# CURIE Academy, Summer 2014
# Lab 2: Computer Engineering – Software Perspective

## Sign-Off Sheet

NAME: ______________________________

NAME: ______________________________

DATE: ______________________________

| Sign-Off Milestone | | TA Initials |
|---|---|---|
| Part 1.A | Test Simple Addition Program | |
| Part 1.B | Examine Corresponding Machine Instructions | |
| Part 2.A | Experiment with LEDs, Switches, and Potentiometer | |
| Part 2.B | Experiment with Grayscale Analog Sensor | |
| Part 2.C | Experiment with Drive Motors | |
| Part 3.A | Develop Move-and-Stop Behavior | |
| Part 3.B | Develop Wander Behavior | |
| Part 3.C | Develop Wander-to-Target Behavior | |
| Optional Extension: | | |
| Optional Extension: | | |
| Optional Extension: | | |

# CURIE Academy, Summer 2014
# Lab Handout 2: Computer Engineering – Software Perspective

Prof. Christopher Batten
School of Electrical and Computer Engineering
Cornell University

In this lab assignment, you and your partner will explore the field of computer engineering from the software perspective using an Arduino-based robotics platform. Feel free to consult the lab notes as you work through the lab. The lab includes three required parts and an set of optional extensions. In Part 1, you will compile and analyze the machine instructions for a simple Arduino program. In Part 2, you will experiment with the sensors and actuators provided as part of the mobile robotics platform. Parts 1 and 2 are meant to gently introduce you to computer engineering from the software perspective, and thus much of the implementation is provided for you. In Part 3, you will gradually develop a more complicated wandering behavior such that the robot is able to explore its environment and find the target. Part 3 is more open-ended and will require critical thinking as you leverage what you learned in Parts 1 and 2. The end of the lab handout includes a list of optional extensions for those groups that are able to quickly finish the first parts. For each part and various subparts you will need to have a teaching assistant observe the desired outcome and initial the appropriate line on the sign-off sheet. **Turn in your sign-off sheet at the end of the lab session.**

Before beginning, take a look at the materials on the lab bench which you will be using to complete the lab:

- Arduino-based robotics platform
- Wooden testing block (some groups may need to share)
- Workstation with black USB cable
- Arduino cheat sheet

## 1. Compiling Applications to Machine Instructions

In this part, we will write our very first Arduino program and examine the corresponding machine instructions. Our initial program will add three numbers together and then display the output both on the host computer and by blinking an LED on the prototyping board. Revisit the lab notes for more information about the various parts of an Arduino program.

Complete the following steps before continuing:

- Ensure that the 9V battery barrel connector is plugged into the Arduino board so that the circuit boards are powered. Ensure the drive motor power switch is in the OFF position.
- Plug the black USB cable into the Arduino board.
- If the Arduino software development environment is not already running, start the development environment by clicking on the Arduino shortcut located on the desktop.
- Ensure that the Arduino software development environment can connect to the Arduino board by using the menu item: *Tools* $\rightarrow$ *Serial Port* $\rightarrow$ *COM4*. Sometimes you might need to select COM3 or COM5.

### 1.A Test Simple Addition Program

Enter the program in Figure 1. The `digitalWrite` routine will write a digital output pin with either a logic high or a logic low. The `delay` routine will essentially wait for the given time specified in milliseconds. Notice that we are using a `for` loop on Lines 28–34 to blink the LED several times; the number of times we blink the LED is equal to the sum of the three input numbers.

To save you some time, we have provided a template which you can use for all of the programs in this lab. To access the template, select the menu item: *File* → *Sketchbook* → *curie_lab2_template*. The template includes all of the global names for pin assignments and the appropriate code for the `setup` routine for all parts in the lab. You can continue to reuse a fresh version of this template for eaech part of the lab.

Compile and upload your program to the robot. Recall that you use the *checkmark* icon to compile your program into machine instructions and the *right-arrow* icon to upload these machine instructions to the Arduino board. Verify that the sum of the three numbers is correctly displayed by the number of times the LED blinks. Experiment with different values for the three inputs.

