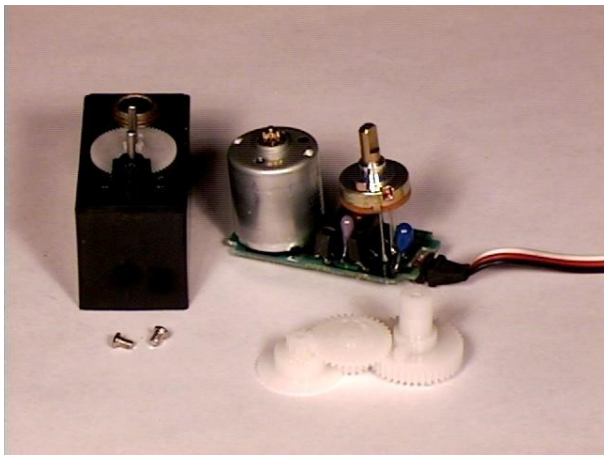
Servo Control

In this exercise, we will learn to control a servo motor with the Arduino. In particular, we'll use a continuous rotation servo so that we can make our Arduino rock-and-roll, but the control procedure that we will learn is applicable to any servo.

Servos



A servo is a small DC motor with the following components added: some gear reduction, a position sensor on the motor shaft, and an electronic circuit that controls the motor's operation. In other words, a servo is to a DC motor what the Arduino is the ATmega microcontroller---components and housing that make the motor easy to use. This will become abundantly clear when we work with unadorned DC motors next week.

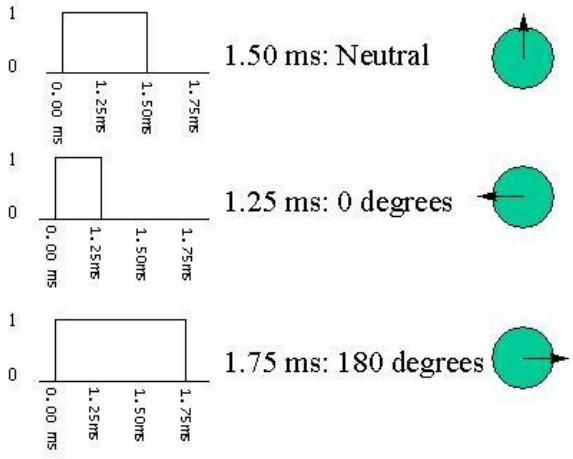
The gear reduction provided in a servo is large; the basic hobby servo has a 180:1 gear ratio. This means that the DC motor shaft must make 180 revolutions to produce 1 revolution of the servo shaft. This large gear ratio reduces the speed of the servo and proportionately increases its torque. What does this imply about small DC motors?

Servo motors are typically used for angular positioning, such as in radio control airplanes. They have a movement range of 0 up to 180 degrees, but some extend up to 210 degrees. Typically, a potentiometer measures the position of the output shaft at all times so the controller can accurately place and maintain its position.

Position Control

An external controller (such as the Arduino) tells the servo where to go with a signal know as *pulse proportional modulation* (PPM) or *pulse code modulation*, not to be confused with *pulse width modulation,* PWM---the form of analog output we learned about in the previous exercise. A control wire communicates the desired angular movement to the servo. The angle is determined by the duration of the pulse applied to the control wire.

PPM uses 1 to 2ms out of a 20ms time period to encode its information. The servo expects to see a pulse every 20 milliseconds (.02 seconds). The length of the pulse will determine how far the motor turns. A 1.5 millisecond pulse will make the motor turn to the 90 degree position (often called the neutral position). If the pulse is shorter than 1.5 ms, then the motor will turn the shaft to closer to 0 degrees. If the pulse is longer than 1.5ms, the shaft turns closer to 180 degrees.



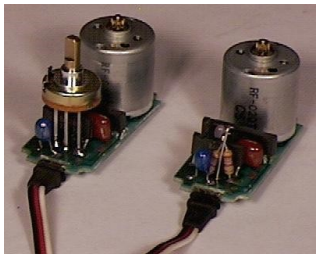
The amount of power applied to the motor is proportional to the distance it needs to travel. So, if the shaft needs to turn a large distance, the motor will run at full speed. If it needs to turn only a small amount, the motor will run at a slower speed.

Continuous Rotation Servos

A servo motor can be modified to provide continuous rotation as opposed to an angular position. The process requires 2 steps:

(1) Remove the mechanical stop that prevents full rotation of the output shaft.

(2) Remove the potentiometer position sensor and replace it with 2 equal-valued resistors whose combined resistance is equivalent to that of the pot. The second modification makes the servo "think" it is in the 90 deg position. For more information and a demonstration of this process, check the following link.



As stated above, the modification makes the servo think that the output shaft is always at 90 degrees. This is done by removing the feedback sensor, and replacing it with an equivalent circuit that creates the same reading as the sensor at 90 degrees. In this configuration, a control signal for 0 degrees will cause the motor to turn full speed in one direction, while a signal for 180 degrees will cause the motor to turn full speed in the other direction. Since feedback from the output shaft has been disconnected, the servo will continue at the same speed and in the established direction as long as the signal remains.

