CS 498 Internet of Things
Lab 1

Team Ace
Alexey Burlakov (NetID: alexeyb2)
Christopher Lewis (NetID: calewis4)
Li Yi (NetID: liyi2)
Pui Sze Pansy Ng (NetID: ppn2)
Zhijie Wang (NetID: zhijiew2)

Table of contents

1. Introduction

In this lab, we implemented a vehicular network and computation infrastructure which is modeling 2019 Honda Civic. The infrastructure contains Raspberry PI 3 B+ as ADAS, two Arduino Uno as microcontrollers, Pi camera as back end camera, ultrasonic sensor and LED light as tail light. The infrastructure performs object detection via pi camera and uses the ultrasonic sensor to calculate the distance. The LED light will turn on if a human is detected and the distance is less than 50CM.
Video is uploaded and shared on Google Drive.
Here is the GitHub link for our program:
https://github.com/alexeyb2-illinois/CS498_IoT_Lab1
https://github.com/yilimail/CS498_IOT_Lab1
Libraries:

MCP CAN https://github.com/coryjfowler/MCP_CAN_lib

2. Method

2.1. ADAS

We use Raspberry PI 3 B+ as ADAS. Back end camera is connected to the raspberry pi via picamera. OpenCV-python and tensorflow are installed to perform object detection.

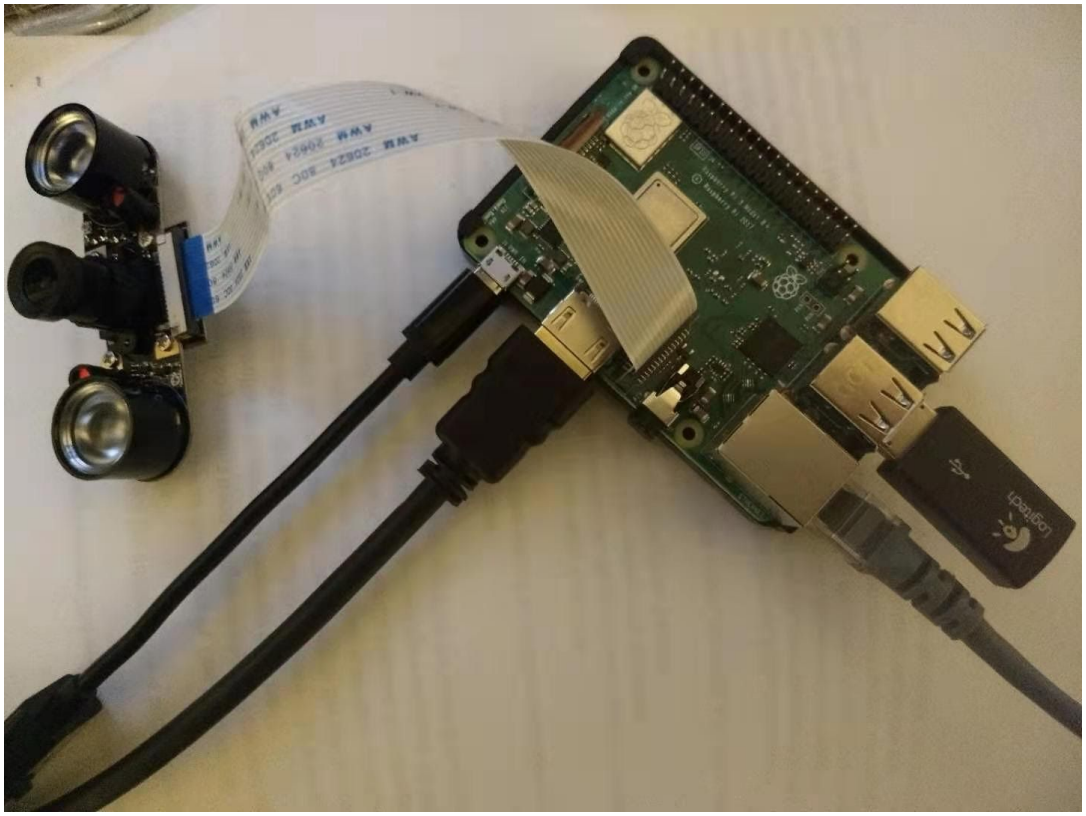

Figure 2.1.1 Raspberry pi 3B+ connected with picamera

2.2. Microcontroller

We use Ardunio Uno as a microcontroller. It connects with an ultrasonic sensor and an LED light. In terms of data transmission, we have MCP2515 CAN bus module connected to Ardunio Uno, so that distance data and LED status can be transmitted by CAN bus. Please see our circuit diagram for detailed information about connections and digital Pin assignments.

