Lab 3: Motion

For the rest of this semester you will be designing S1XT33N, a mischievous little robot who *might* just do what you want - if you design it correctly. Our end goal is to have S1XT33N listen to your voice commands and execute the corresponding drive command. Over the course of this semester, we will be implementing different parts of this project, from the circuits to the algorithms necessary for voice classification and control. In Lab 2, you learned about digital/analog interfaces, which is how your Arduino, which will serve as the brain of S1XT33N, collects information from and interacts with the world. In this lab, **you will build S1XT33N's legs**: you will build the motor control circuits that enable S1XT33N to move around and the encoder circuits that enable your Arduino to detect how far and fast S1XT33N has moved. Furthermore, you will be building voltage regulator circuits that allow for the circuits to be battery-powered, as we need a portable way to power the circuits so that S1XT33N can freely roam on the ground.

The goals of this phase are as follows:

- Construct the motor controller circuits
- Construct the encoder circuits
- Test encoder circuits and motors
- Construct the voltage regulator circuit

Part 1: Motor Drivers

PWM Signal

S1XT33N uses 9V DC motors. This means the maximum voltage that can be delivered to the motors is 9V, but the motors will move (albeit more slowly) with voltages less than 9V. There is some minimum voltage required to deliver enough power to the motors to overcome the static friction and start them, but after that point, we treat the motor speed as approximately linear with the applied voltage (this will be the basis of the system model you develop in a future lab).

Because it is difficult to use the Arduino to control a true DC signal within 0V and 5 V (e.g. 2.2 V), we will instead make use of its PWM function. A PWM, or pulse-width modulation, signal is a square wave with a variable duty cycle D (the proportion of a cycle period for which the power source is turned on, or logic HIGH) shown in Figure 1. PWM is used to digitally change the average voltage delivered to a load by varying the duty cycle. If the frequency is large enough, the on-off switching is imperceptible, but the average voltage (V_{AVG}) delivered to the load changes proportionally with the duty cycle, as shown in the figure. Hence, changing the duty cycle corresponds to changing the DC voltage supplied to the motor. For example, you can get an average voltage of 2.2V by setting the duty cycle to $\frac{2.2}{5} = 44\%$.

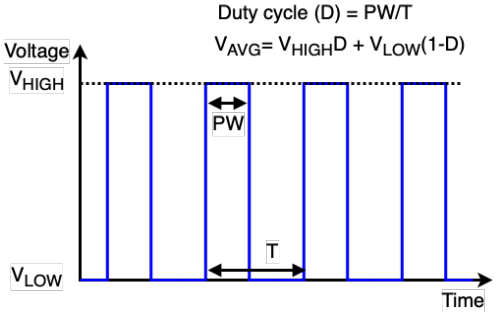


Figure 1: PWM Signal

We will use the Arduino's PWM pins to set the duty cycle for each motor's PWM signal. However, the Arduino's logical HIGH[1] voltage is 5V, which is not enough to directly drive the motors: even if your motor does turn on at 5V

[1]The logic-level signals are denoted as HIGH and LOW, where HIGH=5V and LOW=0V for the Arduino.

when you test it with the power supply, the Arduino cannot supply enough current to power the motors (the motors each need 100-200 mA to run, while the Arduino can only supply around 20 mA from its pin). This is where the motor driver circuits come in.

Bipolar Junction Transistor (BJT)

The PWM pin from the Arduino is connected via a resistor to the "Base (B)" of an NPN BJT (bipolar junction transistor). This transistor, in reality, behaves a bit differently from the NMOS with which you are familiar, but for this class, you may assume that it is functionally the same as an NMOS, behaving as a electronically controlled switch. On the BJT, the three terminals are analogous to those of an NMOS: the "Base (B)" is the gate, the "Collector (C)" is the drain, and the "Emitter (E)" is the source. The reason the resistor is there is because, unlike MOS transistors, there is a current flowing into the "Base (B)" terminal of the BJT which directly affects the current flowing through the "Collector (C)" terminal. In more fancy terminology, this means that the BJT acts like a **current controlled current source** across its C and E terminals.

