Distance Measurement Using OpenCV - Python

MAJOR PROJECT

Submitted in partial fulfilment for the award of Degree of

BACHELOR OF TECHNOLOGY

in

Computer Science and Engineering

to

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal, M.P.

Submitted by

Shubham Dhangar (0537CS191079)

Shubham Thakur (0537CS191080)

Vijay Verma (0537CS191088)

Under the Supervision of

Prof. Ajit Shrivastava

Department of Computer Science and Engineering

SAGAR INSTITUTE OF SCIENCE, TECHNOLOGY & RESEARCH BHOPAL, M.P.

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the work embodies in this major project entitled " **Distance Measurement Using OpenCV - Python***" being submitted* ***by Shubham Dhangar (0537CS191079),Shubham Thakur (0537CS191080), Vijay Verma (0537CS191088)*** *for partial fulfilment of the requirement for the award of „****Bachelor of Technology*** *in* ***Computer Science and Engineering'*** *discipline to* ***Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (M.P.)*** *during the academic year* ***'june - Dec 2022'*** *is a record of bona-fide piece of work, undertaken by him in the supervision of the undersigned.*

Supervised By:	**Approved By:**	**Forwarded By:**
Prof. Ajit Kumar Shrivastava	Prof. Ajit Kumar Shrivastava	Dr. Manish Billore
A.P, CSE	Head of Dept., CSE	Principal
SISTec-R, Bhopal	SISTec-R, Bhopal	SISTec-R, Bhopal

Introduction

Distance measurement between a robot and the object is needed to control the action of the robot such as grabbing an object or even avoiding obstacles. There are many methods to estimate the distance such as ultrasonic ranging, laser ranging, and vision based ranging. Vision based techniques have the merit of its low cost, so in this project we will learn just a method for distance measurement between a single camera and the objects in front of it and implement it.

The challenge is to use a depth map and retrieve features from it to predict the distance of an object in a camera image. There will be discussion on other features that were discovered when investigating. Also included is the use of an object recognition mechanism, which aids in the detection and localization of objects in images.

Objectives

1. To obtain a system which can recognise objects and gives an accurate distance of that from the camera.
2. To make this project cost efficient without using some costly hardwares and some other complex machine learning algorithms.
3. To get solutions of real world machine learning problems.

Scope

1. Machine learning is one of the hottest topics nowadays, the demand of machine learning is increasing day by day and can be seen at its peak in the upcoming 2 to 4 years .
2. Computer Vision has its own scopes in the field of machine learning. Object detection, recognising shapes, patterns and edges around the objects are some features of computer vision.
3. OpenCV was built to provide an infrastructure for computer vision. This library has a huge range of optimized machine learning and computer vision algorithms, learning this technology provides scopes in this field.
4. Distance measurement has many uses in current technologies and will play an important role in upcoming technologies.

5. Self driving cars, Satellite images use object detection, social distance measurement applications are some popular use of distance measurement technology.

Description

The challenge of estimating the distance of objects from a camera remains a hot research subject in the field of computer vision, and it has several applications. In order to go on, it is necessary to understand a few words. Range calculation that is both reliable and accurate remains a difficult challenge in computer vision. Few methodologies exist for this, but they are very complex, and sensors and other heavy forms of machinery are used to measure the distance of objects. Understanding the space between two points is critical in robotics for avoiding collisions and picking up objects. Any sensor can do this, but they have a number of disadvantages. These constraints, however, can be solved by the use of image processing.

As a result, it was ultimately agreed to use image recognition to calculate the object's distance. The definition of depth picture, as well as object detection mechanisms, are heavily used to solve this issue. to build a self-contained artificial intelligence.

1. Mapping machine learning problem

Without the help of any sensors or any heavy hardware can it be possible for a robot or any model to have a rough idea about its vicinity . Robots need to learn spatial location of objects under their vicinity. The proposed mechanism will make use of depth images, object detection mechanisms to find the spatial graph of the environment. Or by looking at the image is it possible to estimate the distance of the object from the camera.This is a Regression problem in Machine Learning, where the task is to predict a real-valued output

(distance) of an object with respect to the camera present in the image or vicinity of the camera. The predicted value should be greater than zero.

Dataset Description

A dataset is required before proceeding with any machine learning or deep learning problem. This chapter would include a brief overview of all current datasets or all steps taken into consideration when constructing a dataset from scratch. Until continuing, it is important to go over the analysis problem statement and motivation. Using object detection methodology and depth images, the goal is to construct a machine learning methodology that can classify the object present in the image and estimate the distance from the camera, that is, what will be the real distance from the camera to the object when those were clicked. There was no prior work and literature to address the issue of distance estimating using machine learning. As a result, the scarcity of jobs and study in this area motivates one to try to tackle the problem from the ground up.

