Arduino Control of Tetrix Prizm Robotics

Motors and Servos

Introduction to Robotics and Engineering

Marist School

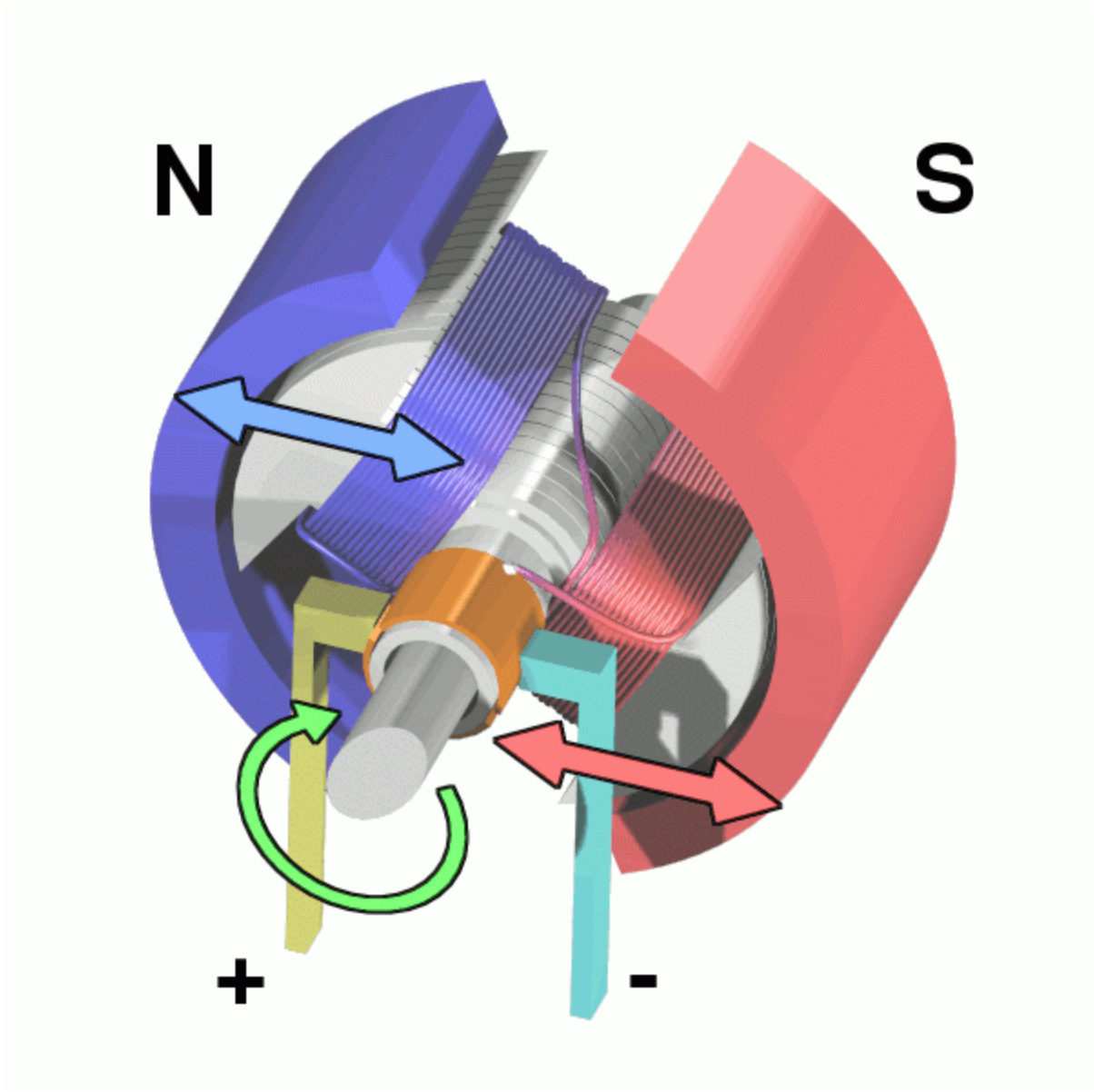
Motor or Servo?

Motor

- Faster revolution but less Power
- Tetrix 12 Volt DC motors have a transmission that slows speed and increases torque
- Speed controlled by varying voltage (Pulse Width Modulation)
- Controlled with Arduino Motor Shield REV3

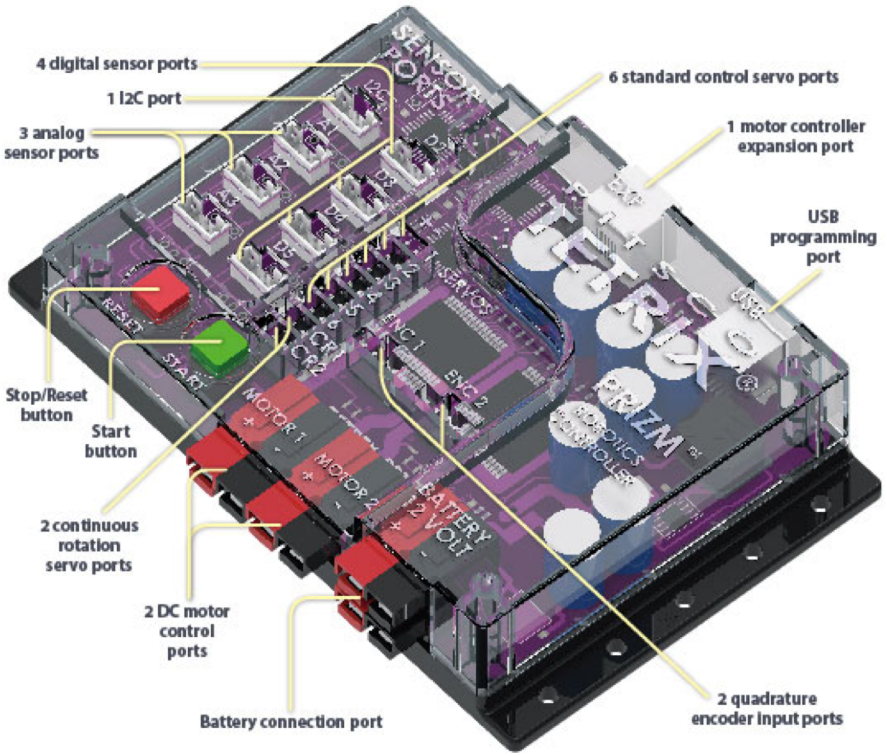
Servo

- Motor combined with encoder.
- More powerful, but slower
- Two types
  - Standard: 180 Degree Rotation
  - Continuous Rotation
- Three leads:
  - Signal (from pin)
  - 5 Volt
  - Ground


