IOT Lab 1: IoT Devices (Part 2)

In part one of this lab you constructed a basic self-driving car. Currently your car acts kind of like a Roomba - it goes until it encounters an obstacle, then backs up and tries another path, etc. In this lab, you will give the gift of computer vision to your car. It will be able to understand what obstacles are in front of it, and much more. To do this, we will have to build a more advanced understanding of computer intelligence, which we will do through the following steps.

Step 6: More Advanced Mapping

Overview:
In the real world, intelligent systems will typically construct a formal representation of their physical environment to assist in their navigation capabilities. Self-driving cars for example will construct a "map" of the environment that consists of a scanned 3-dimensional point representation observed from various sensors on the car, annotated with semantics (e.g., "this part over here is a stop sign, this part over here is a person"). To provide more advanced navigational capabilities to our car, we will start by implementing a more advanced mapping algorithm -- to keep things simple, we'll work in two dimensions, though what you learn generalizes directly to higher dimensional analysis.

Objectives:

- You will implement a more general non-probabilistic mapping model
- You will emulate localization by incrementing position based on velocity readings.

Our car must use data from the ultrasonic sensor and potentially the camera to detect obstacles around it as well as their distances. To do this, you can[14] create a numpy array map data structure in your python code, where you store the data as a numpy array of 0's and 1's, where a 1 represents there is an obstacle within a certain threshold distance, and a 0 indicates there is not[15], and the array index indicates the angle (from the car's perspective).

[14] For some ideas on how to think about this problem, watch this video: https://www.youtube.com/watch?v=MUIy8s2Oo34 (also see https://youtu.be/aXL9dbChui4).
[15] On a real autonomous car, you would extend this idea to create a "point cloud" - measuring objects' distance in a 3D band around the car.

One way to perform the mapping is to scan the surroundings by reading distances from the ultrasonic sensor every few degrees, and interpolating distances in between those points as well to generate a rough map of the surroundings.

> For example, if you find the distance to the nearest object at 60 degrees on the servo to be 20 cm and the distance at 55 degrees to be 20 cm as well, you can consider those as two points on a coordinate grid, calculate the slope between them, and fill in all points between them that lie on that line as being "objects" in your internal map.

Imagine a 100x100 numpy array where each element represents a 1x1 cm grid. If your car is at (0,0)[16] and you find the distance reading on the ultrasonic sensor at 0 degrees shows 10 cm, we would mark (10, 0) as a 1, since that is where we find our first object. Variants of SLAM (Simultaneous localization and mapping) algorithms can also be used for this step[17].

Here (mirror-MOV) (mirror-MP4) is a quick walkthrough of an example conversion from ultrasonic readings to a numpy array.

Step 7: Object Detection

Objectives:

- You will learn how to work with computer vision using OpenCV, leveraging the Raspberry Pi and camera.
- You will leverage neural networks to automate object recognition. You will be running lite CNN models, powered by TensorFlow, on a Raspberry Pi.

Your target is to perform real-time (around 1 fps[18]) object detection via the Raspberry Pi. The Raspberry Pi 4B has a dedicated camera port for high speed video transmission. The challenge lies in the image processing task. The first thing you will certainly want to do is to install the picamera module, which is a python module for transforming video

[16] Now, one issue with using 0,0 is that if you turn in certain directions it will send you into negative x,y coordinates - totally something you could handle but makes things a bit more complex as array indices can get negative. Another option would be to start the car being some distance over in the x direction and 0 in the y direction.
[17] The algorithm you're implementing here is a simple SLAM algorithm - SLAM algorithms can get much more complicated, and you should definitely feel free to do something more advanced here if you would like to explore, but doing something more advanced than what we describe here is not required.
[18] If you wanted to go to higher framerates, you could consider leveraging platforms like Intel Movidius or an NVIDIA Jetson Nano to offload CNN processing to hardware. This would be helpful if you were planning to drive your car around quickly, as fast reactions could help it avoid a crash.

feed into numpy arrays. In the old days, you would need to build the python wheel on Raspberry Pi from scratch (which takes a lot of work), or cross compile the module against the ARM architecture used by Raspberry Pi (which is faster, but still hard to manage the dependencies). However, today you can leverage pip to install these modules with predefined build flows. Simply type `pip install --user picamera[array]` and you would be able to process the camera feed via numpy in real time. (Also you can try to test your camera by taking a still picture using the raspistill command in the CLI.)

