DotStar Pi Painter
Created by Phillip Burgess

https://learn.adafruit.com/dotstar-pi-painter

Last updated on 2023-09-11 11:59:22 AM EDT
# Table of Contents

## Overview
- Required Parts

## Raspberry Pi Setup
- Raspberry Pi Downloads Page
- Install Packages
- Dry Run
- Success!

## Assembly Part 1
- NeoPixel Painter construction
- Test DotStar Strip
- Troubleshooting

## Assembly Part 2
- Planning Wiring
- Test Again

## Assembly Part 3
- Control Buttons
- Battery Mount
- Diffuser
- First Light
- Troubleshooting
- Set Up Auto-Start

## Photograph!
- Troubleshooting

## Image Prep and Fine-Tuning
- Software Adjustments
- Experimental Stuff
Overview

Digital light painting is a consummate maker project, a marriage of electronics, code, photography...and of course lots and lots of LEDs!

Using a long (several seconds) camera exposure, a single row of LEDs under computer control displays an image one line at a time while it's carried or rolled across the frame. The combined result is a luminous picture floating in air.
We’ve done light painting guides before — like our original Light Painting with Raspberry Pi () and the Arduino-based NeoPixel Painter () — it’s a topic we like to revisit whenever there’s a substantial evolution in the available technology. Each used a new generation of addressable LED strips.

This latest blend of Raspberry Pi computer and DotStar LEDs may be the ultimate, with no size constraints, easy image loading (just plug in a USB flash drive full of pictures) and super buttery smooth interpolation and dithering.
Required Parts

- Raspberry Pi computer: the code runs fine on any Raspberry Pi, but we really recommend a recent model with the 40-pin GPIO header (not an original 26-pin Model A or B), because...
- **Perma-Proto Pi HAT** ([http://adafru.it/2310](http://adafru.it/2310)): installs neatly atop the Pi’s 40-pin GPIO header, providing a sturdy connection point for various other parts.
- 2GB or larger microSD card.
• USB flash drive for loading images.
• [DotStar LED strip](http://adafruit.it/2242): any length and density will work, but 144 LED/m is ideal. 1 meter is good for a painter that’s still easy to store and transport. I built a 2 meter painter, but that was mostly to showcase the added horsepower of the Raspberry Pi…it’s harder to build and unwieldy to move around! NeoPixel strip will not work for this project; must be DotStars for speed.
• [74AHCT125 level shifter IC](http://adafruit.it/1787).
• Six (6) [momentary pushbuttons](http://adafruit.it/1010).
• Large [USB battery bank](http://adafruit.it/1566) or other 5V portable power source. The more current it can provide, the brighter the images can be. Or use two.
• Two (2) [USB Type A male](http://adafruit.it/1387) DIY connectors if using a USB battery bank.
• Soldering paraphernalia, wire, etc. A solderless breadboard and jumper wires may also be useful during initial testing.
• An opaque enclosure for the Raspberry Pi (or get creative with tape, paint, etc.)
• Hardware store bits & bobs.
• A 3D printer is optional but may be handy for improvised brackets, cases and parts.

READ THROUGH THE ENTIRE GUIDE FIRST for more parts and construction ideas. Every painter will be a little different, depending on what materials and tools you have access to.
Raspberry Pi Setup

If using a single-USB-port system (Model A, A+ or Pi Zero), either use a USB hub so you can have keyboard and wireless attached during setup, or borrow a Model B/B+/Pi 2 for setup and then move the card over to the target system once finished.

This project works well with the Raspbian “Lite” operating system — a pared-down system that doesn’t include any GUI apps. Don’t bother with the full-featured Raspbian image for this project, it’s enormous!

Fetch Raspbian Lite here:

Raspberry Pi Downloads Page

If you’re new to Raspberry Pi and Linux, we strongly suggest working through the first few guides in the Learn Raspberry Pi tutorial series...know how to “burn” an SD
image, perform a first-time setup and get the Raspberry Pi connected to a network. Some familiarity with one of the text editors (such as the simple nano or the more daunting vi or emacs) is also recommended.

