Handy AR: 3D Model

Jiwei Yu jy2268 Yue Jing yj1142 Yating Yu yy2085 Yiran Yan yy2147

Abstract

Augmented reality is the integration of digital information with the user's real time environment. The objective is when user moving his hand in the front of camera, the 3D model will be placed at right places such as the center of palm with correct angles (using rotation and translation values). To show the 3D Model using AR, we need to use image processing and camera calibration of chessboard and hand. Image processing can help us get the required image points which can help us build matrix and get the position to put 3D model. Using camera calibration of chessboard we can get the intrinsic parameters and distortion coefficient. Camera calibration of hand is using for getting the extrinsic parameters. After get all these parameters, we project our matrix to OpenGL and show our image.

Keyword —— Augmented reality, Image processing, Camera calibration

I. Introduction

"Augmented reality (AR) is a live direct or indirect view of a physical, real-world environment whose elements are augmented (or supplemented) by computer-generated sensory input such as sound, video, graphics or GPS data. [1]" The AR applications first used for military, industrial and medical direction. As time goes by, the utilization of Augmented reality extend to education and people's live." From the first virtual yellow line marker in live NFL games to assisting NASA flight simulations - augmented reality has a progressive impact over the years[2]". There are a lot of computer vision related knowledge used in Augmented Reality such as optical flow, image processing, camera calibration and etcs.

II. Motivation & Related Work

The reason we choose Handy AR: 3D Model as our project is because we want to have more interaction between game and game players. Now there are a lot of card game such as Hearth Stone. In order to let the game become more attractive, they beautify their game interface and show their cards in 3D models. Then can we use handy AR to improve those games? Using computer's camera, phone's camera or AR tools like google glass, we can let game player draw cards directly use their hands or show 3D card model in the palm of their hand.

In the process of setup our project, we read some related papers using different methods to solve handy AR are really helpful. After comparing the advantages and disadvantages of python, C++ and matlab, we choose python 2.7, OpenCV 2.4, OpenGL, pygame, PIL and etcs as tools to start our project. In order to satisfy the requirement of putting 3D model in our hand, the first part we need to do is tracing the position of our figure. There are two method we can use to trace the position of figure, one is optical flow and another is image processing. Because using optical flow is hard to find the boundary of hand, we choose image processing for tracking the moving objects. Next step, we use chessboard calibration to get the intrinsic matrix and distortion coefficient, and use hand calibration to get extrinsic parameters. Using these parameters we can project our matrix to OpenGL and show the 3D model in the palm of our hand.

III. Data & Algorithm

i) Image processing

We use our computer camera to capture our hand image. First we setup the camera and capture the background without our hand.

```
while self.camera.isOpened():
    ret, frame = self.camera.read()
    frame = cv2.bilateralFilter(frame, 5, 50, 100)  # smoothing filter
    frame = cv2.flip(frame, 1)  # flip the frame horizontally
    cv2.rectangle(frame, (int(self.cap_region_x_begin * frame.shape[1]), 0),
                  (frame.shape[1], int(self.cap_region_y_end * frame.shape[0])), (255, 0, 0), 2)
    # half of the original frame for opengl operation
    halfframe = frame[0:int(self.cap_region_y_end * frame.shape[0]),
                int(self.cap_region_x_begin * frame.shape[1]):frame.shape[1]]
    cv2.imshow('original', frame)
```

After capture the background, we put our hand in and remove the background. This shep can help us reduce the interference of background objects and capture the contour of hand more accurate.

```
if self.isBgCaptured == 1:  # this part wont run until background captured
    img = self.handDetect.removeBG(frame)
    img = img[0:int(self.cap_region_y_end * frame.shape[0]),
          int(self.cap_region_x_begin * frame.shape[1]):frame.shape[1]]  # clip the ROI
```

```
def removeBG(self, frame):
    fgmask = self.bgModel.apply(frame)
    # kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    # res = cv2.morphologyEx(fgmask, cv2.MORPH_OPEN, kernel)
    kernel = np.ones((3, 3), np.uint8)
    fgmask = cv2.erode(fgmask, kernel, iterations=1)
    bg = cv2.bitwise_and(frame, frame, mask=fgmask)
    return bg
```

Then we can convert image into binary image. We use cv2.cvtColor to get grayscale image, and smooth image by using cv2.GaussianBlur.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (self.blurValue, self.blurValue), 0)

ret, thresh = cv2.threshold(blur, self.threshold, 255, cv2.THRESH_BINARY)
```

Next step, we get hand contours using cv2.findContours. We can find the required contour after compare the length of all contours and get the longest one. Then we get convex hull of our hand using contour and cv2.convexHull.

