Proceedings of the 5th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/121/2025.19740

Object Detection Based on HSV in OpenCV

Puchuan Song[1,a,*]

[1]Department of Computer Science, Virginia Polytechnic Institute and State University, VA, 24060, US
a. spuchuan19@vt.edu
**corresponding author*

Abstract: In the realm of computer vision and automation, the ability to identify and track objects in real time is critical for various applications, including robotics, surveillance, and machine control. This paper explores the process of controlling a Raspberry Pi computer equipped with a camera to detect and locate circular objects with a green color. By leveraging the power of open-source libraries and image processing techniques, the system performs real-time analysis of camera input to identify target objects based on color and shape. The research employs a literature review methodology, synthesizing information from relevant studies on computer vision, object recognition, and color filtering techniques. The paper focuses on the use of algorithms for color segmentation, contour detection, and shape recognition to achieve accurate object identification. Through this method, it examines the effectiveness and limitations of using a Raspberry Pi for object detection tasks in constrained environments. The findings indicate that Raspberry Pi, when coupled with optimized algorithms, provides a cost-effective solution for basic object detection tasks, though its performance is limited by processing power compared to more robust systems. This review emphasizes the significance of affordable solutions for educational and small-scale automation purposes, highlighting their potential impact on future applications in low-cost robotics and IoT devices.

Keywords: OpenCV, HSV, computer vision, objective recognition

1. Introduction

In recent years, advances in computer vision and embedded systems have led to the development of cost-effective solutions for real-time object detection and tracking. These technologies play an increasingly important role in fields such as robotics, surveillance, and automation. With the integration of low-cost computing platforms like the Raspberry Pi, researchers have been able to develop systems capable of performing sophisticated tasks, such as object recognition and tracking, in a highly efficient and affordable manner. This study focuses on detecting green circular objects using a Raspberry Pi and OpenCV, utilizing color segmentation techniques in the HSV color space for real-time object detection.

Various studies have demonstrated the effectiveness of the Raspberry Pi for color-based object detection. One study highlights how Raspberry Pi combined with the HSV color space offers a more robust method for color segmentation in different lighting conditions, providing a more reliable solution for tracking objects compared to the RGB model [1]. Another study showcased the implementation of Raspberry Pi and OpenCV to detect objects using color thresholds in HSV space, noting that this approach is particularly useful for low-cost applications such as robotics and automation [2]. These findings underscore the practical applications of Raspberry Pi in computer vision projects where affordability and real-time processing are critical.

This paper employs a literature review methodology to analyze existing techniques in color segmentation and shape recognition. By evaluating the effectiveness of Raspberry Pi for identifying green circular objects, this study provides insights into how such low-cost systems can be applied in various fields, particularly for educational and research purposes [3]. The findings from this research contribute to the broader understanding of affordable, accessible solutions for real-time object detection in constrained environments.

2. Code Analysis

2.1. Importing libraries

The code begins by importing several key libraries necessary for image processing and hardware control. The cv2 library from OpenCV is imported to handle image and video processing tasks, allowing the code to perform real-time object detection using methods like color space conversion and contour detection. This is supported by the numpy library, imported as np, which provides efficient array and matrix operations essential for handling image data. Furthermore, the ServoKit class from the Adafruit ServoKit library is imported to control servo motors, enabling physical responses such as adjusting a camera's position or controlling a robotic arm based on the detected object. Finally, the time module is included to manage timing functions, such as introducing delays that ensure synchronization between the detection and motor movements, ensuring smooth operation of the system [4].

2.2. Motors initialization

This part of the code initializes and controls two servos using the ServoKit class, which manages up to 16 channels. The kit = ServoKit(channels=16) command sets up a ServoKit object to control up to 16 servos, which are connected to a Raspberry Pi or similar microcontroller. The next lines initialize default positions for the two servos controlling the pan and tilt movements, typically associated with a camera or sensor. Here, the pan (horizontal rotation) and tilt (vertical rotation) are both set to 90 degrees, positioning the servos at a centered, neutral angle. The lines kit.servo[0].angle = pan and kit.servo[1].angle = tilt apply these angles to the servos connected to channels 0 and 1, respectively, ensuring the system starts in a default, ready position. This setup allows the servos to be adjusted based on future input, such as the location of a detected object, to dynamically move and track the object.

