Real Time Augmented Reality Applications

Shashank Chaudhry, Kuang-Huei Lee and Cheng Han Lee

Abstract – **Augmented Reality (AR) is a technique which aims to augment an existing scene with computer graphics such that the scene contains certain information which may be helpful to the people viewing the scene. AR applications will surely be on the rise with the development of assistive technology such as Google Glass, which makes this field an exciting area for further exploration. The present study attempts to develop some basic and some advanced functionality of AR techniques and implement it in real time. We use marker based AR methods to detect and replace markers in a scene. We have shown 2D AR methods, where the augmentation of a webcam frame is done using 2D images and videos by applying accurate homographies. We then extend this by rendering 3D objects in front of markers. We have also incorporated hand detection code into the AR implementation, to show marker less AR techniques, and are able to suspend 3D objects in front of hands or other objects, which are detected by the code. Finally, we address a common issue in AR technology, i.e. that of occlusion, and show our attempts at rectifying occlusion in a live video feed.**

I. INTRODUCTION

In the current age of massive computing power, where even mobile phones come with quad-core processors and lightning fast graphics rendering, real time Augmented Reality (AR) is a very achievable goal and one that is finding numerous applications in the real world. Silva et al. [1] define AR as "a new technology that involves the overlay of computer graphics on the real world." Unlike Virtual Reality (VR) where a user will be completely immersed in a virtual environment, AR continues to allow a user to see the real world, but with virtual objects superimposed on real world scenes [2]. AR is an interesting area of research as it performs the function of Intelligence Amplification [3] namely allows a user to have information at his fingertips that would not have been available using only his five senses.

There can be numerous application of AR in various fields of study [2]. In medicine, AR can be used for viewing the results of ultrasounds and CT scans using Head Mounted Displays (HMDs). In information systems, we could augment real world images with annotations for greater information. In robot path planning, AR can show the operator the visual view of the path of a robotic arm, before actually implementing the change. AR also holds promise in the manufacturing sector, as the repair information for a product could be show virtually, rather than the repairman having to look at a manual. AR also has numerous applications in the military, where graphics can be overlaid on a pilot's display, to assist in flying an aircraft, or aiming weapons.

A number of companies are developing great ideas in the field of AR, and are implementing it for the latest gadgets, such as Google Glass. For example, a possible application of AR would be to provide tailored advertisements to individuals in the future, with the assistance of Google Glass. Tesco recently tied up with an organization called Blippar to provide such a service [4]. Another extremely exciting application of Google Glass for AR is being developed by LAYAR [5], which would allow functionality such as allowing a user to look at a movie poster and see the trailer on the poster itself, or see real estate listings in the location where you are standing or see augmented information in magazines. AR for mobile phones has been in the marker for a while now, such as Yelp, which introduced a tool called "Monocle" [6] that uses AR to show restaurant ratings for people walking at a location.

The latest research in AR is leading to a number of developments in the field. Lepetit [7] talks about the latest marker-less detection techniques, using tracking by detection. This method uses RANSAC based tracking of an object to detect and then estimates the 3D pose from the object orientation. It then discusses a solution to the occlusion of marker issue, using background subtraction, but then goes on to explain the limitations of this process. We have attempted to implement the same occlusion based process, but have seen poor results. Takacs et al. [8] have implemented an outdoor AR model that compares images being captured by the camera of a mobile phone to a large database of images, and augments the scene being observed. Lee et al. [9] have also developed a marker-less AR system which recognizes the fingertips of individuals to estimate the camera pose and then uses the hand as a marker to interact with 3D objects.

The present study is an exploration into AR, and tries to implement some basic AR techniques and also implement some applications of AR technology. In the present study, we have implemented a real time implementation of marker based AR on a computer, using webcam feed. We have shown a number of applications of AR, such as real time marker detection and replacement with images and inset video frames. We have also looked at some advanced AR techniques such as implementing a marker free AR system with hand detection, incorporating OpenGL based rendering of 3D objects in the scene and have also developed an occlusion handing methodology.

II. DETAILS OF THE APPROACH

The AR project has been broken into stages to ensure that the minimum goals that were set can be achieved. As each goal of the project is completed, another goal has been implemented. The following are the steps of the project:

- Implement an AR solution which detects a marker and replaces it with an image
- Repeat the above steps to implement the solution in real time and embed a video into the live stream of a webcam
- Implement a methodology to help carry out occlusion registering and correction
- Incorporate OpenGL 3D rendering into the video output such that the marker is not replaced with an image or video but with a 3D object
- Utilize a hand-detection code to transform the physical marker to a marker which represents the position of a hand and place the 2D or 3D object with the hand as the marker

The entire code has been written using OpenCV. The reason for selecting OpenCV over MATLAB was that one of the aims of this project is to display results in real time, and OpenCV is reputed to be a faster tool than MATLAB.

A) Augmented Reality with 2D augmentation:

In this stage the objective is simply to take an image with a marker in it, and embed another image in place of the marker. There are three steps into which the problem of AR for 2D augmentation can be divided. These three steps are calculation of the location of the markers in the image, calculation of the homography and its application on a source image or video frame and a merging step to combine the warped 2D image and image containing the marker.

