CS295/CS395/CSYS395: Evolutionary Robotics

Programming Assignment 5 of 10

Assigned: Friday, September 30, 2011

Due: Friday, October 7, 2011 by midnight

Description: In this assignment you will start with the 'empty' simulation you created in assignment 4 and incrementally add objects to construct a quadrupedal robot. In the next assignment you will add joints to your simulation so that the objects are connected together, then sensors, then motors, and then a controlling neural network.

1. Create a directory, Assignment_4, that contains your assignment 4 submitted document. Move the ode-0.11.1 directory into this directory as well. Back up this directory. Now in subsequent assignments if you find your simulation becomes unusable you can go back and retrieve the 'empty' simulation stored in this directory.

2. Create a new directory, Assignment_5, and copy the ode-0.11.1 directory here. For the remainder of this assignment make changes to the files in this ode directory.

3. Draw the image shown in Fig. 1 on a blank sheet of paper. Draw to fill an entire page, as you will be annotating the image through the next few assignments.

4. Now draw the robot from the three additional perspectives as we did in class.

5. For the four upper and four lower legs, mark their positions, sizes (radius and length) and orientations (see Lecture 6, slide 4) in the three panels. You can choose the lengths and radii of the legs to your liking.

6. Now, in the empty demo_buggy program, we need to create body and geom data structures to store all the objects that make up the robot. Replace

```
static dWorldID world;
static dSpaceID space;
static dBodyID body[4];
static dJointID joint[3]; // joint[0] is the front wheel
static dJointGroupID contactgroup;
static dGeomID ground;
static dSpaceID car_space;
static dGeomID box[1];
static dGeomID sphere[3];
static dGeomID ground_box;
```

with

```
static dWorldID world;
static dSpaceID space;
static dJointGroupID contactgroup;
```