We suggest you use OpenCV for image preprocessing and TensorFlow Lite's Interpreter API for object detection. Although we use pre-trained models only for this use case, it can still be challenging to implement the object detection feature if you don't have previous experience with TensorFlow and computer vision. So, we recommend that you read the following reference materials to understand the basics of leveraging TensorFlow in object detection applications:

1. Installing Raspberry Pi Camera:
   https://www.raspberrypi.com/documentation/accessories/camera.html#installing-a-raspberry-pi-camera
2. Checking Raspberry Pi Camera
   Test with command: **libcamera-hello**
   https://www.raspberrypi.com/documentation/computers/camera_software.html#introducing-the-raspberry-pi-cameras
3. How to set up and use Tensorflow Lite on the Raspberry Pi:
   https://www.tensorflow.org/lite/guide/python
4. How to set up and use OpenCV on the Raspberry Pi:
   https://qengineering.eu/install-opencv-on-raspberry-pi.html

```
# check for updates
$ sudo apt-get update
$ sudo apt-get upgrade
# dependencies
$ sudo apt-get install build-essential cmake git unzip pkg-config
$ sudo apt-get install libjpeg-dev libpng-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt-get install libgtk2.0-dev libcanberra-gtk* libgtk-3-dev
$ sudo apt-get install libgstreamer1.0-dev gstreamer1.0-gtk3
$ sudo apt-get install libgstreamer-plugins-base1.0-dev gstreamer1.0-gl
$ sudo apt-get install libxvidcore-dev libx264-dev
$ sudo apt-get install python3-dev python3-numpy python3-pip
$ sudo apt-get install libtbb2 libtbb-dev libdc1394-22-dev
$ sudo apt-get install libv4l-dev v4l-utils
```

```
$ sudo apt-get install libopenblas-dev libatlas-base-dev libblas-dev
$ sudo apt-get install liblapack-dev gfortran libhdf5-dev
$ sudo apt-get install libprotobuf-dev libgoogle-glog-dev libgflags-dev
$ sudo apt-get install protobuf-compiler
# install command
$ pip3 install opencv-contrib-python
```

5. Example code (use this code to get started, you should build off of this)please consider just repurposing this code, it will save you time): https://github.com/tensorflow/examples/blob/master/lite/examples/object_detection/raspberry_pi/README.md

As a first step, try to get a picture of a stop sign recognized. To test this, you can pull up a picture of a stop sign on your cell phone, and hold it several inches in front of your car's camera.

Now we have the camera feed, which should be processed via TensorFlow. However, this is still not enough. Remember that Raspberry Pi is rather constrained in its computation resources, and we need to choose a suitable CNN model for image recognition. Quantized, low precision models, which leverage 8 bit integers to replace floating point operations, are most suitable for inference in mobile devices. Luckily, they provide good performance for the Pi as well . The example above uses Coco[19], a basic pre-trained image recognition model, but you can try loading different models. You may also want to train the model by yourself[20], if you want some more specific image recognition tasks (i.e. increased accuracy on certain objects you might encounter on a street like certain road signs, traffic cones, etc) . However, that takes a lot of GPU cycles, so most likely could not be finished on a laptop (and definitely not on a Pi).