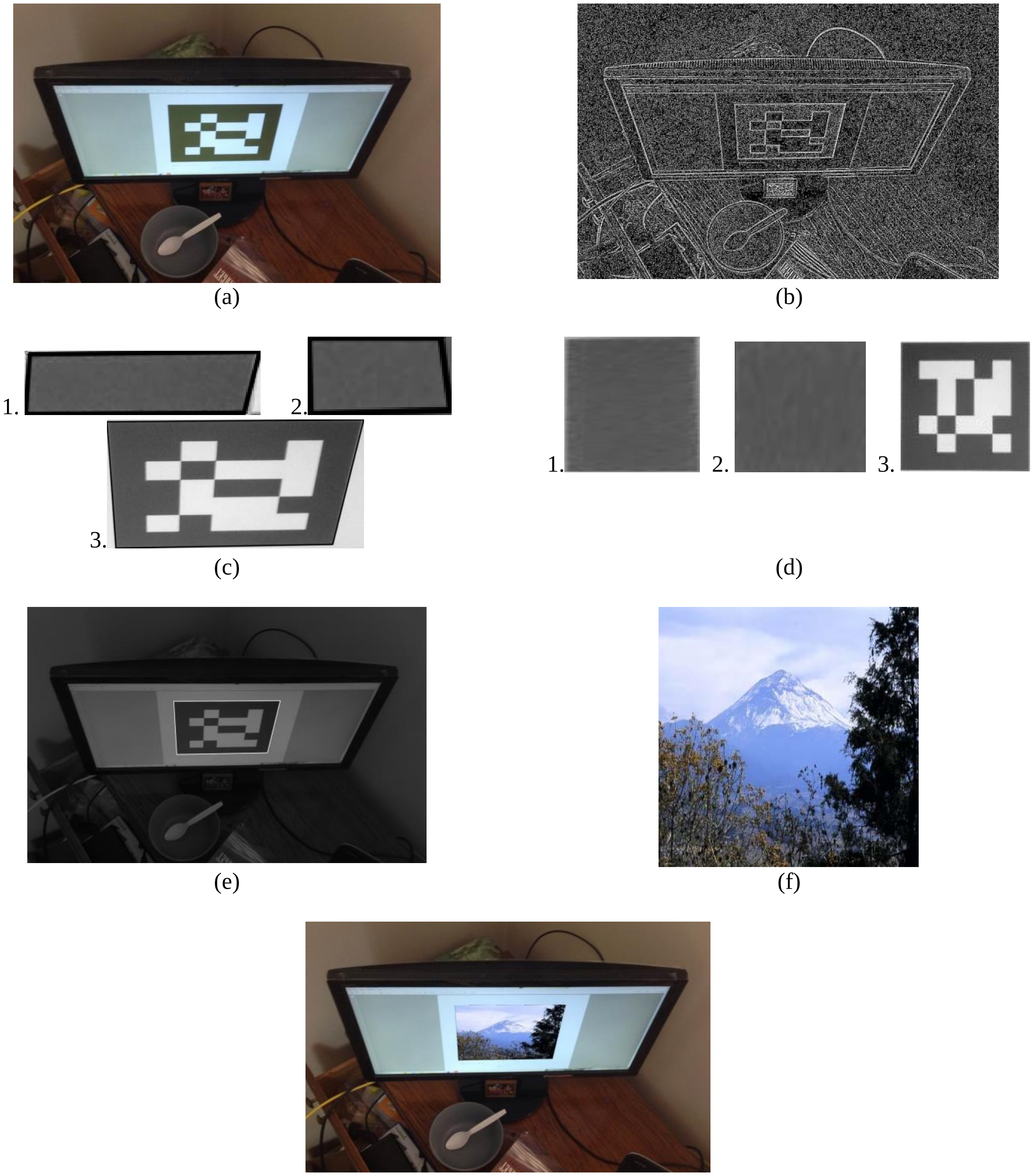
a. Marker Location Estimation:

i. Marker Detection:

The marker detection algorithm is based on the ARToolKit [10] algorithm and is a modification of the code for marker detection by Khvedchenia Ievgen [11]. It applies the following steps:

- Take the given image and convert it to grayscale.
- Convert the image to binary, i.e. with only 0s and 1s, by applying thresholding operations from OpenCV. Used the adaptive thresholding algorithm from OpenCV, and had to play with the parameters. The final parameters set were: adaptive method- Gaussian, threshold type- binary inverse, block size- 7 pixels, constant – 1 pixel.
- Detect all the contours in this thresholded image.
- Reject those contours that cannot be approximated to have 4 sides (representing the marker). Also reject those markers that are too small or those that are too close to another marker, which means that they are the same marker.
- Now, the markers are designed to be easily differentiable from non-marker 4 sided using the Hamming approach explained subsequently. Using this find which of the possible markers are really markers, and find the marker IDs. These IDs can be used if there are multiple markers and the different markers have to be replaced with different images.

The steps that have been explained above are shown in fig. 1.



(a) (b) (c) (d) (e) (f) (g)

Fig. 1. Stages of marker detection. (a) Initial image (b) Image after adaptive thresholding (c) 4 edged contours detected (d) Possible markers after warping the 4 edged contours (e) Good markers, after checking the hamming code in the markers (f) Image to embed in place of the marker (g) AR with embedded image to replace marker

ii. Marker Definition:

The markers have been defined as 7 x 7 grids, with black and white fills into each of the cells. The black and white colors have been chosen, as they are easier to identify after thresholding. The outermost rows and columns are all black as this is used to find the 4 sided contours corresponding to the marker. The 5 x 5 grid inside is like a binary hamming code. For example, consider the image shown in fig. 2.

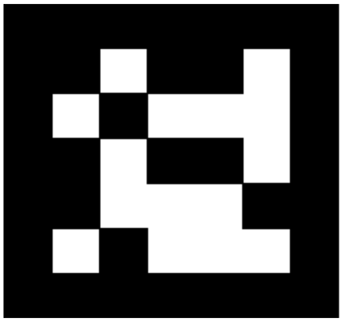


Fig. 2: Defining the 7 x 7 grid marker

Read the 5 x 5 grid as rows. This can be read as:
→ 0, 1, 0, 0, 1
→ 1, 0, 1, 1, 1
→ 0, 1, 0, 0, 1
→ 0, 1, 1, 1, 0
→ 1, 0, 1, 1, 1

Only the highlighted cells of each row (2, 4) hold data. Rows 1, 3, 5 are used for hamming. Thus this marker has an ID given by the binary code: 1001101101 => 621 in int. Thus such a marker design, when being searched in an image reduces the chance for false positives and allows a total of 1024 marker IDs.

b. Homography and Warping:

Perspective transformation is used to calculate the homography between the square sample set of points and the given end points of the marker, which is also square. Using the given homography a warped image on a black background the size of the camera image is generated of the image to insert into the frame. Both these operations are carried out using in-built OpenCV functions. Since the image that is being inserted over the marker does not need to be the same aspect ratio or scale as the marker, the code is modified to allow the aspect ratio of the image to be embedded to be maintained and the also allow the size of the image being embedded to be set as a ratio of the size of the marker.

c. Merging the warped image with the camera frame:

OpenCV has a technique to combine images where a portion of one image needs to be inserted into another. A mask is made, which is a matrix of the size of the webcam image consisting of 0s and 1s, where the 1s represent the locations where you want to replace the image. This masking method was used to combine the camera frame with the inset frame.

