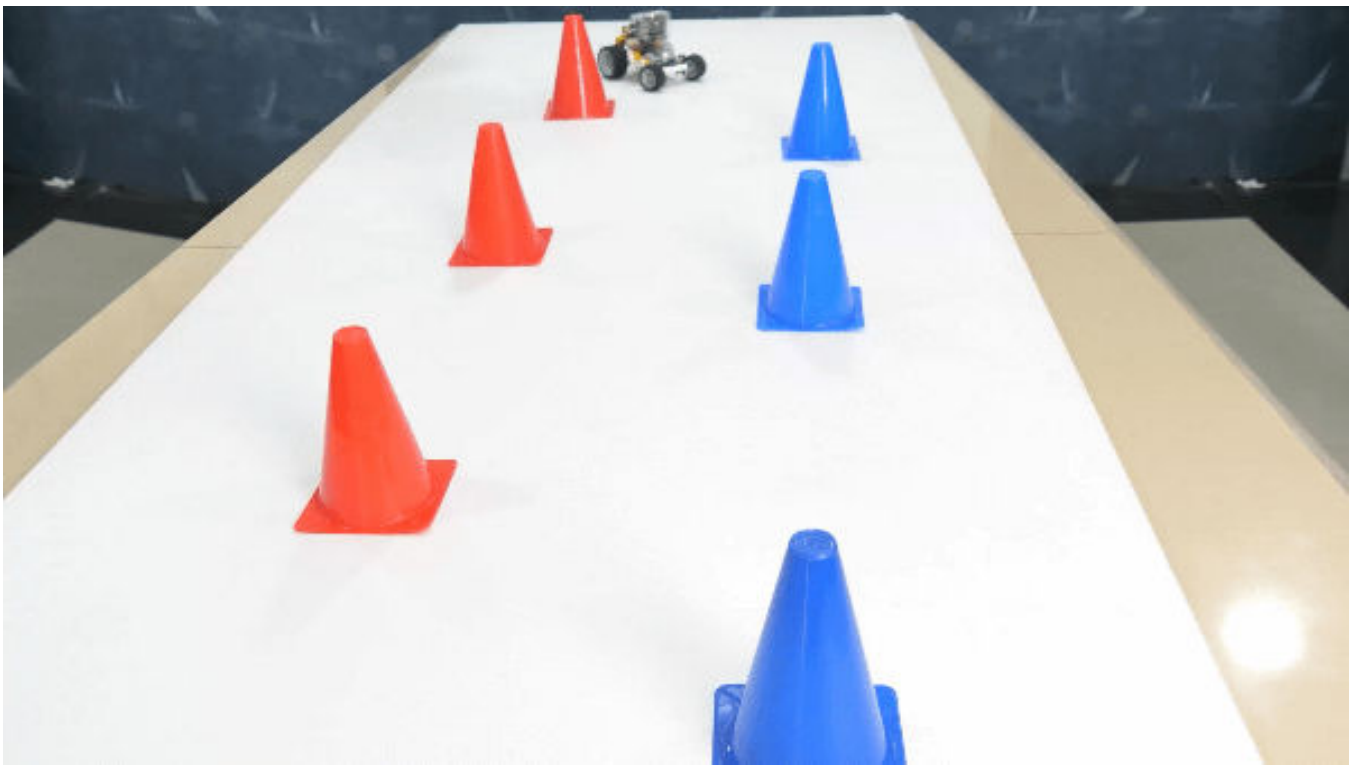
INTELLIO ROVER

Obstacle Avoider



1. Introduction

The Obstacle Avoider is a computer-vision project for the Intellio rover, built and run through PictoBlox. A Python script on your computer receives a live video stream from Intellio's camera over WiFi, looks for red and blue cones in the frame, and sends steering and motor commands back to Intellio in real time — turning left whenever it sees a red cone and right whenever it sees a blue cone. Together, this lets the rover steer itself through a field of cones on its own, without anyone driving it manually.

2. What You Need

- A laptop or PC with **PictoBlox** installed.
- An **Intellio** rover
- A WiFi network that both the computer and Intellio are connected to
- A set of **red cones** and **blue cones** (or red/blue coloured objects) for the track
- A large, open, well-lit area to lay out the track and let the rover move freely

3. Steps to Run the Project

1. Open **PictoBlox** and connect to Intellio using WiFi.
2. Once connected, save the file and run the code directly from PictoBlox.

4. Setting Up the Track

Arrange the cones in a large, open area — place all the **red cones on the right side** of the path and all the **blue cones on the left side**, one after the other spacing them out so the rover has room to react between one cone and the next. Once the code is running, the rover will drive forward and weave through the cones in a zig-zag motion: it steers left whenever a red cone comes into view, and right whenever a blue cone comes into view.

