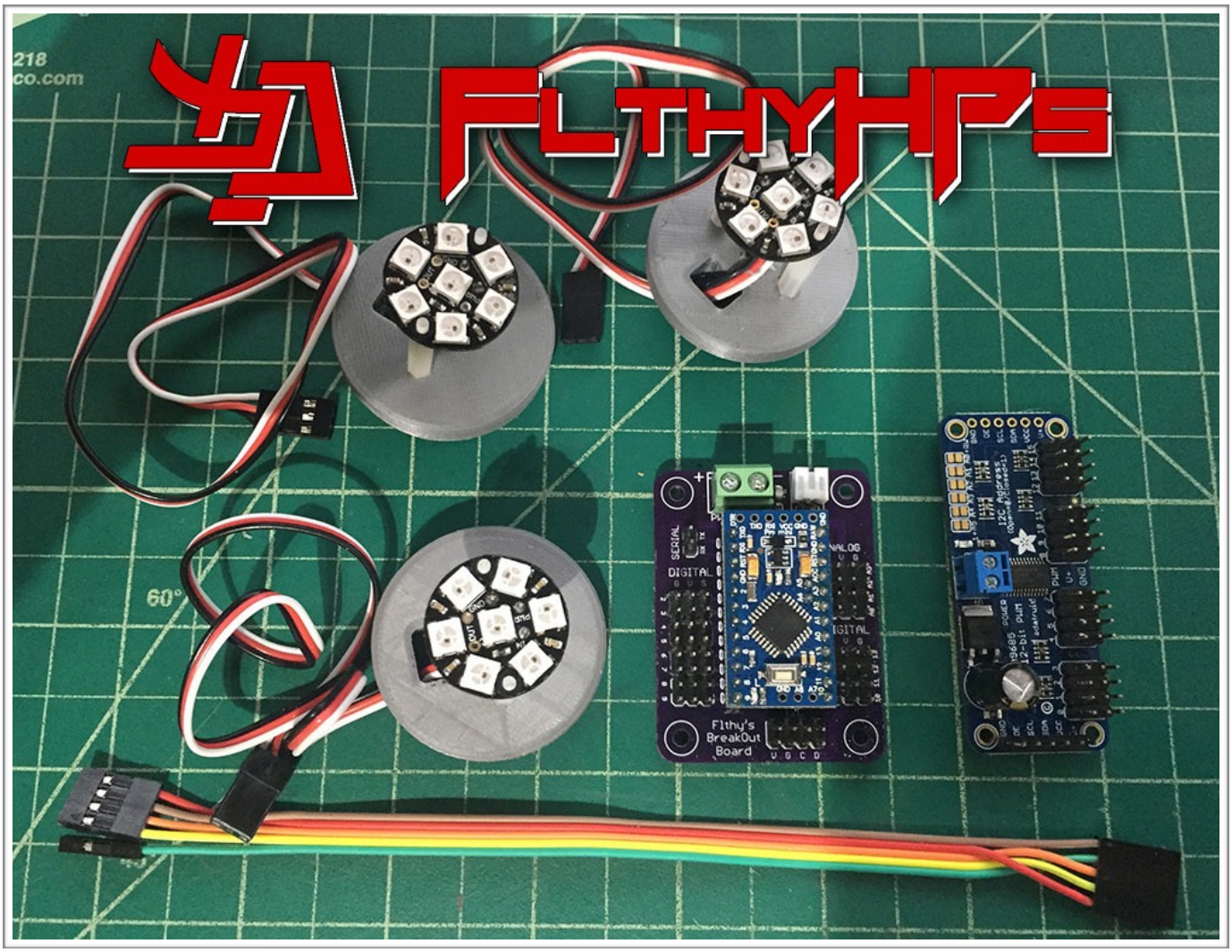
Guide to Using Your FlthyHPs v1.4

by Ryan Sondgeroth

flthymcnsty



updated

June 20th, 2017

Table of Contents

What are the FlthyHPs

What do they do and Why you want a set!

This system and sketch combines the movement and display functions of each Holoprojector into a single easy to control sketch. Instead of using a single RGB like other HP systems, the FlthyHPs utilizes a 7 LED NeoPixel board inside each HP to produce a more life-like representation of a hologram projection.

Servo control of the system is handled by a Adafruit 16 Channel I2C Breakout board, giving increased flexibility over the I2C bus while providing significant current to safely run your HP servos.

This system also allows us to operate the HP Servos with NeoPixel LEDs from the same sketch, a problematic solutions when attempting to use the stock Arduino servo library with Adafruit's NeoPixel library. Big Happy Dude's Slow Servo Library is much better any way, Thanks Brad!

The system incorporates semi randomized auto twitch sequences for both the LEDs and Servos of each HP giving your droid the illusion of autonomy. In addition to these free running auto modes, the LED and Servo behavior can be controlled in unison or separately through a variety of sequences triggered via I2C and Serial communication protocols. To simplify control, the FlthyHP system allows you to control HP functionality by sending all HP commands to a single device.

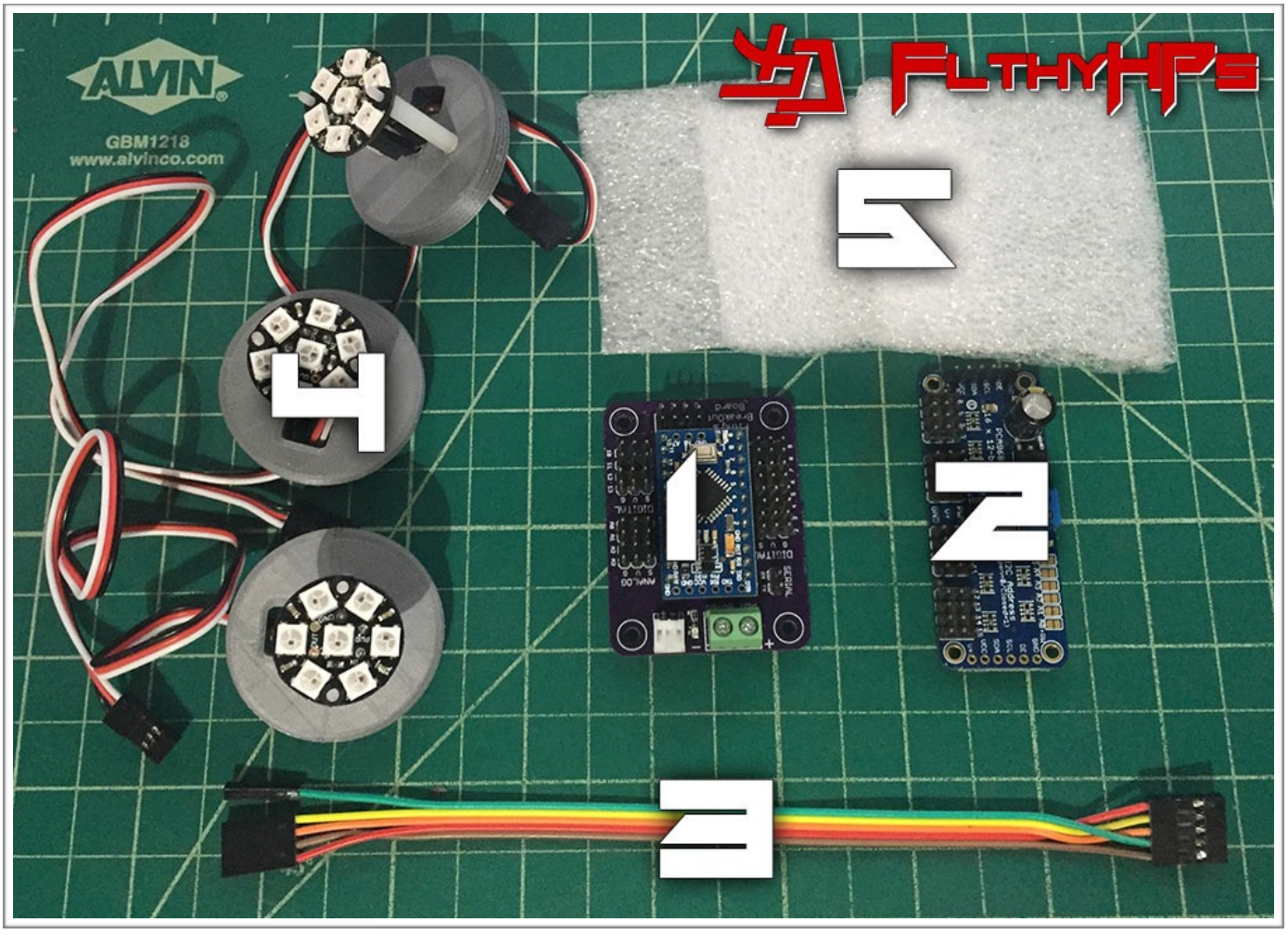
The most current version of this manual can be found at:
http://2geekswebdesign.com/FlthyHPs/Manual/

Rhyno45 is a Wanker, in case Ya'll didn't know!

