# Using A Playstation 2 Controller with your Arduino Project

It occurred to me one day as I was designing a little box with some joysticks and a few buttons to control a project of mine, that it looked familiar. I suddenly asked myself why I was trying to build what was essentially a game controller from scratch when there were perfectly good game controllers around that included the inputs I was hoping to use as well as a heap of other buttons and features far beyond what I needed. And they were cheaper than the small handful of components I was going to need to buy to make my own.

There has been millions of dollars poured into the development of the Playstation Controllers, and so they are very sophisticated, well designed, and robust pieces of equipment that are ideal for using as an interface for mechatronics projects. Even though the Playstation 2 game console is now a rather outdated piece of equipment, the Playstation 2 Controller Clones are still made and can be purchased through Trademe extraordinarily cheaply.

Not only that, but they are easy to connect to an Arduino. In addition to the two high quality joysticks, all the buttons are pressure sensitive, which adds even more functionality to the device.

This article demonstrates how you would connect a Playstation 2 Controller to an Arduino, and how easily it can be used to control various devices thanks to the PS2Xlib Arduino Library.
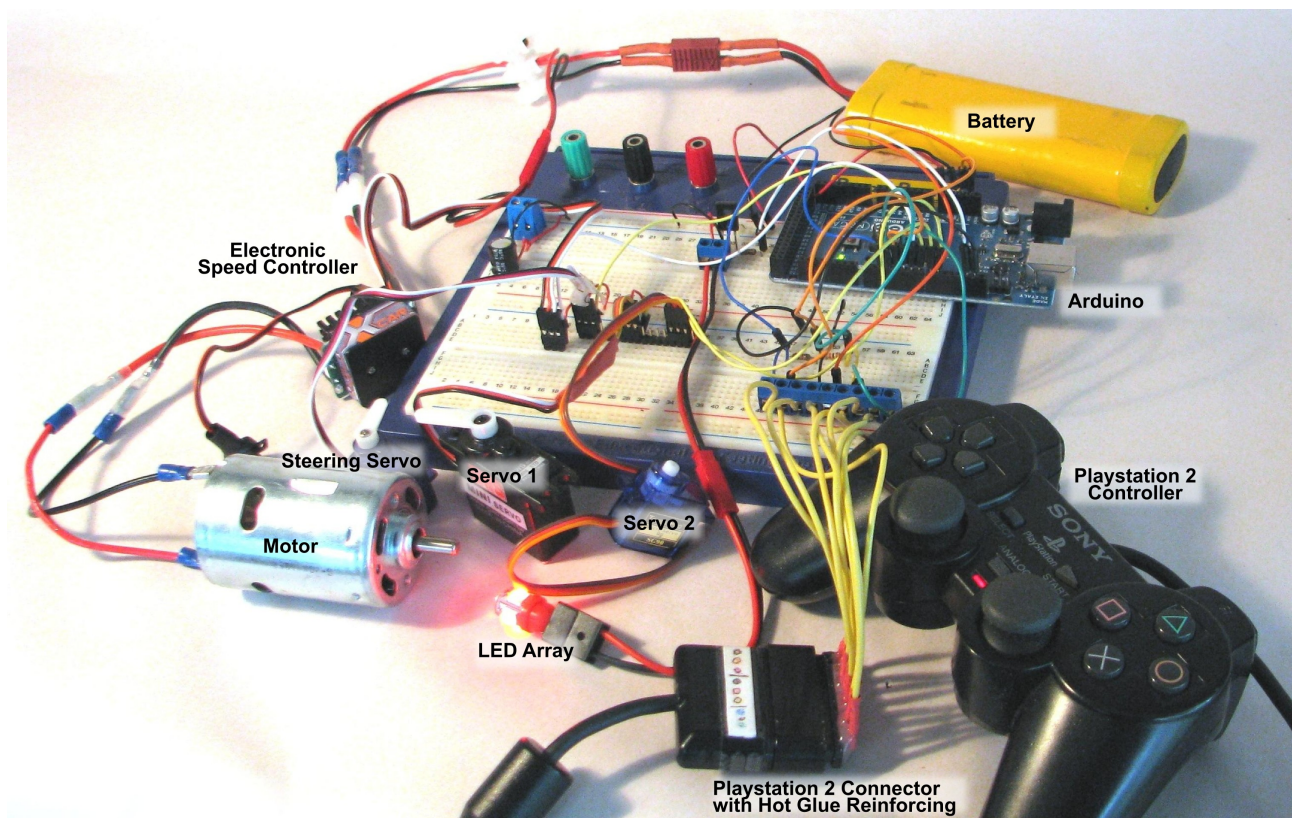
Applications for PS2 Controllers

