ROV Control Sketches – Fourth Edition

This is the fourth and probably final release of the Arduino Sketches used to control the ROV. This edition introduces the HS5803-14BA depth sensor and the HMC5883L Digital Compass both of which are *i2c* devices. Because I had used Arduino Nano pin A5 for the onboard battery voltage monitoring in the earlier control system, a minor modification to the ROV circuit and code was required to free the pin up for the *i2c* sensors.

The sketch for the topside station, *ROVPS2Control_Masterv8.ino*, did not need to be modified from the earlier version (ROVPS2Control_Masterv7.ino) to cater for the added sensors, because there were already placeholders included.

The previous editions of this control system can be found here:

- Version 3
- Version 2
- Version 1

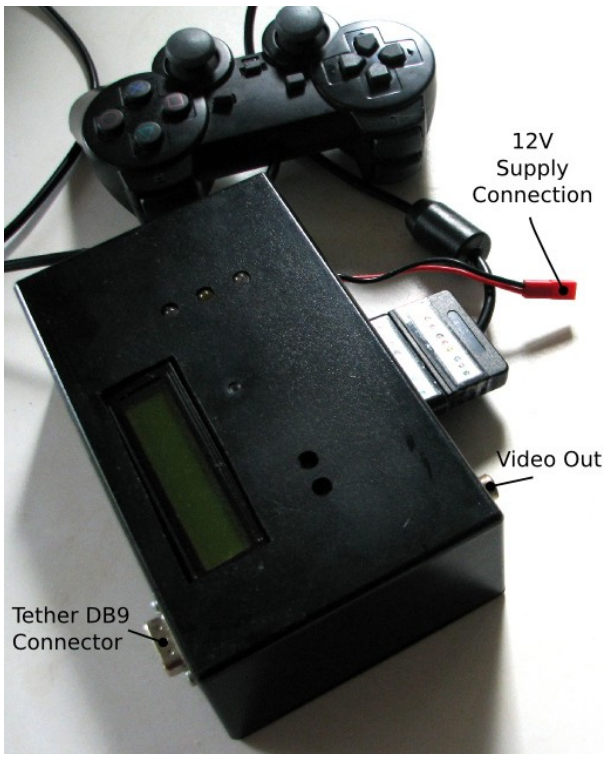
For the sake of completeness, some of the details from the previous editions have been repeated here.

A PS2 controller is used for the operator's input, and a 16x2 LCD display is use to show the battery voltage in the ROV, the temperature within the ROV, the heading and the depth. A series of three LEDs provide feedback on the probable status of the ROV headlights, whether the camera is recording or not and an indication that a photo has been triggered. Another two LEDs provide warning signals in the case of low battery voltage on board the ROV or too high temperature within the ROV.

The ROV and the topside system are connected via a 100m CAT5 tether. Communication between the Topside Arduino (Master) and the ROV Arduino (Slave) makes use of the ***EasyTransfer*** library by Bill Porter (http://www.billporter.info/).

As it happens, the video output from the Horyzon camera includes icons that report the status of the camera. It tells the operator whether it is in video mode, recording video, camera mode, or taking a photo. Because this is an accurate indication of what the camera is doing rather than the "best guess" indication provided by the topside Arduino and status LEDs, it is probably better to ignore the camera control LED indicators on the topside control box and just use the on-screen indicators.

The ROV uses HobbyWing EZRUN 18A ESCs which need to be programmed and calibrated before use in this system. The process to do this is described in this article: ***Programming and Calibrating the EZRUN 18A ESCs***.

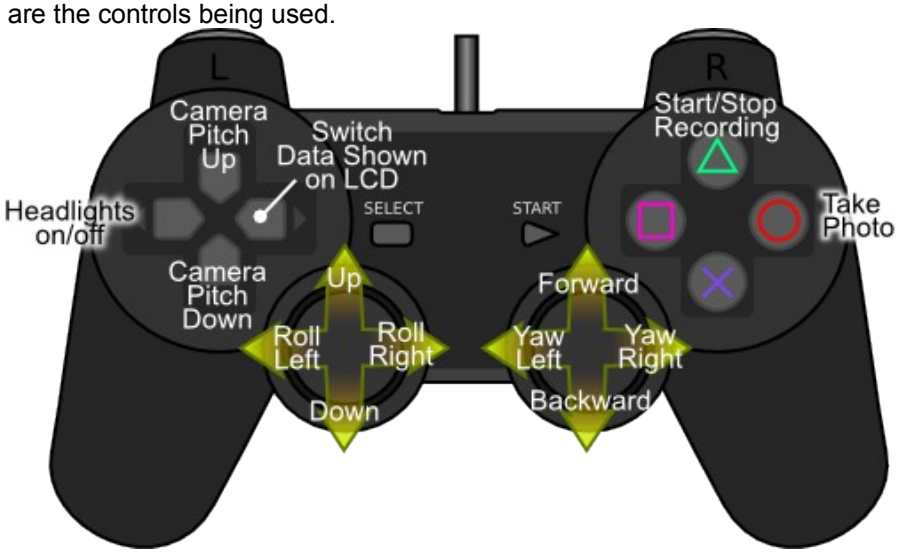


Controlling the ROV

The Playstation 2 controller is a nice cheap control interface with more buttons and controls than are required for this project. Like the previous editions of the ROV control sketches, the PS2 controller is used for controlling things on the ROV such as the motors, lights, and camera, as well as being able to flick through the data being displayed on the topside station's LCD display. This uses the ***PSX library*** from Bill Porter (http://www.billporter.info).

For more discussion on using the PS2 controller and Arduino to manipulate stuff, have a look at this page: PS2 Controller Sketch for ESCs and Stuff.

