Team - Nerd Herd
Lab 1 Report

Alpri Else (aelse2)
Maxon Weis (mweis4)
Berkay Kaplan (berkayk2)
Daniel Kiv (dkiv2)
Ben Sisserman (bens3)

Advanced Driver-Assistance System (ADAS) Report

The ADAS, based on a Raspberry Pi 4, uses data from the rear view camera with a pre-trained machine learning model to detect obstacles and humans in view.

Usage of quantized model for resource-constrained Raspberry Pi

The usage of a quantized model, which substitutes floating point values in tensor calculations with integers, allows for more compact models which fit in the small amount of RAM and leverage vectorization for faster model computations.

Software Stack

Software libraries used:

- TFLite
- Numpy
- PIL
- Websockets
- ReactJS
- Socket
- MCP_CAN_lib
- Arduino Ethernet

Sources referenced:

- TFLite object detection example
- TFLite performance best practices
- TensorFlow Lite benchmarking
- Official Python guide to UDP communication

Best practices for real time video processing pipeline

Hardware acceleration

The current state of this project relies on a Raspberry Pi 4, a portable piece of hardware which utilizes Linux. While this hardware is useful for internet of things applications and is capable of outputting to some display device, it does not have the strongest capabilities for hardware acceleration, considering its low profile and low power consumption. This presents some drawbacks, but by making compromises and

implementing TensorFlow Lite instead of base Tensorflow, performance can be good enough for the sake of this project. However, since machine learning models tend to be computationally expensive, hardware acceleration could only serve to improve performance and efficiency, especially for object detection.

The Raspberry Pi 4 offers faster performance than its predecessors and comes with an upgraded CPU and GPU. The Tensorflow Lite performance is nearly doubled from the previous generation and achieves comparable performance to leading hardware such as Nvidia Jetson. The device uses built-in functions to read in RGB data from the *picamera* module at a specified time interval and processes the array input through the trained machine learning model. The output is a modified array that includes classification labels and boxes throughout the image, which is forwarded to the server to be displayed on a front-facing webpage. While the quality is certainly low, it allows for faster machine learning processing.

Multithreading

Multithreading is used heavily in applying the Tensorflow model for object detection, as this consists of operations which scale well with multithreading such as matrix multiplications. As a result, these operations are significantly sped up on the quad core Raspberry Pi 4.

Our object detection, head unit server code, and the break management code runs on different threads, which prevents any single one of these processes from being bottlenecked by another.

Tradeoff between frame rate and detection accuracy

Currently, our frames per second (approximately 5 fps) is relatively high. However the object detection accuracy is low, especially when detecting people. When this happens, the confidence tends to drop to around 55-60%. We chose higher framerate at the expense of detection accuracy.

Microcontroller Report

The microcontroller sends sensor data to the CAN to IP gateway and receives data from the CAN to IP gateway to light an LED.

Ultrasonic sensor distance calculation

```
long duration = pulseIn(ECHO_PIN, HIGH);
double SPEED_OF_SOUND = 0.0343;
int distance = duration * SPEED_OF_SOUND / 2;
// divided by 2 because the sound is travelling to the object and back
```

CAN to IP Gateway Report

The CAN to IP gateway translates CAN packets from the microcontroller to IP packets for the ADAS and translates IP packets from the ADAS to CAN packets to the microcontroller.

CAN-to-IP/IP-to-CAN translation implementation

In order to implement the IP-to-CAN and vice versa, we decided to use UDP packets to send 4B in a byte array representing a little-endian integer for the distance in cm. The Ethernet shield waits for a CAN msg and forwards that message using the Udp object from the Ethernet library to the ADAS. On the ADAS we bound the socket using the Python socket library and some examples from the Python wiki (https://wiki.python.org/moin/UdpCommunication). The ADAS waits for a UDP packet and decodes it into an integer, and sends a UDP packet back to the Arduino as a response. We can expand this implementation for additional devices by adding an encoding for CAN ID of the sending device within the message.