What's in the Kit

Below is a picture of the complete contents of your HP kit. Please make sure everything is included before you begin. A brief description of each part/assembly is included below.

Each system comes pretested and pre-installed with the current sketch and default settings.



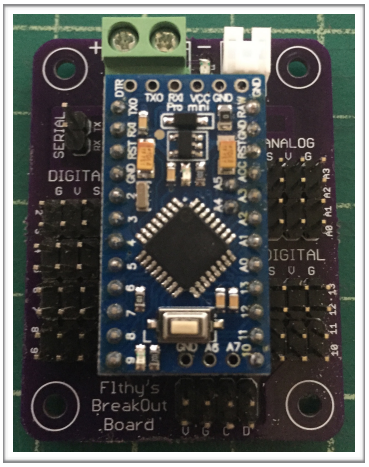
1. Arduino Pro Mini (5v 16MHz) on a Flthy Breakout Board.

The Flthy Breakout Board is a custom designed PCB* which conveniently breaks out the digital and analog pins of the Arduino Pro Mini into easy to use connections. It also breaks out the Serial Tx/Rx and I2C Clock/Data pins.

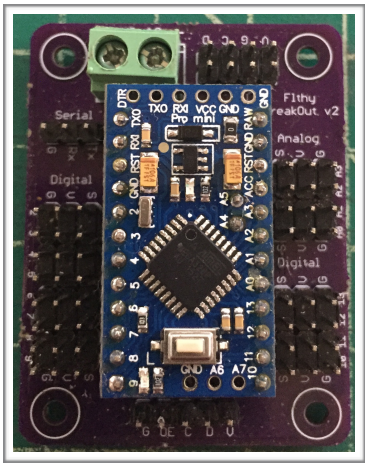
The boards include a screw terminal for connecting your 5v power. The newer v2 & v2.1 boards also break out a header row specifically for connecting to the Servo breakout board discussed below in item 2.

There has been three versions of this board, with v2.1 incorporating slight design improvements. Sketches work on either board version.

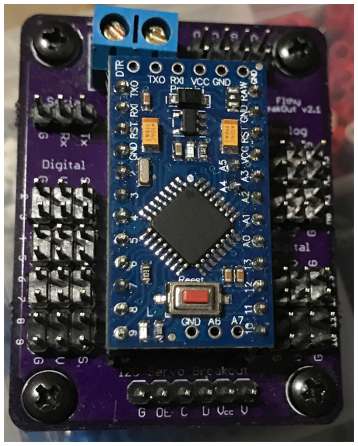
**The breakout board designs are based on the publicly CADEagle files available on letsmakerobots.com.*



Original Board (v1)



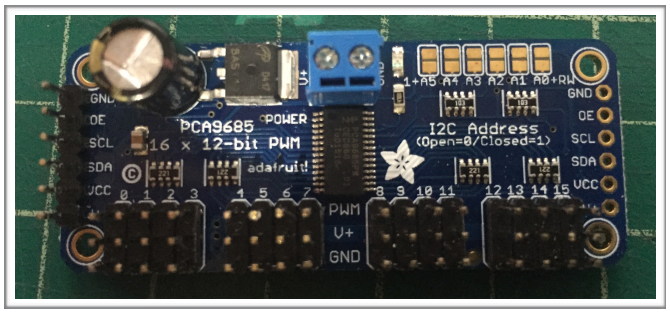
v2 Board



v2.1 Board

**The v2.1 breakout board incorporates a few design improvements making connections to it much easier.*

2. Adafruit 16 Channel I2C Servo Breakout board*



See product info at

https://www.adafruit.com/product/815

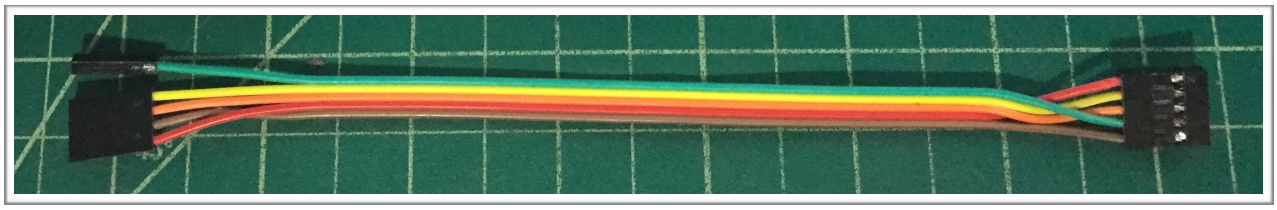
**Optional…Because some R2 Builders may not wish to articulate their HPs, some kits are available without this board.*

3. 6 Wire Jumper Cable

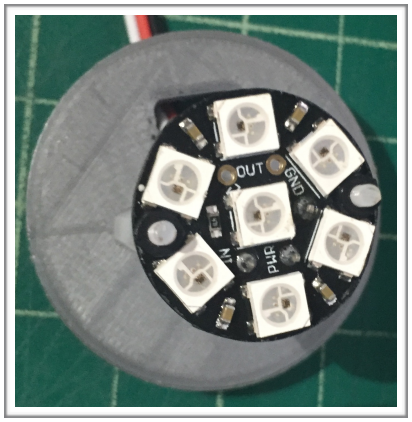
For kits using the newer v2.1 Break Out Board, this is simply a basic straight **6 wire** jumper cable as the header pins are broken out in the same arrangement as they appear on the Servo board including the V line. This was done to simplify the connection process.

For kits using the v2 Break Out Board, this is simply a basic straight 5 wire jumper cable as the header pins are broken out in the same arrangement as they appear on the Servo board minus the V line. This was done to simplify the connection process.

For kits using the v1 Breakout Board, a pre-wired jumper cable is used to connect the Servo Breakout Board to the Arduino. It has the wires swapped correctly for proper connection.



4. HP LED Assemblies (Qty. 3)

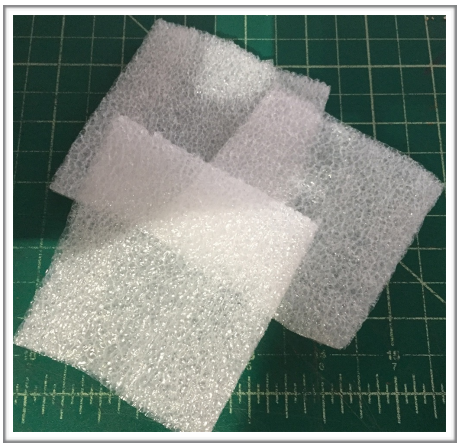


Each assembly consists of a Adafruit NeoPixel Jewel 7 LED Board (https://www.adafruit.com/products/2226), mounted on a custom 3D printed backplate via M2 nylon standoffs and connected with a 30cm 3 wire lead.

There are now 3D printed back plates available for the recent Pwrsrce (1 5/8"-20 threads) aluminum halo-projector run as well as those for the BobC HPs (1 5/8"-24 threads).

The center hole comes tapped for an M4 patterned bolt.

5. 1/4" Micro Foam Packaging Wrap (Qty. 3)