Install the Raspbian Lite image on a 2GB or larger microSD card. You’ll need to connect a monitor and USB keyboard for basic system configuration, but this is only temporary...we’ll set it up to run “headless” later. For networking, connect either an Ethernet cable or a USB WiFi adapter (or use a Pi with built-in WiFi).

The first-time boot takes longer than usual; this is normal as the system does some self-configuration.

Log in as user “pi”, password is “raspberry”, then run the system configuration tool:

```
sudo raspi-config
```

The following options are required:

- Under “Interfacing Options,” enable SPI

The following are optional but recommended:

- Change User Password
- If your Pi will be on a network (wired or wireless):
  - Under “Interfacing Options,” enable ssh to allow remote login (this makes it easier to copy-and-paste the installation commands later)
  - Under “Network Options” select “Hostname” to distinguish it from other Raspberry Pis on your network; mine is called “lightpaint”
  - If using WiFi, under “Localisation Options” select “Change WiFi Country” to match your location, then under “Network Options” configure the wireless network SSID and password

You can further twiddle system settings to your liking. Tab over to the “Finish” button and reboot the system when prompted.

**Using a USB WiFi adapter? The LED on most will create a problematic light streak when painting; unplug the adapter when photographing cover the LED with tape.**
Install Packages

With ssh enabled, you can use a terminal program and log into the system remotely at lightpaint.local or whatever name you’ve given the system. This is preferable as you can copy-and-paste commands directly. Otherwise, you can log in with screen and keyboard as usual, just be very mindful of spelling.

Let's install a few prerequisite packages and fetch the light-painting software:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install git usbmount python-dev python3-pip libopenjp2-7 libtiff5
sudo pip3 install evdev pillow adafruit-circuitpython-dotstar
```

git clone https://github.com/adafruit/DotStarPiPainter

usbmount is a package for sensing when USB drives are plugged in or removed. We need to install a couple of scripts so that it can inform our light-painter code of these events:

```
cd DotStarPiPainter
sudo cp 99_lightpaint_mount /etc/usbmount/mount.d
sudo cp 99_lightpaint_umount /etc/usbmount/umount.d
```

If running a Raspbian “Stretch” operating system (the current/latest major revision of Raspbian), we need to edit an obscure system file for usbmount to work correctly.

```
sudo nano /lib/systemd/system/systemd-udevd.service
```

Add this line for Pi OS Buster and Bullseye in the [Service] section of the configuration file:

```
PrivateMounts=no
```

Look for this line (older Raspbian "Stretch") - This syntax no longer works with Buster / Bullseye releases.

```
MountFlags=slave
```

And change it to:

```
MountFlags=shared
```

Then...
Dry Run

You don’t need any LEDs connected yet, let’s just make sure all of the software parts are working as expected. Start the light-painting program with:

```
python3 DotStarPiPainter.py
```

If all goes well, nothing should happen…it’s waiting for a USB drive to be inserted. Plug in a USB flash drive with one or more images on it, and after a moment you should see it print a message that it’s loading and processing an image. Fantastic! Press Control+C to break out of the program and we’ll do some final setup.

If the program throws an error, or if it doesn’t detect the USB drive:

- Usually it’s a missing package. Did you do the entire `apt-get` and `pip3` commands above?
- Are the `usbmount` scripts installed in the correct locations?
- If it complains about the `lightpaint.so` file, you may need to recompile that (type “make” in the `DotStarPiPainter` directory)

Some USB flash drives have a power-on LED. Cover the LED with tape, or use a different flash drive with no onboard light.

Success!

Let’s set up the Pi to run fully headless now.

Without a keyboard and monitor attached, shutting down the system would be problematic. Linux systems really expect an orderly shutdown, you can’t just pull the plug or the SD card may get corrupted.

One option is to configure the SD card to boot in read-only mode, as explained in this guide. This is probably best, but wait until you have the light painter completely configured and working first!
Another choice is installing our gpio-halt utility. This tiny program lets you connect a button that performs an orderly system shutdown. (One or the other, you don’t need both.)

