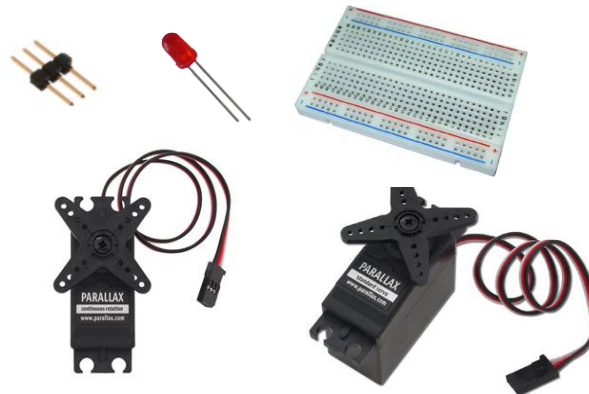


Project #3 – Continuous & Standard Servo Design

Components Needed:

- 1 Breadboard
- 1 Parallax Continuous Servo
- 1 Parallax Standard Servo
- 2 3-pin headers
- Jumper Wires
- 1 Red LED



Introduction:

In Project #2, you learned how a DC motor functions and the electrical components needed to power on and control the speed of the CPU fan. This project introduces you to another type of DC motor known as **servo motors**. Servo motors are simply DC motors that have a servo controller that directs the position of the motor. It is essentially a DC (direct current motor) with gears. Servo motors usually rotate rather slowly and with a relatively **strong torque**.

Servo motors come in different sizes and prices and are based on different technologies. In this course, we will be strictly talking about “hobby servos”. These types of servo motors are usually used in remote-controlled cars, since they are relatively easy to program and have enough torque to move the car. Here are two examples in which these types of servos can be found in:

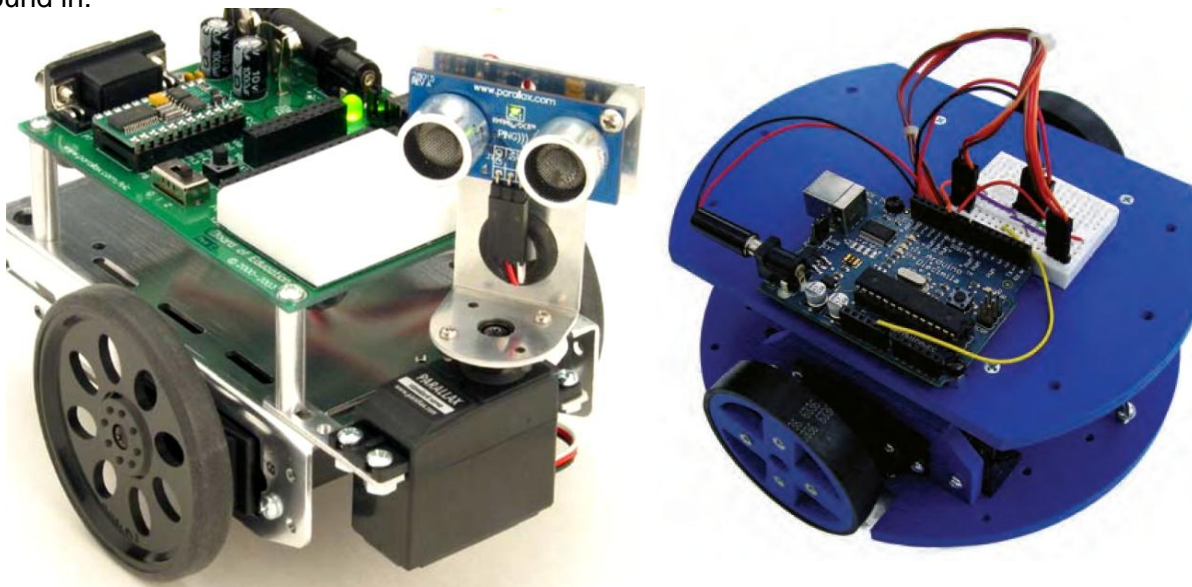


Fig 1-1 *Miniature robots that incorporate servo motors to move*

The servo motors used here allow the miniature robots to move since they rotate slowly but with a very high amount of torque. You will notice that there are two types of servos including in your kits: a **continuous servo** and a **standard servo**. Any guesses as to what type of servos the

