Build instructions for Closed-loop Spectroscopy Lab: Light-mixing Demo

**Sterling G. Baird1,2,\* and Taylor D. Sparks1,3,\*\***

1Materials Science & Engineering Department, University of Utah, Salt Lake City UT USA, 84108

2Technical contact

3Lead contact

\*Correspondence: sterling.baird@utah.edu

\*\*Correspondence: sparks@eng.utah.edu

# Summary

Closed-loop Spectroscopy Lab: Light-mixing Demo (CLSLab:Light) is a teaching and prototyping platform for autonomous scientific discovery. This platform, which consists of a set of LEDs and a light sensor, encapsulates key principles for “self-driving” (i.e. autonomous) research laboratories including sending commands, receiving sensor data, physics-based simulation, and advanced optimization. It serves as a “Hello, World!” introduction to these topics, accessible by students, educators, hobbyists, and researchers for less than 100 USD, a small footprint, and under an hour of setup time. For a full video build tutorial, please refer to <https://youtu.be/D54yfxRSY6s>.

**For context, please refer to Baird et al. (1).**

# Graphical abstract

*Timeline

Description automatically generated*

# Before you begin

The protocol below describes how to set up a “Hello, World!” for a self-driving laboratory(2) using a Pico W microcontroller, LEDs, a light sensor, and Bayesian optimization.

## Order Required Parts

**Timing: 5 min (not including shipping time)**

1. Order the parts: (<https://www.digikey.com/short/qztj2jt7> AND [Pico W with pre-soldered headers](https://www.pishop.us/product/raspberry-pi-pico-w-with-pre-soldered-headers/)) OR <https://www.digikey.com/short/vtzjbvr2>. For the first option, the total is 68.61 USD (or 73.72 USD including optional parts) + shipping as of 2022-03-06.
   1. The authors plan to periodically check and update the “DigiKey Order" link at <https://hackaday.io/project/186289-autonomous-research-laboratories>.
   2. In case of part shortages, many products may also be found on the Adafruit website.
   3. If you’d like to avoid soldering, you will need to source a Pico W with headers or a Pico WH separately, such as PiShop’s [Pico W’s with pre-soldered headers](https://www.pishop.us/product/raspberry-pi-pico-w-with-pre-soldered-headers/). See also [Raspberry Pi's supported resellers for the Pico W](https://www.raspberrypi.com/products/raspberry-pi-pico/?variant=raspberry-pi-pico-w).
   4. The sculpting wire needs to be 14 gauge (2 mm) or thinner, including the insulation jacket, and rigid enough to support the sensor. The sculpting wire is only used for mounting purposes, not to conduct electricity. Sculpting wire is [also available at Amazon](https://www.amazon.com/dp/B01FG9IRM2?ref_=cm_sw_r_cp_ud_dp_TV8WBR44GZVJ3544KA1X). Approximately 3' is required. See Problem 5:.
   5. The purpose of the wall adapter is so that, after initial setup, the demo can be powered standalone where communication happens purely via Wi-Fi.
   6. The hardware and software was designed to work with the Pico W, though the setup can be adapted for other microcontrollers. See Problem 1:.
   7. The bill of materials, not including the sculpting wire, is also [available at Adafruit](http://www.adafruit.com/wishlists/553992).