We can also use the *serial monitor* to display the sum. The serial monitor allows the robot to send strings for display on the workstation through the USB cable. Once the code has been uploaded, to see the output from the serial monitor you need to select the menu item: *Tools* → *Serial Monitor*. You should now see a new window which displays the sum every few seconds.

*Sign-Off Milestone:* Once you have experimented with your program, demonstrate for an instructor that the program can successfully add at least two different sets of inputs.

*Critical Thinking Questions:* What would happen to the LED if we replaced Line 23 with `sum = sum + input_a`? Explain why this would happen.

### 1.B Examine Corresponding Machine Instructions

In this subpart, we wish to examine the machine instructions that correspond to the `loop` routine in the high-level program expressed in Figure 1. To see these machine instructions, first compile and upload the simple addition program and then double click on the *disassemble* icon on the desktop. A new text file should appear on the desktop named *machine-instructions*. Double click on this text file and search through the file until you find the following line:

```
00000118 <loop>:
```

The lines after `<loop>` show the machine instructions which correspond to the code in the `loop` routine, and the first few instructions correspond to Line 23 in Figure 1. Take a few minutes to understand how the load instructions (`lds`), store instructions (`sts`), and add instructions (`add`) implement the arithmetic on Line 23.

*Sign-Off Milestone:* Once you have generated the corresponding machine instructions, show the instructions for the `loop` routine to an instructor.

*Critical Thinking Questions:* The `add` instruction might very likely be implemented with a ripple-carry adder similar to what you built in Lab 1. How many full adders do we need if we can guarantee that the inputs to the instruction are non-negative and less than or equal to 255?

## 2. Robotic Sensors and Actuators

In this part, we will explore various robotic sensors and actuators as a way to gradually learn about the mobile robotic platform and to create a foundation for developing the mobile robot control ap-

```
// Global constants for pin assignments

int pin_led1 = 4;

// The setup routine runs once when you press reset

void setup() {
  Serial.begin(9600);                // Setup the serial terminal
  pinMode( pin_led1, OUTPUT );       // Configure pin_led1 as digital output
}

// Global variables

short input_a = 2;
short input_b = 2;
short sum     = 0;

// The loop routine runs over and over again

void loop() {

  // Do the addition
  sum = input_a + input_b;

  // Print the result to the serial monitor
  Serial.println( sum );

  // Blink LED sum times
  for ( int i = 0; i < sum; i++ ) {
    digitalWrite( pin_led1, HIGH );  // Turn on the LED
    delay(500);                      // Wait 0.5 seconds
    digitalWrite( pin_led1, LOW );   // Turn off the LED
    delay(500);                      // Wait 0.5 seconds
  }

  // Wait four seconds
  delay(4000);
}
```

**Figure 1: Simple Addition Program**

plication in the next part. There are three subparts; after completing each subpart you will need to have a teaching assistant look over your solution and initial the sign-off sheet.

### 2.A Experiment with LEDs, Switches, and Potentiometer

In this subpart, we will write and execute several small Arduino programs which interact with the LEDs, mechanical bump switches, and the potentiometer.

We will now write the simple Arduino program shown in Figure 2 which blinks one of the LEDs on the prototyping board. You should type this program into the Arduino software development environment. Instead of typing the entire program from scratch, you can save the code from the previous part using the menu item: *File* → *Save* and then create a new copy for this part by using the menu item: *File* → *Save As ....* You can then delete some of the old code and replace it with the new code. Ensure that the code within the loop routine exactly matches what is shown in Figure 2. Upload your program to the Arduino board and verify that the LED blinks appropriately. Feel free to experiment with different delays to vary the blink rate.

```
// Global constants for pin assignments

int pin_led1          = 4;
int pin_led2          = 5;
int pin_bump_right    = 6;
int pin_potentiometer = A5;

// The setup routine runs once when you press reset

void setup() {
  pinMode( pin_led1, OUTPUT );      // Configure pin_led1 as digital output
  pinMode( pin_led2, OUTPUT );      // Configure pin_led1 as digital output
  pinMode( pin_bump_right, INPUT ); // Configure pin_bump_right as digital input
}

// The loop routine runs over and over again

void loop() {
  digitalWrite( pin_led1, HIGH );   // Turn on the LED
  delay(1000);                      // Wait one second
  digitalWrite( pin_led1, LOW );    // Turn off the LED
  delay(1000);                      // Wait one second
}
```

