Homework #1 Report

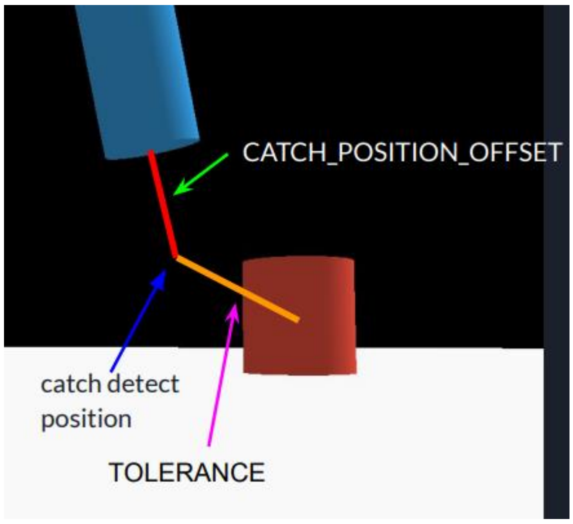
[CSIC30108] Computer Graphics 2022

Student ID: 311605004 **Name:** 劉子齊 **Date:** 2022.10.23

1. Introduction:

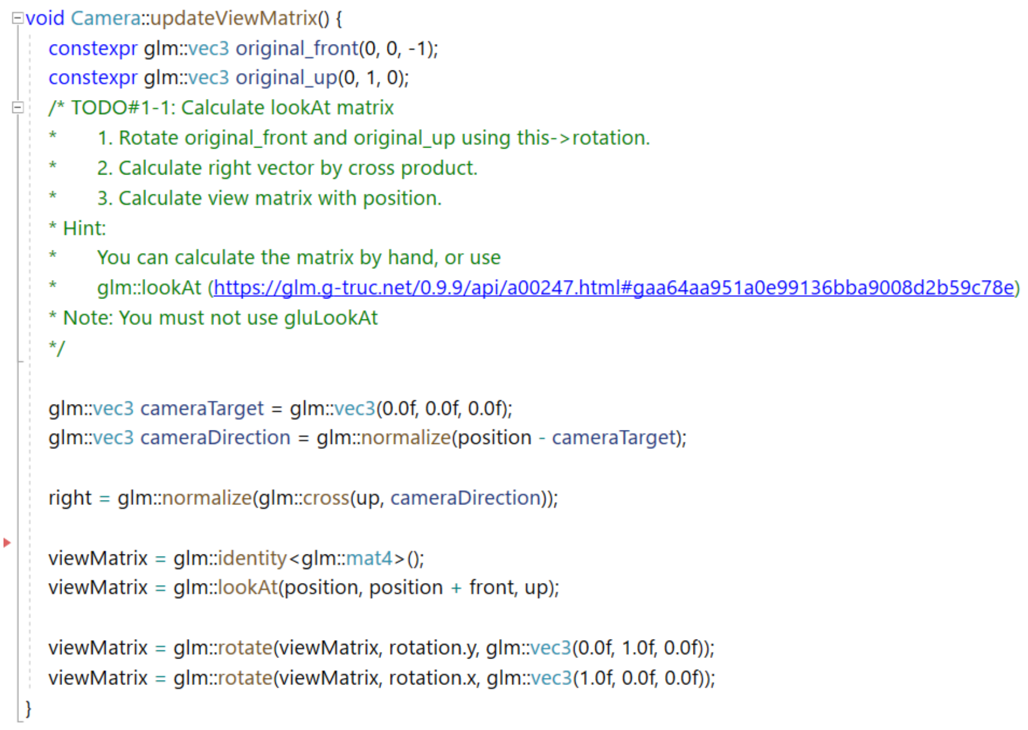
In this homework, our goal is to implement a robot arm which is capable of catching the target object through OpenGL, glfw, glm, and the architecture template given. The robot arm implemented is conposed of a base, three arms, and two joints. We should be able to turn the whole robot implemented on its base. Besides, the robot should be able to rotate its joints based on the keyboard controls of its user.

When catching the target object, as shown in the following figure, we will have a tolerance distance and a catch position offset. The catching position will be at the endpoint of the catch position offset. If the distance between the endpoint and the center of the object is smaller than the tolerance, as the user presses the space bar, the robot arm will catch the target object. If the user releases the space bar, the robot arm will release the target object at its current position, and the robot arm should be able to catch the object as the same way at the previous position which the target object was released.



