Color Sorting Robot Arm

Description:

This project is a system that combines ROS2 with an Arduino-based control unit and a WiFi camera to create a color-detection and block-sorting robot. The system uses ROS2 nodes (written in Python) to send movement instructions over serial communication to the Arduino, which responds with control actions. In order to send the movement instructions, a dedicated camera node retrieves still images from a WiFi-streaming camera, processes those images with OpenCV for color detection (using defined HSV thresholds), and interacts with other functions through ROS2 services. Integrating serial communication, network-based image capture, and real-time control enables the robot to detect and sort colored blocks autonomously.

Motivation:

The primary motivation behind the project was to bridge the gap between software and hardware. By combining ROS2 with Arduino and computer vision techniques, the project allowed for hands-on learning in several key areas: real-time sensor integration, robust serial communication, and image processing with OpenCV. The challenge of creating a modular and interoperable system that could perceive, decide, and act pushed the boundaries of my robotics knowledge and provided an opportunity to develop a seamless integration between diverse technological components.

Results:

The project successfully demonstrated:

- **Reliable Serial Communication:** With proper delays, permission handling, and buffering, movement commands sent from a ROS2 node were correctly received and acted upon by the Arduino.
- **Image Capture and Color Detection:** The WiFi camera node effectively captures still images via an HTTP request. The integrated color detection algorithm, using HSV thresholding, accurately identified dominant colors, which is critical for the block-sorting task.
- **Modular ROS2 Integration:** By implementing both a topic-based velocity node and a service-based camera node, the system showed flexible methods of inter-node communication and control, vital for scaling up to more complex robotic behaviors.

Challenges:

Several challenges needed to be addressed during the project:

- **Serial Communication Latency:** Initially, the system required multiple attempts to send commands reliably. This was mitigated by adding delays to allow the Arduino time to initialize, flushing the serial buffer, and carefully managing port permissions.
- **Permission Issues on Linux:** Working with a VM and USB devices, permissions became a hurdle. The solution involved adding the user to the appropriate groups and adjusting file permissions.
- **Color Range Tuning:** Determining the correct HSV ranges for red (especially given that red wraps around the hue wheel) required additional tweaking.
- **Integration Complexity:** Coordinating multiple ROS2 nodes (for control and image processing) and ensuring they communicated reliably introduced its own set of challenges. Systematic debugging and isolation of issues were key to overcoming these obstacles.

What I Learned:

Throughout this project, I gained invaluable insights across several domains:

- **Interdisciplinary Integration:** I learned how to mix robotics control, network communication, and computer vision into a cohesive system using ROS2.
- **Best Practices for Serial Communication:** Handling initialization delays, permission management, and buffer flushing were critical lessons, emphasizing the importance of timing and system readiness.
- **ROS2 Services and Topics:** Implementing both a service-based and topic-based approach provided a deeper understanding of when and how to leverage ROS2's communication paradigms for modularity and scalability.
- **Image Processing with OpenCV:** Working with HSV color spaces and adjusting thresholds to accurately detect block colors reinforced practical skills in digital image analysis.
- **Problem-Solving and Debugging:** The project honed my troubleshooting skills, from resolving network scanning issues to handling USB device conflicts in a virtualized environment.

Code Description:

There are three parts to the code: Arduino, ROS, and the camera.

- The Arduino code includes movement instructions for the arm. There are three potential movement options: move to green, move to blue, or move to red. In order for the arm to move, it receives serial information from ROS.
- Within ROS, there are two nodes: camera and velocity.
    - The camera node integrates with the WiFi camera and embeds the color detection algorithm. There are three main components to this node:

        - HTTP Image Retrieval: uses the Requests library to send an HTTP GET request to the camera's snapshot URL
        - Color Detection via HSV Thresholding: Converts the captured image to the HSV color space and applies predefined threshold values to detect colors.
        - Toggle Service (Using Trigger Service): the node implements a ROS2 service (/detect_color) that, when invoked, triggers the image capture and color detection routine. The node then responds with a message indicating which colors have been detected.
        - toggle_operation Service: when set to True, the node will operate normally when the Toggle Service is called. Otherwise, nothing happens. The service acts like an on/off switch.
    - The velocity node is responsible for transmitting movement information to the Arduino via serial communication. There are two main components to this node:
        - Opens the serial port (e.g., /dev/ttyACM0) at a specified baud rate (9600)
        - Message Submission: listens (or receives through topic subscription) for identified colors (such as "red"). It then encodes these commands and writes them over the serial port to the Arduino.
- The camera (ESP32-CAM), creates a web server to stream its feed, which allows the camera node to access the latest image when prompted. The examples CameraWebServer.ino was used to establish the web server.

Instructions:

1. Upload the Arduino code to the Arduino and the camera code to the camera. When the code has been uploaded to the camera, copy and paste the link outputted to the serial monitor to confirm the server is up and running.
    a. Because the camera uses WiFi, it needs a username and password to get the server set up.
    b. The url may have changed from the last time the camera was used.
2. Before starting the VM, open its settings and navigate to the USB section and, while the Arduino is plugged in, add it to the Device Filers list. Then, confirm the settings and start the VM.
3. Check if the VM can interact with the port the Arduino is using:
    a. `lsusb`
4. If the port does not show up, change serial permissions:
    a. sudo chmod 666 /dev/ttyACM0
5. Run `lsusb` again to confirm that the port shows up.
6. Build the ROS2 workspace with the package for this project inside and source the environment in two separate terminals:
    a. `cd ros2_ws/`
    b. `colcon build`
    c. `source install/setup.bash`
7. Ensure that the camera url is correct in the camera node.
8. Launch the launch file to start up both the camera and velocity nodes in one of the terminals:
    a. `ros2 launch final_project rob_arm_launch.py`
9. To test the current state of `/toggle_operation`, in the second terminal, run:
    a. `ros2 service call /capture_image std_srvs/srv/Trigger “{}”`
10. To start change the state of /toggle_operation, run:
    a. `ros2 service call /toggle_operation std_svs/srv/SetBool “{data: true}”`
11. Rerun the command from step 9 to capture (and save) the latest image.
    a. If the camera node identifies a red, blue, or green object, the arm will then pick up the object and place it in the designated location.