N
S
+
-

Tetrix Prizm Microcontroller

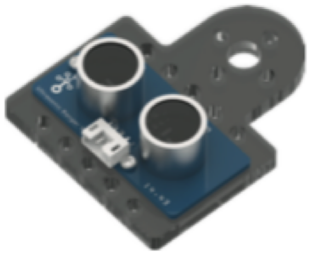
- Left Drive plugged into Motor 1
- Right Drive plugged into Motor 2
- Sonar plugged into D2
- Light plugged into D3
- Starter code available at:



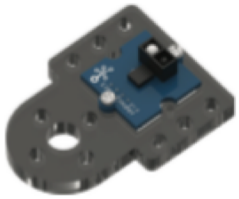
Tetrix Prizm Components

Sensor:

A device that detects and responds to surrounding environmental factors



Ultrasonic Sensor:
Enables a robot to measure distance to an object and respond to movement



Line Finder Sensor:
Enables a robot to follow a black line on a white background or vice versa

Motor:

A machine that produces motion or power for completing work

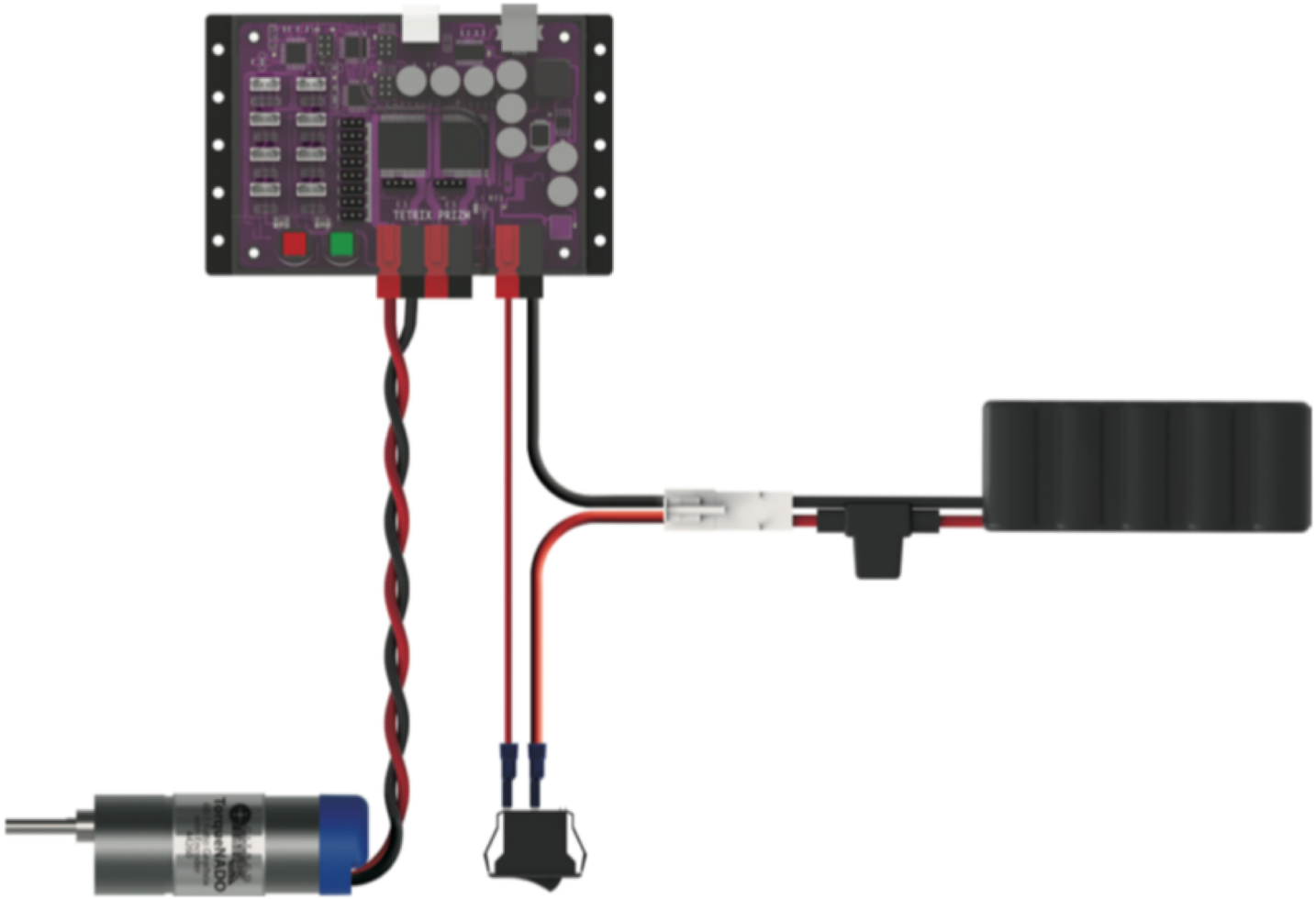
TorqueNADO™ Motor:
12-volt DC motor that features 100 rpm and 700 oz-in. of torque

