eYSIP2018

e-Yantra Automatic Evaluation of Videos

Intern1: Saim Shaikh
Intern2: Siddharth Aggarwal
Mentors: Smita Gholkar Amit Vhatkar
Duration of Internship: 21/05/2018 – 06/07/2018

2018, e-Yantra Publication

e-Yantra Automatic Evaluation of Videos

Abstract

Every Year the e-Yantra Robotics Competition sees 1000s of participants submitting their videos for Line Following Robots. Evaluation of each of these videos manually is very time consuming and involves a lot of human labour. The main aim of the project is to develop a software which can automatically track a line following robot, plot its trajectory and evaluate it with respect to a standard trajectory

Completion status

- Completed Arena Separation and Warping of Arena
- Successfully completed tracking of robot and plotting its trajectory
- Completed Evaluation of line following accuracy using a number of methods like programmatic checkpoints,feature matching,HOG with correlation
- Completed calculation of time required to cross certain zones using physical markers
- Completed handling detection and ID extraction using ARuCO markers
- Improved the Speed of the Code Run using a Multi threading approach
- The color of the physical markers as well as the markers on the top of the robot has been made tune-able
- Developed a GUI for ease of operation of the software

1.1 Software used

- **Python**
- Version: 3.6, Download Python,
- Installation steps:
  - For Windows Users:
    * Download the .exe file using the link provided above
    * Follow the steps as prompted
  - For Linux Users:
    * Open Terminal and type the following commands:
    * sudo apt-get update
    * sudo apt-get install python3.6

- **OpenCV**
- Version: 3.3.2
- Installation steps:
  - For Windows and Linux Users:
    * After Python has been successfully installed
    * Open Terminal/Command Prompt and type
    * pip3 install opencv
    * This will install all main modules for OpenCV but we also need some extra modules like the aruco module. So also run the below command to get them
    * pip3 install opencv-contrib-python

  - For Linux Users:
    * After Python has been successfully installed
    * Open Terminal/Command Prompt and type
    * pip install opencv-python
    * This will install all main modules for OpenCV but we also need some extra modules like the aruco module. So also run the below command to get them
    * pip install opencv-contrib-python

- **PyQt5 (for GUI)**
- Version: 5.10.1 or above
- Installation steps:
  - For Windows and Linux Users:
    * After Python and OpenCV have been successfully installed
    * Open Terminal/Command Prompt and type
    * pip install PyQt5
- **Imutils**
- Version: 0.4.5 or above
- Installation steps:
  - For Windows and Linux Users:
    * After Python and OpenCV have been successfully installed
    * Open Terminal/Command Prompt and type
    * pip install imutils

1.2 Software and Code

Github link for the repository of code
The Software involved can be divided into 3 major parts:

- **Arena Separation and Warping:**
- It involves two major steps:
  - **Separation of Arena from the rest of the Frame**
  - **Warping the Arena to a 500*500 Square Frame**

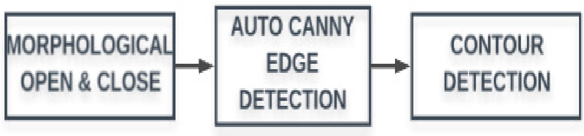


Figure 1.1: Arena Separation FlowChart

```
#Morphological Opening and Closing

blurred = cv2.bilateralFilter(gray, 11, 17, 17) # Removing
    noise from the frame while preserving the lines

kernel = np.ones((5, 5), np.uint8) # Making a 5x5 Kernel

blurredopen = cv2.morphologyEx(blurred, cv2.MORPH_OPEN,
    kernel) # Morphological Opening

blurredopen = cv2.morphologyEx(blurredopen, cv2.MORPH_OPEN,
    kernel) # Morphological Closing

blurredclose = cv2.morphologyEx(blurredopen,
    cv2.MORPH_CLOSE, kernel) # Morphological Closing
```

Snippet 1.1: Morphological Opening is done to remove the Fine Lines in the Frame so that the Lines in the Arena Are better Defined. Morphological Opening is followed by Morphological Closing which Further Refines the remaining lines

```

#Edge Detection and Finding the Contours

edged = cv2.Canny(blurredclose, 30, 200) # Canny Edge
    Detection Algorithm is used for detecting Edges

cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE,
    cv2.CHAIN_APPROX_SIMPLE) # Finding contours from the
    edged frame
```