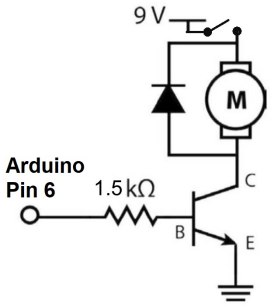


Figure 2: Motor Controller Circuits

To drive our car and control when the motors receive power, we utilize the Arduino's PWM pins and drive the voltage at their output to HIGH (5V) or LOW (0V - GND). The voltage outputted by the pin generates a current from the Arduino pin output to GND. If not for the presence of the base resistor R_B which is set to $1.5k\Omega$ [2], this would create a short to ground and drive a lot of current through the BJT and the Arduino pin potentially damaging both components.

When current is driven into the base (B) terminal, the BJT switches to the ON state and when the flow stops, the BJT can be modelled in the OFF state. But, for the sake of simiplicity, we can simplify the model such that the BJT in Figure 2 is switching between the ON and OFF modes when the Arduino output pin V_A is HIGH (5V) and LOW (0V) respectively. The model for both ON and OFF states are shown in Figure 3. When the BJT turns on, V_{BE} can be modeled as a fixed voltage source between the Base and Emitter terminals with a voltage value of V_{BE0}, which is usually between 0.6V to 0.8V. In ON mode, there is a **Current Controlled Current Source** modeled between the "Collector (C)" and "Emitter (E)", i.e., current at the "Collector (C)" is an amplified version of current at the "Base (B)" (notice that I_B has to flow into the "Base (B)" for the relation $I_C = \beta_F I_B$ to hold.) β_F is called the **Common-Emitter Current Gain**. When the Arduino output pin V_A is HIGH (5V), the voltage across the resistor R_B in Figure 2 is $V_{HIGH} - V_{BE0}$. So we can calculate the current flowing into the "Base (B)", I_B, and the current flowing into the "Collector (C)", I_C, as

$$I_B = \frac{V_{HIGH} - V_{BE0}}{R_B} \tag{1}$$

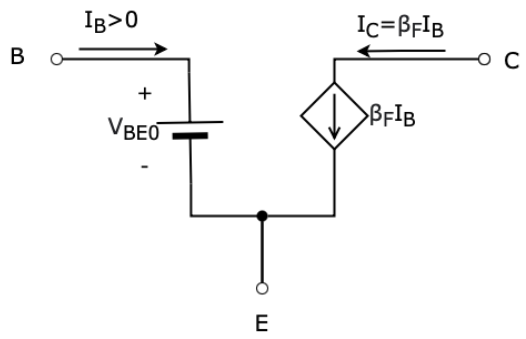
$$I_C = \beta_F I_B = \beta_F \frac{V_{HIGH} - V_{BE0}}{R_B} \tag{2}$$

From equations above, we know that the resistor in the motor controller circuit (Figure 2) is used for defining the current flow through the Arduino output pin, BJT, and motor. Therefore, we should carefully choose the value of the resistor considering the maximum current that BJT and Arduino pin can handle and the current required to drive the

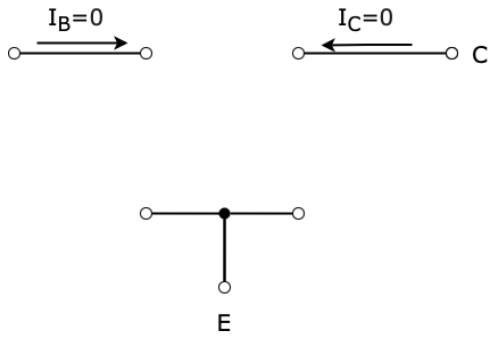
[2] Although the value of the resistor chosen based on testing from various cars, you are free to choose a different value should you want to make your car faster or slower. Though you might want to consider the implications this might have on the controls labs.

motor. As we increase the value of the resistor, the current into the "Base (B)" decreases, which in turn reduces the current flow through the motor, slowing down the motor rotation.

