Modern Robots: Evolutionary Robotics

Programming Assignment 6 of 10*

Description

In this week’s assignment you will be adding eight joints to connect together the nine body parts comprising the robot (Fig. 1). You will accomplish this in the same way as for adding bodies to Bullet: add one joint, compile and run the application so that you can see the effect of the addition on the simulation, only then add a second joint, and so on.

[Note: every time you get something that looks like one of the panels in Fig 1, take a screen shot and include it in what you hand in. Your hand-in should have all five panels from Fig. 1]

1. Back up Assignment 5 on a flash drive or another computer so that you can always return to your completed fifth assignment.


3. In the top of RagdollDemo.h, find the variables we added last assignment

   btRigidBody* body[9]; // one main body, 4x2 leg segments
   btCollisionShape* geom[9];
   bool pause;

   and add the following variables

   btHingeConstraint* joints[8];
   bool oneStep;

4. Before creating joints, we need to grapple with two different coordinate systems. Up till now the only coordinate systems discussed have been the world coordinate system. However, each body has its own coordinate system where the center of mass is its origin (0,0,0) and may rotate. Run the simulation, hit 'w' to turn on the wireframe mode. You will see red, green, and blue axes markers that shows a body’s local coordinate system.

5. Write a function in RagdollDemo.h with this signature:

   btVector3 PointWorldToLocal(int bodyIndex, btVector3 point)

---

*Original material was graciously provided by Josh Bongard. Jeff Clune slightly modified it.
Look up documentation on the method getCenterOfMassTransform and the class btTransform. This function should take the index of one of your objects and a world coordinate, and it should then return the coordinate as a local coordinate of the object at index. Test that it works by printing out the local coordinate of the box’s center of mass: PointWorldToLocal(0, btVector3(0.,1.,0.)). The result should be (0,0,0).

6. Write a variant of the preceding function that converts an axis vector, or direction vector, from the world coordinate system to a body’s local coordinate system (see Figure 2). Do this in RagdollDemo.h with this signature: btVector3 AxisWorldToLocal(int bodyIndex, btVector3 point). If your main body has not been rotated, you can test that it works by printing out the result of AxisWorldToLocal(0, btVector3(0.,1.,0.)). The result should be (0,1,0).

7. Now create a function CreateHinge(int index,...) in RagdollDemo.h that will create a joint. You will need to pass this function a number of parameters (see point 8 below): you need to tell it which two bodies to attach together, where the joint’s fulcrum should be, and the axis of the joint.

8. Inside the CreateHinge function you will use these parameters to create a hinge joint (new btHingeConstraint()).

btHingeConstraint(btRigidBody& rbA, btRigidBody& rbB, 
                 btVector3& pivotInA, btVector3& pivotInB, 
                 btVector3& axisInA, btVector3& axisInB)

Note that the vector arguments to this constructor are given in the local coordinate systems of bodies A and B. At the end of the function, you’ll want to add the joint to the simulation using the following line:

m_dynamicsWorld->addConstraint(joints[index], true);

9. Create another function DestroyHinge(int index) that removes the joint from the simulation.

10. Add a line CreateHinge(0,...); that creates the first joint, which attaches the right lower leg to the right upper leg (Fig. 1a). This line should be placed just after all of the bodies have been created.

11. Make sure you destroy every hinge you create in the exitPhysics method by using your DestroyHinge function in the exitPhysics method. Recompile and rerun until there are no errors and you can restart the simulation by hitting the space key with no errors.

12. Compile and run the simulation in paused mode. You should see that, as the right upper leg falls, it is stays connected to the right lower leg at the knee’. You should see something like that of Fig. 1a. However, this may happen too fast to capture the screen, so we’ll need to add an ability do one step at a time.

13. We want to add the ability to pause the simulation and with one key press advance the simulation only one time step. In the method clientMoveAndDisplay, edit the call to stepSimulation so that it will run one step if it is paused and the variable oneStep is true. oneStep should then be set to false. Recompile and rerun until there are no errors.

14. Edit the keyboard Callback method so that when the key ‘o’ is pressed oneStep is set to true. Recompile and rerun until you can pause the simulation and hitting ‘o’ advances the simulation one time step.

15. Compile and run the simulation in paused mode. Press ‘o’ to repeatedly step through the simulation. You should see that, as the right upper leg falls, it stays connected to the right lower leg at the ‘knee’. You should be able to capture the screen with something like Fig. 1a. Store this image in your document.
16. Now add the second joint that connects the left upper leg to the left lower leg, recompile and rerun.

17. Add the third joint that connects the front (into the screen) upper leg to the front lower leg, recompile and rerun.

18. Add the fourth joint that connects the back (toward the viewer) upper leg to the back lower leg, recompile and rerun. This should now produce an image like that in Fig. 1b. Screen capture and store in your document.

19. Using the same procedure, add the fifth joint that connects the right upper leg to the main body. Recompile, rerun and step through the simulation to get an image like that in Fig. 1c. Screen capture and copy into your document.

20. Now iteratively add the sixth, seventh and eighth joint connect the remaining three legs to the main body. The robot should now ‘sit down’, and its legs should flatten out, producing an image as in Fig. 1d.

21. We now need to constrain the range of motion of each joint. For simplicity, we will set each joint to only rotate through $[-45^\circ, 45^\circ]$. To do this, you will need to add this line to `CreateHinge(...):

   ```cpp
   joints[index]->setLimit(-45.*3.14159/180., 45.*3.14159/180.)
   ```

   What is the x*(3.14159./180.) for?

22. Now when you recompile and rerun the simulation you should obtain an image as in Fig. 1e. Screen capture it and copy into your document, and submit your document.

![Figure 1: Incremental addition of joints to the quadrupedal robot.](image)
Figure 2: A 2D example of the local coordinate system. Point A has the coordinate (4,4) in the global coordinate system, but in the local coordinate system of the rotated object it has the coordinate (1.41, 1.41). Similarly, vector V is a direction vector (http://en.wikipedia.org/wiki/Direction_vector) that points in the direction of (2,2) in the original coordinate system, but to represent that same axis of rotation, we need to describe a line passing through the origin of the local coordinate system that is parallel to V, such as (in local coordinates) (1,0), (2,0), etc.