Note: You should notice that once the application starts running, the Pi will most likely overheat due to high CPU utilization. The CPU will then decrease its frequency (and even shut down itself) for self protection, which leads to significant performance drop in your application. You may want to buy a fan on the device sheet[21] (or simply open the window

[19] Just so you are aware, as it may be useful for future projects you do, Tensorflow's website has a number of other models that can be downloaded. These models differ based both on what they are able to do (e.g., object detection, segmentation, image classification) as well as performance (e.g., quantized, floating point). For more details see this link.
[20] You certainly don't need to do this - the default Coco model should work pretty well for this lab actually. However if you want to learn more about training models you can feel free to give it a try. See this link to learn more.
[21] If you would like to have even more additional components, eg for another project, you can purchase an entire kit for example via this link.

to let the cold wind in Champaign do the work, depending on the season. Just kidding.) to keep the CPU performing well.

Equipped with the above prerequisites, you can start writing the object detection application. There are certainly a lot of methods to do it, so we'll leave the creativity to you! Since the Pi is low in computation power, you should not expect it to be able to process the video with the standard 25FPS. Actually, the best you could get with current Python module support should be around 1 FPS. Optimizing your code to reach this level will be challenging but fun. Ultimately, we want you to derive the best practice in your report to build a real-time video processing pipeline on the Pi. In your report, answer the following questions:

1. *Would hardware acceleration help in image processing? Have the packages mentioned above leveraged it? If not, how could you properly leverage hardware acceleration?*
2. *Would multithreading help increase (or hurt) the performance of your program?*
3. *How would you choose the trade-off between frame rate and detection accuracy?*

Once you have a working Pi that you have connected the Picamera module with as well as working tensorflow code, the last step is using the ultrasonic sensor in conjunction with the chassis to scan your surroundings and map out a path to the goal. For the purposes of this lab, we can set the target to be a relative distance away from the starting point, i.e. 10 feet north and 5 feet east of the starting point. Using the picar-4wd library code, you can write wrapper functions that move the car a certain distance in a certain direction when called.

Step 8: Self-Driving Navigation (Routing)

Overview:
When we want to drive somewhere, we simply put the address into Google Maps and let it tell us where to make turns and which roads to take. Commercial systems like Tesla's autopiloting use similar global planning methods for general navigation, but when changing lanes or making tight turns, following such global methods won't be very useful. Instead, the system must be able to generate a more granular map and a local plan on top of that to follow to reach the goal. An overview of the various autonomous vehicle methods being used commercially can be found here. These systems have millions, if not billions of miles of driving data to train and refine their models on, which we unfortunately do not have the luxury of.

Objectives:

- Utilizing the advanced mapping from step 5, you will implement graph search algorithms to find a path from a given starting point to a goal.

Our mapping can be interpreted as a graph where each coordinate is a node and the edges represent the movements the car can make to each surrounding node. For the purposes of this lab, we can consider 4 possible moves from each node: up right left down.To find a path towards the target, variants of the A* algorithm can be run on the map we generated in the previous step. Like breadth-first-search, A* finds the shortest path between two points but it expands on fewer nodes than BFS and uses a heuristic (such as distance from the goal) to prioritize certain paths, hence using less memory and running faster. Given our 2-D map of the environment of 1's and 0's, each "edge" is a possible move in a certain direction, i.e. forward, left, right, back, and we seek to find the shortest list of moves that will get us to the target. More information on A* and pathfinding algorithms in general, can be found here: http://theory.stanford.edu/~amitp/GameProgramming/.

However, note that once we find the optimal path, we can only follow it to a certain extent, as we must keep updating our map as we move, since we don't know what is behind the obstacles that we move around otherwise. One way to do this is to build a map, run A* to find a route, and follow it for x number of steps before rebuilding the map. That way, we keep updating our knowledge about our surroundings while routing our car. While doing so, we must keep track of the ar's position as well with respect to the target, as that will affect our routing. Variants of A* such as D* could be useful as well, and there are multiple solutions for this problem. Note that you also need this routing to work in conjunction with the tensorflow code; namely, if you detect a person in front of you, please wait until that person is gone before moving!

A good way to test your code is to give the car a target of "5 feet forward, 5 feet right", and place objects in the route it would take and see if it adjusts its path around them and gets to the correct destination. In addition, if you suddenly place a picture of a person in front of the car, it should halt until that person is gone, and continue its normal route.

