Welcome!

Course goal:
- Develop a pragmatic understanding of both theoretical principles and real-world issues, enabling you to design and program robotic systems incorporating sensing, planning, and acting.

Course topics:
- Kinematics
- Inverse Kinematics
- Sensors & Sensor Processing
- Motors & Control
- Planning
- State Estimation
- Embedded Systems
Course Structure

- Labs with milestones
  - Arm lab, Mobile Robot lab
  - Staff-assigned groups
  - Peer Evaluations
  
- Two midterms
  - 5% bonus policy

- Two quizzes

- Final Projects
  - Student-selected groups
Course Policies

- **Late Policy**
  - Labs due at 5p
  - 10% per day late penalty
  - No credit after three days

- **Lab Policy**
  - Food restricted
  - Removal of equipment forbidden
  - Accidents, broken equipment

- **Make-up exams**
  - Quizzes: none
  - Midterms: orals
## Lab Schedule

We share this lab with EECS 373!

<table>
<thead>
<tr>
<th><code> </code></th>
<th>Monday</th>
<th>Tuesday</th>
<th>Wednesday</th>
<th>Thursday</th>
<th>Friday</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Labture (9:30)</td>
<td></td>
<td>Labture (9:30)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Labture</td>
<td></td>
<td>Labture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Labture</td>
<td></td>
<td>Labture</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>(chardson)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(chardson)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>(ebolson, CSE3737)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>CLOSED</td>
<td>CLOSED</td>
<td></td>
<td>(ebolson, CSE3737)</td>
</tr>
<tr>
<td>4</td>
<td>Peak</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>Peak</td>
</tr>
<tr>
<td>5</td>
<td>Peak</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>Peak</td>
</tr>
<tr>
<td>6</td>
<td>Peak</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>CLOSED</td>
<td>Peak</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Communications

- Email lists
  - eecs498-staff@april.eecs.umich.edu
  - eecs498@april.eecs.umich.edu

- Wiki
  - http://april.eecs.umich.edu/courses/eecs498_f09
Teams

- Team dynamics
  - Working on a team is an engineering problem itself
  - Discuss with your team how you will handle:
    - Scheduling of group meetings
    - Disagreements about what to do
    - Problems with load balancing

- Peer Evaluations
Teams

- **Team 1**
  - clba, bechi, rcanebala, rgcarole

- **Team 2**
  - ajgden, egnorka, afeld, mgambril

- **Team 3**
  - jgrohosk, hammonca, hockeyc, huxuli

- **Team 4**
  - brokenex, loveara, mtmccann, tmccurdy

- **Team 5**
  - joberlin, keegreil, brrubin, kshahid

- **Team 6**
  - raymasmi, starchmd, jibble, TBD

- **Team 7**
  - wanyef, rwolcott, tzusman, TBD
Lab 1 Overview

- M1 (due Wed 9/16).
  - Derive the forward kinematics for the arm
  - Implement a visualization that shows the real-time position of the arm and allows the human operator to manually move the joints.

- M2 (due Mon 9/28)
  - Implement a ball finding system using an overhead camera.
  - Implement an inverse kinematics system employing gradient descent.
  - Implement an inverse kinematics system employing a closed-form analytic solution

- M3 (due Wed 10/7)
  - Implement an RRT-based motion planning system that allows your arm to pick up balls while avoiding obstacles.
  - Participate in an informal competition with other teams to see whose system can pick up the most balls in two minutes.
Lab 1 Milestone 1

☐ (demo)
LCM
Essential infrastructure tools

- Message passing
  - Modularity
    - Encourages abstraction and decomposition of large problems into well-defined sub-problems
    - Software reuse
    - Fault tolerance
    - Creates viewports into system’s internal operation

- Logging, Playback
Example

- MIT DUC
  - 40 CPU cores
  - 22+ distinct modules
  - 60+ module instances
Modularization Example

- **Laser driver**
  - `laser_t`

- **Red Ball Finder**
  - `object_position_t`

- **Obstacle Tracker**
  - `laser_t`
  - `obstacles_t`

- **Game Pad Drive**
  - `motor_t`

- **Motion Planner**
  - `object_position_t`
  - `motion_planner_t`

- **Path Follower**
  - `motion_plan_t`
  - `motor_t`

- **Splinter**
  - `pose_t`

**LCM Network**

- Connections between modules indicate data exchange:
  - Laser driver to Red Ball Finder: `laser_t` to `object_position_t`
  - Red Ball Finder to Obstacle Tracker: `object_position_t` to `laser_t`
  - Obstacle Tracker to Game Pad Drive: `laser_t` to `motor_t`
object_position_t.lcm

```c
struct object_position_t {
    int64_t utime;
    double distance; // distance to object
    double theta; // direction to object
}
```

```python
objectpos = new object_position_t();
objectpos.utime = System.currentTimeMillis()*1000;
objectpos.distance = 0.3;
objectpos.theta = -0.12;
```
laser_t.lcm

struct laser_t
{
    int64_t utime;