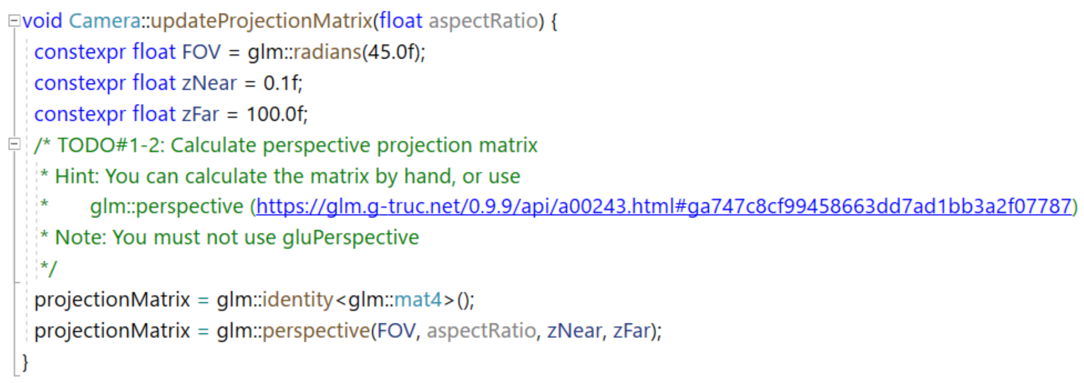
2. Implementation Details:

First of all, I started with working on the view matrix and the projection matrix. For the view matrix, I implemented it through the “lookAt” function. I first calculate the direction of the camera, then I obtain the “right” matrix through normalizing the cross product of the “up” matrix and the camera direction matrix. Afterwards, I obtain the final view matrix through the lookAt function and several rotations.



```
void Camera::updateViewMatrix() {
  constexpr glm::vec3 original_front(0, 0, -1);
  constexpr glm::vec3 original_up(0, 1, 0);
  /* TODO#1-1: Calculate lookAt matrix
   *      1. Rotate original_front and original_up using this->rotation.
   *      2. Calculate right vector by cross product.
   *      3. Calculate view matrix with position.
   * Hint:
   *      You can calculate the matrix by hand, or use
   *      glm::lookAt (https://glm.g-truc.net/0.9.9/api/a00247.html#gaa64aa951a0e99136bba9008d2b59c78e)
   * Note: You must not use gluLookAt
   */

  glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);
  glm::vec3 cameraDirection = glm::normalize(position - cameraTarget);

  right = glm::normalize(glm::cross(up, cameraDirection));

  viewMatrix = glm::identity<glm::mat4>();
  viewMatrix = glm::lookAt(position, position + front, up);

  viewMatrix = glm::rotate(viewMatrix, rotation.y, glm::vec3(0.0f, 1.0f, 0.0f));
  viewMatrix = glm::rotate(viewMatrix, rotation.x, glm::vec3(1.0f, 0.0f, 0.0f));
}
```

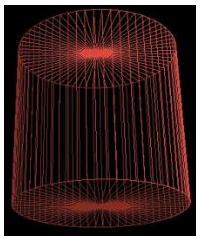
For the Projection Matrix, we update it by setting the FOV value to 45 degree, the zNear as 0.1, and the zFar as 100.