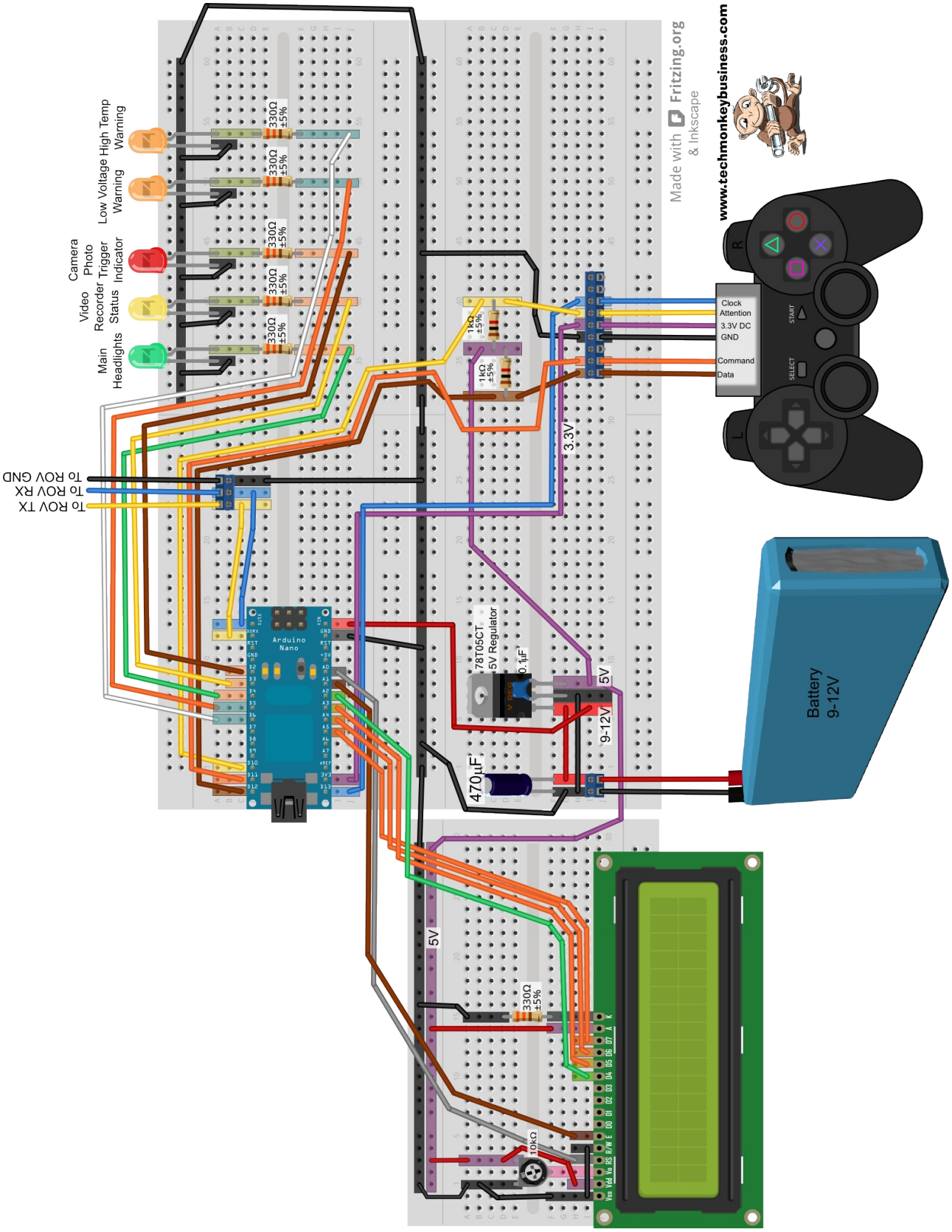
At present these are the controls being used.



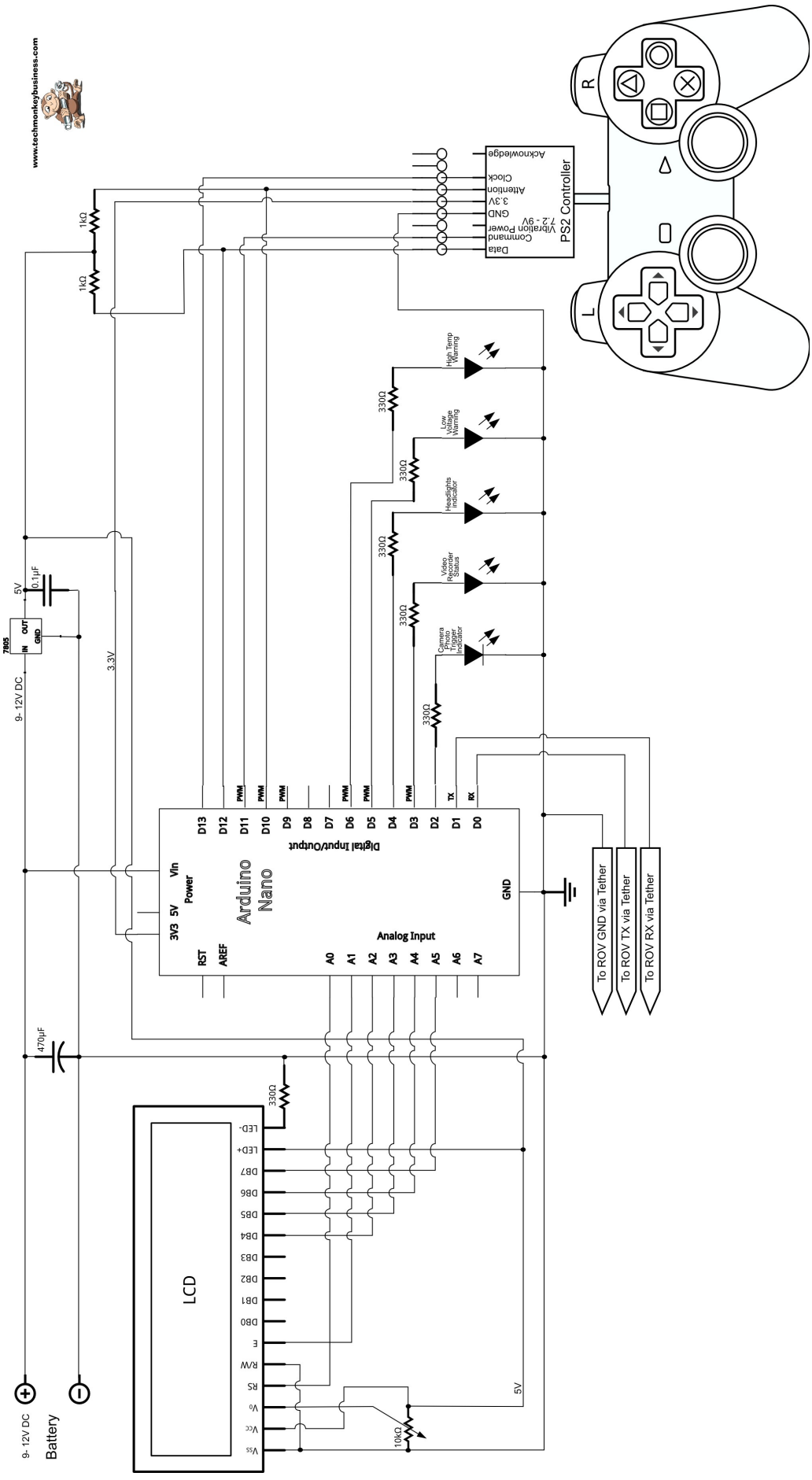
Topside Station Circuits

In addition to the PS2 Controller, the topside station makes use of an Arduino Nano and a 16x2 LCD Display. A ***78T05CT regulator*** is used to provide a 5V supply to the LCD Display and for the Command and Attention

