OPENCV

Real-time distance calculation using single camera

- Bhavya Sheth
- Manas Thorat

TABLE OF CONTENTS

1. PROBLEM STATEMENT 3
2. REQUIREMENTS 3
3. SOLUTION 4
4. IMPLEMENTATION OF PROJECT 8
5. OUTPUT 9
6. FUTURE PROSPECTS 10
7. REFERENCES 10
8. LINK TO CODE 10

PROBLEM STATEMENT

To find (xyz) coordinates of solid cube of given dimension in real world using OpenCV.

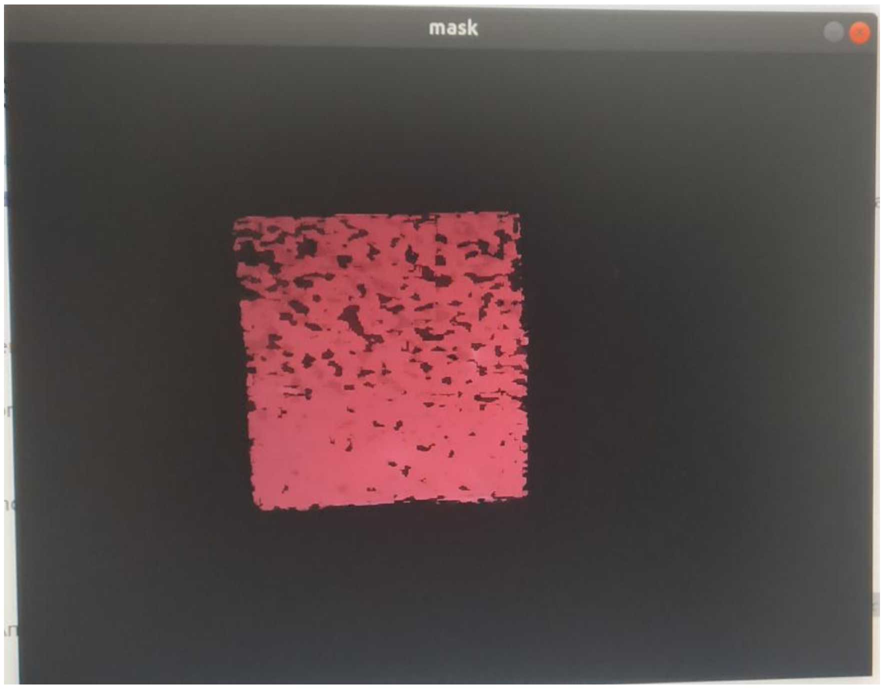
REQUIREMENTS

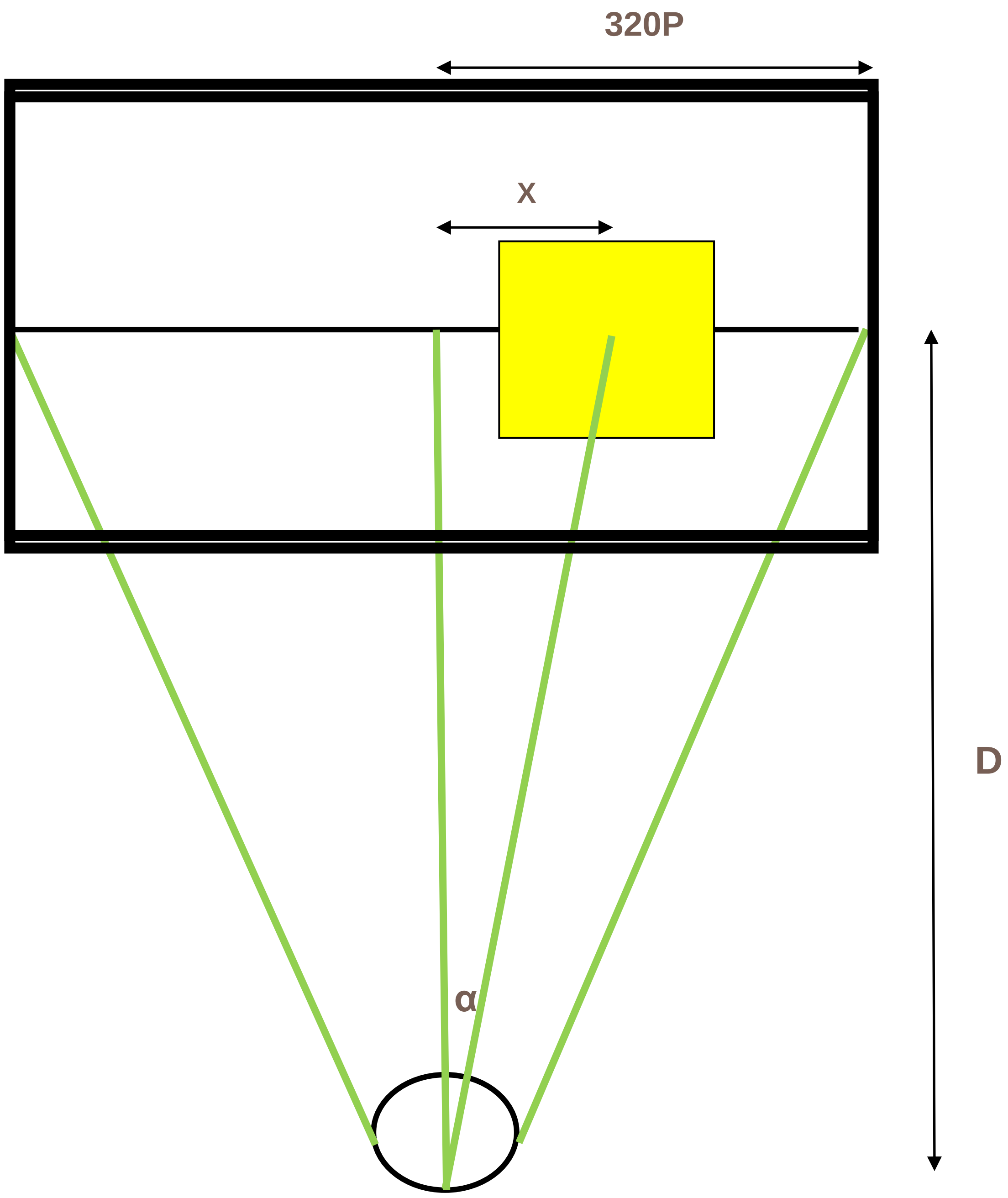
1. Python is installed on the device.
2. OpenCV and Numpy libraries are set up and imported.
3. Web cam (Logitech c710) [(640x480) resolution]
4. Solid cube of red color and dimensions (4.8x4.8x4.8)

SOLUTION

First task is to find a solid cube of specific color from the surrounding. So we are going to convert our colors to HSV, which is "Hue Saturation Value." This can help you actually pinpoint a more specific color, based on hue and saturation ranges, with a variance of value. Than we create a mask, which uses an "inRange" statement, for our specific range. This is true or false, black or white. Next, we "restore" our redish-ness by running a bitwise operation. Basically, we show color where there is the frame AND the mask. The white part of the mask will be red range that was converted to pure white, while everything else became black. Finally we show it all.

This is just an example, with red as the target. The way this works is what we see will be anything that is between our HSV ranges.