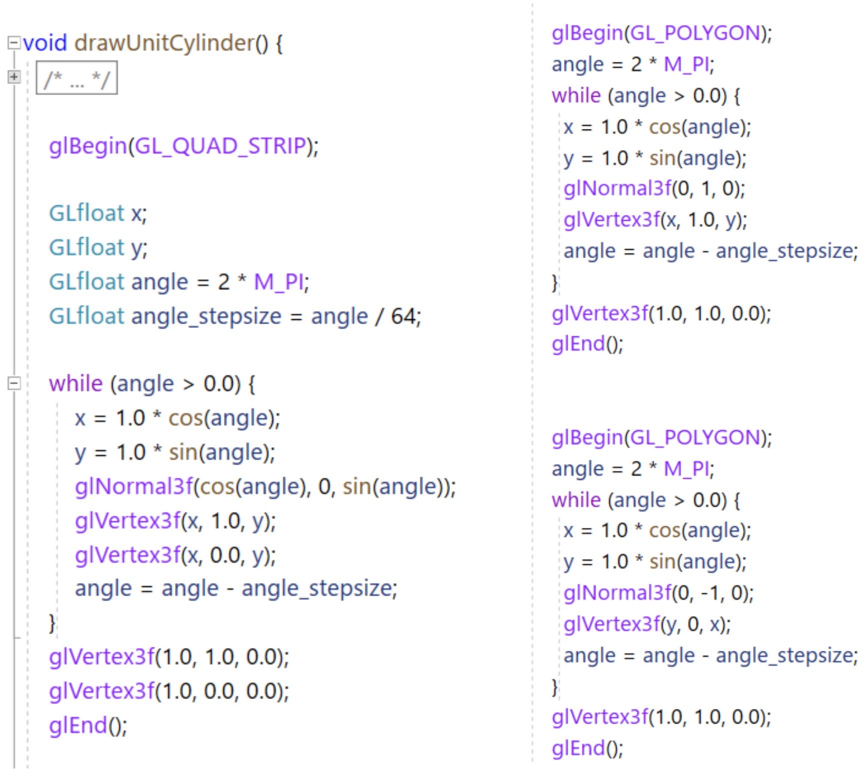
```
void Camera::updateProjectionMatrix(float aspectRatio) {
  constexpr float FOV = glm::radians(45.0f);
  constexpr float zNear = 0.1f;
  constexpr float zFar = 100.0f;
  /* TODO#1-2: Calculate perspective projection matrix
   * Hint: You can calculate the matrix by hand, or use
   *      glm::perspective (https://glm.g-truc.net/0.9.9/api/a00243.html#ga747c8cf99458663dd7ad1bb3a2f07787)
   * Note: You must not use gluPerspective
   */
  projectionMatrix = glm::identity<glm::mat4>();
  projectionMatrix = glm::perspective(FOV, aspectRatio, zNear, zFar);
}
```

After setting up the view matrix and the perspective matrix, I start with working on drawing a single unit cylinder, since the robot arm is basically composed by several cylinders.

I separated the cylinder into 3 parts, the surrounding, the top, and the bottom. I first render the surrounding by separating 2*pi into 64 parts and render 64 rectangles separately in 64 different angles around, which is shown in the following figure. Then I'll obtain the surrounding of the cylinder. After this, I render the top and the bottom of the cylinder.



Finally, I obtain a unit cylinder, which will be used to compose different parts of the robot arm.

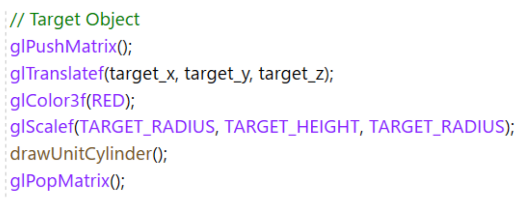


```
void drawUnitCylinder() {
    /* ... */

    glBegin(GL_QUAD_STRIP);

    GLfloat x;
    GLfloat y;
    GLfloat angle = 2 * M_PI;
    GLfloat angle_stepsize = angle / 64;

    while (angle > 0.0) {
        x = 1.0 * cos(angle);
        y = 1.0 * sin(angle);
        glNormal3f(cos(angle), 0, sin(angle));
        glVertex3f(x, 1.0, y);
        glVertex3f(x, 0.0, y);
        angle = angle - angle_stepsize;
    }
    glVertex3f(1.0, 1.0, 0.0);
    glVertex3f(1.0, 0.0, 0.0);
    glEnd();

    glBegin(GL_POLYGON);
    angle = 2 * M_PI;
    while (angle > 0.0) {
        x = 1.0 * cos(angle);
        y = 1.0 * sin(angle);
        glNormal3f(0, 1, 0);
        glVertex3f(x, 1.0, y);
        angle = angle - angle_stepsize;
    }
    glVertex3f(1.0, 1.0, 0.0);
    glEnd();


    glBegin(GL_POLYGON);
    angle = 2 * M_PI;
    while (angle > 0.0) {
        x = 1.0 * cos(angle);
        y = 1.0 * sin(angle);
        glNormal3f(0, -1, 0);
        glVertex3f(y, 0, x);
        angle = angle - angle_stepsize;
    }
    glVertex3f(1.0, 1.0, 0.0);
    glEnd();
```