Requirements

Requirements are pretty simple you need Python, OpenCV, and Haar-cascade files for Face Detection. Having python installed on your machine, just open the terminal, and paste the following command in the terminal and you are done, Installation.

```
pip install opencv-python  # for windows 10
```

Capture the Reference Image

The reference image, allows us to map the real world(object plane) since we lose the depth of the object when we capture it in 2D space(image), otherwise, we need a special camera, that can capture the depth since we want to use our webcam, the reference image is required, you need to take care of few things, so accuracy won't drop, you can use mine reference image it won't affect much, but I will recommend to capture it yourself to get more accuracy.
You have to take care of some things which are pointed below.

- Resolution of image (frame) must be the same, as in reference image I have kept to defaults of OpenCV which is(640, 480)

- Keep the camera straight as possible while capturing the reference images.
- Measure distance(**KNOWN_DISTANCE**)from object camera, note it down and capture the image which is set to 76.2 centimetres.

Software and Hardware Requirements

1)	Development Environment	Visual Studio.Net 2008, Google Colab
2)	Technology Used	OpenCV, Visual Studio Code
3)	Language	Python
4)	Platform	Windows 10
5)	Documentation	Google Document

Hardware Used at Development Side

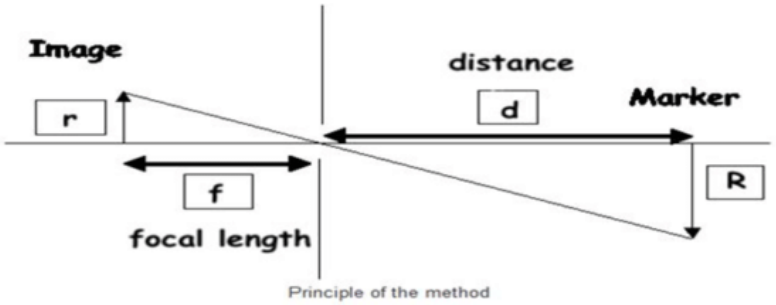
1)	RAM	4GB
2)	Hard Disk	1TB
3)	Processor	Intel Core i3-7020U 2.30 GHZ

Software Design

Triangle Similarity for Object/Marker to Camera Distance

In order to determine the distance from our camera to a known object or marker, we are going to utilize *triangle similarity*.

The triangle similarity goes something like this: Let's say we have a marker or object with a known width *W*. We then place this marker some distance *d* from our camera. We take a picture of our object using our camera and then measure the apparent width in pixels *P*. This allows us to derive the perceived focal length *F* of our camera:



Principle of the method

Using the principle of Similar Triangles, we can obtain the formulas as follows:

$$\frac{f}{d} = \frac{r}{R} \quad (1)$$

$$f = d \times \frac{r}{R} \text{ pixels} \quad (2)$$

$$d = f \times \frac{R}{r} \text{ cm} \quad (3)$$

F = (P x d) / W

For example, let's say I place a standard piece of *8.5 x 11in* piece of paper (horizontally; *W = 11*) *d = 24 inches* in front of my camera and take a photo. When I measure the width of the piece of paper in the image, I notice that the perceived width of the paper is *P = 248 pixels*.

My focal length *F* is then:

F = (248px x 24in) / 11in = 543.45.

As I continue to move my camera both closer and farther away from the object/marker.

I can apply the triangle similarity to determine the distance of the object to the camera:

D = (W x F) / P

Again, to make this more concrete, let's say I move my camera *3 ft* (or 36 inches) away from my marker and take a photo of the same piece of paper. Through automatic image processing I am able to determine that the perceived width of the piece of paper is now *170 pixels*. Plugging this into the equation we now get:

D = (11in x 543.45) / 170 = 35in

Or roughly 36 inches, which is 3 feet.

Note: When I captured the photos for this example my tape measure had a bit of slack in it and thus the results are off by roughly 1 inch. Furthermore, I also captured the photos hastily and not 100% on top of the foot markers on the tape measure, which added to the 1 inch error. That all said, the triangle similarity still holds and you can use this method to compute the distance from an object or marker to your camera quite easily.