```
thresh1 = copy.deepcopy(thresh)
contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
length = len(contours)
maxArea = -1
if length > 0:
    for i in range(length):  # find the biggest contour (according to area)
        temp = contours[i]
        area = cv2.contourArea(temp)
        if area > maxArea:
            maxArea = area
            maxi = i
    contour = contours[maxi]
    hull = cv2.convexHull(contour)
    drawing = np.zeros(img.shape, np.uint8)
    cv2.drawContours(drawing, [contour], 0, (0, 255, 0), 2)
    cv2.drawContours(drawing, [hull], 0, (0, 0, 255), 3)
```

Using cv2.drawContours, we can see the image of our hand with contours and convex hull clearly. Shown in Figure 1-1 Hand Contour and Convex Hull

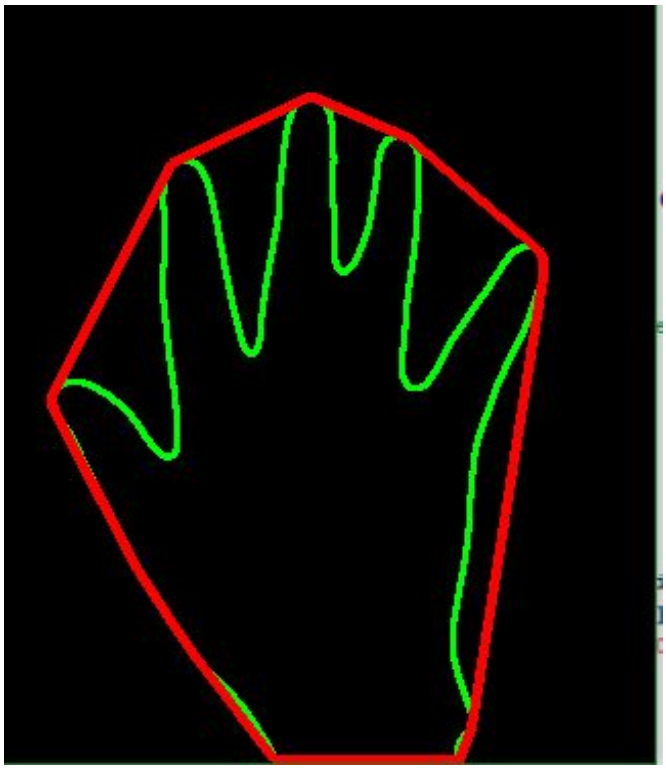


Figure 1-1 Hand Contour and Convex Hull

After getting the hand contour and convex hull, we need to detect our figures and find image points which can help us determine the position of 3D model and place it in right place (our palm center).

If the length of hull larger than 3, we Find the convexity defects of the contour

```
hull = cv2.convexHull(res, returnPoints=False)
if len(hull) > 3:
    defects = cv2.convexityDefects(res, hull)
    if not (isinstance(defects, type(None))):
        count = 0
        p = []
```

We use cosine theorem to calculate angle

```
for i in range(defects.shape[0]):
    s, e, f, d = defects[i][0]
    start = tuple(res[s][0])
    end = tuple(res[e][0])
    far = tuple(res[f][0])
    a = math.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1]) ** 2)
    b = math.sqrt((far[0] - start[0]) ** 2 + (far[1] - start[1]) ** 2)
    c = math.sqrt((end[0] - far[0]) ** 2 + (end[1] - far[1]) ** 2)

    angle = math.acos((b ** 2 + c ** 2 - a ** 2) / (2 * b * c))
```

If the angle less than 90 degree we treat it as figures. For each loop, we find start point, end point and far point, then draw a circle on the point. Shown in Figure 1-2 figure points

```
if self.doStart == True and angle <= math.pi / 2:
    count += 1
    p.append(start)
    p.append(end)
    p.append(far)
    cv2.circle(drawing, start, 8, [200, 200, 0], -1)
    cv2.circle(drawing, end, 8, [0, 0, 255], -1)
    cv2.circle(drawing, far, 8, [211, 84, 0], -1)
```

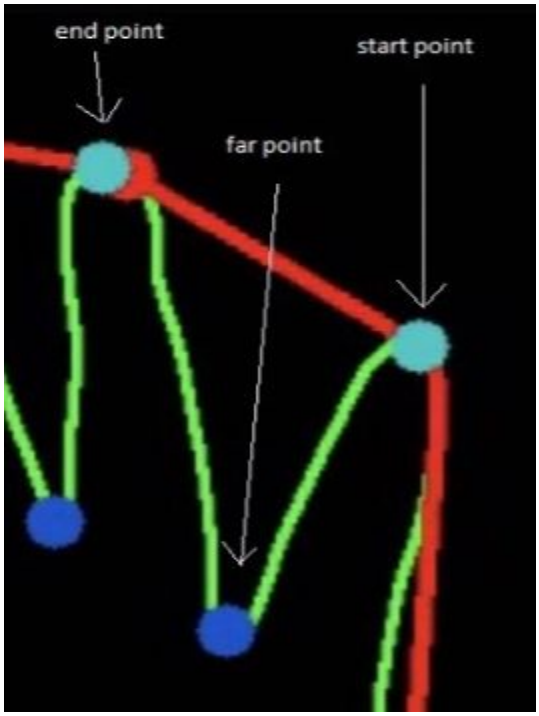


figure 1-2 figure points

Save those points as image points