Graphical user interface

Description automatically generated with low confidence

Figure 1

## Additional Prerequisites

**Timing: N/A**

1. Ensure access to a 2.4 GHz Wi-Fi network (SSID + password). The purpose of using a wireless connection rather than a hardwired one is to capture the principles behind “cloud experimentation”, where the host and the client may be separated by large geographical distances. Additionally, this allows for a computer to only be required for initial setup such that the device can function standalone, waiting to receive commands and send sensor data. This captures best practices of a scaled-up cloud-accessible lab or network of labs. For more context, see <https://github.com/sparks-baird/self-driving-lab-demo/discussions/91> and <https://github.com/sparks-baird/self-driving-lab-demo/discussions/62>. For links to a simple example using a wired connection, see Problem 2:.
   1. The Pico W only supports 2.4 GHz Wi-Fi networks. See [self-driving-lab-demo #76](https://github.com/sparks-baird/self-driving-lab-demo/issues/76) for additional context and recommendations on setting up a 2.4 GHz Wi-Fi network, if not already available.
      1. WPA enterprise networks such as Eduroam and other networks that use captive portals (most schools, coffee shops, etc.) are not yet supported by MicroPython. It needs to be a network such that on a computer, you can click on the Wi-Fi name (SSID), enter the password, and click connect (no additional steps). Check to see if your institution offers network support for internet of things devices (e.g., [ULink at University of Utah](https://www.housing.utah.edu/living-the-u/getting-connected-cable-internet/)).
      2. Home networks can have both a 5G and a 2.4 GHz network (e.g. “My Network 5G” and “My Network”)
      3. If you use a mobile hotspot, you may need to use your device’s “extended compatibility” feature to drop the mobile hotspot from 5G to 2.4 GHz. See also [prepaid, long-expiry hotspot](https://github.com/sparks-baird/self-driving-lab-demo/discussions/83) and [classroom demos with standalone network access](https://github.com/sparks-baird/self-driving-lab-demo/discussions/88) discussions, which includes a summary of recommendations for prepaid mobile hotspots
2. Ensure access to a computer (for initial setup only)
   1. At a minimum, the computer needs to be able to run the Thonny editor (lightweight) and it must have at least one USB-A port
3. If the headers are not already soldered onto the microcontroller, ensure access to a soldering iron and soldering wire (thinner is better in this case)
4. (Optional) Ensure the Pico W can successfully connect to a computer
   1. You can do this by holding the BOOTSEL button on the Pico W while connecting the Pico W to your computer via the USB cable. If a new drive appears, that indicates that the Pico W is working normally
   2. If soldering, be careful only to heat the gold pads to avoid damaging the circuitry

# Key resources table

|  |  |  |
| --- | --- | --- |
| REAGENT or RESOURCE | SOURCE | IDENTIFIER |
| Deposited Data | | |
| Red, Green, and Blue LED Spectral Data | https://github.com/sparks-baird/self-driving-lab-demo/tree/v0.6.0/src/self\_driving\_lab\_demo/data | v0.6.0 |
| Software and Algorithms | | |
| self-driving-lab-demo v0.6.0 | https://github.com/sparks-baird/self-driving-lab-demo | v0.6.0 |
| Other | | |
| AS7341 Color Sensor | DigiKey (Adafruit Product) | Cat#1528-4698-ND |
| Grove to Stemma-QT adapter | DigiKey (Adafruit Product) | Cat#1528-4528-ND OR Cat#1528-4528-ND |
| Raspberry Pi Pico W with pre-soldered headers OR (Raspberry Pi Pico W AND Header pins with 20 positions and 2.54 mm pitch (x2)) | PiShop OR (DigiKey-Adafruit Product AND DigiKey-Amphenol CS) | Cat#ASM-1918 OR (Cat#2648-SC0918CT-ND AND Cat#10129378-920001BLF-ND) |
| USB-A to USB-B Cable | DigiKey (Adafruit Product) | Cat#380-1431-ND |
| Maker Pi Pico base (without Pico) | DigiKey (Adafruit Product) | Cat#3614-MAKER-PI-PICO-NB-ND |
| AC/DC Wall Mount Adapter 5V 5W | DigiKey (Adafruit Product) | Cat#1470-2768-ND |
| 18 AWG Hook-up solid black wire, 100’ (Outer diameter 14 AWG or higher) | DigiKey (Remington Industries) | Cat#2328-18UL1007SLDBLA-ND |
| Terminal Binding Post M2.5 (Optional) | DigiKey (Keystone Electronics) | Cat#36-8737-ND |
| 128MB Micro SD Memory Card (Optional) | DigiKey (Adafruit Product) | Cat#1528-5250-ND |

# Step-by-step method details

## 

## Hardware Setup

**Timing: 20 min**

Unless pre-soldered, attach the headers onto the Pico W, mount the light sensor so that the pinhole is facing the red green blue (RGB) LED, connect the light sensor to the board, and get the microcontroller ready for firmware installation.