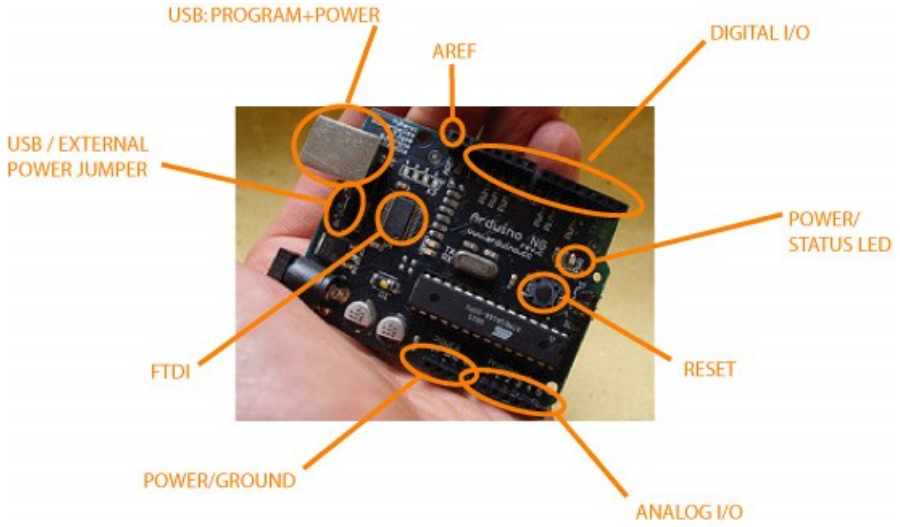
Wiring It Up

So now that we know how it works, let's try it! We are going to use Parallax servos mounted to a chassis with wheels so we can roll when the servos turn!

Mount your Arduino to the chassis using a tie-strap to secure it over the Board of Education (BOE)---the board that is mounted to the chassis. Be sure to place a shield between the Arduino and the BOE so that neither is damaged. Ask your instructor for material to use as a shield if you were not given anything at the start of class.

Because we plan to move, we will power the Arduino via an external power source---the battery pack mounted on the chassis. Note that the plug mates with the power connection on the Arduino. To configure the Arduino for external power, you also need to move the PWR_SEL jumper: move the jumper to cover the two pins closest to the external power jack. Ask your instructor if this is not clear.

HERE-→



The connection to the servos must be made on the breadboard attached to the BOE. As shown below, three connections are required for each servo.

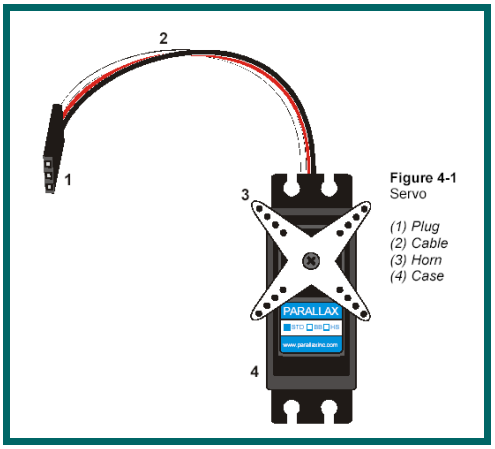
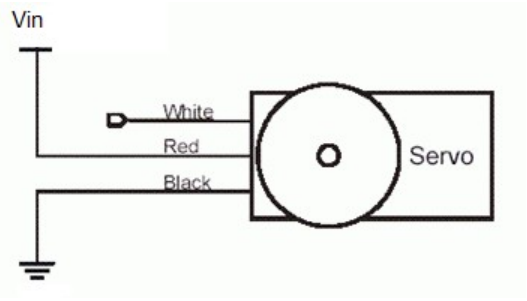


Figure 4-1
Servo

(1) Plug
(2) Cable
(3) Horn
(4) Case



Use the 2 three-pin headers in the BOE-bot kit to connect the servo plugs to the breadboard. The connections to the Arduino should be as indicated in diagram above. Be sure to connect the power lead to **Vin** (not 5V) on the Arduino, and connected the white lead to the Arduino pin issuing the PWM signals.

The Program

We will use the code presented by Community Robotics for the initial investigation of our servos. Read the article (it's quick) and use the code provided. Note that the procedures servoPulse1() and servoPulse2() are identical and therefore only one of them is really needed; the developer of this page was not a CS major ☺

Notice the behavior of the servos when you run the code. Did one servo remain stationary while the speed and direction of the other servo varied? If the servo intended to remain stationary (i.e., receiving the 90 deg. control signal) moved, then you should adjust its trim pot using the small Phillips-head screw driver in the BOE-bot kit. Adjust the trim-pot until the servo remains stationary when receiving the 90 deg. control signal, and then test and adjust the other servo in the same manner.

A Better Program

As it turns out, a "Servo" library exists for the Arduino; harrah for open-source! Look at the reference page for the servo library, and then run the *Sweep* example linked to the bottom of the page. This program takes both servos through their full range of motion. Make sure that you understand how this program works because you will be modifying it in the next section.

Your Turn

To refine your understanding of servo control, rewrite the *Sweep* program (you should also rename it) to move your robot in an interest way. Begin by experimenting with basic motion such as moving in a straight line, and turning right and left. Try going backwards and forwards, fast and slow. As your final demonstration, write a program that causes your robot to move in the shape of a square.