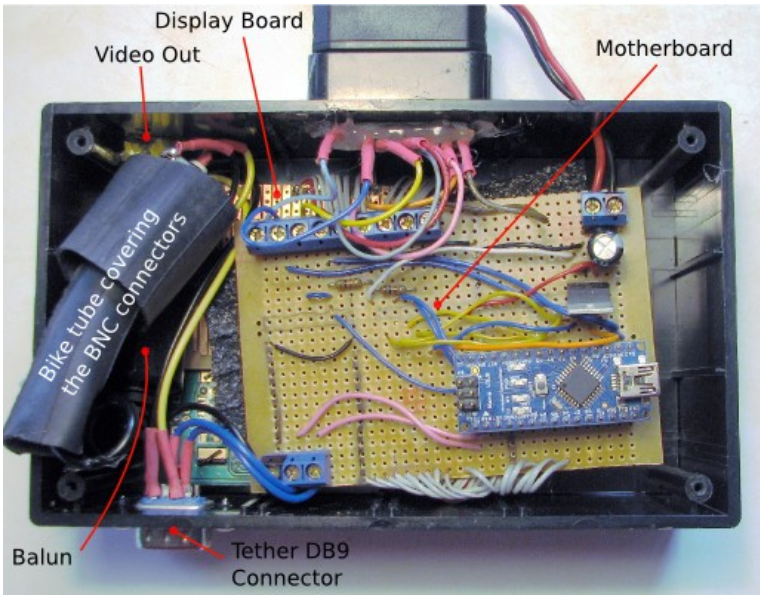
pins of the PS2 controller. The LCD library is part of the standard Arduino IDE, so you will not need to hunt around for this in order to get the Master sketch to work. You will need to find Bill Porter's EasyTransfer and PSX libraries. Both can be found on his website (http://www.billporter.info) or the editions used to develop this project can be found archived on this page – ***Arduino Library Collection***.



Or if you prefer a more classic circuit diagram.

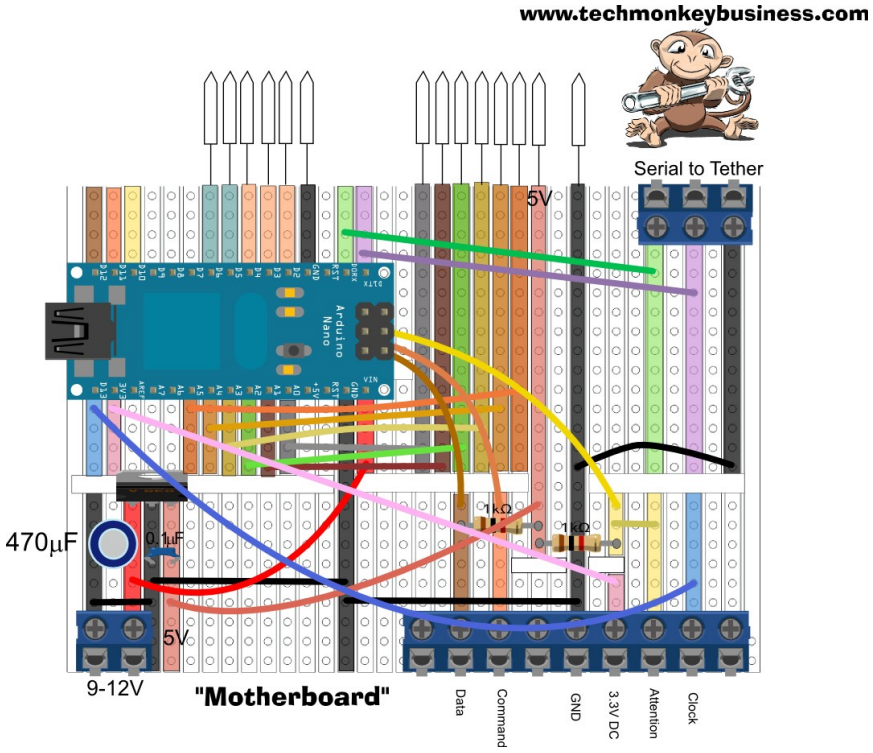


The topside station for my ROV was produced on strip-board and the end product looks like this. The ugly bit of bicycle tube on the left hand side surrounds the BNC connectors that I had on the baluns. Unfortunately Baluns with BNC connectors were the only type available to me here. The rubber tube helps keep the large all metal BNC components from touching any of the strip board tracks (which would let the smoke out of something if they did.)

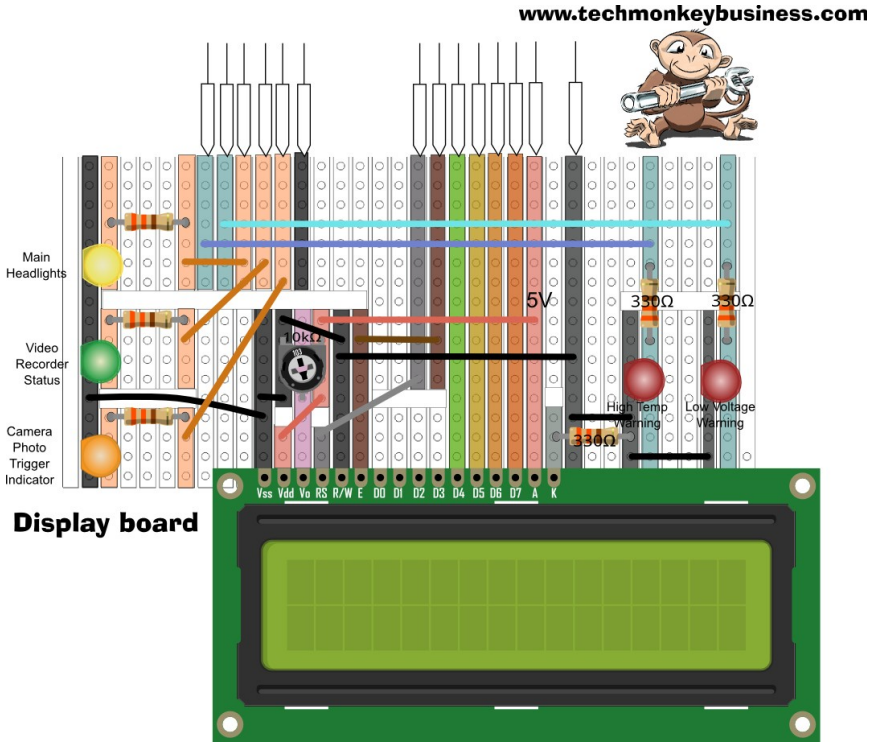


The two strip-board layouts are shown below. Hopefully the colours, layout and all that will make it pretty obvious which connections match each other on the two boards.

The Motherboard.

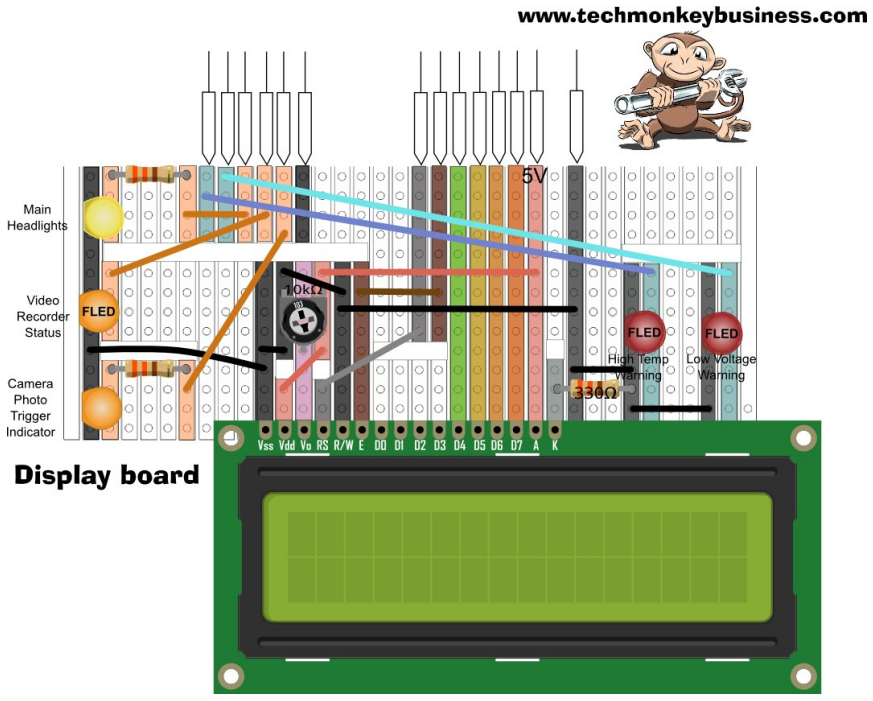


The Display Board



You can download the vectorgraphic files in an SVG format from here. ***ROV_Masterv5_Stripboards.zip***.