5. How the Code Works

Setup and Configuration

At the beginning of the script, the **Mini Expansion** and **Intellio** are initialized. The configuration section then defines the UDP video-stream port, motor speed, servo centre and steering limits, contour detection threshold, and obstacle persistence time.

The rover uses a servo centre of **90°**, with a steering range from **55° (hard left)** to **125° (hard right)**. The forward speed is set to **60**, and an obstacle must have a contour area greater than **3000 pixels** to be considered during detection.

Receiving the Video Feed

The **VideoReceiver** class receives the camera stream through UDP on `127.0.0.1:6000`. It runs in a separate background thread so that receiving video does not block the main control loop.

Each frame starts with a `SIZE:` **header**, which tells the receiver how many bytes make up the complete image. The receiver collects the UDP packets until the expected number of bytes is received, then uses NumPy and OpenCV to decode the data into an image frame.

The most recently decoded frame is stored safely using a **thread lock**, allowing the main loop to access it while the receiver thread continues listening for new frames.

Detecting Red and Blue Obstacles

The functions `get_red_mask` and `get_blue_mask` convert each camera frame from BGR to HSV colour space and apply colour thresholds to isolate red and blue regions.

Red uses two HSV ranges because red appears at both ends of the HSV hue scale. Blue is detected using a single HSV range.

After thresholding, a **5 × 5 morphological kernel** is used with opening and closing operations. Opening removes small noise, while closing fills small gaps in the detected regions.

The resulting masks are then used to find contours representing possible obstacles.

Selecting the Obstacle

The main loop finds contours from both the red and blue masks. Only contours with an area greater than **3000 pixels** are considered valid obstacles.

The **largest valid red contour is checked first**. If a red obstacle is found, the rover treats it as an obstacle requiring a **left turn**.

If no valid red obstacle is found, the code checks for the **largest valid blue contour**. A blue obstacle causes the rover to turn **right**.

For the selected contour, the code calculates its bounding rectangle and finds its horizontal centre (`cx`). This centre position is used to determine how much the steering servo should turn.

Calculating the Steering Angle

The `calc_servo_angle` function converts the obstacle's horizontal position in the camera frame into a servo angle.

The frame position is normalized between `0.0` and `1.0`. A **power curve with an exponent of 2.0** is then used to make the steering response stronger, particularly for obstacles that are farther from the centre.

For a red obstacle, the servo moves from **90° toward 55°**. For a blue obstacle, it moves from **90° toward 125°**.

The final angle is clipped so that the servo never goes outside the configured steering range.

Persistence After an Obstacle Disappears

The code uses `OBSTACLE_PERSISTENCE_SEC = 0.8` to prevent the rover from immediately returning to straight driving when an obstacle temporarily disappears from the camera frame.

Whenever a red or blue obstacle is detected, its direction, steering angle, and expiry time are stored in `last_obstacle`.

If the obstacle disappears but the 0.8-second persistence period has not expired, the rover continues using the last steering angle. Once the timer expires, the stored obstacle is cleared and the servo returns to the centre position of **90°**.

Controlling the Rover

The rover is controlled directly through the initialized hardware objects.

The steering servo is controlled using:

```
intellio.write("S1", servo_angle)
```

The two motors are controlled using the Mini Expansion:

```
mini.runbothmotors('1', '1', FORWARD_SPEED, FORWARD_SPEED)
```

Therefore, the main loop continuously updates the steering servo according to the detected obstacle while commanding both motors to move forward at the configured speed.

If there is no camera frame available, the rover is stopped using `mini.stoprobot()` until the video stream becomes available again.

Displaying Feedback

The program opens three OpenCV windows:

- **Video feed** — shows the live camera image, detected obstacle bounding boxes, obstacle centre points, and the current steering decision.
- **Red Mask** — shows the regions detected as red.
- **Blue Mask** — shows the regions detected as blue.

The video feed displays decisions such as **RED -> LEFT**, **BLUE -> RIGHT**, or **FORWARD**, along with the calculated servo angle.