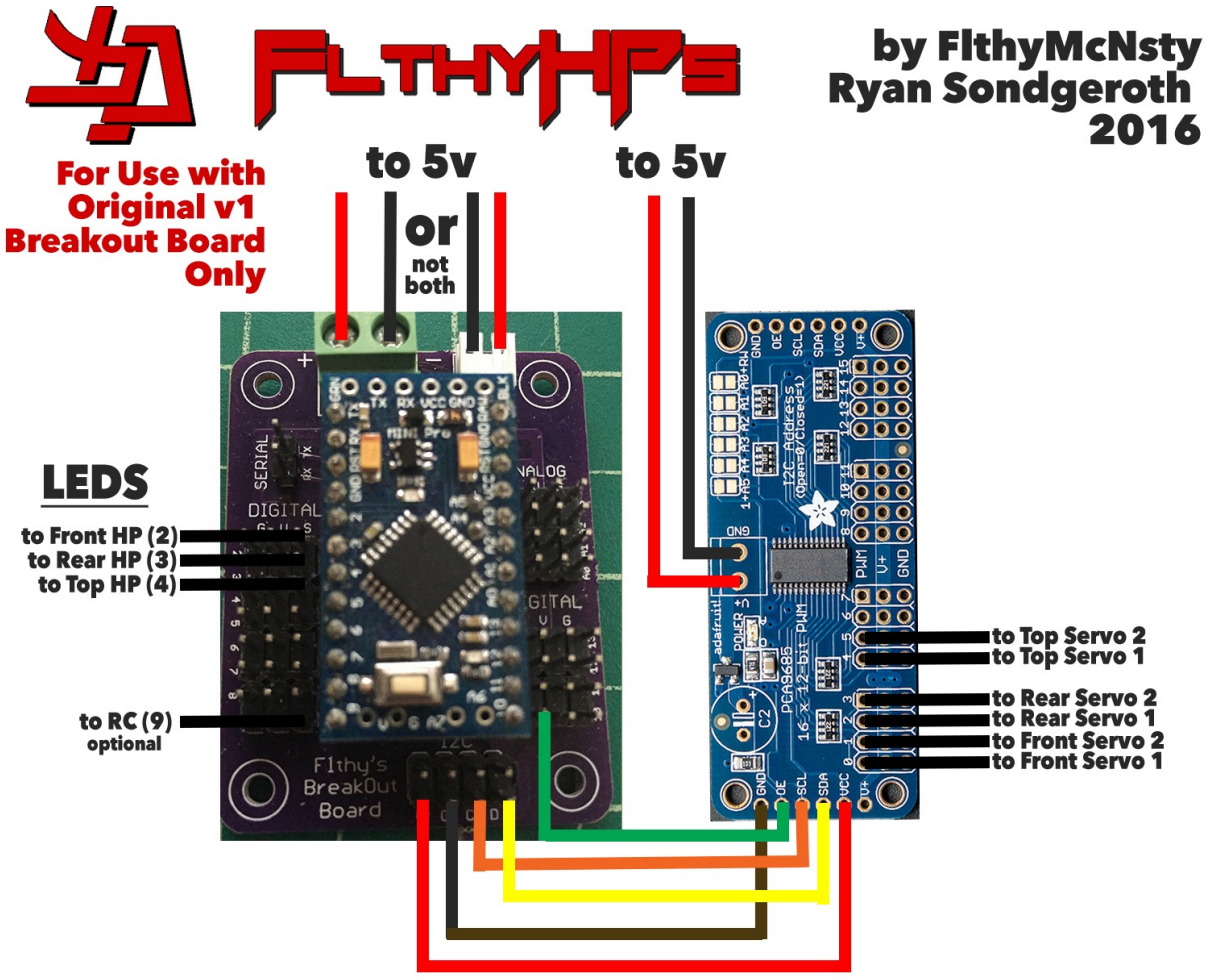
These 3 pieces can serve as a light diffuser between the HP's LEDs and its lens, just cut to size. They help blur the lines between the individual LEDs while maintaining a multicolor display. The foam construction also give the HP's light a unique texture when looking directly into the lens. (Which I don't recommend you do for long periods off time)

Hooking Up Your v1 HPs

What Wires Go Where?

Wiring Diagram for kits with Original v1 Breakout Boards

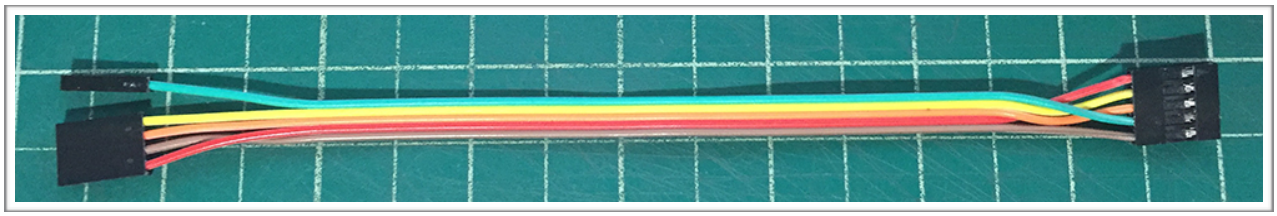
The wiring for this system is pretty straight forward. Both boards take 5v with the servos and LEDs attached as shown.



Arduino Sketch Located at:
http://2geekswebdesign.com/FlthyHPs/FlthyHPs.ino

Attention for Kits Using the Original v1 Breakout Board!

Pay special attention to the wiring between the v1 Arduino breakout board and the servo breakout board. The servo breakout board uses a different pin layout than the standardized I2C pin layout used by other systems created by R2 Builders. Incorrectly connecting the servo board can result in damage. In order to help prevent this occurrence, each kit is provided with a 5 wire jumper cable (pictured below) which has been rearranged into the correct wiring pattern with each wire matching the color used in the wiring diagram above.



Custom Wire for the Original v1 Board Design

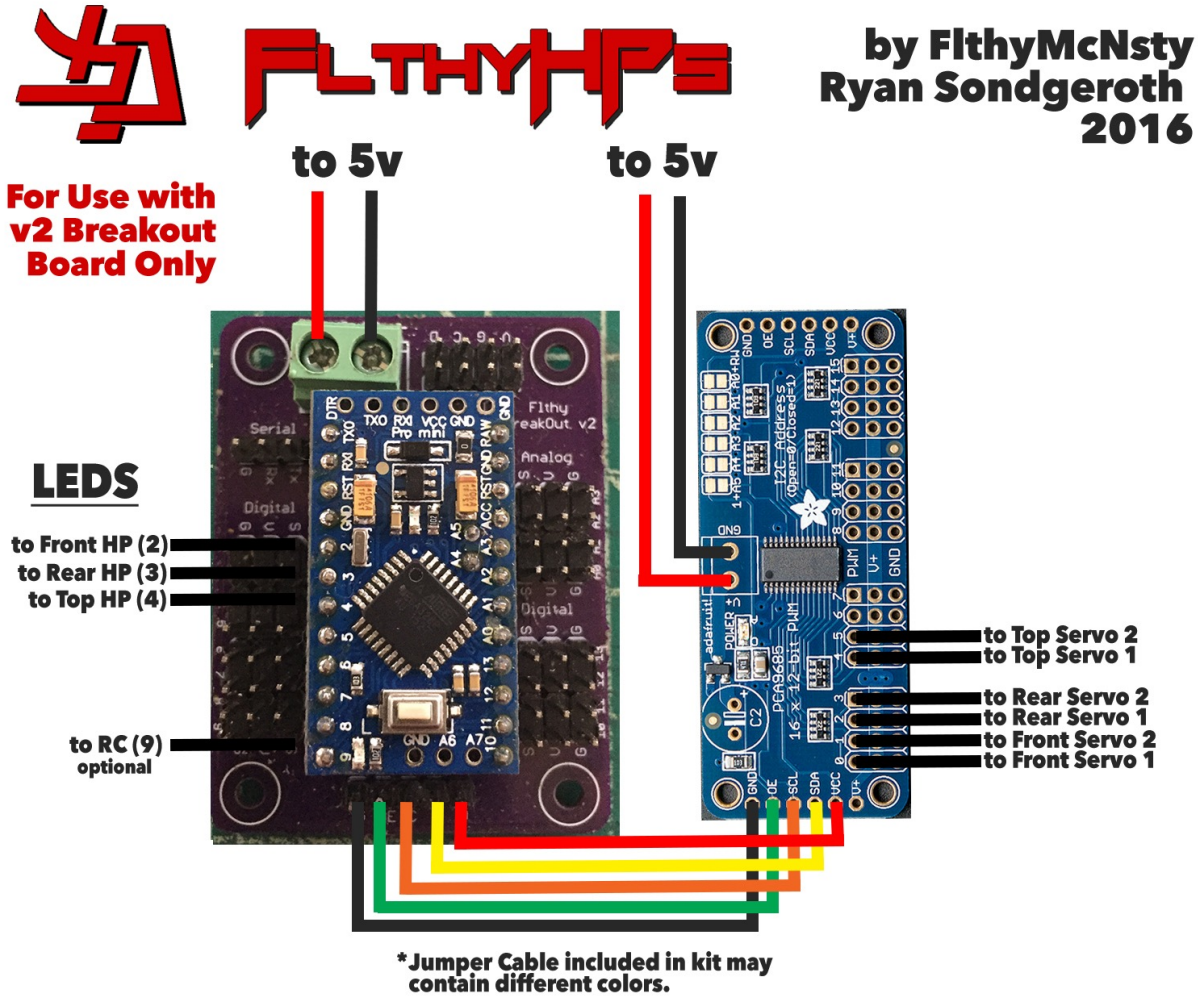
The v2 breakout board breaks out the pins in the same order as the servo board, simplifying the connection, and minimizing the confusing wiring.

Hooking Up Your v2 HPs

What Wires Go Where?

Wiring Diagram for kits with v2 Breakout Boards

The wiring for this system is pretty straight forward. Both boards take 5v with the servos and LEDs attached as shown.