1. Unless pre-soldered, Solder headers onto the Pico W or [use a hammer header pin install rig for Pico W](https://vilros.com/products/copy-of-hammer-header-pin-install-rig-for-raspberry-pi-pico-for-standard-pico-only-not-compatible-with-the-pico-w_) (note, Pico install rigs are not compatible with the Pico W)
   1. If soldering, insert the Pico W headers into the Maker Pi Pico base, place the Pico W on top of the headers, and solder the headers to the Pico W ([MagPi guide](https://magpi.raspberrypi.com/articles/how-to-solder-gpio-pin-headers-to-raspberry-pi-pico), [Tom's hardware guide](https://www.tomshardware.com/how-to/solder-pins-raspberry-pi-pico), or [YouTube video](https://www.youtube.com/watch?v=R11QanPDccs)), and remove the Pico W from the Maker Pi Pico base
2. Prepare 3 feet of sculpting wire (cut with wire cutters or bend until it breaks)
3. Thread the sculpting wire through each mounting hole on the Maker Pi Pico base, then twist the wires together near the RGB LED. This setup will allow the position and orientation of the sensor to be both adjustable and steady. Continue twisting until you have 4 to 6 inches of twisted wire, and ensure that there are at least 3 inches of loose, untwisted wire at each end (the leftover, untwisted wire will be threaded through the mounting holes of the light sensor in the next step). For reference, a diagram is also included below. For a more modular alternative of fixturing the wire ends to the Maker Pi Pico base, see Problem 4:

Diagram

Description automatically generated  
Figure 2

A picture containing text, clock, screenshot

Description automatically generated  
Figure 3

1. Thread the same sculpting wire through the AS7341 light sensor and position the sensor so the pinhole is facing approximately 3 to 4 inches away from the RGB LED.

Website

Description automatically generated with low confidence

Figure 4

1. Connect the Grove/Stemma-QT connector into Grove port 6 (GP26&27) and the AS7341, insert the Pico W, and while holding the BOOTSEL button, connect the Pico W to the computer.

Graphical user interface

Description automatically generated with medium confidence  
Figure 5

## Software Setup

**Timing: 20 min**

Install the MicroPython firmware onto the Pico W microcontroller, enter the Wi-Fi credentials, and upload the source code files.

1. Download and install [Thonny](https://thonny.org/), a Python IDE with native support for microcontrollers, onto your computer. Choose the platform appropriate for you (in my case, this is Windows 64-bit, Python 3.10). When installing, use the default settings: "Standard (default)". Thonny comes with its own version of Python located by default at C:\Users\<username>\AppData\Local\Programs\Thonny\python.exe on Windows computers. It is not anticipated that this will cause conflicts with existing installations of Python; however, for conda users, an isolated installation may be performed via the following commands in a conda shell:  
   conda create -n sdl-demo-thonny python==3.10.\*

conda activate sdl-demo-thonny

pip install thonny  
thonny

1. Click on the lower-right dropdown and click "Install MicroPython", which will install the microcontroller firmware onto the Pico W

Graphical user interface, text, application, email

Description automatically generated  
Figure 6

1. Choose "MicroPython variant: Raspberry Pi - Pico W / Pico WH" and click install

Graphical user interface, text, application

Description automatically generated  
Figure 7

1. Change the interpreter from Local Python 3 to MicroPython (Raspberry Pi Pico), which will open a shell that can be used to enter MicroPython commands that run directly on the Pico W

Graphical user interface, text, application

Description automatically generated  
Figure 8

1. In Thonny's menubar, click "View" then "Files" to open a sidebar which shows both your local computer’s files (top) and the files on the Pico W (bottom)