2.3. CAN to IP gateway

We connected the Ethernet shield to Arduino Uno board to give Arduino Uno ability to transmit data by UDP. After installing an ethernet shield, we plugged in MCP2515 CAN module on Arduino Uno board, connect with another MCP2515 CAN module. The other MCP2515 CAN module is connected to the ECU Arduino Uno board. With correct Pin assignment, and CANL and CANH connection, distance data and LED status can be transmitted to CAN to IP gateway. Next step is plug ethernet cable to Ethernet shield. Using an ethernet switch, we interconnected RPi, CAN to IP gateway Arduino board, and computer. On each device, we set a static IP address and port number. Every device can receive and send UDP packages.

2.4. Head Units

In this part, We gathered all the parts from the above: ADAS, Microcontroller, CAN bus and created a dashboard which is a front-end to visualize all the data.

3. End to End Performance Measurements

We are going to measure it in three different scenarios: people moving toward the camera, people moving away the camera, objects move toward, and objects move away

People move towards

	FPS	Detection Accuracy	Distance
Test 1	0.78	0.87	200cm - 5cm
Test 2	0.77	0.87	200cm - 80cm
Test 3	0.78	0.70	80cm - 30cm
Test 4	0.79	0.62	50cm - 5cm
Test 5	0.80	0.54	30cm - 5cm

People move away

	FPS	Detection Accuracy	Distance
Test 1	0.80	0.72	5cm - 200cm
Test 2	0.79	0.77	80cm - 200cm
Test 3	0.79	0.72	30cm - 80cm
Test 4	0.77	0.66	5cm - 50cm
Test 5	0.73	0.55	5cm - 30cm

Objects move towards

	FPS	Detection Accuracy	Distance

Test 1*	0.79	0.67	200cm - 5cm
Test 2*	0.74	0.52	200cm - 80cm
Test 3*	0.81	0.46	80cm - 30cm
Test 4**	0.80	0.68	50cm - 5cm
Test 5**	0.78	0.66	30cm - 5cm

Objects move towards

	FPS	Detection Accuracy	Distance
Test 1*	0.80	0.75	5cm - 200cm
Test 2*	0.82	0.63	80cm - 200cm
Test 3*	0.76	0.40	30cm - 80cm
Test 4**	0.75	0.65	5cm - 50cm
Test 5**	0.76	0.68	5cm - 30cm

Note: * Used a chair to perform Object Detection tests
** Used a bottle to perform Object Detection tests

4. Discussion on Design Considerations

4.1. ADAS (Raspberry Pi Programming)

4.1.1. Why are quantized models better for resource-constrained devices?

Many resource-constrained devices may not have powerful hardware FPU (float point unit), or no kernel level support for hardware FPU. Using soft FPU is more computational intensive (processing same numbers in Floats will use more cycles than processing it in Ints). In addition, devices are constrained by bandwidth allowances especially in cases of mobile networks such as in cars as described in this lab. Quantization reduces the model size and therefore accommodates for bandwidth limitations.

4.1.2. Would hardware acceleration help in image processing? Have the packages mentioned above leveraged it? If not, how could you properly leverage hardware acceleration?

Hardware acceleration can significantly help image processing, especially leveraging GPUs. GPUs are designed to slice and dice image data (as in the forms of matrix) with its parallel processing capabilities. The Tensorflow model could leverage NVIDIA GPU by default. Raspberry Pi has an ARM chip with Mali GPU, which is only compatible with OpenCL interface. Tensorflow default did not support OpenCL. We did not explore ARM specific toolkits for Tensorflow/ONNX model conversion and offloading on RPi. Devices like Nvidia Jetson offers ARM CPU and Nvidia GPU in one SoC or Google's TPU edge unit in USB form factors could be leveraged for accelerated image processing and model inferencing.

4.1.3. Would multithreading help increase (or hurt) the performance of your program?

There are multiple factors affecting the performance of the program, such as the image decoding etc. On a resource-constrained and low cost device, the TDP (thermal design point) is a limitation factor. As the chip approaches TDP, cores will be slowed down (thermal throttling) to prevent system failure. So multithreading can help initially, but the long term performance impact is negative.