robots are using on the bottom base to move the wheels? They answer should be **continuous** since their range of motion is nonstop, hence its name. The standard servo's functionality then should be the exact opposite, and guess what, it is! Before we get into detail as to how they work different, here is an illustration that describes the servos in more detail:

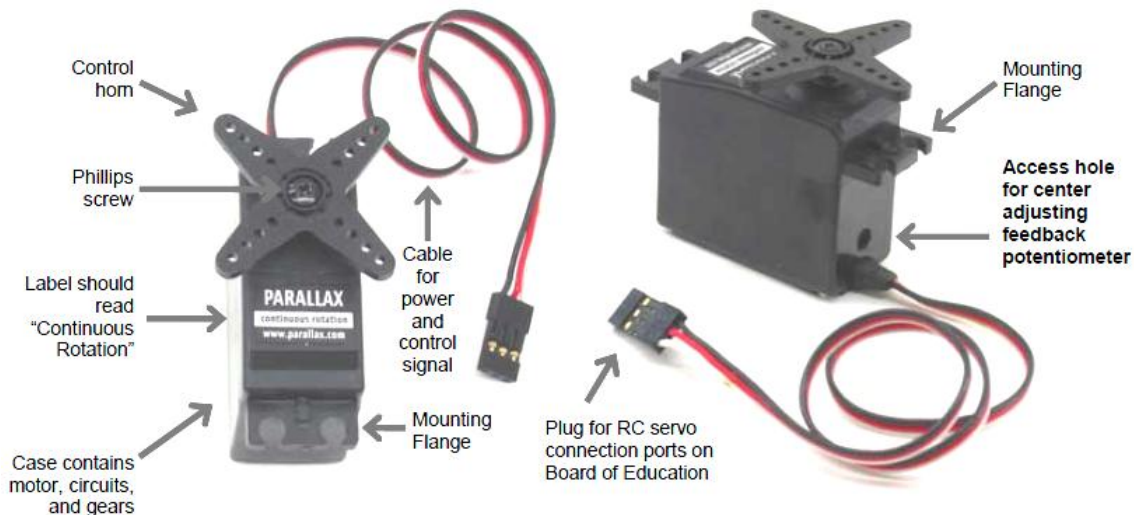


Fig 1-2 Detailed description of a Continuous servo motor

Just like the DC Motor fan, there are **three** wires that are associated with these servo motors. To connect the servo motor to the Arduino, a 3-pin header will be required as well as a breadboard for simplicity. Here is the connection diagram:

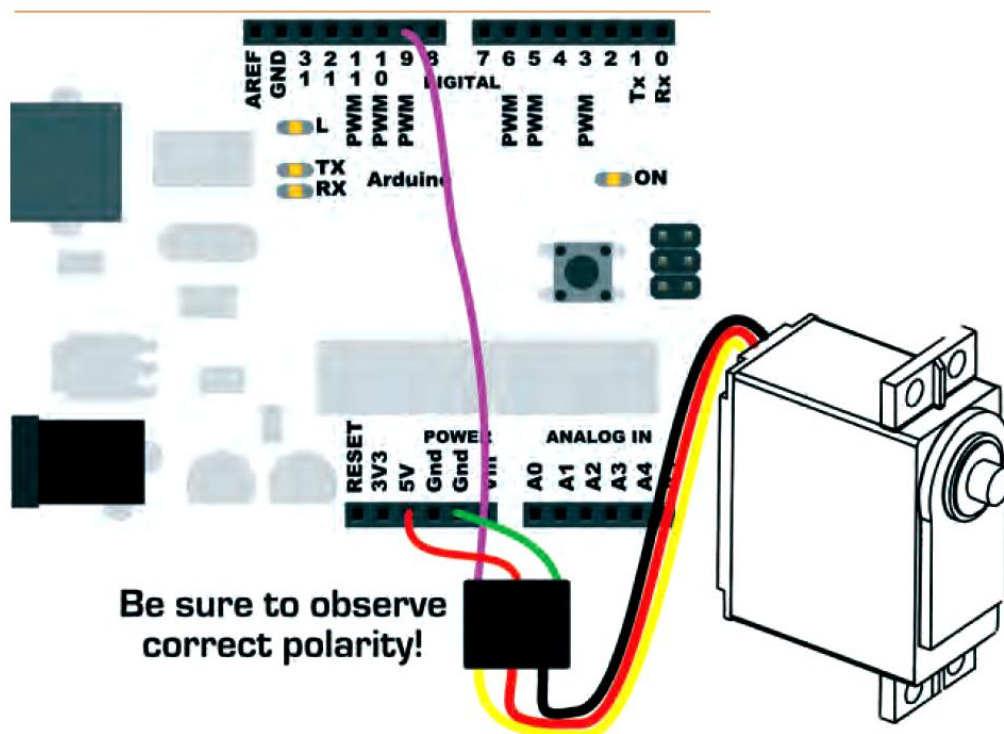
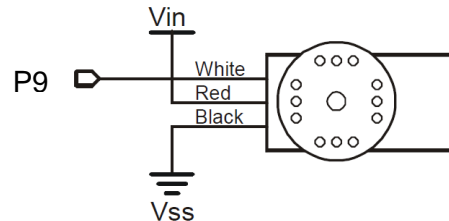


Fig 2-1 Diagram showing how to connect to Arduino Board

The **black wire** from the servomotor to the header is the **ground (GND) pin**. The **red wire** is the **power** (Vdd) for the servomotor which requires 5 Volts. The **yellow wire** (or white as you will observe) from the servo is connected to a pin on the digital I/O lines of the board. This can be connected to any one of the available digital pins that support PWM (they will have this symbol: ~), and in this case, we will use pin number 9. Here is an illustration of the connection:



Why can't we connect to digital pin 13 or 12? They are digital pins right? The answer is **yes**, they are digital pin outputs, but not the **"type" of output we require**. Servo motors need **"pulses"** to activate the gears that will then spin the control horn clockwise or counterclockwise based on the direction specified. This brings forth the question: **what exactly is a PWM signal?**

PWM Signal:

A **PWM** signal stands for **Pulse Width Modulation**, which is a commonly used technique for controlling power to electrical devices, made practical by modern electronic power switches. Servo motors **only move** when there is a changing voltage on the control pin, which is a PWM signal! Due to its "modulation technique", we can program a servo motor to turn in different directions and at various speeds. Here is an example of what a PWM signal looks like:

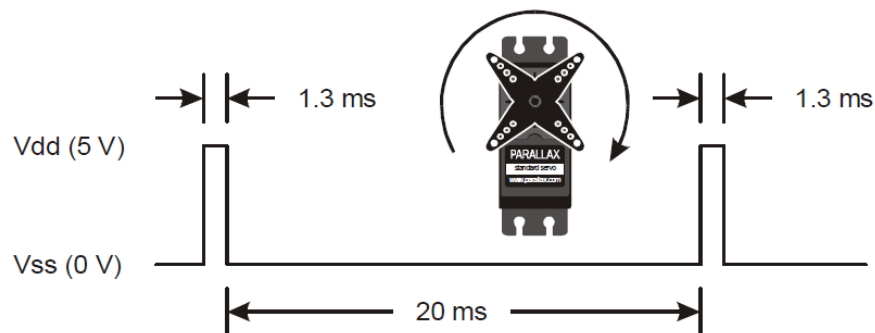


Fig 2-2 PWM Signal activating the Servos with a duration of 1.3 ms

Pulses are square waves that denote a certain time duration that the **voltage will be active**, in which Figure 2-2 shows an active duration rate of 1.3ms. What does that mean? It means that the **duration of the 5V will be active for only 1.3ms, which will make the motor move!** There is a 20ms delay for the next pulse to arrive since the voltage cannot be active all the time. Why? A 20 millisecond delay allows the gears to move steadily and gives the servo motor a smooth flow of movement. If you are curious, this type of modulation exhibits the same principles that of a **pulse train**. A 1.3ms pulse allows the servo motor to move clockwise as shown. Any guesses as to how we can make it move counterclockwise??

If a 1.3ms pulse train allows the servo to move clockwise, **there has to be** another time duration that can allow the servo to move in the opposite direction. Fortunately, the answer is given below:

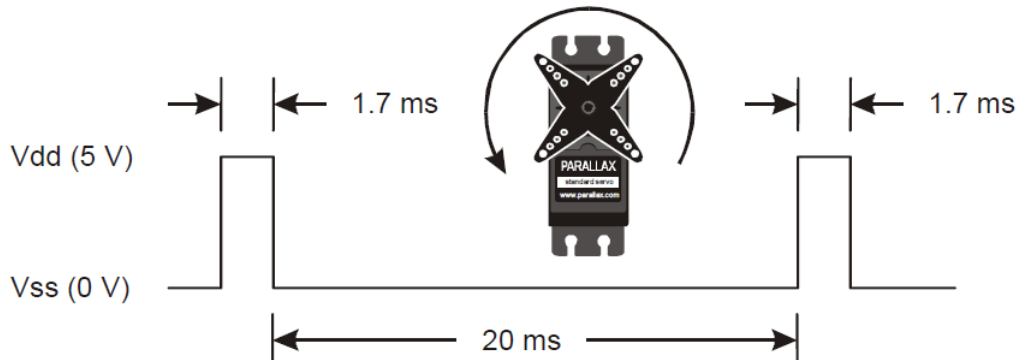


Fig 2-2 PWM Signal activating the Servos to move counterclockwise

A pulse width modulation with duration of **1.7ms allows the servo to move counterclockwise** as shown. It still has the delay of 20ms for the next pulse which is needed for steady movement. There's even a pulse width duration that “centers” the servo (which makes them not move in any direction):