Graphical user interface, application

Description automatically generated  
Figure 9

1. Download *sdl\_demo.zip* from [the latest release at self-driving-lab-demo](https://github.com/sparks-baird/self-driving-lab-demo/releases/latest) to your computer and unzip it
2. In Thonny, navigate to the unzipped *sdl\_demo* folder, open *secrets.py*, and enter your Wi-Fi network name (SSID) and password as Python strings. Optionally, you can create your own MongoDB Atlas database and enter values for MONGODB\_API\_KEY, MONGODB\_COLLECTION\_NAME, and DEVICE\_NICKNAME (see below). Optionally, you can create your own HiveMQ instance and enter the credentials there (see below). Save *secrets.py.*

Graphical user interface, text, application, chat or text message

Description automatically generated  
Figure 10

Graphical user interface, application

Description automatically generated  
Figure 11

* 1. (Optional) Set up a MongoDB database backend. If ignored, the demo will function, just without logging data to a database (i.e., the user becomes responsible for saving the data on the client side). See Problem 3:.
     1. Create an account at <https://www.mongodb.com/cloud/atlas/register>
     2. Create a free, Shared Cluster (optionally rename Cluster0 to something of your choice, e.g. self-driving-labs. You can leave the default provider as-is)

Graphical user interface, text, application, email

Description automatically generated  
Figure 12

* + 1. Navigate to “Data Services” 🡪 “Deployment” 🡪 “Database” and click “Browse Collections” then “Add My Own Data”. Enter a database name (e.g., clslab-light-mixing) and collection name (e.g., test). Copy the names into MONGODB\_DATABASE\_NAME and MONGODB\_COLLECTION\_NAME in secrets.py.

Graphical user interface, text, application, website

Description automatically generated  
Figure 13

* + 1. Navigate to “Data Services” 🡪 “Services” 🡪 “Data API”, use the dropdown to select your cluster, and click “Enable Data Access from the Data API”

Graphical user interface, text, application

Description automatically generated  
Figure 14

* + 1. Note the app name in the “URL Endpoint” box of the form “https://data.mongodb-api.com/app/<data-abc123> /endpoint/data/v1” where <data-abc123> is the app name. Copy the app name into the MONGODB\_APP\_NAME variable in secrets.py.

Graphical user interface, application

Description automatically generated  
Figure 15

* + 1. Click “Create API Key”, enter a name of your choice (e.g. clslab-light), and click “Generate API key”. Copy the API key and store it somewhere secure. Paste the API key into the MONGODB\_API\_KEY variable in secrets.py.

Graphical user interface, text, application

Description automatically generated  
Figure 16

* 1. (Optional) Create your own HiveMQ instance. If this setup is ignored, the demo will function properly; however, the hardware commands and sensor data will be transmitted via a default HiveMQ instance for which [the credentials are public](https://github.com/sparks-baird/self-driving-lab-demo/blob/main/src/public_mqtt_sdl_demo/sample_secrets.py). Setting up your own HiveMQ instance ensures that the data you transfer remains private and secure. Other MQTT brokers such as [Mosquitto](https://mosquitto.org/) or [Adafruit IO](https://io.adafruit.com/api/docs/mqtt.html" \l "adafruit-io-mqtt-api) are available. At the time of writing, we recommend HiveMQ because it provides free instances with generous limits. Setting up a private MQTT broker is in line with [best practices for internet of things (IoT) security](https://iot.stackexchange.com/questions/554/is-there-any-advantage-in-encrypting-sensor-data-that-is-not-private).
     1. Navigate to <https://www.hivemq.com/mqtt-cloud-broker/>, click “Try out for free”, and create an account
     2. Set up credentials by entering a username and password and press “ADD”

Graphical user interface, text, application

Description automatically generated  
Figure 17

* + 1. Navigate to the “Clusters” tab and copy the URL (e.g., abc123.s2.eu.hivemq.cloud) to HIVEMQ\_HOST in secrets.py. Also update HIVEMQ\_USERNAME and HIVEMQ\_PASSWORD with the username and password from the previous step.

Graphical user interface, application

Description automatically generated  
Figure 18

* + 1. Create a certificate using the Google Colab notebook at <https://github.com/sparks-baird/self-driving-lab-demo/blob/v0.7.3/notebooks/7.2.1-hivemq-openssl-certificate.ipynb>. Enter the server address (same as HIVEMQ\_HOST), run the Google Colab cells, and follow the instructions to download the hivemq-com-chain.der file to the unzipped sdl\_demo folder. This file is used to do secure authentication via HiveMQ.