B) Augmented Reality in real time:

There is a lot of overlap in the AR methods used for the first objective and carrying out the implementation in real time. We observed that the code we have written is not fast enough for true real time and that there is a small amount of lag. However, the lag is insignificant, and the code performs fairly well, when it is simply modified to run in real time, by looping the approach we have used for different frames of a webcam feed. Thus, we took the original code and modified it to process the camera and video feed frame by frame. We then replace the designated areas of the camera feed with the desired video and return the frame to the user.

However, we did see that there are some issues with real time that are not present, when you are trying to embed an image into another. This led us to make certain modifications in the code to increase the robustness of the code.

a. *Initialization of the code:*
We observed that the code takes some time to register the camera and video frames, thus for the first few frames, we only display the webcam capture, and do not process the results.

b. *Marker not detected due to blurring:*
In real time, the camera or the marker in the camera frame may be moving periodically. This results in a bit of blurring, which may make the contours of the marker difficult to detect. The marker may thus be detected in a few frames and missed in a few others. We decided to accommodate the motion of the marker into our model. Thus if a marker is detected, we store the end points of the marker in arrays for up to five frames. When the marker is not detected, for up to five frames of the marker not being detected, a linear model of the velocity of the end points of the marker, when the marker was visible, are used to estimate the new “probable” position of the marker. Thus instead of the marker periodically disappearing, it remains visible and moves in the direction of previous motion of the marker, in hope that the marker would continue its direction of motion, and would be redetected soon. If the marker continues to remain unobserved in the subsequent five frames, the inner video frame is not embedded.

This required us to write functions to calculate estimated marker velocity using prior marker positions using a linear fit.

C) Occlusion detection and rectification:

The problem of occlusion is one that definitely poses difficulties in the field of AR and this is one of the yet unsolved problems in the field. The problem is as follows, and has two aspects. Firstly, if a portion of the marker is occluded by an object, how can we detect the marker and if the marker position is recovered, how do we obtain the true shape of the marker such that the image to replace it can be successfully placed in the correct location. The second problem is that even if we detect the marker boundaries and embed the image on the marker, it will cover the occluding object, which would be incorrect, since we want the AR to be realistic and represent the true replacement of the marker, or in other words, the embedded image should appear to be occluded too. The solutions to both aspects of the problem are explained:

a. *Marker Detection:*
The original marker design used the fact that the outer border of the marker can be replaced by a convex polyhedron with four sides. This will no longer be true once a portion of it is occluded. However, we do know that our marker design allows us to distinguish between 1024 marker types. Thus a new marker design has been proposed, which will place 6 markers with different marker IDs inside a rectangular region as shown in fig. 3.

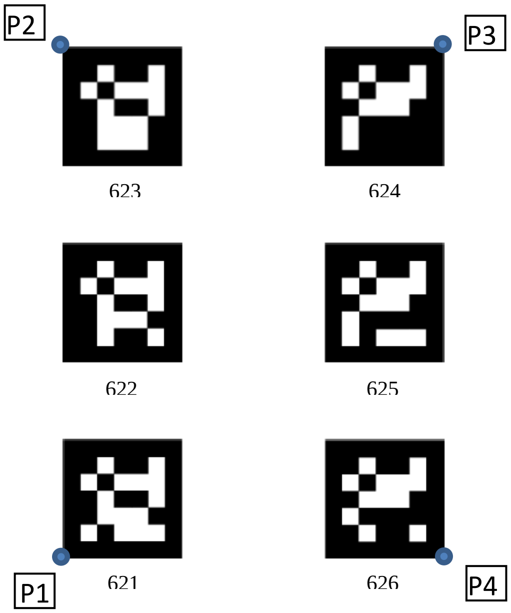


Fig. 3. Marker design for occlusion handling

The advantage of such a combination of markers is that even if a portion of the markers are occluded, the remaining markers will be found and they can be used to determine the positions of the other markers. In other words, you use whichever marker positions you have to somehow estimate a new bigger marker, whose end points are P1, P2, P3 and P4.

Moreover, since all the markers have different IDs, the markers found will give a clear indication of the markers missing and the position of the missing markers in relation to the markers found. Theoretically, if the marker is printed with accurate knowledge of the ratio of lengths between every two markers and the lengths of each individual marker, even the detection of a single marker can approximate the remaining markers. However, this may not result in a marker that is a very accurate representative of the true location of the marker, since it uses geometry which may be skewed, especially if the marker in the image is pixelated. In the present project, as a proof of concept, the code works as long as any two or less markers are occluded.

Assuming the marker looks like the one show in fig. 3, the pseudo code for the sub-routine for calculating occlusion is given below. The sub-routine is inserted between the markers detection and homography calculation steps.

Pseudo code:

```
detectedMarkers = getMarkers(CameraFrame)
if(detectedMarkers.size() == 6)
        set the newMarker as P1,P2,P3,P4
if(detectedMarkers.size() == 5){
        if the markerIDs found include 621, 623, 624 and 626
                set the newMarker as P1,P2,P3,P4
        }
        else {
                if 621 is missing
                        P1 = findMissingMarkerPoint(621)
                else if 623 is missing
                        P2 = findMissingMarkerPoint(623)
                else if 624 is missing
                        P3 = findMissingMarkerPoint(624)
                else
                        P4 = findMissingMarkerPoint(626)
                Set the newMarker as P1,P2,P3,P4
        }
if(detectedMarkers.size() == 4){
        if the markerIDs found include 621, 623, 624 and 626
                set the newMarker as P1,P2,P3,P4
        else {
                if 621 is missing
                        P1 = findMissingMarkerPoint(621)
                if 623 is missing
                        P2 = findMissingMarkerPoint(623)
                if 624 is missing
                        P3 = findMissingMarkerPoint(624)
                if 626 is missing
                        P4 = findMissingMarkerPoint(626)
                Set the newMarker as P1,P2,P3,P4
        }
}

Point findMissingMarkerPoint(missingPointID){
        if (missingPointID == 621){
                Use (622 or 623) and (626 or (624 and 625)) to get P1
                return P1
        }
        if (missingPointID == 623){
                Use (621 or 622) and (624 or (625 and 626)) to get P2
                return P2
```

```
            }
            if (missingPointID == 624)
                  Use (625 or 626) and (623 or (621 and 622)) to get P3
            return P3
            }
            if (missingPointID == 626)
                  Use (624 or 625) and (621 or (622 and 623)) to get P4
            return P4
            }
      }
```

b. *Occlusion rectification:*
Two approaches were attempted for rectifying the occlusion of the marker, assuming we have now got the four end points of the marker using the above technique. The first attempted to resolve the foreground of an image using foreground detection techniques from OpenCV. The second used the fact that the marker color is black and white and used the color in front of the marker area as an indication of occlusion.