Powerpole Motor Cable:
Connects the DC motor to the power source

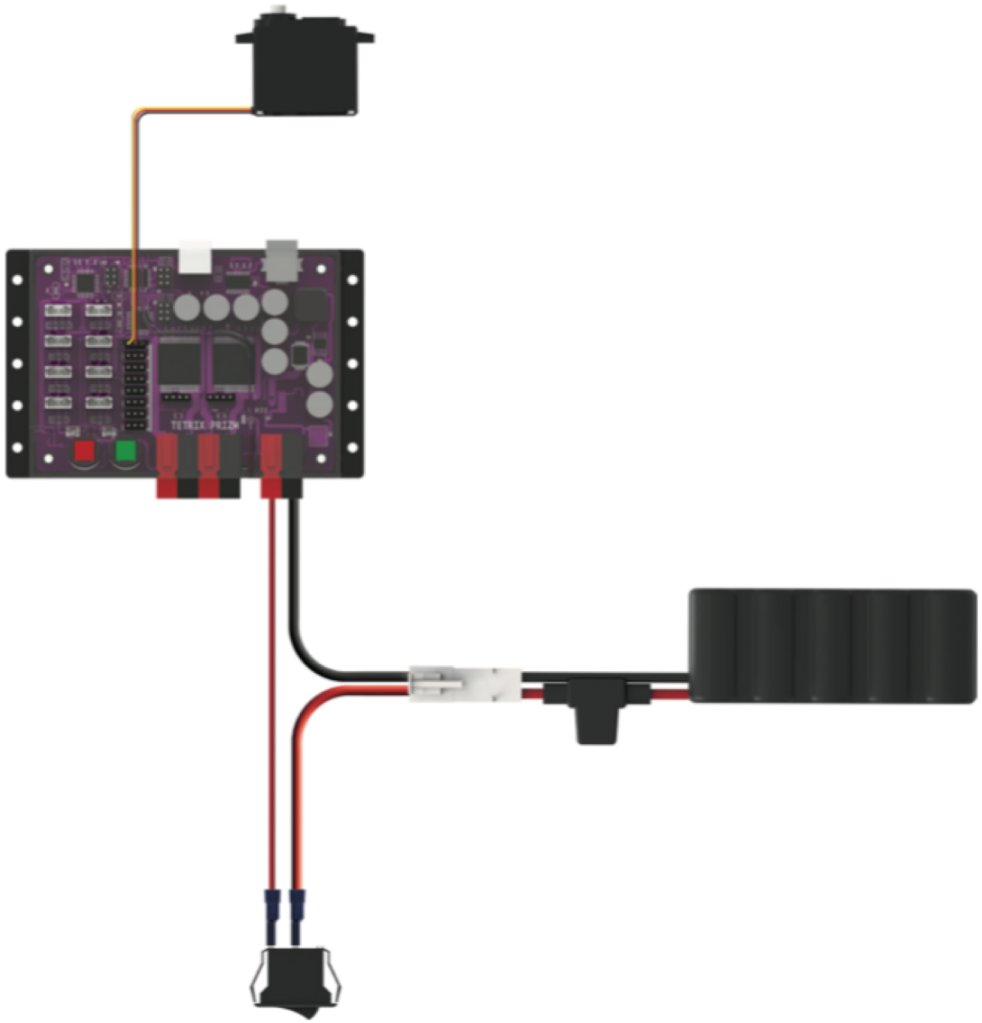
Standard Servo Motor:
Allows for exact positioning within a 180-degree range of motion

Continuous Rotation Servo Motor:
Allows for continuous direction of movement, both forward and backward

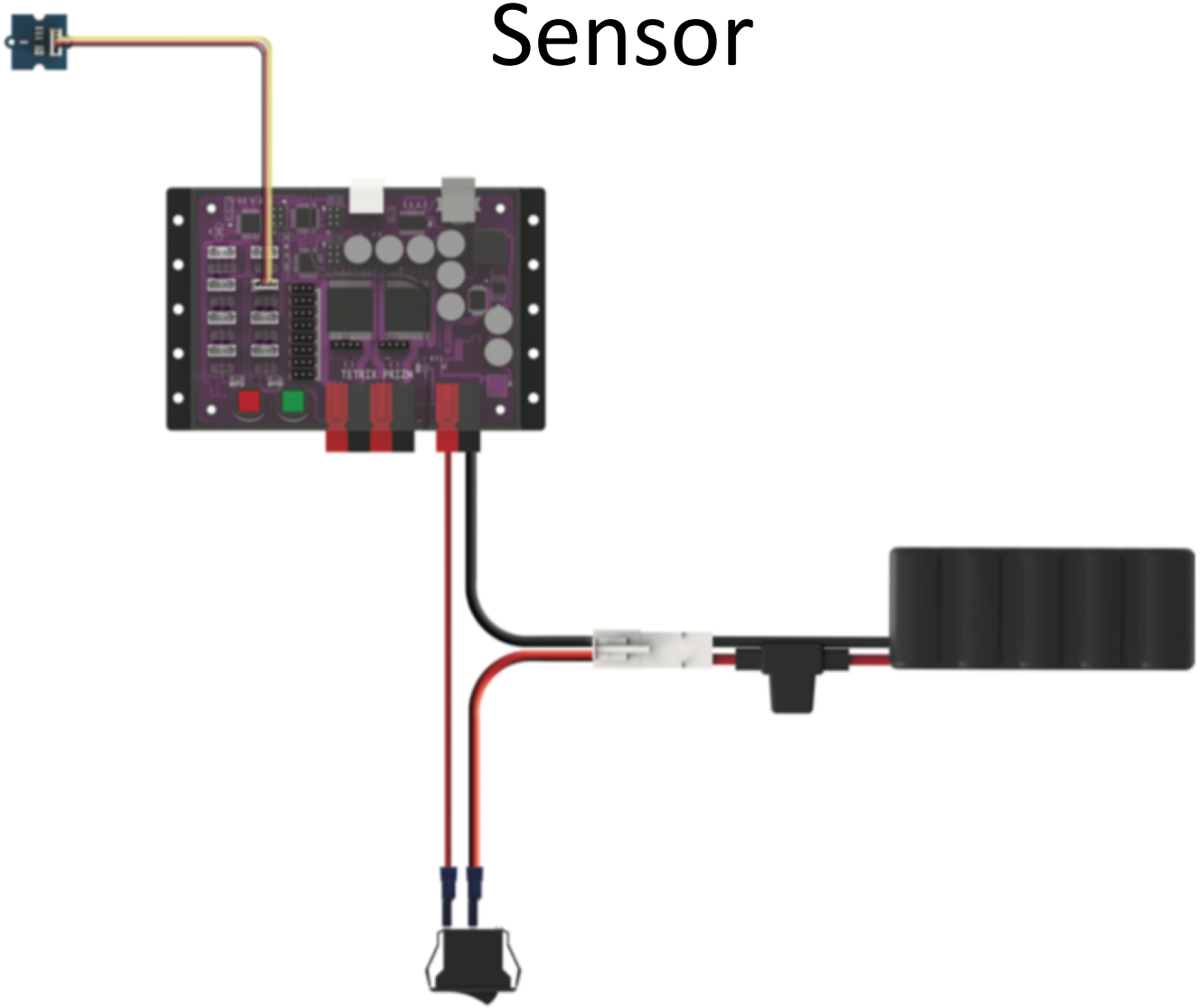
Setup for Prizm, Switch, Battery, and Motor



