Modern Robots: Evolutionary Robotics

Programming Assignment 8 of 10

Description

Description: In this week’s assignment you will add sensors to your robot. You will add one binary touch sensor in each of the four lower legs: the sensor will fire when that leg is touching the ground, and will not fire when the leg is off the ground. Adding touch sensors requires four basic steps:

1. Set all touch sensor values to zero at the outset of each time step of the simulation.

2. Add code so that, when a callback is called because there is a collision between a leg and the ground plane, Bullet knows which leg collided.

3. Set the touch sensor for that leg to 1.

4. For those legs touching the ground, draw a red sphere at the contact point so you can visually tell your sensor worked (see Fig. 1).

The following instructions will realize these four steps.

1. Back up Assignment 7 on a flash drive or another computer so that you can always return to your completed seventh assignment.

2. Copy directory Assignment 7, which contains your submitted document and the entire Bullet folder. Rename the new directory Assignment 8.

3. Bullet allows the user to associate a pointer with each Bullet body using the method setUserPointer(void *data). You will create such a data structure, and store the ID of each Bullet body there. That is, the third Bullet object will have an ID of 3. In RagdollDemo.h below the oneStep declaration, create an array of integers int IDs[10] (9 for the body parts, and 1 for the ground).

4. Just after the array is created, set the $i^{th}$ element of the array to $i$.

5. In CreateBox(...) and CreateCylinder(...), after the Bullet body has been created, associate a pointer to the correct element in this array with the object: body[index]->setUserPointer(&IDs[index]);

Also, associate the ground collision object fixedGround with an ID of 9.

6. Now, we need to setup a means of determining which objects are in contact with the ground. There is more than one way of doing this with Bullet. But we will do this by setting up a callback function that is called after every contact point has been processed. Add the following function to RagdollDemo.cpp before the initPhysics method:

*Original material was graciously provided by Josh Bongard. Jeff Clune slightly modified it.
bool myContactProcessedCallback(btManifoldPoint& cp, void* body0, void* body1)
{
    int *ID1, *ID2;
    btCollisionObject* o1 = static_cast<btCollisionObject*>(body0);
    btCollisionObject* o2 = static_cast<btCollisionObject*>(body1);
    ID1 = static_cast<int*>(o1->getUserPointer());
    ID2 = static_cast<int*>(o2->getUserPointer());
    /* Your code will go here. See the next step. */
    return false;
}

You’ve defined this function, but Bullet does not know about it yet. So, add this as the first line to the
initPhysics method:

gContactProcessedCallback = myContactProcessedCallback;

Recompile and run until there are no errors. You won’t see any difference yet.

7. Once you have these IDs, print them from within the callback function to the output terminal:

    printf("ID1 = %d, ID2 = %d\n", *ID1, *ID2);

Recompile and rerun. You should get something like the following when the simulation is running
unpaused:

    ID1 = 9, ID2 = 6
    ID1 = 9, ID2 = 7
    ID1 = 9, ID2 = 5
    ID1 = 9, ID2 = 7

    If the robot is moving randomly (Fig. 1), only the four lower legs (with IDs 5,6,7 and 8) will come into
contact with the ground plane.

8. Now, in RagdollDemo.h, create a new integer array that will store the values of the four touch sensors:

    int touches[10];

    touches[i] will be set to 1 when the touch sensor in the ith body part is firing, and zero otherwise.
    Only touches[5] through touches[8] will be used. Make sure the array is public.

9. Calls to myContactProcessedCallback are made when stepSimulation is called. So, we need to set
all of the touch sensors to zero before calling stepSimulation, and then set all of the touch sensors that are
firing to one during the myContactProcessedCallback calls. So before calling stepSimulation, add a for
loop that sets each element of touches to zero.

10. In myContactProcessedCallback, after the print statement you can turn on the corresponding touch
sensors as follows:

    printf("ID1 = %d, ID2 = %d\n", *ID1, *ID2);
    touches[*ID1] = 1;
    touches[*ID2] = 1;
Unfortunately, the above code won’t work because * touches is a member of the RagdollDemo instance, and the function myContactProcessedCallback does not know about it. So we will fix it by doing the following things:

(a) Above the definition of myContactProcessedCallback in RagdollDemo.cpp declare a new variable:

```cpp
static RagdollDemo* ragdollDemo;
```

(b) Add a line of code to the initPhysics method:

```cpp
void RagdollDemo::initPhysics() ragdollDemo = this; ...
```

(c) Adjust the code in myContactProcessedCallback like so:

```cpp
printf("*ID1 = %d, *ID2 = %d\n", *ID1, *ID2);
ragdollDemo->touches[*ID1] = 1;
ragdollDemo->touches[*ID2] = 1;
```

(d) In RagdollDemo.h add the line

```cpp
public:
before the declaration of touches.
```

This will allow the function myContactProcessedCallback to see the touches member variable. Recompile and rerun. Your touch sensors might be working, but you can’t see them yet.

11. Somewhere within the if ( !pause ) clause of the method clientMoveAndDisplay create a for loop that prints out the values of touches. When the simulation is run it should produce something like:

```
00001101
00000101
...
00001000
```

(You can see this in Fig. 1).

12. Now we want to see these contact points visually. One way to do this is to use the Bullet demo’s built-in key commands: hit ‘c’ and ‘w’ and the black lines show the contact normals. However, we can do better than that.

13. We want to add something more to the rendering. The Bullet Demos have a renderme method that we will use. Add this method declaration to the RagdollDemo.h class:

```cpp
virtual void renderme() {
  extern GLDebugDrawer gDebugDrawer;
  GlutDemoApplication::renderme(); /* Call the parent method. */
  /* Make a circle with a 0.9 radius at (0,0,0) with RGB color (1,0,0)*/
  gDebugDrawer.drawSphere(btVector3(0.,0.,0.), 0.9, btVector3(1., 0.,0.));
}
```

You should see something like Fig. 2. Capture an image of this and copy it into your document.
14. What we want is to put little red spheres at each of our contact points. First we need to capture the information about where those points are. Add two new public members to the RagdollDemo class:

```c++
int touches[10];
bTVector3 touchPoints[10];
```

15. Capture the points in `myContactProcessedCallback` like so:

```c++
ragdollDemo->touches[*ID1] = 1; ragdollDemo->touches[*ID2] = 1;
ragdollDemo->touchPoints[*ID1] = cp.m_positionWorldOnB;
ragdollDemo->touchPoints[*ID2] = cp.m_positionWorldOnB;
```

16. Now, inside the `renderme` method, you can include an if clause that will draw a red sphere with a radius of 0.2 at the \textit{ith} touch point if it is being touched. When you compile and run the simulation, you should see that whenever a lower leg comes in contact with the ground, a red sphere appears, and that sphere then disappears when the leg is lifted above the ground. Capture three images showing different legs doing this as in Fig. 1, copy them into your document and submit it. You do not need to include the output of the touch sensor array in your figure (i.e. the text under each panel is not necessary).

Figure 1: Visual display of the touch sensors firing: a) no sensors; b) all sensors; and c) one sensor.

Figure 2: Draw a sphere at location (0,0,0) into the scene.