Some function of code work for the implementation are : -

Focal Length Finder:

The Focal Length finder Function takes Three Arguments:

1. Measured_distance: It is the distance from the camera to object while capturing the Reference image, Known_distance = 72.2 #centimeter
2. Real_width: Its measure the width of an object in real-world, here we measure the width of the face which is around Known_width =14.3 #centimeter
3. Width_in_rf_image: It is the width of the object in the image/frame it will be in pixels
4. This function will return the focal length, which is used to find the distance.

Distance Finder :

This function has three arguments.

1. Focal length in pixel, which is a return from the Focal length finder function
2. Real_width measures the width of an object in real-world, here we measure the width of the face which is around Known_width =14.3 #centimeter
3. Width_in_rf_image is the width of the object in the image/frame it will be in pixels

The distance finder function will return the distance in the centimeters

Here is the full implementation code of this project : -

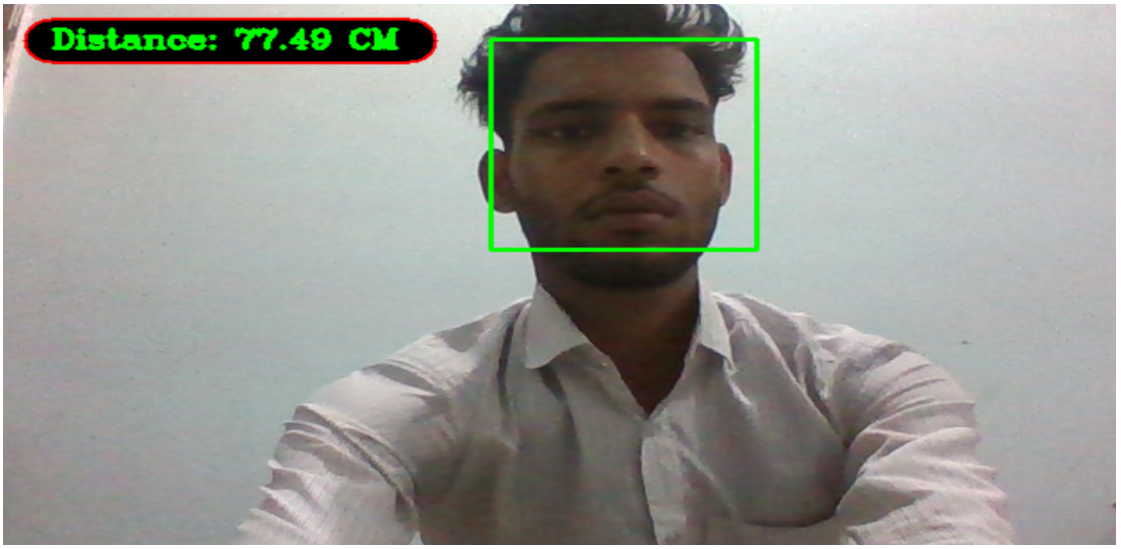
```
# install opencv "pip install opencv-python"
import cv2
import numpy as np

# distance from camera to object(face) measured
# centimeter
Known_distance = 50.2

# width of face in the real world or Object Plane
# centimeter
Known_width = 14.3

# Colors
GREEN = (0, 255, 0)
RED = (0, 0, 255)
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)

# defining the fonts
fonts = cv2.FONT_HERSHEY_COMPLEX

# face detector object
face_detector =
cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
```

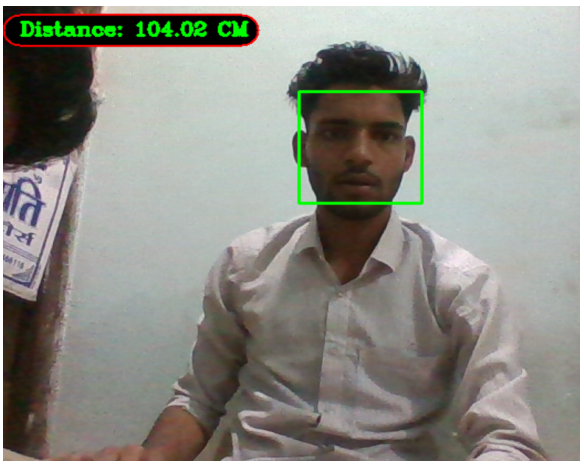
```
# focal length finder function
def Focal_Length_Finder(measured_distance, real_width,
width_in_rf_image):

    # finding the focal length
    focal_length = (width_in_rf_image * measured_distance) / real_width
    return focal_length

# distance estimation function
def Distance_finder(Focal_Length, real_face_width,face_width_in_frame):

    distance = (real_face_width * Focal_Length)/face_width_in_frame

    # return the distance
    return distance


def face_data(image):

    face_width = 0 # making face width to zero

    # converting color image ot gray scale image
    gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

    # detecting face in the image
    faces = face_detector.detectMultiScale(gray_image, 1.3, 5)

    # looping through the faces detect in the image
    # getting coordinates x, y , width and height
    for (x, y, h, w) in faces:

        # draw the rectangle on the face
        cv2.rectangle(image, (x, y), (x+w, y+h), GREEN, 2)

        # getting face width in the pixels
        face_width = w

    # return the face width in pixel
    return face_width


# reading reference_image from directory
ref_image = cv2.imread("n.png")
```