```
imagepoints = np.array([p], dtype=np.float32)
```

After we get 12 image points, we start the camera calibration

```
if imagepoints.shape[1] == 12:
    return imagepoints
```

ii) Camera calibration

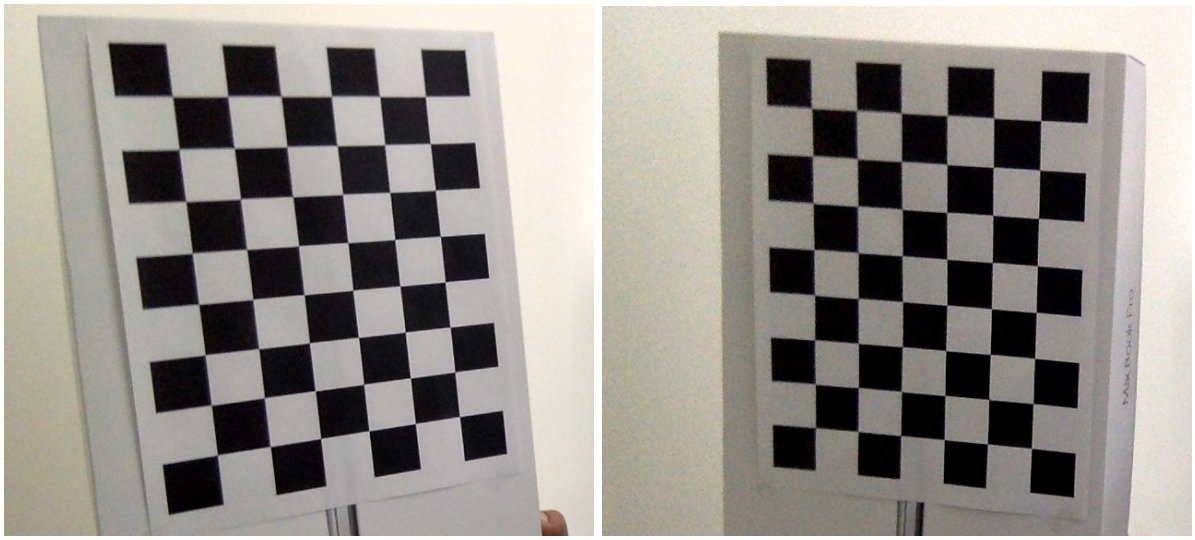
In the beginning of our calibration, we first use cheese board to get intrinsic matrix and distortion.

Prepare objects point like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0) and create arrays to store object point and image points from all the images.

```
objp = np.zeros((6*7,3), np.float32)
objp[:,:2] = np.mgrid[0:7,0:6].T.reshape(-1,2)

objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
```

Use camera to get chessboard images from different angles. (we use 30 images see example images in Figure 2-1 Chessboards)



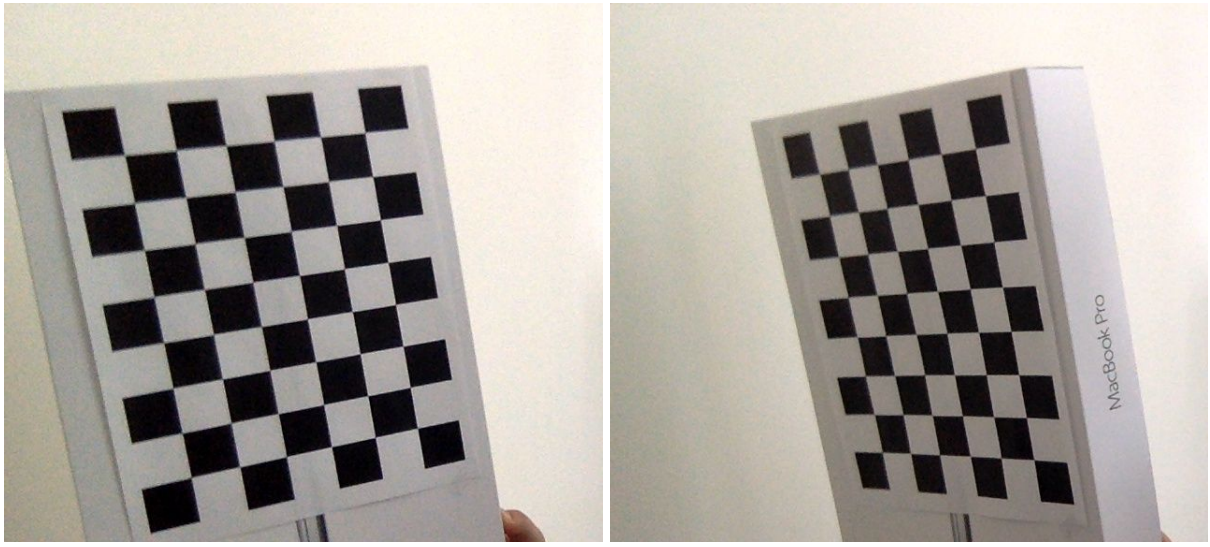


Figure 2-1 Chessboards

Then read images and use cv2.cvtColor to get grayscale image