Obstacle avoidance: You may have noticed that our routing algorithm doesn't account for the space the car occupies. Although there are ways to mitigate this problem algorithmically (example)[22], there is an even simpler workaround for the purposes of

[22] Do not worry - this video shows a fairly advanced approach, showing you the power of A* - in this lab, you will make use of a simpler variant, which will still give you some good experience with A*.

this lab. When we are building our surrounding map of obstacles, we can simply mark each obstacle with a clearance (i.e, marking a certain radius of surrounding cells as done in this link). This prevents our routing algorithm from trying to fit through gaps that are too small.

Figure 13: Examples for adding clearance to obstacles. Left: without clearance, Right: with clearance.

Step 9: Full Self-Driving

Overview:
Now that your car can detect objects and plan routes, we would like to test out the overall integration of your fully self-driving car.

Objectives:

- You will extend your obstacle course with various sorts of objects (stop signs, traffic cones, etc).
- You will let the car self-drive on this extended obstacle course.

To construct an effective obstacle course, you'll need obstacles that look realistic enough to register in OpenCV. You are certainly welcome to use real ones, maybe you can get small ones from a dollhouse or something, but the simplest approach might be to print out pictures of such objects and mount them on cardboard. Here are some example graphics you can use, but you don't have to use these, feel free to get creative!

Maybe you could take a picture of yourself and print yourself out and mount yourself as a little person on cardboard - is your car smart enough to drive around you?

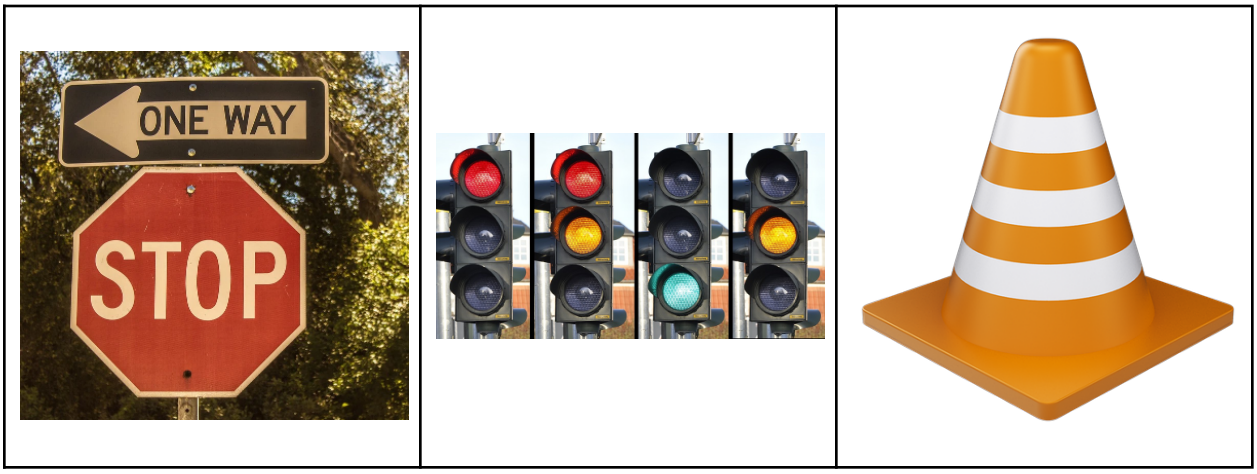


Figure 14: Sample graphics you can used to test: Stop sign, traffic lights, traffic cone [23] ***. You can cut these out and tape them up, maybe mount them on cardboard, to make your obstacle course.***

