UIUC - CS 437 - Internet of Things

Lab 1 Report

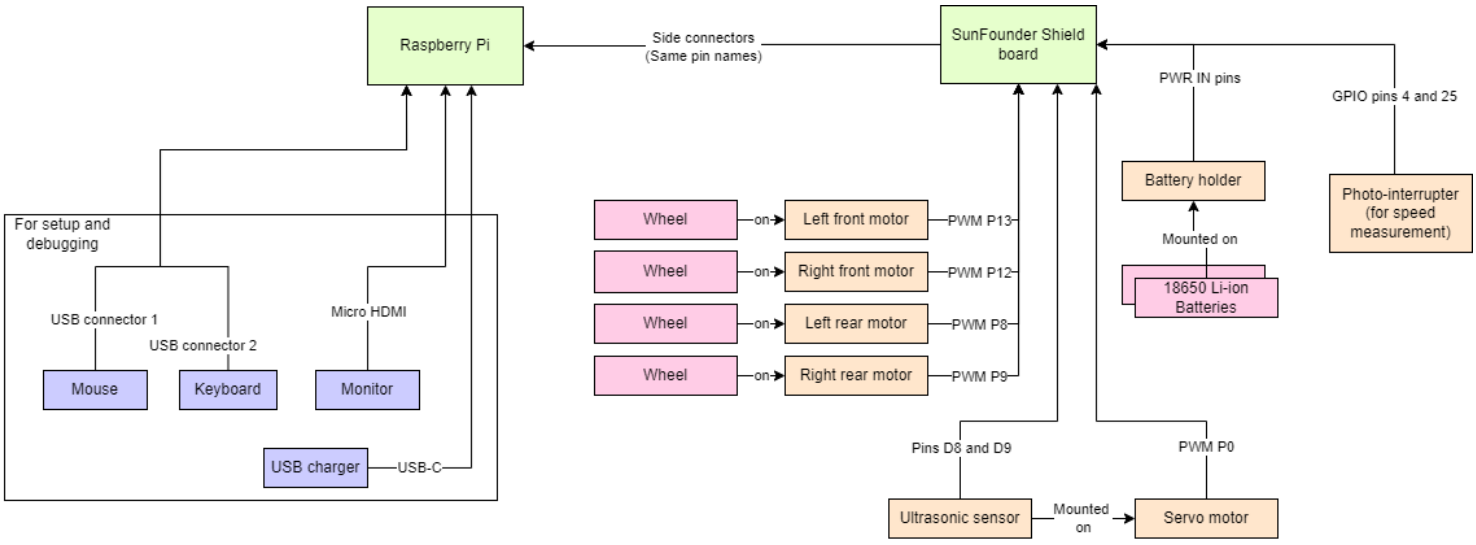
Rodrigo Mendoza - rm36
Iziren Okhamafe - iokha2

Introduction

This project shows how we managed to build and program a self-driving car which could see its environment and stop when necessary. The video can be found in this link. The code can be found in https://github.com/rm36/CarNavigation.

Part 1

1. **A topology specifying the wiring of all your devices**



2. **Your design considerations for each part of the lab**
    a. **Assembly**: We used the SunFounder picar-4wd and the Pibiger camera. Due to the short camera cable and form, it was strapped to the front-right of the car, to see signs on the side. The rest of the system was assembled per the instructions, and the code worked out-of-the box with some minor hiccups (e.g. direction of wheels for forward).
    b. **Moving and turning**: The car's forward(speed) function didn't specify a particular distance, and this was calibrated by moving the car forward for N seconds, then dividing the distance moved by N to compute the

distance per second for each speed supported. Similarly for turning, we calculated the turning time for 90 degrees and based the turning function on that amount. This is part of the 'calibration' code in the github repository.

c. **Camera perception**: We based the project from the tensorflow lite example for object detection in raspberry pi. The 'efficient' version of the detection neural net comes pre-trained with multiple objects, including 'stop sign', and 'person'.
d. **Navigation**: The navigation algorithm we used is A* search, which considers nodes based on a heuristic-based cost. The cost was initially the distance from the current cell to the target location, but it proved to sometimes go back and forth close to the obstacles. Penalizing for distance moved resulted a good thing in practice, so despite the extra time searching, the heuristic-based cost was determined to be diagonal_distance_to_target + k * total_distance_moved_by_car with k=0.5. Increasing k would increase the search space too much for practice.
e. **Quality assurance**: Despite this being a project that might not be maintained in the future, unit tests proved to speed up the development process by guaranteeing correctness. From all the issues we encountered, less than 10% were on the unit-tested code. Furthermore, most of the code was split into small parts to debug one granular module at a time.

3. **Anything you did not have time to show/mention in the demo video**
    a. Frame rate improvement: We tried some techniques to improve the frame rate, including:
        i. Use the most efficient neural net available for object detection.
        ii. Doing the distance computation asynchronously; instead of calling car.forward(speed) and then waiting for some amount corresponding to the distance required, the distance traveled was computed after the frame was processed, by computing first the elapsed time.