4.1.4. How would you choose the trade-off between frame rate and detection accuracy?

This trade off is primarily a result of the design & outcome goals of the system being implemented. You would have to carefully assess all common cases and especially edge cases and weigh the risks and benefits associated with balancing detection accuracy and frame rate. In an ADAS, we would err on the side of safety.

The question then becomes, "Who's safety?" Do we prioritize the safety of the people inside the car or outside the car, animals, or humans? As we examine our model we may find that detection of larger objects such as other cars is easily achieved at a low frame rate with persistent accuracy and that smaller objects require a higher frame rate but are less significant when considering our design goals.

We would also need to examine the amount of time required to safely avoid the obstacle. At 1 fps it is possible that identification of an object posing significant risk to the occupants of the vehicle would be detected too late to implement any appropriate responsive action.

So, as with all engineering decisions achieving the design goals, with minimal risk, high efficiency, and a test and validate approach would be most appropriate.

4.2. Microcontroller (Arduino programming)

4.2.1. Calculation of distance between the sensor and objects

Speed of sound v=340 m/s = 0.034 cm/us
Time = distance/speed t = s/v
Distance in cm:
S = t* 0.034/2

4.3. CAN to IP gateway

4.3.1. Will you use UDP or TCP for this use case? Please briefly explain why in your report.

I would use UDP for this use case. UDP is connectionless and has a higher transfer rate. With many devices on the same bus, it is more important for critical devices to be able to transmit the signal (such as collision detection sensors) than TCP's guaranteed delivery mechanism.

4.3.2. Can you realize full-duplex communication between the CAN ECU and the Pi? Please give your reasons in your report

Full-duplex communication stands for simultaneous data transmission in both directions. In CAN bus there can only be one active transmitter at a time, so it is only capable of half-duplex communication. However, UDP is a full-duplex communication protocol as it can use different ports. In our project, UDP communication is between IP_CAN_Gateway and RPi, CAN Bus communication is between ECU and IP_CAN_Gateway. So between CAN ECU and the Pi, the communication is not full-duplex.

4.4. Head Unit (Dashboard)

4.4.1. Can you make the time interval smaller?

It is possible to achieve a framerate of 1 FPS or higher if we try to render camera frames with lower resolution, use a more performant object detection network, use video streaming or offload frame rendering task to another device. In my setup

Raspberry Pi acts as an HTTP server and saves camera frames with bounding boxes as a base64 encoded string that can be requested from any device. It allows me to achieve 0.8 FPS with a colored 640x480 picture or 1 FPS with a grayscale 320x240 picture.

4.4.2. How sending data over the network influences the Pi's performance?

Raspberry Pi receives distance data from an Arduino with an Ultrasonic sensor by UDP and broadcasts LED status as soon as a new frame is processed also by UDP. This communication is very performant and is not really hindering the performance of Pi. My Raspberry Pi also acts as an HTTP server that the Head Unit requests the status (as well as camera feed) from every 1 second - that should be the most performance-heavy part in terms of a network. However, even by decreasing request interval to 50 milliseconds or having multiple devices request the data from Pi I am still getting roughly the same rendering rate (about 0.8-0.9 FPS) and CPU load. My guess is that in my particular solution object detection and image conversion along with encoding are the most resource-demanding tasks and network communication is relatively simple since it comes down to serving cached values.

4.4.3. How should we interconnect them in the Ethernet network?

The only requirement is for all the devices to be in the same subnet, since the system relies heavily on UDP broadcasting - so it is not even necessary for all the components to know each other's IP addresses. The only exception is that the Head Unit must know the address of Raspberry Pi (ADAS) as the Head Unit (or multiple Head Units) should be able to request the status of the system via HTTP. It is possible to broadcast this information as well as camera feed through UDP, but I wanted to make the Head Unit as simple and universal as possible - in my solution it is just an HTML page that periodically makes GET requests to Pi in JavaScript - such a page can be opened on a laptop, on a mobile phone or any other device that has a modern web browser. The simplest solution to interconnect all the components (Raspberry Pi, Head Unit and Arduino that acts as a CAN to IP translation gateway) is to connect them all into a single Ethernet switch and give them unique IP addresses in the same subnet.

5. Device Topology