Ignoring the diode in the schematic, the signal from the Arduino switches the BJT on and off: when the Arduino signal is HIGH (5V), the BJT is on, we have 9V-V_{CE} dropped across the motor (V_{CE} is the voltage across the current control current source in Figure 3a) and current can flow through the motor and BJT. When the Arduino signal is LOW (0V), the BJT is off. As shown in Figure 3b, the BJT is open at all terminals. Therefore, no current is allowed to flow into the motor and there is no voltage drop across the motor (0V). Our Arduino PWM wave that only spanned 0V-5V thus has been amplified to a signal that roughly spans 0V-9V (neglecting the V_{CE}) across the motor! Note that because the voltage dropped across the motor is HIGH (9V) when the Arduino PWM is HIGH (5V) and LOW (0V) when the Arduino PWM is LOW (0V), the PWM of the voltage dropped across the motor has the same duty cycle as the Arduino PWM. By changing the duty cycle of the Arduino PWM, we can thus modulate the average voltage across the motor to a voltage between 0V to 9V, and therefore change its speed.



(a) Model of BJT in ON mode (when Arduino output pin is HIGH)



(b) Model of BJT in OFF mode (when Arduino output pin is LOW)

Figure 3: Model of NPN BJT in Different Modes

Moreover, the BJT is capable of handling far more current than the Arduino pins can tolerate, allowing the necessary hundreds of mA to flow through the motor so it can run. The Arduino only needs to supply the small amount of current the BJT needs flowing into its base terminal to function (this is a key difference between the NPN BJT and NMOS) and the BJT will amplify that current to be used by the motor. With this circuit, we have thus supplied the motor with a high-power PWM signal.

Diode

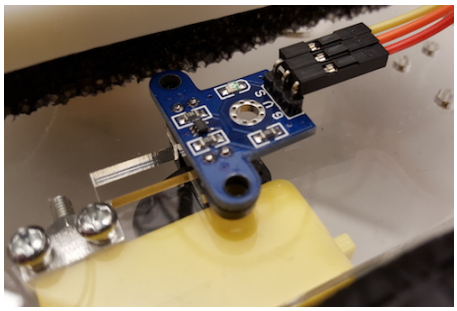
You may be wondering why we have the diode connected across the motor. We add this diode because motors have inductive properties; the current flowing through the motor cannot change instantaneously. When we turn off the BJT, we are cutting off all current from flowing through the BJT. Above, where we ignored the diode, we assumed the motor could instantly go from an ON to OFF state. However, because the motor has inductive properties, if we did not have the diode in the circuit, the current through the motor would have to instantaneously drop from hundreds of mA (ON) to 0A (OFF), causing a huge voltage spike at the Collector of the BJT, which can potentially cause damage. In order to prevent this from happening, we add the diode as a pathway to allow current to flow through it when the BJT is turned off, enabling the current to drop over time instead of instantaneously. Once the current decreases sufficiently, the diode turns off, and the motor is just connected to an open circuit, meaning no current will flow across the motor. This results in no voltage drop across the motor.

We have simplified our analysis and description of the different components we use in the motor circuit drastically so as to remain in scope for this class. If you would like to learn more about these components after doing the lab, please read the extra reading note for this lab for more information.

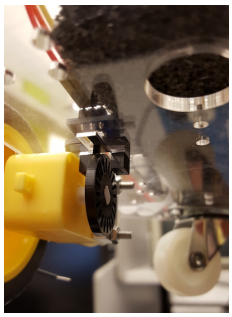
Part 2: Encoders

The encoders we will be using in our labs are called photointerruptors. As a wheel on the car turns, there is an encoder disc with cutouts in it that turns with the wheel. The encoder shines a light between its two "legs", and as the

wheel turns between them, the beam of light is interrupted by the spokes of the encoder disc at a rate proportional to the velocity of the car. The Arduino is therefore able to detect how fast the wheel is turning by looking at the number of times that the encoder records an interruption in the beam of light, which is what we call a "tick", over a specific time interval.