To keep the Arduino running quickly and efficiently, using ***Flashing LEDs*** for the warning lights and the video recording light avoids the extra processing required to flash normal LEDs. The strip-board layout shown below is a variation of the board shown above that makes use of Flashing LEDs (aka "***FLED***s").

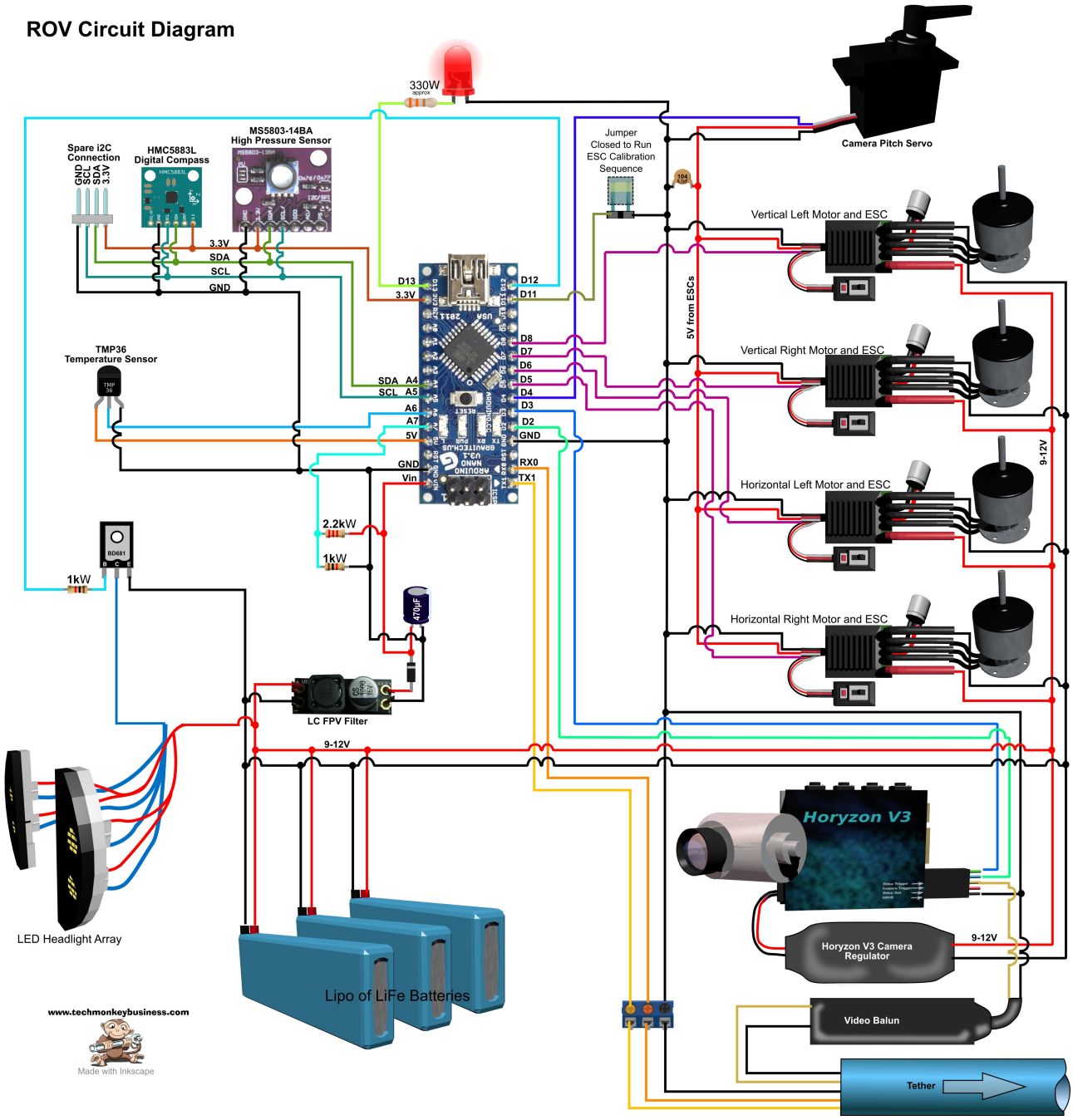


The ROV Circuits.

The ROV circuit is based on an Arduino Nano V3 which is communicating to the topside master through the tether using RS232 serial protocol. On board the ROV the Arduino is issuing servo commands to the thruster ESCs, and the camera pitch servo. Digital controls are used to switch on or off the headlights, and to control the camera modes. It is also monitoring the temperature inside the electronics pod using a ***TMP36 temperature sensor*** and measuring the battery voltage using a simple voltage divider. The new addition is a three port ***i2c*** bus for the HS5803-14BA depth sensor and the HMC5883L Digital Compass. The third ***i2c*** port is unused.

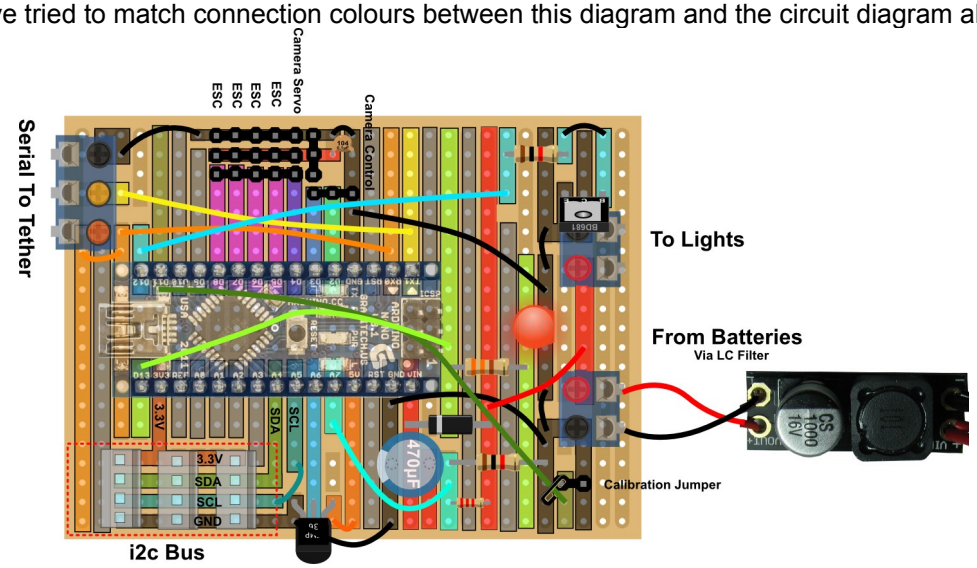
Below is the circuit diagram for the ROV circuit. You will notice the system includes an LC Filter suitable for FPV RC Aircraft systems to help protect the control board from voltage fluctuations from the main battery. The LC Filter is an off-the-shelf item from Banggood.