Setup for Prizm, Switch, Battery, and Servo



Setup for Prizm, Switch, Battery, and Sensor



Key Functions

- `void setup()`
  - Must be defined
  - Runs one time
  - Acts like a constructor in Java
- `void loop()`
  - Must be defined
  - Runs continuously (In a loop . . .)
  - Like the act() method in Greenfoot or Gridworld
- `delay (t)`
  - Will pause code for t milliseconds

Serial Functions

- `Serial.begin(9600)`
  - Initiates Serial Communication at 9600 Baud
  - Called in setup() function
- `Serial.println(<string>)`
  - Prints string to console
  - Example: `Serial.println("Hello");`
- `Serial.println(<number>)`
  - Prints number to console

Example Code

```
// Setup for Serial Printing
// Mr. Michaud

// Setup Function - Runs 1 Time
void setup() {
  Serial.begin(9600);  // Start communicating at 9600 Baud
  Serial.println("Hello World");
  Serial.println("I am the Arduino Uno");
} // end setup

// Required Function:  void loop()
void loop() {
  // No Code here today
} // end loop
```

Data Types and Variables

bool	0	0 or 1 || true or false
byte	0000 0000	Whole number: -128 to 127
char	0000 0000	Can be a character value (ie "a")
float	0000 0000 0000 0000 0000 0000 0000 0000	decimal point number
long	0000 0000 0000 0000 0000 0000 0000 0000	Whole number: -2,147,483,648 to 2,147,483,647
int	0000 0000 0000 0000	Whole number: -32,768 to 32,767
ubyte	0000 0000	Whole number between 0 and 255

Assigning Variables:

```
int  power = 100;  // Creates an integer equal to 100

bool touched = false;

float batteryPower = 6.78;
long clockCycles = 84234;
```

Operators

=	Assigns a value
==	Compares - "is equal to"
>	Greater than
<	Less than
!	Not
||	Or
&&	And
*	Multiply
/	Divide
++	Increase by 1
+	Addition
-	Subtraction
%	Modulo

Declaring Variables and Arrays

Variables:

```
int myAge = 14; // Integer
float angle = 2*PI; // Float
String name = "Mr. Michaud";
```

Arrays:

```
int pins [] = {3, 4, 5, 6};
String gospels [] = {"Matthew", "Mark", "Luke", "John"};
```

For Loop

- Repeats section of code while counting up or down with an index variable
- Example

```
for (int i = 0; i < 10; i++) {
    Serial.println(i);
}
```

Returns:
0
1
2
3
4
5
6
7
8
9

for (int i = 0; i < 10; i++) {}

- i++ means "i = i + 1"
- int i means "integer i"

- for (int i = 0; i < 10; i++) means "For index variable i starting at 0, while i is less than 10, count be 1."

Combined For Loop and Array

```
// Array Example

String gospels [] = {"Matthew", "Mark", "Luke", "John"};

// Setup
void setup() {
  Serial.begin(9600);
  // For Loop
  for (int i = 0; i < 4; i++) {
    Serial.println(gospels[i]);
  }
}

void loop(){
  // No Code here
}
```

Conditional Statements

- ‘if statement’: Checks if a given statement or expression is true and then executes a section of code

```
if (score > 9) {
      Serial.println(“You Win”);
}
```

While Loop

- Executes a Segment of Code while a Condition is True

```
// Initialize integer score as 0
int score = 0;

// Loop while score is less than 20
while (score < 20) {
    // Print the Score
    System.out.println("Your Score is: " + String.valueOf(score));
    score++; // Increase Score by 1
} // end while

System.out.println("All Done!");
```

TETRIX PRIZM Arduino Library Functions Chart

Description	Function	Coding Example
Prizm Begin Is called in the Arduino code setup() loop. Initializes the PRIZM controller.	**PrizmBegin();** Data Type: None	**PrizmBegin**(); *Reset and initialize PRIZM controller.*
Prizm End When called, immediately terminates a program and resets the PRIZM controller.	**PrizmEnd();** Data Type: None	**PrizmEnd**(); *Terminate a PRIZM program and reset controller.*
Set Red LED Sets the PRIZM red indicator LED to on or off.	**setRedLED**(state); Data Type: *state* = integer Data Range: *state* = 1 or 0 or *state* = HIGH or LOW	**setRedLED**(HIGH); *or* **setRedLED**(1); *Turn red LED on.* **setRedLED**(LOW); *or* **setRedLED**(0); *Turn red LED off.*
Set Green LED Sets the PRIZM green indicator LED to on or off.	**setGreenLED**(state); Data Type: *state* = integer Data Range: *state* = 1 or 0 or *state* = HIGH or LOW	**setGreenLED**(HIGH); *or* **setGreenLED**(1); *Turn green LED on.* **setGreenLED**(LOW); *or* **setGreenLED**(0); *Turn green LED off.*
Set DC Motor Power Sets the power level and direction of a TETRIX DC motor connected to the PRIZM DC motor ports. Power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	**setMotorPower**(motor#, power); Data Type: *motor#* = integer *power* = integer Data Range: *motor#* = 1 or 2 *power* = -100 to 100 or *power* = 125 (brake mode)	**setMotorPower**(1, 50); *Spin Motor 1 clockwise at 50% power.* **setMotorPower**(2, -50%); *Spin Motor 2 counterclockwise at 50% power.* **setMotorPower**(1, 0); *Turn off Motor 1 in coast mode.* **setMotorPower**(2, 125); *Turn off Motor 2 in brake mode.*
Set DC Motor Powers Simultaneously sets the power level and direction of **both** TETRIX DC motors connected to the PRIZM motor ports. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The power level range is 0 to 100. Direction is set by the sign (+/-) of the power level. Power level 0 = stop in coast mode. Power level 125 = stop in brake mode.	**setMotorPowers**(power1, power2); Data Type: *power1* = integer *power2* = integer Data Range: *power1* = -100 to 100 *power2* = -100 to 100 or *power1* = 125 (brake mode) *power2* = 125 (brake mode)	**setMotorPowers**(50, 50); *Spin Motor 1 and Motor 2 clockwise at 50% power.* **setMotorPowers**(-50, 50); *Spin Motor 1 counterclockwise and Motor 2 clockwise at 50% power.* **setMotorPowers**(0, 0); *Turn off Motor 1 and Motor 2 in coast mode.* **setMotorPowers**(125, 125); *Turn off Motor 1 and Motor 2 in brake mode.*
Set DC Motor Speed Uses velocity PID control to set the constant speed of a TETRIX DC motor with a TETRIX motor encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the *speed* parameter controls direction of rotation.	**setMotorSpeed**(motor#, speed); Data Type: *motor#* = integer *speed* = integer Data Range: *motor#* = 1 or 2 *speed* = -720 to 720	**setMotorSpeed**(1, 360); *Spin Motor 1 clockwise at a constant speed of 360 DPS.* **setMotorSpeed**(1, -360); *Spin Motor 1 counterclockwise at a constant speed of 360 DPS.*