Where

D = distance between image and camera

α = field of view of camera

In our case field of view of camera is 45°. By using simple trigonometry

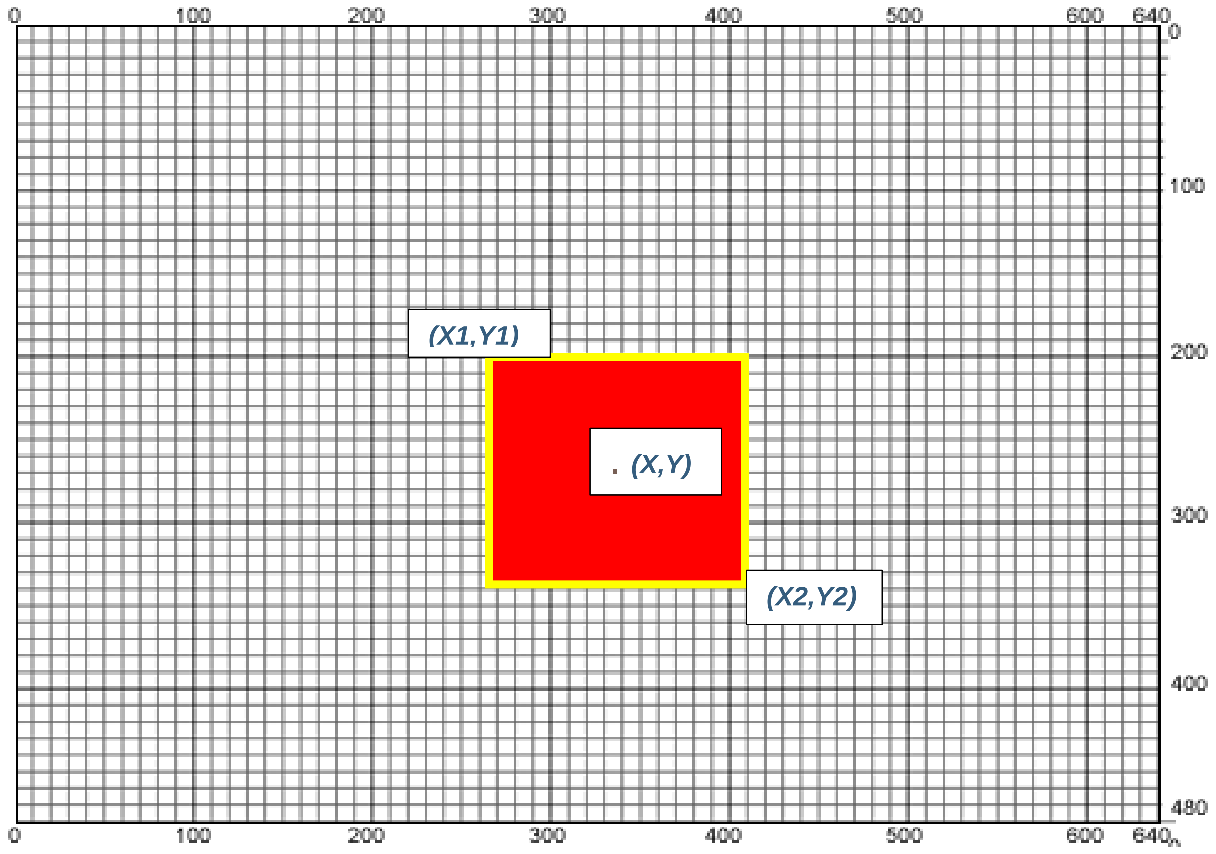
$$\tan(22.5) * D = 320P$$

$$D = 772.5483\ P \qquad \text{.........(1)}$$

Than we bound a rectangle around the coordinates using function

x1, y1, w, h = cv2.boundingRect(contour)

Now we have the (x,y) coordinates of the square on the webcam video but this are not the actual world coordinates. You can assume it as a coordinate of pixel in (640x480) resolution image.



Now consider that above image shows our red cube on a screen with (x1,y1) and (x2,y2) as respective coordinate. The image has 640x480 resolution and each small unit of image is pixel. Y1-Y2 gives us the length of the cube in terms of pixel. But we know that in real world this length resembles to 4.8 cm (given) so we can make a relation

(Y2-Y1) P = 4.8 cm ……(2)

By solving equation 1 and 2 we get

D = 772.5483 x (4.8/(Y2-Y1))
D = 3708.23184 / (Y2-Y1) ……..(3)

FINDING X AND Y COORDINATES IN REAL WORLD

We have to calculate position of the object by taking center of camera as origin. By shifting the origin the new coordinates become *(X-320, Y-240)P.*

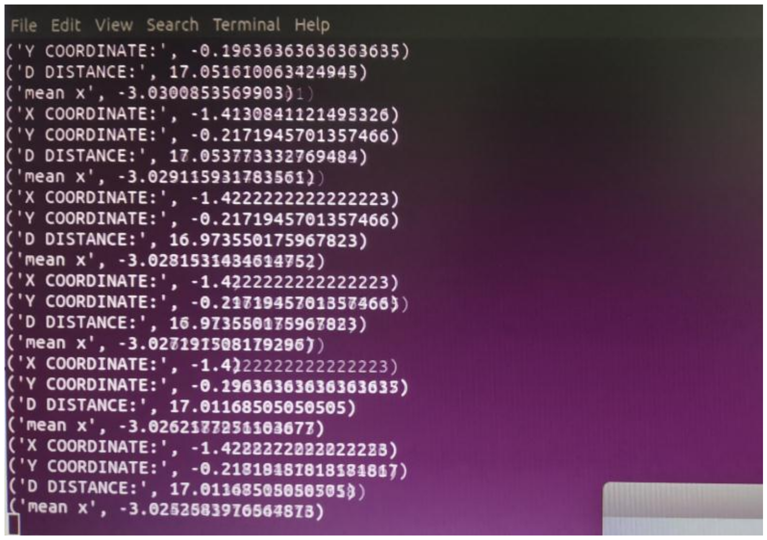
X Coordinate = (x-320) P
= (X-320) * (P/(Y2-Y1))

Similarly Y coordinate is calculated.

