COL 780

Computer Vision

Report - Assignment 3

Submitted By :

*Arjun Bhardwaj*

*2016ME10754*

*Tanmay Goyal*

*2016ME20757*

Link to the Google Drive folder containing output videos and images :

<https://drive.google.com/open?id=1_MGN-mCqDEhXD27Dg1AF-nQNJx9WJ3Ro>

**Initialising the code** :-

First of all the OpenCV, Numpy, time, os and objloader\_simple modules are imported. In the main function of the code, we call the main().

**PART 1 - CAMERA CALIBRATION**

1. **Finding out Camera parameters** :-

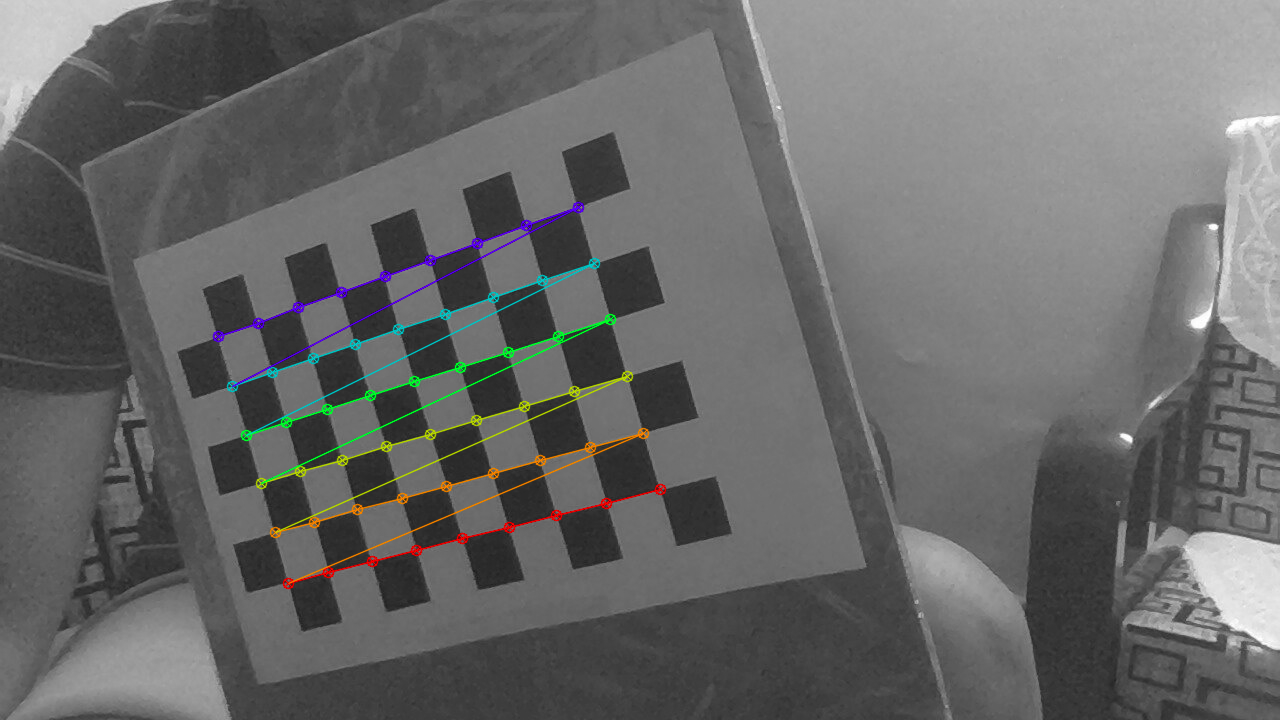
First we found out the camera parameter matrix of webcam used using the calibration code and multiple views of chessboard pattern.

[ 601.1 0 332.8 ]

[ 0 600.2 226.4 ]

[ 0 0 1 ]

*Table 1 - Camera Calibration Matrix*

*Fig. 1 - Calibration using Chessboard Patterns*

**PART 2 - DETECTING MARKERS**

1. **Feature Detection using SIFT** :-

We have chosen SIFT descriptor to find keypoints in the images because SIFT features are robust and better than any other detector. Also it is rotational invariant. It is a bit slow to initialize owing to higher accuracy, but once started runs without any delay. In this way we written all the **reference marker images and frame of live video** in the form of key points. We also then found out the descriptors of each of these key points.

1. **Features matching** :-

We have used BF matcher that takes 2 images and returns an array of the matched keypoints which are present in both the images. BFMatcher is going to try all the possibilities which is the meaning of "**Brute Force**" and hence it will find the best matches because of which it is used to get exact results. We also tried the flann matcher but BFmatcher performed comparatively better.

