ECE 110L Spring 2023

Laboratory 1:

Motion & Control—Serial Communication

Contents

1 Abstract

The concept of **feedback** is important and useful in engineering. Feedback is defined as a comparison between the output of a system and the input of a system. Often, feedback is used to maintain or control certain behavior of a system by enabling informed decisions to be made.

In this lab, you will first learn how to run and characterize **continuous rotation servomotors**. These servomotors are mounted on a BOE-Bot chassis. They first will be used to propel the robot in all directions in an **open-loop feedback system** to tune the Arduino commands used to drive the motors. By the end of the laboratory, you will have a fully mobile robot! The chassis and motors will form the foundation of the robot you will be using to complete the Integrated Design Challenge (IDC), though the interface to the Arduino will be different. Also, during the IDC it will be necessary to implement sensors on your robot. These sensors will generally be part of a **closed-loop feedback system**.

In addition, you will explore wireless communication using your newly built robot. The XBee wireless communication module will be used to perform the task of communication in this laboratory. XBee uses standard wireless protocols to send radio-frequency (RF) signals through the air which are intercepted by nearby robots and can be used to send information between 'bots wirelessly. In this Exploration, you will communicate a single ASCII[1] character with a pre-built sentry bot. Communication is an essential part of the Integrated Design Challenge (IDC) and will be critical to your team's success!

2 Objectives

After performing this laboratory exercise, students should be able to:

1. Explain the difference between an open-loop control system and a close-loop feedback control system
2. Calibrate and program continuous rotation servomotors
3. Communicate wirelessly using an XBee module with a sentry robot

[1] http://www.asciitable.com/

3 Background

3.1 Feedback

Feedback occurs when an output signal or measured value in a system is compared with the input or desired response of the same system. A **feedback control system** uses the information provided through feedback to maintain a certain relationship between the output signal and the reference input signal. An example of a feedback control system is a thermostat used for controlling the temperature in a room. The thermostat compares the current temperature of the room (output signal) to the desired temperature (reference input signal) indicated by the temperature setting established by the user. The system then acts in a way that minimizes the difference between the two signals (e.g., turn heater/AC on/off).

A system can be a **closed-loop system** or an **open-loop system**. The primary difference between the two is whether the output signal is used to determine how to drive the system. A **closed-loop system** is generally one in which the system attempts to minimize the difference between an input signal and an output signal. The thermostat described earlier is an example of a closed-loop system. The desired temperature (input reference signal) set by the user is compared to the actual room temperature (output signal) and the difference between the two signals is minimized by activating (or deactivating) the heater/AC in order to achieve the desired temperature. In practice, a feedback control system is synonymous with a closed-loop feedback system.

An **open-loop system** is one in which the output signal is neither measured nor fed back to the input for comparison and thus has no controlling effect. An example of an open-loop system is a washing machine. In a washing machine, the soak, wash, and rinse times are pre-set. The machine does not measure the output signal (i.e., the cleanliness of clothes) to determine how much longer to continue washing. In an open-loop system, the output signal is not compared to a reference. Clearly, an open-loop system is not a feedback control system.

3.2 Block Diagrams

Block diagrams are often used to visualize and quickly explain the operation of a system without having to examine the physical system itself. A block diagram symbolizes mathematical operations within a system that relate the output signal to the input signal. An example of a block diagram is shown in Figure 1. Note that this system is an open-loop system with no feedback.

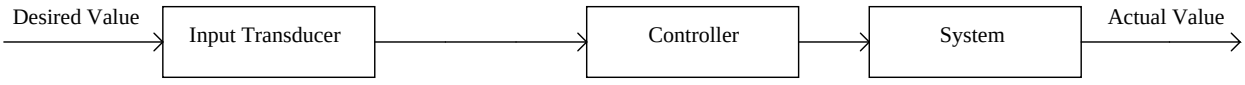

Figure 1. Block Diagram showing an open-loop system.

