ECE 383 / MEMS 442 Fall 2015 Final Exam

Deadline: December 12 at 7pm

Instructions: collect your written answers in a `final.pdf` file, and write your code in `stateestimator.py`, `controller.py`, and any other relevant files needed to run your code. Zip and submit the package on Sakai before the deadline.

You are NOT permitted to consult other students or the TAs for help during this exam. Clarifying questions are allowed, as long as they are posted on the course Piazza site and are public.

Summary: the Klamp't Olympics

In this final exam, you will design and develop a robot's control system to perform in an "event" in the Klamp't Olympics. You will be assigned an event (A, B, or C) which will designate which task that you will work on.

Event A (Robot Goalie): In your event, balls will be hurled toward a goal, and your robot should block as many of them as possible from passing through the goal. Your program will be started by running `finalA.py`.

Event B (Robot Catcher): In your event, balls will be lobbed toward you, and your robot must catch as many of them as possible. Your program will be started by running `finalB.py`.

Event C (Robot Batter): In your event, you will need to hit as many balls into a goal as possible, while obstacles try to block the ball's path. Your program will be started by running `finalC.py`.

Your robot's performance on an event will be judged by an *event score*, which in event A subtracts -10 points for every ball scored, in event B adds +10 points for every ball caught, and in event C adds +10 points for every ball scored. Points are subtracted for various penalties, including collisions, joint limit violations, taking too long, etc.

The final will consist of a written portion (70 points), in which you describe the design and technical details of your system, as well as a programming portion, in which you implement the proposed system (30 points). Furthermore, the student in each event that gets the highest event score will receive 5 points extra credit.

Note: if your implementation is incomplete you will not be penalized on the written portion of this exam. It is still possible to achieve full credit for a written portion that is complete, technically correct, and logically sound even without an implementation. Also, if your implementation of the perception subsystem is incomplete, you will not be penalized on the controller implementation portion of this exam. However, you will not be eligible for extra credit points.

Robot and Sensors

The robot used in this project is our standard 6DOF fixed-base industrial robot, affixed with a paddle, scoop, or bat attachment, depending on your event. The low-level controller drives its motors using a PID controller with gravity compensation. You will be penalized for collisions between the robot and the playing field (20 points per second of violation), and for violating the robot's joint limits or its torque limits (10 points per second of violation).

Two types of sensors are available for you to use: 1) the robot's joint encoders and 2) a simulated camera sensor. The camera is located at a fixed location off of the robot. The coordinates of the camera frame with respect to the world frame is known, and its origin is at the camera focal point, its x direction points in the "right" direction of the camera image, and the y direction points in the "down" direction of the camera image.

Rather than directly accessing the camera's pixels, you will have access to a "blob detector" which produces rectangular windows of detected "blobs" that indicate various objects of interest. In later stages of this project you will use these blobs to estimate object positions (and possibly velocities) in 3D space. However, in early stages of this project we will provide "omniscient," precise readings of object positions and velocities. This will help you debug your high-level controller.

In Events A and B, each subsequent ball will be of a different color. (There may be multiple balls in play at once.) In Event C, the ball will be red (RGB color (1,0,0)) and the other obstacles will be of different non-white colors.

High-Level Controller

Your primary job is to design a high-level controller to make your robot perform the event as best as possible. This involves implementing a control loop in which you read from sensors, process the sensor information, and then send a command to the robot's low-level controller. Your code will most likely be placed in the `myPlayerLogic()` method of the `MyController` class in controller.py. If you need to do custom initialization, you may wish to modify the `reset()` method, and if you wish to visually debug your controller, you may wish to modify the `drawGL()` method.

The processing will involve perception (in particular, state estimation of moving objects), control (in particular, managing the internal state of different control components) and planning (deciding where and how to move in response to detected objects, and the robot's current state).

You will be penalized for excessively long control loops. One point will be subtracted from your event score every time your controller takes more than 1s calculation per 1s of execution time If it takes more than 5s calculation per 1s of execution time, 5 points will be subtracted from your score, and the run will be terminated.

Manual Controller and Simulation Control

When designing your system, it may be helpful to do manual control of your robot. To turn on manual control, press 'u' to switch into User Mode. Your controller's commands will be ignored now. Now, press keys 1, 2, 3, 4, 5, or 6 to increase the robot's corresponding joint angle, or q, w, e, r, t, or y to decrease the corresponding joint angle.