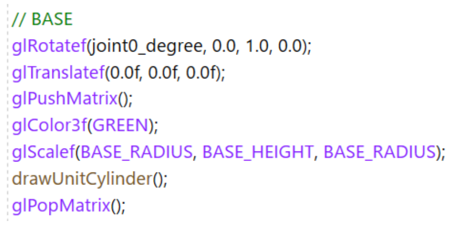
Next, I started rendering the target object and the whole robot arm in the order of the target object, the robot base, the first arm connecting with the base, the first joint, the second arm, the second joint, and the third arm, which is shown in the following figures. Besides, we will need the rotate function to rotate the joints into the right direction, since different from the arms, the joints are placed horizontal.

When rendering the robot arm, the glPushMatrix function and the glPullMatrix function played an important role. Since each of the part of the robot arm has its own model matrix, and we will not want each part of the robot interfere the transformation of the other parts, we need the glPushMatrix function and the glPullMatrix function here to keep the status of all the parts of the robot arm steady and right.

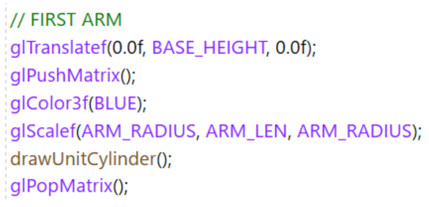
Furthermore, our robot arm should be able to be controlled by the user' s keyboard. Hence, to control the robot arm, I chose to rotate the base and the arm through the rotate function directly, which can be found in the following figures.



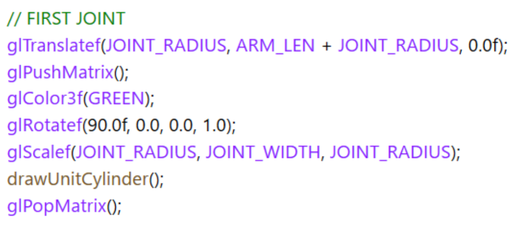
```
// Target Object
glPushMatrix();
glTranslatef(target_x, target_y, target_z);
glColor3f(RED);
glScalef(TARGET_RADIUS, TARGET_HEIGHT, TARGET_RADIUS);
drawUnitCylinder();
glPopMatrix();
```



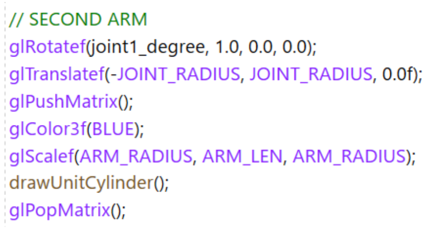
```
// BASE
glRotatef(joint0_degree, 0.0, 1.0, 0.0);
glTranslatef(0.0f, 0.0f, 0.0f);
glPushMatrix();
glColor3f(GREEN);
glScalef(BASE_RADIUS, BASE_HEIGHT, BASE_RADIUS);
drawUnitCylinder();
glPopMatrix();
```



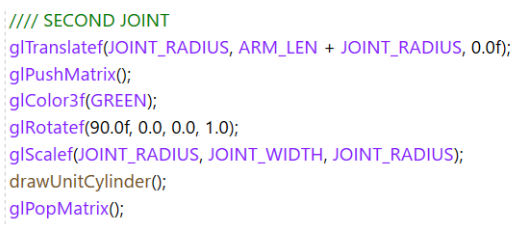
```
// FIRST ARM
glTranslatef(0.0f, BASE_HEIGHT, 0.0f);
glPushMatrix();
glColor3f(BLUE);
glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
drawUnitCylinder();
glPopMatrix();
```