```
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

Using cv2.findChessboardCorners, we can find the corner of the chessboard.

```
ret, corners = cv2.findChessboardCorners(gray, (7,6),None)
```

If the corner has been found, we add object points and image point after refining them. cv2.cornerSubPix can help us determine positions more accurately.

```
if ret == True:
    objpoints.append(objp)
    #determine positions more accurately
    cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
    imgpoints.append(corners)
```

Draw and display the corners and save the chessboard images after calibration. Shown in Figure 2-2 Chessboard calibration

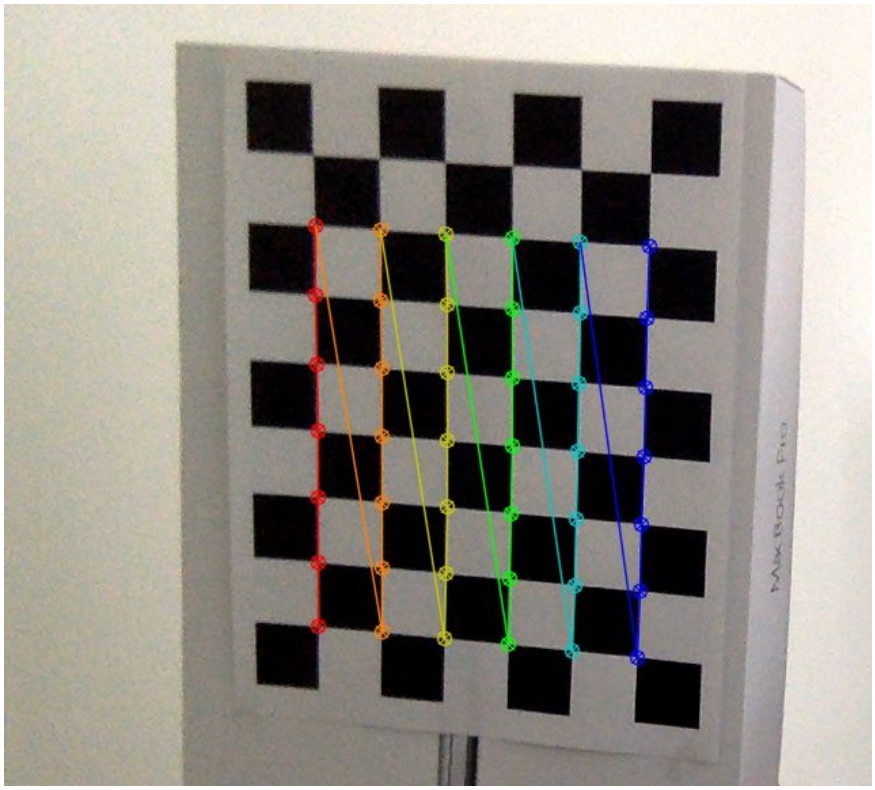


Figure 2-2 Chessboard calibration

```
cv2.drawChessboardCorners(img, (7,6), corners,ret)
cv2.imshow('img',img)
cv2.waitKey(500)
objpoints = np.array(objpoints) * gridLength
imgpoints = np.array(imgpoints)#.reshape(1,42,2)
```

Find the camera intrinsic and extrinsic parameters and distortion coefficients using cv2.calibrateCamera.

```
ret, matrix, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, (gridLength*7,gridLength*6), None, None)
```

The value of intrinsic matrix shown in Figure 2-2 Intrinsic Matrix and the value of distortion coefficients shown in Figure 2-3 Distortion Coefficients

```
1.958781230624130330e+03 0.000000000000000000e+00 6.992906565716300804e+01
0.000000000000000000e+00 2.073355261064662045e+03 1.015638012823010854e+02
0.000000000000000000e+00 0.000000000000000000e+00 1.000000000000000000e+00
```

Figure 2-3 Intrinsic Matrix

```
1.13868846595200412e-01 3.31961487902903984e+00 -1.90836965482237518e-02 -1.03612444658246996e-01 -2.44293257751027859e+01
```