**Figure 2: Blinking LED Code Example**

```
// Global constants for pin assignments

int pin_led1          = 4;
int pin_led2          = 5;
int pin_bump_right    = 6;
int pin_potentiometer = A5;

// The setup routine runs once when you press reset

void setup() {
  pinMode( pin_led1, OUTPUT );      // Configure pin_led1 as digital output
  pinMode( pin_led2, OUTPUT );      // Configure pin_led1 as digital output
  pinMode( pin_bump_right, INPUT ); // Configure pin_bump_right as digital input
}

// The loop routine runs over and over again

void loop() {

  // Blink LED1
  digitalWrite( pin_led1, HIGH );
  delay(1000);
  digitalWrite( pin_led1, LOW );
  delay(1000);

  // Control LED2 with the mechanical bump switch
  int switch_pressed = digitalRead( pin_bump_right );
  if ( switch_pressed ) {
    digitalWrite( pin_led2, HIGH );
  } else {
    digitalWrite( pin_led2, LOW );
  }

}
```

**Figure 3: Blinking LED and Switch-Controlled LED Code Example**

We now extend this program so that if one of the mechanical bump switch is pressed, then the second LED on the prototyping board should turn on. Figure 3 shows the full code. Modify your program to include this extra functionality. Upload your program to the Arduino board and verify that one LED blinks and the other LED turns on when the switch is pressed.

We now extend the program to enable the potentiometer on the prototyping board to control the blink rate. A potentiometer is a variable resistor; turning the blue knob on the prototyping board changes the resistance of the potentiometer. The Arduino can read this resistance as an analog value by using an analog input pin. A close examination of the prototyping board reveals that the potentiometer is wired to the Arduino analog input pin 5. Analog inputs will have a value between 0 and 1024. So you can read the potentiometer on the prototyping board by adding this line to your `loop` routine:

```
int potentiometer_value = analogRead( pin_potentiometer );
```

Write the final modification to enable the potentiometer on the prototyping board to control the blink rate. You will need to determine how to best modify your code on your own.

*Sign-Off Milestone:* Once you have one LED's blink rate controlled by the potentiometer and the other LED controlled by the switch, have a teaching assistant verify that things are working correctly.

*Critical Thinking Questions:* Explain why pressing the switch quickly sometimes does not turn on the LED, and why the LED stays lit for a short duration after you release the switch. Similarly, explain why adjusting the potentiometer sometimes requires a short duration before impacting the blink rate.

### 2.B Experiment with Grayscale Analog Sensor

In this subpart, you will characterize the response of the analog grayscale sensor on the bottom of the robot. Notice how the sensor has a white LED which generates light and a photodetector which senses light; the sensor will report an analog value between 0 and 1024 which indicates how much light from the white LED is reflected back to the photodetector. Since the light-colored plywood will reflect more light than the black target, we can use this sensor to determine when the robot has found the target.

We can use the serial monitor to see the current value of the analog grayscale sensor. Enter and upload the code shown in Figure 4. Instead of typing the entire program from scratch, you can save the code from the previous subpart using the menu item: *File → Save* and then create a new copy for this subpart by using the menu item: *File → Save As ....* You can then delete some of the old code and replace it with the new code. Ensure that the code within the loop routine exactly matches what is shown in Figure 4.

Once the code has been uploaded, to see the output from the serial monitor you need to select the menu item: *Tools → Serial Monitor*. You should now see a new window which displays the analog grayscale sensor value once per second. Record the sensor value when the robot is over the light-colored plywood and when the sensor value is over the black target. Later in the lab, you will want to choose a value somewhere in between these two extremes when determining when the robot is over the target.

*Sign-Off Milestone:* Once you have the analog grayscale sensor values being displayed on the serial monitor, have a teaching assistant verify that things are working correctly.