i. Foreground Detection:
OpenCV has inbuilt techniques for foreground detection. In this approach, you initially ensure that there is no motion in the video capture, and that the first frame has no motion of any kind. Then, the detector has established a definition of the background. In the subsequent frames, the new frame captured is subtracted from the background and a thresholding operation is conducted to make a foreground mask. We then take a bitwise and between this mask and the marker location mask and then apply this mask to the webcam frame to get the final frame. You can also have a training rate which will keep modifying the definition of the background with the number of frames seen. Theoretically, such an approach should have worked well. However, we observed poor performance with this method, as it is not able to properly threshold motion and can give only a semblance of where motion may have occurred in the frame, but not an accurate location of the pixels at which motion occurs.

ii. Color-based Detection:
In this approach, we utilize the fact that the marker is black and white. In the region of the marker location mask, we check each captured pixel. If the pixel is approximately white i.e. the three channels of the frame are all greater than around 240, or is approximately black i.e. the three channels of the frame are all lesser than around 20, we left the pixel in the mask unchanged. If it did not lie in this threshold, we inverted the mask pixel value. We observed much better results with this approach.

D) Augmented reality with 3D object insertion:

In this part, a marker-based 3D AR is implemented using OpenGL. We used solvePnP() function (solving Perspective-n-Point problem [12]) of OpenCV to reconstruct the translation and rotation matrix from position of marker corners and camera matrix. The result was combined with OpenGL to render and transform a 3D object in a real scene.

To implement this part, the OpenCV process and OpenGL process were put in two different threads since their main loops which are responsible for rendering and processing video frames are not really compatible with each other. All the processing with OpenCV is written in a child thread, which continuously refreshes the video frame and the transformation information, and the OpenGL main loop continuously reads the current video frame and the transformation information to generate the final output. The structure of this implementation is shown as follows:

```
global variable: transformation, OpenCV-processed video frame
OpenGL main loop:
      read and draw OpenCV-processed video frame
      transform and render 3D object
      output final video frame
```

```
child thread:
      OpenCV main loop:
            read video frame from camera
            detect marker
            get transformation from solvePnP()
            save video frame to OpenCV-processed video frame
```

A result of rendering a teapot on marker is shown in fig. 4.

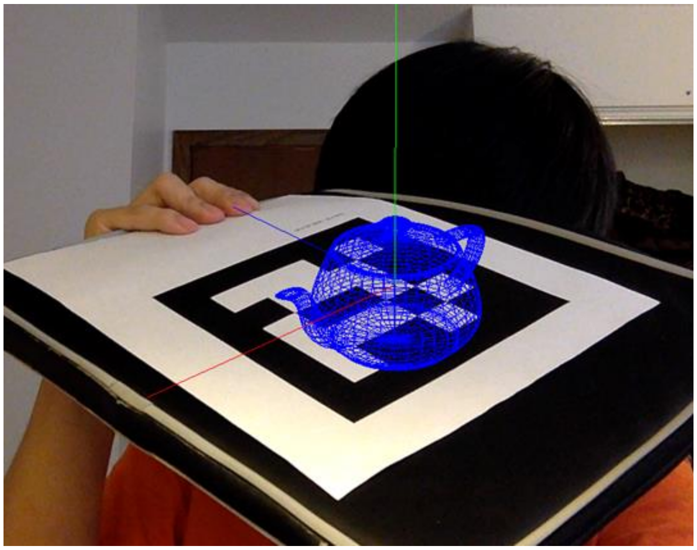


Fig. 4. Marker based 3D rendering using marker pose estimation

E) Markerless 3D AR using color based thresholding:

In this part, we implemented a non-marker-based 3D AR with OpenGL. Instead of using the real marker, a bounding square of a specific color block was used to compute the transformation vector. Rotation was computed, but not always used in this part since we marker substitute bounding box did not contain rotation information. As in marker-based AR, 4 corners of the bounding square was fed to the solvePnP() function to solve for the transformation. The structure of this implementation is similar to marker-based AR:

```
global variable: transformation, OpenCV-processed video frame
Sampling the color to capture
OpenGL main loop:
      read and draw OpenCV-processed video frame
      transform and render 3D object
      output final video frame
child thread:
      OpenCV main loop:
            read video frame from camera
            calculate binary map based on color threshold
            compute bounding box of the largest color block
            get transformation from solvePnP()
            save video frame to OpenCV-processed video frame
```

The code for this section was based on hand tracking and recognition code [13]. The result of rendering a sphere in front of an orange T-Shirt has been shown in fig. 5.

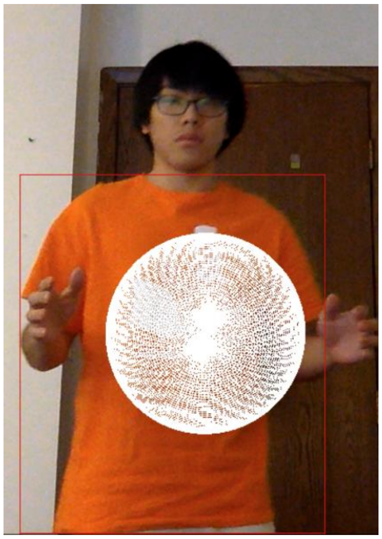


Fig. 5. 3D rendering on a colored T-shirt

a. *Color Thresholding for color block:*

This technique is used in color block detection including skin area detection. According to Zarit et al.[14], HSV is the best representation of the RGB color space in skin detection. HSV yields the highest correction rate of skin pixel detection. Instead of predefining the target color interval, we ask user to register the average color of their skin (fig. 6) and allow upper and lower thresholds to convert each frame to a binary map (fig. 7).