So where could you use a Playstation controller aside from on a Playstation console?
- Controlling a wheeled or tracked vehicle's motion.
- Controlling a robotic arm.
- Interacting with a computer
- Controlling a pan and tilt camera mount
- DIY radio control system by using a bluetooth module, or radio module to transmit the commands

from the Playstation 2 controller to another Arduino which is controlling the vehicle.
• Then there is my own pet project which uses the Playstation 2 Controller as the pilot interface for an ROV (aka. underwater drone).

Connecting the Playstation 2 Controller to the Arduino

There are two options for connecting the controller to the Arduino. The method for masochists, is to cut the connector off and connect the wires into a row of terminal headers or directly solder them to some other plug. The wire colours in the diagram below may help you identify which wire is which (The 7V – 9V rumble motor supply wire is often gray in colour). A better option is to retain the plug and find a suitable socket. In stark contrast to the ease of purchasing a controller, finding sockets is much harder. It is possible to purchase sockets and breakout boards from the Robotshop (http://www.robotshop.com/en/ps2-connector.html) in the US. Unfortunately there are no local suppliers. The cheaper alternative if you are into a bit of hacking, is to extract the sockets from a Playstation 2 Controller to USB adapter (about \$5 - \$10 or so through Trademe). I have found the tabs on the back of the sockets to be quite fragile and it is worth embedding them in hot glue once you have soldered some wires to it.

Playstation 2 Socket
Rare and Expensive

Playstation 2 to USB Converter
**Readily Available and Cheap**

Sources of Playstation 2 Sockets

The illustration below shows the Playstation 2 plug, and the pin labels. In order to connect to the Arduino we only need to connect the Data, Command, Ground, 3.3V, Attention, and Clock pins. The rumble motor pin only needs to be connected to a 7-9V supply if rumble feedback is desired.
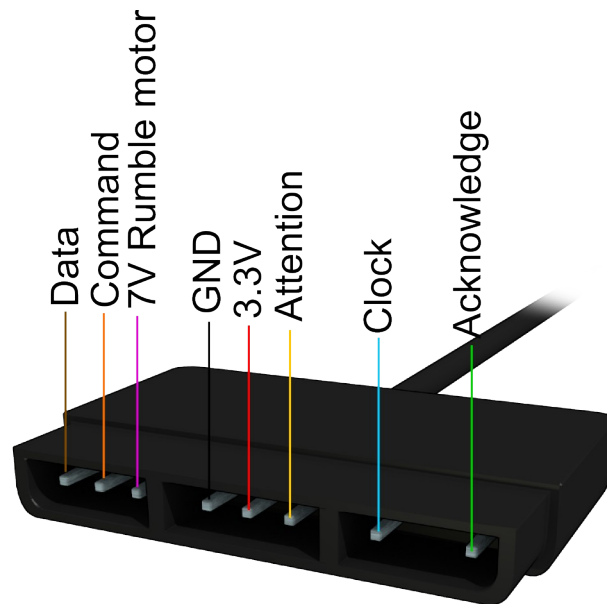
The Software Side