*Critical Thinking Questions:* What happens when the analog grayscale sensor is positioned precisely at the edge of the target? How do your readings compare to the readings collected by other groups around you? Why do you think all robots do not report the same analog grayscale sensor values

```
// Global constants for pin assignments

int pin_grayscale_sensor = A4;

// The setup routine runs once when you press reset

void setup() {
  Serial.begin(9600);  // Initialize serial monitor
}

// The loop routine runs over and over again

void loop() {

  // Read the analog grayscale sensor
  int sensor_value = analogRead( pin_grayscale_sensor );

  // Print the value to the serial monitor
  Serial.print( "sensor = " );      // print does not include a newline
  Serial.println( sensor_value );   // println does include a newline

  // Wait one second
  delay(1000);

}
```

**Figure 4: Analog Grayscale Sensor and Serial Monitor Example**

when positioned over similar materials? What do you think would happen if the target was made out of a very glossy black material?

### 2.C Experiment with Drive Motors

In this subpart, we will begin to experiment with the drive motors. Our test program will attempt to move the robot in a square with two foot sides. Figure 5 shows the code. Two pins are used to control each drive motor. One pin controls the direction that the motor spins (e.g., `pin_motor_left_dir`) while the other pin controls the speed of that motor (e.g., `pin_motor_left_speed`). The motor direction output pin can either be set to `LOW` to spin the motor forward or set to `HIGH` to spin the motor backward. The motor speed pin can be set to a value between 0 and 255 with higher numbers indicating faster motor speed. Moderate motor speeds are between 75 and 125. Keep in mind that the speed of the motor also depends on the voltage from the five AA batteries; as these batteries drain you may need to increase the motor speed setting to maintain the same absolute robot velocity. To simplify your design, we recommend always setting the direction and then the speed together in your code whenever you want to change the movement of the robot.

Enter the code in Figure 5. Instead of typing the entire program from scratch, you can save the code from the previous subpart using the menu item: *File* → *Save* and then create a new copy for this subpart by using the menu item: *File* → *Save As ....* You can then delete some of the old code and replace it with the new code. Ensure that the code within the loop routine exactly matches what is shown in Figure 5. After entering the code, place the robot on the wooden test block; upload the code and verify that both drive motors move forward for two seconds, then one wheel moves in reverse for less than a second, and then both drive motors move forward again. Try this code out on the floor and see if it really does move the robot in a square. Adjust the motor speeds and delays so that the robot more precisely and continuously moves in a square with two foot sides. Note that the floor of the lab has one-foot square tiles which can aid you in tracking your robot's movement.

```
// Global constants for pin assignments

int pin_motor_left_dir    = 12;
int pin_motor_right_dir   = 13;
int pin_motor_left_speed  = 3;
int pin_motor_right_speed = 11;

// The setup routine runs once when you press reset

void setup() {

  pinMode( pin_motor_left_dir,    OUTPUT );
  pinMode( pin_motor_right_dir,   OUTPUT );
  pinMode( pin_motor_left_speed,  OUTPUT );
  pinMode( pin_motor_right_speed, OUTPUT );

  // Motors are initially turning forward but with zero speed

  digitalWrite( pin_motor_left_dir,   LOW );
  digitalWrite( pin_motor_right_dir,  LOW );
  analogWrite( pin_motor_left_speed,  0 );
  analogWrite( pin_motor_right_speed, 0 );

}

// The loop routine runs over and over again

void loop() {

  // Move forward for two seconds

  digitalWrite( pin_motor_left_dir,   LOW ); // set motor direction with _digital_ write
  digitalWrite( pin_motor_right_dir,  LOW );
  analogWrite( pin_motor_left_speed,  100 ); // set motor speed with _analog_ write
  analogWrite( pin_motor_right_speed, 100 );

  delay(2000);

  // Rotate for 0.7 seconds

  digitalWrite( pin_motor_left_dir,   LOW  );
  digitalWrite( pin_motor_right_dir,  HIGH );
  analogWrite( pin_motor_left_speed,  100 );
  analogWrite( pin_motor_right_speed, 100 );

  delay(700);

}
```

**Figure 5: Move Robot in a Square Code Example**

*Sign-Off Milestone:* Once you have the robot continuously moving in roughly a square with two foot sides, have a teaching assistant verify that things are working correctly.