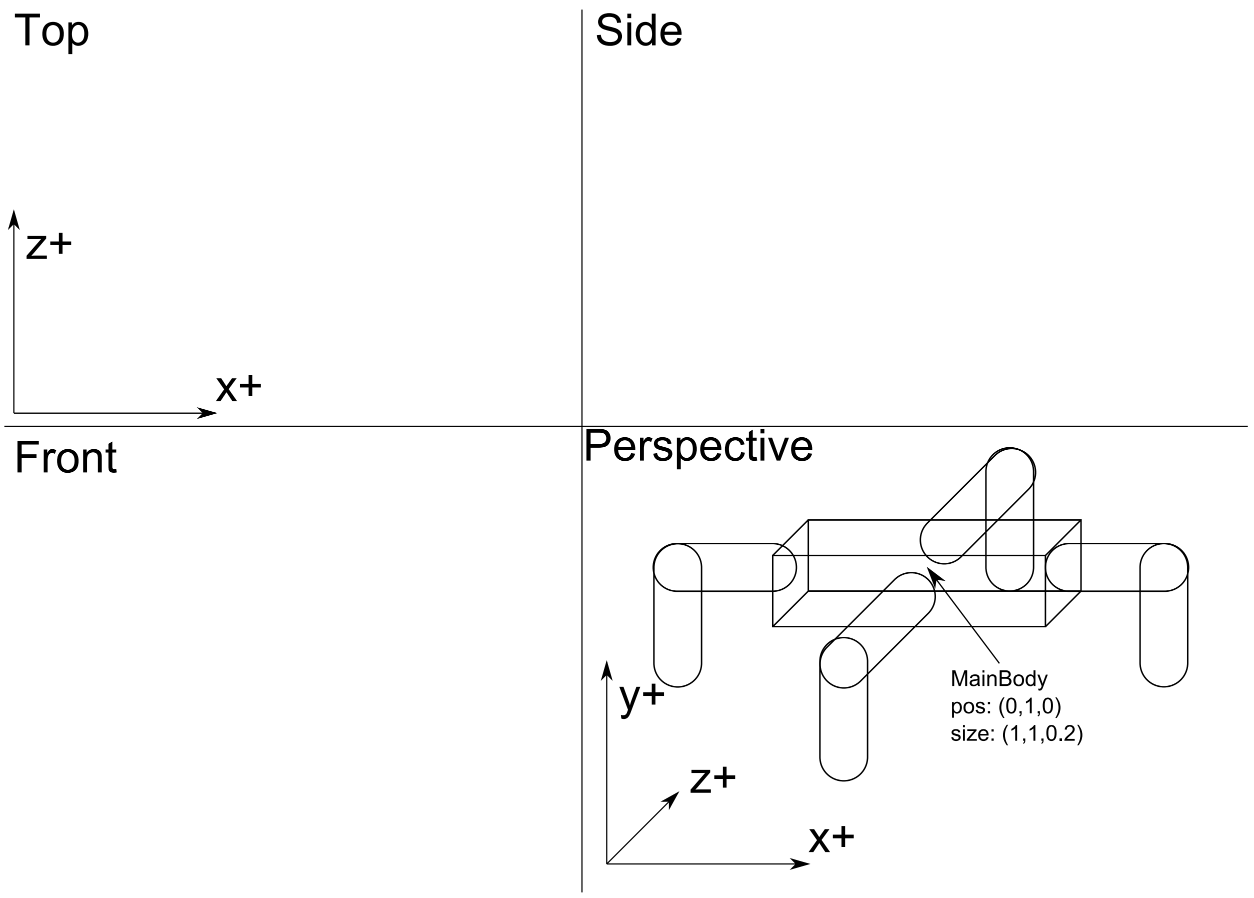

Figure 1: A template for sketching the quadrupedal robot to be simulated.

```
static dBodyID body[9]; // one main body, 4x2 leg segments
static dGeomID geom[9];
static dGeomID ground;
```

7. Recompile the code. If you get errors, delete those parts of the code that refer to the variables you deleted. Continue this process until you compile and run without errors.

8. Now you will create some functions that can be used to add objects to the empty simulator. Create a function of the form
```
void CreateBox( int index,
double x, double y, double z,
double length, double width, double height) {
...
body[index] = ...
...
geom[index] = ...
...
}
```

which will be used to create the main body of the robot. Refer to the commented-out code to figure out how to create an object. Assume the mass of all objects for now is 1. Without calling the function, recompile and run until you have no errors.

9. Create a similar function for creating cylinders, which will be used to create the upper and lower legs of the robot. You will need to add additional parameters for specifying the orientation of the cylinder. Recompile and run until error-free.

10. Create another function
    ```
    void DeleteObject( int index ) {
    ...
    }
    ```
    which can be used to remove objects from the simulation. This function should destroy both the body and the geom associated with the object using `dBodyDestroy` and `dGeomDestroy`. Recompile and run until error-free.

11. Now, in the main function, add code to create the first object, simulate the 'robot', and delete the first object when the simulation terminates:
    ```
    CreateBox(...);
    dsSimulationLoop (argc,argv,352,288,&fn);
    DeleteObject(...);
    ```
    Compile and run until error-free. Note that you will not see the object; why?

12. Two reasons: The simLoop function is commented out, and we have not specified that the object should be drawn, only simulated. Uncomment
    ```
    dSpaceCollide (space,0,&nearCallback);
    dWorldStep (world,0.05);
    dJointGroupEmpty(contactgroup);
    ```
    in simLoop, recompile and run.

13. Create a new function `DrawBox(int index)` that will draw a box defined by `body[index]` and `geom[index]`. Refer to the code commented out at the bottom of `simLoop` to see how to do this. Without calling the new function, compile and run until error-free.

14. Add a line at the bottom of simLoop
    ```
    DrawBox(0);
    ```
    that will draw the first object during each pass through simLoop. Recompile and, when run, you should see the box fall and come to rest on the ground as in Fig. 2a. (You can toggle texture drawing using CTRL-t.) Screencapture this image and paste into your document.

