Green ball detection

Petar Stamenkovic — BioSens Institute

July 2025

1 Introduction

This introductory project features a detection and tracking of a green ball via live stream of a camera. It uses HSV transformation and contour detection using OpenCv and DepthAI python libraries. When it comes to Hardware, OAK D Lite camera by Luxnois is used for video streaming. And yes, you will need a green ball to detect it.

Code is written entirely using a Python programming language and version 3 (**important**) of the DepthAI library.

2 What will you need?

- **OAK D Lite Camera**
- **USB C cable**
- **Tripod** - Optional
- **Ubuntu OS** - Preferable
- **Visual Studio Code** - Preferable to have a good text editor
- **Python libraries** - DepthAI Version 3, Matplotlib, OpenCV, Numpy

Figure 1: OAK D Lite setup

3 Code analysis

In this section, a Python code file is covered from start to end, with additional explanations. Code is written in Visual Studio Code and it is ran from its terminal.

3.1 Importing libraries

As always, we begin by importing all the necessary libraries. We use mentioned OpenCV to draw out the frame and detected objects region of interest (ROI) on the host computer. Collections library is used to create buffer for spatial coordinates to reduce the jittering introduced by a stereo camera sensitivity. Finally, we use DepthAI for creating a pipeline for the OAK Camera and connect required nodes.

3.2 Pipeline

OAK Camera setup uses nodes that represent certain functionalities and pipelines to connect them in a specified order. In this code we use the following nodes.

- **Camera node** - We create one RGB camera *CAM-A* for an actual image preview, and two mono cameras that will link to Stereo Node in order to calculate depth (*CAM-B and CAM-C*). All camera resolutions are 640/480 with 30 fps on the RGB camera.
- **Stereo Depth node** - Calculates Depth from the input of two mono cameras and provides information to the *inputDepth* port of the Spatial Location Calculator node.
- **Spatial Calculator Node** - Calculates and provides spatial coordinate values and lets us print them on the final frame.

After this, we use method *createOutputQueues* and method *createInputQueues* to prepare necessary values for processing in the main loop. Pipeline is started.

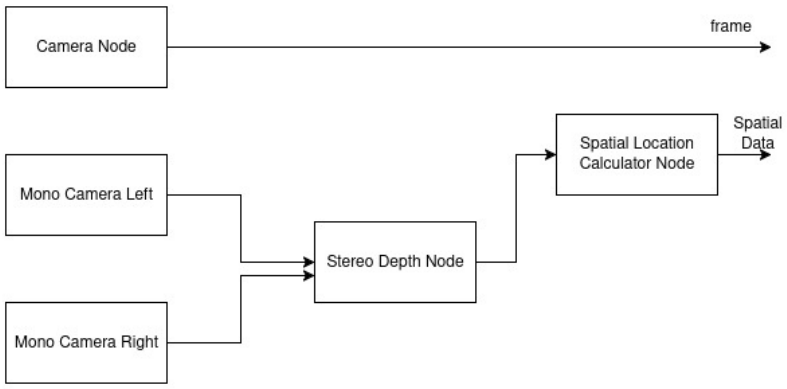

Figure 2: Pipeline sketch

3.3 Main while loop for processing

This loop processes the information we received from our hardware and draws it out on the host computer. The idea is to fetch a frame from the RGB camera in a variable called *frame* and to transform it into a HSV format with a mask for green color. Here is the first part of the code where we can adjust some parameters in terms of the intensity of a color. Of course, this can now be changed to *any* color for detection. HSV short for Hue, Saturation and Value is used to separate **color** and **brightness**, since same color can appear different under a different lightning, in terms of its RGB values. It practically separates color and lightning within reason of course and it is a standard when it comes to a color detection methods.

