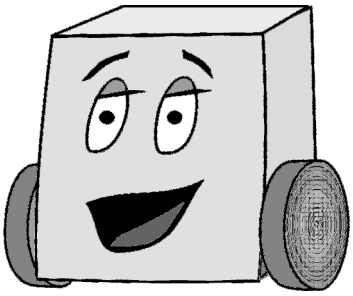


E11: Autonomous Vehicles

Lab 5: Motors and Sensors

By this point, you should have an assembled robot and Mudduino to power it. Let's get things moving! In this lab, you will write code to test your motors and sensors. This lab has three parts:

1. Controlling the DC motors (connected to the wheels)

2. Controlling the servo motor (connected to the distance sensor)

3. Testing the sensors (phototransistor, reflectance sensor, and distance sensor)

Create a sketch called lab5_xx (where xx are your initials). You will have three additional tabs in the sketch called: motors_xx.ino, servotest.ino, and sensors.ino, corresponding to each of the three sections above.

1. Controlling the DC 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. In your lab5_xx sketch, add a separate file using the New Tab button near the upper right corner of the Arduino IDE.

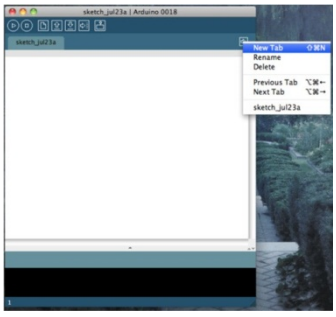


Figure 1. Adding a file to the sketch using New Tab

Name this new file **motor_xx.ino**, where xx are your initials. 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

Figure 2 shows the Mudduino motor control pins coming from the H-Bridge.

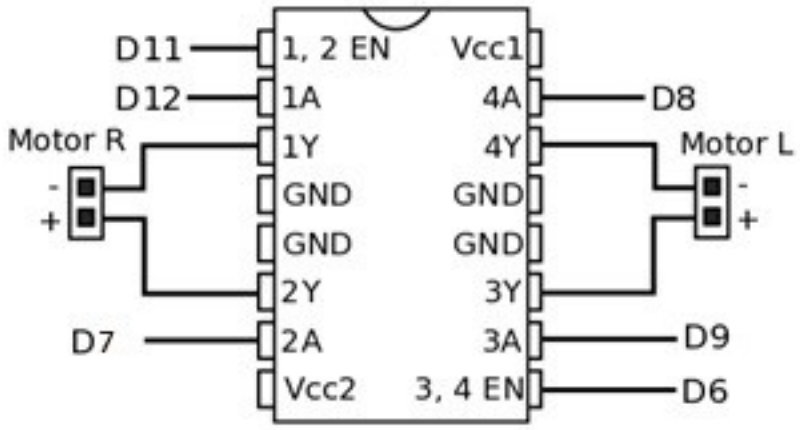


Figure 2. H-bridge connections

It is a good idea to declare these pins using **#define** statements at the top of your code (e.g. **#define LPLUS 9**). Be sure to define LPLUS, LMINUS, RPLUS, RMINUS, LEN, and REN. A copy of the Mudduino pinout is included on the last page of the lab for your convenience.

The motor power can come from either the battery or the 5-volt regulator, as selected by the middle power switch. The battery provides at least 7.2 V, so it will spin your motors faster and is recommended. However, if your battery is not connected, you can still test the motors off USB power.

Basic Driving

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

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

If a motor spins in the opposite direction from what you intend in your code, simply switch the wires to the MR+/- or ML+/- header pins.

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!

Warning: turn the motors off (switch in middle position) before uploading a sketch to the Mudduino.

2. Controlling the Servo Motor

The servo motor rotates from 0 to 180 degrees. 0 degrees is to the right; 90 is straight forward, and 180 is to the left. The servo is controlled by a pulse-width modulated (PWM) signal from the Mudduino. The Mudduino has an optional library to produce such PWM signals.

The code in Figure 3 demonstrates how to use the servo. It `#includes` the `servo.h` library with the servo functions. It also declares `servo` as a global variable to record information about the state of the servo. It then uses `servo.attach(10)` to tell the Mudduino that the servo is connected to pin D10 and `servo.write(90)` to drive the servo motor to 90 degrees (straight forward).

```
// servotest.ino
// David_Harris@hmc.edu 1 October 2011

#include <Servo.h>

// pins
#define SERVOPIN 10

// Global variable for the servo information
Servo servo;

void testServo()
{
     initServo();
     servo.write(90); delay(2000);
}

void initServo()
{
     pinMode(SERVOPIN, OUTPUT);
     servo.attach(SERVOPIN);
}
```

Figure 3. Example servo code

Do not turn your servo by hand, especially when power is applied; you may damage it. Before you run the code, visually inspect that the servo will be able to turn to the left and right without bumping against wires.