ROV Circuit Diagram



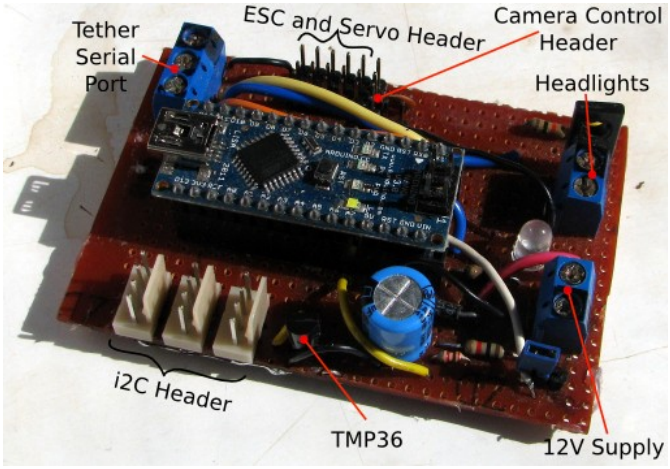
You can download the vectorgraphic version of this file in an ***.SVG*** format from here.
ROV_Slave_Circuit_Diagramv2.zip.

Below is the strip-board layout for the ROV control circuit for the sake of completion. Hopefully this makes sense. I have tried to match connection colours between this diagram and the circuit diagram above.



You can download an ***.SVG*** vectorgraphic version of this diagram to help make clear where the strip-board tracks have been cut. The vectorgraphic version can be downloaded from here.
ROV_Slave_Stripboard_Layoutv2.zip.

The end product looks like this.



The Sketches

The sketches make use of three 3rd party libraries that will need to be installed into the Arduino IDE.

- Bill Porter's PSX library which can be found here: http://www.billporter.info.
- And Bill Porter's EasyTransfer library which can be found on the same website.
- The MS5803-14BA Pressure sensor library from Luke Miller can be found here: http://github.com/millerlp

The remaining libraries for servos, i2c, LCD displays and such are all included with the standard Arduino IDE.

You can also download the libraries from the following webpage: ***A Collection of Arduino Libraries Used in This Project***.

The two sketches presented below can be downloaded from here:
ROVPS2Control_Sketches_Release4.zip

Topside Arduino Sketch - aka the "Master"