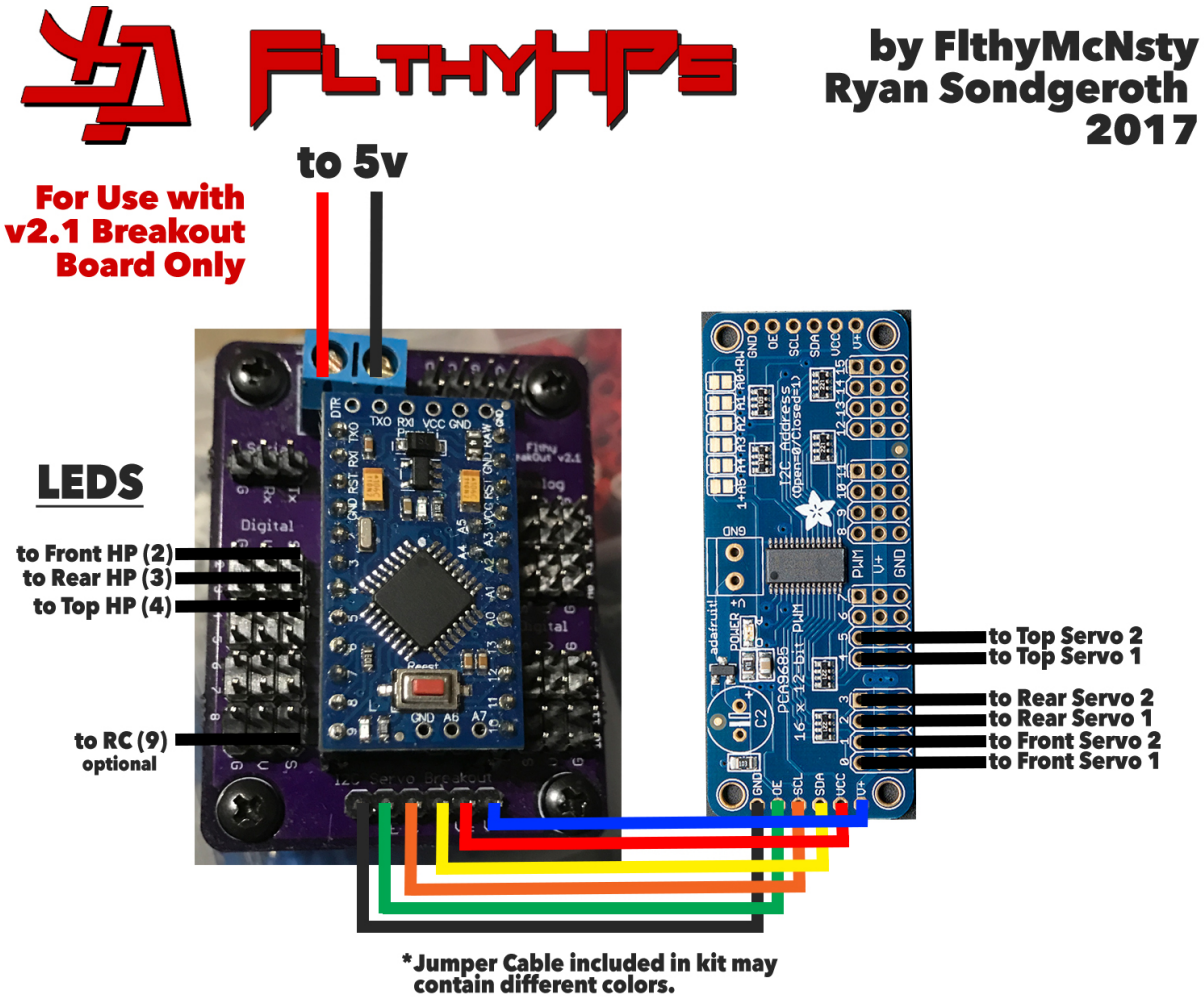
Arduino Sketch Located at:
http://2geekswebdesign.com/FlthyHPs/Sketches/

Hooking Up Your v2.1 HPs

What Wires Go Where?

Wiring Diagram for kits with v2.1 Breakout Boards

The wiring for this system is pretty straight forward. Both boards take 5v with the servos and LEDs attached as shown but the v2.1 now includes a V pin on the Servo Board Pin Out so you do not need to provide separate power to the I2C servo breakout board.



Arduino Sketch Located at:
http://2geekswebdesign.com/FlthyHPs/Sketches/

Understanding the v1.4 Sketch Settings

Explanation of User Definable Settings in the v1.4 Code

I keep the most current versions of my FlthyHP sketch freely available on my web server at: http://2geekswebdesign.com/FlthyHPs/Sketches

For those who use my sketch and feel you may have a way to improve upon it, please share your ideas and thoughts with me. Together we can continue to improve our droids.

As you review the sketch, you will see I have broken out a number of settings into user definable variables in order to let each user tweak the system to their specific needs. Below is a complete list of each of these variables with a brief explanation of its function and purpose.

1. I2C Address of the System

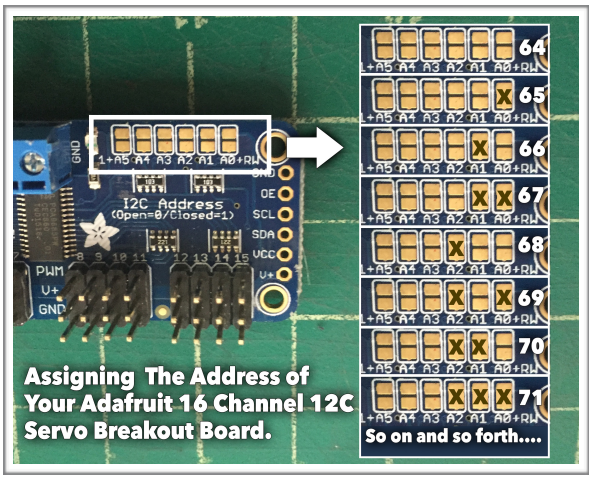
Since each device on the I2C bus can and must be assigned its own unique address, this variable allows you to do so. An I2C address of 25 has been standardized for other HP devices so I used it as well. It can be changed with the following variable. Please note that it is set using the hexadecimal equivalent of the address's decimal value.

```
#define I2CADDRESS 0x19  // 25 in Hexadecmal
```

2. I2C Address of the Servo Board

Similarly, we need to set a unique I2C address for the Adafruit Servo Breakout Board.

Unlike our Arduino based control board, the Servo Breakout board is only capable of using an I2C address in the range between 64 and 126. The specific address the board uses can be set using the set of 6 solder pads located on the top of the PCB. By default the board has an address of 64, and no changes are needed if you are not using the same board elsewhere on the I2C bus or have assigned another device with this address.



However, if you do need to make changes to the boards address you can do so by using the solder pads to select a new address value. To do so, you simply place a bead of soldier across the gap in the requisite pad pairs.

The address value is determined just as you would using binary as show in this image.

Once you have set your Servo board's address, you will

then need to change this address in the Sketch. It can be changed with the following variable. Again, please note that it is set using the hexadecimal equivalent of the address's decimal value.

```
#define SERVOI2CADDRESS 0x40    // 64 in Hexadecmal
```

If there is no need to reassign the Servo board's address, the default value in the sketch can be left as is.

3. HP LED Pin Assignment

This variable allows you to set which of the Arduino's digital pins are used to connect each of the 3 LED displays. The first value being the front HP, the second the rear HP and the third the top HP. Under most situations, the default values can be left unchanged.

```
const uint8_t LEDpins[HPCOUNT]  = {2,3,4};  // {Front, Rear, Top}
```

4. Servo Board Pin Assignment

This variable allows you to set which of the Servo breakout board's pins are used to connect each of the 6 servos. The first value pair being for the front HP, the second value pair for the rear HP and the third pair for the top HP. Under most situations, the default values can be left unchanged.

```
// Front HP Pins, Rear HP Pins, Top HP Pins
const uint8_t HPpins[HPCOUNT][2] = {{0,1},{2,3},{4,5}};
```

Important: *It is recommended that you remain consistent with how you orient your HP mounts and control system so that the two servos for each HP are positioned in a similar manner with the others. Likewise, pin assignments/servo connection should be consistent as well between the HPs. This will help keep their behavior during movements consistent.*

5. Output Enable (OE) Pin Assignment

The servo breakout board is equipped with Output Enabled (OE) capabilities. Output Enable is used to disable/enable power to the servos between movements by pulling the OE pin on the servo break outboard to LOW. This is used to minimize the unnecessary and unwanted hum/noise often heard when a servo fails to reach it's exact destination after a movement command is executed. Under most situations, the default value can be left unchanged.

```
#define OUTPUT_ENABLED_PIN 10
```

6. Enable/Disable OE

In some rare cases* you may wish to not use the Output Enable capabilities. You can disable OE by setting the following variable to 0.

```
#define OUTPUT_ENABLED_ON true
```

** The only reasons you would want to disable this is if…*

1. *You wish to use an additional micro-controller to send servo control commands via the Adafruit library.*
2. *You want to use additional servos on the same servo breakout board as your HP servos.*

Neither scenario is recommended because the increased complexity it introduces far outweighs the costs savings of not purchasing an additional servo board.

