Gheorghe Asachi Technical University of Iași
Faculty of Automatic Control and Computer Engineering
Department of Automatic Control and Applied Informatics

Tracking an object of a specific color

Authors:
Silvia Teodora PORCARASU

2023

CUPRINS:

1. Introduction

This work aims to provide a practical and efficient solution for real-time monitoring and tracking of color-based objects, with applications in fields such as robotics, autonomous assistance, security surveillance, and more.

In the given scenario, a student holds a red/blue/green object in one hand and an object of a different color in the other. The goal of the project, *"Tracking an Object of a Specific Color,"* is to develop a system capable of identifying and tracking only the red/blue/green object in real time.

Steps:

1. Develop a GUI for real-time image acquisition
2. Implement the tracking algorithm
3. Display the result in the graphical interface

2. Development of a GUI for Real-Time Image Acquisition

This chapter details the development of a graphical user interface (GUI) designed for real-time image acquisition and color-based object tracking. The GUI enables users to select specific colors to track within a live video feed, with controls to start and stop the tracking process.

Main Application Entry Point: The core of the application is encapsulated in the function color_tracking_gui, which serves as the primary entry point for launching the GUI.

GUI Window Setup: The main application window is created using the MATLAB uifigure function. This function allows specification of the window's title and its initial screen position.

Video Display Area: An axes object (ax) is instantiated within the main window to serve as the display area for the video stream. This component is critical for presenting real-time video to the user.

Control Elements:

Two control buttons are integrated into the interface:

- **Start Button (start_button)**: Positioned at a designated location in the window, this button initiates the tracking process. Its ButtonPushedFcn callback is configured to invoke the startTracking function upon user interaction.
- **Stop Button (stop_button)**: Similarly placed, this button halts the tracking process. Its ButtonPushedFcn callback is set to call the stopTracking function when clicked.

Additionally, a dropdown menu (color_dropdown) is provided to allow users to select the color they wish to track. The dropdown contains three options: Red, Blue, and Green, with Red set as the default selection.

Video Streaming Functionality:

The GUI incorporates functionality to display a live video stream. This feature supports both real-time viewing and playback of video files within the application window, providing a dynamic interface for the user.

A global variable, vid, is used to manage the video input object, ensuring accessibility across functions within the application.

Tracking Data Management: To manage the tracking data, the variable tracked_bb is initialized as an empty array. This array will be populated with the coordinates defining the bounding box around the tracked object during operation.

Tracking Initialization:

The startTracking function is defined to control the commencement of video streaming and tracking. It accepts the axparameter, indicating the axes object where the video stream should be rendered.

Within this function, the MATLAB preview function is called to start the video stream and display it in the GUI. The image object reference within the axes is stored in the variable hImg to facilitate further manipulation.

The decision to use preview instead of start was driven by limitations in the MATLAB version and toolbox availability, ensuring compatibility and stable video display.

3. Algorithm Implementation for Real-Time Color Tracking Image Processing and Display in the Interface

The core algorithm operates within a continuous loop that processes each video frame captured in real time. The following steps outline the process flow:

1. **Frame Capture:**
   Inside the `while` loop, a single frame is acquired from the video stream using the `getsnapshot` function.
2. **Color Selection and Extraction:**
   The currently selected color from the dropdown menu is read. The corresponding color channel is extracted from the captured image frame.
3. **Color Highlighting:**
   To emphasize the selected color, the extracted color channel is subtracted from the grayscale version of the captured image.
4. **Noise Reduction:**
   Median filtering is applied to the difference image to reduce noise and smooth out irregularities.
5. **Thresholding:**
   The filtered difference image is converted into a binary image by applying a fixed threshold.
6. **Small Object Removal:**
   Small, irrelevant objects are removed from the binary image to improve accuracy.
7. **Connected Component Labeling:**
   Connected components in the binary image are identified and labeled.
8. **Region Property Extraction:**
   The `regionprops` function is used to obtain the properties of each connected component, specifically the bounding box and centroid.
9. **Video Frame Update:**
   The current video frame is updated on the GUI by setting the image data of the axes object.
10. **Object Detection and Selection:**
    - If objects are detected (i.e., the `stats` variable is not empty), the bounding box of the object with the largest area is determined by iterating through all detected objects and comparing their areas.
    - If an object with an area greater than an initial threshold (e.g., `-Inf`) is found, its bounding box coordinates are extracted.
11. **Tracking Logic:**
    - If a previously tracked object exists (`tracked_bb` is not empty), the displacement between the current bounding box centroid and the previous one is calculated.