2.3. Format Convert

```
def bgr8_to_jpeg(value, quality=75):
    return bytes(cv2.imencode('.jpg', value)[1])
```

This section of the code integrates the functionality of capturing image data and converting it into a format that can be easily displayed in a Jupyter Notebook interface. The function bgr8_to_jpeg() takes an image frame in BGR format (the standard format used by OpenCV) and converts it into JPEG format by using the cv2.imencode(), which encodes the image into bytes. This allows for the image to be transmitted or displayed more efficiently.

The code then imports widgets from ipywidgets and traitlets, which are libraries used for building interactive GUIs in Jupyter Notebooks. Two widget objects, FGmaskComp_img and frame_img, are created to hold images with the specified format (jpeg) and dimensions (320x240 pixels). These widgets will be used to display the processed image data, such as frames from the camera and the corresponding mask generated during object detection.

The widgets.HBox() function arranges the two image widgets horizontally, allowing for side-by-side comparison of the original frame and its processed version, such as the masked image. Finally, the display() function outputs this combined image display in the notebook, allowing users to view the real-time object detection results interactively. This integration of visual feedback within the Jupyter Notebook enables seamless development and debugging of the object detection system.

2.4. Threads Control

This section of the code introduces the use of Python's `threading` library to manage and control threads, which are used to run multiple tasks concurrently. In particular, it provides a mechanism for stopping a running thread, which is a non-trivial task in Python [5].

The async_raise() function takes two arguments: the thread ID (tid) and the exception type (exctype). It uses the ctypes library to raise an asynchronous exception in the context of the specified thread. First, the thread ID is cast to a ctypes.c_long type, and the exception type is validated to ensure it's a class. The PyThreadState_SetAsyncExc() function from Python's C API is called to raise the exception in the target thread, which, in this case, is typically used to raise a SystemExit exception to stop the thread.

If the result of calling PyThreadState_SetAsyncExc() is 0, it means the thread ID is invalid, and a ValueError is raised. If the result is not 1, it indicates a failure to raise the exception properly, and a SystemError is raised. This ensures that the thread-stopping mechanism behaves correctly and reliably handles errors.

The stop_thread() function is a convenience function that takes a thread object as an argument and stops the thread by calling async_raise() with the thread's identifier (thread.ident) and a SystemExit exception, which cleanly terminates the thread's execution.

This implementation is a workaround to the absence of a native thread-stopping mechanism in Python, as Python does not provide a direct API for killing threads. However, it's important to note that forcefully stopping threads in this manner can be risky, as it might leave resources in an inconsistent state or fail to properly clean up. Therefore, this approach should be used with caution.

2.5. Camera Setup

This section of the code sets up the camera using the Picamera2 library, an updated Python library for controlling the Raspberry Pi camera with the libcamera framework. The `Picamera2` library is a more advanced and flexible alternative to the original picamera library, allowing for better integration with the libcamera stack, which is designed to support modern camera systems.

The code starts by initializing the Picamera2 object, which manages camera configurations and interactions. The create_preview_configuration() function defines two configurations. Main stream sets the main image format to `RGB888`, which stores images in the RGB color space (8 bits per channel), with a resolution of 320x240 pixels. This is typically used for display or preview purposes. Raw stream captures data in `SRGGB12` format (12-bit raw Bayer data), with a resolution of 1920x1080 pixels. This raw data is useful for advanced image processing tasks that require unprocessed pixel values.

Next, the transformation settings are applied with libcamera.Transform(), specifying that the image should be vertically flipped (vflip=1), which is commonly needed in setups where the camera is mounted upside down or rotated. The configuration is then applied using picamera.configure(), and the camera starts capturing with picamera.start().