The commands from the PS2 controller are translated to actual motor speeds and directions on the Master rather than the slave. The messages back from the ROV are the ROV battery voltage, the internal temperature, depth, and heading. The EasyTransfer library makes it easy to include more data at a later stage.

```
/*
ROVPS2Control_Masterv8.ino
Hamish Trolove – 30 March 2016
www.techmonkeybusiness.com
This sketch takes control commands from a PS2 handset and transmits the
commands using Bill Porter's EasyTransfer Library over a 9600 baud serial
link (100m tether).

This sketch is designed for an Arduino Nano with only one Serial Port.

Pin assignments are:

3.3V output to PS2 red Pin
Pin D10 to PS2 yellow pin
Pin D11 to PS2 orange pin
Pin D12 to PS2 brown pin
Pin D13 to PS2 blue pin

Pin D2 to LED Camera Photo Trigger Indicator
Pin D3 to LED Camera Record Indicator
Pin D4 to LED Main Lights Indicator
Pin D5 to LED ROV Battery Low Voltage Warning
Pin D6 to LED ROV Interior high temperature warning

Communications
Serial Connection: Topside D1 (TX) to ROV D0 (RX)
Serial Connection: Topside D0 (RX) to ROV D1 (TX)
Connect the GND on both

A 16x2 LCD screen is connected as follows
  VSS to GND
  VDD to 5V output of MC78T05CT regulator
  VO to sweep arm of 10kohm variable resistor
  RS to Arduino Nano pin A0
  RW to GND
  E to Arduino Nano pin A1
  D4 to Arduino Nano pin A2
  D5 to Arduino Nano pin A3
  D6 to Arduino Nano pin A4
  D7 to Arduino Nano pin A5
  A to 5V output of MC78T05CT regulator
  K to GND via a 330ohm resistor

5V is supplied from a regulator to the 1Kohm pull up resistors
for PS2 as well as the LCD screen and it's backlight

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

The controls for the ROV are;
Left Stick - X-axis = Roll, Y-axis = Up/down
```

```
Right Stick - X-axis = Yaw, Y-axis = forward/back
Direction button pad left = LED Main lights On/Off toggle
Direction button pad up = turn camera upwards
Direction button pad down = turn camera downwards
Direction button pad right = Change reading on display
Triangle = Start/Stop video recording
Circle = Take photo




*/


#include <PS2X_lib.h> // Bill Porter's PS2 Library
#include <EasyTransfer.h> // Bill Porter's Easy Transfer Library
#include <LiquidCrystal.h>

PS2X ps2x;  //The PS2 Controller Class
EasyTransfer ETin, ETout;  //Create the two Easy transfer Objects for
                           // Two way communication

LiquidCrystal lcd(A0,A1,A2,A3,A4,A5);   //Pins for the LCD display

const int grnLEDpin = 4;  //green LED is on Digital pin 4
const int redLEDpin = 3;  //red LED is on Digital pin 3.
const int yelLEDpin = 2;  //yellow LED is on Digital pin 2
const int VwarnLEDpin = 5;  //Voltage warning LED is on Pin D5
const int TwarnLEDpin = 6;  //ROV temp warning LED is on Pin D6
const int LowBatVolts10 = 96;  //This is for holding the value of the
                           //Low Battery Voltage warning Voltage threshold x10.

int ForwardVal = 0;  //Value read off the PS2 Right Stick up/down.
int YawLeftVal = 0;  //Value read off the PS2 Right Stick left/right
int UpVal = 0; //Value read off the PS2 Left Stick up/down
int RollLeftVal = 0; // Value read off the PS2 Left Stick left/right
float ROVTMP = 0;  //Variable to hold the converted ROV interior temperature.
int DispOpt = 0; //Variable to signal which value to show on the display

long PhotoSignalRunTime = 0; //A variable to carry the time since photo
triggered.
volatile boolean PhotoActive = false;  // A flag to show that the camera signal
has been sent.

struct RECEIVE_DATA_STRUCTURE{
  int BattVolt;  //Battery Voltage message from the ROV.
  int ROVTemp; //ROV interior temperature back from the ROV
  int ROVDepth; //ROV depth reading (m)
  int ROVHDG;  //ROV direction (Degrees)
};

struct SEND_DATA_STRUCTURE{
  int upLraw;  //Variables to carry the actual raw data for the ESCs
  int upRraw;
  int HLraw;
  int HRraw;
  int CamPitch; //Angle of the camera servo.
  volatile boolean CamPhotoShot; // Camera photo trigger signal
  volatile boolean CamRec;  //Camera record function toggle
  volatile boolean LEDHdlts; //LED headlights on/off toggle
```

```
};

//give a name to the group of data
RECEIVE_DATA_STRUCTURE rxdata;
SEND_DATA_STRUCTURE txdata;

void setup()
{
  ps2x.config_gamepad(13,11,10,12, false, false);
  //setup pins and settings: GamePad(clock, command, attention, data,
Pressures?, Rumble?)
  //We have disabled the pressure sensitivity and rumble in this instance and
  //we know the controller type so we have not bothered with the error checks
  pinMode(grnLEDpin, OUTPUT);  //Sets the grnLEDpin to output
  pinMode(redLEDpin, OUTPUT);  //Sets the redLEDpin to output
  pinMode(yelLEDpin, OUTPUT);  //Sets the yelLEDpin to output.
  pinMode(VwarnLEDpin, OUTPUT);  //Sets the low voltage warning pin to output
  pinMode(TwarnLEDpin, OUTPUT);  //Sets the overtemperature warning pin to
output.
  txdata.CamRec = false;  //Sets the Camera default to not recording
  txdata.CamPhotoShot = false; //Sets the Camera default to no phototaken
  txdata.CamPitch =90;  //Sets the Camera Pitch to be level
  lcd.begin(16, 2);
  lcd.clear();  //make sure screen is clear.
  lcd.setCursor(0,0);  //Move cursor to top left corner
  lcd.print("Initialising");

  delay(10000);    //The 10 second delay to allow opportunity to upload new
programs.
  Serial.begin(9600); //Begin Serial to talk to the Slave Arduino
  ETin.begin(details(rxdata), &Serial); //Get the Easy Transfer Library
happening through the Serial
  ETout.begin(details(txdata), &Serial);
  lcd.clear();  //make sure screen is clear again.
  lcd.setCursor(0,0);  //Move cursor to top left corner
  lcd.print("Ready");

}

void loop()
{
  ps2x.read_gamepad(); //This needs to be called at least once a second
                       // to get data from the controller.
  if(ps2x.Button(PSB_PAD_UP))  //Pressed and held
  {
    txdata.CamPitch = txdata.CamPitch + 2; //increase the camera pitch
  }

  if(ps2x.ButtonPressed(PSB_PAD_LEFT))  //Pressed
  {
    txdata.LEDHdlts = !txdata.LEDHdlts; //Toggle the LED light flag
  }


  if(ps2x.Button(PSB_PAD_DOWN))  //Pressed and Held
  {
    txdata.CamPitch = txdata.CamPitch - 2; //decrease the camera pitch
  }
  txdata.CamPitch = constrain(txdata.CamPitch,20,160);  //Constrain the camera
pitch
```

```
     //to within range servo can handle.

  if(ps2x.Button(PSB_PAD_RIGHT))  //Pressed and Held
  {
    DispOpt = DispOpt + 1; //step through the data to display.
    if(DispOpt == 2) //At the moment there are only two items of
    //data to display.  This will need to be changed as extra data is added
    //This just resets the data to be displayed to the start of the list.
    {
      DispOpt = 0;
    }
  }

  if(ps2x.ButtonPressed(PSB_GREEN))  //Triangle pressed
  {
    txdata.CamRec = !txdata.CamRec; //Toggle the Camera recording Status
  }


  if(ps2x.ButtonPressed(PSB_RED))  //Circle pressed
  {
    txdata.CamPhotoShot = true;  //Set to indicate photo shot taken.
  }

//Analogue Stick readings
  ForwardVal = ps2x.Analog(PSS_RY);
  YawLeftVal = ps2x.Analog(PSS_RX);
  UpVal = ps2x.Analog(PSS_LY);
  RollLeftVal = ps2x.Analog(PSS_LX);

//Translate the Stick readings to servo instructions
//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 actual raw data for the ESCs
  txdata.upLraw = (128-UpVal)-(128-RollLeftVal)/2;  //This will be up to a value
of 192
  txdata.upRraw = (128-UpVal)+(128-RollLeftVal)/2;  //This will be up to a value
of 192
  txdata.HLraw = -(128-ForwardVal)+(128-YawLeftVal);  //This will be up to a
value of 256
  txdata.HRraw = -(128-ForwardVal)-(128-YawLeftVal);  //This will be up to a
value of 256

//Scale the values to be suitable for ESCs and Servos
// These values will be able to be written directly to the ESCs and Servos
  txdata.upLraw=map(txdata.upLraw,-193,193,0,179);
  txdata.upRraw=map(txdata.upRraw,-193,198,0,179);
  txdata.HLraw=map(txdata.HLraw,-256,256,0,179);
  txdata.HRraw=map(txdata.HRraw,-256,256,0,179);



  // Send the message to the serial port for the ROV Arduino
   ETout.sendData();

 //Based on Bill Porter's example for the Two Way Easy Transfer Library
 //We will include a loop here to make sure the receive part of the
 //process runs smoothly.
```

```
  for(int i=0; i<5; i++){
    ETin.receiveData();

    if(rxdata.BattVolt < LowBatVolts10)  //The factor of 10 is included to
    // match the factor of 10 used in the reported value which is an int
multiplied
    //by 10 to give 0.1 precision to the value.  Make sense?
    {
      digitalWrite(VwarnLEDpin,HIGH);  //If the battery voltage too low,
                                       //trigger the warning LED
    }
    else
    {
      digitalWrite(VwarnLEDpin,LOW);  //Otherwise if voltage above the
                                      //defined low voltage threshhold
                                      //leave the LED off.
    }
    ROVTMP = (rxdata.ROVTemp * 0.004882814-0.5)*100; //converts the 0-1024
                                                     //data value into
temperature.
    if(ROVTMP > 50)
    {
      digitalWrite(TwarnLEDpin,HIGH);  //If the Interior temp too high (over 50
degC),
                                       //trigger the warning LED
    }
    else
    {
      digitalWrite(TwarnLEDpin,LOW);  //Otherwise if interior temperature within
the
                                      //acceptable level, leave the LED off.
    }

    if(DispOpt == 1)
    {
      lcd.clear();  //A nice clean screen with no remnants from previous
                    //messages.
      lcd.setCursor(0,0); //Top left hand corner
      lcd.print("ROV Volts:");
      lcd.setCursor(0,1); //Bottom left corner
      lcd.print("ROV Temp:");
      lcd.setCursor(11,0);
      lcd.print(float(rxdata.BattVolt)/10,1); //factor of 10 used to get
        //extra precision from Integer value and then displayed to 1 decimal
place.
      lcd.setCursor(11,1);
      lcd.print(ROVTMP); // Display the ROV temperature
    }
    else
    {
      lcd.clear();  //A nice clean screen with no remnants from previous
                    //messages.      lcd.setCursor(0,0); //Top left hand corner
      lcd.print("Depth:");
      lcd.setCursor(0,1); //Bottom left corner
      lcd.print("Heading:");
      lcd.setCursor(11,0);
      lcd.print(rxdata.ROVDepth); //Display ROV depth in metres
      lcd.setCursor(11,1);
      lcd.print(rxdata.ROVHDG);  //Display ROV heading in degrees.
```

```
    }
    delay(18);
  }

// Signalling the probable status of the camera using LEDs.

  if(txdata.CamPhotoShot && !PhotoActive)
  {
    PhotoSignalRunTime = millis();  //Set the start time for the signal
    digitalWrite(grnLEDpin,HIGH);
    PhotoActive = true;  //record that the photo has been triggered
  }
  if(txdata.CamPhotoShot && PhotoActive && millis() - PhotoSignalRunTime > 2000)
//See if the trigger
 // signal has been running for two seconds
  {
    digitalWrite(grnLEDpin,LOW);
    txdata.CamPhotoShot = false; //Set the camera trigger to off
    PhotoActive = false;  // record that the photosignal has finished.
  }

  digitalWrite(redLEDpin,txdata.CamRec); //Light the redLED based on camera
recording status flag
  digitalWrite(yelLEDpin,txdata.LEDHdlts); //Light the LED based on headlights
status flag
  delay(18);
}
```