```
git clone https://github.com/adafruit/Adafruit-GPIO-Halt
cd Adafruit-GPIO-Halt
make
sudo make install
```

Then edit the rc.local file to start up our code automatically at boot time:

```
sudo nano /etc/rc.local
```

Just before final “exit 0” line, insert this line:

```
/usr/local/bin/gpio-halt 21 &
```

Change the “21” to whatever GPIO pin your shutdown button is connected to (the other leg of the button should connect to a ground pin). On Pi models with the 40-pin header, GPIO 21 is at the very end (nearest the USB ports) with an adjacent ground pin, so it’s very convenient to connect a button across these two pins. On older Pi models with a 28-pin header, GPIO 7 is similarly situated at the end of the header:

This will take effect on the next reboot. Then, to shutdown, tap the button (or jumper across the pins with a screwdriver if no button installed yet) then wait at least 15 seconds before disconnecting power. Test it once with a monitor connected to confirm that it’s working before running the system headless.
This is the “personal experience” part of the project. Every light painter will be different depending on your skillset and available tools and materials. But they’ll have certain parts in common: a Raspberry Pi, some DotStar LEDs, a few buttons and power of some sort. The rest...improvise!

Rather than rehash a lot of construction steps, much can be gleaned by skimming the NeoPixel Painter construction page. We’re using a different processor and different LEDs, but the principles behind the physical thing are unchanged. Seriously, check it out. I’ll wait here for you!

**NeoPixel Painter construction**

Due to Arduino RAM constraints and the SD card block size, the NeoPixel Painter had a hard upper limit of 170 pixels. For the DotStar Pi Painter, in order to showcase some of the Raspberry Pi’s benefits, I built a 2-meter, 288-pixel beast that dominated my garage workbench:
No really. It’s much bigger than it looks there.

While I wouldn’t actively discourage you from building one this big (or even more!), I just need to mention that it’s more complex to make and really unwieldy at this scale, bumping into doorframes and hard to move around for anything but straight-across light painting, not to mention the challenge of getting adequate power to all those LEDs. This was more for “showoff value” than anything practical. A one-meter stick (or even 1.5m if you’re tall) still looks and works great and is easier to build and swing around for dynamic photos!
One more sound reason for making a 1-meter painter: high-density (144 LED/m) DotStar strip comes in 1-meter lengths and is ready to use as-is in its protective sheath.

Creating a longer contiguous strip involves removing the covering, desoldering and cleaning up the strip ends, and re-joining strips end-to-end. This is deceptively challenging, getting all four solder connections working strip-to-strip while avoiding shorts side-to-side, working quickly to not overheat & kill the end pixels. You'll need soldering skill, patience to re-do it a few times if needed, and a multimeter for testing the result each time. All of this is avoided with a 1-meter painter!
Test DotStar Strip

Before embarking on an elaborate construction project, let’s test the DotStar strip first. It’s easier to do troubleshooting up front (and get replacement parts if needed), before it’s built into a thing. Ask me how I know this.

A solderless breadboard and some jumper wires, perhaps alligator clips, are really useful during this test phase. But otherwise, you can solder parts and wires on the Pi HAT board, replacing some wires later for final installation.

The DotStar LED strip needs to be powered from a 5V source (e.g. a “wall wart” or bench supply, or a USB battery bank). The LED supply ground and Pi GND must be connected, but do not join the 5V lines!

144 LED/m DotStar strips don’t have data in/clock in labels! First, examine the strip closely, look for arrows showing the direction of data from “in” to “out”...we’re connecting to the “in” end. + and – usually are labeled, or if your strip has wires attached, these are red (+) and black (–). The data line is adjacent to ground (usu. green wire if attached), clock is the other (usu. yellow wire). Occasionally our supplier changes these up though...we’ll troubleshoot in the next step.

A 74AHCT125 IC is used to interface the 3.3V Raspberry Pi logic to the 5V LED strip:

![Diagram of 74AHCT125 IC]

Connect Vcc (pin 14) to 5V from the DotStar strip (not the Pi), and pin 7 to ground.

The 74AHCT125 has four inputs (1A, 2A, 3A, 4A), four outputs (1Y, 2Y, 3Y, 4Y) and four output-enables (1OE, 2OE, 3OE, 4OE). For this project we’re using only two of each:
1OE (Pin 1) and 2OE (Pin 4) should both be connected to ground to enable the 1Y and 2Y outputs.

Though we're not using channels 3 and 4, it's good practice not to leave these inputs "floating." Connect 3A, 4A, 3OE and 4OE to ground. 3Y and 4Y can be left unconnected.

For reference, here's the Pi header again, highlighting GPIO10 and 11. (These are sometimes labeled "MOSI" and "SCLK" on some reference cards, etc.)

Now we'll use a Python program on the Raspberry Pi to test the LED strip. We need to fine-tune some details first though...