(a) Top view of encoders



(b) Bottom view of encoders

Figure 4: Encoders

From an electrical point of view, the way the Arduino interacts with the encoder is with different voltage levels. When the beam of light is unblocked, the encoder's signal pin (S), which the Arduino takes as input, will output a logic HIGH voltage (roughly 5V). When the beam of light is blocked, the encoder's signal pin outputs a logic LOW voltage (roughly 0V). As the wheel rotates and causes the encoder disc to repeatedly block and unblock the beam, the encoder will repeatedly output alternating logic HIGH and LOW signals, generating a square wave. By counting the raw number of rising edges (logic LOW to logic HIGH) and falling edges (logic HIGH to logic LOW) that arrive during a specified period of time, the Arduino can determine how far the car has driven and how fast it's moving, respectively. For this lab, we have given you a .ino file for you to use to test your encoders with. It cycles through 4 tests: neither wheels move, left wheel moves, right wheel moves, and both wheels move. Make sure that the readings match up with what you'd expect (i.e. should not record ticks when the wheel is not moving).

Part 3: Regulator Circuits

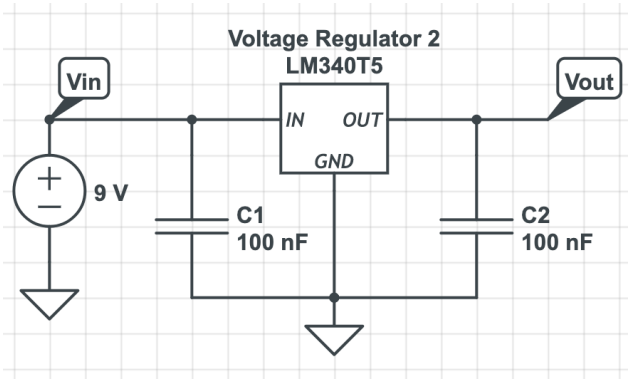
S1XT33N, like any other electronics, requires power to run. If we were only using it at the lab stations, we could just hook it up to the DC power supply. However, because we want to let S1XT33N roam freely on the ground without confinement to a lab station, we need to power it with a more portable source: batteries. However, we only have access to 9V batteries, which cannot directly supply the 5V we need for powering the rest of our circuitry. In most electrical systems, you only have access to one or two main supply voltages (such as 9V from a 9V battery) that often times are not what you need for your circuits. This is where the voltage regulators come in.

In order to generate specific voltages, you need to build a voltage regulator to "regulate" the supply voltage to what you require. The inner workings of these regulators is beyond the scope of the class, so we will not discuss them in detail, nor build our own. Although the Arduino has an internal 5V regulator to produce 5V outputs, we typically do not want to use that to power the rest of our circuit because of the Arduino's limited output current.

5V Regulator Circuit

The LM340T5 is really just meant for 5V regulation (hence the 5 in LM340T5), so we use it to regulate the 9V input down to a 5V output. Adding "decoupling" capacitors to the input and output is useful for removing all sorts of noise and interference from the 9V supply and the 5V output to make them both more stable.

