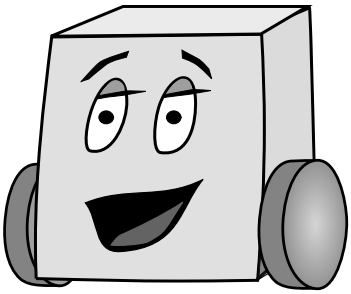


E11: Autonomous Vehicles

Fall 2010

Harris & Lape with Keeter & Ong

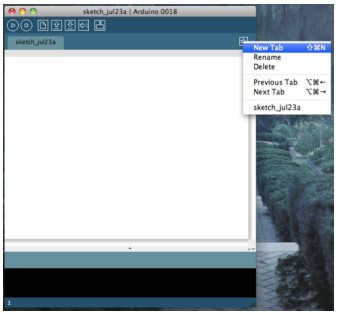
Lab 6: Motors and Sensors

Introduction

By this point, you should have an assembled robot and Mudduino to power it. Let's get things moving!

Getting started with motors

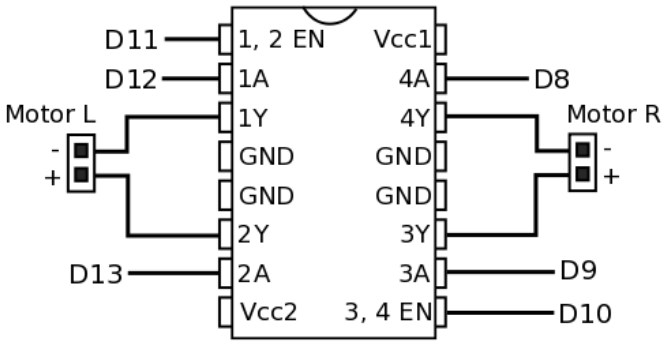
It's important to give your robot a way of moving around. Since driving will show up in many different assignments, we recommend making a separate file to contain motor control code, so that you can copy the file into each new sketch without much trouble. To make a separate file, use the New Tab button near the upper right corner of the Arduino IDE.



Name this new file **motor_xx**, where xx are your initials, and save the entire sketch with the name **lab6_xx**. In **motor_xx**, you will write functions that allow the robot to move around. These functions will activate the H-bridge to bring power to your motors; once they are written you'll be able to ignore the H-bridge's inner workings.

Motor control pins

The following schematic shows the Mudduino motor control pins:



It is a good idea to declare these pins using **#define** statements at the top of your code (e.g. **#define LPLUS 13**).

You will discover that the Arduino bootloader toggles some pins connected to the motors every time you download code or press the reset button. The right motor is likely to spin for a while, and there is nothing your program can do to prevent this. Hold on to your robot so it doesn't scoot away!

Basic driving

In your **motor_xx** file, write the following set of functions:

- **void initMotors()**: Declare relevant pins as outputs, make sure that robot is stopped
- **void halt()**: Stop all motors
- **void forward()**: Drive forward indefinitely
- **void turnL()**: Turn left indefinitely
- **void turnR()**: Turn right indefinitely
- **void backward()**: Drive backwards indefinitely

Note that indefinitely does not require an infinite loop in your program. If you write the appropriate values to the pins, the robot will keep doing the same thing until the values on those pins change.

Try out each function after you've written it, to make sure it works. The lab should be filled with the joyous noises of robots running into walls and trying to jump off tables!

Timed driving

One common task is to drive for a certain amount of time, then move on to something else. Using the power of overloading, you can write two distinct functions: **void forward()**, and **void forward(int time)**. Depending on whether you invoke **forward()** or **forward(SOME NUMBER)**, the correct function will be called. Write alternate versions of the earlier functions that take a single argument **time**. These functions should do the same task as before, but only for the specified amount of time (in units of milliseconds). They should then stop the robot.

Hint: These functions can be written in three lines each if you use what you've already written!