Snippet 1.2: Once the Edges in the frame have been refined the Canny Edge detection algorithm is used to detect and separate the edges. Then we look for contours in the edged image

```


#Edge Detection and Finding the Contours

for c in cntsSorted:
# looping through the various contours found and
    approximate the contour

    peri = cv2.arcLength(c, True) #Calculating the Contour
    Perimeter
    approx = cv2.approxPolyDP(c, 0.01 * peri, True)
```

```
 # if our approximated contour has four points, then  we can assume that we have found our arena

 if len(approx) == 4: # Checking if the Contour found has 4 corners

 contour_area = (cv2.contourArea(c)) # Finding contour Area
 areapercent = (contour_area / frame_area) * 100 # As the arena will occupy Maximum Area of the Frame, we will first calculate the contours area percentage

  if areapercent > 25: # If Contours's Area > 25 percent of the Total Area of the Frame, Then it is the Arena

      screenCnt = approx
      contours = screenCnt
      flag_contour = 1 # Setting flag_contour to 1 as the Arena has been found
      if flag_contour == 1 and ids != None:
          break
```

Snippet 1.3: After finding the contours we need to filter them in order to get the contour of the Arena. In order to do that we haveset certain conditions like, 1. Contour of the Arena will have 4 points 2. Itwill cover atleast 25 percent of the frame

- **Plotting the Trajectory of the Bot**
- Steps:
  - For Plotting the Trajectory a color marker is placed on the bot
  - The marker is then Filtered Out using HSV filtering
  - Trajectory is calculated by plotting the centroid of the filtered out marker

```

#HSV Filtering

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Converting the frame to HSV

lower_red = np.array([92, 103, 191]) #Upper HSV Ranges for Magenta Color

```

```
upper_red = np.array([111, 195, 255]) #Lower HSV Ranges for
    Magenta Color

mask = cv2.inRange(hsv, lower_red, upper_red) # Applying a Mask
    to filter out color

res = cv2.bitwise_and(frame, frame, mask=mask) # Doing bitwise
    and between frame and the mask to subtract all other colors
```

Snippet 1.4: HSV Filtering is used to filter out the Color marker in the Frame. First the HSV values are filered and placed on a mask then the mask is used to subtract all other colors from the frame

```

#Detecting Marker Contours and Plotting its Centroid

(_, contours, _) = cv2.findContours(edged.copy(),
    cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE) # FInding contours of
    the Marker

        if contours.__len__() != 0:
                cnt = contours[0]

                (x, y), radius = cv2.minEnclosingCircle(cnt) #
    Making Minimum Enclosing Circle around the contour to get
    the coordinates of the centre

                center = (int(x), int(y))

                radius = int(radius)

                cv2.circle(res, center, radius, (0, 255, 0),
    2)#plotting the centres

                if (3.14) * (radius * radius) < 700: # This
    will filter out small contours which are found to be too
    small
                        x = 0
                        y = 0
```

Snippet 1.5: Once the marker has been filtered out its contour is detected and a minimum enclosing circle is plotted around the contour.Then the centroid of the circle is plotted in order to get the trajectory of the bot

- **Evaluation of the Trajectory**
- Evaluation of the trajectory can be done using a number of methods

and weightages can be assigned to each method depending upon the requirements of the arena

- The various evaluation methods are:
  - Programmatic Checkpoints
  - Feature Matching
  - Time Check using Physical Markers
  - HOG with Correlation
  - Follow Accuracy Check

```python

#Evaluation using Programmatic Circles

#Circles are plotted at certain points with reference to the
    standard trajectory

#iterating through the co-ordinates
for i in coordinates:

    #extracting the roi of the plotted circle
    roi = img_circle[b - (3 * circle_radius): b + (3 *
    circle_radius),
          a - (3 * circle_radius): a + (3 * circle_radius)]
    roi = roi.reshape(int(roi.size / 3), 3)

    #checking whether the trajectory is passing through the roi
    by looking for pixel color in roi
     if [255, 255, 255] in roi.tolist():
        check_list.append(1)
        check_counter += 1
```

Snippet 1.6: In this method circles are plotted according to various checkpoints of a standard trajectory. It is checked whether the new trajectory passes through those checkpoints

```python

#Evaluation using Feature Matching

akaze = cv2.AKAZE_create() #create the akaze homography matrix

(akazekps1, akazedescs1) = akaze.detectAndCompute(gray1, None)
    #compute the image descriptors as well as the keypoints
    where the descriptors are located
```

```
bfakaze = cv2.BFMatcher(cv2.NORM_HAMMING) #match the
    descriptors with the descriptors of the standard image

akazematches = bfakaze.knnMatch(akazedescs1, akazedescs2, k=2)
    #refine the matches using the knnmatcher

for m, n in akazematches:
if m.distance < 0.9 * n.distance: #calculate the distance
    beteen the actual and the matches
     goodakaze.append([m])
     goodakaze = np.asarray(goodakaze)#calculate score based on
    the number of good matches
```