Finally, the width and height of the main image stream are set to 320x240 pixels, which defines the resolution at which the camera captures and processes preview images. This setup allows for a dual-purpose capture: one for real-time preview with lower resolution and one for raw data capture, which can be processed later.

This configuration is useful in applications requiring real-time preview and high-quality image capture, such as object detection systems or image processing tasks where the preview helps visualize the scene, while the raw stream captures more detailed data for further analysis.

2.6. UI

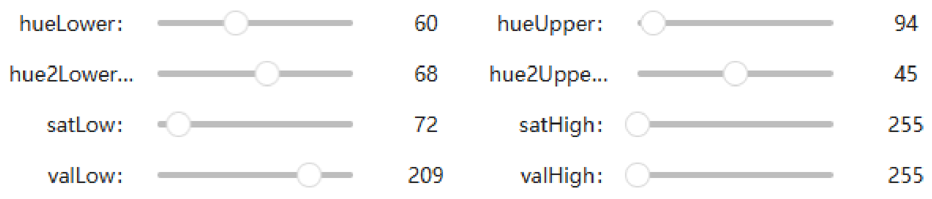
This portion of the code sets up interactive sliders using the ipywidgets library to dynamically adjust HSV (Hue, Saturation, Value) values for image processing tasks in real time. Sliders allow the user to fine-tune the thresholds for color segmentation, making it easier to detect objects based on specific color ranges in the HSV color space.

The hueLower and hueUpper sliders allow for the adjustment of the lower and upper bounds for the hue component of the HSV color space. This is particularly useful for defining the color range of the object you want to detect. The range is set from 40 to 90 for hueLower, and from 90 to 140 for hueUpper, which corresponds to shades of green. The unused sliders hue2Lower and hue2Upper are set to zero, indicating they aren't needed for this specific implementation.

The satLow and satHigh sliders adjust the saturation bounds, determining how "intense" the color must be to be detected. Lower values correspond to more muted colors, while higher values correspond to more vibrant ones. In this case, the lower limit is set at 100, meaning that very dull or grayish objects are excluded from detection, while the upper limit is fixed at 255, capturing the most saturated tones.

The valLow and valHigh sliders control the value (brightness) range. These settings determine how light or dark the detected object can be. The lower limit is set to 100, excluding very dark areas, while the upper limit remains fixed at 255, capturing the brightest parts.

Finally, the widgets.VBox() and widgets.HBox() functions arrange these sliders into a vertical box layout, with each pair of sliders (for hue, saturation, and value) grouped together horizontally. The display() function outputs the sliders in the Jupyter Notebook, allowing real-time adjustments during the object detection process. This setup is ideal for fine-tuning the color detection thresholds and visually observing how the changes impact the object segmentation, making it a flexible tool for interactive development in image processing.



```
hueLower:      60    hueUpper:    94
hue2Lower...   68    hue2Uppe...  45
satLow:        72    satHigh:    255
valLow:       209    valHigh:    255
```

Figure 1: User Interface for adjustment on HSV

2.7. Frame capture and motor control

This function, Video_display(), is designed to continuously capture video frames from the camera, process them to detect specific objects based on color thresholds, and adjust servo motors accordingly to track the detected object. The function is executed in an infinite loop, ensuring that the system continually updates its detection and response in real time [6].

The code captures each frame using the picamera.capture_array() function and converts it from BGR to the HSV color space using cv2.cvtColor(). This HSV conversion is necessary because the hue-saturation-value model is more efficient for color-based segmentation, allowing the system to focus on detecting specific colors.

The values for the HSV thresholds are dynamically retrieved from sliders (hueLower, hueUpper, satLow, etc.) created earlier in the script. These allow for flexible real-time tuning, meaning users can modify the color detection ranges interactively. Two sets of lower and upper boundaries are defined (l_b, u_b for the primary hue and l_b2, u_b2 for a secondary range), though the secondary set appears to be optional.

The cv2.inRange() function is used to create binary masks (FGmask and FGmask2) that isolate pixels within the specified HSV range. These two masks are then combined with cv2.add() into FGmaskComp, which is the final mask used for object detection. The mask highlights areas of the frame that match the specified color range.

