Description

In assignment 7 you added eight motors to your robot. In assignment 8, you added four sensors. In this assignment, you will add a randomly-weighted synapse connecting each sensor to each motor. This will cause the robot to exhibit non-random behavior, even though it has a random neural network: each time a foot comes in contact with the ground, the motors will respond in the same way. When you capture five images of the robot using a neural network, your images should show that the robot is no longer moving randomly: as can be seen in Fig. 1, the robot maintains more or less a similar pose as it moves about.

1. Back up Assignment 8 on a flash drive or another computer so that you can always return to your completed eighth assignment.

2. Copy directory Assignment 8, which contains your submitted document and the entire Bullet folder. Rename the new directory Assignment 9.

3. Create a member variable in RagdollDemo.h that is a two dimensional array after the touches member:

   ```
   double weights[4][8];
   ```

4. At the beginning of the initPhysics method, set each value in the array to a random value in the range [-1, 1]. You will need to use rand() as you did previously. Now, element weights[i][j] encodes the weight of the synapse connecting touch sensor i to motor j.

5. In clientMoveAndDisplay, just before you call ActuateJoint(i,motorCommand, ...) (which sends the desired angle motorCommand to motor i), you need to set motorCommand based on the values of the sensors, instead of setting it randomly. This is done by creating a for loop that iterates over each of the four sensors before calling ActuateJoint().

6. Initialize motorCommand to 0 and then, during each pass through the loop, multiply the value of that touch sensor by the weight of the corresponding synapse, and add it to motorCommand.

7. After the loop, you must threshold motorCommand so that it lies in the range [-1,1]:

   ```
   motorCommand = tanh(motorCommand);
   ```

8. You must then expand the range of motorCommand so that it can specify desired angles between [-45°, 45°]:

   ```
   motorCommand = motorCommand*45;
   ```

*Original material was graciously provided by Josh Bongard. Jeff Clune slightly modified it.*
9. Finally, send `motorCommand` to `ActuateJoint(...)`.

10. Compile, run and debug your simulation. If you find that your robot moves and then comes to a stop, you may try increasing the speed and force of the motors. Alternatively, you may have to decrease the simulation step size. I found that a max motor force of 40 N worked well for the position control in `ActuateJoint`:

```cpp
void ActuateJoint(int jointIndex, double desiredAngle, double dt)
{
    ...
    double maxForce = 4.;
    joints[jointIndex]->setMaxMotorImpulse(maxForce);
    ...
}
```

For the velocity control in `ActuateJoint2`, a proportionality constant k of 100 m/(so) worked well on my machine:

```cpp
void ActuateJoint2(int jointIndex, double desiredAngle, double dt)
{
    ...
    double k = 10.;
    double maxForce = 4.;
    joints[jointIndex]->enableAngularMotor(true, k * diff, maxForce);
}
```

11. Now your robot is moving, but it does not look very realistic. Make the robot more realistic by adding some weight, friction and dampening to your robot. For the weight, set the mass of the main body to 6 and set the weight of the cylinders to 3. For the friction, find the lines that create your rigid bodies, probably looking something like this:

```cpp
btRigidBody* body = new btRigidBody(cInfo);
and add,

cInfo.m_friction = 1.5;

just before creating the new body. For dampening, add

```cpp
body->setDamping(0.05, 0.85);
```

right after creating the rigid body. This should make your robot move more realistically, although it also means that your random controller will be less effective at moving your robot around. Note: these are suggested values. You may need to tweak these values to make things work better in your setup.

12. You may find that the particular synapse weights do not produce motion for different settings of the motor speed, force and simulation time step. You can get your code to produce a different sequence of random numbers by setting the random seed at the very beginning of your code using `srand(int s)`. So, `srand(0)` will always produce the same sequence of random numbers every time you run your code, but changing to `srand(1)` will cause a different sequence of random numbers. Try running your code with different initial random seeds, and you should get different motion patterns, one of which will cause the robot to self-displace as in Fig. 1.

13. Once you have a self-displacing robot, capture five images from its motion illustrating that it maintains
a similar pose, but does move, as in Fig. 1. Copy them into your document and submit!

Figure 1: Visual display of the robot with randomly-weighted synapses connecting the four touch sensors to the eight motors.