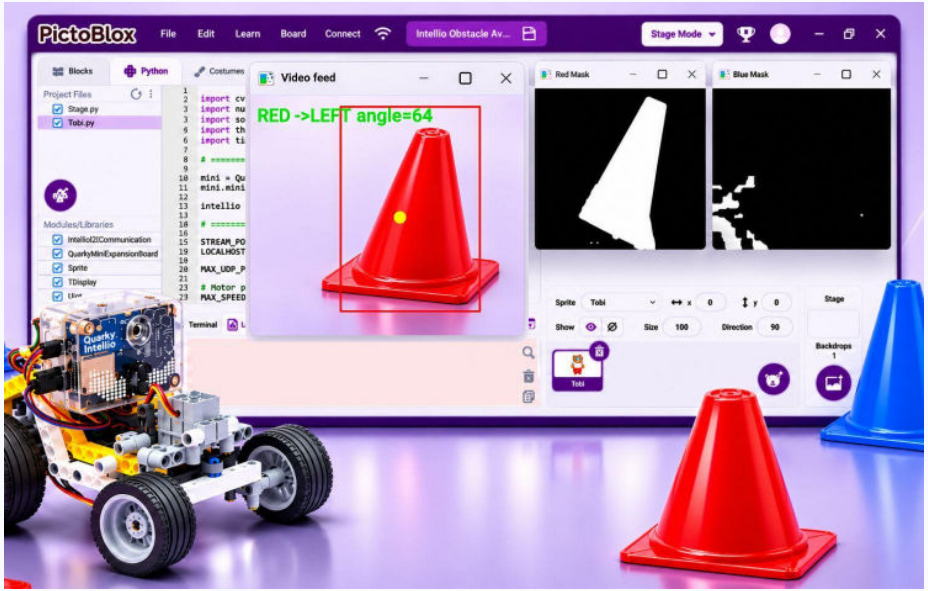
PYTHON & OPENCV

Quarky Intellio Obstacle Avoiding Robot

Autonomous visual navigation system built with Quarky Intellio, real-time color cone detection, and smart steering control.



🤖 ESP32-S3

🐍 Python 3

📷 OpenCV

Introduction

System Overview

The **Obstacle Avoider** is an autonomous computer-vision project for the Intellio rover, powered by PictoBlox. A Python script receives a live video feed from Intellio's camera over WiFi, detects red and blue cones, and transmits real-time steering commands—**turning left for red cones** and **right for blue cones** to navigate the field unassisted.

STEP 01



See

Live video stream from Intellio camera over WiFi.

STEP 02



Detect

Identify red & blue cones in the video frame.

STEP 03



Decide

Turn Left for red cones, Right for blue cones.

STEP 04



Steer

Send motor & steering commands in real time.

STEP 05



Continue

Rover navigates the cone field seamlessly.

What You Need To Run The Project

ITEM 01

Laptop / PC

A laptop or PC with PictoBlox installed.

ITEM 02

Intellio Rover

An Intellio rover hardware unit ready for operation.

ITEM 03

WiFi Network

A WiFi network connected to both the computer and Intellio.

ITEM 04

Track Cones

Red cones and blue cones (or red/blue coloured objects) for the track.

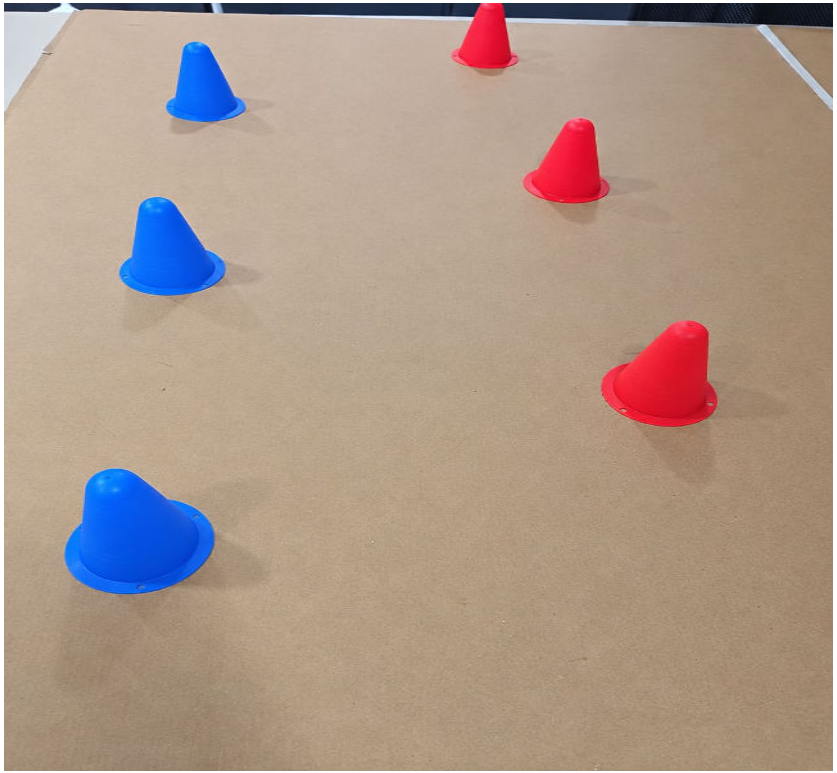
ITEM 05

Open Space

A large, open and well-lit area where the rover can move freely.