7. RC Input Pin Assignment

This variable allows you to set the digital pin used to feed a single RC input into the system to be read when the system's RC function has been triggered via Serial or I2C command. The RC function allows you to move a selected HP up or down using one of your controllers sticks. At this time the system only supports movement on a single axis and its use is completely optional. Under most situations, the default values can be left unchanged.

```
#define RCPIN 9
```

8. RC Pulse Width Range

Since RC controllers/PWM sources are different, this variable allows you to set a min and max pulse width value to map to the end positions of the up/down axis of movement. Which value corresponds with up and which corresponds with down is dependent on your specific set up so you may need to switch the order of the two values.

```
const int RCRange[2] = {1250, 1750};    // {Min, Max}
```

9. Adafruit Jewel Board Version

The Adafruit Jewels now come in RGB and RGBW versions, uncomment the following if you have the RGBW version.

```
//#define NEO_JEWEL_RGBW
```

10. LED Brightness

If you need to adjust the brightness of your LEDs, simply change the value of the following variable. It accepts a range from 0-255. The higher the brightness, the more power the system requires.

```
#define BRIGHT 100
```

11. Enable/Disable LED & Servo Auto Twitch

Each HP can be set independently from each other to twitch its servos and LEDs automatically at various intervals to give your droid a sense of autonomy. Furthermore, the servos and LEDs of a single HP can be set to twitch independently from each other as well. This auto twitch behavior can be enabled/disabled using the following variables. Enabled = true, Disabled = false

```
boolean enableTwitchLED[HPCOUNT]  = {true,true,true};  // Leds:   Front, Rear, Top
boolean enableTwitchHP[HPCOUNT] = {true,true,true};  // Servos: Front, Rear, Top
```

12. Default LED Auto Twitch Command String

This setting allows you to assign the default twitch sequence behavior for each HP utilizing command stricture strings.

```
String defaultLEDTwitchCommand[3] = { "F001",
                                      "R001",
                                      "T001"  };
```

13. LED & Servo Auto Twitch Interval Ranges

The length between twitches can be adjusted by using these values to select the range (min, max) in seconds to randomly select the next auto twitch interval. This helps give the illusion these displays & movements are totally autonomous as the interval between the same HP twitching will randomly change each time. First value is the min, and the second value is the max time in seconds.

```
const unsigned int LEDTwitchInterval[HPCOUNT][2] = {{240,360},      //Enter min & max seconds, Front LED
                                                    {300,420},     //Enter min & max seconds, Rear LED
                                                    {300,420}};   //Enter min & max seconds, Top LED

const unsigned int HPTwitchInterval[HPCOUNT][2] = {{45,120},     //Enter min & max seconds, Front Servos
                                                   {60,180},     //Enter min & max seconds, Rear Servos
                                                   {60,180}};   //Enter min & max seconds, Top Servos
```

Since R2-D2 is seen in the films to move his HPs often yet rarely projects from them, the default settings have the servo twitch occurring more often than the LED twitch.

14. LED & Servo Auto Twitch Runtime Ranges

In addition to the interval between LED twitches, you can also set the duration the LEDs remain illuminated when executed. This too helps give the illusion these displays are totally autonomous as they operate for a different length of time each time a twitch is executed. First value is the min, and the second value is the max time in seconds.

```
const unsigned int LEDTwitchRunInterval[HPCOUNT][2] = {{5,25},   // (5-25s) Front LED Runtime
                                                       {5,25},   // (5-25s) Rear LED Runtime
                                                       {5,25}};  // (5-25s) Top LED Runtime
```

15. Servo Speed Range

The servo speed range can be set setting the min and max speed settings via the variable below. Values are in ms with lower values resulting in faster servo movement. By using a range, the servos will not always move at the same speed giving the subtle illusion that each movement is purposeful and unique.

```
const int SERVO_SPEED[2] = {150, 400};
```

16. Preset HP Servo Position Coordinates

The following settings can be confusing so I made every effort here to explain them at length.

Since each Builder's droid is assembled differently, we found using position coordinates generated randomly on the fly was problematic due to the occasional servo movement attempting to travel beyond the outer cowl path. Therefore we use a set of 9 position pairs for each HP, set to ensure they lie with in a freely accessible position. These coordinates consist of the servo position of each servo using its pulse width value in microseconds. I suggest using a servo tester with a digital display to find these position pairs while the HPs are mounted in the dome.

The first coordinate pair in each row references the DOWN position of the HP, the second pair is the CENTER position, and the 3rd pair the UP position. LEFT, UPPER LEFT & LOWER LEFT are plotted using the fourth, fifth and sixth pairs while RIGHT, UPPER RIGHT & LOWER RIGHT are plotted using the seventh, eight and ninth pairs

Random twitch movement is then chosen to randomly travel between these 9 safe points minimizing the chance of the movement being impeded causing excessive and unpleasant servo noise.

Important: *The following values are unique to my HP Control setup and are only used as place holders. They will not work as intended in your set up without first finding your own position values and entering them below. Likewise, your values will not work in another builders set up.* There is simply too many variables involved in how HPs are mounted to create a global set of positions that work between droids.

```
const int HPpos[HPCOUNT][HPPOSITIONS][2] = {{{1377,1706},   // Front HP Down
                                             {1477,1426},   // Front HP Center
                                             {1422,1161},   // Front HP Up
                                             {1611,1425},   // Front HP Left
                                             {1611,1155},   // Front HP Upper Left
                                             {1611,1739},   // Front HP Lower Left
                                             {1255,1434},   // Front HP Right
                                             {1255,1115},   // Front HP Upper Right
                                             {1255,1729}},  // Front HP Lower Right
                                            {{1582,1330},   // Rear HP Down
                                             {1500,1512},   // Rear HP Center
                                             {1466,1773},   // Rear HP Up
                                             {1705,1598},   // Rear HP Left
                                             {1705,1797},   // Rear HP Upper Left
                                             {1705,1389},   // Rear HP Lower Left
                                             {1286,1460},   // Rear HP Right
                                             {1286,1602},   // Rear HP Upper Right
                                             {1285,1283}},  // Rear HP Lower Right
                                            {{1507,1208},   // Top HP Down
                                             {1501,1467},   // Top HP Center
                                             {1443,1680},   // Top HP Up
                                             {1624,1468},   // Top HP Left
                                             {1624,1667},   // Top HP Upper Left
                                             {1653,1272},   // Top HP Lower Left
                                             {1267,1468},   // Top HP Right
                                             {1258,1756},   // Top HP Upper Right
                                             {1395,1176}}}; // Top HP Lower right
```

Basic Positioning*:

If you would rather do things the simple way, you can set the Basic Position mode below. This will use the Center, Min and Max pulse width values set in the HPposBasic array. These are based on the assumed default center value of a normal servo so it is important that your servo linkage is installed in a manner where the center servo position corresponds with the orientation you want your HP to be at when centered.

```
#define ENABLEBASICHP true      // false - Disabled, true- Enabled, Basic Servo Positioning
const int HPposBasic[2][3] = {{1500,1300,1700},       // HP Servo 1 Values for Basic Mode (Center, Min, Max)
                              {1500,1300,1700}};      // HP Servo 2 Values for Basic Mode (Center, Min, Max)
```

** enabled by default on new kits in order to increase their plug and playability.*

Important: *If you disassemble your HPs and servo linkage after setting custom Pre-Set values or custom Basic Position values, you will most likely need to recheck and reset them as the chances of reinstalling them exactly the same way is approximately 3,720 to 1.*
I also suggest once you measure your position values, you maintain a backup list of those values separate from your saved sketch. There is a couple of reasons for this, the most important being in the event you need to reinstall the default sketch, or I release an updated version you wish to use, this backup list will keep you from needing to measure your position values again. No one likes to have to mess with dome wiring after it has been neatly ran.

RC mode requires preset position coordinates to work well, so you must have enableBasicHP set to 0 and the HPpos array set with your specific values.

17. Default Color Settings

These settings allow you to set the default colors for the Auto Twitch/Leia and Short Circuit Sequences respectively. Currently, there is a choice of 8 basic colors you can use when setting these defaults.

