Task 1.2 - Problem Statement

Welcome to Task 1.2 !!
In this task you will get familiar with the basics of openGL and how to use it for augmented reality. Similar to Task 1.1, you will be required to complete a set of functions to make your code work.

Expected Output

In this task, you will be required to complete predefined functions in a python file. The python file should do the following things when executed:

1. Identify ArUco markers appearing in Webcam feed.
2. Calculate position and orientation of ArUco markers
3. Overlay 3D model of a teapot on the ArUco markers.

An example of the output is given at this **link**

Main Function and Initialisations

Go to *Problem Statement* folder in Task 1.2 and open the *GLteapot.py* python file.
We will now explain the parts of the code to help you modify the functions.

MAIN CODE

```
def main():
        glutInit()
        getCameraMatrix()
        glutInitWindowSize(640, 480)
        glutInitWindowPosition(625, 100)
        glutInitDisplayMode(GLUT_RGB|GLUT_DEPTH|GLUT_DOUBLE)
        window_id = glutCreateWindow("OpenGL")
        init_gl()
        glutDisplayFunc(drawGLScene)
        glutIdleFunc(drawGLScene)
        glutReshapeFunc(resize)
        glutMainLoop()
```

```
def init_gl():
        global texture_object, texture_background
        glClearColor(0.0, 0.0, 0.0, 0.0)
        glClearDepth(1.0)
        glDepthFunc(GL_LESS)
        glEnable(GL_DEPTH_TEST)
        glShadeModel(GL_SMOOTH)
        glMatrixMode(GL_MODELVIEW)
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_LIGHTING)
        glEnable(GL_LIGHT0)
        texture_background = glGenTextures(1)
        texture_object = glGenTextures(1)

def resize(w,h):
        ratio = 1.0* w / h
        glMatrixMode(GL_PROJECTION)
        glViewport(0,0,w,h)
        gluPerspective(45, ratio, 0.1, 100.0)

def drawGLScene():
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        ar_list = []
        ret, frame = cap.read()
        if ret == True:
##              draw_background(frame)
                glMatrixMode(GL_MODELVIEW)
                glLoadIdentity()
                ar_list = detect_markers(frame)
##              for i in ar_list:
##                      if i[0] == 8:
##                                              overlay(frame, ar_list,
i[0],"texture_1.png")
##                      if i[0] == 2:
##                                              overlay(frame, ar_list,
i[0],"texture_2.png")
##                      if i[0] == 7:
##                                              overlay(frame, ar_list,
i[0],"texture_3.png")
##                      if i[0] == 6:
##                                              overlay(frame, ar_list,
i[0],"texture_4.png")

                cv2.imshow('frame', frame)
                cv2.waitKey(1)
        glutSwapBuffers()
```

Firstly, we have the main() function. The first 6 lines of main() initialise and create an OpenGL window. This window displays any object placed in the OpenGL scene.

After this, the init_gl() function is called which initializes some parameters related to our OpenGL scene.

The last 4 lines of main() contain 3 callback functions and a call to glutMainLoop()

glutDisplayFunc: sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called.

glutIdleFunc: sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received.

glutReshapeFunc: sets the reshape callback for the current window.

glutMainLoop: enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

OpenGL programs typically run in an event processing loop where callback functions are called as and when required inside the loop. This event loop is started when glutMainLoop() is called.

In our code, glutReshapeFunc calls the callback function resize(). Resize function sets the **OpenGL Projection Matrix.**

glutDisplayFunc and glutIdleFunc calls the callback function drawGLScene().

In the function init_gl(), there are two global variables **texture_background** and **texture_object**. The last 2 lines of init_gl() assign 2 texture names to texture_background and texture_object using the **glGenTextures()** function

Finally we move to the callback function drawGLScene(). In this callback function the following happens in order:

a) **GL_COLOR_BUFFER_BIT** and **GL_DEPTH_BUFFER_BIT** are cleared using glClear.
b) A frame is read from the webcam using cap.read()
c) draw_background() function is called. This function converts the current webcam frame into a openGL texture and sets it as background in the OpenGL window
d) ArUco marker is detected using detect_markers() function. This function was completed in Task 1.1.
e) overlay() function is called. Overlay function applies a texture from a texture file provided in Problem Statement folder to a 3D model of a solid teapot, and overlays the teapot on the ArUco marker as per the Expected Output.

f) For different IDs of ArUco marker, different texture files are applied. All these texture files have been provided to you in the Problem Statement folder.

The drawGLScene callback function is called again and again inside the event processing loop. The event processing loop only ends when we close the OpenGL window.

The function calls for draw_background() and the overlay() function have been commented out in drawGLScene(). These functions need to be completed by you. The functions can be uncommented once you have completed the functions.