A toaster is another example of an open-loop system. The *Desired Value* might be the darkness of the toast. That desire needs to be translated into a signal the system can understand - perhaps a voltage. On many toasters, the “desired toastiness” value is established by turning a dial on the front of the toaster - this is a potentiometer (variable resistor) knob. Based on the position of the knob, a resistance will change. If that variable resistor is placed in series with a constant voltage source and another constant resistor, you have a voltage divider whose voltage across the potentiometer is determined by the position of the dial. This *transduces* a difficult-to-measure

signal ("desired toastiness") into something easily measured - a voltage. This voltage can be fed into a timing circuit, which acts as the *Controller*. The controller in this case would determine how long the heating coils are energized before the toast pops out. The heating coils and the rest of the thermal, mechanical, and electrical systems of the toaster (and the toast) are collectively called the *System*. Once the toast pops out, you can determine the *Actual Value* by seeing just how toasted the toast is. This system is open-loop because no feedback mechanism exists for the controller to know what the output *actually* is.

Block diagrams consist of functional blocks, directional arrows, summing points, and branch points. The *functional block* contains information about how the signal entering that block is acted upon by the system. For example, is the signal amplified, sped up, or slowed down? A circuit that filters the signal, such as a series RC circuit, could be represented by a functional block. An *arrow* pointing in a given direction represents a signal and its flow through the system. The signal can be an input, an output, or an intermediate signal. A *summing point* is represented by a circle with a "+" sign, indicating that the signals at this junction are added together. Note that a "-" next to a summing point means that particular signal is actually *subtracted* when calculating the output of the summing point. A *branch point* is simply a place in the system where a signal is split, often to be fed back to another point in the system.

Figure 2 shows a general closed-loop feedback system. Note the branch point beneath the words `Actual Value` where the signal is both an output of the system and an input to the Output Transducer.

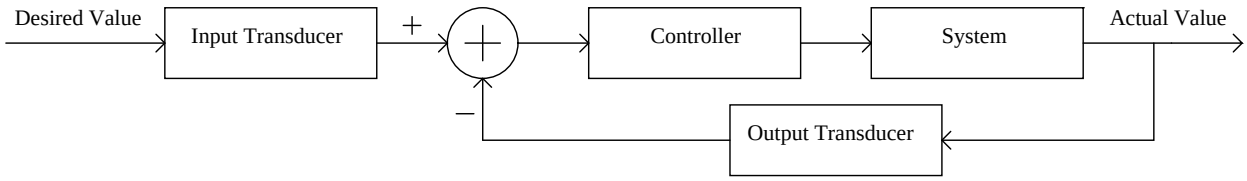

Figure 2. Block diagram showing a closed-loop feedback system.

In this case, the *Desired Value* is converted to a signal that can be compared with another signal. The *Actual Value* is *also* converted to a signal that can be compared with another signal. For example, with a heating, ventilation, and air-conditioning (HVAC) system, the desired temperature can be set via a digital interface or the position of a physical slider or rocker, but somehow that value needs to be converted into some kind of signal—perhaps a voltage. The actual temperature would need to be converted to comparable voltages as well so the control system could determine a signal that represents the difference in temperature. That difference signal—which is the output of the summation block—is fed to a *Controller*. The *Controller* then determines what to do with the heating and air conditioning system to bring the desired and actual values for temperature closer together. The *System* in this case includes the heating, ventilation, and air conditioning system, the house and everything in it, and basically all the things that need to be known in order to determine how turning your air conditioning on might impact the temperature of your house.

3.3 Servomotors

Servomotors are unique motorized devices that can be used to turn an object either a fixed distance or at a fixed speed. They are often used in remote control (RC) applications such as the ailerons on an airplane, the rudder on a boat, or the wheels on a robot. There are two major types of servomotors: standard servomotors and continuous rotation servomotors. Both types of servomotors are controlled by a **pulse-width modulated (PWM)** input signal. There are two important parameters of a PWM signal: the frequency (or its reciprocal, the period) and the duty-cycle (the ratio of the time the signal is high to the period). For the servomotor used in this lab, the PWM input is a series of brief high pulses that last anywhere from 1.3 to 1.7 ms and occur every 20 ms. (Note: These parameters can vary with each specific brand of servomotor). There is a graph of a signal such as this near the middle of the page at http://learn.parallax.com/node/179. This figure shows a pulse-width modulated signal with pulses of 1.5 ms occurring every 20 ms (i.e., a frequency of $f = 1/0.02 = 50$ Hz and a 7.5% duty cycle). This kind of figure is called a **timing diagram**.