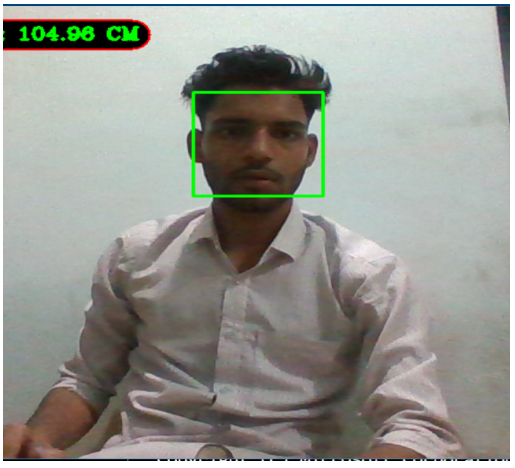
```
# find the face width(pixels) in the reference_image
ref_image_face_width = face_data(ref_image)

# get the focal by calling "Focal_Length_Finder"
# face width in reference(pixels),
# Known_distance(centimeters),
# known_width(centimeters)
Focal_length_found = Focal_Length_Finder(
    Known_distance, Known_width, ref_image_face_width)

print(Focal_length_found)

# show the reference image
cv2.imshow("ref_image", ref_image)

# initialize the camera object so that we
# can get frame from it
cap = cv2.VideoCapture(0)

# looping through frame, incoming from
# camera/video
while True:

    # reading the frame from camera
    _,frame = cap.read()

    # calling face_data function to find
    # the width of face(pixels) in the frame
    face_width_in_frame = face_data(frame)

    # check if the face is zero then not
    # find the distance
    if face_width_in_frame != 0:

        # finding the distance by calling function
        # Distance distance finder function need
        # these arguments the Focal_Length,
        # Known_width(centimeters),
        # and Known_distance(centimeters)
        Distance = Distance_finder(
            Focal_length_found, Known_width, face_width_in_frame)
```

```
        # draw line as background of text
        cv2.line(frame, (30, 30), (230, 30), RED, 32)
        cv2.line(frame, (30, 30), (230, 30), BLACK, 28)

        # Drawing Text on the screen
        cv2.putText(
            frame, f"Distance: {round(Distance,2)} CM", (30, 35),
        fonts, 0.6, GREEN, 2)

    # show the frame on the screen
    cv2.imshow("frame", frame)

    # quit the program if you press 'q' on keyboard
    if cv2.waitKey(1) == ord("q"):
        break

# closing the camera
cap.release()

# closing the the windows that are opened
cv2.destroyAllWindows()
```

