6.270: Autonomous Robot Design Competition

• Assignment 2: General Comments
• More on sensors
• Servos
• RF receiver
• Robot control and state machines
• Threads
• Assignment 3 handed out

Lecture 3: Advanced Techniques
Delinquent Teams

• Assignment 2 teams not finished:
  – 11, 14, 15, 18, 22, 40, 55

• Assignment 3 handed out today, due tonight!!
Rules Clarifications

- Are prices measured in 1-u or 100-u quantities?
  - We’ll use 100-u quantities for pricing purposes
- Can we use rubber bands to add friction with game balls?
  - Yes
- Can we disable an opponent?
  - You cannot intentionally damage or flip
  - You can drive into the other robot or push them around
- Can we cut apart the baseplate and glue it back together?
  - Yes, but things glued together are not structural
Power Usage

- Your HandyBoard has 8 rechargeable 1.5 volt batteries built in
- They don’t last too long when driving actuators
- Next week, we’ll give you high capacity lead acid batteries from Hawker to power actuators
Some 6.002/8.02 Lovin’

- Voltage, Current, Resistance: \( V = I \cdot R \)
  - Resistance: ohms (\( \Omega \))
  - Current: amps (A)
  - Voltage: volts (V)
- Power: \( P = I \cdot V \)
Shorting the Batteries

- $v = 6$ V, $R = 0.02$ Ω
- $i = 300$ A
  - Household wiring rated 15 A
- $p = 1800$ W
  - Thirty 60-watt light bulbs
- Lesson: ensure battery leads are well-insulated!
Phototransistors

- It’s an art
- Need to figure out an effective way of reading the color off the board or object
  - Factors: glossiness, ambient lighting
  - It’s not really color; it’s grayscale
  - Contest night
  - Wear and tear of contest board
  - Can’t rely on just light provided by the world alone
Providing Your Own Light

- LEDs are polarized, and you must use a resistor
- Light dispersion
  - What is the best way to put LEDs
  - For color detection, look for reflection at an angle, not perpendicular to surface
Providing Your Own Light

• Turning LED on and off gives more info
• Use FET to turn multiple LEDs on and off using the digital output
• See handouts page for FET datasheet
Wiring Multiple LEDs

- Use a separate resistor for each LED

![Diagram of LED wiring with resistors and digital output]
Shielding

- Control the light made available to the sensor
- Help focus what the sensor is looking at
- Cardboard, heat shrink (black), electrical tape
- Some things aren’t as opaque as you think
- Calibration (and 60-second set-up time)
Distance Sensor

- Follow hookup instructions in the notes

**Sharp Distance Sensor**

- type: analog
- powered: yes, polarized: yes
- applications: measuring distance

![Diagram of distance sensor](image)
Distance Sensor on the HB

- Distance sensor provides variable voltage output
- Must disconnect internal pull-up resistor

\[ v = 5 \text{ V} \]

\[ \text{VDD} \]

\[ \text{IN} \rightarrow \text{to ADC} \]

\[ \text{GND} \]

\[ R = 47 \text{ k\Omega} \]
Disconnecting the 47kΩ Pull-Up

- Remove main HB PCB from the plastic case
Disconnecting the $47\,\text{k}\Omega$ Pull-Up

- Analog Inputs 2, 4, and 5 can be modified
Disconnecting the 47kΩ Pull-Up

- Cut traces (make sure you know where!)
Distance Sensor

- Range: 15-150 cm
  - 6-60 in
Distance Sensor

- You probably don’t need more than 3
- But if you’re really *that* needy, cut port 0 or 1

R = 47 kΩ

v = 5 V

GND

IN_0

VDD

to ADC
The Gyroscope

- Gives you a rate of rotation
- You can integrate to get a position
  - Usually accurate to within a couple of degrees over 60 seconds
- Example code given on contestant’s information page
Gyroscope Considerations

• Reducing drift and inaccuracy
  – Correct gyroscope data when you know what it must be
    • Backed against a wall
  – Use relative positioning
    • Make turns based on a change of 90 degrees, rather than turning to 270 absolute degrees
Gyroscope Usage

- You can have ONE gyroscope
  - 5 sensor points
  - Talk to us about how to hook it up
  - We will not replace broken gyroscopes
  - Use any analog sensor port
The RF Receiver

- Lets us give you information during the match:
  - Start/end of match
  - Vote tally
  - Position of robots
Using the RF receiver

- First thing in your code:
  - rf_team = YOURTEAMNUMBER;
  - rf_enable();

- This will disable the IC interface
  - To turn on your handyboard without enabling RF, hold START while turning on

- Plug your telephone cable into the HandyBoard and the RF receiver
Start/stop of match