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.



Using Mini Expansion and Servo in PictoBlox Python

```
mini=QuarkyMiniExpansion()
mini.miniinitialise()
mini.runmotor('L','1',100)
mini.stopmotor('L')
mini.runbothmotors('1','1',100,100)
mini.stoprobot()
mini.runrobot('2',100)
mini.reversemotordirection('L')
mini.runmotorfortime('L','1',100,1)

intellio=QuarkyIntellio()
intellio.setservo('S1',45)
```

Basic Understanding of OpenCV & NumPy

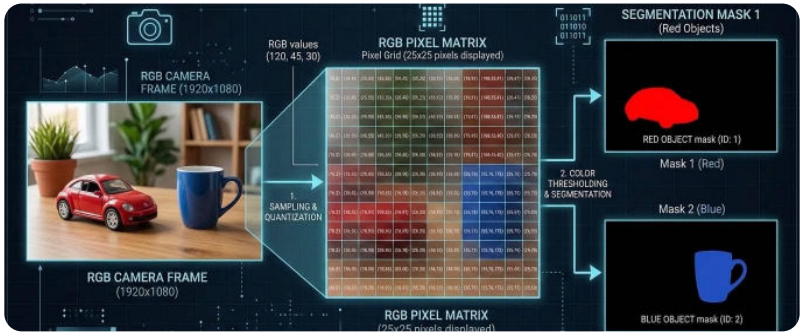
What is OpenCV & Why Use It?

- **What it is:** An open-source computer vision library used for real-time image and video processing.
- **Why it's needed:** Allows rovers/robots to "see" and interpret visual data to detect objects, colors, and track obstacles automatically.

How It Works with NumPy

- **Images as Numerical Arrays:** OpenCV converts camera frames directly into multi-dimensional NumPy arrays (Height × Width × RGB channels).
- **Matrix Analysis:** Performs high-speed mathematical operations on pixel matrices to isolate specific colors (e.g. Red/Blue cones).

Image Processing & Analysis Example



1. **Camera Input:** Live video frame captured
2. **NumPy Matrix:** Converted to pixel grid numbers
3. **Color Threshold:** Filters red/blue color channels
4. **Action Output:** Triggers steer command (Left/Right)

Understanding The Code: Core Architecture

01. Video Receiver Thread

- Runs UDP video stream listener concurrently on port 6000.
- Buffers incoming packet chunks to reconstruct complete image frames.
- Uses OpenCV to convert bytes to standard RGB frames.

```
class VideoReceiver:
  def receive_loop(self):
    ...imdecode(...)
```

02. HSV Color Filtering

- Converts camera frames to HSV color space for stable object detection.
- Isolates **Red** and **Blue** track cones using dynamic threshold masks.
- Applies morphological operations (open/close) to suppress image noise.

```
get red mask(frame)
get_blue_mask(frame)
```

03. Steering & Control Loop

- Finds largest object contours & computes horizontal center offsets (cx).
- Calculates non-linear steering angle curves for smooth rover turns.
- Maintains state persistence (0.8s) so steering continues after obstacle leaves frame.

```
calc servo angle(...)
intellio.write("S1", angle)
```

Understanding The Code: Advanced Logic

04. Speed & Differential Control

- Dynamically adjusts left/right motor PWM speeds based on steering sharpness.
- Reduces base velocity during high-angle turns to maintain rover stability.
- Enforces minimum threshold power to overcome stall torque on rough surfaces.

```
calculate_motor_speeds(angle)
intellio.set_motors(left, right)
```

05. Fail-Safe & Timeout System

- Monitors video frame rate and network packet latency continuously.
- Triggers automatic emergency stop if video stream drops for >1.5 seconds.
- Resets steering angle to neutral position when no track cones are detected.