```bash
cd
cd DotStarPiPainter
nano strandtest.py
```

Look for this line early in the code:
numpixels = 30  # Number of LEDs in strip

Change “30” to the number of LEDs in your DotStar strip; 144 or whatever size you’ve built there.

Save the changes and try running the script:

```
python3 strandtest.py
```

If all goes according to plan, you’ll see a few LEDs chase down the length of the strip…first red, then green, then blue…repeating forever (or until you press Control+C).

Let it run, watch closely and make sure every LED along the strip lights all three colors.

Success? Skip ahead to the “Assembly” section below.

**Troubleshooting**

**No lights! Nothing at all!**

- Confirm DotStar strip is connected to 5V power source.
- Confirm data/clock connection to the INPUT end of the strip (look closely for arrows printed on the strip — these show the direction of data from “in” to “out”).
- Confirm ground is connected between the Raspberry Pi and the DotStar power sources.
- Confirm 74AHCT125 is oriented the right way. The “bite” on one end of the chip indicates the Pin 1 end; pins are numbered counterclockwise from there.
- Double-check wiring between the Raspberry Pi, 74AHCT125 and DotStar strip using diagram and explanation above. Are you connected to GPIO 10 and 11?
- Try switching the “data” and “clock” pins; you might have a different generation of DotStar strip with the wires in a different order.

**LEDs are flickering madly!**

- Confirm ground is connected between the Raspberry Pi and the DotStar power sources.
- Try switching the “data” and “clock” pins; you might have a different generation of DotStar strip with the wires in a different order.
LEDs work, but the sequence isn’t red-green-blue.

You might have a different revision of DotStar strip; the native color order was changed between the initial and current product. Change the “order” line in the script to read:

```
order     = dotstar.GRB
```

Other problems?

If the above steps aren’t working or there’s some other issue that’s not addressed here, start a new thread in the Adafruit Forums. It’s extremely helpful if you can provide quality photos showing any wiring and/or soldering between the Raspberry Pi, level-shifter chip, DotStar LED strip and power source.

Do not continue until you have a working DotStar LED strip with the Raspberry Pi “talking” to the correct end.
Assembly Part 2

This is roughly what we’re aiming for:
In the NeoPixel Painter guide (), I recommended 3/4" pine molding for a frame and white elastic as a diffuser. For most people, if you’re building a 1-meter DotStar Pi Painter, this is still the best approach...these materials are the easiest to acquire and work with.

For the giant 2-meter painter, the same pine molding felt just a little too flexy, but thicker 1" stuff was heavy. I opted for 1" aluminum square tubing...extremely light and stiff, and wires could be run down the middle...but in hindsight, simple old wood really is fine. I’m just stubborn and spent way too long finishing the project with aluminum. Use whatever material you’re comfortable working with.

One construction element that has improved since the prior guide is 3D printing, and I used it every opportunity for enclosures and other parts. You really don’t need 3D printing though...all that stuff in the NeoPixel guide about mint tins and improvised parts is still 100% relevant!

Batteries too have changed, with high-current, high-capacity USB battery banks (for charging tablets, etc.) now much more common. We’ll use this instead of the Neopixel Painter’s NiMH + UBEC arrangement (though that’s still a viable option too, if you already have parts around).

Planning Wiring

For a 1-meter DotStar Pi Painter, using a USB power bank with two outputs, plan for wiring something like this:
For illustration purposes only; not to scale. Not shown here, the level-shifter chip should be mounted on a Perma-Proto Pi HAT (or a solderless breadboard is OK while prototyping & testing). Additional connections like the controls and halt button aren’t shown here (we’ll get to that later). Some points of interest here:

- Connect LED strip to the higher-current USB port.
- Use beefy 18 gauge wire for power, to minimize voltage drop. Narrower-gauge wire is fine for everything else.
- +5V and ground are connected at opposite ends of the LED strip. Doesn’t matter which. This is on purpose and ensures more uniform power to all LEDs.
- The 74AHCT125 chip Vcc is connected to the + wire on the LED strip, not 5V from the Raspberry Pi.
- Ground is linked between the Raspberry Pi and the LED parts of the circuit, but keep the 5V supplies separate.
- Try to minimize the distance between the Pi, level shifter and LED strip input; high-speed SPI signals fare best over short distances. To help keep it upright and stable, install the Pi at the bottom of the frame with the data direction arrows pointing upward.
Use **USB DIY connector shells** ([http://adafruit.it/1387](http://adafruit.it/1387)) (Type A male) for power, do not hack up old USB cables for this…most have narrow-gauge power wires and can’t handle a lot of current.

The outer two pins carry power. Get the polarity correct or there will be…trouble.

You may need to Dremel or file the exit channel slightly to accommodate 18 gauge wires.

After soldering and testing polarity with a multimeter, I secured everything with 5-minute epoxy to provide some strain relief, then closed and clamped the shells.

It’s OK to use a regular USB cable for powering the Raspberry Pi…but it sticks out the side and might get accidentally unplugged. The custom DIY cable can go straight to any 5V and ground points on the Perma-Proto Pi HAT and makes for a neater package.

Things are changed slightly for a 2-meter painter:

+5V and ground connect near the middle of the strip rather than the ends. This is possible since the LEDs strips were previously unsheathed.

Again the 74AHCT125 is powered from the LED strip, not the Raspberry Pi. Doing this wrong could possibly damage the first LED on the strip!
See notes on Assembly Part 3 page about power; with an optional second battery bank powering just the Raspberry Pi, you can crank up the brightness of the LEDs!

This photo is from the NeoPixel Painter guide, but the idea is the same here: LED strip is secured to the frame with Permatex 66B adhesive or with carpet tape.

With a wooden frame, wires can be carefully stapled in place.

Since my LED strip was unsheathed, and the support frame was conductive aluminum, I 3D-printed a bunch of these channel pieces. This was time-consuming and ridiculous overkill and is why I'd suggest the wooden frame.
An opaque enclosure for the Raspberry Pi is vital — it prevents LEDs on the board from making light streaks in photographs. This one is 3D-printed from a design on Thingiverse, but you can use a ready-made case or improvise something from cardboard or a mint tin.

The lid pops off to access the halt button... a small tactile switch between GPIO21 and ground. Here you can spot the 74AHCT125 level-shifter chip on the Perma-Proto HAT.

Every DotStar Painter will be unique, built from your personal skill set and materials on hand. On the chance that some of the 3D files are useful to anyone else, they’re available on Thingiverse:

Download 3D Files from Thingiverse

The Pi enclosure is derived from the Raspberry Pi B+ Face Case.

Test Again

There’s still lots more to do, but first...

Double-check your connections, power it up (push the button on the USB power bank to turn it on) and be ready to pull the plugs if you observe a “blue smoke” problem. If everything looks good, log in and try the strandtest script again:

```
cd DotStarPiPainter
python3 strandtest.py
```
You should see the red-green-blue light sequence down the entire strip again. If not, review the troubleshooting steps on the prior page.

Yes, this is actually from the light painter, it’s not just a copied-and-pasted overlay.

Assembly Part 3

Home stretch! Nearly there! At this point, you should have LEDs mounted on a support bar, the Raspberry Pi (with HAT) in an enclosure, and have the strandtest code working.

Next we’ll add painting controls, mount the battery to the frame and optionally add a light diffuser.

Control Buttons

Everyone loves buttons! We’ll add a few to trigger the light painter, adjust speed and select images.
The controls should be near the handle. Since the Pi is kept close to the strip input, you'll need six long wires or some ribbon cable to bridge the distance.

I used five tactile buttons (http://adafruit.it/1010) soldered on a 1/4-size Perma Proto breadboard (http://adafruit.it/1608). Any sort of momentary buttons will do, if you already have some around.

Disregard the “+” label and join both outermost rails to ground.

On the four buttons in contact with the ground rails, it was necessary to clip away one pin using flush cutters. This should be the pin straight across from each button’s corresponding ribbon cable wire.
A little double-stick foam tape is sufficient to hold the controls in place, but I went nuts with 3D printing and made this little press-fit enclosure that holds the Perma Proto and buttons, with an escape slot for a ribbon cable at the top.

The ribbon cable (or separate wires, if that’s what you have) then snake their way back to the Raspberry Pi, where they’re soldered to points on the Pi HAT (or add a socket connector).

A sixth button (not shown here) can go on the Pi HAT for the halt function. I happened to use a smaller tactile button for that (shown on the prior page), but it’s all good.

For posterity, here’s the GPIO pin map again. Any of the green pins is fair game for each of the five buttons...

...but here are the pin assignments normally used in the software:

```python
pin_go    = board.D22
pin_next  = board.D17
pin_prev  = board.D4
pin_faster = board.D23
pin_slower = board.D24
```
If you follow this pinout, then your painter will continue to work after any updates to our code; you won’t have to edit that part each time.

Battery Mount

To hold the USB battery bank, I made this 3D-printed bracket and affixed it to the frame with E6000 glue. Battery then clips into place.

No 3D printer? No problem. Double-stick foam tape will work as well.

Pressing the button on the battery bank switches it on, but also turns on an annoying blue LED that will ruin photographs. Pressing the button a second time turns off the LED, but continues to deliver USB power. To power down the Pi and LED strip, physically unplug the USB cables, which will also shut off the battery bank.

Diffuser

Slightly diffusing the light from the LEDs makes them photograph better. You get a wash of light over a small area rather than a single focused dot.
In the NeoPixel Painter guide we used 3/4" elastic as a diffuser. Still the recommended approach!

In my overly-complex DotStar Pi Painter build, the same channel that holds the LED strip has a second set of grooves to hold elastic, pulled taut and secured at the ends with more 3D-printed bits.

First Light

Now lets give it a test run!

Plug the LED strip and the Raspberry Pi into the USB battery bank and switch it on. While that boots, on your “main” computer, dig up a USB flash drive, format it as a Windows FAT32 filesystem, and drop a few JPEG, PNG or GIF images on there. These should go at the root level of the drive, not inside any folders.

Log in to the Raspberry Pi (either ssh over wireless, or plug in display and keyboard) and run the light painting script manually:

```
cd DotStarPiPainter
python3 DotStarPiPainter.py
```

Nothing will happen at first. Plug in the USB flash drive and you should see a message as it loads and does some processing on the first image. The LED strip will also give a simple indication as its working…red while loading, yellow while processing, green indicating success (ready). The position along the strip shows which image number is being loaded.
If that works, press the “go” button.

You should see the LEDs glimmer for a few seconds, though it won’t make any sense to the naked eye.

Try the faster & slower buttons. A blue light should move up and down the strip indicating the duration of the paint time; about 0.1 second at one end of the strip, 10 seconds at the opposite end. The default time is 2 seconds. Try longer and shorter durations and press the “go” button again.

Try the next & previous image buttons. If there’s multiple images on the USB drive, it’ll show the red-yellow-green sequence as a new image is loaded. Press the “go” button again.

No lights! No go!

First, confirm that the USB drive is detected and contains valid image files; you’ll see the DotStarPiPainter.py script printing messages as it works.

If images are loading but nothing’s displayed, refer to the troubleshooting steps on the Assembly Part 1 page.

Troubleshooting ()

I got some lights for a moment, but now it’s just stuck there!

This can happen if the LEDs draw a lot of current. Voltage from the battery bank dips and the Raspberry Pi locks up. We can fix this!

Reboot the system (unplug, re-plug USB), wait for it to boot, log in and then edit the DotStarPiPainter.py script.