15. The object will have fallen too fast for you to follow it with your eyes. In order to start simulations paused and then step through them, modify simLoop to
    ```
    if ( !pause ) {
    dSpaceCollide (space,0,&nearCallback);
    dWorldStep (world,0.05);
    dJointGroupEmpty(contactgroup);
    }
    ```

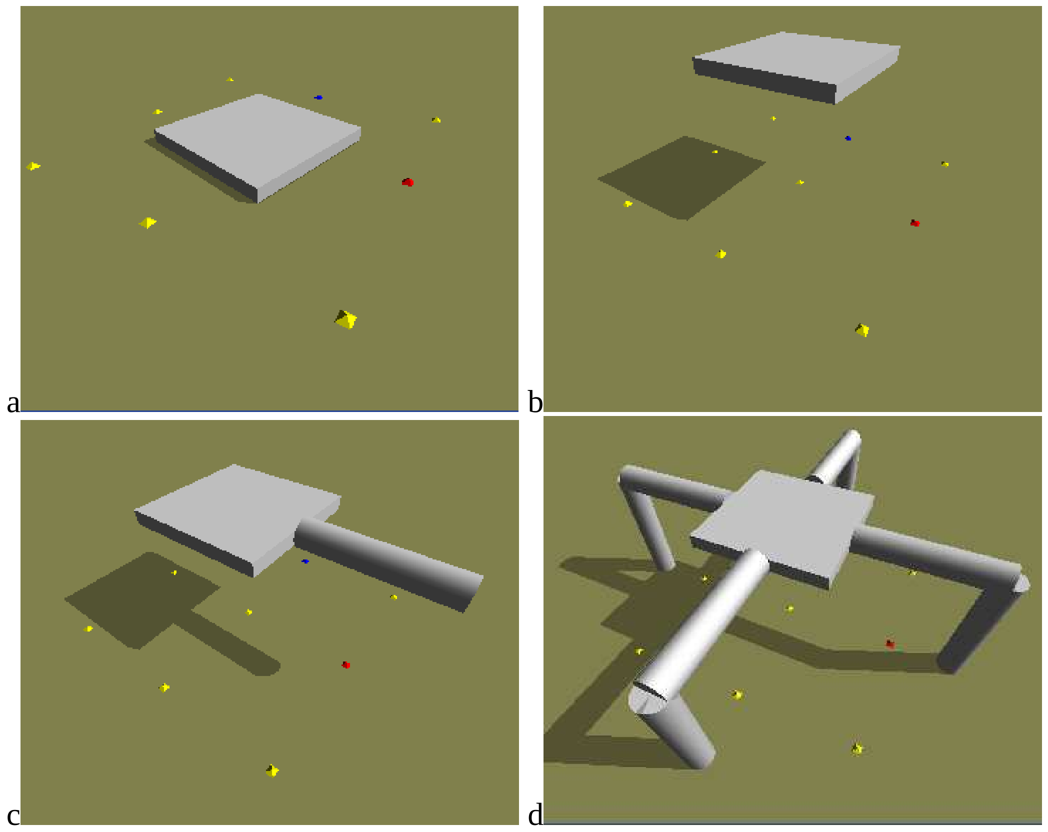

Figure 2: The quadrupedal robot under construction in ODE.

```
DrawBox(0);
```

16. When you run test_buggy, add `-pause` to start the program paused. Consult your compiler's documentation to determine how to add command arguments. When you re-run the program you should see the box hanging in midair as in Fig. 2b. Screencapture and paste into your document. CTRL-o will advance the simulation by one time step; CTRL-p will unpause the simulation and cause the object to fall to the ground.

17. Add `CreateCylinder(1,...)` before simLoop is called, `DeleteObject(1)` after simLoop ends, and `DrawCylinder(1)` within simLoop to add the next object to the robot. Note that you will have to specify the orientation of the cylinder, which you do not have to do for the main body. When run in paused mode you should see both objects as in Fig. 2c. Screencapture and copy and paste into your document.

18. Add a third object, compile, run and ensure that the object appears where you expect it. Continue to add an object and recompile until all nine objects are added. This should produce a simulation as shown in Fig. 2d. Screencapture, copy and paste into your document, and submit.