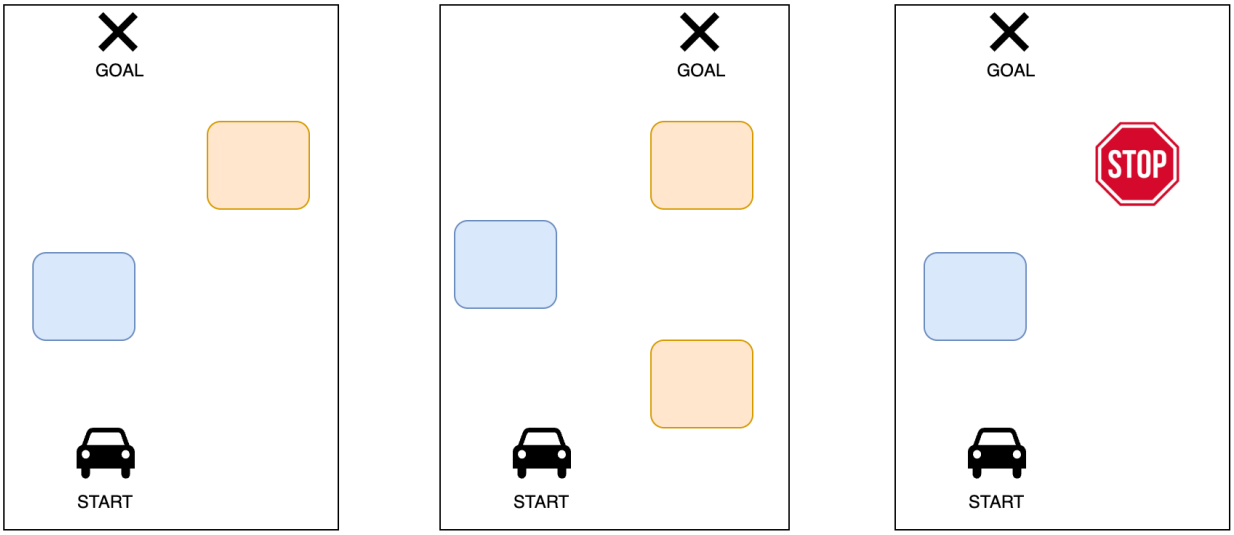


Figure 15: Example obstacle course setups. The boxes are the obstacles and the X is the destination goal. You can be creative with your obstacle course, it doesn't have to be exactly the same as the ones above.

[23] More graphics can be found on Pixabay (e.g., https://pixabay.com/photos/traffic-light-signal-traffic-street-876056/), Google image search, etc.

Part 2 Submission and Grading

Like part 1, you will create a **demo video** (under 10 mins). The video should be hosted privately on cloud, with access permission given to the course staff. **Do not upload the video to Coursera**. The demo video should contain:

1) 4 main points in the rubric.
2) The code you wrote as well as a walk through

Your final submission should be a **report** using this template in PDF format**. Please include NetIDs of all group members.** The report must contain:

1) The link for the demo video
2) Your design considerations for each steppart of the lab

Demo Video (70 pts)	
Advanced Mapping: Are you able to generate a map indicating obstacles? (10 pts) This could be a 2D binary array indicating 1's and 0's for obstacle and no-obstacle, or vice versa. To get more innovative you could use OpenCV to build up a map image using the binary array - something like red pixels where an obstacle is detected, green elsewhere. A perfect submission should show the car and the obstacle(s), the car scanning its surroundings, and the map it generates in one continuous video.	
Students show that the car is able to scan and generate a map of the surroundings with at least one obstacle present in the periphery.	10
The map isn't correct but the students have correct code that they can explain and pin-point possible issues.	4
Object Detection (10 pts)	
The car can correctly detect objects and students demonstrate some ideas that they tried so to speed up the fps.	10
The car can detect objects but identifies them incorrectly (wrong labels)	6