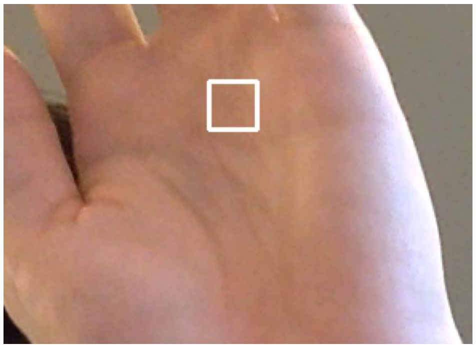


Fig. 6. Color block detection

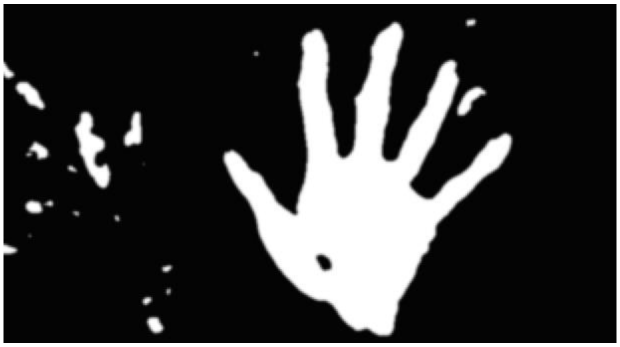


Fig. 7. HSV based thresholding

In order to achieve accuracy, the user is asked to sample at 7 color points and each of them generates one binary representation each. Summation and median blur of the binary representations can provide quite an accurate contour of a color block. The representation of this method is shown in fig. 8.

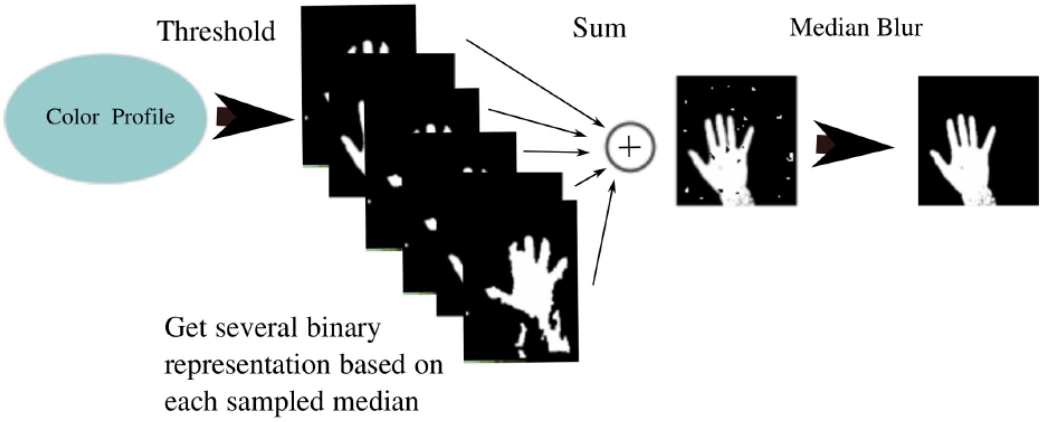

Threshold
Sum
Median Blur
Color Profile
Get several binary representation based on each sampled median

Fig. 8. Median blur of binary representations

III. RESULTS

We have recorded videos of a number of techniques that we have used, and along with the images. The videos are available at the following link: https://www.dropbox.com/sh/alrftiqv7fm6qko/AACJkUHePEuDBupk2u7ArsD-a.

A) Augmented Reality with 2D augmentation (post-processed and in real time):

The code for AR with 2D image and video embedding works very well. The initial code was designed for embedding square images into the square marker. Results of this embedding for various orientations of the marker are shown in fig. 9 (for image in image embedding).

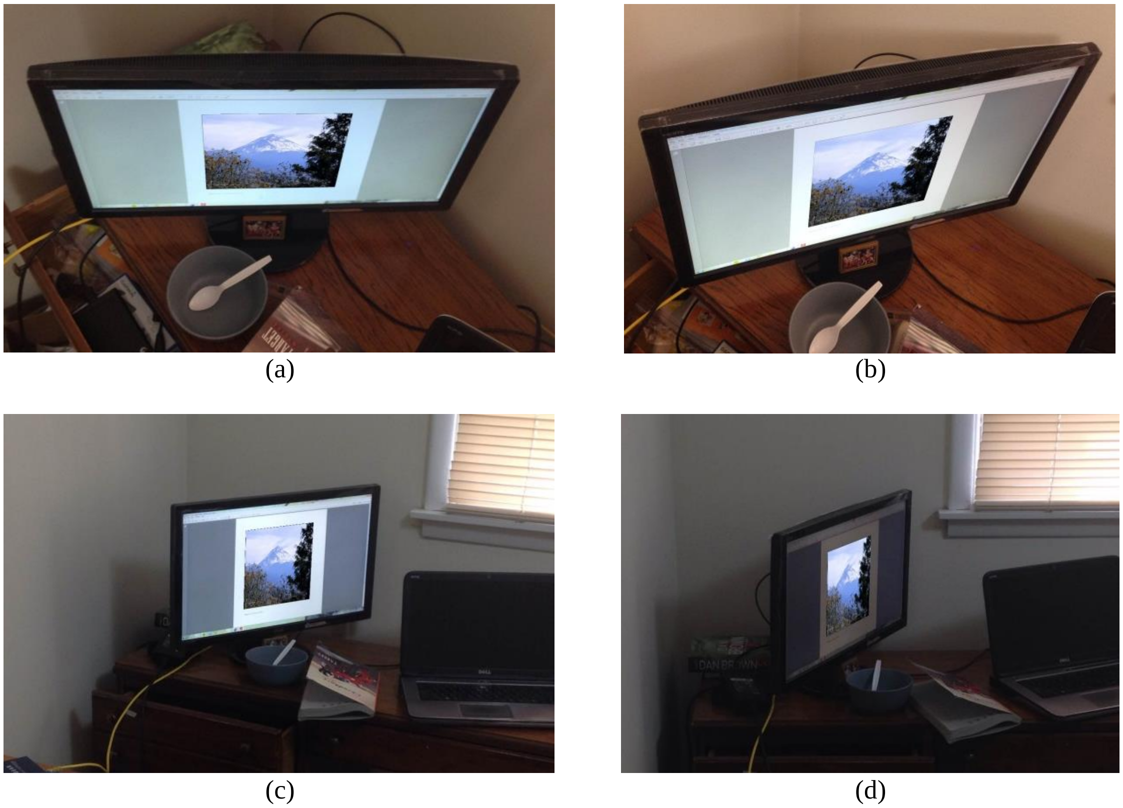
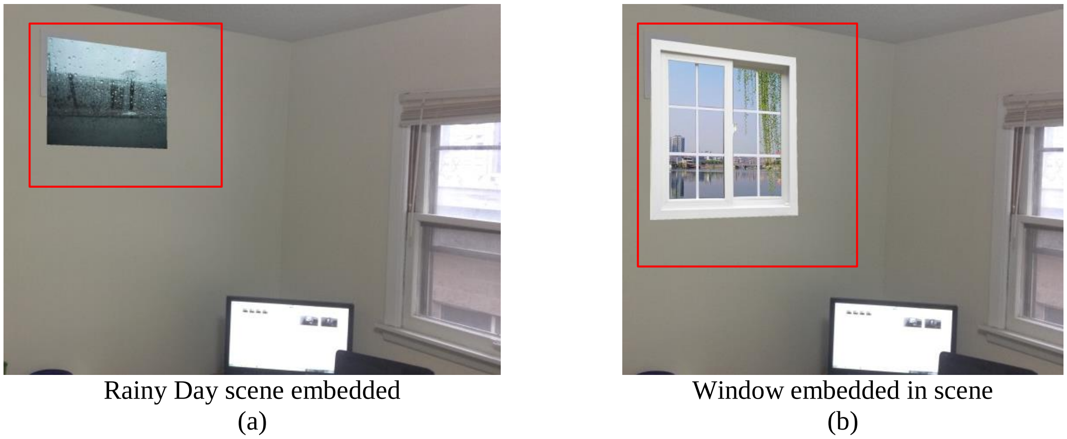