Description	Function	Coding Example
Set DC Motor Speeds Uses velocity PID control to simultaneously set the constant speeds of **both** TETRIX DC motor channels with TETRIX motor encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The sign (+/-) of the speed parameter controls direction of rotation.	**setMotorSpeeds**(speed1, speed2); Data Type: *speed1* = integer *speed2* = integer Data Range: *speed1* = -720 to 720 *speed2* = -720 to 720	**setMotorSpeeds**(360, 360); *Spin Motor 1 and Motor 2 clockwise at a constant speed of 360 DPS.* **setMotorSpeeds**(360, -360); *Spin Motor 1 clockwise and Motor 2 counterclockwise at a constant speed of 360 DPS.* **setMotorSpeeds**(360, -180); *Spin Motor 1 clockwise and Motor 2 counterclockwise at a constant speed of 180 DPS.*
Set DC Motor Target Implements velocity and positional PID control to set the constant speed and the encoder count target holding position of a TETRIX DC motor with a TETRIX encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	**setMotorTarget**(motor#, speed, target); Data Type: *motor#* = integer *speed* = integer *target* = long Data Range: *motor#* = 1 or 2 *speed* = 0 to 720 *target* = -2147483648 to 2147483647	**setMotorTarget**(1, 360, 1440); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 count equals 1,440. When at encoder target count, hold position in a servo-like mode.* **setMotorTarget**(2, 180, -1440); *Spin Motor 2 at a constant speed of 180 DPS until encoder 2 count equals -1,440 (1 revolution). When at encoder target count, hold position in a servo-like mode.*
Set DC Motor Targets Implements velocity and positional PID control to simultaneously set the constant speeds and the encoder count target holding positions of **both** TETRIX DC motor channels each with TETRIX encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder count target position is a signed long integer from -2,147,483,648 to 2,147,483,647. Each encoder count = 1/4-degree resolution.	**setMotorTargets**(speed1, target1, speed2, target2); Data Type: *speed1* = integer *target1* = long *speed2* = integer *target2* = long Data Range: *speed1* = 0 to 720 *target1* = -2147483648 to 2147483647 *speed2* = 0 to 720 *target2* = -2147483648 to 2147483647	**setMotorTargets**(360, 1440, 360, 1440); *Spin Motor 1 and Motor 2 at a constant speed of 360 DPS until each motor encoder count equals 1,440. When a motor reaches its encoder target count, hold position in a servo-like mode.* **setMotorTargets**(360, 1440, 180, 2880); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 count equals 1,440. Spin Motor 2 at a constant speed of 180 DPS until encoder 2 equals 2,880. Each motor will hold its position in a servo-like mode when it reaches the encoder target.* ***Note:*** *One encoder count equals 1/4-degree resolution. For example, 1 motor revolution equals 1,440 encoder counts (1,440 / 4 = 360).*
Set Motor Degree Implements velocity and positional PID control to set the constant speed and the degree target holding position of a TETRIX DC motor with a TETRIX encoder connected. The *speed* parameter range is 0 to 720 degrees per second (DPS). The encoder degrees target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	**setMotorDegree**(motor#, speed, degrees); Data Type: *motor#* = integer *speed* = integer *degrees* = long Data Range: *motor#* = 1 or 2 *speed* = 0 to 720 *degrees* = -536870912 to 536870911	**setMotorDegree**(1, 180, 360); *Spin Motor 1 at a constant speed of 180 DPS until encoder 1 degree count equals 360. When at encoder target degree count, hold position in a servo-like mode.* **setMotorDegree**(2, 90, 180); *Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree count equals 180. When at encoder target degree count, hold position in a servo-like mode.*

Description	Function	Coding Example
Set Motor Degrees Implements velocity and positional PID control to set the constant speeds and the degree target holding positions of both TETRIX DC motor channels with TETRIX encoders connected. Both PRIZM Motor 1 and Motor 2 channel parameters are set with a single statement. The speed parameter range is 0 to 720 degrees per second (DPS). The encoder degree target position is a signed long integer from -536,870,912 to 536,870,911 with a 1-degree resolution.	**setMotorDegrees**(speed1, degrees1, speed2, degrees2); Data Type: *speed1* = integer *degrees1* = long *speed2* = integer *degrees2* = long Data Range: *speed1* = 0 to 720 *degrees1* = -536870912 to 536870911 *speed2* = 0 to 720 *degrees2* = -536870912 to 536870911	**setMotorDegrees**(180, 360, 180, 360); *Spin Motor 1 and Motor 2 at a constant speed of 180 DPS until each motor encoder degree count equals 360. When a motor reaches its degree target count, hold position in a servo-like mode.* **setMotorDegrees**(360, 720, 90, 360); *Spin Motor 1 at a constant speed of 360 DPS until encoder 1 degree count equals 720. Spin Motor 2 at a constant speed of 90 DPS until encoder 2 degree equals 360. Each motor will hold its position in a servo-like mode when it reaches the encoder target.*
Set Motor Direction Invert Inverts the forward/reverse direction mapping of a DC motor channel. This function is intended to harmonize the forward and reverse directions for motors on opposite sides of a skid-steer robot chassis. Inverting one motor channel can make coding opposite-facing DC motors working in tandem more intuitive. An *invert* parameter of 1 = invert. An *invert* parameter of 0 = no invert. The default is no invert.	**setMotorInvert**(motor#, invert); Data Type: *motor#* = integer *invert* = integer Data Range: *motor#* = 1 or 2 *invert* = 0 or 1	**setMotorInvert**(1, 1); *Invert the spin direction mapping of Motor 1.* **setMotorInvert**(2, 1); *Invert the spin direction mapping of Motor 2.* **setMotorInvert**(1, 0); *Do not invert the spin direction mapping of Motor 1.* **setMotorInvert**(2, 0); *Do not invert the spin direction mapping of Motor 2.* ***Note:*** *Non-inverting is the default on PRIZM power-up or reset.*
Read Motor Busy Status The busy flag can be read to check on the status of a DC motor that is operating in positional PID mode. The motor busy status will return "1" if it is moving toward a positional target (degrees or encoder count). When it has reached its target and is in hold mode, the busy status will return "0."	**readMotorBusy**(motor#); Data Type: *motor#* = integer Data Range: *motor#* = 1 or 2 Data Type Returned: *value* = integer	**readMotorBusy**(1); *Return the busy status of Motor 1.* **readMotorBusy**(2); *Return the busy status of Motor 2.*
Read DC Motor Current Reads the DC motor current of each TETRIX DC motor attached to PRIZM Motor 1 and Motor 2 ports. The integer value that is returned is motor load current in milliamps.	**readMotorCurrent**(motor#); Data Type: *motor#* = integer Data Range: *motor#* = 1 or 2 Data Type Returned: *value* = integer	**readMotorCurrent**(1); *Read the motor load current of the PRIZM Motor 1 channel.* **readMotorCurrent**(2); *Read the motor load current of the PRIZM Motor 2 channel.* *Example: 1500 = 1.5 amps*