- Use the function `start_machine()` (described in the course notes)
- Will automatically start your robot when the match begins, and stop it when the match ends
Voting Information

- rf_vote_red
- rf_vote_green
- rf_vote_winner
  - You need some way of determining a winner when there is a tie
  - All automagically updated
Position Information

- rf_x0, rf_y0
- rf_x1, rf_y1
  - Tell you the x and y coordinates of the two robots
  - No guarantees about which of the two robots you will be
    - Consistent during the match, but not across matches
  - Also automagically updated
Position Information

- rf_x0, rf_y0
- rf_x1, rf_y1
  - Approximately 8000 units per foot
  - Center of table is (0,0)
Position Information

• How do we determine the position information?
  – You’ll be required to put a colored swatch (which we provide) on top of your robot
  – We look at the table and find the swatches
  – More details later
The Bigger Picture
The Bigger Picture

- You have tools:
  - Sensors
  - Actuators
  - Mechanical chassis
  - Task-specific mechanical devices
  - Processor

- How to put it all together?
The Bigger Picture

• Combining Sensors
  – Servo + distance sensor
  – Servo + beacon
  – Beacon + distance sensor

• Do you even need sensors?
  – Wall following / going straight
  – Making precise turns
What Are the Sensors Doing?

- They prevent you from dead reckoning
- What matters is where the robot is, not where it thinks it is
- Provide information to make decisions
The AI: How to Code a Robot

- Programming language is easy; programming style is difficult, especially with a team (any 6.170 alums?)

- Some patterns have emerged in regards to having an effective coding style
  - Finite State Machines
  - Control
  - Coding Techniques
Finite State Machines

• What is a finite state machine (FSM)?
  – Defines what the robot should do at a given point in time
  – Each state has predefined outputs
  – Transitions to other states depend on inputs

• Why?
  – Effective way of thinking about your strategy
  – Define what to do for any combination of inputs
Implementing a State Machine

- Each action is a state
  - Moving forward
  - Turning
- Actuators are outputs of the FSM
- Sensor inputs determine next state
Example FSM

Orient Robot

Move Straight

Ball not detected

Release ball

Collect Ball

Detect Ball not scored

180-degree turn

Detect opposing robot

Wall Follow Forward

Detect scoring area
Coding an FSM

• While loops
  – Continue an action until input is received

• Multithreading
  – Processes that determine the inputs
  – Processes that determine outputs and state transitions

• Don’t do it the 6.111 PAL 20V10 way
  – Don’t need a variable to keep track of what state you’re in
  – Instead think conceptually; think before you code
FSM Issues

• Inputs
  – Check only those that matter at that state
  – Determine what is important

• Storing State
  – Make your robot smarter
    • Use the state as well as the inputs to determine action
    • Store last actions in state variables
  – Helpful if robot gets disoriented
Driving Straight

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

<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>↓ left</td>
</tr>
<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>● left</td>
</tr>
<tr>
<td>●</td>
<td>●</td>
<td>●</td>
<td>● left</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>n/a</td>
</tr>
<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>● left</td>
</tr>
</tbody>
</table>

Sensor Inputs
○ on ● off
Drive Mechanisms

- Differential Drive
- Synchro Drive (servos)
- Rack-and-Pinion Drive (car)
- Independent Drive (gearboxes; Assignment 2)
**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>n/a</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
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
Shaft Encoding

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

- Works better on some ports:
  - Ports 7 and 8 have hardware counters (faster, more accurate)
  - Others use software counters
  - If you need more than 2, try using ports 2-6
- 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
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

while (...) {
  if (sensor hit)
    steer away from wall
  else
    steer towards wall
}
Driving Straight—Advantages and Disadvantages

- 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
Code Implementation

• Start on paper
• Use functions and comments
  – Code is then legible for everyone on your team and for us (impounding)
Programming Methodology

• Top-down programming
  – Good for initial design
  – Overall view without details

• Bottom-up programming
  – Good for code creation
  – Allows individual testing of functions
Programming Methodology

• Figure out the actions you want to take
• Figure out the functions you need
• Implement
• Test
• Integrate into other code
• Repeat
Testing and Debugging

• Most important part of the design
• Significant testing is necessary to do well
  – Things will break
  – Things will happen that you don’t expect
  – Try to see these things in advance
• Test and debug incrementally
Hints

• Test sensors before mounting
• Test small pieces of code before combining into larger procedures
• Use the LCD screen
• Remember mechanical reliability
Error Detection

• Your robot **will** mess up
• How can it find out what’s wrong?
• **Timeouts** are key
Timeouts

- Detect when robot is stuck in a state
  - Probably waiting for input – bump into wall, light reading
- Force out of stuck state
  - Error correcting routines
Error Correction

