|  |  |  |
| --- | --- | --- |
| |  | | --- | | Wrist\_y\_Left, α10  Hand\_x\_Left, α11  Shoulder\_z\_Left, α8  Shoulder\_y\_Left, α7  Elbow\_x\_Left, α9  **Figure 2**. Description of Left Hand Joints | |  |     **Figure 1**. Description of Left Leg Joints  Foot\_x\_Left, α6  Foot\_y\_Left, α5  Knee\_y\_Left, α4  Hip\_y\_Left, α3  Hip\_z\_Left, α2  Hip\_x\_Left, α1 |

1. Measurements and setting up of coordinate frames

The first task was to assign each joint with a reference coordinate frame and an angle of rotation. Figure 1 and Figure 2 shows the assignments of coordinate frames for joints on the left leg and left arm of the humanoid respectively. There are 22 joints in total for the humanoid, with each leg having 6 joints and each arm having 5 joints. In addition to the coordinate reference frames provided in the Solid Works model of the humanoid, few additional coordinate frames were added. These include the Torso reference frame, a reference frame at the center of the base of each foot, and a wrist reference frame for each hand (shown in Figure 3).

The SE3 class was used to form the transformation matrices for each reference frame. The translation measurements from one joint to next were made by using the measurement tool in Solid Works. The measurement is made from the origin of one coordinate frame to that of the next coordinate frame. The rotation matrices are defined according to the axis about which the joints rotate.

1. Center of Mass for the Humanoid

Center of mass for the humanoid is calculated by using the traditional method shown below:

Defining a link as the rigid body between two joints, the mass and center of mass of each link was measured using the Mass Properties tools in Solid Works. Then by applying the above mentioned formula, we were able to calculate the center of mass of the humanoid with respect to three different frames of reference. Therefore, the function to calculate the center of mass allows the user to choose the required frame of reference. The variable called frameofReference is used for this purpose, where a value of 0 returns the center of mass with respect to the Torso reference frame, a value of 1 returns the center of mass with respect to right leg’s base, and a value of 2 returns the center of mass with respect to left leg’s base.

|  |
| --- |
| **Figure 3**. Humanoid with added frame of references |

1. Visualization

|  |
| --- |
| **Figure 4**. Visulaization results |

After the definition of the Matlab model of the humanoid, the visualization of the model was accomplished as well. The visualization uses the shBlock function written by Dr.Vela to define each motor as a block. First of all, the transformation from Torso to the center of each motor is defined using the SE3 class. Then the shBlock function is called in order to plot a block such that it is centered on the point defined by the transformation matrix. Colors are used to differentiate between different motors. Also on the same figure, the support polygon and the center of mass of humanoid is plotted. An example of the end result of the visualization function is shown in Figure 4.

1. Support Polygon

Static walking assumes that a robot is statically stable at times. This means, if the robot were to stop at any point in it’s motion, it will remain upright. This static stability is attained by having the robot’s center of mass projected on to it’s support polygon. For a robot walking on a flat surface, the support polygon is the convex hull of it’s foot or feet (depending on how many are on the ground). To calculate and plot the support polygon we created the following Matlab function:

function supportPolygon(object,howManyFeetOnGround,whichFootOnGround)

Parameter List:

1.*howManyFeetOnGround* : Value to indicate how many of the robot’s feet are currently making contact with the ground. Possible Values include 1 or 2.

2*.whichFootOnGround* : Value to indicate which foot is on the ground. Set to 1 if only right foot is making contact with the ground. Set to 2 if only left is making contact with the ground. If both feet are on the ground, value does not matter.

There are three cases to consider when calculating the support polygon for our

First Case: Right Foot On Ground

The first step in calculating the support polygon is finding the position of a point on the foot respective to the torso frame. Since the g\_foot\_xRight represents the location of the joint right above the foot and not a location on the foot itself, another transformation matrix is created to be able to reach the center of the robot’s foot called g\_foot\_base.Using the SE3 class, g\_foot\_base is created by the following statement:

g\_foot\_base = SE3(d\_foot\_base,eye(3));

d\_foot\_base is the distance from the foot\_x joint to the center of the foot and was found using Solidworks. eye(3) is the identity and the R for this matrix. The coordinate frame is not rotated relative to the foot\_x frame hence the rotation matrix is the identity matrix.

