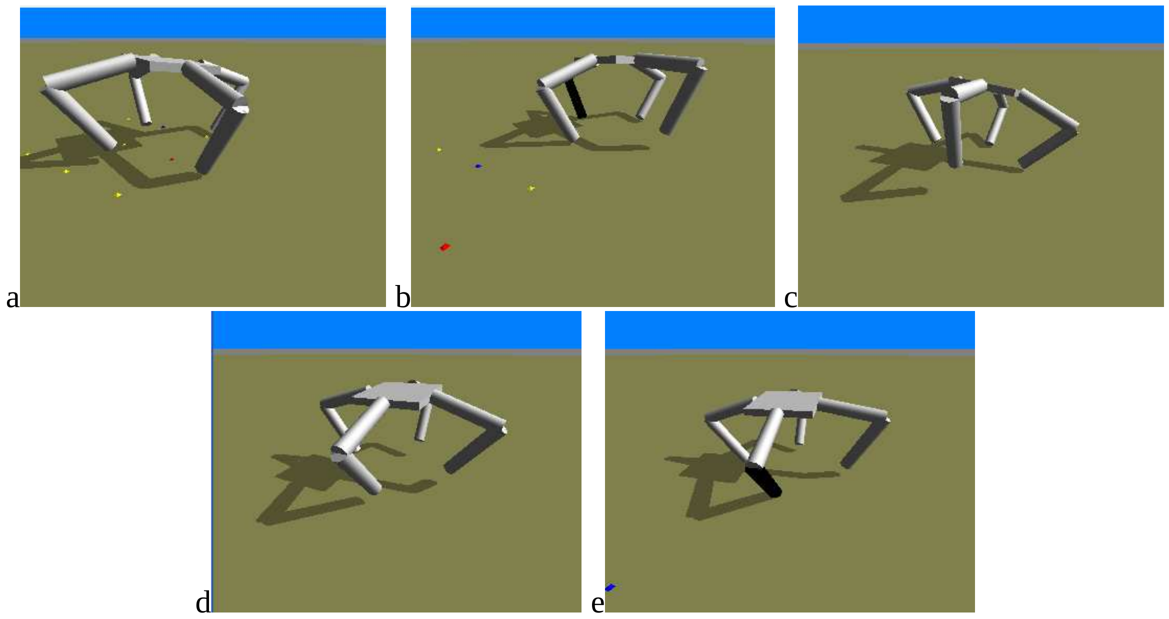

Figure 1: Visual display of the robot with randomly-weighted synapses connecting the four touch sensors to the eight motors.

# CS295/CS395/CSYS395: Evolutionary Robotics

## Programming Assignment 9 of 10

Assigned: Friday, October 28, 2011

Due: Friday, November 4, 2011 by midnight

**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 ODE folder. Rename the new directory Assignment_9.

3. Create a global variable that is a two dimensional array at the top of your code:
   `double weights[4][8];`

4. At the beginning of your `main` function, 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 `simLoop`, just before you call `ActuateJoint(i,motorCommand)` (which sends 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. During each pass through the loop, the value of that touch sensor is multiplied by the weight of the corresponding synapse, and added 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 $[-45^o, 45^o]$:
   `motorCommand = motorCommand*45;`

9. Finally, send `motorCommand` to `ActuateJoint(...)`.

10. Before running your code, move the lines that call dSpaceCollide, dWorldStep and dJointGroupEmpty in simLoop into the following sequence:

    ```
    dSpaceCollide (space,0,&nearCallback);
    ...code that sets the desired angles for all eight motors...
    dWorldStep (world,0.005);
    dJointGroupEmpty(contactgroup);
    ```

    Last week I mistakenly explained that calls to `simLoop` alternate with calls to `nearCallback`. In fact the `dSpaceCollide` triggers zero, one or more calls to `nearCallback` from within `simLoop`, depending on the proximity of pairs of objects.

11. 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 motor force of $10$ and a speed of $1$, with a step size of 0.05 worked well on my machine:

    ```
    double force = 10.0;
    double speed = 1.0;
    dJointSetHingeParam(joints[jointIndex],dParamFMax,force);
    dJointSetHingeParam(joints[jointIndex],dParamVel,speed*error);
    ```

12. You may also 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 to the T.A.