LECTURE 2: High Level Sensor Strategy and Motors
Today's Lecture

• Announcements/reminders
• Overview of what's available to you.
• Examples of sensor strategy.
• Overview of Driving.
Check-Off for Assignment 2 by January 13, Tuesday.

- Check-off with any lab staff member
  - Know your TA/Organizer
  - Bring car!
  - We will be checking if your car can drive.
Other General Announcements

• Reminder: if the 6th floor doors to lab are locked, try entering through the 5th floors and going up the stairs in the lab.

• Tell us if you want credit now (also, make sure you are registered on websis).

• Check the website (and your email) religiously for updates.
Sensors and Motors

- See course notes for sensors available to you (dist. sensors, phototransistors, bump sensors and switches, gyro) and how to do the wiring for them
- Extra motors cost sensor points
- Keep in mind your 20 sensor point budget and the number of inputs you have on your happyboard.
Your Kit

The Brain: Happyboard

- You will soon receive your Happyboards.

- Once you receive your Happyboard, run through the test suite to ensure the Happyboard is in working order.

- Manual is not included, can get it from “Handouts” section of contestants site.
Happyboard Sensor Inputs
JoyOS

• Works with your Happyboard.
• Lets you focus on Strategy.
• Takes care of plenty of finicky details.
What JoyOS does for you

• Manages multiple threads
  – More details in lecture #3
• Communicates directly with HappyBoard hardware
  – FPGA, SPI, AVR ports, ADC, etc…
  – Provides simple API to talk w/ board
    • analog_read(), motor_set_vel(), etc…
• Libraries
We’ll be compiling with GCC
 GNU 99 Standard

- Means you can do this:
  • Declare new variables anywhere
  • For (int i=0; i<... 
  • Say you know the standard you’re using
What Are the Sensors Doing?

• They prevent you from dead reckoning
• What matters is where the robot is, not where it thinks it is
• Does the robot need to know exactly where it is?
• How can the robot know where it really is? Exactly?
The Bigger Picture

- Combining Sensors
  - Servo + distance sensor
  - Servo + Breakbeams
  - Phototransistors + distance sensor

- Do you even need sensors for a particular task?
  - Wall following / going straight
  - Making precise turns
Sense-Plan-Act Paradigm

Match code umain()
Wait... why am I pausing?

• Pause gives more CPU time to other threads
• Not necessary, but good practice
• Speed of CPU >> Mechanical response of robot
Example code structure

```c
int umain () { // start of match
    while (1) { // loop forever
        sense (); // read sensors
        plan (); // figure out what to do
        act (); // drive motors
        pause (10); // pause 10 milliseconds
    }
    return 0;
}
```
Methods to Drive Straight

- Shaft encoding
  - Relies on initial alignment
  - Relatively fast
  - Can be tricked by slipping
- Line following
  - Robust
  - Relatively slow
- Wall following
  - Requires continuous stretch of wall
  - Can be fast
- Gyroscope
  - Very fast
  - Instantaneously accurate
  - Can drift over time
Control Systems

- Robots are deaf, dumb, and blind
  - Only capable of following explicit instructions
- Control systems required to create desired motions
Open Loop Control

• Simply a set of sequential instructions
• Does not rely on external inputs
  – Dead reckoning / using timing
• Errors accumulate
Feedback Control

- Sense environment to correct errors
- Avoid dead reckoning
Driving Straight

- Drive mechanism
- Line following
- Shaft encoding
- Wall following
- Gyroscope
Drive Mechanisms

- Differential Drive
- Synchro Drive (servos)
- Rack-and-Pinion Drive (car)
- Independent Drive (gearboxes)
Line Following

- Use set of light sensors to look at color under robot
- Set of lines and contrasts on board
- Follow contrast

<table>
<thead>
<tr>
<th>Left</th>
<th>Middle</th>
<th>Right</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>●</td>
<td>RIGHT!</td>
</tr>
<tr>
<td></td>
<td>●</td>
<td></td>
<td>RIGHT!</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>right</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>right</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Left</th>
<th>Middle</th>
<th>Right</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>LEFT!</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>LEFT!</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td></td>
<td>left</td>
</tr>
</tbody>
</table>

Sensor Inputs
- ○ on
- ● off
Line Following

if prev_state == hard_right
then keep turning right
if prev_state == hard_left
keep turning left

if prev_state == right
turn left
if prev_state == left
turn right
Shaft Encoding

- Breakbeam sensor + pulley
- Count interruptions to find revolutions
- Useful for:
  - Driving straight
  - Turning
  - Moving a specific distance
  - Better than timing
    - Doesn’t rely on battery charge
Shaft Encoding

- Use last 4 sensor ports
- Both wheels may not turn at same speed
- Use revolutions for feedback
- Determine difference in speed and adjust
- Hint: place encoder high in gear train for greater resolution
Pseudo-Code

if (right encoder value - left encoder value) > 100 ticks
    slow down right wheel or speed up left wheel

if (left encoder value - right encoder value) > 100 ticks
    slow down left wheel or speed up right wheel
Wall Following

- Easy way to go straight
- Simple to implement
  - Bump sensors on side
  - Distance sensors

```java
while (...) {
    if (sensor hit)
        steer away from wall
    else
        steer towards wall
}
```
Let's use the gyro!

Match code `umain()`

- Read gyro
  - Drifting right? Turn left
  - Drifting left? Turn right
  - Set motor speeds

Pause
Simple program

```c
int umain () { // start of match
    int left_speed, right_speed;

    while (1) { // loop forever
        double deg = gyro_get_degrees();

        if (deg < 0) {
            left_speed = 140;
            right_speed = 160;
        } else {
            left_speed = 160;
            right_speed = 140;
        }

        motor_set_vel (0, left_speed);
        motor_set_vel (1, right_speed);

        pause (10);
    }
    return 0;
}
```
What’s Next (until Lecture 3)

• C Lecture (Today): Highly recommended
• C Workshop (Monday): also recommended
• Workshop 2 (Monday): Prepare your Motors (soldering, shrink wrap, etc)
• Checkoff (Tuesday): Can your 'bot drive?
• Lecture 3 (Wednesday): Threads, timeouts, debugging.
Purchasing Tools

• A number of staff tools necessary for electronics assembly will be available in lab
  – Everyone shares these tools, don’t hoard them!
• A limited number of tools is available for purchase to use outside of lab
  – Helping Hands $8
  – Wire Cutters $3
  – Soldering Iron $4
  – Needle Nose Pliers $4
  – Diagonal Cutters $5
• Tools Package $20
EOL

Questions?