the trials, the module correctly identified the landing area, enabling the quadcopter to complete the landing task safely. This high success rate confirms that the selected algorithm is sufficiently accurate for practical deployment in indoor drone missions. However, a notable limitation emerged due to the fixed focal length of the onboard camera. In certain cases, especially when the drone experienced motion blur or variable lighting, the Ramer-Douglas-Peucker (RDP) algorithm used for vertex detection exhibited divergence. This occasionally led to incorrect estimation of the squares corners, resulting in positional errors that could disrupt the drones alignment and produce unpredictable flight behavior. These findings highlight the importance of refining the perception pipeline, especially under dynamic visual conditions.

4 Conclusion and Future Work

In this report, we presented the design and implementation of a lightweight, real-time computer vision algorithm that enables the drone to autonomously recognize both the pickup package and the designated landing pad. The algorithm is deliberately constructed to operate without GPU acceleration, leveraging efficient techniques such as the Excess Red (ExR) index

for color segmentation and the RDP algorithm for geometric simplification. This approach balances computational efficiency with detection reliability, making it well-suited for deployment on embedded systems such as the Orange Pi. Despite its simplicity, the method proved to be robust against typical visual noise and lighting variation encountered during indoor flights. Looking ahead, there are several promising directions for improvement. One potential enhancement is to introduce adaptive or soft thresholding techniques, which could dynamically adjust to lighting changes in real time. Alternatively, lightweight machine learning models, such as quantized CNNs or edge-optimized vision transformers, could be explored to improve detection precision without significantly increasing computational load. These upgrades would allow the system to better handle edge cases such as partial occlusions, reflective surfaces, or non-uniform illumination, ultimately improving the reliability and versatility of the vision module.

References

[1] Nizamuddin Maitlo, Nooruddin Noonari, Sajid Ahmed Ghanghro, Sathishkumar Duraisamy, and Fayaz Ahmed. Color recognition in challenging lighting environments: Cnn approach. *ArXiv (Cornell University)*, 02 2024.

[2] Handoko Handoko, Jehoshua Hanky Pratama, and Banu Wirawan Yohanes. Traffic sign detection optimization using color and shape segmentation as pre-processing system. *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, 19:173, 02 2021.