    // range data (meters)
    int32_t nranges;
    float ranges[nranges];

    // intensity data, in sensor-specific units
    int32_t nintensities;
    float intensities[nintensities];

    // the angle (in radians) to the first point in nranges,
    // relative to the laser scanner’s own coordinate frame.
    float rad0;

    // the number of radians between each successive sample
    float radstep;
}
public class RedBallChaser implements LCMSubscriber
{
    LCM lcm = LCM.getSingleton();

    public RedBallChaser()
    {
        lcm.subscribe("RED_BALL", this);
    }

    public void messageReceived(String channel, LCMDataInput ins)
    {
        try {
            if (channel.equals("RED_BALL")) {
                red_ball_t redball = new red_ball_t(ins);

                motor_t motor = new motor_t();
                double K = 10;
                motor.leftWheelSpeed = redball.distance - K*Math.sin(redball.theta);
                motor.rightWheelSpeed = redball.distance + K*Math.sin(redball.theta);

                lcm.publish("MOTOR_COMMAND", motor);
            }
        } catch (IOException ex) {
        }
    }
}
LCM Logging

- (demo)
Once system is decomposed into modules, you can freely mix logged data sources, simulated data sources, and “live” modules.

Example:
- How do you test a robot (e.g. DUC vehicle)?
  - Perception
    - logged data + “live” processing
  - Control
    - simulated data + “live” control + simulated vehicle
    - logged data + “live” control (but actions not executed)

This sort of mixed-mode off-line testing comprised the *vast majority* of DUC development.
Vis
Visualization is *the single best use of researcher time.*
- Find bugs faster
- Verify algorithms and build intuition
- Generate figures/movies for papers/talks.

Matlab strength:
- Ability to easily plot just about anything with very little effort
Vis: Overview

- High-Level 3D visualization library for Java
  - Implementation uses OpenGL: quite fast

- Plot almost anything, almost anywhere.

- Convenient features:
  - E.g., right click, “Make movie”
Vis: Architecture

VisWorld: the set of objects in the world

VisWorld.Buffer
- Name: “my buffer”
- Front: VisObject
- Back: VisObject

VisView: where is our eye/camera?
- eye: double[3]
- lookAt: double[3]
- up: double[3]

VisCanvas: The JComponent that renders the objects in the world according to the current view

EventHandlers
- VisCanvasEvent_Handler
public class VisExample {
    VisWorld vw = new VisWorld();
    VisCanvas vc = new VisCanvas(vw);

    JFrame jf;

    public VisExample() {
        jf = new JFrame("VisExample");
        jf.setLayout(new BorderLayout());
        jf.add(vc, BorderLayout.CENTER);

        jf.setSize(600, 400);
        jf.setVisible(true);

        ArrayList points = new ArrayList();
        points.add(new double[]{0, 0});
        points.add(new double[]{0, 1});
        points.add(new double[]{1, 1});

        VisData vd = new VisData(points,
                                    new VisDataPointStyle(Color.blue, 4),
                                    new VisDataFillStyle(Color.red));

        VisWorld.Buffer vb = vw.getBuffer("mybuffer");
        vb.addBuffered(vd);
        vb.switchBuffer();
    }
}
public class VisExampleRobot {

    VisWorld vw = new VisWorld();
    VisCanvas vc = new VisCanvas(vw);
    JFramejf;

    public VisExampleRobot()
    {
        jf = new JFrame("VisExampleRobot");
        jf.setLayout(new BorderLayout());
        jf.add(vc, BorderLayout.CENTER);

        jf.setSize(600, 400);
        jf.setVisible(true);

        VisWorld.Buffer vb = vw.getBuffer("robot");

        while(true) {
            for (int x = 0; x < 10; x++) {
                double xy[] = new double[] {x, 0, 0};

                vb.addBuffered(new VisChain(LinAlg.xytToMatrix(xyt), new VisRobot()));
                vb.switchBuffer();

                try {
                    Thread.sleep(100);
                } catch (InterruptedException ex) {

                }

            }
        }
    }

public class VisExampleRobot {
Working at home

- Simulation
  - `java april.ax12.AX12ArmDriver sim`

- Software setup
  - VMs
  - ATI
  - Classpath
  - `java.library.path`
Next Time…