Figure 2-4 Distortion Coefficients

Save intrinsic matrix and distortion coefficients and we will use them in the following calibration

```
np.savetxt('./intrinsic/matrix.out', matrix)
np.savetxt('./intrinsic/dist.out', dist)
```

Next step, in order to calibration our hand, we need to get the world object points (hand points)

```
objectpoints = np.array([[[-6.4, 7.5, 0], [-3.4, 10.7, 0], [-2.8, 3.5, 0],
                          [-3, 11, 0], [-0.3, 11.7, 0], [-0.8, 4.7, 0],
                          [0.3, 11.7, 0], [4.0, 10, 0], [1.3, 4.5, 0],
                          [4.7, 10, 0], [7.7, 3.4, 0], [3.6, 1, 0]]], dtype=np.float32)
```

Get the initial of required parameters. The intrinsic matrix and distortion coefficients get from previous chessboard calibration.

```
def __init__(self):
    self.MTX = np.loadtxt('./intrinsic/matrix.out', dtype=np.float32)
    self.DIST = np.loadtxt('./intrinsic/dist.out', dtype=np.float32)
    self.RVEC = [] # rotation
    self.TVEC = [] # translation
    self.frame = None
```

Get grayscale images and use solvePnP to get extrinsic parameters

```
def calibrate(self, imagepoints, drawing, frame):
    self.frame = drawing
    gray = cv2.cvtColor(drawing, cv2.COLOR_BGR2GRAY)

    ret, rvecs, tvecs = cv2.solvePnP(self.objectpoints, imagepoints, self.MTX, self.DIST)
```

Then we can draw a 3D pixels in the center of our hand. Shown in Figure 2-5 3D Pixels

```
def drawAxis(self, img, axispts):
    center = tuple(axispts[3].ravel())
    cv2.line(img, center, tuple(axispts[0].ravel()), (255, 0, 0), 5)
    cv2.line(img, center, tuple(axispts[1].ravel()), (0, 255, 0), 5)
    cv2.line(img, center, tuple(axispts[2].ravel()), (0, 0, 255), 5)
```

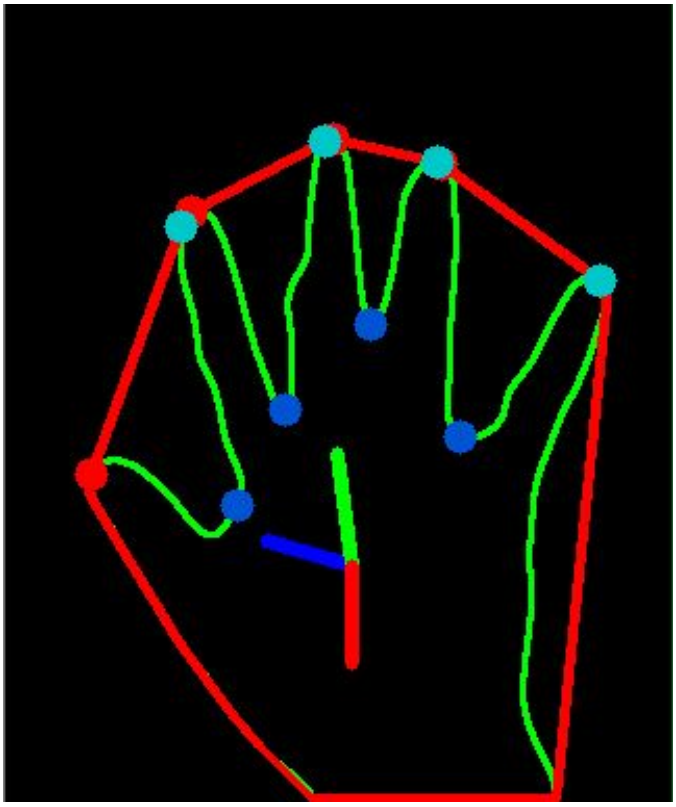


Figure 2-5 3D Pixels

Or build a pyramid in the center of our hand. Shown in Figure 2-6 Pyramid

```
axis = np.float32(
    [[-2, -2, 0], [-2, 2, 0], [2, 2, 0], [2, -2, 0], [0, 0, -4]])
axispoints, _ = cv2.projectPoints(axis, rvecs, tvecs, self.MTX, self.DIST)
self.drawAxis(drawing, axispoints, frame)
self.RVEC = rvecs
self.TVEC = tvecs
```

```
def drawAxis(self, img, axispts, frame):
    imgpts = np.int32(axispts).reshape(-1, 2)

    cv2.drawContours(img, [imgpts[:4]], -1, (0, 100, 255), 4)
    for i in (range(4)):
        cv2.line(img, tuple(imgpts[i]), tuple(imgpts[4]), (0, 100, 0), 4)

    for i in range(0,5):
        imgpts[i][0] = imgpts[i][0] + 0.5 * frame.shape[1]

    cv2.drawContours(frame, [imgpts[:4]], -1, (0, 100, 255), 4)
    for i in (range(4)):
        cv2.line(frame, tuple(imgpts[i]), tuple(imgpts[4]), (0, 100, 0), 4)
    cv2.imshow('original', frame)
```