```
cd DotStarPiPainter
nano DotStarPiPainter.py
```

Look for the following line of code (around line 74, just before “INITIALIZATION”):

```
power_settings = (1450, 1550)
```

Those two numbers are the average and peak current (very approximate-ish) to deliver to the LEDs. Although the battery bank can provide more current than this, the
voltage drops when it does so. We need to dial it back a bit. Try changing this to (1000, 1000), save the changes and re-run the script.

If it’s successful now, take note of the value(s) last used, then try something a little higher...like (1200, 1200). If it works, you can continue dialing it up in smaller increments. If it locks up again, reboot and dial down.

Another option, if you’ve got a second battery bank around: dedicate the larger/ higher-current pack 100% to the LEDs and the second one fully to the Pi. The DotStar LEDs work fine with some voltage drop, colors should remain true even at 4V, whereas the Pi will start to have trouble. So if you give the LEDs their own big battery, you can push the current up quite a bit (e.g. (2100, 2600) or so)...the Raspberry Pi, on its own dedicated battery supply, will be totally unaffected by the heavy current draw. The Pi is perfectly happy on a smaller phone-charging battery bank (e.g. 500 mA) if that’s what you have onhand.

**Set Up Auto-Start**

If everything looks good, last step is just to run the DotStarPiPainter.py script automatically when the system boots. If it’s currently running, press Control+C to stop the program. Then enter:

```
sudo nano /etc/rc.local
```

Just before the final “exit 0” line (and after the “gpio-halt” line, if you’ve installed that), insert the following line:

```
python3 /home/pi/DotStarPiPainter/DotStarPiPainter.py
```

Now walk through a full power cycle to confirm everything’s set:

- Press the halt button.
- Wait at least 15 seconds.
- Unplug both USB cables.
System is now fully powered down. The USB battery bank will sleep automatically when there's no load. Now let's boot:

- Plug in both USB cables; LEDs to the high-current port, Pi to the other port.
- Wait 30-60 seconds for system to boot.
- Plug in a USB drive containing images. You should see the loading sequence on the LEDs.
- Press the “Go” button. LEDs should activate.

Does it work? Grand! You can remove the WiFi dongle now; it's not needed during normal use.

If it doesn’t work (but did pass the “first light” test), it’s probably a typo in the rc.local file.

---

**Photograph!**

Now you need a dark place, a tripod and a camera with a long shutter speed (most DSLRs and even point-and-shoot cameras have this). It’s also extremely helpful to have an assistant, either working the camera or the painter...it’s much faster working cooperatively than going back and forth to the camera to see what needs adjustment.

Before taking any photos, establish the center point and boundaries of the frame. Place some tape or other markers on the ground at the left, middle and right limits.

The default painting time is 2 seconds. Try setting the camera’s shutter speed to 4 seconds. As the shutter opens, call it out! The painter then starts walking left-to-right across the frame, pressing the “Go” button when they’re up to speed.
As you can see, photos have a lot more “pizazz” when you have a human subject or something else for scale. Foreground subjects can be illuminated with a camera flash or any modest light source...or even lit using the painter itself. As long as the person holding the rig keeps moving (or is out of the frame when a camera flash is triggered), they won’t show up in the resulting photographs.

You’re also not confined to that one flat plane...things really get interesting when you enter the third dimension! Multiple exposures can also be composited to create complex shots.

It’s helpful sometimes if the person holding the painter wears dark clothes and shoes. But even with lighter attire, as long as they keep moving, it won’t register in the frame.

Troubleshooting
Image is narrow and “squished.”

Use the “slower” button to increase the painting time (and select a longer shutter time on the camera if needed), or walk faster.

Image is wide and “stretched.”

Use the “faster” button to decrease the painting time (can select a shorter shutter time if desired), or walk slower.

Image is off-center.

Try again. Practice! Most shots take several attempts to get a really good one.

Image is too dark, or too bright (“blown out”).

Adjust the camera’s exposure setting independent of shutter time.

Image has some glitches and noise.

The SPI data rate needs to be slowed down. See the next page for code adjustments.
Image Prep and Fine-Tuning

Thanks to the Python Imaging Library, the DotStar Pi Painter supports many common image formats and should read most any JPEG, PNG or GIF image, among others. No special size constraints or pre-processing is needed, just place files in the root directory of a USB flash drive (do not put them inside folders, the software won’t look for them there).

Images will be loaded in alphabetical order.

Each column of the image is “painted” left-to-right. If you need something else (like top to bottom), you’ll need to alter the image file (e.g. rotate 90° counter-clockwise) or tweak the software to perform the desired operations after loading.
Software Adjustments

Depending how your painter is built, you may need to tweak some values in the DotStarPiPainter.py script.

There’s a section near the top of the code that’s commented “CONFIGURABLE STUFF.”

num_leds is the total number of DotStar pixels in the LED strip.

pin_go, pin_next, pin_prev, pin_faster and pin_slower are the GPIO pin numbers to which buttons are connected to begin painting with the current image, load the next (or previous) image, and to set the paint time faster or slower. A momentary pushbutton should be connected to each of these pins, with the opposite leg connected to ground. If you installed the gpio-halt utility, do not use the same pin number for the painting controls.
order is the R/G/B color order used internally by the DotStar strip. Usually this will be 'brg' (with quotes)...but if you have some early DotStars (before 2015), use 'gbr' for proper colors.

vflip indicates whether the input end of the strip is at the bottom ('true' in quotes) or top ('false'). The Pi should be close to the strip input, so I placed this at the bottom to make the center of gravity lower.

spispeed sets the SPI data transfer rate, in bits per second. The default is 12 MHz (12000000). If you find the image “glitching,” you may need to try a smaller value here, like 8 MHz. Or you can try pushing a little higher for better dithering, but no guarantees.

A little further down, gamma and color_balance set the gamma correction curves and maximum brightness for red, green and blue.

power_settings is a list of two values, the average current (in milliamps) that the battery can sustain, and a peak current that it can provide for very short intervals (if in doubt, set peak to the same value as the average).

IMPORTANT: while some battery packs will have a rating like “2.1 Amps,” the output voltage drops slightly (on both USB ports) as current increases. If the voltage drops too low, the Raspberry Pi locks up and the LED strip will be stuck. If this happens, pull the plug and reboot the system, then edit these numbers to smaller values and try again.
Experimental Stuff

The original NeoPixel Painter had an option for adding an optical rotary encoder to provide absolute positioning; with the light-painting rig installed on a bicycle or other wheeled motion base, it then had the spatial sense to provide a consistent aspect ratio for images and even multi-image animation.

The problems with bringing this same feature to the Pi are many: a quality optical rotary encoder is damn expensive, it would require some hairy C code to process the encoder input fast enough, and depending on the encoder it might not have sufficient spatial resolution to work well with the DotStar Pi Painter's X-axis image interpolation.

A different tack was taken. Or, as they say, “It’s a UNIX system, I know this!”

Although we’re running the Raspberry Pi “headless,” it’s still possible for code to read the position of a mouse connected to one of the USB ports. If we can physically install an optical mouse just so (e.g. a millimeter or two from a tire’s sidewall), it might be a good enough position sensor...

A quick kludgey test with tape and a spare-parts-drawer mouse looks promising enough that I've added this feature to the code. If a mouse is plugged in when the DotStarPiPainter.py script starts, it'll use the mouse position for painting speed (rather than a set time interval). Could still use some refinement, but it's a start.
It’s very experimental and there’s some “gotchas” to be aware of:

- Mice can track only so fast. Perhaps a maximum of 1 foot per second, it totally varies from one make and model of mouse to another.
- Some mice are even more sensitive to voltage sags than the Raspberry Pi. If painting a particularly bright image, the voltage may dip low enough that the mouse drops the USB connection (though the Pi is OK). Currently the code doesn’t handle reconnects. Reduce the power_settings values in the code by 100 to 200 milliamps to ensure there’s enough overhead for the mouse to always work.
- The LED on the underside of the mouse might illuminate the wheel and create light streaks in photos (some mice even have corny top-side lighting). You may need to block this somehow.
- Physical installation is quite fussy, needs to hover just above or alongside the tire (most optical mice won’t work against the reflective wheel rim). Later I plan to make a nice 3D-printed bracket to hold it securely in just the right spot.