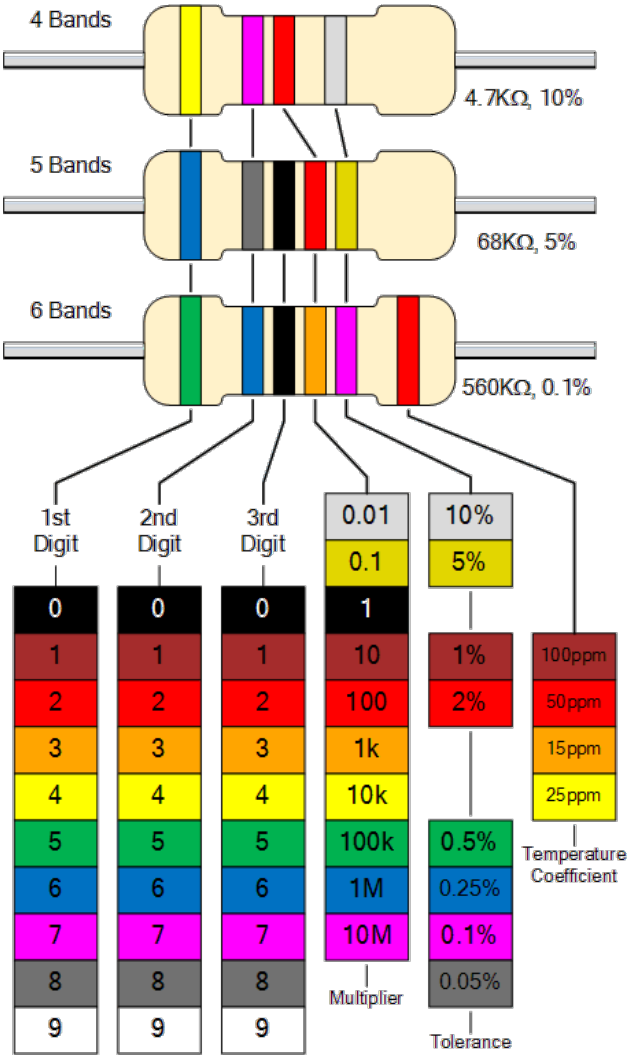
The pinout is reproduced below for your convenience. The image is taken from a front angle where the metal tab that sticks out is at the back of the regulator. Note the difference between the 5V regulator and 3.3V regulator: the 3.3V regulator has 2 rectangular notches/cutouts in the metal tab, while the 5V regulator does not. The 5V regulator also has a thicker metal tab than the 3.3V regulator. Please verify that you haven't used a legacy 3.3 V regulator!



