|  |  |  |
| --- | --- | --- |
| **LEFT SIDEBAR** | **MAIN CONTENT** | |
| **IMAGE** |
| **TITLE: CODING LEVEL 1** |
| **BLURB:** Learn the very basics of computer coding. |
| **LEFT MENU:** Overview, Pulse Width Modulation, Servo Comms, Serial Comms, Conclusion |

|  |  |
| --- | --- |
| *URL: engimake.com/coding/l1/0*  **Overview (H1)**  **Microcontrollers would be quite limited if digital IO and analog input were the only capabilities. In reality there’s a whole lot more IO capabilities microcontrollers can achieve…**    **Let’s explore some of these...** | |
| **RIGHT BUTTON**  TEXT  URL | **LEFT BUTTON**  TEXT: Pulse Width Modulation  URL: *engimake.com/coding/l1/1* |

|  |  |  |
| --- | --- | --- |
| *URL: engimake.com/coding/l1/1*  **Pulse Width Modulation (H1)**  **Pulse width modulation (PWM) is a method used to send analog data from a digital output. This seems impossible since a digital output can only either be 0V or 5V.**  **The trick is that we can vary the time that the signal is at 5V, like a pulse. By varying the time that the pulse of the, we can, send out an analog signal. However it’s not the voltage that is analog, but the pulse width. We *modulate* the pulse width to send analog information.**  **We can start with a signal that has the same time duration for 5V and 0V.**    **We call this a PWM signal with 50% duty cycle, because the signal is 5V for 50% of the time. The average of this signal is 2.5V.**  **If we connected an LED to this, we’d expect the brightness to be roughly half that if we had applied 5V.**    **This PWM signal has a duty cycle of 75%. The average voltage is 75% of 5V (3.75V). On our LED, we’d expect the brightness to be slightly less than at 5V.**  **In this way we can output an analog voltage from a digital pin, meaning we can dim LEDs, control motor speed, and many more robot related shenanigans. This method is used everywhere, in fact it’s how your laptop dims the brightness of its screen!**  **To achieve this in our code, we do the following…**   |  | | --- | | **int pin, dutyCycle;**  **void setup(){**  **pinMode(pin, OUTPUT);**  **}**  **void loop(){**  **analogWrite(pin, dutyCycle);**  **}** |   **To output a duty cycle of of 100%, we set dutyCycle = 255, for a duty cycle of 0%, we set dutyCycle = 0. For 50%, dutyCycle = 128 and so on.** | |
| **RIGHT BUTTON**  TEXT: Overview  URL: *engimake.com/coding/l1/0* | **LEFT BUTTON**  TEXT: Servo Comms  URL: *engimake.com/coding/l1/2* |

|  |  |  |
| --- | --- | --- |
| *URL: engimake.com/coding/l1/2*  **Servo Comms (H1)**  **Normal DC motors are simple. You apply a voltage, and the motor shaft spins with a speed corresponding to the voltage. Servo motors are different. They are designed to move to a specific angle, then hold that angle against all resistance. That’s why they’re so darn useful for legged robots like QuadBot.**  **To control them we use a special type of PWM signal from our microcontroller. This PWM signal uses a specific timing that the servo understands. The timing is a HIGH pulse between 1mS to 2mS.**  **A 1mS pulse means the servo shaft will turn to 0 degrees, whilst a 2mS pulse means the servo shaft will turn to 180 degrees.**    **Thankfully we don’t need to worry the specifics because Arduino has a built in library for this.**  **To code this we write the following.**   |  | | --- | | **#include <Servo.h> //Include the servo library**  **Servo myServo; //Declares a servo called myServo**  **int pin, angle;**  **void setup(){**  **myServo.attach(pin) //Tell the program which pin the servo connects to**  **}**  **void loop(){**  **myServo.write(angle); //Send an angle to the servo**  **}** | | |
| **RIGHT BUTTON**  TEXT: Pulse Width Modulaion  URL: *engimake.com/coding/l1/1* | **LEFT BUTTON**  TEXT: Serial Comms  URL: *engimake.com/coding/l1/3* |

|  |  |  |
| --- | --- | --- |
| *URL: engimake.com/coding/l1/3*  **Serial Comms (H1)**  **Serial might sound like something you eat for breakfast, but the meaning here is quite different. Here serial means one after the other, like serial killer. Serial communication sends data by sending one bit after the other on a single pin.**  **Data is sent from the microcontroller by turning a digital pin on and off to signal each bit. It’s really no different from what we did to blink an LED, only faster and with a specific set of rules.**    **When the microcontroller sends eight bits in succession, that is a byte. Each 8-bit byte can store a character (a char data type).**  **We can send as many Chars as we like via serial communication to our computer, from the microcontroller. Here’s how we do it...**   |  | | --- | | **void setup(){**  **Serial.begin(9600); //Set up the serial port with 9600bps**  **}**  **void loop(){**  **Serial.print(“Hello Mr Roboto”); //Print the text to the serial port**  **}** |   **The Serial.begin(9600) tells the microcontroller to get ready to send data with serial communication. 9600 is the speed of the data transfer in bits per second. Serial.print() sends the characters inside the quotes over the serial port, one by one. Each character needs 8 bits to be transmitted.**  **We can read this data on our computer by listening to the serial port. We can also send data from the computer to the microcontroller in a similar way.**  **Serial communication is very helpful when we’re debugging our code, or wanting to control the QuadBot from another device.** | |
| **RIGHT BUTTON**  TEXT: Servo Comms  URL: *engimake.com/coding/l1/2* | **LEFT BUTTON**  TEXT: Conclusion  URL: *engimake.com/coding/l1/4* |

|  |  |
| --- | --- |
| *URL: engimake.com/coding/l1/4*  **Conclusion (H1)**  **Now you’re fully schooled up in input and output, you’re ready to tackle programming QuadBot fully. Go check out the getting started guides!** | |
| **RIGHT BUTTON**  TEXT: Serial Comms  URL: *engimake.com/coding/l1/3* |  |