Why UDP?

We chose to use UDP because we are prioritizing speed over reliability. Given the context of the brake light, we want to be able to make in-moment decisions, so the superior speed of UDP over TCP is a huge benefit for moving vehicles. We also are not worried about possibly losing packets over UDP because the distance is transmitted constantly, and an accurate distance measure would be repeated over multiple

messages, thus there is no risk in losing a few packets over time, as long as the majority of the packets are received.

Full-duplex communication between CAN ECU and Raspberry Pi

We cannot achieve a full duplex between the Pi and the CAN bus, because we implemented our bidirectional communication via UDP. The limitation comes from the need for the Pi to “listen” for the UDP packet in the ADAS. While listening, the Pi is unable to use the port for communication to the Gateway. It is only able to transmit after it receives an incoming message and exits the socket.recvfrom() function. Now, if we had gone with TCP for communication between the Pi and Gateway, we would have been able to achieve full-duplex communication rather than our half-duplex with UDP. However, as we explained in the previous section, UDP is a better fit for this application.

Dashboard Report

The dashboard shows a real time video stream from the ADAS camera with object detection applied. In addition, the distance reported by the ultrasonic sensor is displayed below the video feed (with object detection results). When there is no video feed, the dashboard displays a message that there is no reception.

Optimal dashboard update frequency

From our camera capturing an image, to the TFLite model running on the image annotating it with object identifications, to the image being streamed over a WebSocket to the head unit, we observe approximately 5 frames per second.

We used the lightweight identification model mobilenet_ssd_v2_quant to maintain this framerate consistently on the resource-constrained Raspberry Pi 4.

This time interval was reduced by using a TFLite model over a Tensorflow model, which significantly increased the object classification speed.

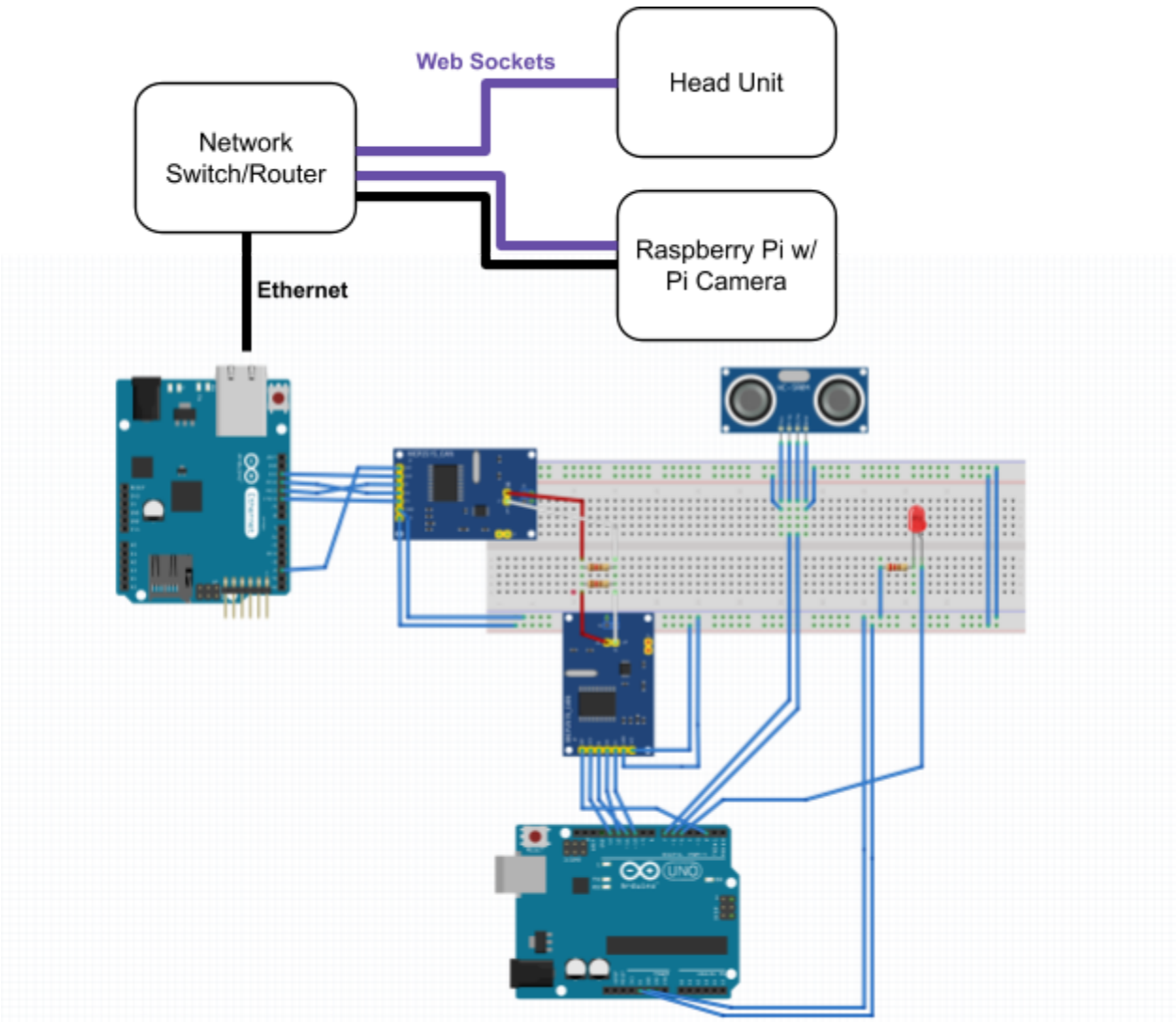
Method of interconnection of Ethernet devices