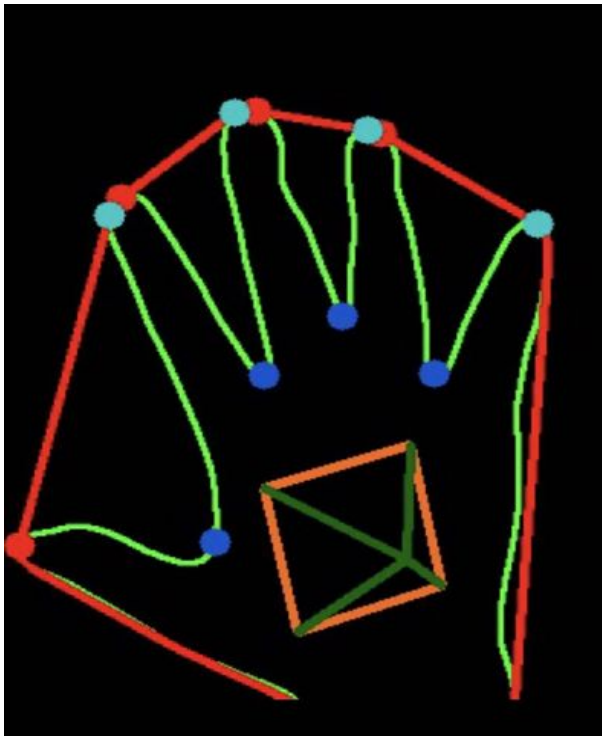


Figure 2-6 Pyramid

After get intrinsic and extrinsic parameters, we can use function to get projection matrix in OpenGL

```
def getGLP(self):
    height = self.frame.shape[0]
    width = self.frame.shape[1]
    P = np.zeros(shape=(4, 4), dtype=np.float32)

    fx = self.MTX[0, 0]
    fy = self.MTX[1, 1]

    cx = self.MTX[0, -1]
    cy = self.MTX[1, -1]

    near = 0.1
    far = 100.0

    P[0, 0] = 2 * fx / width
    P[1, 1] = 2 * fy / height
    P[0, 2] = 1 - (2 * cx / width)
    P[1, 2] = (2 * cy / height) - 1
    P[2, 2] = -(far + near) / (far - near)
    P[3, 2] = -1.
    P[2, 3] = -(2 * far * near) / (far - near)

    p = P.T
    return p.flatten()
```

And get the view model using numpy

```
def getGLM(self):
    R, _ = cv2.Rodrigues(self.RVEC)
    Rt = np.hstack((R, self.TVEC))
    Rx = np.array([[1, 0, 0], [0, -1, 0], [0, 0, -1]])
    M = np.eye(4)
    M[:3, :] = np.dot(Rx, Rt)

    m = M.T
    return m.flatten()
```

iii) Draw 3D object using OpenGL

To get the 3D model in OpenGL, first we need to get the initial parameters and initial pygame

```
def __init__(self, frame, glP, glM):
    self.frame = frame
    self.glP = glP
    self.glM = glM
    self.image = None
    # Init pygame.
    pygame.init()
    self.screen = pygame.display.set_mode((427, 384),pygame.OPENGLBLIT | pygame.DOUBLEBUF)
    pygame.display.set_caption('pygame')
```

Then we need to draw background, set projection matrix, set model view and load 3D model.

```
def draw(self):
    drawBackground(self.frame)
    setProjection(self.glP)
    setModelview(self.glM, 0.2)
    load_and_draw_model()
```

Draw background

```
def drawBackground(frame):
    bg_image = pygame.surfarray.make_surface(frame)
    bg_image = pygame.transform.rotate(bg_image, 90.0)
    bg_data = pygame.image.tostring(bg_image, 'RGBA', False)
    width, height = bg_image.get_size()

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    glEnable(GL_TEXTURE_2D)
    glGT = glGenTextures(1)
    glBindTexture(GL_TEXTURE_2D, glGT)
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, bg_data)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)

    glBegin(GL_QUADS)
    glTexCoord2f(0.0, 0.0);
    glVertex3f(-1.0, -1.0, -1.0)
    glTexCoord2f(1.0, 0.0);
    glVertex3f(1.0, -1.0, -1.0)
    glTexCoord2f(1.0, 1.0);
    glVertex3f(1.0, 1.0, -1.0)
    glTexCoord2f(0.0, 1.0);
    glVertex3f(-1.0, 1.0, -1.0)
    glEnd()
    glDeleteTextures(1)
```

g

Set projection matrix

```
def setProjection(glP):
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    glLoadMatrixf(glP)
```