Each time you turn on the robot, the servo may twitch and move to a new position before your `servo.write` command is executed. However, `servo.write` should point it in the desired direction. Again, avoid turning the servo by hand.

Test your servo with the sample code:Add another tab called servotest.ino and copy the code from Figure 3 into that tab. Then call testServo() in the lab5_xx.ino setup() function. Comment out tests from previous parts of the lab as needed. Be sure the distance sensor is pointing straight forward when at 90 degrees. If necessary, loosen the pan-head screw and adjust the bracket to point forward.

Modify the testServo() function to test your servo by positioning it at 0^{o}, 45^{o}, 90^{o}, 135^{o}, and 180^{o}. Give a 2 second pause between each position.

If the servo doesn't turn as expected and the voltage regulator on the Mudduino board gets very hot, it is possible that you have damaged the servo and need to replace it.

3. Testing the Sensors

In this section, you will test and use each of the three sensors on the Mudduino: the phototransistor, the reflectance sensor, and the distance sensor. 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 measurement. The relationship could be linear, exponential, or even inverted. That's why it's important to get to know your sensors. In the last lab, you soldered together your sensors but did not yet test them. Now it is time to find out if you assembled them correctly.

To start,create a new tab in your lab5_xx sketch called sensors.ino. In that tab, write a simple function that:

(1) sets up all three sensor pins as inputs,
(2) reads from the sensor pins twice per second, and
(3) prints the sensor readings to the serial port.

Then, expose each sensor to a range of stimuli, so that you know what its response looks like across that range. Call your new function from the loop() function in lab5_xx.ino. Comment out tests from previous parts of the lab as needed. To get you started, the following lines of code set up the distance sensor for reading and read the value.

```
#define DISTSENSOR 14

void testSensors() {
     pinMode(DISTSENSOR, INPUT);
     dist = analogRead(DISTSENSOR-14);
}
```

The distance sensor should vary as you move your hand in front of it. The reflectance sensor should give significantly different results when placed over different color surfaces such as the lab bench and a piece of white paper. The phototransistor should give significantly different results when pointed at a light source. Look at the results and convince yourself that the sensors are operating correctly. If you find trouble, check the voltages with a voltmeter and look for shorts between pins.

Phototransistor

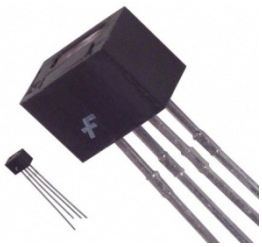
The phototransistor on the front of the robot detects light. When it sees a bright light, you should detect a low value on the corresponding analog pin. Write a new function in sensors.ino for testing the phototransistor. This function should cause the robot to turn in a circle until it sees a bright light, at which point it drives forward. Call your function in setup() to test it.

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

Phototransistor (image 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. Now write a new function in sensors.ino that simply stops on a line of black tape. The robot should drive forward until it sees a dark line, then stop immediately. Call this function from setup() in lab5_xx.ino to test it.



Reflectance Sensor (image 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 poll 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!

Distance Sensor

Calibrate the distance sensor by plotting the reading vs. the distance from your robot's front bumper to a wall. A yardstick is helpful. Note that the sensor has some ambiguity when measuring very short distances. Make sure that wires or other nearby objects aren't contaminating your data.

Distance Sensor (image from Pololu)

Optional Topics

If you have time remaining, you can play with any of the following three topics:

- Timed driving
- Driving with variable power
- Driving in a square

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 f**orward(SOME NUMBER)**, the correct function will be called. In **motors_xx.ino**, write additional 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

Sometimes, 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 (PWM) is a technique to imitate an analog value with a digital signal. Instead of being consistently 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), and the pin number is the digital pin number. Power levels below about 50 may not provide enough torque to move your bot.

In your motors_xx.ino file, 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** for all motor drive functions. Finally, add another function named **setPowerLevel(int pl)** that sets the global variable powerLevel to pl (the input argument).

Now, to drive slowly, you can use:

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

Hint: It may be helpful to create a function that takes signed power levels (positive or negative) and moves forward, backward or turns left, right depending on those values.

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 dark 4" square pattern in the hallway tiles just outside 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. This is hard, so don't worry if you don't get perfect results. Remember this if you try to use open-loop control in the final project.

Mudduino Pinout

Digital Pin #	Analog Pin #	Notes
0		Serial TXD – don't use
1		Serial RXI – don't use
2		Header D2
3		Team (0 = green / 1 = white) read only
4		Header D4, Buzzer
5		Header D5 / green LED / programming indicator
6		Left Motor Enable
7		Right Motor +
8		Left Motor -
9		Left Motor +
10		Header D10 / Servo (use servo.write)
11		Right Motor Enable
12		Right Motor -
13		Header D13 / red LED
14	0	Distance Sensor
15	1	Header A1
16	2	Header A2
17	3	Header A3
18	4	Header A4, Reflectance Sensor
19	5	Header A5, Phototransistor