Description	Function	Coding Example
Read Encoder Count Reads the encoder count value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. Each 1/4 degree equals one pulse, or count, or 1 degree of rotation equals 4 encoder counts. The current count can be read to determine a motor's shaft position. The total count accumulation can range from -2,147,483,648 to 2,147,483,647. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	**readEncoderCount**(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1 or 2 Data Type Returned: *value* = long	**readEncoderCount**(1); *Read the current count value of encoder 1 (ENC 1 port).* **readEncoderCount**(2); *Read the current count value of encoder 2 (ENC 2 port).*
Read Encoder Degrees Reads the encoder degree value. The PRIZM controller uses encoder pulse data to implement PID control of a TETRIX DC motor connected to the motor ports. The PRIZM controller counts the number of pulses produced over a set time period to accurately control velocity and position. This function is similar to the encoder count function, but instead of returning the raw encoder count value, it returns the motor shaft position in degrees. The total degree count accumulation can range from -536,870,912 to 536,870,911. A clockwise rotation adds to the count value, while a counterclockwise rotation subtracts from the count value. The encoder values are set to 0 at power-up and reset.	**readEncoderDegrees**(enc#); Data Type: *enc#* = integer Data Range: *enc#* = 1 or 2 Data Type Returned: *value* = long	**readEncoderDegrees**(1); *Read the current degree count value of encoder 1 (ENC 1 port).* **readEncoderDegrees**(2); *Read the current degree count value of encoder 2 (ENC 2 port).*
Reset Each Encoder Resets the encoder count accumulator to 0.	**resetEncoder**(enc#); Data Type: *enc#* = integer Data Range: enc# = 1 or 2	**resetEncoder**(1); *Reset the encoder 1 count to 0.* **resetEncoder**(2); *Reset the encoder 2 count to 0.*
Reset Both Encoders (1 and 2) Resets the encoder 1 and encoder 2 count accumulators to 0.	**resetEncoders**(); Data Type: None	**resetEncoders**(); *Reset the encoder 1 count to 0 and encoder 2 count to 0.*

Description	Function	Coding Example
Read Line Sensor Output Reads the digital output of the Line Finder Sensor connected to a PRIZM sensor port. The value read is "0" when reflected light is received (detecting a light-colored surface) and "1" when light is not received (detecting a dark-colored surface, such as a line).	**readLineSensor**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (0 or 1)	**readLineSensor**(2); *Read the digital value of a Line Finder Sensor on digital sensor port D2.* ***Note:*** *The Line Finder Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input. See technical section on sensor ports for a more detailed explanation of sensor ports and pinouts.*
Read Ultrasonic Sensor in Centimeters Reads the distance in centimeters of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 3 to 400 centimeters. The value read is an integer.	**readSonicSensorCM**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (3 to 400) Min and max might slightly vary.	**readSonicSensorCM**(3); *Read the distance in centimeters of an object placed in front of the Ultrasonic Sensor connected to digital sensor port D3.* ***Note:*** *The Ultrasonic Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input. See technical section on sensor ports for a more detailed explanation of sensor ports and pinouts.*
Read Ultrasonic Sensor in Inches Reads the distance in inches of an object placed in front of the Ultrasonic Sensor. The sensor is modulated at 42 kHz and has a range of 2 to 150 inches. The value read is an integer.	**readSonicSensorIN**(port#); Data Type: *port#* = integer Data Range: *port#* (See note in adjacent column.) Data Type Returned: *value* = integer (2 to 150) Min and max might slightly vary.	**readSonicSensorIN**(4); *Read the distance in inches of an object placed in front of the Ultrasonic Sensor connected to digital sensor port D4.* ***Note:*** *The Ultrasonic Sensor can be connected to any digital port D2-D5, or analog ports A1-A3 configured as digital input.*
Read Battery Pack Voltage Reads the voltage of the TETRIX battery pack powering the PRIZM controller. The value read is an integer.	**readBatteryVoltage**(); Data Type: None Data Type Returned: *value* = integer	**readBatteryVoltage**(); *Read the voltage of the TETRIX battery pack powering the PRIZM controller.* *Example: A value of 918 equals 9.18 volts.*
Read Start Button State Reads the state of the green PRIZM Start button. A returned value of "1" indicates a pressed state. A returned value of "0" indicates a not-pressed state.	**readStartButton**(); Data Type: None Data Type Returned: *value* = integer (0 or 1)	**readStartButton**(); *Read the Start button. A value of 1 means button is pressed. A value of 0 means button is not pressed.*