Thanks to one Bill Porter an easy to use library is available which will allow users to use a Playstation 2 controller (or even a Guitar Hero controller) with an Arduino. You can find it through Bill Porter's website: "The Mind of Bill Porter" - http://www.billporter.info/. More specifically here is the link to his page with links to his Library - http://www.billporter.info/2010/06/05/playstation-2-controller-arduino-library-v1-0/ and the Library Source Code can be obtained from Github https://github.com/madsci1016/Arduino-PS2X, just click on the "Download ZIP" button on the right side of the page. Once you have the zip file downloaded, start your Arduino Interface, and navigate through the menu **Sketch>Import Library>Add Library**. This will open a dialogue where you can navigate to your downloaded the PS2X Library zip file.
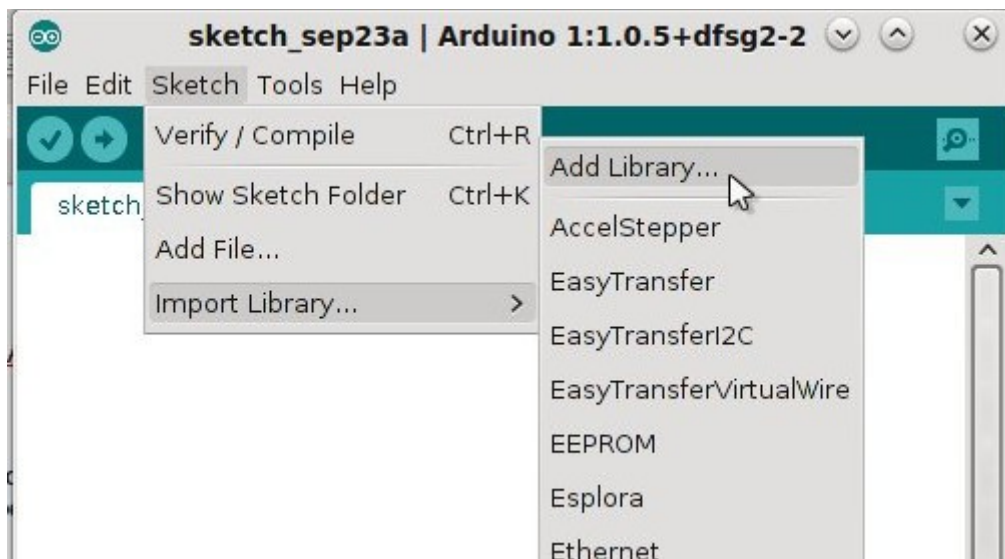
The example sketch included with the library is a great demonstration of the capabilities of the library and makes use of all of the features of the Playstation 2 Controller. The demonstration that is outlined in this article is not quite so sophisticated but covers the essentials needed to use a Playstation 2 Controller to actuate motors, servos, and switch things on and off.

To use the library in an Arduino Sketch, it needs to be called using the following commands;

```
#include <PS2X_lib.h>
```

```
PS2X ps2x;
```

In the **setup** part of the sketch the command to let the Arduino know how the controller is connected to it, looks like;

```
ps2x.config_gamepad(5,4,3,2, false, false);
```

where the numbers are the Arduino's digital pins where the Playstation 2 Controller pins are connected as below;

 gamepad(clock, command, attention, data, pressure sensitivity enabled, rumble enabled)

If you wanted to use the button pressure sensitivity feature, in the command line above you would set this to ***"true"*** and if you wanted the rumble motor available to provide feedback then you would set this to ***"true"*** too. For use of the rumble feature please look at the example sketch that comes with the PS2X library.

Once the controller is set up, the Arduino loops through the sketch continuously. Once per loop through, the Arduino needs to communicate with the controller to gather all input data. This is done with the following command.

```
ps2x.read_gamepad();
```

Now we can read which control has been used. The labels used in the PS2X library are very logical. The diagram below shows each of the names for the buttons and sticks.
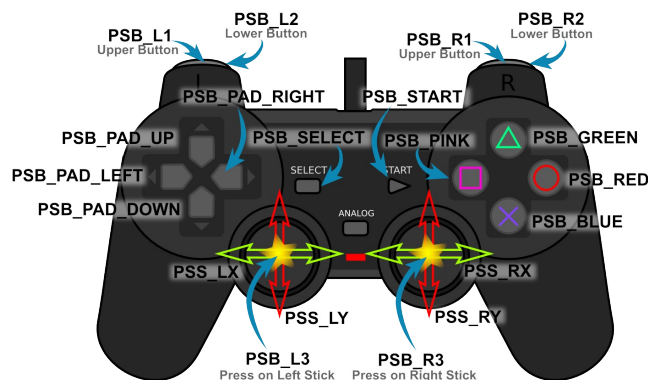
The buttons with the coloured shapes can also be referred to by the names PSB_TRIANGLE, PSB_CIRCLE, PSB_CROSS, and PSB_SQUARE.

To use the analogue pressure sensitivity on the keys the names are the same except for substituting "PSAB" for "PSB". So to allow a pressure reading from the Green Triangle button the name would be PSAB_GREEN or PSAB_TRIANGLE.

