Hunter Brochu
CS 206

Tutorial for Creating the Four-legged and Three-legged Ball Moving Robots

In this tutorial you will create a four-legged and three-legged robot and run several evolutionary trials to determine two things: which of the robots performs more effectively and which robot evolves faster? You will design two separate robot simulations. One simulation will alter the original four legged robot simulation to generate a sphere object and communicate it's location to your python code. You will then need to alter the evolutionary algorithm to redefine your fitness value to result in the robot evolving to move the ball further into the Z-axis while also trying to keep up with it. The next step will be to create a three-legged version of our four-legged robot. With this three-legged robot you will need to tweak some of your code in order to make it work properly.

Steps:

1. First you will start by modifying your existing quadruped robot from assignment 10.
2. The first thing that needs to be done is you need to create a spherical object in the simulation for the robots to be able to move around.
3. The easiest way to do this is to create a btSphereShape using the Bullet library. Write a Create_Ball method in your RagdollDemo.h file. You will then call this method in your .cpp file when you are generating the body of the robot. Keep in mind where the ball is spawning. You will want it to start not very far from the robot initially but also not so close that it will immediately start to roll away at the beginning of the trial.

4. A key part of the evolutionary algorithm for this experiment is knowing the position of the sphere at the end of the simulation execution. In order to communicate this information with the hill climber code in your python file you will need to write a Save_Ball method in your RagdollDemo.h file. It should be almost the same as your method for saving the “fit.dat” file since both are creating files that will hold a single number. You will write to this file the Z-coordinate of the sphere at the end of the 1000th time step of the simulation just before it exits.
5. At this point run the simulation a few times to work out the bugs. You will want to get to the point where when the simulation starts the robot and the sphere are sitting near each other in the center of the screen, where the ball is not in motion until the robot knocks into it. If you are seeing the ball rolling at the start of the trial try changing the radius of the ball or it’s distance from the robot at the beginning of the trial.
6. Now lets jump to your python code. The trick here is finding an evolutionary algorithm that will get the results you are looking for. You want to see two big things: you want the robot to move the ball more and more into the Z-axis, but you also don’t want the robot to just kick the ball and have it roll away. You want to see the robot follow the ball while it rolls further along the Z-axis.
7. To do this you will first need to modify your Evolve method in your python code. You will want to call a new Fitness4Get method that will handle generating the new fitness value for this experiment. You will want to change all the calls to your old fitness method to this new one within the Evolve method. This change means that

you can leave the rest of your Evolve method the same since your new fitness function will generate a similar output as your old fitness function did.

8. Next you will need to create this new Fitness4Get function. The first part of the function will be the same as your old fitness function because it is still important to know the position of the robots body when calculating the new fitness value.
9. Next you will need to import the balls location from the ball.dat file you created. The way you do this should be similar to how you got the fitness value from the fit.dat file. Now with both the Z-axis value of the robot and the ball you will need to use them to create a formula that will promote the traits we are looking for. You want the generated value to reward for a higher Z-axis coordinate of the ball, but it will also punish the larger the distance between the ball and the robot. You will then return this new fitness value. You will also want to return the value of the balls location in the Z-axis.
10. Back in the Evolve method in your python code you will need to create a variable for the value of the ball's Z-axis coordinate to pass into from your fitness function. After you have that make it so that when your print out the parent and child fitnesses to the IDLE shell you will now need to also print out the value of the balls location. Now run the python code a few times. You will want to see that the fitness value saves trials that succeed in moving the ball further into the Z-axis while also seeing that in the trials that the robot follows the ball after it starts rolling result in a higher fitness value then ones where the robot goes in a different direction then the ball. You should eventually get an evolved robot that knocks the ball along the Z-axis and then chases after it.

11. Now to create the three-legged robot simulation. A lot of this simulation will be identical to the four-legged robot simulation but with a few differences.
12. The first thing that will need to change obviously is how many legs the robot has. This is easily changed by commenting out the code used to create one of the sets of legs of the four-legged robot. Be wary that by doing this you will most likely need to change some of the values of the loops you created to assign hinges and weights to your legs. Once you change what is necessary you should be able to run the simulation and see a three-legged robot spawn instead of a four-legged one.
13. Another thing that needs to be changed in your code are values associated with the motor strengths of the robots legs. Since this four sided robot only has three legs now it is rather unstable. Adjust your motor impulse values and how often the ActuateJoint method is called until you get a robot that does not fall over at the beginning of a trial. Note that we could change more of the code to create a more stable three-legged robot, but that is not what we are trying to do here. We want to see the three-legged robot evolve to walk with the legs the way they are.
14. Now run your python code so that it executes the three-legged robot simulation instead of the four-legged one.
15. Which robot takes longer to evolve a working controller? Which robot appears more effective at moving the ball? Record a picture of both robots at an early evolved state and also record a picture of both robots in a higher evolved state. Also record pictures of each robots change in parent fitness over time.