CAV: KINEMATICS

The implementation is done so the results can be seen directly in the edit mode of Unity. A custom editor was created; it can be accessed from the menu bar: CAV -> Kinematics. This editor contains the option to perform forward and inverse kinematics.

Preliminaries

Quaternions

Quaternions are a mathematical concept. In computer graphics and animation they are to perform rotations among other things. The main advantage of quaternions is that they avoid the gimbal lock problem of Euler angles. Also, it is not necessary to indicate the order of rotation around the canonical axes (x, y, and z), something necessary with Euler angles.

Degree of freedom (DOF)

Degree of freedom is the number of dimensions in which a parameter can vary. For instance, in the 3D space, an object can have 1, 2, or 3 DOF for translation. This means, that the object can move in 1, 2 or 3 directions simultaneously.

Forward kinematics

The forward kinematics method is meant to move a kinematic chain by directly assigning values to the joint angles. We go from angles to a final position of the end effector. In this case, the final configuration of the chain is unique. In animation, this method can be useful to perform coordinated motion among objects in a scene, where the final position of the end effector is not restricted.

Kinematic chain

The kinematic chain corresponds to the left arm of the robot with three components: wrist, forearm, and upper arm. The corresponding joint angles are set to update the entire chain.

Implementation

The implementation is straightforward; the text boxes in the “Options FK” in the kinematics window shown in Figure 1 correspond to the values of the corresponding joints. To move the chain, simply type values in every text box and click on Rotate. The left arm of the robot will be updated accordingly.

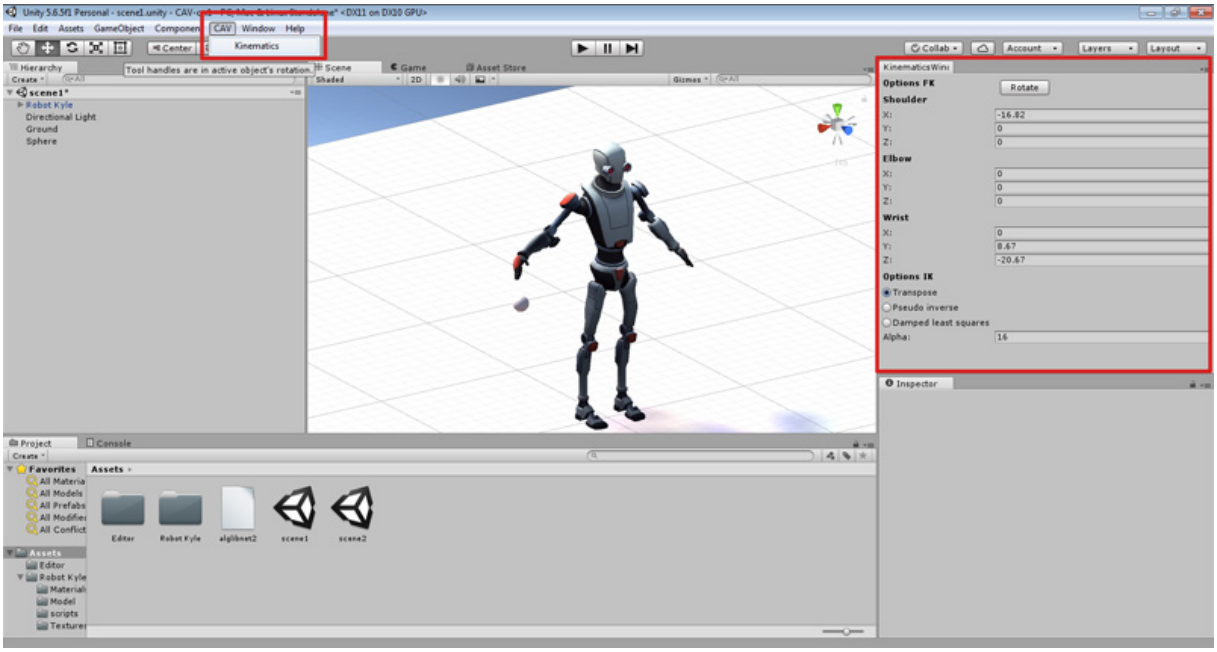

Figure 1: Custom Editor in Unity to perform kinematics

In the code, the rotation of every link in the chain is set by rotations around the x, y, and z canonical axes. This procedure is performed using three quaternions (one per axis) where the axis of rotation corresponds to one of the canonical axes, and the angle of rotation is the corresponding value typed in the given text box. The final rotation is the result of the multiplication of the three previous quaternions.

Inverse Kinematics

Inverse kinematics is meant to configure to joint angles of a kinematic chain so that the end effector reaches the desired position. In this case, the is no a single solution and there are several methods to calculate the joint angles. In animation, this method can be used to make a character move the legs or arms so that they point a to given position.

Kinematic chain

The kinematic chain corresponds to the right arm of the robot with three components: wrist, forearm, and upper arm. The position of the last segment of the middle finger in the hand is set as the final point of the end effector; this is the point that we want to reach the target position.

Target

The target is the white sphere object in the scene as seen in Figure 2. The position of this element is the point that has to be reached by the extreme of the end effector. This point has to be moved with the translation gizmos.