The buttons can be pressed, pressed and held, or pressed with a varying pressure, so there are a number of *methods* that can be applied to the buttons and joystick. The *methods* are; `Button Pressed`, `Button`, and `Analog`. Here are some examples of how these are used.

- `ps2x.ButtonPressed(PSB_RED)` is for a simple press of the red circle button.

- `ps2x.Button(PSB_PAD_DOWN)` is for the down button on the pad being pressed and held.

- `ps2x.Analog(PSAB_CROSS)` is the command for measuring the pressure applied to the "X" button, if pressure sensitivity has been enabled. As you can see the "PSAB" form of the name has been used for the button.

- `ps2x.Analog(PSS_RY)` is the command to obtaining readings off the right analogue stick in the vertical direction.

Bringing it Together

To illustrate how this works, here is an example project. It makes use of a DC motor, an electronic speed controller (in this case an RC car ESC), a series of servos, and an LED array designed to run on 7-12V. If this were to be applied to a real project, it could be a car with steering, and two extra servos to control a pan and tilt camera mount. A single LED would not need the transistor, but I have included a 12V LED array and transistor to illustrate that other high voltage and high current devices can be triggered by such a system.

The is nothing demanding about the sketch and it can run on any Arduino or Arduino clone.

Below is a diagram of the circuit and components. The ESC supplies 5.6V to the servos through its control connector. The ESC I have used has a switch on it that needs to be turned on before the motor will run and before any electricity is available to the servos. Other than that, the only components are the three resistors for the connection to the Playstation 2 controller and the LED's transistor. The capacitor is probably not necessary because the battery should be able to deliver a relatively smooth supply.
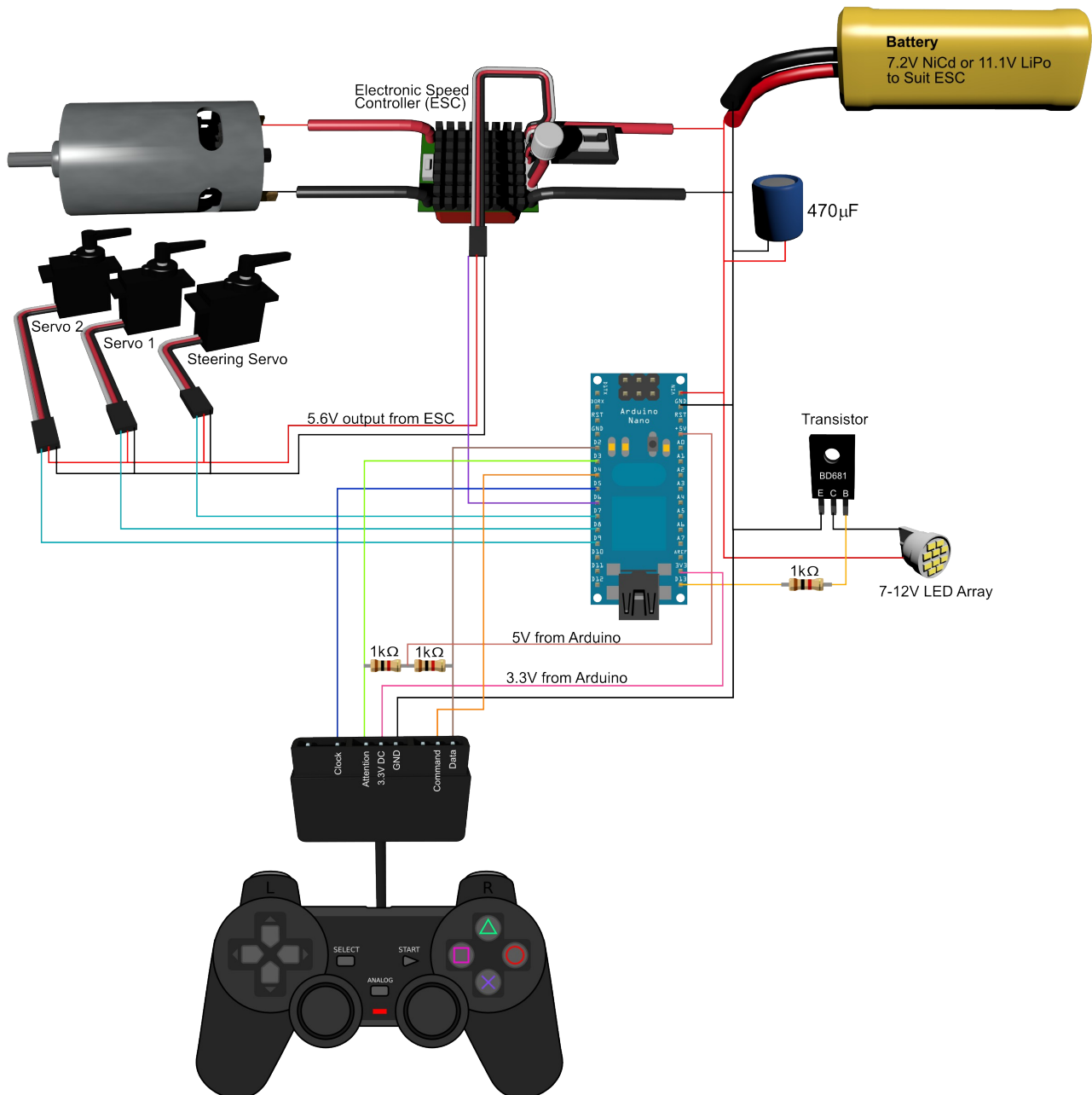
The controls we want to use are; a switch for the LED array, and the two analogue joysticks to actuate the servos and ESC. Because the ESC "speaks" Servo, we just treat it as a servo in the code.
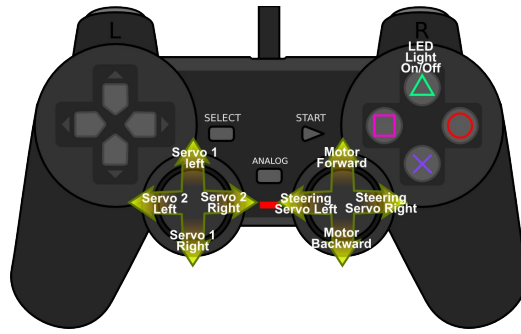
Here is the sketch.

```
/*
PS2Controlv0.ino
21 September 2015
Hamish Trolove - www.techmonkeybusiness.com

This sketch illustrates the use of a Playstation 2 Controller to
actuate a series of servos and an Electronic Speed Controller (ESC)
as you might do with a vehicle.  An LED light is used to illustrate
the use of the Playstation 2 Buttons.

Pin assignments are:

3.3V output to PS2 red pin
5V output to 1Kohm pull up resistors for PS2.
Pin D02 to PS2 brown pin (data)
Pin D03 to PS2 yellow pin (attention)
Pin D04 to PS2 orange pin (command)