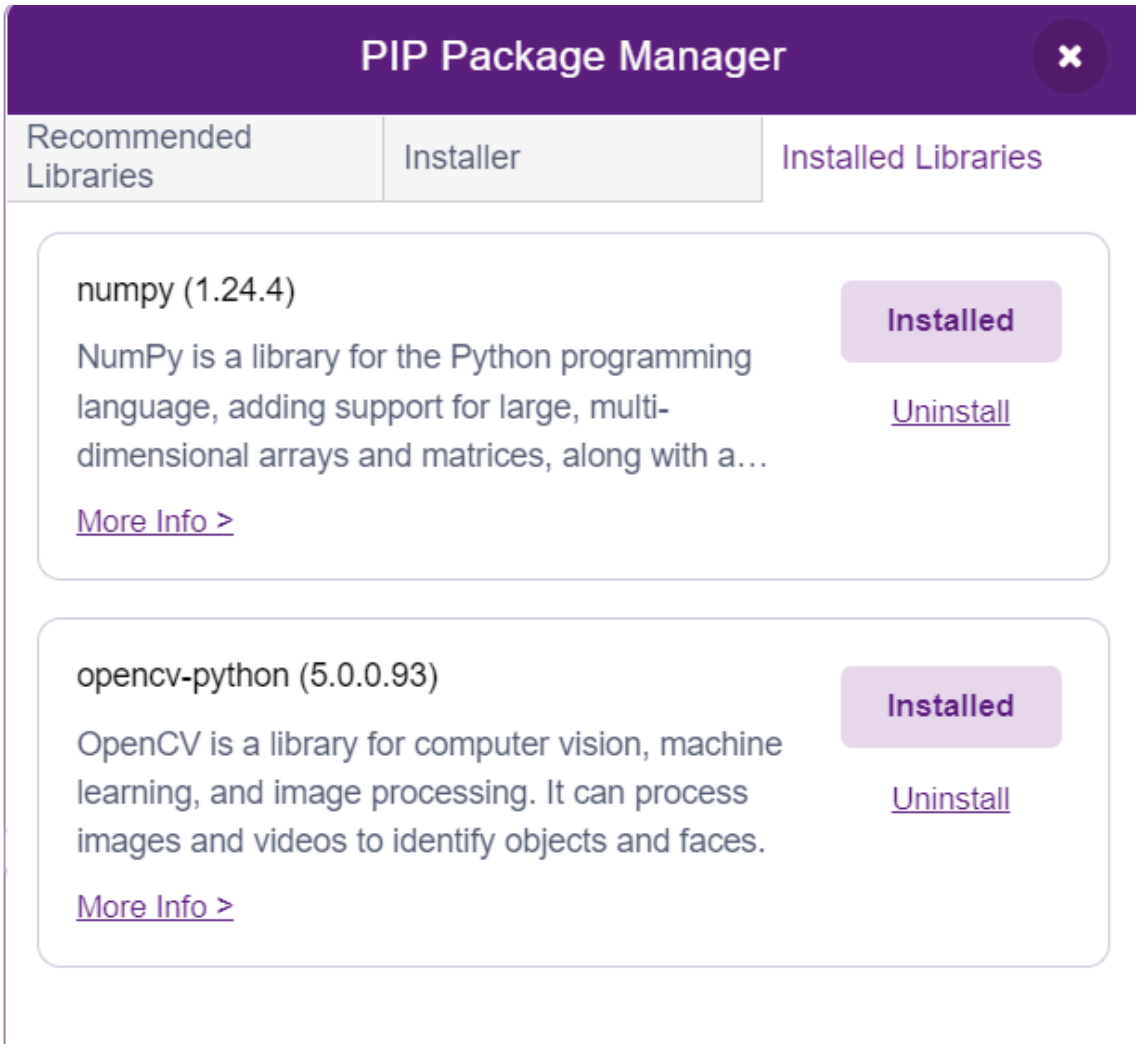
Pressing **q** stops the main loop. The program then stops the rover, stops the video receiver, and closes all OpenCV windows.

6. Common Mistakes to Avoid

- **If these errors come**

  `ModuleNotFoundError: No module named 'cv2'`

  `ModuleNotFoundError: No module named 'numpy'` -Make sure you have installed the following libraries from the Pip Package manager in PictoBlox: numpy and opencv.



- **Rover doesn't move at all**— check that the mini expansion board's power switch is turned ON and Intellio is charged.
- **Cones aren't being detected** — make sure the area has proper, even lighting; too dark a room or strong glare can throw off the red/blue colour detection.
- **Rover turns at the wrong time or in the wrong direction** — confirm the cones are placed correctly, with all red cones on the right and all blue cones on the left, and enough space between them.
- **No video feed or a laggy feed** — check that the computer and Intellio are on the same WiFi network, and that no firewall is blocking the UDP ports used for video and control.
- **If the rover starts moving in backward direction instead of forward-** Switch to block coding mode and under the mini expansion extension click on the block of “reverse motor direction”.

7. Tips for Best Results

- Keep the surrounding area free of other red or blue objects that could confuse the detection.
- Test with just two or three cones first, then build up to a full track once it's working reliably.
- Watch the Red Mask and Blue Mask windows while testing — if a cone doesn't show up clearly there, it won't be detected on the track either.