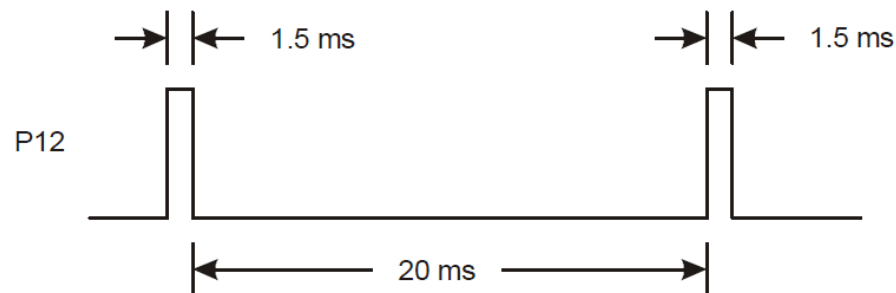


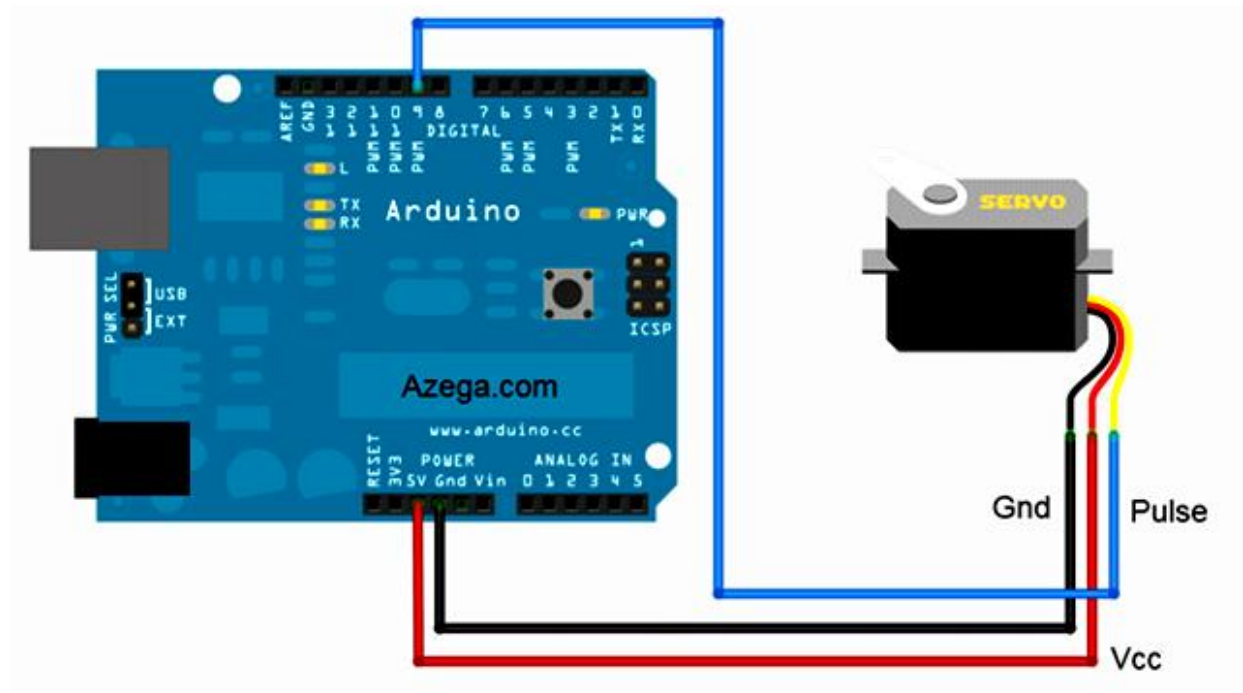
Fig 2-2 PWM Signal activating the Servos to remain still

Continuous Servo vs. Standard Servo

What is the difference between a Continuous servo vs. a Standard servo? Standard servos are designed to receive electronic signals that tell them **what position to hold**. These servos control the positions of radio controlled airplane flaps, boat rudders, and car steering. Continuous rotation servos receive the same electronic signals, **but instead of holding certain positions, they turn at certain speeds and directions**. Continuous rotation servos are ideal for controlling wheels and pulleys.

A continuous servo can potentially move in any direction with a set specified speed as opposed to a standard servo where the direction of movement is **limited to 90° either left or right**, or 180° from left to right completely. When you program the servos to move in the Arduino IDE, the pulses are already set by the library (we will discuss this later), so you do not have to worry about the pulse durations needed to move the servos at a certain speed or directions.

Essentially, the **white wire is the control wire** that sets the servos for direction and speed based on the pulses assigned:



Now that we have discussed the hardware overview as to what a servo motor is and how it functions, we can now discuss the software and the commands needed to send these pulses.

Arduino Code:

Using the Continuous servo with the wheel attached, highlight the following code:

Code Listing 1.1

```
#include <Servo.h>           //include the Servo library
Servo myServo;               //create a Servo object to control the Servo
int delayTime = 4000;        //delay period, in milliseconds

void setup()
{
  myServo.attach(9);          //Servo is connected to pin 9 (~PWM)
}

void loop()
{
  myServo.write(0);            //Rotate servo to position 0
  delay(delayTime);            //Wait delay as assigned above
  myServo.write(180);          //Rotate servo to position 180
  delay(delayTime);            // Wait again!
}
```


Once the code is uploaded, observe the direction of the wheel and see if it moves clockwise for 4 seconds, then counterclockwise for 4 seconds and repeats itself again. **If it does, success!!!** If not, check to see if it is a continuous servo and that the numbers are the same as given on Code listing 1.1. The syntax is explained below.