Pin D05 to PS2 blue pin (clock)

Pin D06 to ESC Signal Pin
Pin D07 to Steering Servo Signal pin
Pin D08 to Servo 1 Signal pin
Pin D09 to Servo 2 Signal pin

Pin D13 to LED Transistor Base

The ESC servo connection supplies 5.6V to the servos.

The coding pulls on the PS2X library developed by Bill Porter.
See www.billporter.info for the latest from Bill Porter and to
download the library.

The controls used for this sketch are;
Right Stick - X-axis = Steering Servo left/right, Y-axis = ESC forward/backward
Left Stick - X-axis = Servo 2 left/right, Y-axis = Servo 1 left/right

Triangle = Toggle the LED
*/


#include <Servo.h>  //For driving the ESCs and Servos
#include <PS2X_lib.h> // Bill Porter's PS2 Library

PS2X ps2x;  //The PS2 Controller Class
Servo SteeringServo;  //Create servo object representing SteeringServo
Servo ServoN1;  //Create servo object representing Servo 1
Servo ServoN2;  //Create servo object representing Servo 2
Servo ESCcontrol;  //Create servo object representing ESC

const int LEDpin = 13;  //green LED is on Digital pin 13
```

```
volatile boolean LEDHdlts; //LED headlights on/off toggle

int PlyStnRStickUpDn = 0;   //Value read off the PS2 Right Stick up/down.
int PlyStnRStickLtRt = 0;   //Value read off the PS2 Right Stick left/right
int PlyStnLStickUpDn = 0; //Value read off the PS2 Left Stick up/down
int PlyStnLStickLtRt = 0; // Value read off the PS2 Left Stick left/right


int ESCSetting = 90; //Setting for the ESC (degrees).
int StrServoSetting = 90; //Setting for the Steering Servo
int ServoN1Setting = 90; //Setting for the Servo 1
int ServoN2Setting = 90; //Setting for the Servo 2



void setup()
{
  ps2x.config_gamepad(5,4,3,2, false, false);
  //setup pins and settings: GamePad(clock, command, attention, data, Pressures, Rumble)
  //We have disabled the pressure sensitivity and rumble in this instance.
  pinMode(LEDpin, OUTPUT);  //Sets the LEDpin to output

  LEDHdlts = false;  //Sets the Headlights to off
  SteeringServo.attach(7);// attaches the Steering Servo to pin 7
  ServoN1.attach(8);// attaches the Servo 1 to pin 8
  ServoN2.attach(9);// attaches the Servo 2 to pin 9
  ESCcontrol.attach(6,150,2250);// attaches the ESC to pin 6
  //The ESC attachment command above also includes the signal settings
  //for the maximum and minimum that the ESC will recognise.  This
  //varies for different ESCs.

  //Set all ESCs and Servos to a neutral 90 degree position
  //this avoids the ESC trying to calibrate.
  ESCcontrol.write(90);
  SteeringServo.write(90);
  ServoN1.write(90);
  ServoN2.write(90);
  delay(5000); //Five second delay to allow ESC and controller to
               // fully initialise.

}

void loop()
{
  ps2x.read_gamepad(); //This needs to be called at least once a second