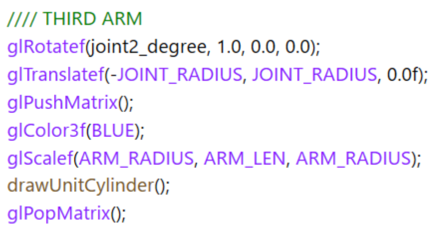
```
// FIRST JOINT
glTranslatef(JOINT_RADIUS, ARM_LEN + JOINT_RADIUS, 0.0f);
glPushMatrix();
glColor3f(GREEN);
glRotatef(90.0f, 0.0, 0.0, 1.0);
glScalef(JOINT_RADIUS, JOINT_WIDTH, JOINT_RADIUS);
drawUnitCylinder();
glPopMatrix();
```



```
// SECOND ARM
glRotatef(joint1_degree, 1.0, 0.0, 0.0);
glTranslatef(-JOINT_RADIUS, JOINT_RADIUS, 0.0f);
glPushMatrix();
glColor3f(BLUE);
glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
drawUnitCylinder();
glPopMatrix();
```

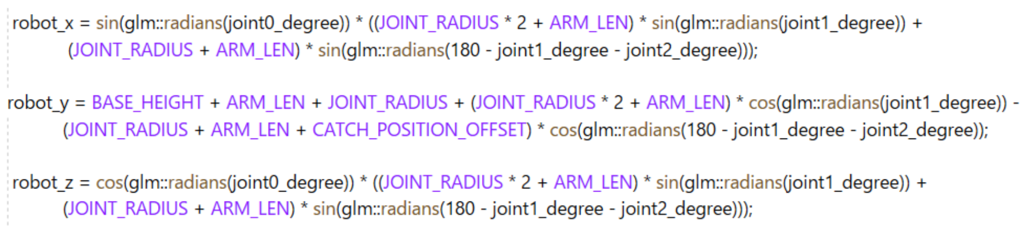


```
//// SECOND JOINT
glTranslatef(JOINT_RADIUS, ARM_LEN + JOINT_RADIUS, 0.0f);
glPushMatrix();
glColor3f(GREEN);
glRotatef(90.0f, 0.0, 0.0, 1.0);
glScalef(JOINT_RADIUS, JOINT_WIDTH, JOINT_RADIUS);
drawUnitCylinder();
glPopMatrix();
```



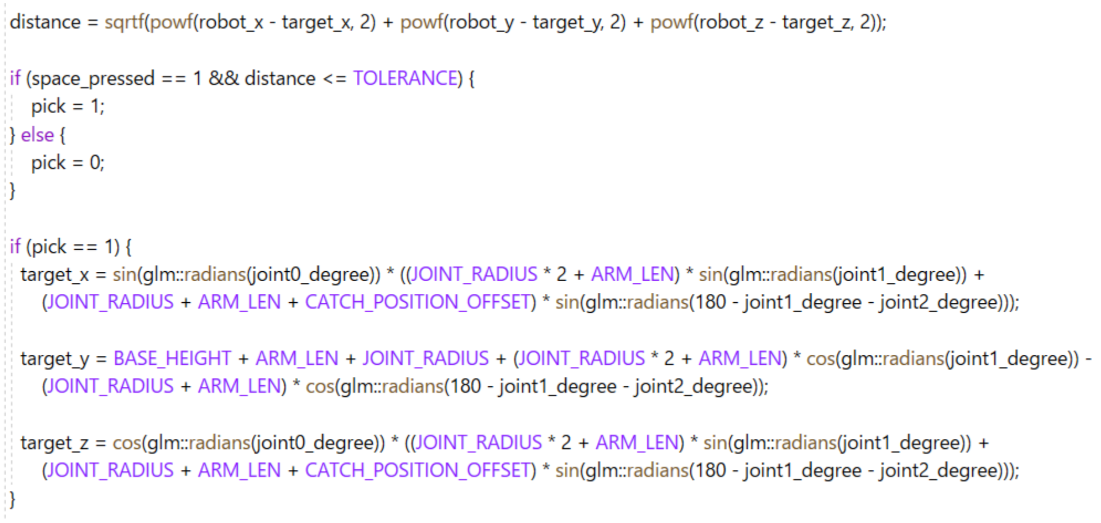
```
//// THIRD ARM
glRotatef(joint2_degree, 1.0, 0.0, 0.0);
glTranslatef(-JOINT_RADIUS, JOINT_RADIUS, 0.0f);
glPushMatrix();
glColor3f(BLUE);
glScalef(ARM_RADIUS, ARM_LEN, ARM_RADIUS);
drawUnitCylinder();
glPopMatrix();
```