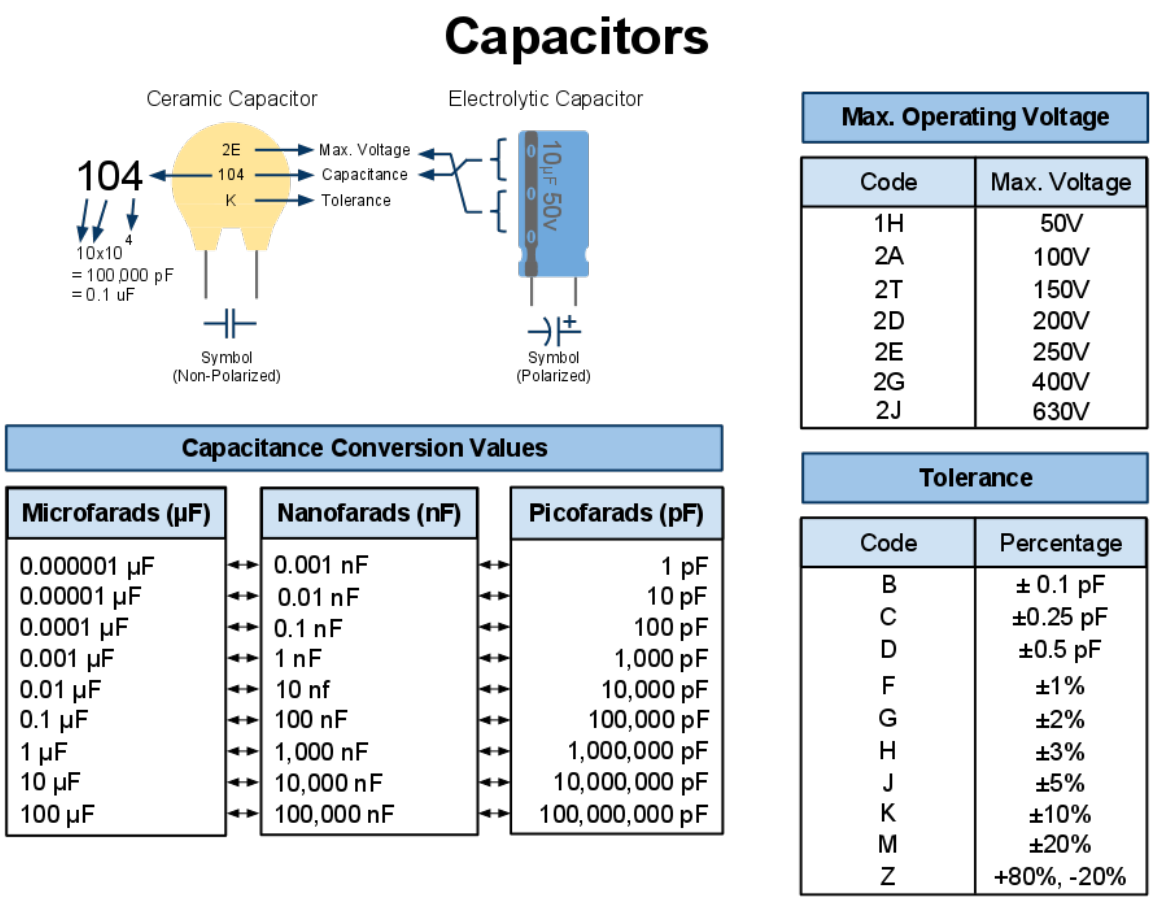
INPUT	1
GND	2
OUTPUT	3

Written by Mia Mirkovic (2019)
Edited by Steven Lu, Yi-Hsuan Shih (2021). Version 2.0, 2021
Edited by Steven Lu (2022). Version 3.0, 2022
Edited by Megan Zeng (2022, 2023). Version 4.0, 2022
Edited by Junha Kim, Ryan Ma (2023). Version 4.1, 2023
Edited by Venkata Alapati (2024). Version 4.2, 2024

Appendix A: Resistor Codes



Appendix B: Capacitor Codes



Capacitance Conversion Values

Microfarads (µF)	Nanofarads (nF)	Picofarads (pF)
0.000001 µF	0.001 nF	1 pF
0.00001 µF	0.01 nF	10 pF
0.0001 µF	0.1 nF	100 pF
0.001 µF	1 nF	1,000 pF
0.01 µF	10 nf	10,000 pF
0.1 µF	100 nF	100,000 pF
1 µF	1,000 nF	1,000,000 pF
10 µF	10,000 nF	10,000,000 pF
100 µF	100,000 nF	100,000,000 pF

Max. Operating Voltage

Code	Max. Voltage
1H	50V
2A	100V
2T	150V
2D	200V
2E	250V
2G	400V
2J	630V

Tolerance

Code	Percentage
B	± 0.1 pF
C	±0.25 pF
D	±0.5 pF
F	±1%
G	±2%
H	±3%
J	±5%
K	±10%
M	±20%
Z	+80%, -20%

References

Original Project Part 1 notebook written by Nathaniel Mailoa and Emily Naviasky (2016).

Horowitz, P. and Hill, W. (2015). *The Art of Electronics*. 3rd ed. Cambridge: Cambridge University Press, ch 4.

Horowitz, P. and Hill, W. (2015). *The Art of Electronics*. 3rd ed. Cambridge: Cambridge University Press, ch 2.

Sedra, A. S. and Smith, K. C. (2015). *Microelectronic Devices and Circuits*. 7th ed. Oxford: Oxford University Press, ch 3.

Sedra, A. S. and Smith, K. C. (2015). *Microelectronic Devices and Circuits*. 7th ed. Oxford: Oxford University Press, ch 6.

https://www.electronics-tutorials.ws/wp-content/uploads/2018/05/resistor-res5.gif

https://www.bragitoff.com/wp-content/uploads/2015/09/CapacitorsCheatSheet.png