1. While holding Ctrl (Windows) or Cmd (Mac), select "lib", "main.py", “hivemq-com-chain.der”, and "secrets.py", right click in the gray region, and click "Upload to /". This will upload the files from your computer to the Pico W microcontroller

Graphical user interface, application

Description automatically generated  
Figure 19

1. Double click to open *main.py*, click the green play button (i.e., run the code on the Pico W), and note the PICO ID that prints to the command window ("prefix/picow/<PICO\_ID>/"). This will act as the “password” to control the demo.

Graphical user interface, text, application, email

Description automatically generated  
Figure 20

## Control from the cloud

**Timing: 10 min**

Control the device via internet-of-things style communication (MQTT) and run a basic optimization comparison of grid search vs. random search vs. Bayesian optimization.

1. [Open notebooks/4.2-paho-mqtt-colab-sdl-demo-test.ipynb in Google Colab](https://colab.research.google.com/github/sparks-baird/self-driving-lab-demo/blob/main/notebooks/4.2-paho-mqtt-colab-sdl-demo-test.ipynb)
2. Scroll to the first code cell and click the play button to install the self-driving-lab-demo Python package

Text

Description automatically generated  
Figure 21

1. Copy the PICO ID from the Thonny editor and paste it in place of "test" (without quotes). The following is an example image of the output; the actual output to the command window may vary in future releases.

Graphical user interface, text, application, chat or text message

Description automatically generated  
Figure 22

Graphical user interface, application

Description automatically generated  
Figure 23

1. Run the remaining code cells
   * 1. Instantiate a SelfDrivingLabDemo class
     2. Perform optimizations for grid search, random search, and Bayesian optimization

# Expected outcomes

1. Successfully set up the hardware and software for a closed-loop experiment
2. Run the first “autonomous drive” given in an example interactive notebook
3. Explore [additional example notebooks](https://github.com/sparks-baird/self-driving-lab-demo/blob/main/notebooks/README.md)

Figure 17 shows a comparison of optimization results for grid search vs. random search vs. Bayesian optimization averaged over repeat campaigns with standard deviation error bands, where Bayesian optimization, on average, performs the best. Figure 18 shows one of the outputs from the cloud-based control notebook of best error so far vs. iteration number comparing grid search vs. random search vs. Bayesian optimization. Typically, grid search is the least efficient, Bayesian optimization is the most efficient, and random search is somewhere in-between. Figure 19, Figure 20, and Figure 21 show the points that were searched for a given campaign for grid search, random search, and Bayesian optimization, respectively. Finally, Figure 22 shows the true, underlying target color (defined by red, green, and blue values) and the best parameter set based on minimizing error between the observed spectrum and the target spectrum for each of the optimization methods.

Chart, histogram

Description automatically generated  
Figure 24

Chart, line chart

Description automatically generated  
Figure 25

Chart

Description automatically generated  
Figure 26

Chart

Description automatically generated  
Figure 27

Chart, scatter chart

Description automatically generated  
Figure 28

Chart, scatter chart

Description automatically generated  
Figure 29

# Quantification and statistical analysis

Discrete Fréchet distance, as implemented in <https://github.com/cjekel/similarity_measures>, is used to assess the mismatch between the currently observed spectrum and the target spectrum, where the target spectrum is determined by arbitrarily choosing a random set of RGB values and measuring the sensor data for the fixed, random set of RGB values. Lower Fréchet distances correspond to better matches between the observed and target spectra (i.e. lower error).