Catching the target object is one of the main missions in this project. To know if the distance of the endpoint of the robot arm to the target object is close enough to catch, we will need to calculate the absolute position of the endpoint of the robot arm first. The following figure is how I obtain the absolute position of the endpoint of the robot arm.



```
robot_x = sin(glm::radians(joint0_degree)) * ((JOINT_RADIUS * 2 + ARM_LEN) * sin(glm::radians(joint1_degree)) +
        (JOINT_RADIUS + ARM_LEN) * sin(glm::radians(180 - joint1_degree - joint2_degree)));

robot_y = BASE_HEIGHT + ARM_LEN + JOINT_RADIUS + (JOINT_RADIUS * 2 + ARM_LEN) * cos(glm::radians(joint1_degree)) -
        (JOINT_RADIUS + ARM_LEN + CATCH_POSITION_OFFSET) * cos(glm::radians(180 - joint1_degree - joint2_degree));

robot_z = cos(glm::radians(joint0_degree)) * ((JOINT_RADIUS * 2 + ARM_LEN) * sin(glm::radians(joint1_degree)) +
        (JOINT_RADIUS + ARM_LEN) * sin(glm::radians(180 - joint1_degree - joint2_degree)));
```

I the distance of the endpoint of the robot arm to the target object is smaller than the tolerance distance, the robot arm will catch the target object. At the same time, I will start updating the position of the target object, since as the target object was caught by the robot arm, it will start to move with the endpoint of the robot arm until being released. The following is how I implemented it.



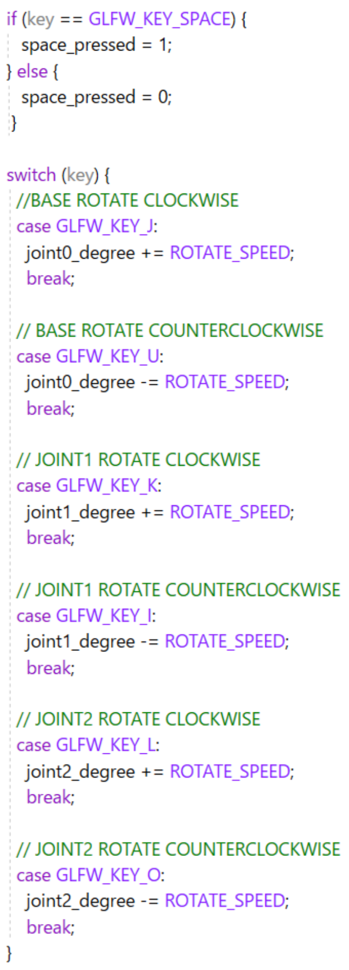
```
distance = sqrtf(powf(robot_x - target_x, 2) + powf(robot_y - target_y, 2) + powf(robot_z - target_z, 2));

if (space_pressed == 1 && distance <= TOLERANCE) {
    pick = 1;
} else {
    pick = 0;
}

if (pick == 1) {
  target_x = sin(glm::radians(joint0_degree)) * ((JOINT_RADIUS * 2 + ARM_LEN) * sin(glm::radians(joint1_degree)) +
     (JOINT_RADIUS + ARM_LEN + CATCH_POSITION_OFFSET) * sin(glm::radians(180 - joint1_degree - joint2_degree)));

  target_y = BASE_HEIGHT + ARM_LEN + JOINT_RADIUS + (JOINT_RADIUS * 2 + ARM_LEN) * cos(glm::radians(joint1_degree)) -
     (JOINT_RADIUS + ARM_LEN) * cos(glm::radians(180 - joint1_degree - joint2_degree));

  target_z = cos(glm::radians(joint0_degree)) * ((JOINT_RADIUS * 2 + ARM_LEN) * sin(glm::radians(joint1_degree)) +
     (JOINT_RADIUS + ARM_LEN + CATCH_POSITION_OFFSET) * sin(glm::radians(180 - joint1_degree - joint2_degree)));
}
```

For keyboard controlling, I applied the switch to find the key I want the detect, which are:

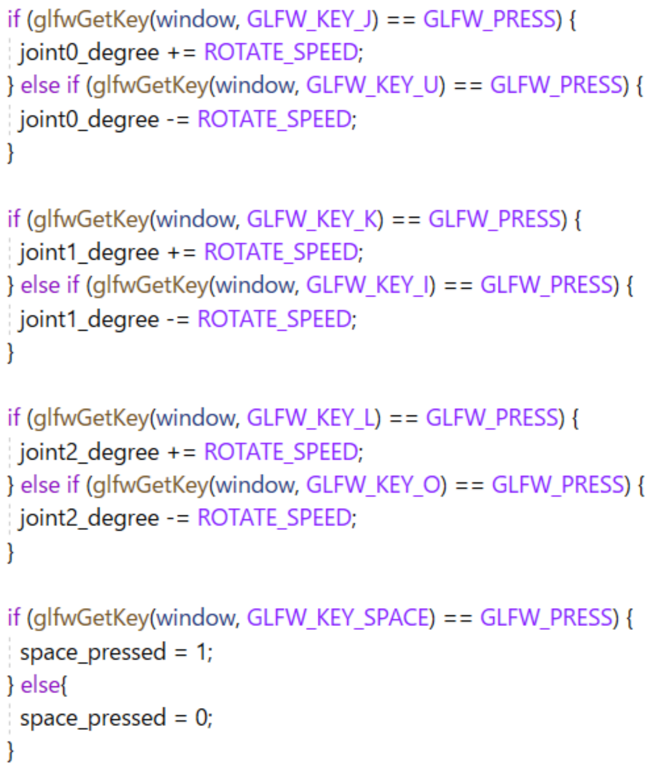
- J: Rotate the base clockwise
- U: Rotate the base counterclockwise
- K: Rotate the first joint clockwise
- I: Rotate the first joint counterclockwise

- L: Rotate the second joint clockwise
- O: Rotate the second joint counterclockwise
- Space bar: Catch object

Besides, I also applied the glfwGetKey function to make the keyboard detection continuous, so we can control the robot arm with more convenience. The implementation of the control of the robot is shown in the figures below.



```
if (key == GLFW_KEY_SPACE) {
  space_pressed = 1;
} else {
  space_pressed = 0;
}

switch (key) {
  //BASE ROTATE CLOCKWISE
  case GLFW_KEY_J:
    joint0_degree += ROTATE_SPEED;
    break;

  // BASE ROTATE COUNTERCLOCKWISE
  case GLFW_KEY_U:
    joint0_degree -= ROTATE_SPEED;
    break;

  // JOINT1 ROTATE CLOCKWISE
  case GLFW_KEY_K:
    joint1_degree += ROTATE_SPEED;
    break;

  // JOINT1 ROTATE COUNTERCLOCKWISE
  case GLFW_KEY_I:
    joint1_degree -= ROTATE_SPEED;
    break;

  // JOINT2 ROTATE CLOCKWISE
  case GLFW_KEY_L:
    joint2_degree += ROTATE_SPEED;
    break;

  // JOINT2 ROTATE COUNTERCLOCKWISE
  case GLFW_KEY_O:
    joint2_degree -= ROTATE_SPEED;
    break;
}
```



```
if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS) {
  joint0_degree += ROTATE_SPEED;
} else if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS) {
  joint0_degree -= ROTATE_SPEED;
}

if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS) {
  joint1_degree += ROTATE_SPEED;
} else if (glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS) {
  joint1_degree -= ROTATE_SPEED;
}

if (glfwGetKey(window, GLFW_KEY_L) == GLFW_PRESS) {
  joint2_degree += ROTATE_SPEED;
} else if (glfwGetKey(window, GLFW_KEY_O) == GLFW_PRESS) {
  joint2_degree -= ROTATE_SPEED;
}

if (glfwGetKey(window, GLFW_KEY_SPACE) == GLFW_PRESS) {
  space_pressed = 1;
} else{
  space_pressed = 0;
}
```

3. Result

In the figures below, figure 1 is how the whole robot arm looks like in the beginning, without being controlled by the user.

In figure 2, we can see no matter the view of the camera or the robot arm can be freely controlled.

In figure 3, we can see how the robot looks like when it caught the target object.

In figure 4, we can find that the target object will stop at the position where the robot arm release it.