Nothing works but the students show what they tried and the issues they faced	1-3
Self-Driving Navigation (Routing) (20 pts) A perfect submission should satisfy the following requirements: 1. The destination is clearly marked. You can use any format for the destination, such as specify the (x, y) end goal coordinate when you run your code. 2. There are at least 2 obstacle obstructing the car 3. The same recording must show the car navigating to at least 2 different destinations with different obstacle setup. This will help us verify that the routing and navigation indeed works. 4. Routing is done with A* or derivation of A* 5. The car must periodically rescan and recompute its mapping and routing as it navigates the course.	
The car can correctly route to destination in both runs, while avoiding the obstacles.	20
The car can correctly route to both destination but run into obstacles occasionally	15
Routing does not seem to work at all, but students show the car should be able to sense its environment and calculate the optimal path using A* by showing correct code, and can pin-point possible issues.	0-10
Routing is done without using A* or derivation of A*, or without rescans, or with obstacles oriented in a way that makes routing trivial.	0
Full Self-Driving: Is the car able to navigate the complete route correctly along with recognizing a sign (eg. stop sign) using the camera and performing necessary maneuvers? (20 pts) A brief description of what a necessary maneuver means - If the car arrives at a stop sign, then the car must come to a halt. It should know that a stop sign is not an obstacle that it can maneuver around. Moreover, it is not necessary for the sign to obstruct the path of the car for it to come to a halt. Even if a stop sign is on one side of the path, the car must stop since the camera will capture the sign. This is just an example, you can have your own set of signs and traffic rules (please don't let skipping traffic signals be one of them!) A perfect submission should satisfy the following requirements: 1. The destination is clearly marked.	

2. Students should explain what traffic sign(s) they choose to recognize and what the correct response should be. 3. Full self-driving is done with at least 1 traffic sign and 1 obstacle that clearly obstructs the car	
The car can correctly route to destination while avoiding at least 1 obstacle and correctly recognize and react to at least 1 traffic sign	20
The car can correctly route to destination but occasionally runs into the obstacle, or the car is slow to react to the traffic signs (perhaps due to low FPS)	15
The car can correctly route to destination, but object detection does not work. Students demonstrate that the code is correct and should work and can pin-point possible issues.	10
Routing does not work or is implemented incorrectly (eg. did not use A*), but the car can recognize the traffic sign(s) and react correctly.	5
Code Walkthrough (10 pts)	
Students clearly show and explain the code they wrote	10
Students explain the code without showing, or show the code without explaining, or show and explain only part of the code	0-5
Report (20 pts)	
Report Comprehensiveness (20 pts)	
Report contains detailed information about the students' thought process, such as how they approach the problem, how they reach their final implementation, the challenges they faced, etc.	20
Report is not comprehensive enough	0-19
Submission Format (5 pts)	
Report is submitted in PDF format, video is hosted privately on cloud with access permission granted and does not exceed the 10 minutes limit	5

Category	Points
Video	70
Report	20
Submission format	5
Total	**95**

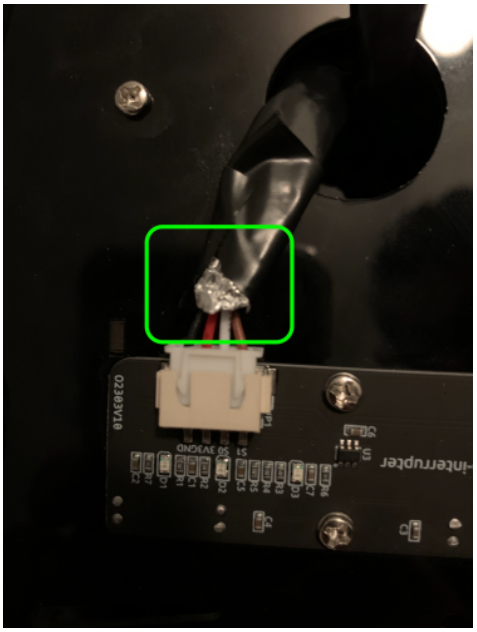
Frequently Asked Questions

My ultrasonic sensor is not providing good readings, and/or my car keeps stuttering!

