Report of the Project

Zijun Huang, 1848233

1. Introduction

This project is a game playing Gomoku against computer. The difference between my game and plenty of Gomoku games online is that my game is a 3D game. The computer will use a robot arm to move the chess to the chessboard. It may give the user the feeling that he is playing against an agent rather than a computer program. My program uses Three.js for rendering and dat.GUI for some control.
Except the files in the folders “js” and “fonts” are downloaded from Internet. All the other files are written by myself.

2. Hierarchical Models

The most complex model in my program is the robot arm. This robot arm has three degree of freedom. There are mainly 4 parts in that model. The model is showed in figure 1.

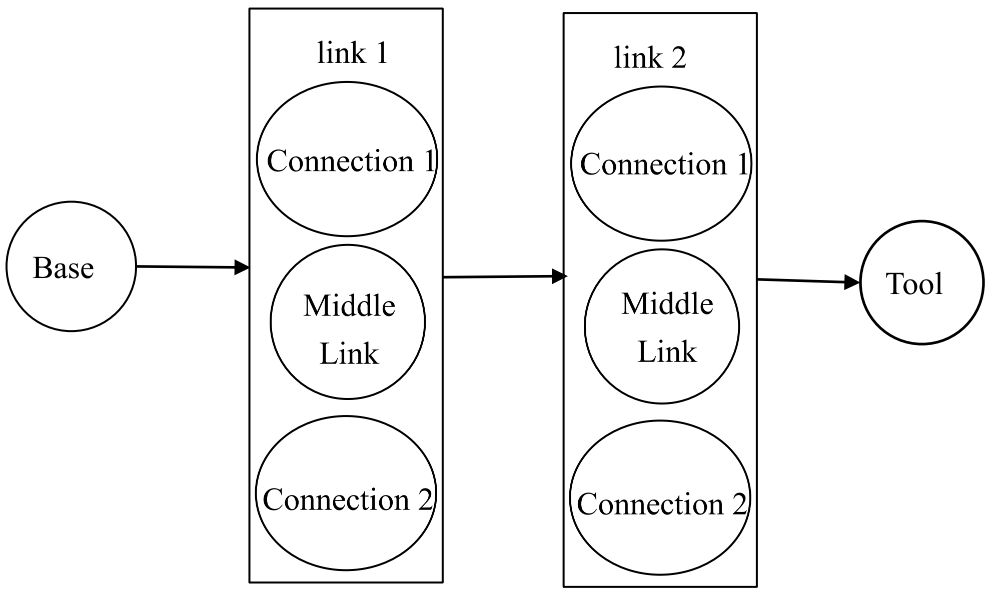


Figure 1. The Hierarchical Model of Robot Arm

All the geometries I used in robot arm are cylinders. The base and tool are only one cylinder. The link 1 and link 2 respectively include 3 cylinders, which is the connection to the link before, the connection to the link after and a cylinder between the two connections. These three cylinders will not change their position and orientation relative to each other. So, I use the class “Group” in Three.js to group these 3 cylinders together and treat them as one object.

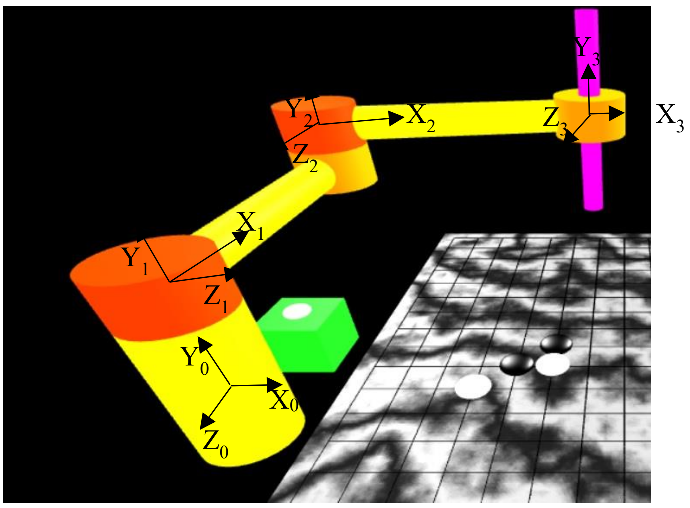


Figure 2. The Coordinate Systems of the Robot Arm

With using the class “Group”, it can be easier to set the coordinate system as you want. The coordinate system of the robot arm is showed in figure 2. For the link 1 and link 2, the origins of the coordinates are not in the middle of the object. Instead, they are in the points that the objects rotate relative to their parent objects, which helps to easily set the rotation of the objects.