                       // to get data from the controller.

  if(ps2x.ButtonPressed(PSB_GREEN))  //Triangle pressed
  {
    LEDHdlts = !LEDHdlts; //Toggle the LED light flag
  }


//Analogue Stick readings
  PlyStnRStickUpDn = ps2x.Analog(PSS_RY); //Right Stick Up and Down
  PlyStnRStickLtRt = ps2x.Analog(PSS_RX); //Right Stick Left and Right
  PlyStnLStickUpDn = ps2x.Analog(PSS_LY); //left Stick Up and Down
  PlyStnLStickLtRt = ps2x.Analog(PSS_LX); //Left Stick Left and Right

//Readings from PS2 Controller Sticks are from 0 to 255
//with the neutral being 128.  The zero positions are to
//the left for X-axis movements and up for Y-axis movements.

//Variables to carry the settings for the ESCs and Servos
//The values from the PS2 Sticks are mapped to 0 to 180 degrees

  ESCSetting = map(PlyStnRStickUpDn,-256,256,0,179);
  StrServoSetting = map(PlyStnRStickLtRt,-256,256,0,179);
```

```
  ServoN1Setting = map(PlyStnLStickUpDn,-256,256,0,179);
  ServoN2Setting = map(PlyStnLStickLtRt,-256,256,0,179);


//Write it to the Servos or ESCs

  ESCcontrol.write(ESCSetting);
  SteeringServo.write(StrServoSetting);
  ServoN1.write(ServoN1Setting);
  ServoN2.write(ServoN2Setting);


  digitalWrite(LEDpin,LEDHdlts); //Light the LED based on headlights status flag
  delay(15);
}
```

All things going well, when you connect this up and connect in the battery, you should be able to push the triangle button to light the LED and move the joysticks to move the servos and run the motor.

If it does not automatically go into Analogue Mode (as indicated by the read light on the controller), press the Analog button. This will enable the joysticks on the controller and set things going.

The circuit and sketch are designed with a car ESC in mind. This means a servo command of 90 degrees is the neutral position. The `ESCcontrol.write(90);` line in the Setup routine sets the throttle to neutral to allow the ESC to initialise. If you are using a Radio Control Aircraft ESC you will need to initialise it at 0 degrees.

Depending on the quality of the Arduino you are using you may need to have the USB connected to the computer to power the Arduino. Ordinarily the Arduino should be able to run on the 7.2v being supplied from the NiCd battery. If you have an 11.1V LiPo battery, and ESC designed to suit it then you should not have any problems.

Final Remark

With an input device as sophisticated as the Playstation 2 controller there are a wealth of different mechatronics projects that they can interface with. The PS2X library makes it very easy to access the full range of functions available from the Playstation 2 controller. Why reinvent the wheel when there are already ultra-high performance mags available for cheaper prices than a couple of components?