Snippet 1.7: In this method the descriptors and keypoints of the trajectory are calculated these features are matched with the features of the standard trajectory and based on the number of good matches a score is calculated

```


bin_n = 16

img = cv2.imread(path_to_perfect_image) # first perfect image
    is read for processing
gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
# Calculating gradient in x-axis
gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
# Calculating gradient in y-axis
mag, ang = cv2.cartToPolar(gx, gy)
# quantizing binvalues in (0...16)
bins = np.int32(bin_n * ang / (2 * np.pi))
# Divide to 4 sub-squares
bin_cells = bins[:10, :10], bins[10:, :10], bins[:10, 10:],
    bins[10:, 10:]
mag_cells = mag[:10, :10], mag[10:, :10], mag[:10, 10:],
    mag[10:, 10:]
hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in
    zip(bin_cells, mag_cells)]
hist1 = np.hstack(hists)
# then the new image is read for same processing
img = cv2.imread(path_to_plot)
rows, cols, _ = img.shape
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 0, 1)
img = cv2.warpAffine(img, M, (cols, rows))
gx = cv2.Sobel(img, cv2.CV_32F, 1, 0)
gy = cv2.Sobel(img, cv2.CV_32F, 0, 1)
mag, ang = cv2.cartToPolar(gx, gy)
# quantizing binvalues in (0...16)
```

```
bins = np.int32(bin_n * ang / (2 * np.pi))
# Divide to 4 sub-squares
bin_cells = bins[:10, :10], bins[10:, :10], bins[:10, 10:],
    bins[10:, 10:]
mag_cells = mag[:10, :10], mag[10:, :10], mag[:10, 10:],
    mag[10:, 10:]
hists = [np.bincount(b.ravel(), m.ravel(), bin_n) for b, m in
    zip(bin_cells, mag_cells)]
hist2 = np.hstack(hists)
# Once both the histograms are obtained, correlation is found
    between them
hog_result = ((np.corrcoef(hist1, hist2)[0][1]) * 100)
return hog_result
```

Snippet 1.8: In this method the histogram of gradients of the images are found and are correlated to get a comparison between the two images

```

#Evaluation using Physical Markers(Filtering the Markers)

hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)#Convert to HSV
lower_red = np.array(hsv_values[hsvtop][0]) #HSV rangers for
    physical marker color
upper_red = np.array(hsv_values[hsvtop][1])
mask = cv2.inRange(hsv, lower_red, upper_red) #create mask for
    the marker
res = cv2.bitwise_and(frame, frame, mask=mask) #separate the
    marker by sybtracting other colors
```

Snippet 1.9: Physical Markers are placed on the arena time can be calculated depending upon the number of markers crossed by the bot

```

#Evaluation using Physical Markers(Frame Count)

#if the pixel value at the marker is 0 i.e the bot has crossed
    over the marker the frame count is initiated
#it is stopped when the bot passes over its pair
                if (x + y + z) == 0:
                    self.cnt_pm += 1
                    self.li_pm.pop(self.li_pm.index(c))
                    if self.li_pm.__len__() % 2 != 0:
                        self.flag_cnt = False
                    else:
                        self.flag_cnt = True
                      )
                        if self.li_pm.__len__()==2:
```

```
                    self.pm_framecounts.append(self.cnt_pm)
                    self.cnt_pm=0
                if self.li_pm.__len__() == 0:
                    self.pm_framecounts.append(self.cnt_pm)
                    break
```

Snippet 1.10: Frames are counted whenever the bot passes over a marker and are stopped whenever the bot passes over its pair

```

#Evaluation using Physical Markers(Score Calculation)

tmarker1 = self.pm_framecounts[0] / self.fps
tmarker2 = self.pm_framecounts[1] / self.fps
```

Snippet 1.11: Once the frame count has been calculated it is divided by the actual frames in the video and the time in seconds is calculated