Arduino Code explained:

The first line of code is something newly introduced, which are C libraries:

```
#include <Servo.h>           //include the Servo library
```

“**Servo.h**” is a library that is already preinstalled in the Arduino IDE folder. Remember all those pulse durations we discussed that activate the servos to move in a set direction and speed? The servo.h library already includes these pulse durations so you don’t have to worry about them!

Remember, Arduino is pseudo code whose programming techniques model that of C programming, in which there are various libraries that help make programming more easier and simpler. In this case, Servo.h is the library that includes everything we need to get the servos spinning!

Now let’s talk about this:

```
Servo myServo;               //create a Servo object to control the Servo
```

Any ideas as to what **Servo** means? The line “Servo myServo” creates, or “instantiates,” a servo object. The functionality of this object is defined in the Servo.h library and its accompanying Servo.cpp programming code file (don’t worry about what a .cpp file is).

Guess what? **Servo** is actually the name of a class, which is how the Arduino uses its libraries. With a class, you can create multiple instances (copies) of an object, without having to duplicate lots of code. In the case of servos, you could create two objects: one for each physical Servo on your robot.

For example, if you want to connect three servos to the Arduino, you could say **any of the following**:

```
Servo myServo1; or Servo motor1;  
Servo myServo2; or Servo cont1;  
Servo myServo3; or Servo motorfinal;
```

Third line of code is simply stating a variable with a value:

```
int delayTime = 4000;        //delay period, in milliseconds
```

The line “int delayTime = 4000” creates a **data variable named delay**. Variables are used to hold information for use throughout the sketch. The int tells the Arduino compiler that you wish to create an integer type variable which can store any whole number from -32,768 to 32,767.

The last 5 lines of code are explained on the next page:

The `setup()` function contains one statement, **`myServo.attach(9)`**. Here's what it all this means:

- “**`myServo`**” is the name of the servo object that was defined earlier.
- **`Attach`** is a method that you can use with the Servo object. Methods are actions that you use to control the behavior of objects. In this case, attach tells the Arduino that you have physically connected the servo to PWM digital pin 9 and you want to activate it. It is similar to how we activate an LED, using the **`digital.write`** command!

The void loop function is explained below:

```
myServo.write(0);           //Rotate servo to position 0
delay(delayTime);          //Wait delay as assigned above
myServo.write(180);         //Rotate servo to position 180
delay(delayTime);          // Wait again!
```

- **`myServo.write(0)`** is another method using the `myServo` object. The **`write` method** instructs the servo to move all the way in one direction (0 = clockwise)
- **`delay(delayTime)`** tells the Arduino to wait the period specified earlier in the `delayTime` variable, which is 4000 milliseconds (4 seconds).
- The two statements are repeated again, this time with **`myServo.write(180)`** to make the servo go the other direction (180 = counterclockwise).

Since **0** makes the servo move clockwise and **180** makes the servo move counterclockwise, then **90** must make it not move in either direction. If we say the following:

```
myServo.write(90);          //Rotate servo to position 90
```

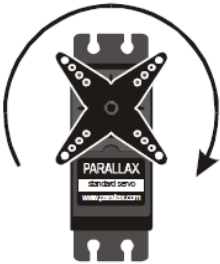

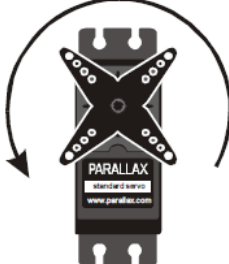
The servo motor should not make any movement at all since its been “centered”. If it does move, then you must re-center the servo by **calibrating** it. A demo is given below:



Fig 3-1 *Calibrating the Continuous servo using the potentiometer*

As the servo is spinning while set at 90, turn the potentiometer until it **stops moving**. Once it stops moving, take the screwdriver out since it has been **successfully calibrated!!!**

When completing the tasks assigned, you will need to know which value sets the direction of the servo motor. The following table lists the values needed:

Values that Set the Direction of the Servo		
0	90	180
		
Clockwise	None	Counter-clockwise

Let's try another example:

Code Listing 1.2

```
#include <Servo.h>
Servo myServo;
int delayTime = 4000;

void setup()
{
  myServo.attach(9);      //Servo is connected to pin 9 (~PWM)
}

void loop()
{
  myServo.write(0);        //Rotate servo to position 0 (clockwise)
  delay(delayTime);        //Wait delay as assigned above
  myServo.write(90);       //Rotate servo to position 90 (shouldn't move)
  delay(2000);             // 2 second delay for next directional movement
  myServo.write(180);      //Rotate servo to position 180 (counter-clockwise)
  delay(delayTime);        //Wait delay as assigned above
  myServo.write(90);       //Rotate servo to position 90 again (stops servo)
  delay(3000);             // 3 second delay before loop restarts itself again!
}
```

Notice **something different** in this code? We are assigning “**myServo.write(90)**” to stop the servos from moving. This will become essential in completing the tasks assigned. Now let's discuss standard servo's and their methods of operation:

Standard Servo's work differently:

As we mentioned before, Continuous servos set the speed and direction while Standard Servos set the position.

What does that mean? Let's take a closer look at the following line of code:

```
myServo.write(0);           //Rotate servo to position 0
```

The **continuous servo** will interpret this by turning the **control horn clockwise**. Upload that same line of code to a standard and it will **do something completely different!** That's because the standard servo is moving **towards the position of 0** (position based).

Let's try this example:

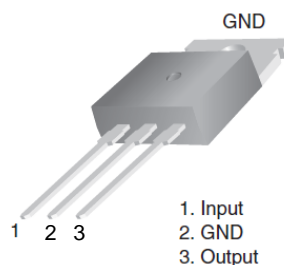
```
myServo.write(20);          //Rotate servo to position 20 (not angle measurement of 20)
```

In this case, **20** is the position indicator (not the angle) for the Standard servo. For the tasks associated with this project, you will need to determine the position as to which angle it is associated to. An illustration of the common movements for Standard Servos is given below for reference (based on angular measurements):



Fig 3-2 Standard Servo movements based on angular position

Everything else follows the intuition of a continuous servo. **If your standard servo is disabling the Arduino board (high current draw)**, use the voltage regulator and the 9V Battery as the power source. Here is the connection diagram for the LM705AC regulator:



The tasks for this project are outlined below. For this project, there is a **challenge** given to test what you have learned from the previous projects and see if you can reinforce those concepts here. While it may seem a little complex at first, it is actually quite easy!

Using all the intuition learned for this project, you can complete the following tasks without any trouble. As always, ask for help if you do not understand the question given.

TASKS

- 1) Program the Continuous servo to rotate **clockwise** for 4 seconds, pause for 2 seconds, and then rotate **counterclockwise** for 5 seconds in a repetitive loop.
- 2) Program the Standard Servo to move to a home position of 90° and stay there for 3 seconds. From home move **counterclockwise** 45°, pause for 5 seconds, then rotate **clockwise** 90°, stay there for 3 seconds. Do this all in a repetitive loop.
- 3) Now do both task 1 and 2 together in the same program.

*When we say “repetitive loop”, we want to see the standard servo move towards the two indicated **positions** based on the angular direction specified, not the angle itself! If the question is still confusing, ask a TA for further clarification.