Basic Color Integer Values
0 = Off (Black}
1 = Red
2 = Yellow
3 = Green
4 = Cyan (Aqua)
5 = Blue
6 = Magenta
7 = Orange
8 = Purple
9= White

Simply use the integer value from the color list above in each of the default color variables below for each HP.

```
const uint8_t defaultColor[HPCOUNT] = {5,5,5};     // {Front, Rear , Top} Color value for default sequence.
const uint8_t shortColor[HPCOUNT] = {7,7,7};       // {Front, Rear , Top} Color value for Short Circuit Sequence
```

The few times we see R2-D2 use his projector in the films, the Holograms projected had a blue tint to them, we selected blue as the default color to closely resemble this behavior. We also chose it because MKelly and Ripcord were whiney little girls about it. :-)

18. Dim Pulse Speed Range and Default Speed

Use this variable to set the default speed of the Dim Pulse function.

```
#define DIMPULSESPEED 5                    // Default speed if no value is given
const int dimPulseSpeedRange[2] = {5, 75};  // Range used to map to value options 0-9, Lower is faster.
```

19. HP Servo Wag Count

Use this variable to set the number of times to wag up/down or left right during Wag function.

```
#define WAGCOUNT 5
```

I2C/Serial Command Structure

How to Make Your HPs Do Some Unique Things

Command Structure

Most of the commands available to to you over I2C or Serial communication come in the form of a 4-5 character alphanumeric string . This string uses a well defined composition utilizing the following structure:

DT##C *or* ***DT##CS*** *or* ***DT##P***

D *- the HP designator; used to select the specific HP you wish to send the command to.*

Value Options: *F - Front HP*
R - Rear HP
T - Top HP
A - All 3 HPs

T *- the Sequence Type;*

Value Options: *0 - LED Function*
1- Servo Function

*- the Sequence Value (including leading zero);*

LED Sequences

Value Options: *01 - Leia Sequence, Random shades of blue to mimic Leia Hologram**
02 - Color Projector Sequence, Like Leia above using color command value
03 - Dim Pulse Sequence, Color slowly pulses on and off (color & speed options)
04 - Cycle Sequence, using color command value
05 - Short Circuit, Led flashes on and off with interval slowing over time
06 - Toggles Color, Simply sets LEDs to solid color value.
*07 - Rainbow Sequence**
*98 - Clears LED, Disables Auto LED Sequence**
*99 - Clears LED, Enables Auto LED Sequence**
** Color Value Below Unneeded, Therefore Ignored, Can Be Omitted From Command String*

Servo Sequences

Value Options: *01 - Sends HP to a Preset Position (requires position value)**
*02 - Enables RC Control on HP (Left/Right)**
*03 - Enables RC Control on HP (Up/Down)**
04 - Sends HP to a Random Position
*05 - Wags HP Left/Right**
*06 - Wags HP Up/Down**
98 - Disables Auto HP Twitch
99 - Enables Auto HP Twitch
**Function disabled or severely limited when Basic HP Positioning in enabled.*

C *- Color Value from the list below:*

Value Options: 0 - Off (Black}
1 - Red
2 - Yellow
3 - Green
4 - Cyan (Aqua)
5 - Blue
6 - Magenta
7 - Orange
8 - Purple
9 - White

S *- (optional), speed setting integer for the Dim Pulse LED function below (0-9)*

P *- (optional), the Position integer value from list below:*

Preset Position Integer Values
0 = Down
1 = Center
2 = Up
3 = Left
4 = Upper Left
5 = Lower Left
6 = Right
7 = Upper Right
8 = Lower Right

Some Simple Examples:

"R0063" - This toggles the Rear HP LEDs to Green (R/0/06/3)
"F0036" - This pulses the front HP LEDs in Magenta (F/0/03/6)
"T007" - This starts the Rainbow sequence in the Top HP LEDs. (T/0/07)
"A098" - This clears the LEDs in all 3 HPs and disables the LED auto twitch mode. (A/0/98)
"F103" - This sends the Front HP to the UP position. (F/1/03)
"A101" - This sends all three HPs to the center position. (A/1/01)
"R104" - This enables RC mode on the rear HP. (R/1/04)
"T199" - This enables the Servo auto twitch mode. (T/1/99)

Important: *Command strings sent via Serial communication require the addition of a carriage return character (\r) to the end of the command string. So the command string "R0063" from the examples above, would become "R0063\r" when using serial communication. This carriage return character lets the system know it has reached the end of a command.*

I2C communication does not require this extra character.

Special Sequence Commands

The commands sent using the structure outlined in the last section only allow you to control either the LEDs or the Servos using any given command. There are times when you would like to use a single command to trigger several different behaviors of both types at once without needing to stack commands. Therefore we use special or "S"-series commands that combine a number of individual type functions and behaviors into a single complex sequence.

Currently we only have 3 of these special commands:

"S1" - Leia Mode: Front HP moves to down position, Leia LED Sequence begins, and the other two HPs are disabled.

*"S8" - Clear and Disable: Clear all LEDs, Disable Auto Servo and LED Twitch. Used to quickly **stop** all HP functions.*

*"S9" - Clear and Enable: Clear all LEDs, Enable Auto Servo and LED Twitch. Used to quickly **reset** all HP functions.*

Timed Commands

New to the v1.3 Sketch is the added ability to add a run time value to any command. This allows you to dictate how long you wish the sequence to run before it halts, returning the system to its last know auto state. This keeps you from needing to manually stop sequences with follow up commands.

To add a run time value, simply add a pipe character , | , followed by the number of seconds you want the sequence to run. So the following command….

A007|45

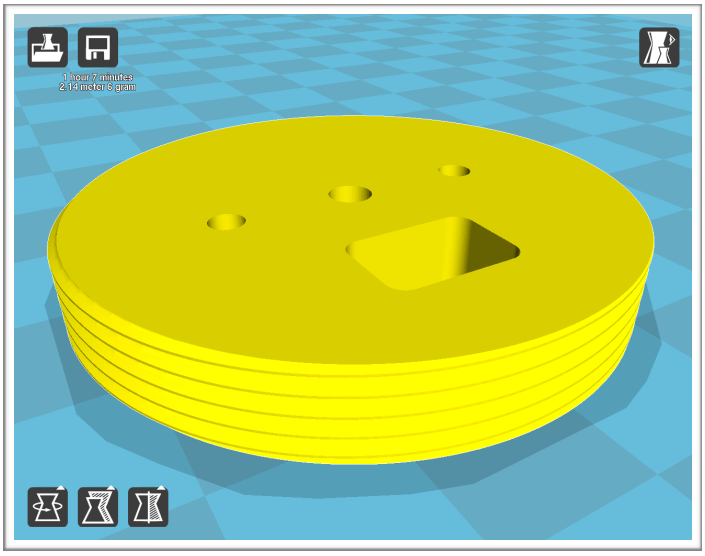
…would run the Rainbow Sequence on all 3 HPs for 45 seconds, at which point they will clear and the system will return to the Auto state it was in proceeding the command.

***Note:** This feature has little effect on servo commands as they are not usually a continuous function by default.*

Replacement Back Plug

In the event you need to print a replacement rear plug due to damage or loss, the model file can be found at: http://2geekswebdesign.com/FlthyHPs/Rear_Plug.

***Important:** I have some tips for getting the best results when printing this plug.*



First, if the model renders incredibly small in your 3d printing software, the units used are incorrect. Just scale the size up by a factor of 25.4.

Second, before printing, orient the plug so that the holes with the bevelled and recessed edges face down towards the build plate. This will insure the edge that must thread into the HP is printed cleanly.

Thirdly, when using Hatchbox brand 1.75mm PLA filament on my Printrbot Metal Plus, I have achieved good results using the following print setting. Maybe you will too.

Layer Height: 0.1mm
Shell Thickness: 1.2mm
Bottom Thickness: 1.2mm
Fill Density: 30%
Print Speed: 40mm/s
Print Temp: 210C
Bed Temp: 50C
Flow: 100%

Nozzle Size: 0.4mm
Retraction Speed: 45mm/s
Retraction Distance 4.5mm
Initial Layer Thickness: 0.3mm
Initial line width: 100%
Bottom Layer Speed: 35mm/s
Cooling Fan: Enabled

Once printed, you will need to manually tap the center hole for M4 pattern bolt. The two mounting holes for the Adafruit Jewel LED board will most likely need slightly enlarged with a drill accommodate M2 sized screws.

To assemble, I use 2-M2 Nylon Screws (cut to length), 2-M2 20mm male/female Nylon standoffs. and 2-M2 Nylon nuts to mount the NeoPixel Jewel to the Back plug.

Finding Your Preset Servo Position Coordinates

As mentioned previously in the explanation of the sketch's HP position variable array, this system works best when we provide a complete set of 6 predetermined coordinate pairs per HP corresponding to specific position, namely the Down, Center, and Up positions along with 3 additional auxiliary positions.

I have found the easiest way to do the is to use a set of servo testers with digital displays. These servo testers will allow us to manually move each HP into the various positions and then quickly gather the two pulse width values in microseconds that make up each coordinate pair.

I used a pair of G.T. Power Digital Servo Performance testers to gather my coordinates pretty quickly once my HPs were mounted in my dome and the control linkage installed. A pair of these can be purchased for less than $50 from Banggood here:

http://www.banggood.com/G_T_Power-Digital-Servo-Performance-Tester-p-999251.html

Before you begin, I must make it clear your HPs need to be mounted with servo plates/linkage in place. These values are dependent on a number of variables and removing your HPs or their linkage you will more than likely require you to perform this process again.

First, take a piece of masking tape or a stick on label and number your two servo testers "1" and "2". This ensures the values gathered aren't mixed up. Tester "1" will always provide the first value in the coordinate pair, and tester "2" will provide the second value.