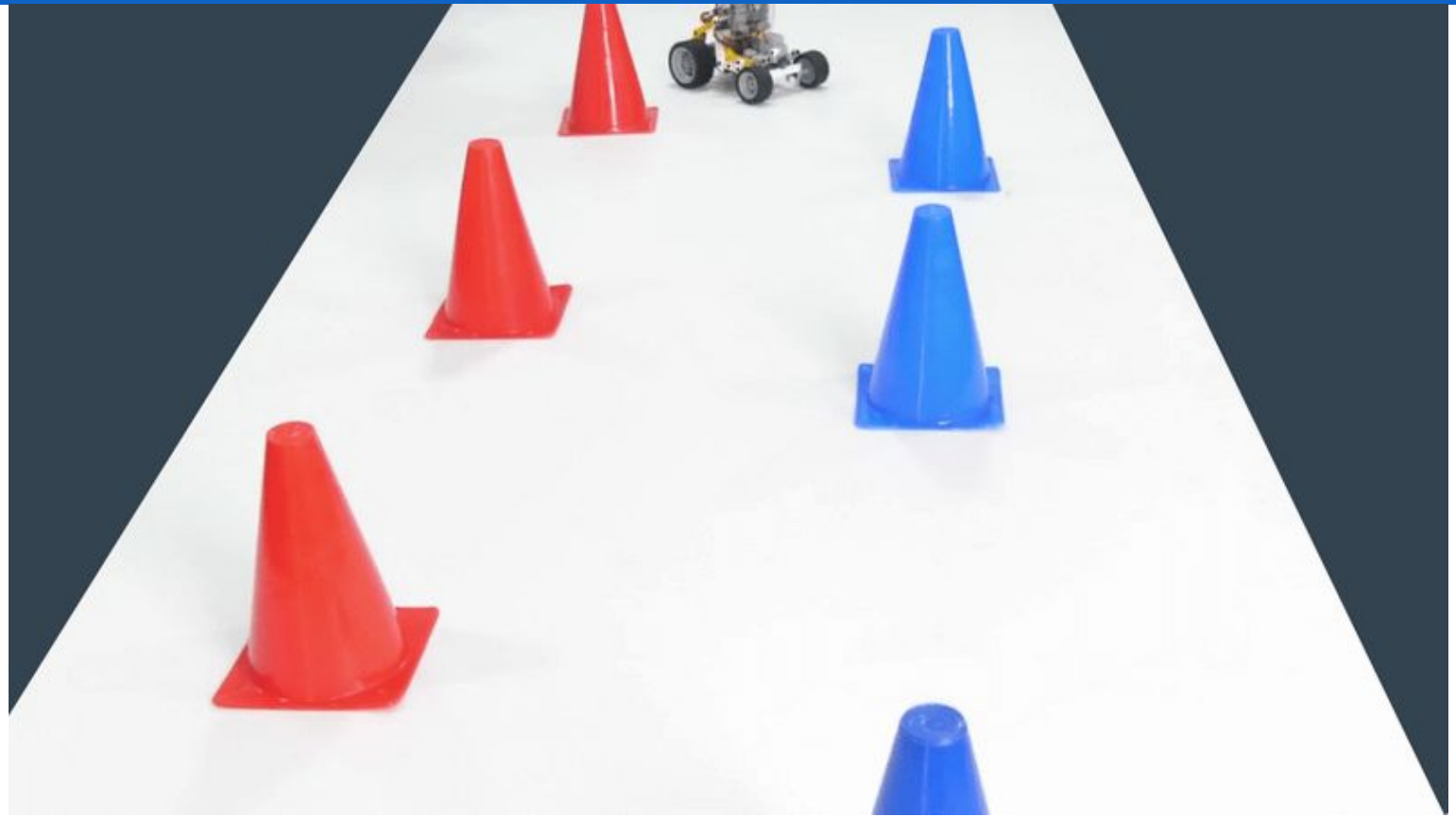
```
check_connection_timeout()
intellio.emergency_stop()
```

06. Telemetry & Debug Overlay

- Draws target centroids, bounding boxes, and error vectors on live feed.
- Logs real-time steering commands and processing FPS to PictoBlox terminal.
- Provides interactive dynamic HSV threshold sliders for live calibration.

```
cv2.putText(frame, telemetry)
draw_debug_overlays(frame)
```

Project Demonstration Video



Common Mistakes And Fixes

01. Rover doesn't move at all

check that the mini expansion board's power switch is turned ON and Intellio is charged.

02. 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.

03. 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. Also the camera feed must be coming in real-time and not be lagging.

04. Rover moves 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”.

05. 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.

06. `ModuleNotFoundError`

Make sure you have installed numpy and opencv libraries from the Pip Package manager in PictoBlox.

Tips for Best Results

01. Lighting & Calibration

- **Tip:** Calibrate HSV color thresholds in the exact room lighting before starting runs.
- **Tip:** Use non-reflective track surfaces to prevent false color detection.

02. Power & Stability

- **Tip:** Maintain fully charged batteries for consistent motor PWM output.
- **Tip:** Firmly secure the camera frame to eliminate mechanical vibration jitter.

03. Control

- **Tip:** Enable state persistence to handle brief obstacle occlusions smoothly.

04. Area & Environment

- **Tip:** Keep the surrounding area free of other red or blue objects that could confuse the detection.

05. Mask & Window Testing

- **Tip:** 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.

Prerequisites before running the code

01 An assembled Intellio rover.

02 Ensure the Intellio is fully charged.

03 At least 2 red objects and 2 blue objects to be used as obstacles.

04 PictoBlox updated to version **10.0.0 and above**

05 Intellio's Firmware updated to version **2.0.0 and above**

06 Opencv and Numpy library installed in PictoBlox

All set? Let's get rolling!

READY, SET, ROVE!

Now it's your turn to build, code & test the Intellio Obstacle Avoider.

Reference Documentations:

Intellio Obstacle Avoider Documentation