Set model view

```
def setModelview(glM, scale=1.):
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()
    glLoadMatrixf(glM)
    glTranslate(0.5, 0.5, -1)
    glRotate(180, 0, 1, 0)
    glRotate(180, 0, 0, 1)
    glScalef(scale, scale, scale)
```

We can load our 3D model using file objloader.py from pygame wiki

```
def load_and_draw_model():
    glLightfv(GL_LIGHT0, GL_POSITION, (-50, 200, 250, 0.0))
    glLightfv(GL_LIGHT0, GL_AMBIENT, (0.2, 0.2, 0.2, 1.0))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (0.5, 0.5, 0.5, 1.0))
    glEnable(GL_LIGHT0)
    glEnable(GL_LIGHTING)
    glEnable(GL_COLOR_MATERIAL)
    glEnable(GL_DEPTH_TEST)
    glShadeModel(GL_SMOOTH)
    obj = OBJ('./obj/skyscraper.obj', swapyz=True)
    glCallList(obj.gl_list)
```

Get image buffer from OpenGL buffer

```
buffer = glReadPixels(0, 0, 640, 576, GL_RGBA, GL_UNSIGNED_BYTE)
self.image = Image.frombuffer(mode="RGBA", size=(640,576), data=buffer)
```

Use np.asarray to convert image buffer to array and show the image. Shown in Figure 2-7 Handy AR

```
img = np.asarray(self.image)
cv2.imshow('Handy AR', img)
```

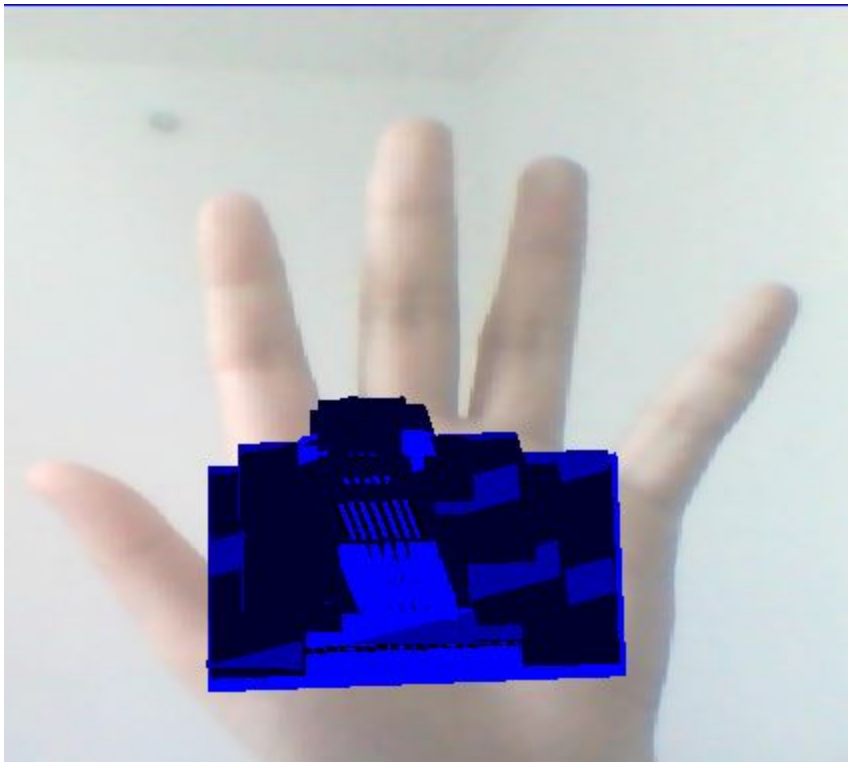


Figure 2-7 Handy AR

IV. Challenges & Conclusion

After running our program, we can press b to capture the background, press s to start calibration, and press r to reset the background.

```
# Keyboard OP
k = cv2.waitKey(10)
if k == 27:  # press ESC to exit
    exit(0)
elif k == ord('b'):  # press 'b' to capture the background
    self.handDetect.bgModel = cv2.BackgroundSubtractorMOG2(0, self.bgSubThreshold)
    self.isBgCaptured = 1
    print('Background Captured')
elif k == ord('r'):  # press 'r' to reset the background
    self.handDetect.bgModel = None
    self.isBgCaptured = 0
    self.handDetect.setDoStart(False)
    print('Reset BackGround')
elif k == ord('s'):
    self.handDetect.setDoStart(True)
    print('Start Calibration')
```