- The tracked bounding box is updated only if this displacement is below a predefined threshold (e.g., 50 pixels). The previous bounding box is cleared, and the new bounding box is drawn on the GUI.
- If the displacement exceeds the threshold, the tracked bounding box is cleared, indicating loss of tracking.

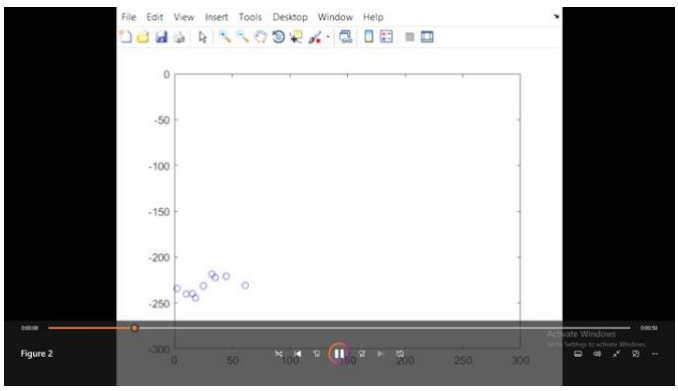
12. **Initial Object Tracking:**
    - If no previously tracked object exists, the current bounding box is assigned as the tracked object.
13. **No Object Detected:**
    - If no objects are detected, the tracked bounding box is cleared.
14. **Axis Update and Loop Delay:**
    - The axes (`ax`) is refreshed and a short pause is introduced to allow processing and display updating.

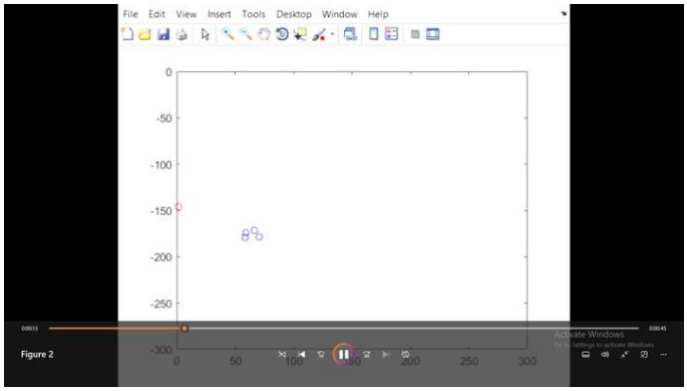
Stopping the Tracking Process: The stopTracking function halts the video stream by calling the stoppreview function and clears the axes with the clafunction, effectively resetting the display.

4. Approach and Overcoming Challenges

During the development of the real-time color tracking GUI, I consulted various online resources to understand the implementation of object tracking algorithms. One such resource provided useful insights; however, it did not perform object tracking within the same GUI window. Instead, it opened a separate graphical window that displayed only the centroid (center of mass) of the detected object.

This limitation led to additional challenges, as integrating tracking visualization directly within the main GUI window was essential for a seamless user experience. To overcome this, I adapted and extended the approach by embedding the tracking visualization—such as drawing bounding boxes and updating the video feed—directly within the primary interface. This ensured that users could monitor the tracked object in real time without the distraction of multiple windows.

