**TRF Summer internship and training course2020Task Report**

**INDEX**

Group No. : 17

Project Title: **Distance** **Calculator for bots, using Camera.**

**Members**

Name: **Archit Jain.**

GR No.: 11911174

E-mail ID: [archit.jain19@vit.edu](mailto:archit.jain19@vit.edu)

Name: **Vaishnav Sonawane.**

GR No.: 11910585

E-mail ID: [vaishnav.sonawane19@vit.edu](mailto:vaishnav.sonawane19@vit.edu)

Name: **Prithviraj Gotepati**l

GR No.: 11910959

E-mail ID: [prithviraj.gotepatil19@vit.edu](mailto:prithviraj.gotepatil19@vit.edu)

1. **INTRODUCTION (Abstract)**

Estimating distance is important in daily life for us and that too applies for the bots. A main task in robotic vision is to determine the spatial relations between **the robot and the objects that surround it**. Usually the robot needs to know the distance to certain objects in order to **localize, navigate or do some high-level planning**. To overcome this challenge we decided to use **camera of robot** to find the distance.

In this project we focussed on a very unique entity of cameras i.e. Focal length, which can be calculated by Triangle similarity method and then can be used for distance measurement.

This project has been efficiently coded using **Python and Opencv.**

2**. WORKING METHODOLOGY(Body)**

**Triangle similarity method :**

If 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:

F = (P x  D) / W

As we continue to move the camera both closer and farther away from the object/marker, we can apply the triangle similarity to determine the distance of the object to the camera, by using formula:

D’ = (W x F) / P

**Algorithm :**

* Importing packages
* Creating a function to **find out our object in the image** :
  + - * + convert the image to grayscale.
        + Creating high and low value RGB value arrays for masking purpose
        + Masking is applied on image using cv2.inrange()
        + Edge detection is applied on image using cv2.Canny()
        + Finding out the contours using updated image.
        + Returning the boundary box (x, y coordinates and height and width of box in pixels) of largest contour found.
* Creating a function to calculate the distance using triangle similarity method i.e. formulae which are mentioned above.
* It takes known width W, focal length F and width in pixels P of object at current position.
* And returns distance by D’ = (W x F) / P
* Storing the initial object distance and object width in variables.
* Loading the reference image and calling function to find object.
* Finding the focal length using F = (P x  D) / W
* Start to receive frames from video using cv2.VideoCapture(0) / or passing list of images
* Finding object from frame by previously defined function.
* Finding distance by calling the function to measure distance.
* Drawing a box around object in frame.
* Printing the distance calculated on the image output

We can also switch between two color objects by calling mouse click event cv2.setMouseCallback("image", click\_event) and can find the distance between multiple objects simultaneously.

**3. INSIGHTS**

We developed our project by moving from smaller complexity to larger. Firstly we did detection of object as one part as it wasn't easy for first time to measure the distance. Then after detecting the object we have two ways to go with. In the first method we need to find the angle between the object and the camera so as knowing the angle we can measure the distance by simple trigonometry. But we went in difficulty with finding the angle, so we went with the second method. The second method was triangular method. In this we need to find only width of object and with that we would easily find the distance using pre defined focal length.

After overcoming this situation we went through colour detection and distance measurement and completed our task.

**4.Code**

import cv2  
from imutils import paths  
import numpy as np  
import imutils  
import os  
  
def find\_marker(image):  
   
 gray = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)  
 # detecting the image  
 bgrl = np.array([0, 100, 21], np.uint8) # took masking value by masking in previous example and taking the values  
 bgrh = np.array([181, 255, 255], np.uint8) # storing the masking color values in the array of unsigned integer 8  
 # COLOR MASKING  
 mask = cv2.inRange(gray, bgrl, bgrh) # masking so as to be more efficient  
 edged = cv2.Canny(mask, 35, 125)  
 cv2.imshow('canny', edged)  
  
 # finding the contour in image  
 contours = cv2.findContours(edged.copy(), cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)  
 contours = imutils.grab\_contours(contours)  
 # finding the largest contour in image  
 c = max(contours, key = cv2.contourArea)  
  
 # returning the boundary box  
 return cv2.minAreaRect(c)  
  
def distance\_to\_camera(objWdth, fclLnth, perWdth):  
 # calculating distance between camera and object  
 return (objWdth \* fclLnth) / perWdth  
  
# object pre-defined distance from the camera  
objDist = 20  
  
# object pre-defined width  
objWdth = 8  
  
# reading pre-defined image  
image = cv2.imread("redmug.jpg")  
# finding contour in image  
marker = find\_marker(image)  
# manipulating focal length through pre-defined data  
fclLnth = (marker[1][0] \* objDist) / objWdth  
print('fclLnth : ', fclLnth)  
  
cap = cv2.VideoCapture(0)  
  
while True:  
 ret, image = cap.read(0)  
 marker = find\_marker(image)  
 # finding distance  
 distInch = distance\_to\_camera(objWdth, fclLnth, marker[1][0])  
 # drawing a box around the image  
 box = cv2.cv.BoxPoints(marker) if imutils.is\_cv2() else cv2.boxPoints(marker)  
 box = np.int0(box)  
 cv2.drawContours(image, [box], -1, (255, 255, 0), 3)  
 # displaying output on image  
 cv2.putText(image, "%.2f Inch" % (distInch), (80, 50), cv2.FONT\_HERSHEY\_SCRIPT\_SIMPLEX, 2, (255, 255, 0), 3)  
 print("Image-", ": %.2f Inch" % (distInch))  
 # full-screen window  
 cv2.namedWindow('image- ', cv2.WND\_PROP\_FULLSCREEN)  
 cv2.setWindowProperty('image- ', cv2.WND\_PROP\_FULLSCREEN, cv2.WINDOW\_FULLSCREEN)  
 # displaying image  
  
 cv2.imshow('image', image)  
 if cv2.waitKey(1500) == ord('q'):  
 break  
  
cap.release(0)  
cv2.destroyAllWindows()

5. **PHOTOS (Results)**