The contours of the detected regions are identified using cv2.findContours(). These contours represent the boundaries of detected objects. The contours are sorted by area, ensuring that the largest object is considered first. A loop processes each contour, checking if its area is larger than a minimum threshold (50 in this case) to filter out small, irrelevant objects. For each valid contour, the code calculates the minimum enclosing circle using cv2.minEnclosingCircle(), which approximates the object’s shape as a circle.

If the detected object's radius is larger than 10, its position (center coordinates) is used to adjust the servos controlling the camera’s pan and tilt angles. Errors (errorPan and errorTilt) are calculated as the difference between the object's location and the center of the frame. These errors are used to incrementally adjust the servo angles, ensuring that the camera tracks the object. Boundaries are imposed to ensure that the pan and tilt angles remain within the physical range of the servos (0 to 180 degrees). When the object moves within 15 pixels of the frame's center, adjustments are minimized to prevent jittery movements.

The kit.servo[0].angle and kit.servo[1].angle lines adjust the angles of the servos that control the camera's movement, ensuring the camera tracks the object smoothly. The pan and tilt values are inverted (180 - pan/tilt) to account for the physical orientation of the servos.

The processed video frame (with the detected object outlined by a circle) and the corresponding binary mask (FGmaskComp) are converted to JPEG format using the bgr8_to_jpeg() function and displayed in real-time using the frame_img and FGmaskComp_img widgets. This provides a visual interface for monitoring the system’s operation.

In summary, this function continuously captures video, processes it for object detection, and controls servos to track a detected object, all while providing real-time feedback via a graphical display. This system is well-suited for applications where real-time object tracking and response are necessary, such as in robotics or surveillance systems.

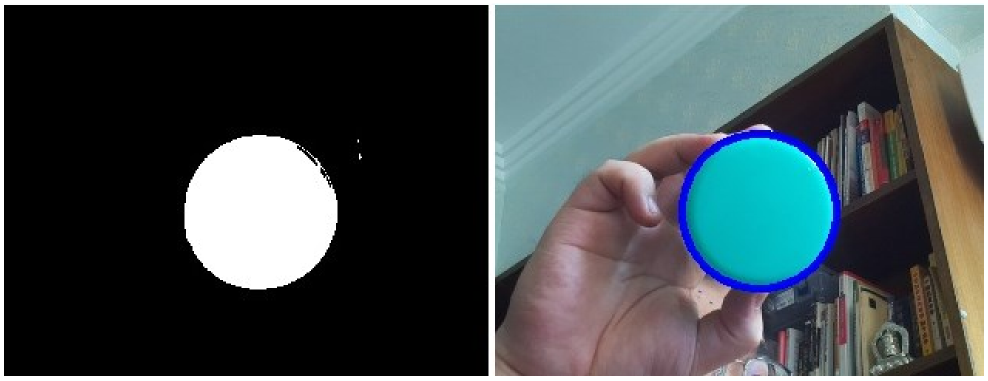


Figure 2: performance on detecting a specific object

3. Discussion

A key strength of the system is its integration with servo motors via the Adafruit ServoKit library, enabling real-time physical responses to detected objects. The servos' pan and tilt adjustments ensure that the camera tracks the object, allowing the system to follow the detected object smoothly and maintain it within the frame. This capability is crucial in applications such as robotics, where the camera's ability to autonomously follow a moving object is essential for tasks like surveillance, automated inspection, or interactive robotics.

However, certain limitations can be observed in the system. First, the reliance on HSV thresholds means that it may struggle with objects of similar colors to the background or in environments with extreme lighting conditions [7]. Although the HSV model is superior to the RGB model for color segmentation, it is still sensitive to lighting and shadows, which could lead to false positives or missed detections.

Another area for improvement lies in the control of the servos [8]. Currently, the pan and tilt values are adjusted based on the center of the detected object, but the system could be made more responsive by implementing more sophisticated algorithms, such as PID (Proportional-Integral-Derivative) controllers, to smooth out the tracking movements and reduce overshooting or jitter. This would enhance the stability of the tracking system, especially when the object is moving erratically or quickly.