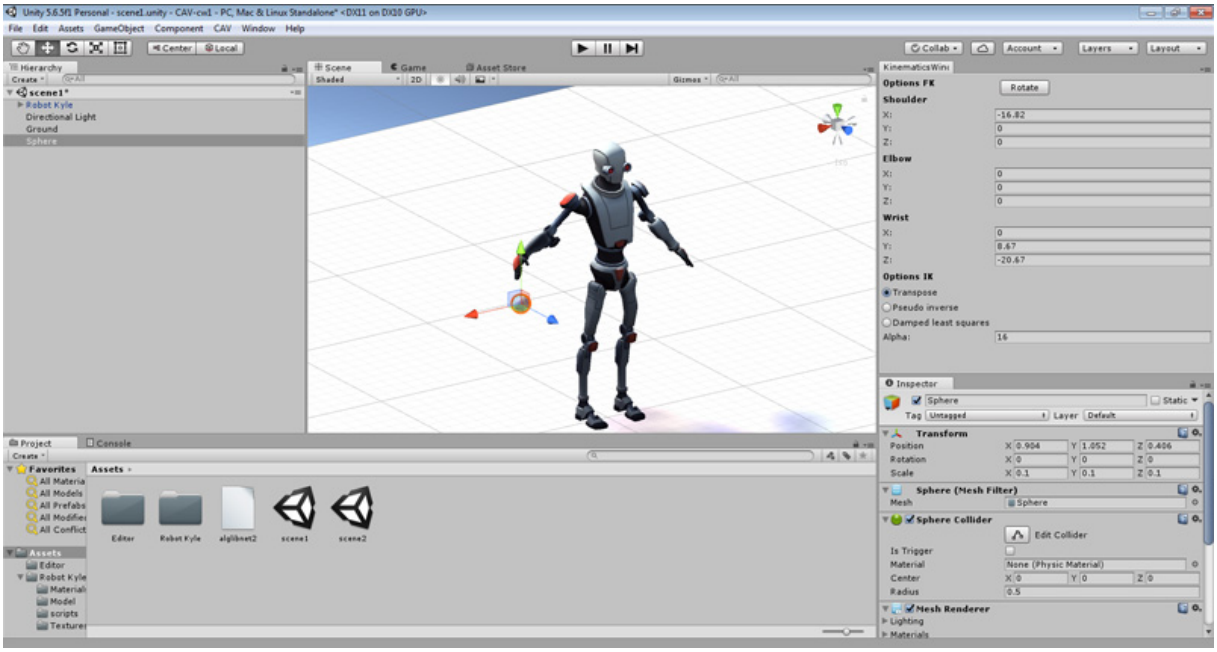

Figure 2: Target to be reached by the end effector

The Jacobian Matrix

The Jacobian matrix `J` corresponds to the relationship between the position of the target and variation of the joint angles of every link in the chain to reach that point. For every link in the chain, the calculation of the corresponding Jacobian entry has to follow these steps:
\begin{enumerate}

- Obtain the positions of the link `p`, the target `t`, and the extreme of the end effector `s`.
- Transform the previous position to the local coordinates of the link.
- Obtain the vectors from `t` to `s`, and from `p` to `s`.
- Calculate the axis of rotation `v`. This vector is perpendicular to the previous vectors.
- Calculate the entry by performing the following cross product: `v` x [`s` - `p`].

In order to find the variations of the joint angles, we need to calculate the inverse of the Jacobian matrix. There are two problems that do not allow us to do such calculation:

- The Jacobian matrix is not always square, therefore is not invertible.
- If the Jacobian matrix has singular values, the solutions can be unstable.

For those reasons, we need alternative methods.

The Jacobian transpose method

In this case, the variation of the joint angles is calculated by using the transpose of the Jacobian matrix. This matrix is multiplied by a scalar factor. The value of this factor determines the size of the angle variations for the joints. Finally, the matrix is multiplied by the difference of the target `t` and the end effector extreme `s`: `e` = `t` - `s`. `J` and `e` are matrices, so we can multiply them.

The key component of this method is the scalar factor. A proper value can lead to good results, but there is a deterministic way to set such value; it requires a level of experience. Nonetheless, when a good scalar value is set, the motion of the chain is smooth.

The pseudo-inverse method

The pseudo-inverse matrix of the Jacobian is calculated using the Singular Value Decomposition method (SVD). The functions to perform SVD are obtained from the third party library ALGLIB.

In this case, the motion of the chain is smoother compared to the previous method. However, at certain positions, the system becomes unstable. Moreover, the motion can be too slow.

The damped least squares method

This method is similar to the previous but we add a damping parameter. This parameter is meant to avoid singularities; in the code, this is means to avoid division by zero or a number very close to zero. This is a problem when the elements of the diagonal matrix of the SVD approach zero and the system becomes unpredictable.

This method is better than the previous one, the stability of the system increases and the motion is even smoother. However, since it depends on an additional parameter, the motion can be unexpected is the value of that parameter is not carefully chosen.

To choose one of the described methods, select the corresponding option in the “Options IK” section in the kinematics window shown in Figure 1. The transpose and damped least square methods require to input a value for the corresponding parameters; 16 and 0.1 are the recommended values respectively.

Updating the joints

Once the angle variations have been solved with one of the methods described above, they are used to updated the joints. In this case, we use the axis of rotation `v` calculated when creating the Jacobian matrix. Basically, we need to perform a rotation corresponding the angle variation calculated previously; we create a quaternion `q` using the axis `v` and the angle variation corresponding to the given joint. We also need another quaternion `r` which corresponds to the current rotation of the joint. The final rotation to update the joint is the multiplication of `q` and `r`.

Additional notes

- Scene1 is the displayed one in Figures 1 and 2. Please use this scene to test the implementation.
- The alglibnet2.dll file contains the functionality from ALGLIB.
- The code to create the custom window is in the file CAV-cw1/Assets/Editor/KinematicsWindow.cs It has the code to perform forward kinematics.
- The scripts with the code to perform inverse kinematics are in the folder CAV-

cw1/Assets/Editor/Robot Kyle/scripts

– Matrix.cs is provided

– Serializer.cs contains all the code to perform the operations related to the Jacobian matrix.

– Sphere is the script attached to the target ball in the scene. It has calls to the functions in Serializer.cs