1. **Finding Homography from the matches** :-

Now we need to find the homography between the images. If the 2 images are different, then we create arrays src\_pts and dst\_pts . These arrays have dtypes float32. Src\_pts is the array of all the keypoint1 and dst\_pts is the array of all the keypoint2 that have a match in the other image. Now we findHomography between the 2 sets of key points and try to fit model using RANSAC which is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. In this way we get the homography between any 2 images and know how to get the 2 images in the same coordinate frame.



*Fig. 2 - Detecting ‘Mario’ Marker*

**PART 3 - 3D OBJECT RENDERING**

1. **Finding out Projection Matrix** :-

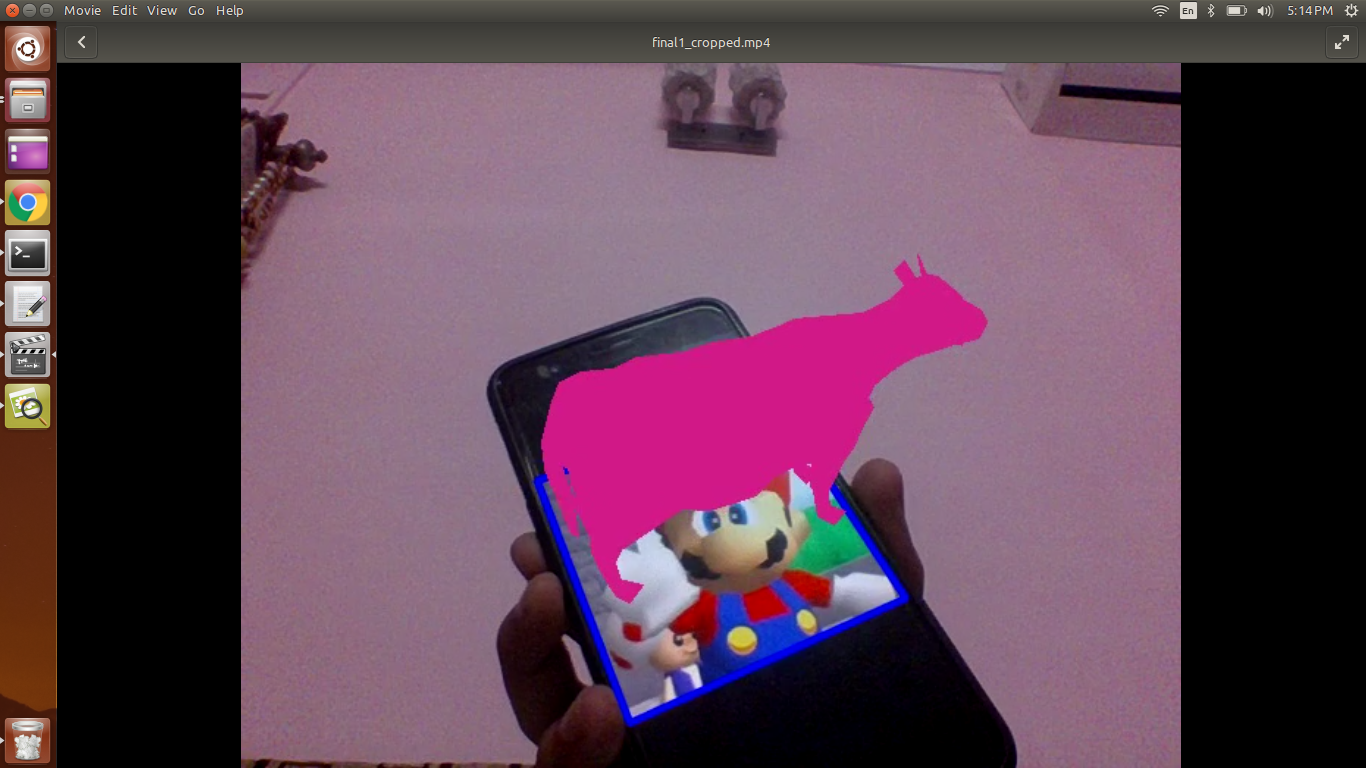
Once we have found out the homography between images that have sufficient matches, we transform the points of the marker to the frame plane and joined the extreme coordinates of the matched portion of the frame to get a rectangle. Now we found out the projection matrix between camera and homography found out. This will be required to project the coordinates of the 3D object onto the image.

1. **Loading .obj file** :-

The .obj file is loaded using the ObjLoader class. It stores the coordinates of the vertices and faces of the 3D object. These will be used to find out the position of the object on the images.

1. **Real-Time Rendering** :-

Then, we called the render() function which renders the 3d object on the frame. Using the projection matrix, it calculates the projection of the vertices of the 3D object. Each face of the object is then drawn on the image. The webcam is used take a live feed of images. We perform the above tasks on each frame and the store the processed images as a video file.