Next, locate the two servos for your first HP. (It is probably most efficient to start with the Front HP.) Plug one of the servos into Tester "1" and the other servo into Tester "2".

Remember the orientation of the servos in relation to how they are mounted in the dome along with which tester you connected to them. You want to maintain the spacial relationship when moving on to the other two HPs. This will help keep things consistent.

Using the up/down and menu/enter keys on each servo tester, you will want to select "Analog"->"LN Mode"->"Extend". This will take you to a screen that reads "PULSE" with a 4 digit number followed by "US" (which stands for microseconds). You may have also noticed that once you reached this screen,

the HP servos moved a bit, that is because you can now control the PWM values being sent to each servo.

Use the “Adjust” knob on each servo tester to move its respective servo. Using both in unison will allow you to move the HP to specific positions. Feel free to mess around and get a feel for it.

Once you are ready to collect your position coordinates, move the HP to the location you would like to assign as the Down position. Once you are happy with the HP’s orientation, record the value shown on the screen of tester “1” followed by the value seen on tester “2”. You now have collected your first position coordinate pair.

Repeat this process to determine the coordinates for the HP’s Center and Up positions. Once these first 3 coordinates are completed, you will then need to collect coordinates for the 3 auxiliary positions. These positions can be anywhere you wish as long as they lie safely in the range of motion of your HP. These are used to provide random movements different than the standard Down/Center/Up positions, but with the added benefit of being safe from impeding movement which could cause excessive noise or binding.

After collecting these last three auxiliary position coordinates, you have a complete set of positions for this particular HP and you can move on to the next, repeating the process.

Once you have completed collecting your position data for all three HPs, you will need to add them to the HPpos variable array in your custom copy of the FlthyHP Arduino sketch.

As suggested before, I strongly suggest you also save these values separately from your sketch. That way if you ever need to reinstall the sketch or a new version of the sketch is released, you don't need to go through these steps again.

It is probably a good idea to occasionally recollect this data as things get pulled on and tweaked in a manner that can gradually throw your alignment off in one direction or the other. Think of it as a sort of regular maintenance need of Artoo.

Build it Yourself

Open Source/Hardware Because Sharing is Caring

I believe that by freely sharing my hardware design files, Arduino sketch, and 3d models in the spirit of open hardware and software, we can continue to build better droids.

I will continue to host all current digital files for this system online at the following location: http://www.2geekswebdesign.com/FlthyHPs/

**The only file I currently don't have is the EagleCad BRD file for the Arduino Pro Mini Breakout Board because I accidentally deleted it after submitting it to Oshpark for initial fabrication . However, it is shared at Oshpark so you can order it from them at: https://oshpark.com/shared_projects/aIzj6gSL*

The Rules

These files are shared for personal use only. You may not offer/sell a system using my design in part or whole without my express permission.

I offer no warranty or liability for those choosing to built their own from the files provided. Doing so requires a basic familiarity with soldering small SMD components, the knowledge to install an Arduino sketch, and use/access to a 3D printer. If you don't have these skills or know someone who is willing to help you learn, I advise not attempting this build. I simply don't have the time each day to answer questions regarding the prerequisite skill sets needed to accomplish this build.

Thus…I offer complete assembled and tested systems for those not wishing to construct their own!

How to Update & Flash Sketch on Arduino Pro Mini

1. Download and install the current version of the Arduino IDE software from https://www.arduino.cc/en/Main/Software

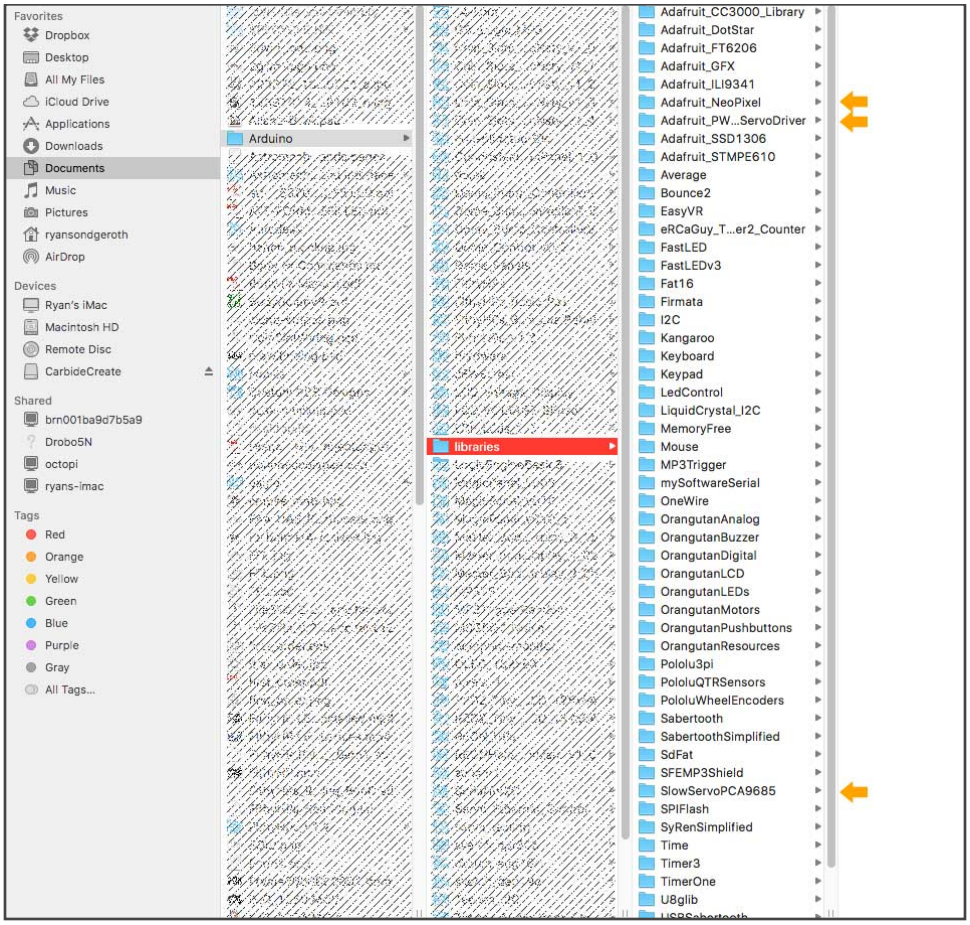
2. Download current sketch from http://2geekswebdesign.com/FlthyHPs/Sketches/

3. Download the current version of the required Adafruit Neopixel library from https://github.com/adafruit/Adafruit_NeoPixel.

4. Download the current version of the required Adafruit PWM Servo Driver library from https://github.com/adafruit/Adafruit-PWM-Servo-Driver-Library.

5. Download the current version of the required BHD Slow Servo library from https://drive.google.com/file/d/0B5B8A65frsBgZ3VpeGxpM1lzaFE/edit.

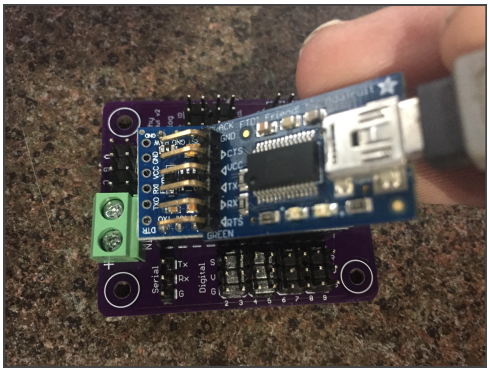
6. Extract each of the three downloaded libraries into the "libraries" directory of your Arduino IDE installation. They should stay in their individual folders inside the "libraries" directory. See Image Below



7. Open the downloaded FlthyHP sketch using the the Arduino IDE software. You may be prompted to create a new directory for the sketch if it is the first time opening it, select ok. After this initial time you can access the saved sketch through the file drop down menu within the Arduino IDE software.

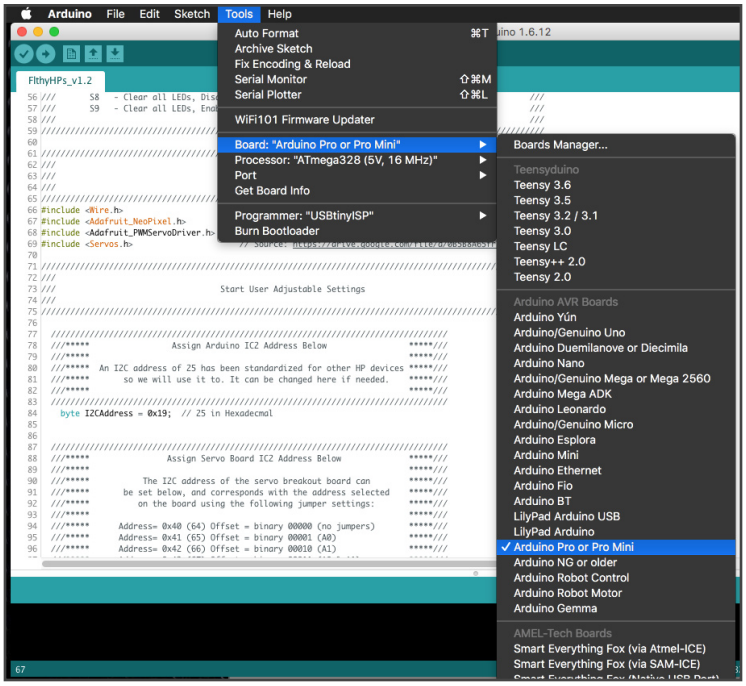
8. Make any desired changes to the sketch settings as explained in this manual.

