Gesture-Paint Algorithm using OpenCV

Submitted by
Venkat Musunuru- 31660
Vinod Alase- 31634

June 28, 2019

Contents

Chapter 1

Introduction

In recent years, Many advancements in the fields of Image Processing, Artificial Intelligence have made significant changes in our lifestyle and these advancements have increased our reliance on Computers. Either it is Object detection in the field of Autonomous driving or Working of a service Robot installed in our home this area has made our life more comfortable however Computer Vision is having huge significance as it extends its contribution in every possible application we can imagine in the area of Artificial Intelligence.

Figure 1.1: Robot trying to pick up object based on Gestures shown by Human

Various training methods have been developed over the past few years to enhance the performance of the Robots and improve the learning rate and reduce the manual work that was previously required to be handled by humans. Of all those techniques one of the prominant and widley used technique is Learning by demonstration, where the robot is made to do the task manually with the help of humans, one of the prominant examples of this application is pouring coke or water into a cup. But this involves in large work that is required to be done by humans and if the environment changes or the objects in the environment are set to change then the experiment must be performed again to acheive the goal state.

In such cases the Gesture recognition plays prominant role and has huge applications not only considering this problem but also various other complex problems such as Medical Imaging etc.

Chapter 2

Problem Statement and Requirements

2.1 Problem Statement

The problem statement is slightly close to the above mentioned example i.e. picking or dropping objects based on the gestures but in our case the Goal is to make a paint using the orientation of a particular object that has a unique colour over its whole surface. The further implementation of this project can be character recognition in the paint we have saved.

2.2 Software Requirements

As we are only considering the painting and storing of the image we require Python 3+ version and corresponding OpenCV version which will be 4.1.0 version(tested and prefferred). It is always better to work in a virtual environment as python version supported for OpenCV 4.1 is Python 3.7 and this can be installed using Anaconda in a virtual environment so that the Base Python version will not be altered. But utmost care must be taken while installing Anaconda because Caffe will not work if the host machine has Anaconda installed so if the host machine contains any Caffe model then its better to create a python Virtual environment instead of Anaconda or use the base version of python.

Instructions to install Anaconda: https://docs.anaconda.com/anaconda/install/

For this project implementation we created a new virtual environment named "*cs231n*".

Chapter 3

Implementation

The Implementation is done in 3 stages as indicated in the project proposal.

- The First Stage of project is Object Detection. Based on the BGR values of the pre-defined color we will detect the object and this closes the 1st stage.
- The Second Stage of the project is to get the Orientation of the object and track the movement of the object in the field of view of the camera. This concludes the 2nd stage.
- The Final Stage will be drawing the pattern based on the movement of the object in the space that was specified in the 2nd stage. And that will acheive the goal.

3.1 Setting up the Paint window

Before starting the process we need to setup the paint window that hosts all the options we can provide like different color options and Clear all function. Also we need to initialize a paint window where the lines are drawn, hence initially we define a paintWindow using *paintWindow = np.zeros((471,636,3)) + 255* this command creates a blank white image where our paints will be drawn and this image will be stored. Next we provide options like colors with which I need to draw, Here we provided 2 options Blue and Green, In addition to that we provide a Clear all option that resets the screen which were defined as:

```
colors = [(255, 0, 0), (0, 255, 0)]
paintWindow = cv2.rectangle(paintWindow, (40,1), (140,65), (0,0,0), 2)
paintWindow = cv2.rectangle(paintWindow, (160,1), (255,65), colors[0], -1)
paintWindow = cv2.rectangle(paintWindow, (275,1), (370,65), colors[1], -1)
```

3.2 Stage - 1: Color Detection and Isolation of Object in the Environment

As previously described the BGR values of the colors *Blue, Black and Red* are pre-defined as

```
#BLUE
Lower = np.array([100, 60, 60])
Upper = np.array([140, 255, 255])

#BLACK
Lower = np.array([0 , 0 , 0])
Upper = np.array([180, 255, 30])

#RED
Lower = np.array([100, 60, 60])
Upper = np.array([140, 255, 255])
```

Now these values are converted into (h,s,v) by using the function cv2.cvtColor(), for coverting the values from BGR to HSV we use cv2.COLOR_BGR2HSV.

After this step we begin looking for our corresponding color which we choose, For example I choose Blue for my implementation hence my algorithm will look for the blue color in the frame using the function cv2.inRange(). Once we find our object, we use the Morphological Operations like Erosion and Dilation to remove noise and for Isolation of individual elements from the whole frame.