ArUco Detection

The detect_markers() function was completed in Task 1.1. You can copy the same function here. The only change here is that the camera parameters have been declared globally so you do not need to pass them as parameters to detect_markers().

```
def detect_markers(img):
        aruco_list = []
        ##################################################################
        ################ Same code as Task 1.1 ###########################
        ##################################################################
        return aruco_list
```

Functions to be completed

The following functions need to be completed by you.

```
def draw_background(img):
        ################### INSERT CODE HERE ###################

        ########################################################
        return None


def init_object_texture(image_filepath):
        glEnable(GL_TEXTURE_2D)
        tex_image = cv2.imread(image_filepath)
        ################### INSERT CODE HERE ###################

        ########################################################
        return None
```

```
def overlay(img, ar_list, ar_id, texture_file):
        for x in ar_list:
                if ar_id == x[0]:
                        centre, rvec, tvec = x[1], x[2], x[3]


        rmtx =
        view_matrix =
        view_matrix =
        view_matrix =


        init_object_texture(texture_file)
        glPushMatrix()
        glLoadMatrixd(view_matrix)
        glutSolidTeapot(0.5)
        glPopMatrix()
```

draw_background() : This function takes an image as input and converts it into an OpenGL texture. Then that texture is applied on a quadrilateral and set as background for OpenGL window.

Parameters

img: Image expressed as a numpy array.

Return

None

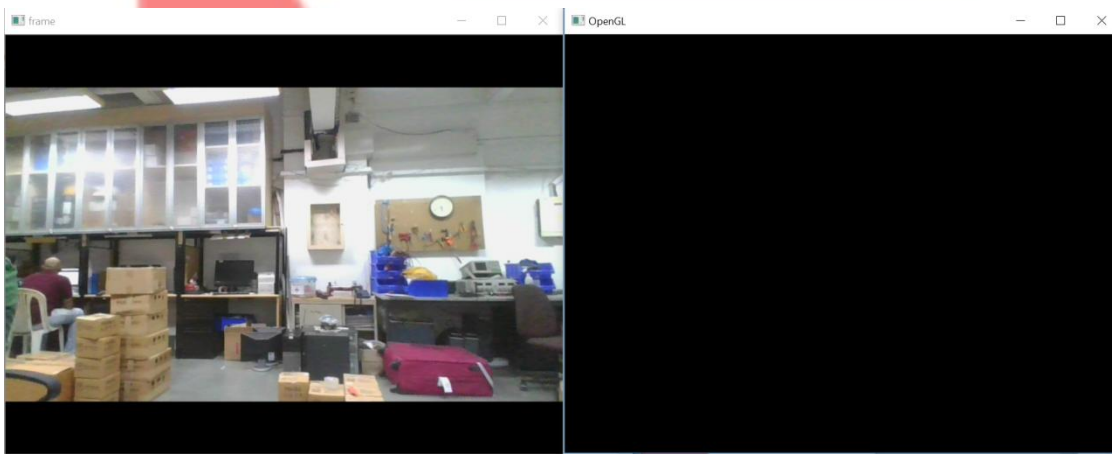

Figure 1: No Background

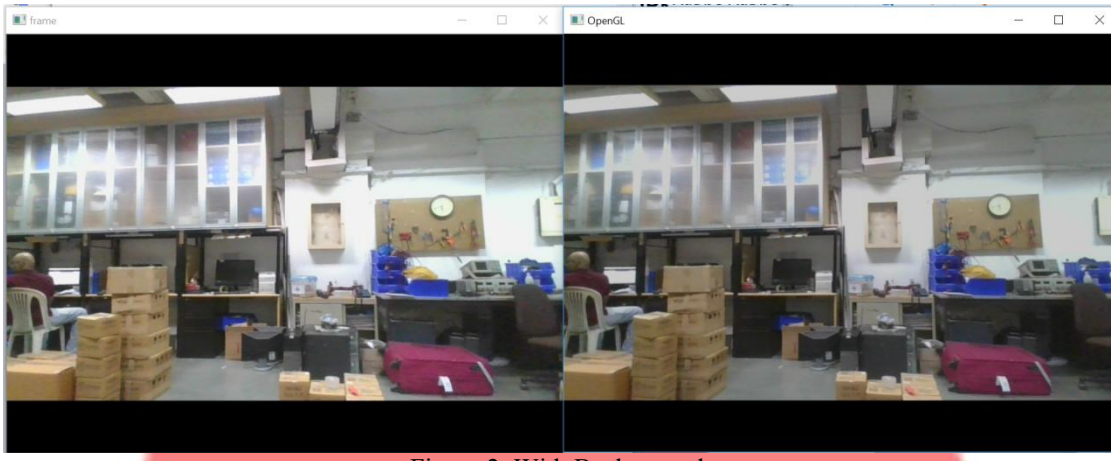

Figure 2: With Background

In Figure 1, the OpenGL window is blank. In Figure 2, the OpenGL window shows the webcam feed as background, same as the original frame.

overlay() - This function overlays the 3D model of the teapot on the ArUco marker as depicted in Figure 3.