Continuous rotation servomotors use a PWM input signal to move the servomotor at a specific speed and in a specific direction. Your servomotors interpret a train of 1.5 ms wide pulses repeated every 20 ms as a "centering" command. This command instructs the servomotors to remain still. To achieve motion, the pulse duration can be modified (although the period of the signal will remain 20 ms). The speed and direction of the motor both depend upon how far from "center" the pulse duration is set (i.e., how much it deviates from 1.5 ms) and whether it is greater than or less than "center." If the pulse width is greater than 1.5 ms, the motor will turn counterclockwise (CCW). If it is less than 1.5 ms, it will turn clockwise (CW). A deviation of 0.2 ms (e.g., 1.7 ms or 1.3 ms) tells the motor to rotate at full speed. Smaller deviations will cause the motor to rotate more slowly. For example, a pulse duration of 1.7 ms tells the servomotor to turn CCW at full speed. A pulse duration of 1.6 ms also tells the servomotor to turn CCW, but at a slower speed. A pulse duration of 1.3 ms indicates full speed CW rotation. Continuous rotation servomotors are useful for applications where rotary motion is desired such as a dispensing wheel for pharmaceuticals, a variable speed fan, or the wheels on a robot.

Standard servomotors are also controlled using a similar PWM signal input. The difference is that a standard servomotor uses this input signal to very precisely control the *position* of a shaft rather than its *speed*. The shaft on a standard servomotor can be set to a position within a certain number of degrees of its 0° center location. To alter the position of the shaft, the pulse width of the modulated signal is varied. For a standard servomotor, the pulse turns the motor toward or away from its 0° position and holds it there. Standard servomotors are useful for applications for which careful control of position are required such as the ailerons on a plane, the rudder on a boat, or the valve opening on an air-intake system.

For this lab, servomotors will be part of an **open-loop** control system. You will be running the motors for a pre-set period of time in the hope of getting the 'bot to move a certain distance or turn a certain angle. In a later lab, you will use sensors designed to read whether or note the 'bot is centered on a dark line placed on a light background. Those sensors will give feedback to the 'bot and the 'bot will use a **closed-loop** control system to follow the line.

3.4 BOE-Bot Continuous Rotation Servomotor: Hardware

Figure 3 shows a continuous rotation servomotor used with the BOE-Bot to make the wheels turn. Let us turn our attention to some of the parts called out in the diagram of the servomotor.

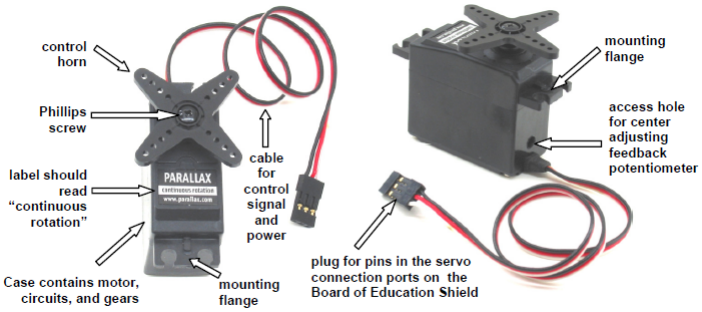

Figure 3. Continuous Rotation Servomotor supplied with BOE-Bot from Chapter 2 Activity 4 of Robotics with the Board of Education Shield for Arduino

(a) **Label:** The label on the servomotor indicates the servo type. There are two basic types of servomotors - standard and continuous - make sure you choose the right one for your application.

(b) **Case:** The case houses all of the electronics and hardware including gears that receive the PWM signal and turn the servo shaft.

(c) **Control horn:** The servo comes equipped with two kinds of plastic mounting apparatus. One (as shown) is a control horn. This is most useful for armature position control. The second is a plastic wheel. This is most useful for motion.

(d) **Phillips screw:** The screw located in the center of the servo mounting apparatus is a self-tapping type which holds the mounting assembly tightly to the rotating servo shaft. This screw makes it easy to replace wheels or other shaft-mounted hardware on the servo.

(e) **Power Cable (3 wires):** Notice that the servo power cable has three wires bundled together. These are power (red), control (white), and ground (black). The servomotors for the CX-Bot need a minimum of 5 V to turn, but typically operate between 6 and 9 V DC. The control signal is a PWM signal as described above. Ground connects to the negative terminal of the voltage source in the system.