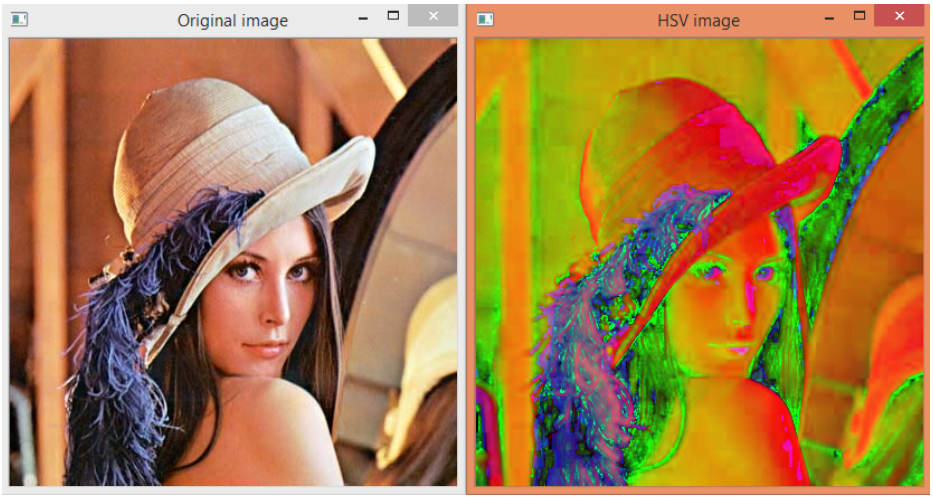

Figure 3: HSV format conversion

After detecting a color, we also need to detect the circular shape. For this we use OpenCV method *findContours* with the defined mask (*green*). This finds the outlines for green objects on the image. We find the biggest contour that fits the filters. We fetch parameters for the circle via *minEnclosingCircle* and use another parameter (*circularity*) to adjust the "perfection" of the circle.

Now that a circle is found we calculate region of interest for it. With variables x1, x2, y1 and y2 we draw ROI out on the frame we see, and for the Spatial Location Calculator we need to normalize the values before sending them. ROI is configured, depth thresholds are set (*how close and how far can camera detect an object*) and configuration is sent to a Spatial Location Calculator node.

Spatial data is now fetched from the output of the Spatial Location Calculator node and printed out on the frame, for us to see them. Also, horizontal and vertical lines are drawn on the frame just to check the quality of spatial data.

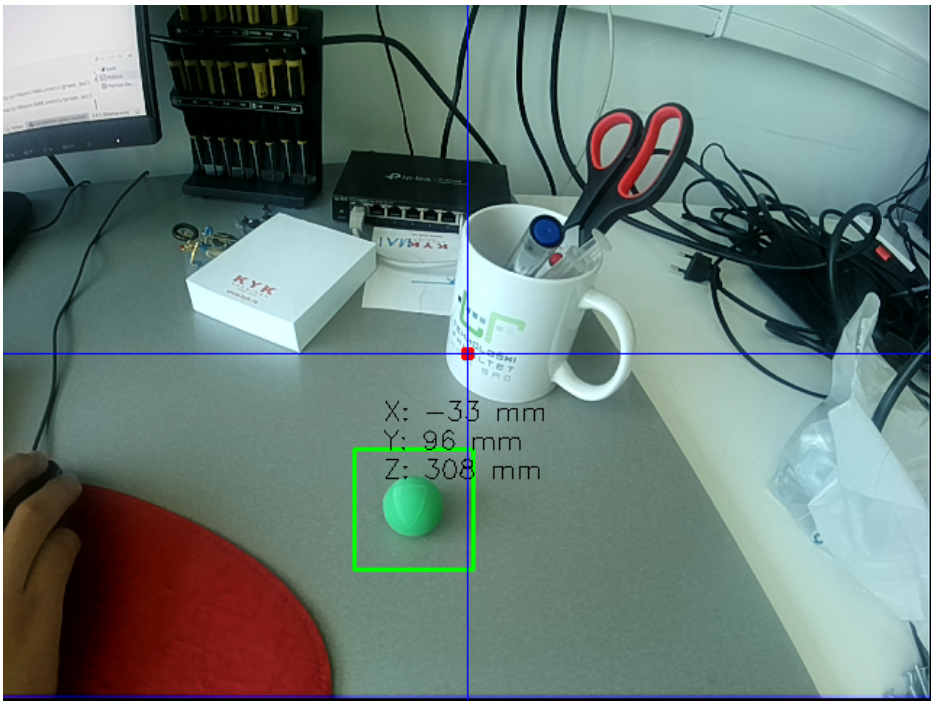

Figure 4: Detected green ball

Note that y axis value gets bigger as we go down, this is slightingly different from the actual coordinate system used in simple math. To exit the frame and camera steam press 'q'.

4 Possible improvements

- **Multiple object detection** - Currently, as mentioned, we detect one objects spatial coordinates. This can be improved to detecting and tracking multiple green balls on the frame.
- **Spatial coordinates jitter** - Due to a nature of the Stereo and depth calculations, spatial coordinates can change quite fast, especially in the edge of cameras vision. I am not sure how to fix that, ball detection works fine close to a center of a cameras eyes.
- **Z axis** - The most problematic axis is Z, the depth metric. It works in exclusive positions right, but as soon as object gets to close or far, values are inaccurate and start changing chaotically. To be discussed with mentors.
- **Light and odd angles** - Light conditions affect the quality of detection by a lot, even though HSV conversion is used. More research on this to be done...

This task is still in progress, but all helpful information are welcome. Send me an email petastamenkovic35@gmail.com

Cheers!