(a) (b) (c) (d)

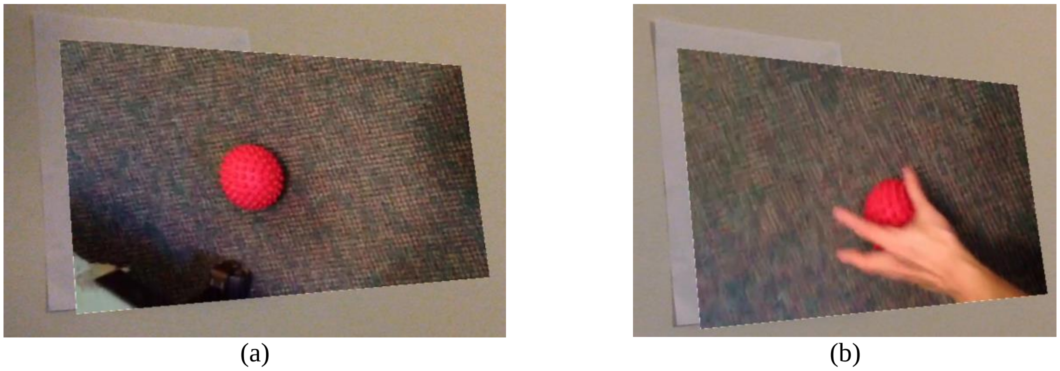
Fig. 9. Image in image embedding for 2D augmentation with image embedded in marker

Subsequently, the code was modified to maintain the aspect ratio of the images and video frames being embedded and also allow the user to control the size of the embedded image as a ratio of the size of the marker. Fig. 10 shows the 2D augmentation for image in image embedding and fig. 11 shows the same result for video in video embedding.


Rainy Day scene embedded
(a)

Window embedded in scene
(b)

Fig. 10. Image in image embedding for 2D augmentation with image maintaining original aspect ratio


(a)

(b)

Fig. 11. Video in video embedding for 2D augmentation with inner video maintaining original aspect ratio

B) Occlusion rectification:

Occlusion rectification methodologies, as explained in Sec. II., describe two different occlusion methods that were attempted. The first method used background subtraction to detect motion in the foreground. The result of this technique is shown in fig. 12. The foreground detection technique, as can be seen from the figure, does not perform too well. The other approach used the fact that the average color of the marker is black and white. The results of this section are shown in fig. 13.

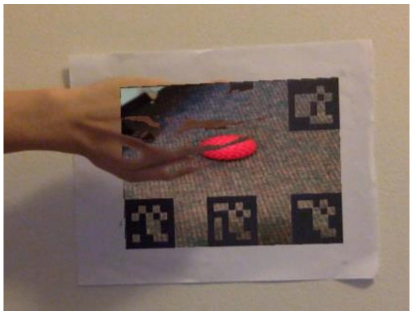
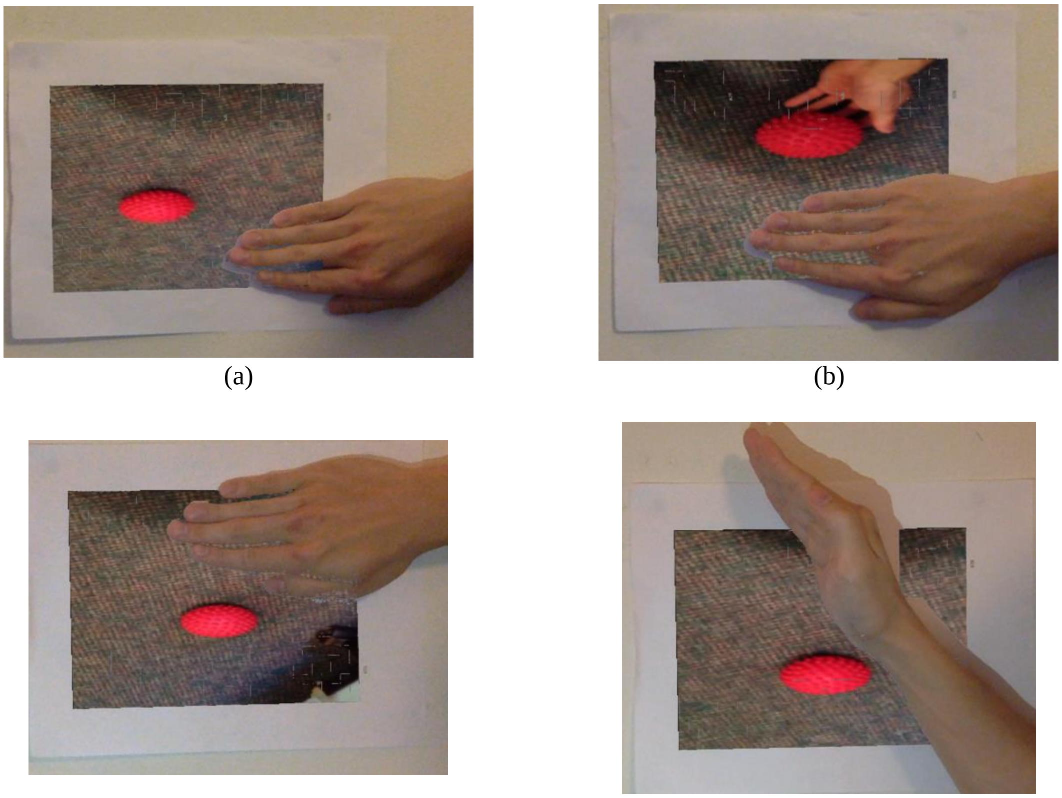