*Fig. 3 - Rendering ‘3D cow’ on ‘Mario’ marker*

**PART 4 - 3D ANIMATION**

1. **2 Markers detection:-**

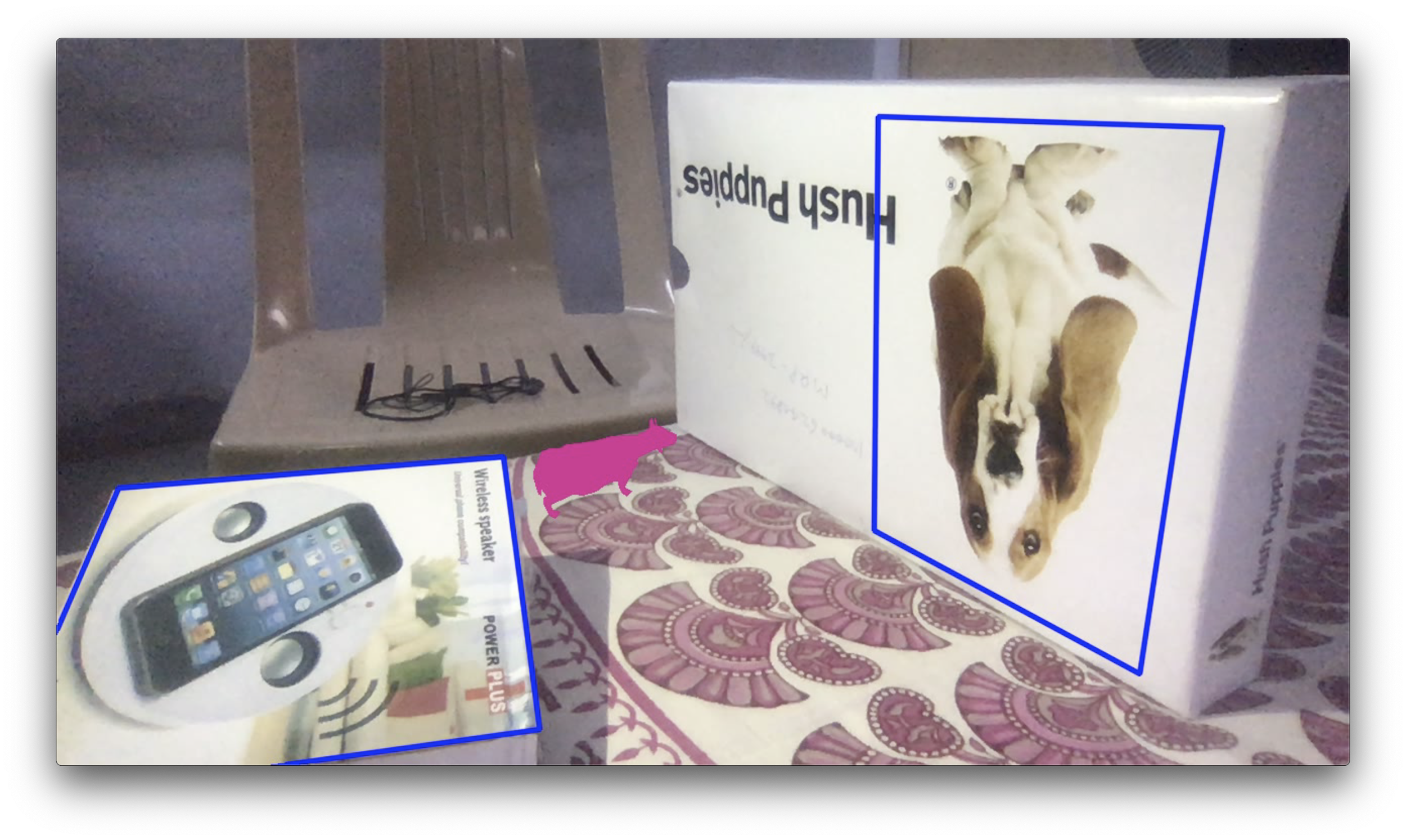
We used the marker detection and 3d rendering this time giving 2 markers as input to detect the 2 markers in the live video feed. Then we chose one as source and one as destination and rendered the 3D object at centre of starting marker.

1. **Translation direction:-**

We found out the corners of both the markers identified in the live video feed. Now we took average of X-coordinates and Y-coordinates of marker-1 to get the centre. Similarly we got the centre of marker-2. Now we found the difference between the coordinates of the centres of both markers. In this way we know the slope we have to move along joining 2 centres.

1. **Movement:-**

Once we know the starting and ending coordinates, we divide it into steps and increase current X and Y coordinates. We parse these new coordinates to the renderAtPoint function which render object at the new coordinate. In this we keep on rendering 3D object at new points until we reach the marker of second object.