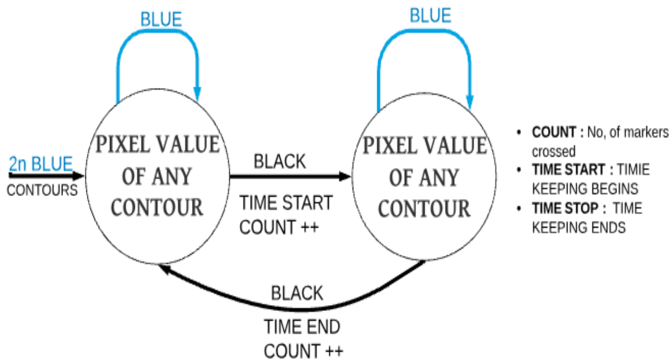


Figure 1.2: Working of Blue Physical Color Markers

```

#Evaluating the follow accuracy by minimizing offset

#The offset is first adjusted by minimizing the distance
    between the first point of the standard trajectory and the
    first ppoint of the user submitted trajectory
if int(x) != 0 and int(y) != 0:
    x = x + self.adj_x
```

```
    y = y + self.adj_y

if img_plot[int(y),int(x),0]==255: # Check if the pixel is
    plotted on White Foreground or Black Background
    self.list_white.append(1)
else:
    self.list_white.append(0)
```

Snippet 1.12: Here the plot of the standard trajectory is used and it is seen how closely the new trajectory follows that path

- **Handling Detection and Extraction of ID**
- Aruco Markers are used for Extraction of ID and Handling Detection and will be placed on the arena
- The Participants will be instructed to Cover the Aruco Marker whenever they are handling the bot
- The Frames where the Aruco Marker is covered wont be counted
- The final count on the number of handlings will be given by the software

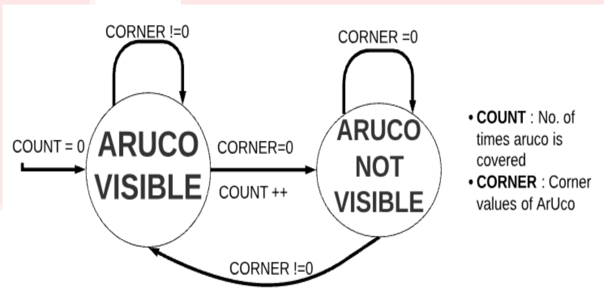


Figure 1.3: Working of Handling Detection

```

#Aruco ID extraction and Handling

aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250) #select
    the dictionary for ARUCO detection
parameters = aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray,
    aruco_dict, parameters=parameters) #detect the id and get
    the corners of the aruco marker
```

```

#check if the corners are equal to 0 in every frame
#if the corners are not equal to 0process the frame else
    increase count
if ret == True and (tlx, tly, trix, triy,
    blx,bly,brx,bry)!=(0,0,0,0,0,0,0,0):
     flag = True

     warped_frame = warping(image,contours)#begin warping

     filter_top_of_robot(warped_frame) #begin plotting trajectory
```

Snippet 1.13: Aruco Handling and ID Extraction

- **Multithreading**

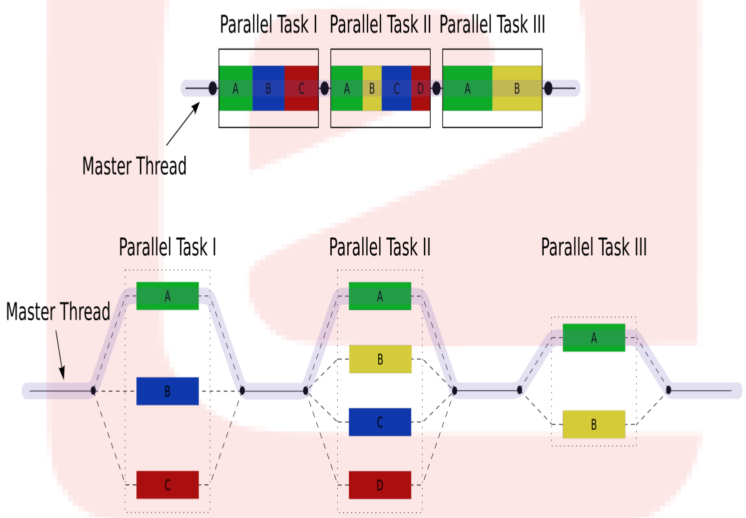


Figure 1.4: Multithreading

- To optimize the code and to make it more efficient, we have applied the concept of multithreading
- The code is designed to work on a fixed number of videos at any given time
- We have tested it to the range of 15 videos running simultaneously using a Quad core processor
- This will increase the speed of evaluation by many folds at the cost of computation