IMPLEMENTATION OF PROJECT

5. Python is installed on the device.
6. OpenCV and Numpy libraries are set up and imported.
7. Create mask to detect block using the built-in InRange() function in OpenCV using HSV colour values for red colour.
8. Unwanted noise in video feed is eliminated using morphological transformations (erosion and dilation).
9. Contours around the blocks are drawn using mask created in step 3. A bounding rectangle is drawn in cases of large contours, giving us the outline of the block.
10. Centroid of the outline is found out.
11. The perpendicular distance is calculated along with x,y coordinates using trigonometry.

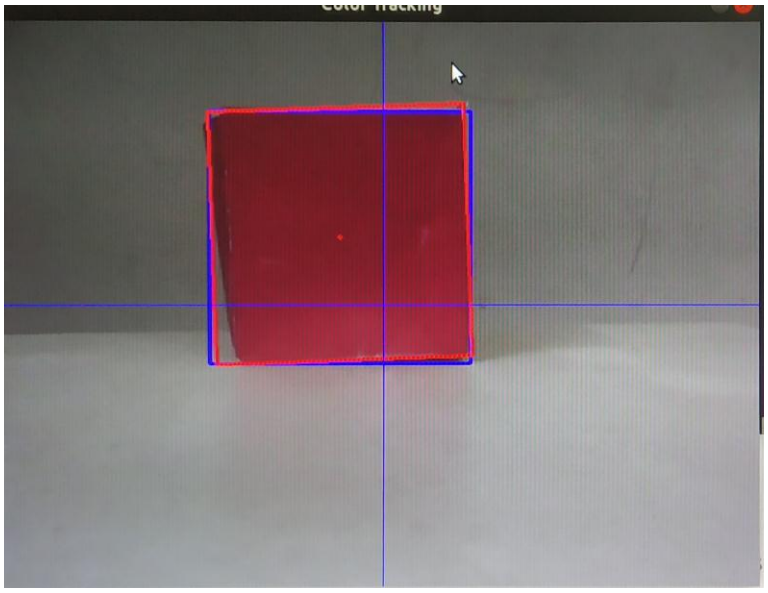
OUTPUT



```
('Y COORDINATE:', -0.19636363636363635)
('D DISTANCE:', 17.051610063424945)
('mean x', -3.0300853569903)
('X COORDINATE:', -1.4130841121495326)
('Y COORDINATE:', -0.2171945701357466)
('D DISTANCE:', 17.053773332769484)
('mean x', -3.029115931783561)
('X COORDINATE:', -1.4222222222222223)
('Y COORDINATE:', -0.2171945701357466)
('D DISTANCE:', 16.973550175967823)
('mean x', -3.028153143461475 2)
('X COORDINATE:', -1.4222222222222223)
('Y COORDINATE:', -0.2171945701357466)
('D DISTANCE:', 16.973550175967823)
('mean x', -3.027191508179296)
('X COORDINATE:', -1.4222222222222223)
('Y COORDINATE:', -0.19636363636363635)
('D DISTANCE:', 17.01168505050505)
('mean x', -3.026217725110367 7)
('X COORDINATE:', -1.4222222222222223)
('Y COORDINATE:', -0.2181818181818181 7)
('D DISTANCE:', 17.01168505050505)
('mean x', -3.025258397656487 3)
```

The output is given as the x and y coordinates of the centroid, along with The perpendicular distance D. Mean x is also given to reduce fluctuations, however due to a few outlier values it is often inaccurate.

The contour around the block is detected as follows:



FUTURE PROSPECTS

This setup sometimes is error-prone in calculating distances because unwanted contours are detected and hence this leads to wrong d and x,y calculations. This can be tackled by formulating a graph of the change in height and width with respect to distance from camera taking into account camera parameters.

Moreover, this setup is largely dependent upon the lighting conditions. It needs sufficient lighting to detect the red colour effectively and without errors. This can be tackled by introducing object detection using Haar cascades. Different images in different lighting conditions can be fed into the cascade to improve object detection.

REFERENCES

12. https://docs.OpenCV.org/3.1.0/d6/d00/tutorial_py_root.html

13. https://www.pyimagesearch.com/2014/08/04/OpenCV-python-color-

14. https://www.pyimagesearch.com/2014/04/21/building-pokedex-python-finding-game-boy-screen-step-4-6/

LINK TO CODE

https://github.com/Bhavya-sheth/OpenCV-Eklavya