- Please twist your wires, cover them with aluminum foil, and then with electrical tape.
- Add blinders around the "eyes" of your ultrasonic sensor. Make sure your obstacles, the floor, and the wall are not reflective.
- Unplug the grayscale sensor, and build your car without it
- Tom Pawelek FA 21 discovered a race condition in the source code. To fix it, at the top of the set_power() function, add the following lines:

```
self.pwm_pin.__init__(self.pwm_pin.channel)
self.pwm_pin.pulse_width_percent(power)
```

- "I believe that if you jam cables together, there might be some sort of interference in signals due to electromagnetic fields product of current traveling in different directions. Because signals are unlikely to go through an error correcting protocol, like TCP/IP for example, a single disruption might change the value of a reading. In order to prevent that, I covered each set of cables (not each individual cable) with aluminum foil, and on top, covered them with electrical tape. I was trying to simulate the cover of Ethernet cables. After that, stuttering practically disappeared and readings from things like the ultrasonic came back pretty accurate, false positives went down like 95% or so. Additionally, since I have two set of cars, I changed the cable intended for the Ultrasonic with the one intended for the photo interrupted of the second car. It is a little larger and allows more room to move the US while scanning without potentially pulling the sensor due to a short cable." - Fauzi Gomez, FA 21

- 

The servos/motors on my car aren't working. (thanks Christopher Fischer, Matt Williamson, Kim Westfall)

- Make sure your screws aren't too tight.
- Could be a power issue - make sure your batteries are fully charged; try changing power adapters.
- Make sure all your wires are seated firmly (connected well). For example, check to make sure the wires to the power supply are not loose.
- Try resetting your car (e.g., "picar-4wd soft-reset")
- If your servo works but poorly, consider ordering an MG90S (eg https://www.amazon.com/Replace-Helicopter-Airplane-Controls-Vehicle/dp/B09KXM5L7Z/) and see if that works better.

I have a motor that is going in the wrong direction.

- Try reversing polarity on the input wires
- If you use the "Speed" command/class, try taking that out.

My components arrived but something is damaged.

- Many manufacturers will ship you free replacements. Contact the manufacturer and let them know what happened.

My SD card isn't working. I'm getting a Flash failed error or some similar sort of problem.

- Try formatting your flash card again using the Raspberry Pi imager (https://www.raspberrypi.org/software/)
- Also see https://forums.balena.io/t/flashing-raspberry-image-to-sd-card-always-fails/3091 or https://www.raspberrypi.org/forums/viewtopic.php?t=196472

My car keeps running out of power.

- While you're developing/debugging, you can power the Raspberry Pi from the USB-C, to avoid draining your batteries.
- Your batteries may have arrived damaged. Consider purchasing a more expensive version of the 18650 batteries, eg from here).

I'm getting errors like ImportError and ModuleNotFoundError

- Make sure you ran the install command, eg "sudo python3 setup.py install" (and make sure you put "sudo" before that command)

I'm having trouble getting VNC working

- If you have a spare monitor, and keyboard/mouse, you can connect it to your Raspberry Pi. Many TVs have HDMI interfaces too, so you could also try connecting that if you don't have a spare monitor.

Video Length

- The video should not be more than 10 mins in length

I'm having trouble getting RP set up in Headless mode
I'm using:

- RPI 4 Model B, 32GB SD Card, Mac OS Ventura 13.1, RPI Imager v1.7.3
- For OS>Other>Full 32-bit
- Gear Icon>
    - set hostname
    - Enable SSH password
    - set username/pass pi/raspberry (for backwards compatibility, not sure if necessary)
    - Configure Wireless LAN, Enter your SSID and Password
    - set locale settings
- My previous efforts with adding the other steps regarding SSH, wpa_supplicant.conf and userconf.txt didn't let the RPI4b to boot.
- I have an Apple Airport, which seems to intentionally make it difficult to find ip addresses for devices. After some time the RPI4 showed up Airport Utility, but initially I used arp -a

- Edit: Also followed these steps to make a backup image. https://blog.jaimyn.dev/the-fastest-way-to-clone-sd-card-macos/
- Edit: Easier way to find ip address is to use the command ping raspberrypi.local See here for additional details. https://www.raspberrypi.com/documentation/computers/remote-access.html#remote-access

I want to run Fritzing but I have a Mac.

- If you want to get the free version of it for Mac, you can go to Github and find the old version (which is free) for Mac.https://github.com/fritzing/fritzing-app/releases