The chessboard is another hierarchical model. The base of the chessboard is a box geometry. The texture of the chessboard is a procedural texture similar to texture of the marble. I use the algorithm from (Vandevenne 2004). The original algorithm in the website is written in C++ and I rewrote it in JavaScript. For the material of the chessboard is a “MeshPhongMaterial” and there are two point-lights and one ambient light in the environment, the texture is both for the map and specular map in the program. There are also some grids above the chessboard. At first, I consider using texture to create those grids. Then I wrote a procedural texture about some vertical and horizontal lines and combined it with the marble texture. But I found that the lines have so many sawtooth. Even I set the Three.js to use anti-aliasing cannot solve the problem. Some of the lines will disappear when the camera changes its pose. I gave up using this method and set all the lines to line geometry which have the same parent the box geometry. Also, when the robot arm or user put a chess on the chessboard, the chess will become a child of the box geometry.

The chess box is a box that continually generates chesses. The new generated chess will be a child of the box. Then when the robot arm takes the chess, it will be the child of the tool in the robot arm. At last it will be the child of the chessboard.

3. Trajectory Planning for Robot

For the animation, we need to give the keyframe of each joint angle of the robot arm and interpolate the angle between them. But here, each time the desired position of the tip of the tool of robot arm may be different. I use a different way to get the joint angles. Every time the robot arm moves the chess from one position to another position, it will plan a trajectory of the tip of the tool of robot arm in Cartesian space. For the x and z direction, it will use a quintic polynomial which will have the zero velocity and acceleration in both the starting point and ending point. The function is showed in (1).

$$p(\tau) = p_0 + (p_1 - p_0)(6\tau^5 - 15\tau^4 + 10\tau^3), \tau = t/T \tag{1}$$

p_0 and p_1 are the starting and ending position. τ is a normalized variable and T is the time to finish the whole trajectory.

To avoid that the tool of the robot arm hitting other chesses already in the chessboard, I insert one point between the starting and ending position. Because of that I use two 4 order polynomials in such a way the starting and end position have zero velocity and acceleration and the velocity and acceleration of the middle point is continuous.

$$p_A(\tau) = \begin{bmatrix} \tau^4 & \tau^3 & \tau^2 & \tau & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 & -1 \\ -3 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix}, \tau = t/T \tag{2}$$

$$p_B(\tau) = \begin{bmatrix} \tau^4 & \tau^3 & \tau^2 & \tau & 1 \end{bmatrix} \begin{bmatrix} -1 & -1 & 2 \\ 3 & 2 & -5 \\ -3 & 0 & 3 \\ 1 & -2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \end{bmatrix}, \tau = t/T \tag{3}$$

The formula (2) is the first 4 order polynomial and the formula (3) is the second 4 order polynomial. p_0, p_1 and p_2 are the starting, middle and ending points respectively.

After getting the trajectory of the tip of the tool, we should map it to the angle value of each joint which will determine how to render the robot arm. We assume that three joints parameters are q_1, q_2 and q_3. The forward mapping from the joints' parameters to the position (x, y, z) of the tip of the tool is formula (4).

$$\begin{cases} x = -320 + 300\cos q_1 + 350\cos(q_1 + q_2) \\ y = 100 + q_3 \\ z = 100 - 300\sin q_1 - 350\sin(q_1 + q_2) \end{cases} \tag{4}$$

Given one group of joints' parameters (q_1, q_2, q_3), we will have one related position (x, y, z). But if we do the inverse mapping, the result is not unique. This will give some difficulty when generating the joints' parameters. To solve this problem, I use the mapping in velocity space. By differentiate formula (4), we can get formula (5).

