Agenda

- General Updates
- ModelZ
- Basic motor control, sensors
- Java API
- Wall-follower
- Midterm
General Updates

- Package updates (vim, gvim, my .vimrc)
- ARL software updates pushed
- Window and terminal settings (!hover_selection)
- Personal machine setup notes on wiki (L2M0)
Netbooks

- *Never* to leave the lab
- Battery doesn’t last incredibly long
- Setup for ARL software, just like desktops
- Internet: CAEN wireless, personal web authentication
  - For L2M2, internal wifi network (no internet)
  - Wired connections appear to be on alternate network
  - Wireless is typically off, use the button and indicator (above) to enable
  - Our ethernet cables don’t always lock. Trouble? Check this first
- Vis too slow? Use `-Dvis.faster=true` at runtime (we can add this to the `~/.bashrc` if you like)
Class, meet ModelZ
Model Z

- Model Z is an inexpensive, rapidly-manufacturable robot
  - Four IR range finders
  - Bi-directional motors (differential drive)
    - Sensors for current (≈torque), angular rate
  - 6DOF Inertial measurement
  - Powered by uOrc
    - Open source robotics controller
    - Interfaces to laptop via ethernet
  - Optional: Soda-can arm, camera, ultrasound
ModelZ: Hardware Notes

- IR receivers numbered 0-3 (L, FL, FR, R)
- Remember to power off!
- Battery lasts a long time
Motors and Encoders
Rotary Encoders

- Single channel encoders
  - Provide speed (not velocity!)
  - Direction is ambiguous
- Quadrature phase encoders
  - Provide speed and direction
Driving Motors

- **In theory:**
  - Drive motor with some voltage $[-V_{\text{max}}, V_{\text{max}}]$  
  - More voltage $\rightarrow$ More current $\rightarrow$ More torque  
    (Simplified model ignoring transients. EECS461 covers detailed motor models. Ask for detail)
  - Voltage $\propto$ Velocity
  - Current $\propto$ Torque
- **In reality, hard to generate low-impedance analog signals**
  - **Solution:** Pulse-width modulation
Pulse-Width Modulation

- Quickly alternate input voltage between 0 and $V_{\text{max}}$

- Why does this work?
  - Motor is a low-pass filter

- Which switching frequency do we want to use?
Odometry

- Constraint: the two wheels travel on concentric circles
- Lab2M1T3.1: Solve for $\Delta x, \Delta y, \Delta \theta$ (need 3 equations)
Interface, Java API, etc
uOrc Interface

- uOrc is an ethernet device
  - DHCPd server: plug ‘n play
- Commands received via UDP-based protocol
- All low-level details are handled by Java libraries
Java API

- orc.Orc represents a connection to the uOrc board (to which our netbook is connected)
  - Orc orc = Orc.makeOrc();

- Sensors and actuators are represented by objects whose constructors take an ‘orc’ object
  - Motor motorLeft = new Motor(orc, 0, true);

- See template on wiki (Lab2, Milestone0)
Software Tips

- A few key function calls
  - Orc
  - Motor
  - WiiMotionPlus
  - IRRangeFinder
- uOrc board is open source, you can `svn co` the source if you need a closer look
- Add class to extend VisCanvasEventAdapter to capture keys
/** Accesses Nintendo WiiMotion Plus via I2C (use the 3.3V port only!) **/  
// PINOUT:  
// Green:  Data  
// White:  Ground  
// Red:  3.3V  
// Yellow:  Clock  
public class WiiMotionPlus  
public WiiMotionPlus(Orc orc)  
/** Returns a three-dimensional array giving the raw angular rate  
* data for each of the three axes.  
**/  
public int[] readAxes()  
public static void main(String args[])
*/
* Orc wrapper for Infrared range sensors.
* The manufacturer (Sharp) suggests that a line can be fit to a
* function of the form 1/(d + Xd), where d is the distance and Xd is
* some calibration value. We use parameters Xm and Xb as the
* parameters of the line.
* 
* V = 1/(d + Xd) * Xm + Xb
* 
* Solving for d:
* d = (Xm/(V-Xb)) - Xd
**/

public class IRRangeFinder
{

/** Create an IRRangeFinder without any parameters-- useful only
 * when you provide your own parameters. Otherwise, use (e.g.)
 * makeGP2D12.
 **/
public IRRangeFinder(Orc orc, int port)

/** Get the range of the sensor in meters, returning 0 if the
 * sensor appears to be outside its working range.
 **/
public double[] getRangeAndUncertainty()

public double getRange()

/** Estimate the standard deviation of the range measurement. This
 * assumes a perfect distance versus voltage model, and that noise
 * only occurs on the voltage reading.
 * @param v The voltage
 **/
public double getRangeUncertainty(double v)

/** Configure distance and other parameters. **/
public void setParameters(double Xd, double Xm, double Xb,
 double voltageStdDev)

/** Create and return an IRRangeFinder configured with approximate
 * parameters for a Sharp ZY0A02. **/
public static IRRangeFinder makeZY0A02(Orc orc, int port)

/** Create and return an IRRangeFinder configured with approximate
 * parameters for a Sharp GP2D12.
 * @port [0,7]
 **/
public static IRRangeFinder makeGP2D12(Orc orc, int port)
Wall-follower
Algorithm: Straight Wall

- When we see the wall we want to follow && we have room to move forward:
  - Error $e = \text{measured distance} - \text{ideal distance}$
  - Basic controls: adjust angular rate to control lateral error
  - If too far from wall, turn toward wall “more,” if too close, “less”
Algorithm: Inside Corner

- When an obstacle appears in direction of travel:
  - Turn away and follow new wall
  - insideMotor.setPWM(0.5);
  - outsideMotor.setPWM(-0.5);
Algorithm: Outside Corner

- When the wall disappears (distance jumps discretely)
  - Make a wide turn and follow new wall
  - insideMotor.setPWM(0.2);
  - outsideMotor.setPWM(0.5);
Demo: ModelZViewer
Demo: Wall-follower
Midterm

- When: Wednesday during class
- Location: lab
- Covers any material in lab or lecture
  - Certainly represents what we’ve focused on
  - Study the lecture notes and the quiz (same type of structure)
- Consider studying with team, generate questions (from notes)
- Pressing questions?