(f) **Plug:** The plug at the end of the servo power cable allows an easy slip-on connection to the Board of Education servo ports.

(g) **Mounting flange:** The mounting flanges, located on either end of the servomotor are used for attaching the servo to a fixed structure for stability. The mounting flanges on the continuous rotation motors provided with the CX-Bot attach nicely to the robotic chassis.

(h) **Feedback potentiometer:** The access hole for adjusting the feedback potentiometer (used to center the servomotors at 1.5 ms pulse duration) is recessed below the face of the servo case to protect it from being adjusted unintentionally.

4 Pre-Laboratory Exercises

1. A few examples of open- and closed-loop systems were given in the background section. These included the toaster oven and washing machine for open-loop systems and the thermostat/HVAC system for closed-loop systems.

   Pre-lab Deliverable (1): Provide one more example that you can think of for each type of feedback system (i.e. closed- and open-loop). For the closed-loop system, indicate how the output signal is measured or fed back and used to control in the system.

2. Look at the code in Chapter 4 Activity 1 of the Robotics tutorial.[2] This code is meant to make the robot move forward at full speed for three seconds.

   Pre-lab Deliverable (2): Sketch the PWM waveforms that will result from the following Arduino code:

   ```
   servoLeft.writeMicroseconds(1700)
   servoRight.writeMicroseconds(1300)
   ```

   Be sure to label the time axes with time in milliseconds on the same scale for each waveform. Draw the two waveforms one above the other and clearly indicate which is for the left motor and which is for the right.

3. In Tinkercad, start a new project and then drag in the Button starter for Arduino. Convert the code from blocks to text and examine the code and the circuit. During the lab, you will use similar code, except pushing a button will be a trigger for your 'bot to wirelessly communicate a letter to another 'bot. There is no deliverable for this, but you are expected to understand the code and circuit before lab.

5 Pre-Laboratory Assignment

The documentation for the pre-lab includes the following contained in a single PDF file: Your document must include your name, NetID, and the Duke Honor Code statement: "I have adhered to the Duke Community Standard in completing this assignment" at the top. **EACH INDIVIDUAL** should submit their own assignment. Your document should also include:

1. Your examples for an open-loop system and a closed loop system, along with an explanation of how the closed-loop system uses feedback to control the system.

2. Sketches of the two different PWM signals generated by the code in Chapter 4 Activity 1. These are sketches, so they can be drawn by hand, but you must include axis labels (with units) and a title for each indicating which motor's signal you have drawn. Your sketch needs to have an accurate scale, so you may consider using graph paper or a ruler. You are also allowed to make a plot, versus a sketch, if you so choose. You can take a picture of your sketches and import them into your document.

[2] https://learn.parallax.com/tutorials/robot/shield-bot/robotics-board-education-shield-arduino/chapter-4-boe-shield-bot-16

This file should be uploaded to the ECE 110L Laboratory **Gradescope** site by the assignment deadline. Each student must submit their own **INDIVIDUAL** assignment.

6 Equipment

1. (1) LED—Red
2. Parallax screwdriver (Phillips #1 point screwdriver 1/8"(3.18 mm) shaft)
3. 1/4" Combination wrench (Optional)
4. Needle-nose pliers (Optional)
5. Bot-Box (Rubbermaid®) including the following items:
    - CX-Bot shield with mini-breadboard
    - BOE-Bot chassis—with (2) Parallax Continuous Rotation servos mounted and (2) Phillips screws, threaded (for servomotor wheel mounting at hub)
    - (4) 1" Standoffs
    - (4) Pan head screws, 1/4" 4-40
    - Rubber grommet, 13/32"
    - (10) Nuts, 4-40
    - Li-Ion Battery Pack[3]
    - 1/16" Cotter pin
    - Omni tail wheel
    - (2) Plastic wheels with rubber grip treads
6. (2) Resistors 10 kΩ (Brown-Black-Orange)
7. Jumper wires
8. XBee Communicaton Module
9. Pushbutton
10. Parallax text: **Robotics with the Board of Education Shield for Arduino**, available online at: http://learn.parallax.com/ShieldRobot

[3]Remember to move the jumper position on the CX-Bot to accommodate the power scheme used.