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} -300\sin q_1 - 350\sin(q_1 + q_2) & -350\sin(q_1 + q_2) & 0 \\ 0 & 0 & 1 \\ -300\cos q_1 - 350\cos(q_1 + q_2) & -350\cos(q_1 + q_2) & 0 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = J \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{5}$$

After planning the trajectory with (1) to (3), we can differentiate them and the get trajectories of the velocity $(\dot{x}, \dot{y}, \dot{z})$. By formula (6), we can get the velocity of each joint.

$$\begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = J^{-1} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{6}$$

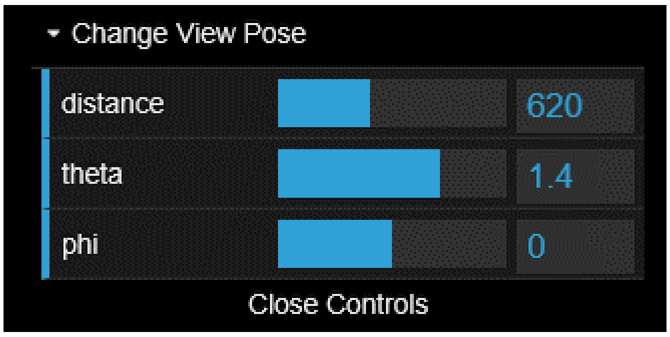
And each time it just needs to update the joints' parameters to get the new pose of the robot arm by formula (7).

$$q_i = q_i + \dot{q}_i * dT \tag{7}$$

I have also carefully designed the length of each link and the relative position among the robot arm, the chessboard and the chess box. Just to ensure that the Jacobian matrix J in formula (5) always has full rank.

4. Interactions

There are two scenes in the program. The first scene is asking the user to choose if they want to move the stone first or if they want the agent to play first. After clicking the choice words, the program will render the second scene which includes the robot arm, the chessboard and the chess box. If they want to move the stone first, the robot arm will wait until the user clicks the desired position otherwise the robot arm will start to move immediately. After the finish of the game, it will return to the first scene and wait for the new game to start.


Change View Pose

distance 620

theta 1.4

phi 0

Figure 3. The control panel.

In the second scene, the user can use the control panel in figure 3 in the up-right corner to change the camera position in such a way the user can watch the scene in different direction. Also, the program has to detect which position the user want to put its chess in. Firstly, we know the global position ${}^{G}X_i$ of each point that the user can put the chess in. Then the position of the point in the scene will be computed by formula (8).

$$ {}^{S}x_i = PM\,{}^{G}X_i \tag{8} $$

P is the projection matrix. M is the world view matrix. The program stores all the scene positions of those points. When the user clicks one point, the program will compare that point with all the stored points. If the smallest distance is under some threshold, it will think that the user wants to put the chess in that point. If it is in the user turn, there will be a chess appear in that point. When the user changes the position of the camera, those points will be recomputed and stored.

Another important part of interaction of this program is how the agent plays with the user. The idea of the strategy is from (Feng 2019). It is a greedy algorithm. For every position that can be put the chess in, there are two value associated, the value for opponent and the value for itself. The agent will choose the position will the biggest value. For the design of the algorithm is a little conservative, if there are two position with the same biggest value, one of the biggest values is for the opponent one and the other is for itself, the agent will choose the position with the biggest opponent value.

To compute the value which is initialized with 0, the agent will iterate the four directions, the up-left, the up, the up-right and the right. In each direction the agent will detect that if it can put 5 chesses in it. If it can, it will count how many its chesses are already in this direction which we call it N. Then $v = v + 10^N$. Also, it will count how many its chesses are around that position. The number will be added in v.
After some games against that agent, I found that it prefers to defend rather than attack. The games will last a little long and when you are a little tired, the agent will win. It is not smart enough and may be the better way to do is to use Monte Carlo Tree Search combined with a trained value network like the agent playing Go (Silver 2017). I tried to do it a little bit but I found that it was very slow to generate training data with one computer and I had to give up.

5. Conclusion

In this project, I develop a program that uses robot arm to play Gomoku with user. It includes the hierarchical model, animation and interaction inside. The main goal is that we want the user to feel more like playing with the real robot. But it may need the better design of the appearance of the robot.

References

Feng, XiaoXiao. *Greedy Strategy of Gomoku AI.* 2 16, 2019. https://blog.csdn.net/itnerd/article/details/87472224 (accessed 7 3, 2019).

Silver, David. *et al*. "Mastering the Game of Go without Human Knowledge." *Nature*, 2017: 354-359.

Vandevenne, Lode. *Lode's Computer Graphics Tutorial. Texture Generation using Random Noise.* 2004. https://lodev.org/cgtutor/randomnoise.html (accessed 2019).