Fig. 12. Occlusion handling in real time video-in-video AR by foreground detection method



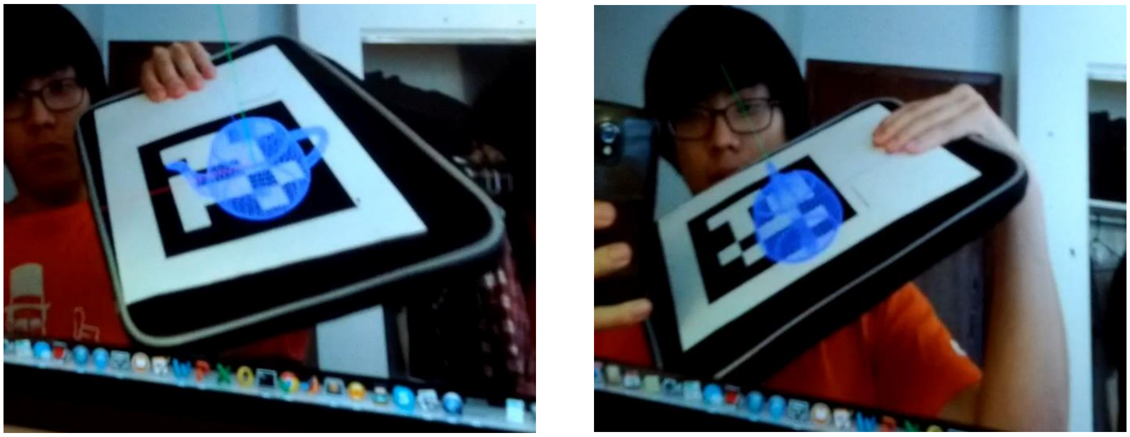
(a) (b)

(c) (d)

Fig. 13. Occlusion handling in real time video-in-video AR by color-based method

C) Augmented reality with 3D object insertion:

The results of the 3D object insertion have been shown in fig. 4. Some additional images are shown in fig. 14.

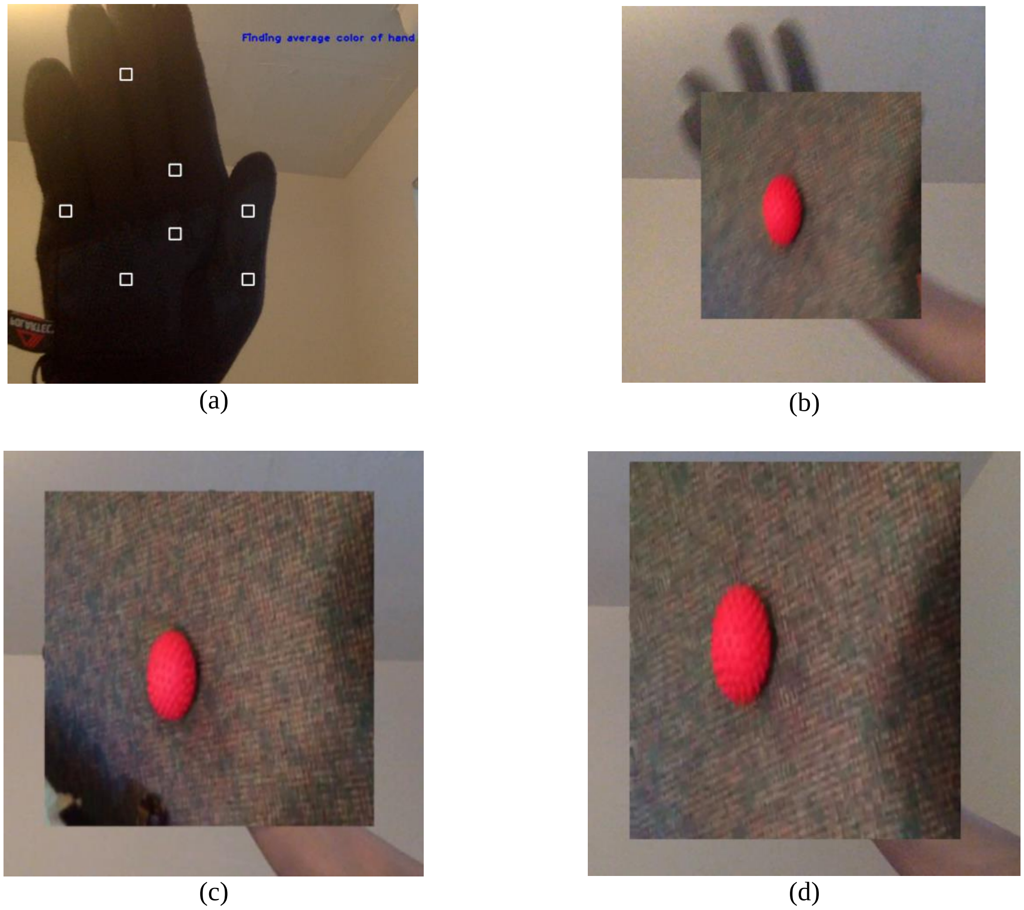


(a) (b)

Fig. 14. 3D rendering in real time with marker pose detection

D) Markerless 3D AR using color based thresholding:

The results of the Markerless 3D AR have been shown in fig. 5. Some additional results for markerless AR for hand detection and 2D embedding are shown in fig. 15. The images show the initial stage where the color to be observed is sampled, and subsequent images where a 2D image is warped and embedded on a bounding box of the detected hand.