It may also be useful to pause the simulation and take single simulation steps. To do so, press the 's' key to pause the simulation, and then press [space] to step the simulation forward. Pressing 's' again turns the simulation back on.

A final useful tool might be the Klamp't resource.py module. If you write `q = resource.get('NAME.config',world=sim.world)`, you will be able to manually pose the robot into the configuration that you want. It will then be returned in the value q for use elsewhere in your controller, and moreover it will be saved to disk under the name 'NAME.config'. Then, every time you run your program, q will be loaded from 'NAME.config'.

If you want to re-edit your resource, you can either delete it from disk, or call `q = resource.get('NAME.config',world=sim.world,doedit=True)`. You can also edit points and transforms using this method. You can also use the RobotPose program built into Klamp't to edit resources. See http://motion.pratt.duke.edu/klampt/pyklampt_docs/namespaceklampt_1_1resource.html for more information.

Note: if you use the resource editor, don't forget to package up your saved resources along with your code when submitting your exam.

1. System Design (Written)

A. (15 pts) **Summary**. Design your system overall, including perception, planning, and control components. Summarize the approach you have taken. As part of this summary, include a block diagram naming each component as well as information flow between each component. Each block is a component, and each arc (arrow) is a piece of information. Label each block and arc.

B. (15 pts) **Components**. For each component and subcomponent, describe in precise, technical language:
   a. Its purpose, including how it relates to neighboring components
   b. Its inputs, including the data type, format, and reference frame (if applicable)
   c. Its output, including the data type, format, and reference frame (if applicable)
   d. How it is invoked. (Is it run at a constant rate? On request?)
   e. How it will be implemented. (What algorithm? Will it be an off-the-shelf or custom implementation?)
   f. Potential failure cases

C. (15 pts) **Planning and Control Strategy.** Describe your planning / control components in more detail. As part of this description, include a state machine diagram. The diagram should document, by appropriate naming and/or clear annotation, how the high-level controller processes perception inputs and invokes planning components to produce commands for the low-level controller. How does it handle planning failures?

D. (15 pts) **Perception Strategy.** Describe your perception (i.e., state estimation) components in more detail. Do you need to estimate all components of 3D object positions and velocities in order to perform your task? To what accuracy do you expect to measure them? What strategy or algorithm do you use to implement these components? If there is a mathematical model you use, state this model. For example, if you are using a Bayesian filter, describe the transition model, observation model, and belief initialization.

E. (10 pts) **Reflection.** After implementing your system, write 1-2 paragraphs describing how well the system worked, what you could have improved, what was harder than expected, etc. Also, describe what additional challenges you might face when designing a robot to perform your event in real life rather than in simulation.

2. System Implementation

The programming portion of this exam consists of two independent subsystems. Part A is the high-level controller subsystem, and code will be located mostly in `controller.py`. Part B is the perception (i.e., state estimation) subsystem, and code will be located mostly in `stateestimation.py`.

It is possible to work on Part A without having Part B implemented. The "`omniscient = True`" flag in `final[X].py` provides your controller access to the true state of the world, circumventing the need for state estimation.

A. (15 pts) Implement the high-level controller subsystem. You can test it on easy, medium, and hard instances of your problem by running "`python final[X].py [difficulty]`" where "difficulty" is one of "easy", "medium", or "hard" (without quotes).

   By default, the event is randomized with a fixed random seed. This tests your event deterministically which may help you eliminate bugs. To test your event more generally, you can uncomment the line "`random.seed()`" in final[X].py. (This call seeds the random number generator with the current time, ensuring a new random event every time the program is run).

B. (15 pts) Implement the object state estimation subsystem. You are free to copy the sensor transform `Tsensor` found in final[X].py into your own code. You are also free to use the Kalman filtering code provided for you in `kalmanfilter.py`, but you are under no obligation to use it.

   When you are ready to test your state estimation system in practice, you should set "`omniscient = False`" at the top of the file and comment out the line "`multiObjectStateEstimate = omniscientObjectState`" in `controller.py`.

(Note: those of you in Event C will want to see the TODO under `MyController.drawGL()` to get better feedback about how well your state estimator is doing)

The Main Event

The Klamp't Olympics will be held in Teer 203 during our scheduled final exam time, Dec. 12 at 7pm. Please let the instructors know *by noon on Dec. 12* whether you would like to demo your entry (you do not have to be present, nor does your entry need to be complete). Besides individual event winners, a "most creative" award will be given. Refreshments will be served.