Part 2

1. **Would hardware acceleration help in image processing? Have the packages mentioned above leveraged it? If not, how could you properly leverage hardware acceleration?**
    a. Yes, hardware acceleration would help enormously for the camera detections. We didn't buy any specialized hardware, but using another

module for hardware acceleration could've given the driver.py module more time to work on the navigation solving and SLAM.

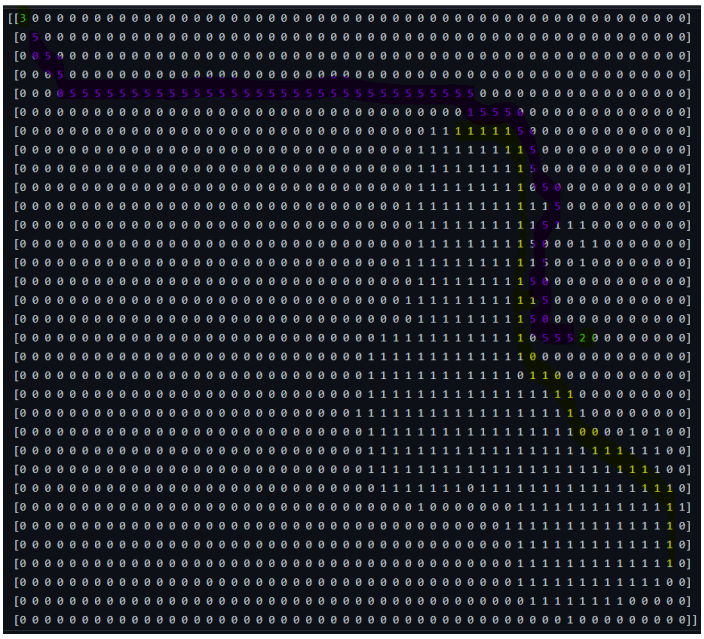
2. **Would multithreading help increase (or hurt) the performance of your program?**
    a. Multithreading would only increase the performance, because while the detections are being computed in the neural net threads, the navigation path algorithm could run on another thread. Also, by running the object detector itself in multiple threads, the frame rate would increase (e.g. by running some convolutions on each thread and then joining the results).
3. **How would you choose the trade-off between frame rate and detection accuracy?**
    a. There is a rule of thumb that says that premature optimization is not a good thing. Because the detection code ran properly with our test conditions, there wasn't need to optimize it too much. However, in a production self-driving car, detection latency could mean saving someone's life. So I would choose to speed up detection and post-processing as much as possible while keeping the accuracy high. A drop in accuracy could mean harming the user as well. Here's one way: add multiple hardware platforms for different levels of latency and accuracy all running at the same time. Then as soon as a low-accuracy-low-latency system detects something consistent with a high-accuracy-high-latency system, it has more confidence on it, and acts quickly. The trade-off is high system hardware and development cost.

Rubric questions

1. **Is the car fully assembled?**
    a. Yes, following the diagram above.
2. **Can the car move by itself?**
    a. Yes, after starting the program via ssh (python3 project.py) it navigates to a particular coordinate objective.
3. **Can the car "see"?**
    a. Yes, in two ways: via ultrasonic measurements and with camera detections.
4. **Can the car "navigate"?**
    a. Yes, it'll start with the command and stop when it arrives at its destination.
5. **More Advanced Mapping: Are you able to generate a map indicating obstacles?**
    a. Yes. The map isn't perfect due to the quality of ultrasound measurements, but it creates a map based on the detections with multiple heuristics, including:

i. Non-linearity of results with respect of distance (objects farther away return a smaller value than they should).
ii. Consecutive detections are considered part of a single bigger obstacle as long as they're close enough (within a certain distance).
iii. Each detection is 'inflated' in grid space with a certain clearance value. A more probabilistic approach would be better (e.g. inflating mostly in the axis perpendicular to the ultrasound normal vector), but for our solution a circular blob was enough.

6. **Object Detection: Are the objects in the camera feed detected correctly?**
    a. Yes, the quality of the objects wasn't perfect but the objects we wanted to detect had a label as well as x,y relative to the camera plane.
7. **Self-Driving Navigation (Routing): Is the car able to route correctly?**
    a. Yes, see the demo portion of the video and the map generated by the A* algorithm with the path to follow. Here is an example grid (colored manually) that was computed online during the demo in the video. (green: car and target, yellow: obstacles, purple: route computed)



8. **Testing: 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?**
    a. Yes, it wasn't exactly a stop sign, but it did avoid it, as well as detected it and stopped when it was supposed to. The logic followed was to wait a little time after detection to stop, such that it would stop at the right location.

Conclusion

This was a complicated project because there were so many parts involved. By far, the most challenging feature in this lab was the timeframe due to the issues finding parts (raspberry pi, batteries, etc). However, following good practices such as the following saved us time and our project worked as expected: 1. modularizing the code into small parts, 2. doing unit testing, 3. reusing platforms to avoid reinventing the wheel.