- The program works by initialising a fixed number of threads, and then i=after a interval of every 2 seconds, it checks if any thread is complete. If it is, then a new thread is initialised and so on. This happens until all the files have been executed or are initialised
- When only the main thread remains, the control is sent back to the calling function

```

files = glob.glob(path + '*.mov' or '*.mp4')
index=0
for i in range(0,files.__len__()):
    if i<3:
        th.append(compute_frame(files[i],i))
        th[i].start()
        index=i

while True:
    time.sleep(2)
    for i in range(1):
        if not(th[i].is_alive()):
            print("thread"+str(i)+"is closed")
            index+=1
            if index<files.__len__():
              th[i] = compute_frame(files[index],index)
            else:
                print("All files are in thread")
                if threading.active_count()==1:
                    return
```

Snippet 1.14: Multithreading

1.3 Use and Demo

- A graphical user interface has been developed by us so that it is easy to interact with the software The GUI performs the following functions:
  - It takes the reference video and the participants video folder as inputs
  - To make it more efficient, the GUI also finds the CSV file which is written by the perfect Video and if it finds one, it will give a pop up message asking to use the same reference video, or to replace it

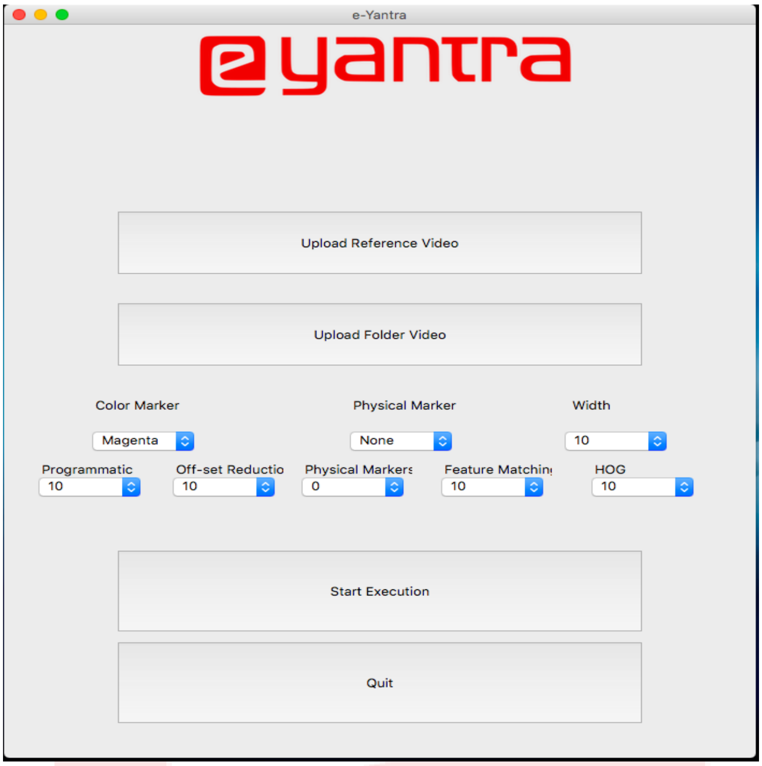


Figure 1.5: GUI for evaluation

- The evaluator can also provide the color of the marker on the top of the bot,as well as the thickness of the reference trajectory to be plotted. The default values are COLOR MARKER : Magenta, PHYSICAL MARKER : Blue, WIDTH : 20
- The evaluator can decide whether the physical markers are to be used ,if yes the color of those markers can also be specified
- The evaluator can assign weights to different evaluation techniques used based on the requirements of the arena. The default values are 20,20,20,20,20, i.e, equal weightage to all methods
- Once all the parameters have been set, the Start Execution is to be pressed which starts running the python script in background

• The software can also be run using the command line. For this, follow the steps below:

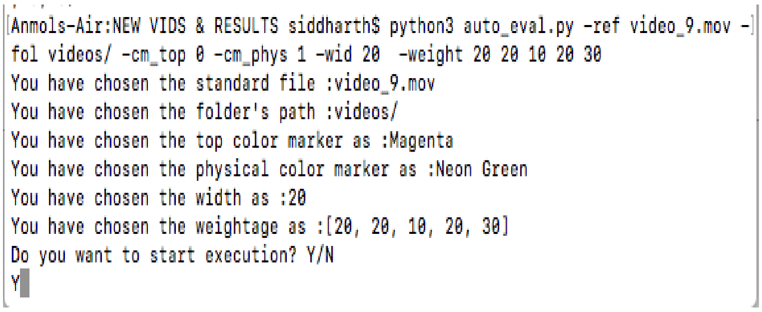
  - It takes the reference video and the participants video folder as required inputs. To input reference video path, use "-ref" OR "–reference" followed by the reference video path

```
Anmols-Air:~ siddharth$ cd Desktop/EYSIP/NEW\ VIDS\ \&\ RESULTS/
Anmols-Air:NEW VIDS & RESULTS siddharth$ python3 auto_eval.py -h
usage: auto_eval.py [-h] -ref REFERENCE -fol FOLDER [-cm_top COLORMARKER_TOP]
                    [-cm_phys COLORMARKER_PHYSICAL] [-wid WIDTH]
                    [-weight [WEIGHTAGE [WEIGHTAGE ...]]]

optional arguments:
  -h, --help            show this help message and exit
  -ref REFERENCE, --reference REFERENCE
                        Path to mp4 or mov video
  -fol FOLDER, --folder FOLDER
                        path to the folder where videos are stored
  -cm_top COLORMARKER_TOP, --colormarker_top COLORMARKER_TOP
                        0 - Magenta, 1 - Neon Green , 2 - Green , 3 -
                        BlueDefault Value = 0
  -cm_phys COLORMARKER_PHYSICAL, --colormarker_physical COLORMARKER_PHYSICAL
                        -1 - None ,0 - Magenta, 1 - Neon Green , 2 - Green , 3
                        - BlueDefault Value = 3
  -wid WIDTH, --width WIDTH
                        Any integer value between 10 to 40
  -weight [WEIGHTAGE [WEIGHTAGE ...]], --weightage [WEIGHTAGE [WEIGHTAGE ...]]
                        A tuple of weight given to each technique of
                        evaluation
Anmols-Air:NEW VIDS & RESULTS siddharth$
```

Figure 1.6: Command Line Execution

- To input the folder containing videos for evaluation, use ”-fol” OR ”-folder” followed by path of the folder. Make sure to type ”/” after the folder name.
- The evaluator can also provide the color of the marker on the top of the bot,as well as the thickness of the reference trajectory to be plotted. The default values are COLOR MARKER : Magenta, PHYSICAL MARKER : Blue, WIDTH : 20. To change them, use ”-cm_top” OR ”–colormarker_top” followed by integer. Use 0 for Magenta, 1 for Neon Green , 2 for Green , 3 for Blue. To change physical color marker, use ”-cm_phys” OR ”–colormarker_physical” followed by integer. Here use -1 for None and rest same as above.
- The evaluator can decide the width of the reference trajectory. To do so, use ”-wid” OR ”–width” followed by a integer between 10 and 40.
- The evaluator can assign weights to different evaluation techniques used based on the requirements of the arena. The default values are 20,20,20,20,20, i.e, equal weightage to all methods. These can be changed using ”-weight” OR ”–weightage”followed by a list of 5 elements.
- Once all the parameters have been set, press enter to begin execution.



```
Anmols-Air:NEW VIDS & RESULTS siddharth$ python3 auto_eval.py -ref video_9.mov -
fol videos/ -cm_top 0 -cm_phys 1 -wid 20  -weight 20 20 10 20 30
You have chosen the standard file :video_9.mov
You have chosen the folder's path :videos/
You have chosen the top color marker as :Magenta
You have chosen the physical color marker as :Neon Green
You have chosen the width as :20
You have chosen the weightage as :[20, 20, 10, 20, 30]
Do you want to start execution? Y/N
Y
```

Figure 1.7: Command Line Execution

1.4 Future Work

- Evaluation can be further made robust by extracting corners from arena and mapping them to a function
- Physical Markers can be used to see whether the bot has completed all the tasks assigned to it
- The project can be further expanded to include drones which can be tracked using ArUco markers on the top

1.5 Bug report and Challenges

- As it is a software based evaluation the arena should follow certain contraints
  - Aruco Marker Should be printed on the Arena
  - The video should be recorded parallel to the aruco edge in order to ensure consistent warping
  - The video should be recorded in a good lighting condition
  - There should be atleast some amount of contrast between the arena and the floor
  - If physical markers are to be used they should be printed on the arena
- The only challenge is some amount of offset which appears with Warping although it has been minimized to a large extent

Bibliography

[1] J.Canny, *A Computational Approach to Edge Detection*, 1986.

[2] L.Yu, *An Improved ORB Algorithm of Extracting and Matching Features*, 2015.

[3] N.Dalai and B.Triggs, *Histograms of oriented gradients for human detection*, 2005.