Description	Function	Coding Example
Set Speed of a Servo Motor Sets the speed of a servo motor connected to a PRIZM servo port 1-6. The *speed* parameter can be 0 to 100%. The servo motor channel parameter can be any number 1 to 6. If not specified, the speed of a servo defaults to 100 (maximum speed). When a servo speed has been set, it will always move at the set speed until changed. Unless we are changing speeds, it will need to be called only once at the beginning of a program.	**setServoSpeed**(servo#, speed); Data Type: *servo#* = integer *speed* = integer Data Range: *servo#* = 1 to 6 *speed* = 0 to 100	**setServoSpeed**(1, 25); *Set the speed of servo channel 1 to 25%.* **setServoSpeed**(2, 50); *Set the speed of servo channel 2 to 50%.*
Set Speeds of All Servo Motors Sets the speeds of all six servo channels simultaneously with a single command. All six speeds are in sequential order and can be 0 to 100%. All six servo speeds may be the same or different.	**setServoSpeeds**(speed1, speed2, speed3, speed4, speed5, speed6); Data Type: *speed1-speed6* = integer Data Range: *speed1-speed6* = 0 to 100	**setServoSpeeds**(25, 25, 25, 25, 25, 25); *Set the speeds of all six servo channels to 25%.* **setServoSpeeds**(25, 35, 45, 55, 65, 75); *Servo 1 speed = 25%* *Servo 2 speed = 35%* *Servo 3 speed = 45%* *Servo 4 speed = 55%* *Servo 5 speed = 65%* *Servo 6 speed = 75%*
Set Position of a Servo Motor Sets the angular position of a servo motor connected to a PRIZM servo motor port 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	**setServoPosition**(servo#, position); Data Type: *servo#* = integer *position* = integer Data Range: *servo#* = 1 to 6 *position* = 0 to 180	**setServoPosition**(1, 90); *Set the angular position of Servo Motor 1 to 90 degrees.* **setServoPosition**(2, 130); *Set the angular position of Servo Motor 2 to 130 degrees.*
Set Positions of All Servo Motors Sets the angular positions of all six servo motors connected to the PRIZM servo motor ports 1-6. The *position* parameter can be any value between 0 and 180 degrees. Any value outside this range is ignored. Not all servos are the same, so be careful when operating a servo motor at the extreme ranges. Listen closely; if a servo is buzzing, it is pressing against its mechanical stop, which might damage the motor. If this happens, limit the range to values slightly greater than 0 and slightly less than 180 to avoid damage to the servo motor.	**setServoPositions**(position1, position2, position3, position4, position5, position6); Data Type: *position1-position6* = integer Data Range: *position1-position6* = 0 to 180	**setServoPositions**(90, 90, 90, 90, 90, 90); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6 to 90 degrees.* **setServoPositions**(10, 20, 30, 40, 50, 60); *Set the angular positions of all six servo motors connected to PRIZM servo ports 1-6.* *Servo 1 position = 10 degrees* *Servo 2 position = 20 degrees* *Servo 3 position = 30 degrees* *Servo 4 position = 40 degrees* *Servo 5 position = 50 degrees* *Servo 6 position = 60 degrees*

Description	Function	Coding Example
Read a Servo Position Reads the most recent commanded position of a servo motor connected to PRIZM servo ports 1-6. The value returned will be 0-180.	**readServoPosition**(servo#); Data Type: *servo#* = integer Data Range: *servo#* = 1 to 6 Data Type Returned: *value* = integer (0 to 180)	**readServoPosition**(1); *Read the most recent commanded position of Servo 1.* **readServoPosition**(2); *Read the most recent commanded position of Servo 2.*
Set Continuous Rotation (CR) State Sets the on/off and direction state of a CR servo connected to PRIZM CR1 and CR2 ports. A *state* parameter of -100 equals spin counterclockwise. A *state* parameter of 100 equals spin clockwise. A *state* parameter of 0 is off or stop. *CRservo#* parameter can be 1 or 2 commanding either CR1 or CR2 servo ports.	**setCRServoState**(CRservo#, state); Data Type: *CRservo#* = integer *state* = integer Data Range: *CRservo#* = 1 or 2 *state* = -100, 0, 100	**setCRServoState**(1, 100); *Spin CR1 servo continuously clockwise.* **setCRServoState**(1, 0); *Stop CR1 servo.* **setCRServoState**(1, -100); *Spin CR1 servo continuously counterclockwise.*

Complete Programming Guide for Tetrix Prizm Arduino

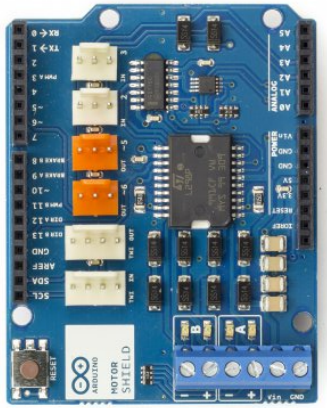
- http://nebomusic.net/ftc/tetrix-prizm-programming-guide-44716.pdf

Other Arduino References

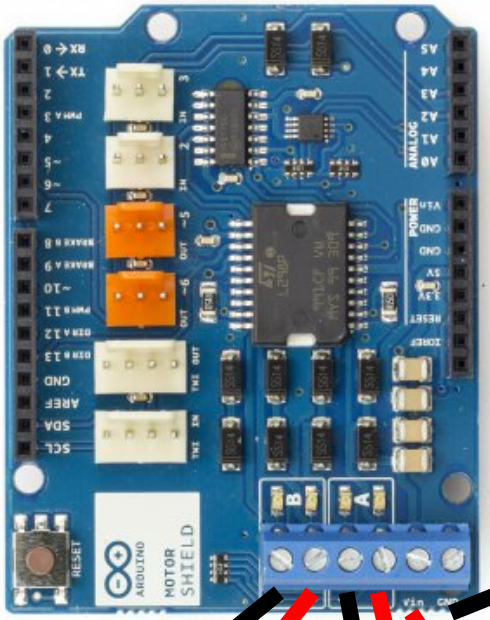
- Java Key Vocabulary: (http://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html)
- Arduino IDO Reference: https://www.arduino.cc/reference/en/
- Another Good Reference: http://processing.org/reference/

REV3 Motor Control and Tetrix

- Left Drive is plugged into MotorA
- Right Drive is plugged into MotorB
- Starter Code Available at:
  - http://nebomusic.net/ftc/Arduino_FTC_Starter.txt
- Motor Control
  - Speed (Converts PWM signals)
  - Direction (HIGH, LOW)
  - Brake On or Off (HIGH, LOW)



Arduino REV3 Motor Controller



Switch

Motor B (Right)

Motor A (Left)

Red wires are positive (current)

Black are negative (ground)