An example JSON document logged to a MongoDB database backend containing experimental data for a single run is given as follows:

{

"utc\_timestamp": "2022-11-4 06:51:16",

"ch510": 354,

"ch620": 5671,

"ch410": 188,

"ch440": 3675,

"ch583": 2756,

"\_input\_message": {

"\_session\_id": "542e6e80-9c50-4c41-95a5-832603b96238",

"B": 31,

"atime": 100,

"gain": 128,

"astep": 999,

"\_experiment\_id": "9b50c819-db8f-476f-b601-dbe79e871a46",

"G": 3,

"integration\_time": 280.78,

"R": 41,

},

"onboard\_temperature\_K": 294.1085,

"sd\_card\_ready": True,

"ch470": 2827,

"ch550": 498,

"ch670": 277,

}

The experimental parameters for two JSON documents are given in Table 1.

Table 1. Example of data obtained from two experiments. The LED parameters are red (R), green (G), blue (B). The sensor settings are atime, gain, astep (affects integration time and intensity). The measured output values are of the form “ch###” where the three digit number corresponds to the full-width half-max (FWHM) wavelength being measured.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **utc\_timestamp** | **onboard\_temperature\_K** | **R** | **G** | **B** | **atime** | **gain** | **astep** | **ch410** | **ch440** | **ch470** | **ch510** | **ch550** | **ch583** | **ch620** | **ch670** |
| 11/4/2022 6:40 | 292.7041 | 41 | 3 | 31 | 100 | 128 | 999 | 188 | 3674 | 2828 | 354 | 498 | 2748 | 5661 | 276 |
| 11/4/2022 6:51 | 294.1085 | 41 | 3 | 31 | 100 | 128 | 999 | 188 | 3675 | 2827 | 354 | 498 | 2756 | 5671 | 277 |

The code for grid search, random search, and Bayesian optimization is hosted at <https://github.com/sparks-baird/self-driving-lab-demo/blob/main/src/self_driving_lab_demo/utils/search.py> [[permalink](https://github.com/sparks-baird/self-driving-lab-demo/blob/268c21438ee6e98385b7aaad5e0a73034e095799/src/self_driving_lab_demo/utils/search.py)].

# Limitations

Environmental noise (e.g. light conditions) and hardware variation (LED, sensor, sensor positioning, etc.) may affect the results obtained.

# Troubleshooting

See the [GitHub issue tracker](https://github.com/sparks-baird/self-driving-lab-demo/issues) for existing known issues or to post a new issue. See the [GitHub discussions](https://github.com/sparks-baird/self-driving-lab-demo/discussions) for general questions and discussion.

## Problem 1:

Can I use this with alternate microcontrollers or firmware?

## Potential solution:

The hardware configuration and software were designed based on Raspberry Pi’s Pico Wireless (Pico W) microcontroller. Libraries exist for LED control and the AS7341 light sensor in CircuitPython and Arduino. The hardware and configuration and software can be adapted for other microcontrollers. Contributions at [https://github.com/sparks-baird/self-driving-lab-demo/](https://github.com/sparks-baird/self-driving-lab-demo/issues) are welcome. See page 3.

## Problem 2:

Can I use this without connecting to the internet?

## Potential solution:

A simple example of wired communication between a computer and the microcontroller for the microcontroller host code and a Jupyter notebook tutorial (client) can be found at <https://github.com/sparks-baird/self-driving-lab-demo/tree/main/src/extra/nonwireless> [[permalink](https://github.com/sparks-baird/self-driving-lab-demo/blob/c270484b4d5e1c8d5c75558e939885683d1215be/src/extra/nonwireless/main.py)] and <https://github.com/sparks-baird/self-driving-lab-demo/blob/main/notebooks/5.0-nonwireless-search.ipynb> [[permalink](https://github.com/sparks-baird/self-driving-lab-demo/blob/268c21438ee6e98385b7aaad5e0a73034e095799/notebooks/5.0-nonwireless-search.ipynb)], respectively. While possible with some modification, data communication via a USB cable is not actively supported for [new releases of microcontroller host code](https://github.com/sparks-baird/self-driving-lab-demo/releases) nor [the advanced tutorials](https://github.com/sparks-baird/self-driving-lab-demo/blob/main/notebooks/README.md" \l "6-advanced-optimization). For private, secure, wireless communication between the Pico W microcontroller and the client (e.g., Jupyter notebook running locally), a free, private HiveMQ instance can be set up per the instructions in Software Setup. For recommendations regarding connecting to a 2.4 GHz network (e.g., in university classroom settings) see <https://github.com/sparks-baird/self-driving-lab-demo/discussions/83> and <https://github.com/sparks-baird/self-driving-lab-demo/discussions/88>. See also page 3.