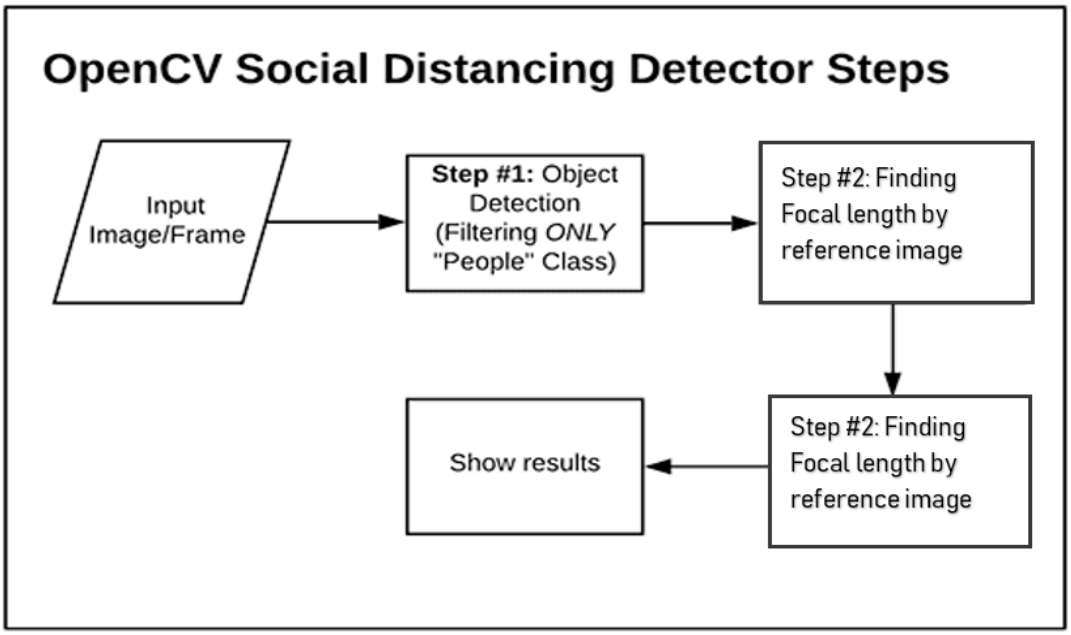
Snapshots



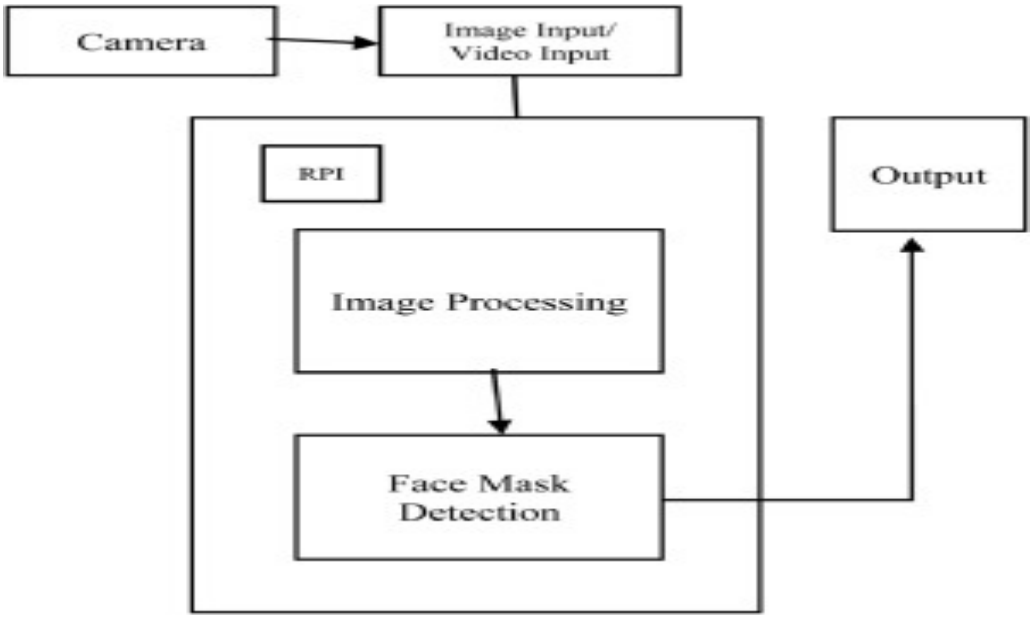




ER - Diagram



Block Diagram :–



Limitations

As the software is our first step towards making a robot which can detect objects and go to hold and carry an object, so it has some drawbacks also. It may have some less accuracy than an idle system.

Following are the few limitations associated with it as:

Using only one camera may have some less accuracy.
It may detect similar things as real one and can give false output in some cases.
Knowledge of mathematics needed as formula for finding focal length of camera used.
Accurate values required at the time of training of the project, any minor error can give large error in the output.

Further Enhancements

Can add new functionalities, checking distance between objects and can get distance of more than one object at time.

It will help in making robots which can clean window glass dust on tables and shoes.

We can make it more accurate by using dual cameras or stereo cameras.

Making it more accurate can help robots in grabbing things, holding and carrying from one place to another.

Completion of the project

Time duration - About 4 month in 7th semester .

In contribution with

Project members – Shubham Dhangar (0537CS191079)

Vijay Verma (0536CS191088)

Shubham Thakur (0537CS191080)

In Guidance of - Prof. Ajit Kumar Shrivastava (HOD).

Bibliography

www.W3Schools.com (for solving issues related to code)

www.kaggle.com (for getting datasets)

www.stackoverflow.com (for general questions)