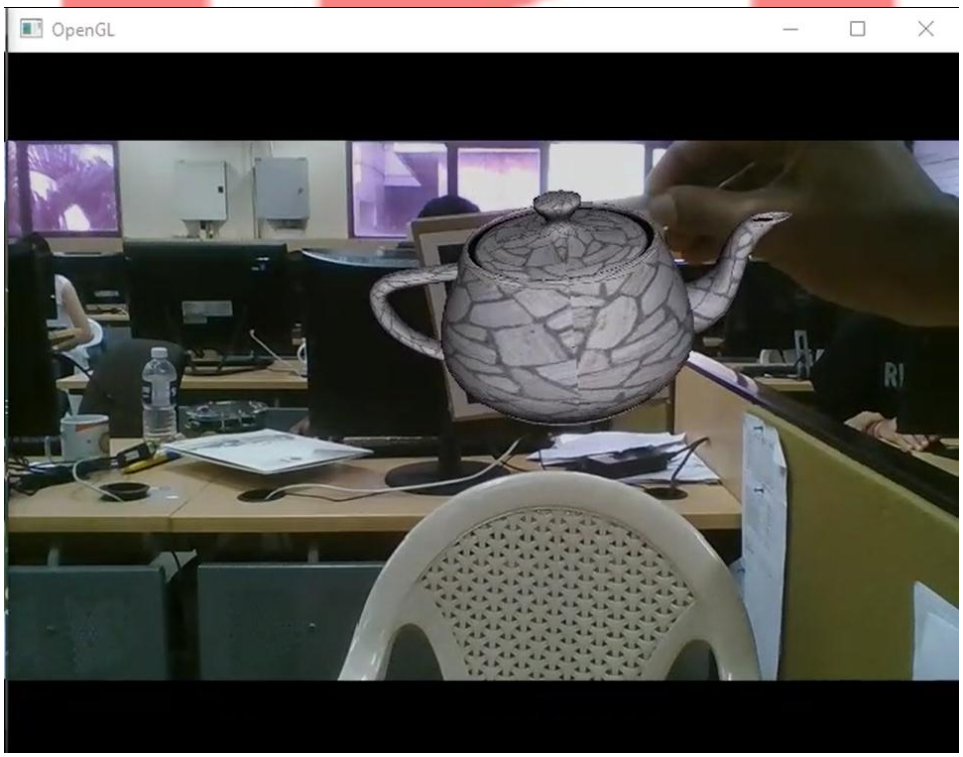

Figure 3: Teapot Overlay over ArUco marker

Parameters

img: Image expressed as a numpy array.

ar_list: ArUco list returned by detect_markers()

ar_id: ID of the ArUco marker

texture_file: Filepath of the texture files provided. Filepaths can be **texture_1.png**, **texture_2.png** etc.

Return

None

The overlay() function calculates the **rotation matrix (rmtx)** using the values of rotational vector **(rvec).** The **view matrix (view_matrix)** is calculated using values of **rotation matrix** and **translation vector(tvec).**

After these matrices are calculated, the init_object_texture() function is called which initializes the texture to be applied for the object to be rendered on screen.

The view_matrix calculated is then loaded as the current **OpenGL ModelView Matrix.** After this the glutSolidTeapot function is called which renders the 3D model of teapot on screen.

Parts of this code are already completed, you just need to fill in the blanks. You may however add your own code in this function.

init_object_texture(): This function accepts a filepath of a texture file as input and converts it into an OpenGL texture. Then it applies it to the next rendered object on the OpenGL scene.

Parameters

texture_file: Filepath of the texture files provided. Filepaths can be **texture_1.png**, **texture_2.png** etc.

Return

None

Recording Video of Solution

After you have completed the code in *GLteapot.py* and tested it successfully, you need to record a video of your solution (similar to the video provided by us).

You can use Apowersoft online screen recorder or any other screen recorder software to record the video of your submission.

Once you have created the video, do the followings steps:

1. Upload the video on Youtube. The name of the video should be **eYRC#TC#<Team_id>#Task1.2_Video.** For example if your Team_id is 2343 then the video name should be **eYRC#TC#234>#Task1.2_Video. Make the video as unlisted**.
2. Open Notepad and create a new text file. Copy and paste the Youtube video link into the text file. Save the file as **eYRC#TC#<Team_id>.**

Submission Instructions

- ✓ Create a new folder and name it as **eYRC#TC#<Team_id>#Task1.2 .** Copy your modified python file *GLteapot.py* and the text file *eYRC#TC#<Team_id>.txt* into that folder
- ✓ Convert your folder into a .zip archive using WinZip or any other software.
- ✓ Check the *Submission Instructions.pdf* for further instructions on how to upload your Task 1.