## Problem 3:

Can I use this without logging to a MongoDB backend?

## Potential solution:

If the MongoDB credentials are left to their default dummy values in secrets.py, then logging to the MongoDB backend will fail and the device will simply notify the user rather than exit the program. In other words, the device will function normally without database logging. The same applies for logging to an onboard SD card. If an SD card is detected, the microcontroller will write backup data to it, otherwise this step will be skipped. See page 11.

## Problem 4:

The Stemma-QT to Grove connectors are out-of-stock.

## Potential solution:

First, look at Adafruit and other vendors to see if it is available. Note that Cat#[1528-4424-ND](https://www.digikey.com/en/products/detail/adafruit-industries-llc/4424/11569122) is incompatible with the Maker Pi Pico base due to the adapter housing blocking it from being plugged in fully. If no Stemma-QT to Grove connectors can be located, another alternative is using a Stemma-QT to header pin cable (DigiKey Cat#[1528-4209-ND](https://www.digikey.com/en/products/detail/adafruit-industries-llc/4209/10230003)) and plugging directly into the GPIO pins that correspond to Grove Port #6 of the Maker Pi Pico base. See page 2.

## Problem 5:

The sculpting wire doesn’t fit through the mounting holes.

## Potential solution:

Ensure that the outer diameter of the sculpting wire is 14 AWG or higher (i.e., 1.628 mm or thinner). Enameled wire (often advertised as sculpting wire) has a very thin coating, whereas electrical wiring typically has a non-negligible insulation thickness. Alternatively, for a more modular setup, a single M2.5 binding post ([Digikey Cat#36-8737-ND](https://www.digikey.com/en/products/detail/keystone-electronics/8737/2170367)) can be used to clamp the wire via a single mounting hole instead of looping the wire through each of the mounting holes. See page 3.

Resource availability

***Lead contact***

Further information and requests for resources and reagents should be directed to and will be fulfilled by the lead contact, Taylor D. Sparks [sparks@eng.utah.edu](mailto:sparks@eng.utah.edu).

***Materials availability***

This study did not generate new unique reagents.

***Data and code availability***

The datasets and code generated during this study are available on GitHub: <https://github.com/sparks-baird/self-driving-lab-demo>. The recommended option for ordering parts is <https://www.digikey.com/short/qztj2jt7> AND [Pico W with pre-soldered headers](https://www.pishop.us/product/raspberry-pi-pico-w-with-pre-soldered-headers/) to avoid soldering. Alternatively, <https://www.digikey.com/short/vtzjbvr2> is a standalone DigiKey order, but requires soldering headers onto the Pico W. For a full video build tutorial, please refer to <https://youtu.be/D54yfxRSY6s>. Code for grid search, random search, and Bayesian optimization is hosted at https://github.com/sparks-baird/self-driving-lab-demo/blob/main/src/self\_driving\_lab\_demo/utils/search.py.

# Acknowledgments

This work was supported by the National Science Foundation under Grant No. DMR-1651668.

# Author contributions

Sterling G. Baird: Conceptualization, Methodology, Software, Writing – Original Draft, Writing – Review & Editing, Visualization, Taylor D. Sparks: Supervision, Funding Acquisition

# Declaration of interests

The authors have been exploring selling at-cost kits via a crowdfunding platform called [GroupGets](https://groupgets.com/). As of 2023-03-04, the cost-breakdown is as follows: List price: 75 USD + shipping (depends on location). GroupGets fee: 15.57 USD (10%+2.9%+5 USD). Hardware cost: 56.19 USD. Labor/testing: 20 min (est. 6 USD). Profit: -2.76 USD, offset somewhat by bulk pricing discounts. Note that this assumes purchasing a Pico W with presoldered headers. See [round 1](https://groupgets.com/campaigns/1112) and [round 2](https://groupgets.com/campaigns/1129) and a discussion of [packaging open-source hardware as commercial kits](https://github.com/sparks-baird/self-driving-lab-demo/discussions/124).