ROV Arduino Sketch - aka the "Slave"

Again the ROV sketch uses Bill Porter's EasyTransfer library, but other than that it is fairly straightforward. Hopefully the comments will explain it all.

```
/*
ROVPS2Control_Slavev8.ino
Hamish Trolove - 30 March 2016
www.techmonkeybusiness.com
This sketch takes commands sent to it from the Master unit with
the PS2 Controller attached and converts it to motor commands,
servo commands, light controls etc.  The data is sent from
the handset (Master) to the ROV(Slave) using Bill Porter's EasyTransfer
Library over a 9600 baud serial link (100m tether).
The MS5803_14 library is from Luke Miller http://github.com/millerlp

Data sent from the Master are raw settings for the ESC control.

This sketch is designed for an Arduino Nano with only one Serial Port.

The pin assignments are;
D13 = RED LED pin.
D12 = Headlight Control
D11 = Jumper pin
D8 = ESC Vertical Left
D7 = ESC Vertical Right
D6 = ESC Horizontal Left
D5 = ESC Horizontal Right
```

```
D4 = Camera Pitch Servo
D3 = Video Trigger
D2 = Photo Trigger

A7 = Voltage Divider connection
A6 = TMP36 temperature sensor output pin

i2c bus
GND pins on MS5803-14BA and BMP180 sensors to Nano GND pin
Vcc pins on MS5803-14BA and BMP180 sensors to Nano 3.3V pin
SDA pins on MS5803-14BA and BMP180 sensors to Nano A4 pin
SCL pins on MS5803-14BA and BMP180 sensors to Nano A5 pin

5V = Supply to the TMP36 temperature sensor.

Communications
Serial Connection: Topside D1 (TX) to ROV D0 (RX)
Serial Connection: Topside D0 (RX) to ROV D1 (TX)
Connect the GND on both

Please note that the ESCs will all have been programmed by this
point in the project.

The onboard voltage, heading, depth, and internal temperature
data is sent through the Serial link back to the Master
for display on a 16x2 LCD screen.

The heading is from an HMC5883L Digital Compass (i2c address 0x1E)
and the depth from a MS5803-14BA high pressure sensor (i2c address 0x76)

See also: HoryzonTrigger.ino, ROVPS2Control_Masterv0.ino,
ROVDoNothing.ino, ROVSubBv0.ino, DigitalCompassv2.ino,
PTLoggerv4.ino and TMP36_Temperature_Sensor.ino.


*/

#include <Wire.h>  //i2c library for the digital compass and depth sensor
#include <Servo.h>
#include <EasyTransfer.h> // Bill Porter's Easy Transfer Library
#include <MS5803_14.h> //Library for the MS5803-14BA

EasyTransfer ETin, ETout;  //Create the two Easy transfer Objects for
                            // Two way communication

MS_5803 sensor = MS_5803(512);

Servo ESCVL;  // Create Servo Object ESC Vertical Left
Servo ESCVR;  // Create Servo Object ESC Vertical Right
Servo ESCHL;  // Create Servo Object ESC Horizontal Left
Servo ESCHR;  // Create Servo Object ESC Horizontal Right
Servo CamAng; // Create Servo Object for the Camera Pitch Servo.

const int RedLEDpin = 13; // The indicator LED pin is 13.
const int HeadLts = 12; // The Headlight Control is on pin 12
const int CamRecTrig = 3; //Camera video recorder trigger is on pin D3
const int CamPhotTrig = 2; //Camera photo trigger is on pin D2

const int hmc5883Address = 0x1E; //0011110b, I2C 7bit address for compass
const byte hmc5883ModeRegister = 0x02;
```

```
const byte hmcContinuousMode = 0x00;
const byte hmcDataOutputXMSBAddress = 0x03;

volatile boolean CamRecd;  //Camera record function toggle
volatile boolean CamPhoto;  //Camera photo function toggle

const int Voltpin = A7; // analogue pin used to read the battery voltage
const int Temppin = A6; // analogue pin used to read the TMP36 Temp sensor
//Analogue pins A4 and A5 are taken by the i2c bus.

int volts;    // variable to read the voltage from the analog pin
int x,y,z; //triple axis data for the digital compass.
int angle; //calculated horizontal heading angle.

float MS5803Press;  //Pressure from the MS5803 Sensor.
float MS5803Temp;  //Temperature from the MS5803 Sensor.

const float RefVolts = 5.0; // 5-Volt board reference voltage on Nano
const float ResistFactor = 319.68; //Calculated from 1023.0*(R2/(R1 + R2)
     //where R1 = 2200 ohms and R2 = 1000 ohms for a 15V max voltage.
long TriggerHoldTm = 0;  // the time since the camera button was triggered
long TriggerHoldDuration = 150;  //The time in milliseconds to hold the camera
triggers LOW.

struct RECEIVE_DATA_STRUCTURE{
  int upLraw;  //Variables to carry the actual raw data for the ESCs
  int upRraw;
  int HLraw;
  int HRraw;
  int CamPitch; //Angle of the camera servo.
  volatile boolean CamPhotoShot; // Camera photo trigger signal
  volatile boolean CamRec;  //Camera record function toggle
  volatile boolean LEDHdlts; //LED headlights on/off toggle
};

struct SEND_DATA_STRUCTURE{
  int BattVolt;  //Battery Voltage message to the Master.
  int ROVTemp; //ROV interior temperature back to Master
  int ROVDepth; //ROV depth reading (m)
  int ROVHDG;  //ROV direction (Degrees)
};

//give a name to the group of data
RECEIVE_DATA_STRUCTURE rxdata;
SEND_DATA_STRUCTURE txdata;



void setup()
{
  pinMode(RedLEDpin,OUTPUT);
  pinMode(HeadLts,OUTPUT);
  pinMode(CamRecTrig,OUTPUT);
  pinMode(CamPhotTrig,OUTPUT);

  digitalWrite(HeadLts, LOW); //Set the Headlights to Off
  CamRecd = false;  //Sets the Camera default to not recording
  CamPhoto = false; // No photos triggered.
  digitalWrite(RedLEDpin,LOW);
  digitalWrite(CamRecTrig,HIGH); //Both camera functions are controlled
```

```
  digitalWrite(CamPhotTrig,HIGH); // by making the pin low.

  ESCVL.attach(8,600,2250); //attach the ESCVL to pin 8
  ESCVR.attach(7,600,2250); //attach the ESCVR to pin 7
  ESCHL.attach(6,600,2250); //attach the ESCHL to pin 6
  ESCHR.attach(5,600,2250); //attach the ESCHR to pin 5
    //Due to problems with the ESC recognising the maximum