*Fig. 4 - Translation from 1 marker to other in live video feed*

**PART 5 - PING PONG GAME**

1. **Random movement**

We first allowed to circle object to move in a random direction

1. **Contour making**

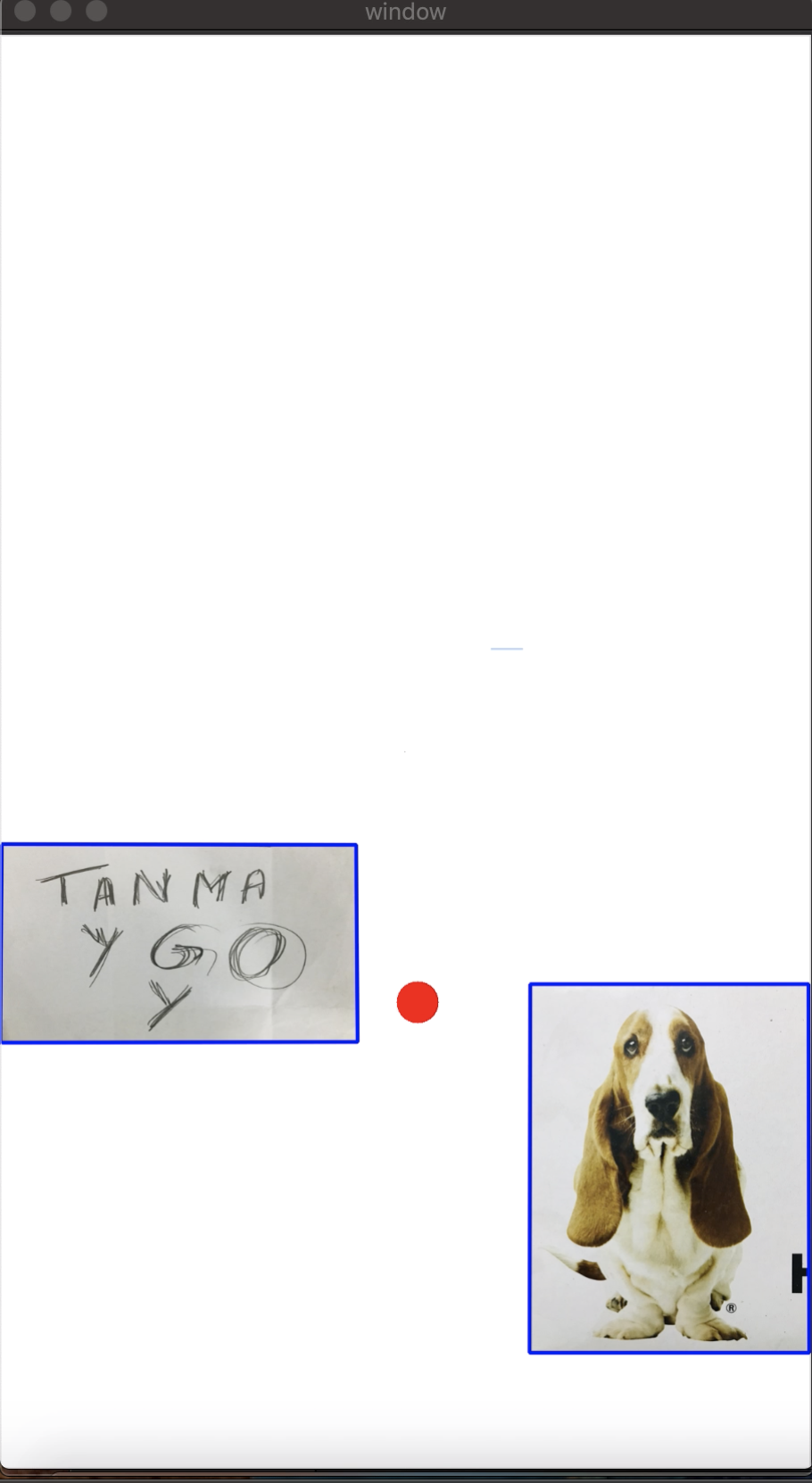
We are detecting the markers in the live video feed in every loop. This makes the loop speed a bit slow but is necessary. Now, once we get the points making rectangle around the both markers, store the coordinates in a contours one of each marker. In this way we get 2 contours.

1. **Distance**

Now we calculated the distance of circle from either of the 2 markers and determined if we were inside any of the contour or not. The function returns positive if inside contour else negative

1. **Angle**

If we find that we are in contour, then we call “changedir” which first finds the vector of the normal to marker and then transforms it into image coordinate system. Now we find the angle between the normal and the direction in which the ball was moving. In this way we get the direction in which ball should move after collision. We then give this new direction to d vector. If this give error then we take line joining centre of that markers and point of collision as normal and perform remaining steps accordingly



*Fig. 5 - Ping pong game*