# References

* 1. S.G. Baird, T.D. Sparks, “What is a Minimal Working Example for a Self-driving Laboratory?” Matter, Cell Press, 2022. 5 (12), 4170–4178. <https://doi.org/10.1016/j.matt.2022.11.007>.
  2. Seifrid, M.; Hattrick-Simpers, J.; Aspuru-Guzik, A.; Kalil, T.; Cranford, S. Reaching Critical MASS: Crowdsourcing Designs for the next Generation of Materials Acceleration Platforms. Matter, Cell Press, 2022. 5 (7), 1972–1976. <https://doi.org/10.1016/j.matt.2022.05.035>.

# Figure legends

Figure 1: Visual bill of materials

Figure 2: Wire mounting instructions

Figure 3: Wire mounting schematic

Figure 4: Light sensor mounting instructions

Figure 5: Hardware connections

Figure 6: Firmware installation dropdown

Figure 7: MicroPython installation dialogue box

Figure 8: Interpreter dropdown

Figure 9: Opening the files sidebar

Figure 10: Editing secrets.py

Figure 11: Saving secrets.py

Figure 12: Setting up a MongoDB shared cluster

Figure 13: Create a MongoDB database

Figure 14: Enable the Data API

Figure 15: Retrieve MONGO

Figure 16: Create Data API key

Figure 17: Set up HiveMQ credentials

Figure 18: Locate the HiveMQ host URI

Figure 19: Uploading source files to microcontroller

Figure 20: Running main.py

Figure 21: Python package installation

Figure 22: Copying the Pico ID from the Thonny editor

Figure 23: Pasting the Pico ID into the Google Colab form box

Figure 24: Example optimization comparison between grid search, random search, and Bayesian optimization averaged over repeated campaigns. Lower Fréchet distance between observed and target spectra is better.

Figure 25: Example optimization comparison between grid search, random search, and Bayesian optimization. Lower error is better.

Figure 26: Twenty-seven grid search points colored by the Fréchet distance between the target spectrum and the sensor data evaluated at each grid point.

Figure 27: Twenty-seven random search points colored by the Fréchet distance between the target spectrum and the sensor data evaluated at each grid point.

Figure 28: Twenty-seven Bayesian optimization points colored by the Fréchet distance between the target spectrum and the sensor data evaluated at each grid point.

Figure 29: The true, underlying RGB target (purple diamond) and the best observed points for grid search (blue circle), random search (red circle), and Bayesian optimization (green circle). Bayesian optimization gave the closest match to the true target.

Methods Video S1: Thread the mounting wire through the mounting holes of the Maker Pi Pico base, related to step 3

Methods Video S2: Thread the remaining mounting wire through the mounting holes of the AS7341 light sensor and position the sensor above the LEDs, related to step 4

Methods Video S3: Attach the Pico W and the AS7341 light sensor to the Maker Pi Pico base, then connect the USB cable from the Pico W to the computer while holding down the BOOTSEL button, related to step 5

Methods Video S4: Download the Thonny editor and install the MicroPython firmware onto the Pico W, related to steps 6, 7, 8, and 9

Methods Video S5: Download the source code from GitHub, unzip it, and enter Wi-Fi credentials, related to steps 10, 11, 12, and 13

Methods Video S6: Upload the source code to the Pico W and run the main.py script, related to steps 14 and 15

Methods Video S7: Open the cloud-control Jupyter notebook via Google Colab and install the self-driving-lab-demo Python package, related to steps 16 and 17

Methods Video S8: Copy-paste the PICO ID from Thonny to Colab and control the setup remotely through the “evaluate” command, related to steps 18 and 19.

Methods Video S9: Perform the “Hello, World!” of optimization, comparing grid search vs. random search vs. Bayesian optimization, related to step 19

Methods Video S10: Visualize the results of the optimization comparison, related to step 19