    //position at the default settings, the figures after
    //the pin number are the microsecond signals for the
    //minimum and maximum that the ESC will recognise.
    // 600 and 2250 work.
  CamAng.attach(4); //Attach the camera Pitch Servo to pin 4

//  throttle = 90;  //Set throttle to the neutral position.
  ESCVL.write(90);  //Set the ESCVL signal to the neutral position.
  ESCVR.write(90);  //Set the ESCVL signal to the neutral position.
  ESCHL.write(90);  //Set the ESCVL signal to the neutral position.
  ESCHR.write(90);  //Set the ESCVL signal to the neutral position.
  CamAng.write(90); //Set the camera servo pitch to be level.

  Wire.begin(); // Start the i2c communication

  //Initialise the Digital Compass
  Wire.beginTransmission(hmc5883Address);  //Begin communication with compass
  Wire.write(hmc5883ModeRegister);  //select the mode register
  Wire.write(hmcContinuousMode); //continuous measurement mode
  Wire.endTransmission();

  // Initialize the MS5803 sensor.
  sensor.initializeMS_5803();

  delay(10000);   //Ten second delay
    //The ESC should now be initialised and ready to run.

  Serial.begin(9600); //Begin Serial to talk to the Master Arduino
  ETin.begin(details(rxdata), &Serial); //Get the Easy Transfer Library
happening through the Serial
  ETout.begin(details(txdata), &Serial);


  //The camera starts in record mode probably due to Arduino startup signals
  //and so this needs to be stopped.  The sequence below sends a toggle to
  //the camera to stop it from recording.  obviously this will leave a small
  //waste video file, but we will need to live with that.

  digitalWrite(CamRecTrig,LOW); //Trip the photo trigger.
  delay(100);
  digitalWrite(CamRecTrig,HIGH);

}

void loop()
{
  // Send the message to the serial port for the ROV Arduino
  ETout.sendData();

 //Based on Bill Porter's example for the Two Way Easy Transfer Library
 //We will include a loop here to make sure the receive part of the
 //process runs smoothly.
  for(int i=0; i<5; i++){
```

```
    ETin.receiveData();
    // We'll do something properly with the returned data at a later s
    ESCVL.write(rxdata.upLraw);  //Set the ESCVL signal to the defined throttle
position.
    ESCVR.write(rxdata.upRraw);  //Set the ESCVR signal to the defined throttle
position.
    ESCHL.write(rxdata.HLraw);  //Set the ESCHL signal to the defined throttle
position.
    ESCHR.write(rxdata.HRraw);  //Set the ESCHR signal to the defined throttle
position.
    CamAng.write(rxdata.CamPitch); //Set the camera servo pitch to the defined
angle.
    digitalWrite(HeadLts,rxdata.LEDHdlts);//Light the headlights based on the
Message data
    delay(18);
  }

//The camera settings are status flags so we will need to trigger the events
based on
//changes in the data.
  if(rxdata.CamRec && !CamRecd)  //If the signal is to trigger video recording
  //and the Camera is not already triggered drop the camera recording pin to
LOW.
  {
    CamRecTrigger(); //Run the camera triggering signal
    CamRecd = true; //update the flag.
  }
  if(!rxdata.CamRec && CamRecd)  //If the camera no longer needing to trigger
  // then signal the camera and turn off the flag.
  {
    CamRecTrigger(); //Run the camera triggering signal
    CamRecd = false; //update the flag.
  }

  if(rxdata.CamPhotoShot && !CamPhoto)  //If the camera is required to fire a
shot trigger
  // the camera photo pin.
  {
    digitalWrite(CamPhotTrig,LOW); //Trip the photo trigger pin.
    TriggerHoldTm = millis();  //Reset the time that a camera trigger was used.
    CamPhoto = true; //update the flag.
  }

  if(!rxdata.CamPhotoShot && CamPhoto)  //If the camera photo signal ceases
  // reset the camera flag.
  {
    CamPhoto = false; //update the flag.
  }


  if(millis() - TriggerHoldTm > TriggerHoldDuration)  //If camera button held
long enough release it.
  // hopefully this little routine will speed up the sketch processing.
  {
    digitalWrite(CamRecTrig,HIGH);
    digitalWrite(CamPhotTrig,HIGH); //Just make them both inactive
  }


  delay(18);  //This delay is added to give the ROV a chance to
```

```
                //return data
  volts = analogRead(Voltpin)/ResistFactor*RefVolts*10;  //Read the voltage
  //from the battery through the voltage divider.  Factor of 10 used
  //to help achieve an integer with 0.1V accuracy.
  txdata.BattVolt = volts; //Send back the onboard battery voltage.
  txdata.ROVTemp=analogRead(Temppin); //This reads the pin keeps it as a 0-1024
value.

  //Read the digital compass
  //Tell the HMC5883L where to begin reading the data
  Wire.beginTransmission(hmc5883Address);
  Wire.write(hmcDataOutputXMSBAddress);  //Select register 3, X MSB register
  Wire.endTransmission();

  //Read data from each axis
  Wire.requestFrom(hmc5883Address,6);
  if(6<=Wire.available())
  {
    x = Wire.read()<<8; //X msb
    x |= Wire.read();   //X lsb
    z = Wire.read()<<8; //Z msb
    z |= Wire.read();   //Z lsb
    y = Wire.read()<<8; //Y msb
    y |= Wire.read();   //Y lsb
  }

  angle = atan2(-y,x)/M_PI*180;
  if (angle < 0)
  {
    angle = angle + 360;
  }
  txdata.ROVHDG = angle;  //ROV direction (Degrees)

  //Reading and MS5803-14BA Sensor
  // Use readSensor() function to get pressure and temperature reading from the
MS5803.
  sensor.readSensor();
  MS5803Press = sensor.pressure();  //Pressure in mbar absolute
  MS5803Temp = sensor.temperature();  //Although we have gathered this
  //it won't be used at this stage.

  txdata.ROVDepth = (MS5803Press-1013)/98.1; //ROV depth reading (m)

}


void CamRecTrigger()
{
    digitalWrite(CamRecTrig,LOW); //Trip the recorder toggle.
    TriggerHoldTm = millis();  //Reset the time that a camera trigger was used.
}
```