The result we get: when we change the positions and angles of our hand, the 3D model can be located at right place (center of our palm), and shown in different angles because of the changing of rotation and translation value. Shown in Figure 3-1 Handy AR: 3D Model

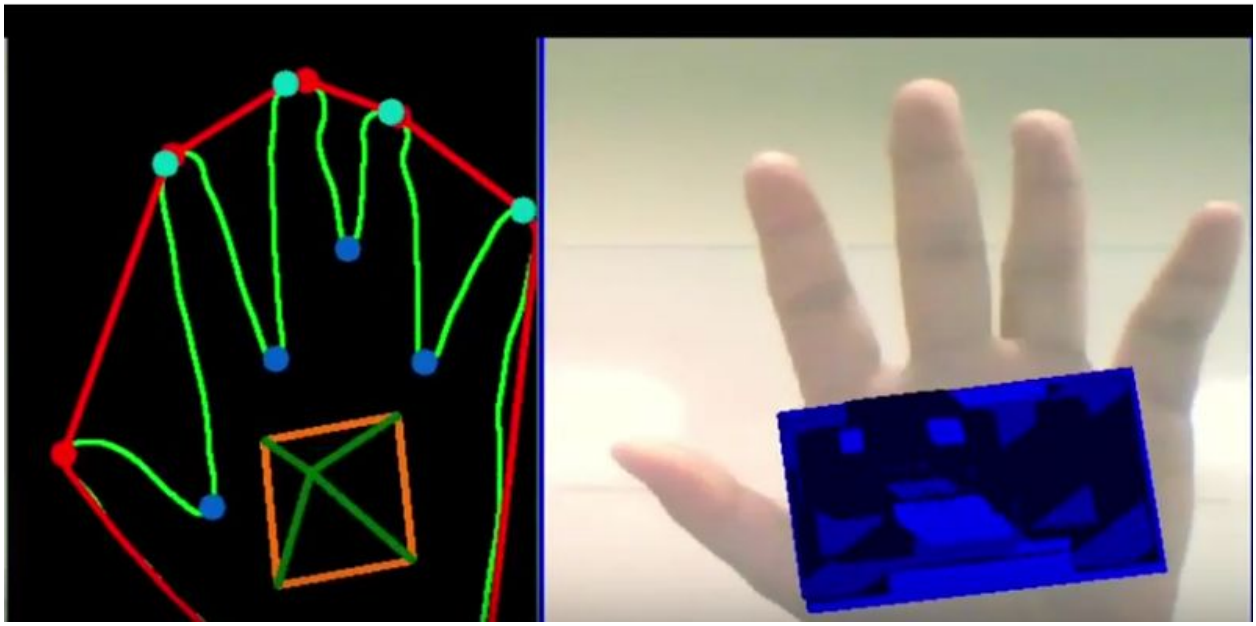


Figure 3-1 Handy AR: 3D Model

Although the results we get are performing as we expected, we still have a lot of challenges. The capture of our figures was easily affected by the background such as light, shadow and other moving objects. Since we need to capture 12 image points to start out hand calibration, if we turn around our hand or show the side of our hand, the 3D model cannot be shown correctly. To improve our projects we can both improve our algorithm to recognize more hand gestures and develop more 3D objects to satisfy game needs.

Video: https://www.youtube.com/watch?v=e5Ue92W3JFg

Contribution:

#	Task	Owner
1	Subject Research and Analysis	Yating Yu, Yue Jing, Yiran Yan, Jiwei Yu

2	Environment Setup	Yating Yu, Yue Jing, Yiran Yan, Jiwei Yu
3	Research about Opencv, OpenGL, pygame	Yating Yu, Yue Jing
4	Image Processing	Yiran Yan, Jiwei Yu
5	Show model in OpenGL	Yue Jing, Yating Yu
6	Test and Improve	Jiwei Yu, Yue Jing
7	Write Presentation PPT	Yiran Yan, Yating Yu
8	Write Project Report	Yating Yu, Yue Jing, Yiran Yan, Jiwei Yu

V. Reference

[1] "Augmented Reality", Wikipedia, the free encyclopedia. Available at https://en.wikipedia.org/wiki/Augmented_reality
[2] "Infographic: The History of Augmented Reality". Available at

http://www.augment.com/blog/infographic-lengthy-history-augmented-reality/
[3] Markerless Inspection of Augmented Reality Objects Using Fingertip Tracking
https://www.cs.ucsb.edu/~holl/pubs/Lee-2007-ISWC.pdf
[4] Camera Calibration with Opencv
http://docs.opencv.org/2.4/doc/tutorials/calib3d/camera_calibration/camera_calibration.html
[5] Model Load with Opengl:
http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/
[6] OBJFileLoader - wiki. Available at
https://www.pygame.org/wiki/OBJFileLoader