Driving with variable power

Some times, you might want your robot to slow down. To make your robot go slower, we will use pulse-width modulation on the enable lines of the H-bridge. Pulse-width modulation is a technique to imitate an analog value with a digital signal. Instead of being consistantly high, a PWM signal is high only some fraction of the time. A PWM signal is used with the command **analogWrite(pin, level)**. Level can be anywhere from 0 (always off) to 255 (always on). Power levels below about 50 may not provide enough torque to move your bot.

Add a global variable named **int powerLevel**, with an initial value of 255. Now, modify your initial set of driving functions to use **analogWrite** instead of **digitalWrite** when enabling the H-bridge. Finally, add another function named **setPowerLevel(int pl)** that redefines the global variable.

Now, to drive slowly, you can use:

```
setPowerLevel(128); // set half power
forward(10);        // drive forwards for 10 seconds
```

Congratulations! This completes the standard set of driving functions. You will be using this file often, so now would be a good time to test it out. Try turning left and right slowly, as well as reversing. Which leads us to…

Square driving challenge

Right now, your robot is operating with open-loop control. This means it is operating with no feedback from the sensors. Your robot is essentially blind, following a pre-set list of instructions. This type of control is simple, but limits the precision of the robot's maneuvers.

There is a square drawn in electrical tape on the floor of the lab. Try to perfect your turn timing to make your robot drive around the square as quickly as possible. This is a good time to experiment with timed turns, to get the best 90 degree turn you can. If your robot doesn't drive quite straight, you may need a different turn. You should aim to get the robot to park as close to its starting location as possible. Don't spend too much time on this problem, though; save some time to experiment with your sensors, and come back to this part if you have time at the end.

Getting started with sensors

The sensors used in this class are primarily analog: instead of a digital on/off value, they give a voltage that is somehow related to the intensity of the measurement. The relationship could be linear, exponential, or even inverted. That's why it's important to get to know your sensors. When you're first using a sensor, write a simple program that prints the sensor's reading to the serial port. Then, expose the sensor to a range of stimuli, so that you know what its response looks like across the board.

In the last lab, you soldered together your sensors but did not yet test them. Now it is time to find if you assembled them correctly. Write a short Arduino program to read the reflectance sensor on A5 and the phototransistor on A4 and to print the results to the console. The reflectance sensor should give significantly different results when placed over different color surfaces. The phototransistor should give significantly different results when pointed at a light source. Look at the results and convince yourself that the sensors are wired correctly. If you have trouble, check the voltages with a voltmeter and look for shorts between pins.

Phototransistor

The phototransistor on the front of the robot detects light. When it sees a bright light, you should detect a low value on the corresponding analog pin.

Write code that causes the robot to turn in a circle until it sees a bright light, then drive forward. Put your code in a new sketch with a name of your choosing. Use the Sketch -> Add File command to add your motor code from the previous part.

The instructor will set up a large lamp on the floor. If your code works, the robot will attack the lamp.

Photo from Jameco Electronics

Reflectance sensor

The IR reflectance sensor can tell you whether the ground underneath it is dark or light. It has two halves: one side emits infrared light, and the other detects the IR light bouncing back. The challenge for this sensor is simply to stop on a dark line. Drive forwards until the robot sees a dark line, then stop immediately.

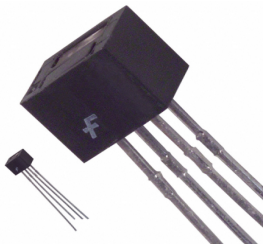


Photo from DigiKey

This technique can also be used to prevent the robot from jumping off tables! Since the floor is far away, very little light is reflected back to the sensor. Write a program to constantly polling the reflectance sensor while the robot drives forward. If the reading ever drops too low, the robot should halt immediately and play a song on the speaker to indicate what happened. Test this code out and see if your robot can stop fast enough to avoid taking the plunge off the table!

Congratulations!

You should now be familiar with the core set of sensors. The possibilities are endless!