CS 4/6545 Autonomous Robotics: Feedback control

Read all instructions carefully. Your submission will consist of all code and plots.

1. Install a C++ compiler. If you are using *Ubuntu*, do:

   ```
   tmp $ sudo apt-get install g++
   ```

   On *Mac OS X*, you'll need to install *X Code* and the command line development tools.

2. Install *Gazebo*, *Python*, *matplotlib*, and *cmake* on your computer. If you are using *Ubuntu*, installation of the latter three are as simple as:

   ```
   tmp $ sudo apt-get install python cmake python-matplotlib
   ```

   If you are using `Mac OS X`, binary packages of the latter three are available (search for them!).

   Go to `http://gazebosim.org/download` and follow the instructions for installing *Gazebo*.

3. Your first goal is to get the double pendulum to move to the position $\phi_1 = 0, \phi_2 = 0$ (the pendulum starts at $\phi_1 = -\frac{\pi}{2}, \phi_2 = 0$: the pendulum will have to fight gravity to do this. Implement a proportional-derivative (PD) controller in `PD_hold.cpp`, and select gains for KP and KV (look for "TODO"s; note that 0.0 is used as a placeholder so you can get the plugin to compile immediately).

4. Build the plugin:

   ```
   feedback_control $ mkdir build
   feedback_control $ cd build
   build $ cmake ..
   build $ make
   build $ cd ..
   ```

5. You have to set environment variables to tell *Gazebo* where to find the necessary models for this project (`double_pendulum_bobs`) and also where to find the plugins. On both *Linux* and *Mac OS X*, you can set the environment variables like follows (you'll obviously need to customize this for your system):

   ```
   feedback_control $ export GAZEBO_MODEL_PATH=/home/drum/AuRo/feedback_control
   feedback_control $ export GAZEBO_PLUGIN_PATH=/home/drum/AuRo/feedback_control/build
   ```

   Note that I'm assuming that your shell is the default (*bash*)—you need to use a different command for a different shell. To verify that your shell is indeed *bash*, you can do:

   ```
   feedback_control $ echo $SHELL
   /bin/bash
   ```

   If it doesn't say `/bin/bash`, you'll need to search for how to set environment variables for that particular shell.

6. Now run *Gazebo* on the `PD_hold` world. We're going to use the "verbose" option, so *Gazebo* will inform us if the plugin is missing:

```
feedback_control $ gazebo -u --verbose PD_hold.world
```

The "-u" option pauses *Gazebo* when it starts, so you can see the controller acting at the very beginning (otherwise, the simulator runs for a few seconds before you see anything). Press the play button to start the simulation.

7. The plugin generates data of the joint angles over time. Plot ten seconds of data using:

```
feedback_control $ python plot_hold_PD.py
```

Use this plot to tune your PD gains so that you improve performance further: you should be able to get the pendulum within an error band of 0.05. Identify the rise time, peak time, overshoot, and settling time (see Figure 1) by modifying `plot_hold_PD.py`. *Note that* matplotlib *generates a PNG file every time you plot, which will make it easy to hand in your finished plots for homework.*

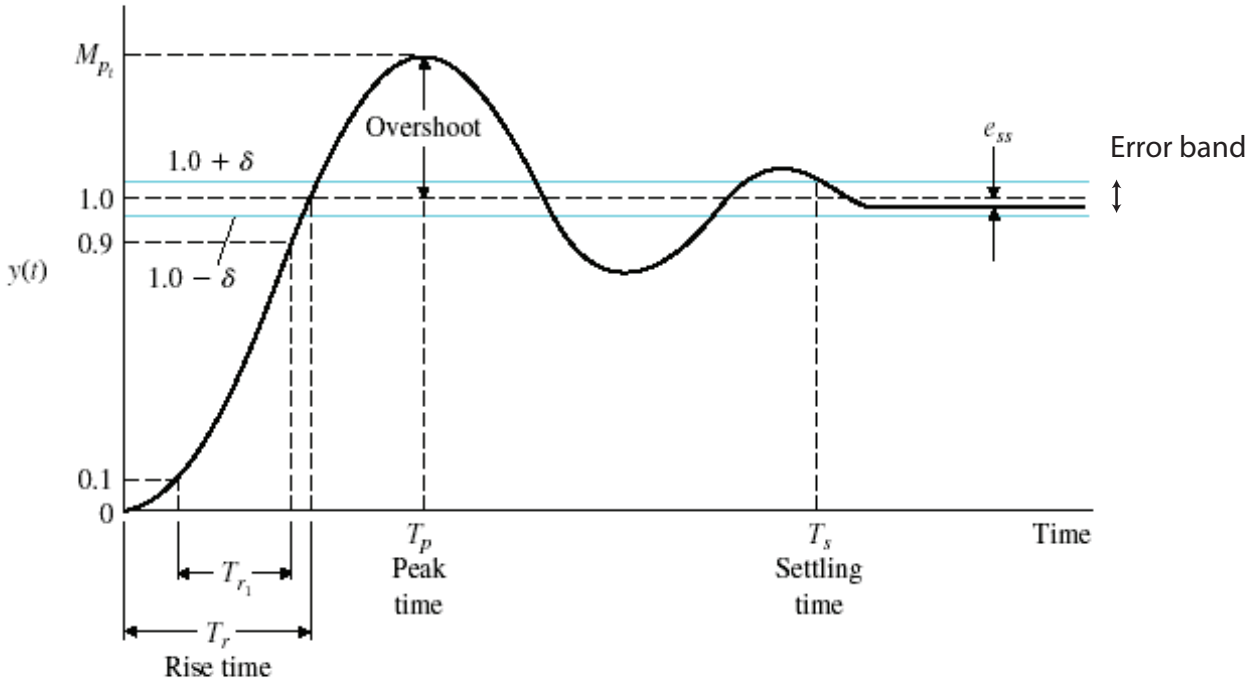

Figure 1: Depiction of rise time, peak time, overshoot, settling time, and error band for a feedback controller attempting to maintain a set point of 1.0.

8. Implement a PID controller in `PID_hold.cpp` (look for "TODO"s). Re-make:

```
feedback_control $ cd build
build $ make
build $ cd ..
```

9. Now run *Gazebo* on the `PID_hold` world:

```
feedback_control $ gazebo --verbose PID_hold.world
```

10. Plot ten seconds of data using:

```
feedback_control $ python plot_hold_PID.py
```

Use this plot to tune your PD gains further so that you improve performance on the PD controller. Identify the rise time, peak time, overshoot, and settling time by modifying `plot_hold_PID.py` (look for "TODO"s).

11. Copy `PD_hold.cpp` to `track_sinusoidal.cpp`. Alter the desired positions for the joints in `track_sinusoidal.cpp` to $\phi_1 = \sin t, \phi_2 = \sin 3t$. Re-make:

```
feedback_control $ cd build
build $ make
build $ cd ..
```

12. Run *Gazebo* on the `track_sinusoidal` world:

```
feedback_control $ gazebo --verbose track_sinusoidal.world
```

13. Plot the desired *vs.* actual joint position over ten seconds of motion using:

```
feedback_control $ python plot_track_sinusoidal.py
```

14. Tune your PD gains such that you attain reasonable tracking performance, using the plotting mechanism to improve performance iteratively.