7 Experimental Exercises

7.1 Motion Control

In the following exercises, you will connect the servomotors to the CX-Bot and calibrate and test each motor. Finally, you will command the CX-Bot to perform a few basic maneuvers and will record your observations regarding its performance.

- Start with **Chapter 2** of the **Robotics with the Board of Education Shield for Arduino** site. You will go from "Chapter 2. Shield, Lights, Servo Motors" through "How to Use the Arduino Servo Library".
  Note: Skip Activity 4 because the battery packs you are using and the servos have been installed already.

  Checkpoint (1): Demonstrate the sketches you created, ending with the BothServosStayStill sketch in Chapter 2 Activity 3 "How to Use the Arduino Servo Library" section with your CX-Bot.

- Continue **Chapter 2** at: "Activity 5: Centering the Servos" and go to "Chapter 2 Summary".
  Note: The CX-Bot is already built, the shield is mounted on the chassis, and the servomotors are working.

  Checkpoint (2): Demonstrate the sketches in Chapter 2 Activity 6 with your CX-Bot.

- Next, go through **Chapter 3**, Activities 1 and 2, of the **Robotics with the Board of Education Shield for Arduino** site. Although most of this has already been done, it is good for review. Note that "Servo Troubleshooting" is especially good for troubleshooting issues with the servos.

- Finish by learning how to navigate with the CX-Bot. Go through **Chapter 4** Activities 1 and 2 of the **Robotics with the Board of Education Shield for Arduino** site. This starts at "Chapter 4. BOE Shield-Bot Navigation" and ends at "Tuning the Turns". Here, you will make the CX-Bot
  - turn exactly 90° in both directions
  - move forward in a straight line for 6 ft.

  Tuning the servomotors is an extremely important part of robot navigation. If there are issues, correct them here and note in your notebook the corrections you needed to make.

  Checkpoint (3): Demonstrate sketches that show your CX-Bot rotating 90° to the left, 90° to the right, and then going straight for 6 ft.

  Deliverable (1): Save the sketch that you used for this last exercise. Note that you may need to adjust the arguments to the `writeMicroseconds` commands to make the 'bot travel in a straight line.

7.2 Wireless Communication with XBee Modules

7.2.1 What is XBee?

XBee modules are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing. XBee modules are ideal for low-power, low-cost applications. Part of the XBee family of RF products, these modules are easy-to-use, share a common footprint, and are fully interoperable with other XBee products utilizing the same technology. Module users have the ability to substitute one XBee module for another with minimal development time and risk.

7.2.2 Serial Communication with the XBee

In general, there are two categories of communication protocols: parallel and serial[4]. XBee modules operate as an asynchronous serial interface. Asynchronous serial interface means that data is transferred without the help of an external clock. The asynchronous serial protocol takes advantage of four different parameters to avoid using an external clock: data bits, synchronization bits, parity bits, and baud rate. The data bits are the information you wish to send over the serial connection. Synchronization bits, or start and stop bits, indicate the beginning and end of a packet. Parity is optional as a means of error checking and it is not used in the standard XBee configuration for ECE 110. Finally, baud rate is how many bits per second are sent through the serial line. XBees in ECE 110 use the 9600-8-N-1 configuration. The number 9600 corresponds to the baud rate in bits per second. The number 8 indicates the number of data bits. The letter N means that there is no parity bit. The number 1 equals the number of stop bits. Since XBee's in ECE 110 are configured as 9600-8-N-1, only one ASCII character at a time can be sent in a serial packet. See the ASCII table at http://www.asciitable.com/ for information on what characters are available.

7.2.3 XBee Communication Networks

We will explore XBee communication as part of a hub and spoke network. When drawn out, a hub and spoke network resembles a bicycle wheel. There is a central node, the hub, which sends and receives transmissions to and from the other nodes, the spokes. Figure 4 below shows a simple hub and spoke network. In this exploration, the XBee located at the Teaching Assistant (TA) bench will act as the hub, and your lab section's XBees will act as the spokes.

Figure 4. Wiring your XBee Module for Communication.

[4] https://learn.sparkfun.com/tutorials/serial-communication

7.2.4 Wiring your XBee Module

Figure 5 depicts the XBee module you will be plugging into your CX-Bot. Note that in this diagram, V_{ss} is connected to GND. V_{dd} is connected to +5V. Each bot's XBee should be wired according to this configuration. The **TX** and **RX** pins will be wired to Serial 2 on the communication bus.