9. Once changes are made, plug your FTDI programmer into your computers USB port. Your programmer may be different as there is a multitude of options available but I use a FTDI Friend from Adafruit. Some selections in the steps below may be different based on the programmer you are using.

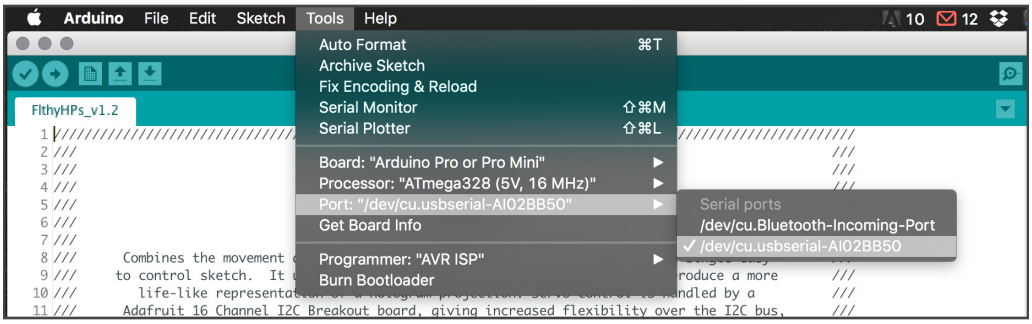


10. Next plug your FTDI programmer into the 6 pin holes on the end of the Arduino Pro Mini. Make sure you align the holes correctly as different models have the pin holes ordered differently. Make sure the Tx pin on the FTDI programmer is plugged in to the Rx pin of the Arduino Pro Mini. Likewise the Rx pin of the FTDI programmer is plugged into the Tx pin of the Arduino Pro Mini. The Arduino's power LED should light up if power is correctly applied over USB through the programmer. See Image ->

11. In the Arduino IDE menu, select Tools->Board and ensure "Arduino Pro or Pro Mini" is selected from the board list. See Image Below.



12. Likewise, in the Arduino IDE menu, select Tools->Port and select the port of the programmer you just connected. It should have a name like “COM#” on a Windows machine or something like “cu-usbserial-XXXXXXX” on a Macintosh/Linux machine. If no port shows up when connecting your programmer, you may need to install the correct drivers for your programmer. This guide does not tackle that process as it is dependent on OS and programmer. Refer to the programmer’s documentation for instructions. See Image Below



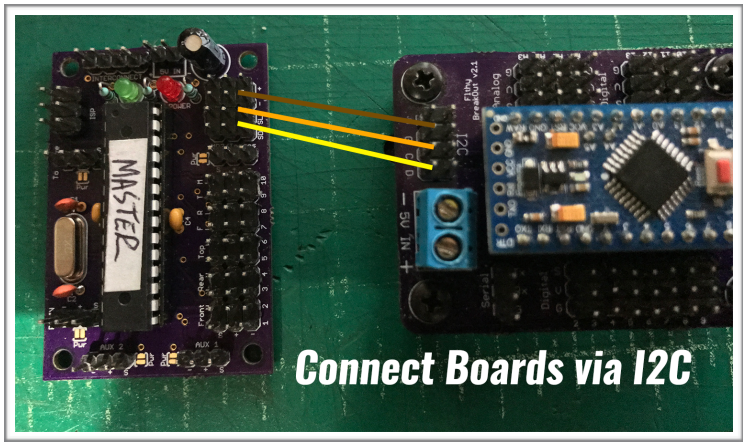
13. At this point you are ready to upload the new sketch. Select the upload button in the upper left corner of the Arduino IDE software window. There are two steps to this process. First, the software will verify the sketch ensuring there are no issues and errors. If no errors are found the sketch is then compiled into machine code and uploaded/flashed to the Arduino Pro Mini. This should only take a few moments, maybe 10 to 30 seconds max. Watch the progress bar at the bottom of the software’s window, you will receive confirmation when the process is complete. If there are any errors they will appear in the lower black window of the software’s interface. Most errors have to do with incorrect syntax or improperly installed libraries.

14. At this point you are done and the sketch has been updated.

Using the FlthyHPs with your Marcduino v1.5/v2

For those wishing to utilize their MarcDuino v1.5 or v2 boards and the R2-Touch app to control their FlthyHPs, Marc added I2C communication support to his system and a pretty slick command structure to facilitate it.

To get started, you will first need to ensure you have both the MarcDuino Master board and the FlthyHPs board connected via the I2C pins. This is accomplished by simply connecting….



the MarcDuino ***SDA*** pin to the the FlthyHP **D** pin,
the MarcDuino ***SCL*** pin the the FlthyHP **C** pin,
and the MarcDuino **-** (ground) pin to the FlthyHP **G** pin.

Note: *Connecting the voltage pins between the two isn't necessary but you can do so if you wish to power one boards via the other. I however recommend you power each board separately from a quality 5v power supply capable of providing 5amps+. Optimal performance can only be achieved if both systems are provided ample, stable power.*

Also, I don't recommended using the I2C pins on the Slave board to connect to the FlthyHPs because there is default behavior in the MarcDuino source code for communicating with the REON HP system. Both the front REON HP and the FlthyHP system use an I2C address of 0x19 (25) and this causes issues. If you must use the I2C of the MarcDuino slave or you have both systems and wish to run them in unison, you can change the I2C address in the FlthyHPs sketch rather easily to remove this conflict. Simply assign it a value not currently in use on your I2C bus, making sure to reference this new address in the command structure explained below.

MarcDuino I2C Commands

Marc's I2C command structure allows for a user to send many different data types, but the FlthyHP command structure utilizes string commands only, so I will only cover this command format here. If you wish to learn more about Marc's entire command structure, visit the documentation on his web site here.

To send a command via I2C to your FlthyHPs, your MarcDuino utilizes the following command structure:

```
&25,"A007|10\r
```

Ok…It may seem confusing at first glance but I assure you it isn't. Lets break it down for you.

`&` is the character specifying the command is an I2C command.

`25` is the I2C address of the target device in decimal format.

`,` (The comma) is a delimiter between the address information and the command information.

`"` (Double quote) tells the MarcDuino you want to send a String command to the FlthyHPs.

A007|10 is the FlthyHP command string you want to send but you already knew that because you read the section covering it earlier in this very manual. In this case it is the command to set all 3 HPs to the Rainbow LED sequence and run it for 10 seconds.

\r is the equivalent of the carriage return character and is utilized by the MarcDuino to signify the end of a command. All commands for your MarcDuino must end with this character.

Please Note: *There is no closing or trailing double quote in this sequence as it is not required.*

Stringing or Combining Multiple Commands

Often you may want to assign multiple commands to a single button in the R2 Touch app as a way off constructing more elaborate droid sequences. Marc had the forethought to add this functionality to his command structure. You can simply add additional commands to your string, keeping each separated by the trailing carriage return character (\r).

So taking the command example above, **&25,"A007|10\r**, we could add the command to open all of Artoo's dome panels, **:OP00\r**. So your new combined command would become **&25,"A007|10\r:OP00\r**. Simple enough.

However, in some cases depending on the amount of processing time required for each command, you may need to introduce a brief pause between the execution of each command. You can do so easily by adding one or more **\p** between your commands in the string.

I have found in my testing this is often the case when sending command to the FlthyHPs, so our command grows just a touch longer to become **&25,"A007|10\r\p\p\p:OP00\r**.

You will notice I added three **\p** to the command string following the first command's carriage return to pause things briefly, giving the system time to process everything correctly.

You will have to do some testing yourself, as it isn't possible to know every possible combination you guys come up with. But a little trial and error will help you get an idea of how best to handle your specific set up.

R2 Touch App

I am not going to go in to much depth regarding the process for adding and assigning custom commands to the R2 Touch mobile app. Marc has provided wonderful documentation regarding the process on his web site here:

http://www.curiousmarc.com/r2-touch-r2-d2-remote-control-iphone-app/customizing-r2-touch

That's all there is to getting these two systems working together. I hope you guys find my explanation clear and helpful.