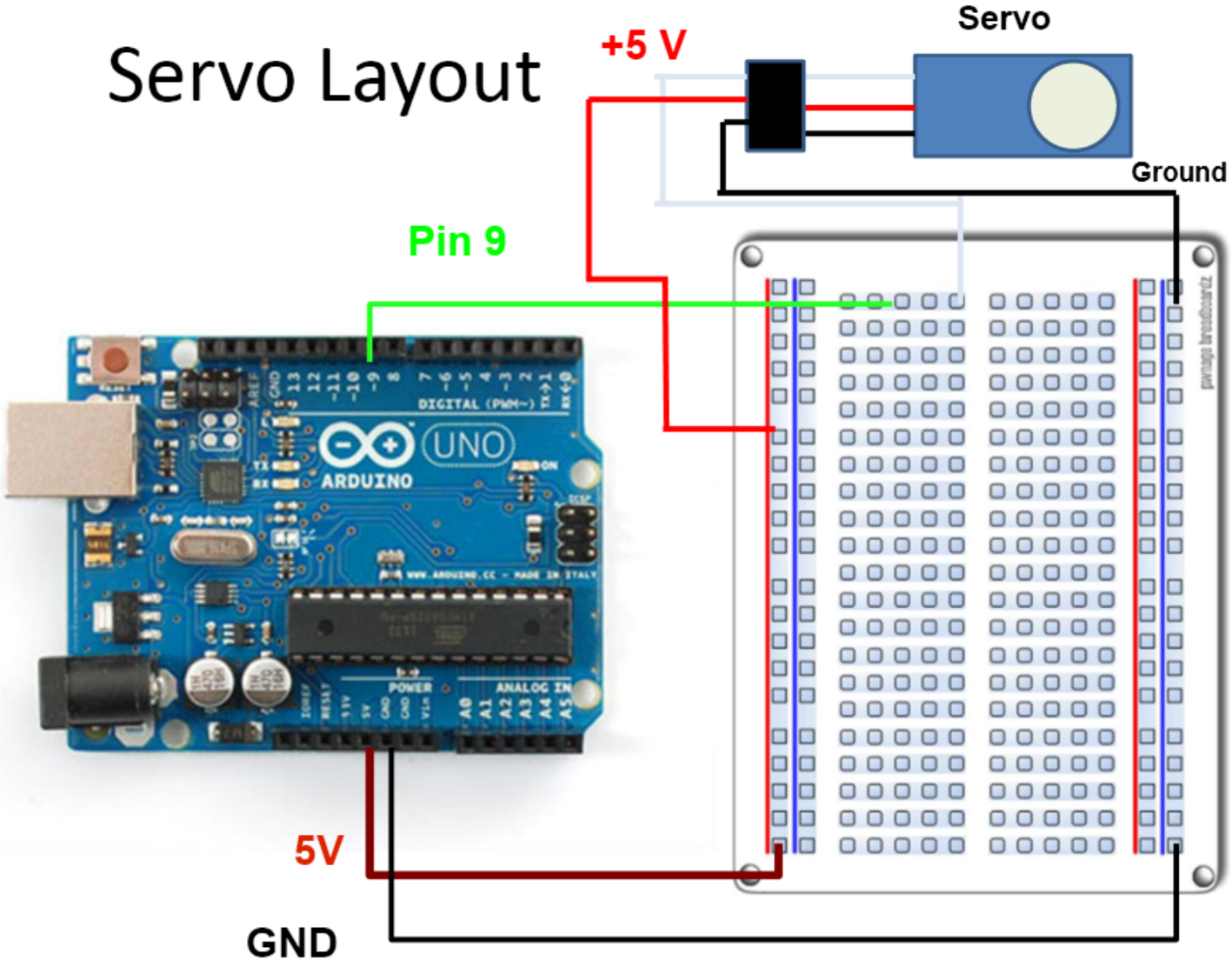
Utility Functions Provided for REV3

Function Name	Description	Example
`void motorA(int power)`	Turns on Motor A at power indicated. Range from -255 to 255. Negative numbers are backwards, Positive is forward. Zero means stop	`motorA(150);`

`void motorA(int power, int t)`	Turns on Motor A at power indicated for t microseconds. Stops motor when finished. Range from -255 to 255. Negative numbers are backwards, Positive is forward. Zero means stop.	`motorA(150, 1000);`
		`// motor A at 150`
		`// power`
		`// for 1 second`

`void motorB(int power)`	Turns on Motor B at power indicated. Range from -255 to 255. Negative numbers are backwards, Positive is forward. Zero means stop	`motorB(100);`
`void motorB(int power, int t)`	Same as motorA() except runs motor B	`motorB(-100, 2000);`

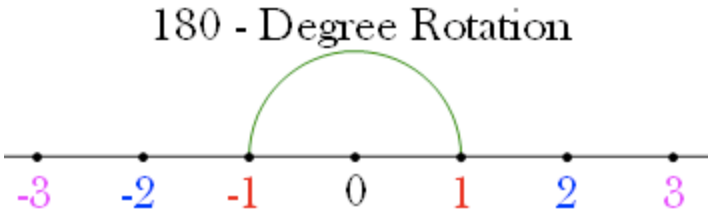
Single Power Source Setup



Two Types of Servos

Standard

- Only rotates within a 180 Degree range
- Write Value to Servo – holds the position
- Good for “Armatures”
  - Hands, Arms
  - Holding Devices



Continuous Rotation

- Rotates like a motor – all the way around
- Values written to Servo set “speed”
  - 0 -> Full Counterclockwise
  - 180 -> Full Clockwise
- Good for drive systems
  - Wheels, Rollers

Servo Commands

Servo Library: Must have this line in the beginning of each program

```
#include <Servo.h>
```

To Create Instance of Servo Object: (myServo can be any name)

```
Servo myServo;
```

To Attach a Servo to a Pin:

```
myServo.attach(9);
```

To Start Servo:

```
myServo.write(0);
```

```
// Servo Test
// Range: 0 to 180

// Bring in Servo Object and Functions
#include <Servo.h>

// Create Instance of Servo Object
Servo leftWheel;

// Setup - Run One Time
void setup() {
  // Attach Servo to Pin 9
  leftWheel.attach(9);

  // Rotate Servo at 0 and 180
  leftWheel.write(0);
  delay(4000);
  leftWheel.write(180);
  delay(4000);
  leftWheel.write(95);
}

void loop() {
  // No Code Here
}
```