In summary, while the current implementation achieves its core functionality of real-time object detection and tracking using affordable hardware, there are opportunities for enhancing robustness, precision, and efficiency. This system, as it stands, provides a solid foundation for affordable, interactive vision-based applications in robotics, education, and home automation.

4. Conclusion

This study presents a cost-effective and accessible approach to real-time object detection and tracking using a Raspberry Pi, OpenCV, and servo motors. By leveraging HSV color space for image segmentation and dynamically adjusting thresholds via sliders, the system effectively tracks and follows a green circular object. The integration of servo motors controlled by the Adafruit ServoKit library enables the camera to respond physically, ensuring continuous tracking of the detected object. The research demonstrates the practical application of these technologies in automation, robotics, and educational projects, highlighting their value in creating interactive and real-time vision systems.

However, there are some limitations in this study. The system's reliance on basic HSV-based color detection means it can struggle in environments with varying lighting or where the object's color is similar to the background. Future improvements could include the use of advanced object detection techniques such as deep learning-based models or edge detection algorithms, which would enhance robustness and accuracy. Additionally, the tracking system could benefit from more sophisticated servo control algorithms, such as a PID controller, to reduce jitter and improve stability in object tracking.

Looking ahead, the potential for vision-based object detection systems is significant. With continuous advancements in machine learning, image processing, and hardware capabilities, more robust and versatile systems can be developed for broader applications, including autonomous navigation, advanced robotics, and IoT-based monitoring systems. This study lays the groundwork for further exploration and development in affordable real-time vision systems.

References

[1] *A. Whittaker, "Colour-based object tracking with Raspberry Pi - Raspberry Pi, " Colour-based object tracking with Raspberry Pi, https://www.raspberrypi.com/news/colour-based-object-tracking-with-raspberry-pi/ (accessed Sep. 13, 2023).*

[2] *M. Aqib, "How to create object detection with opencv and Raspberry Pi: Raspberry pi, " Maker Pro, https://maker.pro/raspberry-pi/tutorial/how-to-create-object-detection-with-opencv (accessed Oct. 28, 2024).*

[3] *S. Sural, Gang Qian and S. Pramanik, "Segmentation and histogram generation using the HSV color space for image retrieval, " Proceedings. International Conference on Image Processing, Rochester, NY, USA, 2002, pp. II-II, doi: 10.1109/ICIP.2002.1040019.*

[4] *S. Patil, U. Bhangale and N. More, "Comparative study of color iris recognition: DCT vs. vector quantization approaches in rgb and hsv color spaces, " 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, India, 2017, pp. 1600-1603, doi: 10.1109/ICACCI.2017.8126070.*

[5] *H. Zhu et al., "Computer Vision-Based Beef Grading for Marbling Beef Based on RGB-HSV Color Model and Python Software, " 2023 3rd International Signal Processing, Communications and Engineering Management Conference (ISPCEM), Montreal, QC, Canada, 2023, pp. 609-614, doi: 10.1109/ISPCEM60569.2023.00115.*

[6] *B. Muhammad and S. A. Rahman Abu-Bakar, "A hybrid skin color detection using HSV and YCgCr color space for face detection, " 2015 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 2015, pp. 95-98, doi: 10.1109/ICSIPA.2015.7412170.*

[7] *E. Susilowati, S. Madenda, S. Arief Sudiro and L. ETP, "Color Features Extraction Based on Min-Max Value from RGB, HSV, and HCL on Medan Oranges Image, " 2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT), Makassar, Indonesia, 2018, pp. 312-317, doi: 10.1109/EIConCIT.2018.8878516.*

[8] *L. Feng, L. Xiaoyu and C. Yi, "An efficient detection method for rare colored capsule based on RGB and HSV color space, " 2014 IEEE International Conference on Granular Computing (GrC), Noboribetsu, Japan, 2014, pp. 175-178, doi: 10.1109/GRC.2014.6982830.*