*Critical Thinking Questions:* Why don't the same motor speed and delay values always result in every robot moving in a perfect square? If the robot begins moving in a perfect square, does it continue to move in a square? Why or why not? Propose a way to enable the robot to more reliably move in a perfect square; Hint: You probably need to add more sensors to the robot. Explain the difference between an open loop controller and a closed loop controller.

## 3. Robotic Behaviors

In this part, you will leverage what you have learned in the first parts to gradually develop a mobile robot control application which can wander its environment and find a target. The first subpart will just have the robot move forward and stop when it bumps into an obstacle. The second subpart will build off of this initial behavior to enable the robot to wander its environment, and the final subpart will allow the robot to spin in a circle when it finds the target.

Note that it is recommended that you start from a fresh version of the code template provided for you. If you have not done so already, you can access the template by selecting the menu item: *File* $\rightarrow$ *Sketchbook* $\rightarrow$ *curie_lab2_template*. Delete any old code in the `loop` routine so you can start from a fresh version.

### 3.A Develop Move-and-Stop Behavior

Write a program which causes the robot to move forward and then stop when it bumps into an obstacle. Figure 6 provides a possible start for your program. Your program will need to use the mechanical bump switches and drive motors. You will not need to use the LEDs or potentiometer, although you can certainly experiment with these added devices in the optional extensions.

Note that the drive motors can cause quite a bit of noise and this can sometimes lead to the robot thinking the mechanical bump switches have been pressed even when they have not. We can try and filter out the these transient spikes in hardware, but we can also filter out the noise in software. If you have issues where your robot appears to incorrectly detect an obstacle which is not really present, simply check each mechanical bump switch twice with a small delay between checks (a delay of 10 milliseconds works well). If the mechanical bump switch is closed during both checks, then it is very likely that the robot has indeed bumped into an obstacle. It is very unlikely that transient noise will cause both checks to be incorrect.

*Sign-Off Milestone:* Once you have the robot moving forward and stopping when it bumps into an obstacle, have a teaching assistant verify that things are working correctly.

### 3.B Develop Wander Behavior

Extend the program from the previous subpart so that the robot now wanders around its environment. For now we will ignore the target. Your robot should move forward until it bumps into an obstacle at which point it should briefly move in reverse and rotate before continuing forward. Recall that the lab notes included a finite-state machine algorithm; this behavior focuses on the FWD, REV, and ROT states. Feel free to sketch out your program on a piece of paper and discuss it with a teaching assistant before starting to write it up in the Arduino software development environment.

*Sign-Off Milestone:* Once you have the robot successfully wandering the environment, have a teaching assistant verify that things are working correctly.

### 3.C Develop Wander-to-Target Behavior

Extend the program from the previous subpart so that the robot now wanders around its environment until it finds the target, and when it finds the target it spins 360 degrees and stop. Recall that the lab notes included a finite-state machine algorithm which might help you design this final mobile robot control application. Feel free to sketch out your program on a piece of paper and discuss it with a teaching assistant before starting to write it up in the Arduino software development environment. Try timing your robot to get a feel for how quickly it can find the target.

```
// Global constants for pin assignments

int pin_bump_right        = 6;
int pin_bump_left         = 7;

int pin_motor_left_dir    = 12;
int pin_motor_right_dir   = 13;
int pin_motor_left_speed  = 3;
int pin_motor_right_speed = 11;

int pin_grayscale_sensor  = A4;

// The setup routine runs once when you press reset

void setup() {

  pinMode( pin_bump_right, INPUT );
  pinMode( pin_bump_left,  INPUT );

  pinMode( pin_motor_left_dir,    OUTPUT );
  pinMode( pin_motor_right_dir,   OUTPUT );
  pinMode( pin_motor_left_speed,  OUTPUT );
  pinMode( pin_motor_right_speed, OUTPUT );

  // Motors are initially turning forward but with zero speed

  digitalWrite( pin_motor_left_dir,   LOW );
  digitalWrite( pin_motor_right_dir,  LOW );
  analogWrite( pin_motor_left_speed,  0 );
  analogWrite( pin_motor_right_speed, 0 );

}

// The loop routine runs over and over again

void loop() {

  // Insert code to check if bump switches are pressed (Hint: See Part 2.A)
  int switch_left_pressed  = ...
  int switch_right_pressed = ...

  if ( switch_left_pressed or switch_right_pressed ) {
    // Insert code to stop robot by setting motor speed to zero
  }
  else {
    // Insert code to move robot straight at moderate speed (Hint: See Part 2.C)
  }

}
```