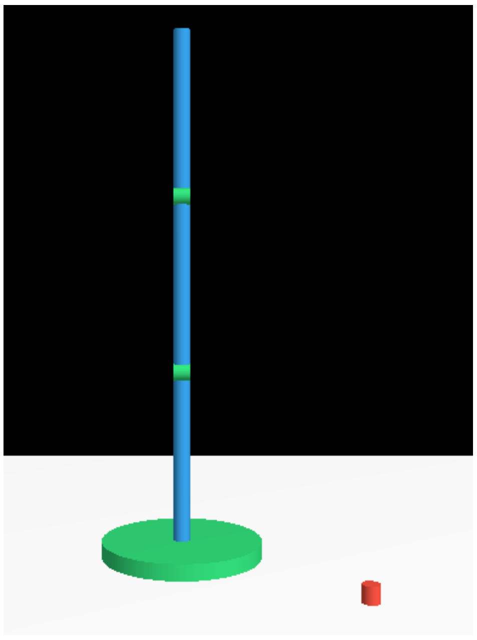


Figure 1

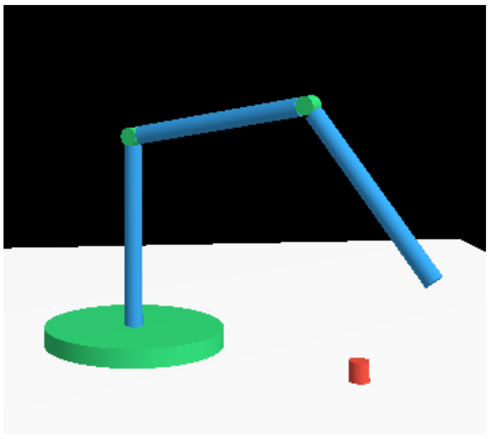


Figure 2

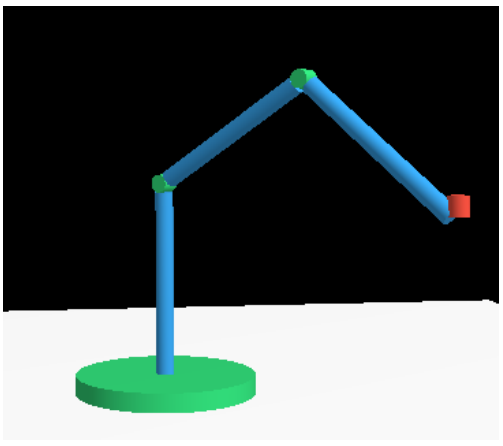


Figure 3

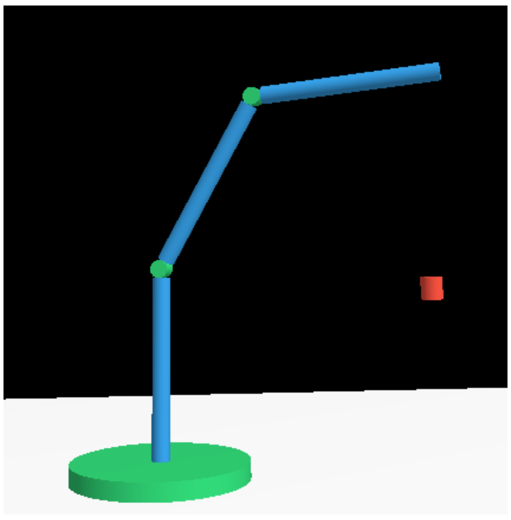


Figure 4

4. Problems Encountered

- **Render the first unit cylinder**

  Since this is my first–time using OpenGL, I was confused about how to render a cylinder' s surrounding at a time. Then through the TA' s suggestion, I realized that I can use segments to render a cylinder, which solved the problem.

- **Render the whole robot arm**

  At the beginning rendering the robot arm, I found my robot arm was kind of abnormal. Some of its parts overlapped with others. Then I found that I misused the glPushMatrix function, the glPullMatrix function, and the glTranslatef function. Hence, the initial base position of the parts of the robot is incorrect initially, which made them overlap.

- **Getting the absolute position of the endpoint of the robot arm**

  When my process came to catching the target object, I started to think about how can I get the absolute position of the endpoint of the robot arm properly. I tried several thoughts but all of them acted strangely. Then it came to me that I should make use of the property of the isotropic angle. Finally, I got the endpoint position of the robot arm nicely.

- **Update the target object position properly**

  After being able to catch the target object, I found my target object moved strangely in the air when it was moving along the endpoint of the robot arm. Then I found I had some misunderstood upon the coordinate system of the project. After correcting my cognition upon the system, I successfully fixed the problem.