(a) (b)

(c) (d)

Fig. 15. Markerless AR with 2D video embedding (a) Hand color sampling (b), (c), (d) embedded video

IV. DISCUSSION AND CONCLUSION

We have managed to achieve all the goals we have set and have presented the results. There are however limitations to our approaches. A limitation with a few of our techniques was due to lighting issues. The occlusion detection and hand detection algorithms rely heavily on color detection, and color is modified with poor light intensity. However, we were able to tweak parameters so that it works well under normal room lighting conditions.

For our basic goal of implementing an AR system which detects a marker and replaces it with an image, we were able to achieve consistent results. We tested this code with a variety of marker angles, and the marker was generally detected correctly. While the code works quite well, issues may arise if the scene lighting is too poor, as the algorithm relies on being able to distinguish black squares in the marker from the white ones. There are other techniques which remove this limitation, but we did not have time to implement such a technique.

Next, we modified this code so that it works real time in a camera feed. The marker was still detected consistently and the marker was replaced on the screen in real time. We observed that the real time performance of the algorithm showed a small amount of lag. However, we do not feel that this algorithm can be sped up any further without modifying OpenCV source code as the major time taken was at two functions which were inbuilt in OpenCV, namely the warping and the image merging algorithms.

Another thing that we observed was that when we moved the camera feed slightly too fast for the code to detect the marker again, the camera sometimes loses track of the marker, possibly due to blur. We fixed this by calculating the marker velocity using a linear fit as mentioned earlier. However, this could be better improved by using a quadratic fit or other methods that allow us to predict movement of the camera based on past movement.

Our next major goal was to implement occlusion using foreground detection and color-based detection. The method to observe an occluded marker worked very well, and our algorithm was still real time. As for the occlusion handling, as mentioned earlier, foreground detection did not work well. We then moved on to our color-based detection, which works much better. However, a major issue that we faced with the color-based detection was lighting which causes the white to be not 'white' enough and black to be not 'black' enough. As such, we had to tweak the thresholds of what is considered as black and white under different lighting conditions. Also, when using a single light source to shine onto the marker, this created shadows which also distorts the final result. An improvement to our occlusion code could be to use the camera to detect the lighting condition and automatically determine the thresholds. A smoothing technique such as a median filter may also remove stray errors.

Next, we embedded a 3D object onto the object. The final code works smoothly without many problems. As seen in the uploaded videos, despite the 3D rendering the code is still real time.

Our last goal was to implement a hand-detection code which acts as a marker for either a video or 3D object. Lighting was also an issue here since we had to track the color of the user's hand. Under different lighting, the code results varying results. However, to prove the concept, we wore a black glove for our demos and this allows the code to consistently detect the hand. Another limitation of our code was that the hand detected was only covered with a bounding box of largest magnitude. This meant that the orientation of the hand was never truly determined by the code, as the rotation of the hand could not be accounted for.

V. MEMBER ROLES

Shashank

- Understanding how the markers are identified in AR applications
- Understanding and modifying code based on ARToolKit[10]
- Worked on the estimation of homographies and developing methodology to merge the homography of the inner video frame into the outer one
- Developed occlusion marker and detection

Cheng Han

- Worked on embedding a video into a camera feed real time using OpenCV
- Code integration with Shashank (converting image based AR to real time video based AR)

Kuang-Huei

- Object and gesture recognition
- Interaction with 3D AR objects
- Rendering 3D objects real-time

VI. REFERENCES

[1] R. Silva et al., "Introduction to Augmented Reality", Technical Report: 25/2003, LNCC, Brazil, 2003.

[2] R. T. Azuma, "A Survey of Augmented Reality", Presence: Teleoperators and Virtual Environments 6, 4 (August 1997), 355 - 385.

[3] Brooks, Frederick P. Jr., "The Computer Scientist as Toolsmith II". CACM 39, 3 (March 1996), 61-68.

[4] Web link: http://www.prweb.com/releases/blippar-augmented-reality/tesco/prweb8833643.htm. Accessed: 03/14/2014.

[5] Web link: https://www.layar.com/glass/. Accessed: 05/12/2014.

[6] Web link: http://mashable.com/2009/08/27/yelp-augmented-reality/. Accessed: 05/12/2014.

[7] Lepetit, Vincent, “On computer vision for augmented reality’, Proceedings of the International Symposium on Ubiquitous Virtual Reality. 2008.
[8] Takacs, Gabriel, et al. "Outdoors augmented reality on mobile phone using loxel-based visual feature organization." Proceedings of the 1st ACM international conference on Multimedia information retrieval. ACM, 2008.
[9] Lee, Taehee, and Tobias Hollerer. "Handy AR: Markerless inspection of augmented reality objects using fingertip tracking." Wearable Computers, 2007 11th IEEE International Symposium on. IEEE, 2007.
[10] Kato, H.; Billinghurst, M., "Marker tracking and HMD calibration for a video-based augmented reality conferencing system," Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on , vol., no., pp.85,94, 1999. doi: 10.1109/IWAR.1999.803809
[11] Web link: https://github.com/MasteringOpenCV/code. Accessed: 05/12/2014.
[12] Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. "Epnp: An accurate o (n) solution to the pnp problem." International journal of computer vision 81.2 (2009): 155-166.
[13] Web link: http://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/. Accessed on: 03/18/2014
[14] Zarit, Benjamin D., Boaz J. Super, and Francis KH Quek. "Comparison of five color models in skin pixel classification." Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, 1999. Proceedings. International Workshop on. IEEE, 1999.
[15] Web link: http://en.wikipedia.org/wiki/HSL_and_HSV. Accessed on: 03/18/2014