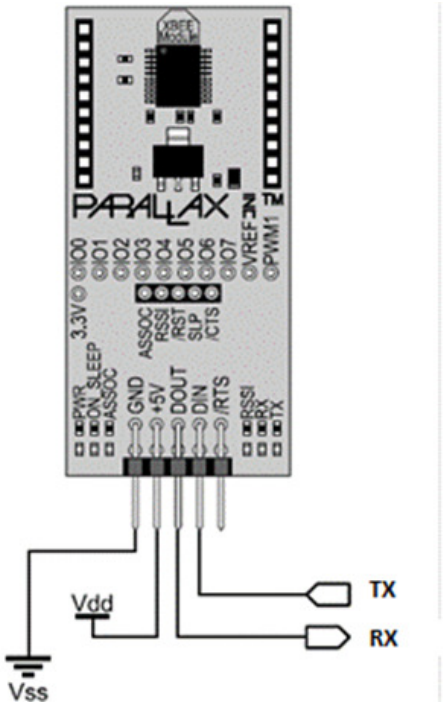

Figure 5. Wiring your XBee Module for Communication.

7.2.5 Serial Communication for the Arduino ATMEGA 2560

Some of the sensors that are available for the IDC require the use of serial communication (XBee, ColorPAL, RFID, etc.). The ATMEGA 2560 offers three serial port options: Serial 1, 2, and 3. On your CX-Bot, Serial 2 is already dedicated to the XBee. When connecting the XBee to the CX-Bot, make sure the **GND** pin of the XBee and dedicated bus are lined up. **Note:** The DOUT and DIN labels are labeled one sport further than they should be!

Port Name	TX Pin	RX Pin	
Serial1	18	19	
Serial2	**16**	**17**	**⟵ USED IN THIS LAB**
Serial3	14	15	

Table 1. ATMega 2560 Serial Port Options.

Note: Serial port 0 (Serial) is connected to the USB–to–TTL chip on the Arduino. This port should be reserved for communication with a PC only.

Additional information about the I/O pins for the ATMEGA2560 can be found in Sensor/Component & Info documentation student resource on Sakai.

7.2.6 Exercise 1: Transmitting data with the XBee Module

The first task of this XBee exploration is sending a unique character to a "Sentry" robot in the lab which will be controlled by your TA. The Sentry bot will receive your character and display it on the serial monitor. The Sentry bot uses the code shown below to display the unique characters it receives from all of the bots in your lab section. The sentry bot is using the communication bus to achieve serial communication.

```
void setup() {
  Serial.begin(9600); // initialize serial monitor
  Serial2.begin(9600); // initialize Xbee Tx/Rx
  delay(500);
}

void loop() {
  if(Serial.available()) { // Is serial monitor data available?
    char outgoing = Serial.read(); // Read character
    Serial2.print(outgoing); // Send to XBee
  }
  if(Serial2.available()) { // Is XBee data available?
    char incoming = Serial2.read(); // Read character
    Serial.println(incoming); // Send to serial monitor
  }
  delay(50);
}
```

Table 2. Sentry 'Bot Communicaton Code

REMINDER: Directly pasting .pdf code into an Arduino sketch will result in extraneous characters, often blank spaces, that will not allow the sketch to compile.

Your goal for this exercise is to write code for your 'bot so that it transmits a unique character to the sentry when you press a push button. Basically, you need to take the code that uses a pushbutton to control the output to an LED and change it so it uses a pushbutton to determine when to send a unique character over Serial2 to the sentry. Your TA will give you your unique character. Once your teaching assistant verifies that you have successfully transmitted a character to the sentry, proceed to Exercise 2.

Checkpoint (4): Show your TA that your 'bot can transmit a unique character to the sentry.

Deliverable (2): Save the sketch you used for this exercise, as it will be included in your lab assignment.