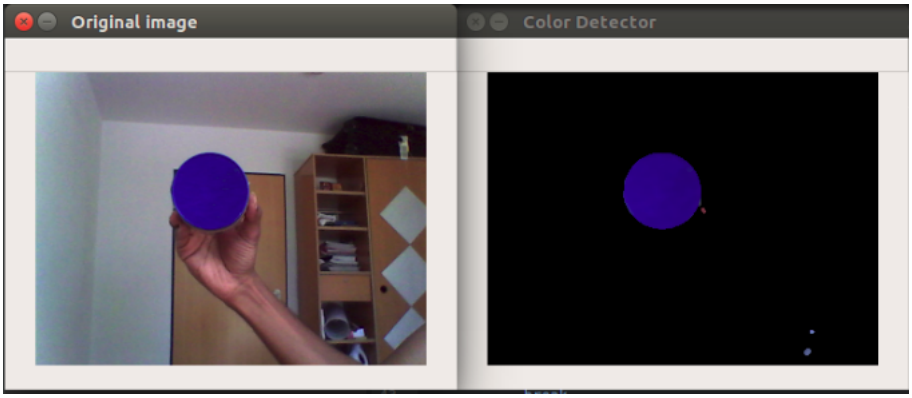

Figure 3.1: Detection of Object with Blue Surface

3.3 Stage - 2: Object Movement Tracking

This is the most important step of all the process, Initially from the Stage-1 we have the object isolated from the environment. Now we must track the movement of the object, hence we first find the contours of the objects for this implementation we use the following code,

(cnts,_) = cv2.findContours(blueMask.copy(), cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

We have to look for any contours are found or not, simply stated we are detecting if there is any object with our color or not. so we use *len(cnts)>0* as check point and moving on we sort the contours found in the environment and we grab the contour that has large area compared to others in the frame. Now we get the radius of the Enclosing circle and draw a circle around the object that completes the Object detection.

We now have a blue color object detected and a circle drawn around it, The next step is to find the centre of the object with the help of cv2.moments() function as follows:

```
M = cv2.moments(cnt)
center = (int(M['m10'] / M['m00']), int(M['m01'] / M['m00']))
```

Till now we have an object with a blue surface and a circle drawn around the surface of the object with its centre detected. Now the tracking of the object begins, We will track the moment of the object based on the centre point that was computed above. Based on the orientation of the centre if it is moving towards the rectangles we defined in the Setup stage, the colorIndex value is set accourding to the position of the center where it is moved. Later based on the colorIndex value we append the points into the respective queues that were defined previously which will be used in the next stage.

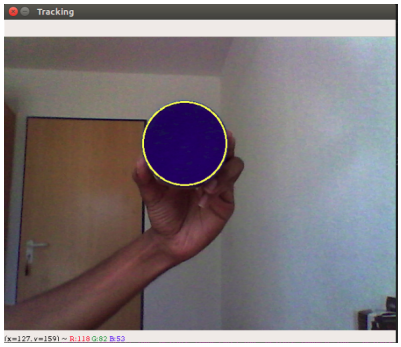

Figure 3.2: Circle drawn on the Object found based on Contours

3.4 Stage - 3: Painting

Major portion of our Gesture-paint algorithm is finished and now we need to draw the lines that were stored in the arrays bpoint or gpoints based on the selection of color. We loop through all the points and find the points that were tracked and draw lines using the points that were stored previously in the indices using the function cv2.line().

```
for i in range(len(points)):
  for j in range(len(points[i])):
    for k in range(1, len(points[i][j])):
      if points[i][j][k - 1] is None or points[i][j][k] is None:
        continue
      cv2.line(paintWindow, points[i][j][k - 1], points[i][j][k], colors[i], 2)
```

We display the frame and paintWindow simultaneosly using cv2.imshow() function. Once finished painting the image is saved using the function cv2.imwrite() and this is done by pressing the button *"q"* from your keyboard. This concludes our project.

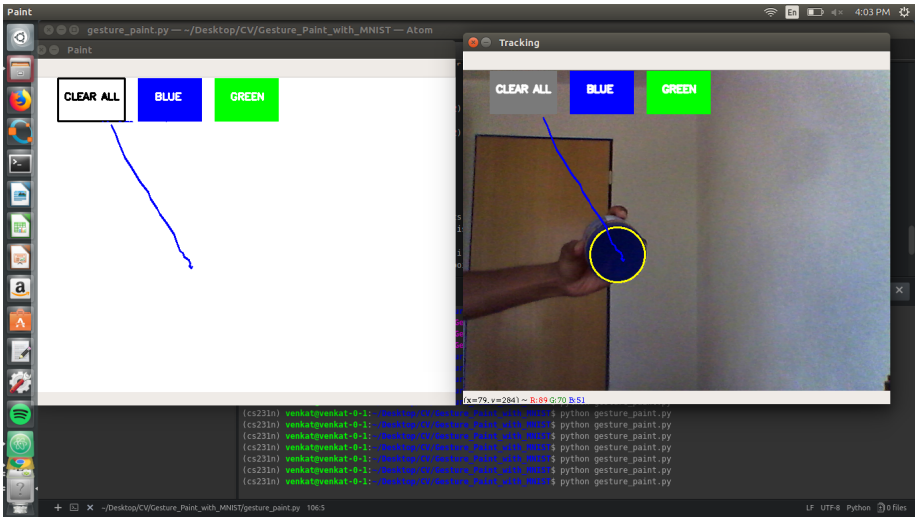

Figure 3.3: Line drawn using the tracked points in bpoint array

Chapter 4

Results and Conclusions

4.1 Results

The following are the results acheived by performing the Gesture paint algorithm in real time environment using a bottle cap with blue color over its surface and the the environment is having proper lighting with less disturbance in the background.