- Try again, harder
- Back up, try again
- Wiggle around
- Guess what it should try next
- Skip to next part of routine
- Line following: what to do about the n/a states
  - In this case, using an FSM may help you figure out what to do
Quick Note on Threads
Quick Note on Threads

• What is a Thread?
  – Separate task running at the same time
  – Allows you to multi-task
    • Motors run *and* watch if a sensor is pressed

• How does one processor run two threads?
  • Executes a process certain number of ticks (ms)
  • Processor switches from one thread to another
The Methods for Threading

- **int start_process(function_call(), [TICKS], [STACK_SIZE]);**
  - Default run is 5 ticks, or 5 ms
  - Stack size is by default 256 bytes
  - Returns process ID (pid) of the new process
  - You shouldn’t need to pass `ticks` or `stack_size`

- **int kill_process(int pid)**
  - Returns 0 (process was destroyed), 1 (process not found)
Interacting in IC

• \texttt{kill\_all}
  – Kill all currently running processes

• \texttt{ps}
  – Prints out list of process status
  – Provides:
    • Process ID
    • Status code
    • Program counter
    • Stack pointer
    • Stack pointer origin
    • Number of ticks
    • Name of function that is currently executing

• Refer to Handy Board manual for more information
Example

```cpp
main() {
    while (true) {
        go forward
        wait until sensor pressed
        go backward
        wait until sensor pressed
    }
}
```

```cpp
main() {
    while (true) {
        if (vote is tied) {
            play tone 1
            while (red is winning) {
                play tone 2
            }
            while (green is winning) {
                play tone 3
            }
        }
    }
}
```
Example

```plaintext
move() {
    while (true) {
        go forward
        wait until sensor pressed
        go backward
        wait until sensor pressed
    }
}
```

```plaintext
watch_vote() {
    while (true) {
        while (vote is tied)
            play tone 1
        while (red is winning)
            play tone 2
        while (green is winning)
            play tone 3
    }
}
```
Example

```c
void move() { ... }
void watch_vote() { ... }

void main() {
    int move_pid;
    int watch_votePid;
    move_pid = start_process(move());
    watch_vote_pid = start_process(watch_vote());
    sleep(60);
    kill_process(watch_vote_pid);
    kill_process(move_pid);
}
```
Why Was the Example Easy?

- Threads are independent of each other
- Do not share any common variables, or common information
- Did not attempt to communicate or change each other’s state
How Threads Can Communicate

- Communicate through global variables
- Variables declared above and outside of all functions are global variables (like C)
- One thread can use the global variable that another thread is changing
For the Contest

- You will be using threads, even if you don’t know it
  - We provide start code that makes sure that you start and stop at the right times: `start_machine()`

- See Appendix A for more details
Thread Tips

- Outside of `start_machine()`, you most likely won’t need threads
  - Work around threads with control statements: `for`, `while`, `if...then...else`, `return`, `break`
- Don’t use `reset_system_timer()`
  - Our start system code depends on the timer
- Don’t `sleep()`; use `while` loops
  ```
  sleep(3.0); float start_time = seconds();
  
  while (seconds() - start_time < 3.0) {
    /* check for anything (like sensor inputs) */
    if (you_really_need_to_leave_the_while_loop)
      break;
  }
  ```
Your Winning Strategy

• Sufficient sensors and AI to determine location of robot
• Be able to react to potential problems that the robot might face
• Be aware of your limitations
  – Amount of LEGO
  – Power and speed of the motors
  – Robot size
  – Time of the round (60 seconds)
  – How long until Tuesday, January 25, 5:00 pm
Your Winning Strategy

• Reliability and robustness are the keys
  – 90% reliability means 43% chance of not failing in 8 rounds
  – KISS
  – Leave a lot of time for testing and debugging

• Impossible to counter every opposing strategy, so don’t try
Assignment 3

- Due Friday night (TONIGHT!) at 11:45 pm
- One task to complete:
  1. Romeo and Juliet
- Pick up assignment after lecture
Assignment 4

• Due Tuesday night (January 11) at 11:45 pm

• Two tasks to complete:
  1. Discuss with your Organizer/TA pair your strategy
  2. Submit a one-page write up of intentions
What’s Next

• No workshops today
• Monday, January 10, and Tuesday, January 11
  – Workshop 5 – Servos, Sensors, and Shaft Encoders
    • Using analog sensors
    • Servo – the other motor
    • Shaft encoding with breakbeam sensor
    • Gyroscopes
  – Workshop 6 – Advanced LEGO
    • Using the unique pieces
    • Interesting gadgets
  – Workshop 7 – Code & Sensors II: Advanced Techniques
    • Open vs. closed loop control
    • Line following
• Don’t forget to sign up for workshops in lab!
Good LUCK!