To calculate the position of the center of the right foot with respect to torso, all 8 g matrices were multiplied in order. Since this product matrix is also part of the SE3 class, the position can be extracted using the getTranslation() function.

In order to create the convex hull a set of points are needed. To attain these set of points we used solid works to calculate distance between the center of the foot and various points on the perimeter of the foot. Using these distances we created 12 points that represented points on the perimeter of the right foot with respect to the torso frame. Using Matlab’s convhull() with the 12 points as the input, we plotted the convex hull of the right foot.

Second Case: Left Foot On Ground

The second case is almost exactly similar to the first case however, when multiplying the transformation matrices, the ones that represent the left side are used instead of the right side. Every step after that is exactly the same.

Third Case: Both Feet On Ground

This case is somewhat of a combination of the First Case and Second Case. In Case One, 12 points on the perimeter of the right foot with respect to the torso are found. In Case Two, 12 points on the perimeter of the left foot with respect to the torso are found. These 12 points were then inputs of the convhull() function. In this case, we will use 12 points from both Case One and Case Two and give the convhull() function 24 points.

1. Inverse Kinematics

Manny’s foot can be called as an end-effector as it is the last link on it’s leg. The position of Manny’s foot in respect to the torso is a function of 6 g- matrices multiplied together. Since these g- matrices contain a Rotation matrix which is dependent on an alpha(joint parameter), it is safe to say that joint parameters determine the position of an end-effector, and in our specific case, alpha angles 1-6 determine the position of the robot’s left leg. Using joint parameters to determine end effector position is known as Forward Kinematics and is as simple as multiplying the 6 g-transform matrices and extracting the translation component of the resulting g matrix. The opposite, using the end-effector position to determine the joint parameters is known as Inverse Kinematics.

Since Manny’s end effector position is determined by 6 joint parameters, we used the Jacobian method. The Jacobian maps the joint parameter velocities to the end-effector velocities. Thus, if we are aware of our end-effector velocities, we can calculate our alpha velocities(or alpha dots) and them to the current alpha values.

The first step was to compute Jacobian to map the velocities of alpha values 1-6 to Manny’s left foot’s velocity. This was done by finding Manny’s left’s foot’s position as a function of alpha 1, alpha 2, alpha 3, alpha 4, alpha 5, alpha 6. Then, a partial derivative was taken for the function with respect to each alpha. Each partial derivative produced a 3x1 matrix. Concatenating 6 3x1 matrices together created a 3x6 matrix, which was our Jacobian.

Since, End-Effector velocity = Jacobian \* AlphaVelocity, we must somehow use the Jacobian and the End-Effector Velocity to calculate the Alpha-Velocity. Since the Jacobian is

a 3x6 matrix, a pseudo-inverse needs to be taken. The pseudo-inverse provides the least squares solution. However, some of the joint angles have joint limits, so we used a weighted pseudo-inverse of the for :(JT\*J – W)-1 \* JT where J is the Jacobian and W is a 6x6 diagonal matrix. Multiplying the resulting matrix by the End-Effector Velocity calculates the alpha dot.

1. Lift Left Leg

For Manny to start walking, he must do so by lifting his knee up, which in turn lifts his foot off the ground and then bend his knee so that his foot is off the ground. Using, this information we set Manny’s foot’s final position. We then calculated Manny’s current position and initialized a time variable (how long should it take him to kick). Calculating the difference between the final and current gives us the distance. We then created a while loop which goes on until distance between the current position and final position is minimal enough.

Inside the loop, we first call the visualization function. Then we set the end-effector velocity as distance/time. Next, we call the inverse Kinematic Function to attain the alpha velocity/alpha dot. This resulting alpha dot values are added to the current alpha values to create the new alpha values which are then sent to the corresponding Dynamixel motors (only alpha values 1-6 are affected by this). Then, setGTransforms called to use the updated alpha values to update the g matrices. Finally, the new foot position is calculated which leads to calculating a new distance between the final position and current position, and time is also decremented.