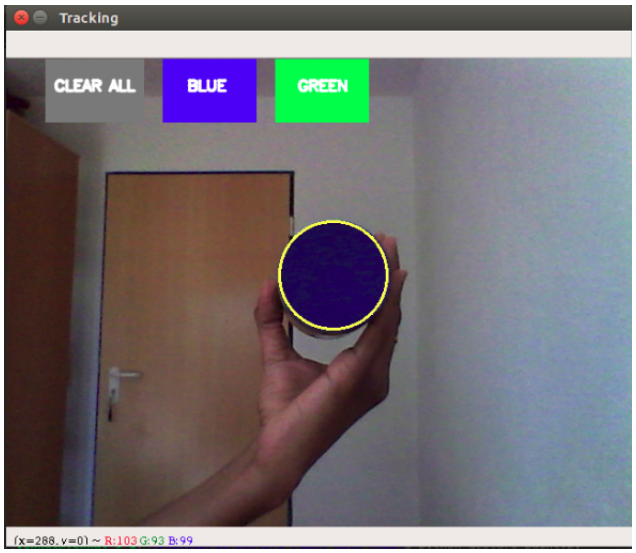

Figure 4.1: Object Detected and Contours with Centre are found

Paint Starts:

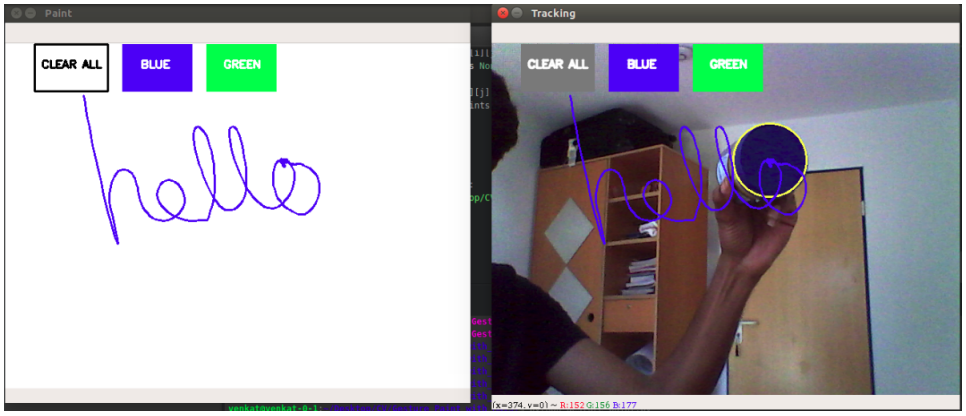

Figure 4.2: Painting is in process with the orientation of the object is tracked

Paint is finished and the paitWindow is saved in *.jpg* format to a user specified path.

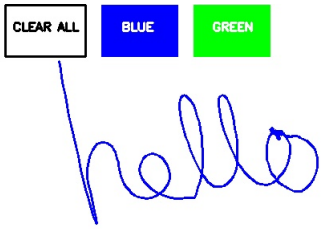

Figure 4.3: Gesture_paint.jpg

4.2 Conclusions

This is a simple Gesture-paint algorithm that was implemented to draw pictures for fun and save them, Most of the smart phones have this functionality but they are using touch sensors or other appliances that have to be purchased, but our gesture paint algorithm enables user to draw with any object they prefer(for example: Finger).

Chapter 5

Applications and Future Scope

We now look at a few ideas which have significant applications based on this project.

- The Object tracking part can be used to model a Mouse cursor that can issue commands like Right and Left click for some basic windows applications like "Refresh",Creating New Folder etc.
- Based on the characted recognition simple commands like "GO", "STOP" can be written and instructed to Robot.
- This project can be implemented as a sub project in the area of Gesture Controlled Robotics where we can use gestures to control the Robot's actions.

5.1 Future Scope

This project can be extended by recognizing the characters that were drawn in the paint. For example use a basic Neural Network and train on MNIST dataset to detect any numbers present in the image.

Chapter 6

References

- www.pyimagesearch.com/2015/09/21/opencv-track-object-movement/
- https://docs.opencv.org/2.4/doc/tutorials/core/basic_geometric_drawing/basic_geometric_drawing.html
- https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/bounding_rects_circles/bounding_rects_circles.html
- Tracking Multiple Moving Objects Using Unscented Kalman Filtering Techniques(https://arxiv.org/pdf/1802.01235.pdf)