7.2.7 Exercise 2: Receiving data with the XBee Module

In this exercise, your teaching assistant will transmit a character using the Sentry bot. The Sentry bot will use the same code from Exercise 1 to transmit this character. Your goal for this exercise is to modify your transmission code to indicate that you have received a character s from the Sentry by turning on the on-board RGB LED for one second and displaying the character on the serial monitor. For information on how to use the RGB LED, go through the EGRWiki page at https://egrwiki.com/wiki/ECE_110/Equipment/Built_in_RGB_LED and specifically the "Simple with functions" sample code. You should be able to upload that sketch to your 'bot to get a flashing red light. Make sure you understand the different parts of the code.

Checkpoint (5): Show your TA that your 'bot can receive and display a character from the sentry.

Deliverable (3): Save the sketch you used for this exercise, as it will be included in your lab assignment.

8 Exploration

8.1 Vowels and Consonants and Y, Oh My!

Now that your 'bot can receive a character and control the on-board RGB, you are going to turn your 'bot into a letter classifier. Specifically, your bot will need to determine if it received a vowel (A, E, I, O, or U), the letter Y (which is sometimes a vowel and sometimes not), or a consonant. It will then display that information on the serial monitor. Also, if the letter is a vowel, the RGB LED should be green. If the letter is Y, the RGB LED should be blue. If the letter is a consonant, the RGB LED should be red. The light should stay on for one second after the letter has been received.

Notes:

- Be sure to reference the EGRWiki page at https://egrwiki.com/wiki/ECE_110/Equipment/Built_in_RGB_LED and specifically the "Simple with functions" sample code. You should be able to upload that sketch to your 'bot to get a flashing red light. Make sure you understand the different parts of the code.
- Create a new sketch and then copy the code from Exercise 2 above into it. You will now make changes to it so that it can classify what kind of letter you have and make the light a particular color based on that classification.
- For this exploration, you may assume that the sentry 'bot will only be sending out capital letters.
- To see if a letter is a vowel, you will need to compare the information received from the XBee to the *numerical* value of a given letter from the ASCII table. For instance, if you want to see if the letter is "A", you could write:

```
if(incoming==65){
...
}
```

- To generate a logical OR in Arduino code, use the "double pipe" operator ||. For example, if you want to see if the incoming letter is either A or E, you could write:

```
if(incoming==65 || incoming==69){
...
}
```

- Arduino code allows for if...else if...else statements. Each statement has its own code in curly brackets:

```
if(LOGIC FOR IF){
... runs if logic for if is true
}
else if(LOGIC FOR THIS BRANCH){
... runs if logic for if is *not* true but logic for this
   branch is
}
else if(LOGIC FOR BRANCH 3){
... runs if logic for if and branch above are both false but
   branch 3 logic is true
}
else{
... runs if no other branch runs
}
```

Checkpoint (6): Show your TA that your 'bot can receive a and display a character from the sentry.

Deliverable (4): Save the sketch you used for this exploration, as it will be included in your lab assignment.

8.2 No Such Thing as a Free 'Bot

As part of the Integrated Design Challenge, you will need to account for the cost of your finished product. To help, as a part of this laboratory you need to carefully approximate the cost of the 'bot and any hardware you have used to complete the work for all parts this assignment.

First, identify and list each part of the 'bot. For several of the pieces, you may use the cost of the `Robotics with the Boe-Bot Parts Kit` from Parallax as a starting point, but should carefully estimate how much to take *off* that cost based on parts of the kit that you did *not* use and then add back costs for additional parts that are not in that kit. For purposes of this exercise, the CX-Bot Shield (just the shield, not the Arduino attached to it, but including the built-in RGB LED) costs approximately $30 for production and shipping.

Deliverable (5): Provide a detailed parts list and cost estimate of your 'bot.

9 Assignment

The assignment for this laboratory involves submitting a single PDF file. Your document must include your name, NetID, and the Duke Honor Code statement: "I have adhered to the Duke Community Standard in completing this assignment" at the top. **EACH INDIVIDUAL** should submit their own assignment. Your document should also include the following:

1. Four Arduino sketches: one for making sure the 'bot can rotate 90^o in either direction and travel straight for 6 feet, one for sending a character to a sentry 'bot, one for receiving and displaying a character delivered from a sentry bot, and one for classifying the letters received and controlling the RGB LED.
2. A detailed parts list and cost estimate of your 'bot.

This file should be uploaded to the ECE 110L Laboratory **Gradescope** site by the assignment deadline. Each student must submit their own **INDIVIDUAL** assignment.