**Figure 6: Template for Move-and-Stop Robot Behavior**

As a hint, we recommend adding some code after you check the bump switches to read the analog grayscale sensor. If the value from the analog grayscale sensor is greater than the threshold you found in Section 2.B set a variable to one otherwise set this variable to zero. Then you can use this variable to create an additional if/then conditional control statement similar to what we have used to handle obstacles. Finally, to stop the robot you might consider just using a delay statement with a very long delay value (e.g., 10 seconds).

*Sign-Off Milestone:* Once you have the robot successfully wandering the environment and finding the target, have a teaching assistant verify that things are working correctly.

## Optional Extensions

Parts 1–3 are the required portions of the lab. If you complete these parts quickly, you are free to try one or more of the following extensions which are roughly arranged in order of difficulty. You can also make up your extension. These extensions are purely optional; do not feel like you must attempt them.

- **LED State Indicators –** Modify your program so that LED1 turns on when the robot is in the REV state, LED2 turns on when the robot is in the ROT state, and both LEDs turn on when the robot finds the target.
- **Potentiometer Controlled Speed –** Modify your program so that the potentiometer on the prototyping board controls the motor speed.
- **Randomized Wandering –** Modify your program so that the robot randomly determines which direction to rotate and for how long to rotate. How does this impact the robot's ability to explore the space and find the target?
- **Predefined Pattern –** Modify your program so that if the button on the top of the prototyping board is pressed, the robot will move in a predefined pattern (e.g., move in a square) before continuing to wander the space. Your robot should abort the pattern and return to the wandering behavior if it encounters an obstacle while trying to complete the predefined pattern. Can you modify the finite-state machine diagram to appropriately reflect this new behavior?
- **Smart Rotation –** Modify your program so that it takes into account which mechanical bump switch is pressed; when the robot backs up it should always rotates *away* from the bump switch which was closed. How does this impact the robot's ability to explore the space and find the target?
- **Infrared Bump Sensor –** Use the infrared sensor on the front of the robot as an additional bump sensor. You will need to experiment with the analog values from the infrared sensor to determine an acceptable threshold (similar to what you did with the grayscale analog sensor). Augment your finite-state-machine algorithm so that if the robot senses an obstacle either from the mechanical bump switches or the infrared sensor it will then enter the REV and ROT states. How does this impact the robot's ability to explore the space and find the target?
- **Infrared Wall Following –** With careful tuning, there may not be a need to move in reverse after sensing an obstacle with the infared sensor. Try to build a more elegant wall following control application: when the robot senses an obstacle with the infrared sensor it can stop and immediately rotate until the infared sensor indicates there is open space; the robot can then proceed. If done correctly, this will cause the robot to follow the walls without needing to move in reverse.
- **Infrared Mapping –** It may be possible to generate a simple map by rotating in place and sampling the infrared sensor. For example, the robot could rotate in-place at the very beginning to determine which direction has the most open space. The robot could then move in this direction and use similar "scans" to determine how to proceed.
- **Navigate Around More Obstacles –** Ask the instructors for additional obstacles to place in the environment. How does the robot handle these new obstacles? How can you modify your control application to robustly handle more complicated environments?

- **Gray Line Following –** Ask the instructors for a strip of gray tape and place this tape in the robot's environment. Is it possible to design a control application which allows the robot to "follow" this tape to the target?