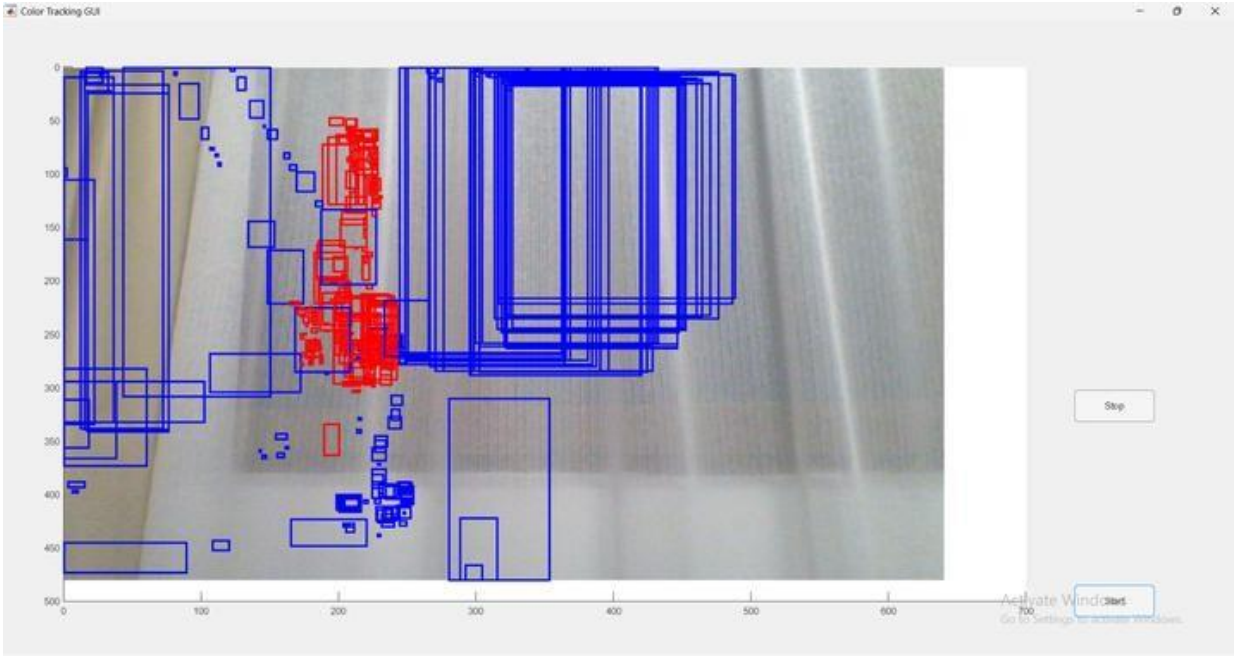




Following this, I enhanced the tracking visualization by adding a bounding box around the detected object using the `rectangle` function. This replaced the previous `plot` method, which was responsible for opening a new, separate window. By doing so, the tracking rectangle was integrated directly into the main GUI window, improving the user experience.

Additionally, I implemented a dropdown menu (`color_dropdown`) to allow users to select the desired color for tracking dynamically. This addition provided greater flexibility and interactivity within the application.

However, after introducing the bounding box, a new challenge arose: the rectangle did not disappear automatically when the tracked object left the frame. This required further logic to ensure that the bounding box would be removed or hidden promptly whenever the object was no longer detected, maintaining the interface's visual clarity.



To address the issue of the bounding box not disappearing when the tracked object left the frame, I implemented a variable named **displacement**. This variable represents the difference between the current position of the object and its previous position.

Initially, **displacement** was assigned the value of the previous position of the object. It effectively measures the distance the object has moved between two consecutive time points. To verify whether the displacement was acceptable, I compared its value against a threshold of 50 pixels. If the displacement was less than 50, it was assumed that the tracked object had moved within an acceptable range, and the tracking rectangle was updated accordingly. In this case, the bounding box was shifted to the new position, and the previous value of **displacement** was cleared.

For real-time video display, the program was set to capture frames every 0.01 seconds using the `getsnapshot` function. This ensured smooth and continuous video streaming within the GUI.

Challenges in Green Color Detection:

One of the difficulties encountered during the detection of green shades was related to the limitations of the RGB/grayscale color space. RGB and grayscale spaces can struggle to precisely separate subtle variations of green from other colors in the spectrum, leading to less accurate tracking.

After research, a proposed solution to this problem is to switch the color space from RGB/grayscale to **HSV (Hue, Saturation, Value)**. The HSV color space offers a more natural and intuitive representation of colors by separating hue (color type), saturation (color intensity), and value (brightness). This separation

facilitates more accurate color segmentation and improves the robustness of detecting different shades of green.