To interconnect our devices, we decided to use a local router and have communication over WiFi. The Pi 4 was able to connect via its Wifi adapter, and we did the same for the laptop viewing the Head Unit. The Ethernet shield was connected to the router directly by an ethernet cable. This was not our preferred method. We spent a significant amount of time trying to set up packet forwarding between the Pi and Ethernet Shield so that we could connect them via Ethernet directly. As we did not have any experience with networking, it took us time to identify that we needed IP forwarding to avoid the router. I actually posted on Piazza when none of the examples from the Ethernet library worked, and I was convinced that my Ethernet Shield was faulty.

After we realized that the Shield needed to be connected directly to a router for DHCP to correctly initialize the Shield, I started looking into IP forwarding to get the Ethernet shield to work on my Linux system. After a few hours of troubleshooting, I managed to initiate IP forwarding between the Shield and my laptop by disabling DHCP, manually setting the IP for the Arduino on my system (Pop_OS!) and setting the DNS and Gateway on the Arduino as well as the MAC and IP address. Unfortunately this

procedure did not translate from Pop_OS! to Raspberry Pi OS. We decided to use the router, since we would need a router to communicate with the Head Unit regardless of the Shield. However, we still want to be able to initiate IP forwarding between the Pi and Shield in future labs so that the Gateway Arduino can be mobile, rather than limited by the router.

Project Topology



Design Considerations

Part 1 ADAS

In order to transfer the image with annotations and boxes to the front-facing web page, we decided to use Pillow / Python Imaging Library (PIL) to combine the data into one array. This data was then sent over the Python websocket in a separate thread.

Part 2 Microcontroller

For this part we did not have many design considerations besides calculating the distance and converting it into an integer, which represented distance in centimeters. We chose to use centimeters, because integers are simpler to decode from a byte array, which is the format used in both CAN and UDP communication. This allowed us to standardize most of our code to byte arrays of length 4.

Part 3 IP to CAN

We considered how communication between the ultrasonic sensor and the ADASthrough the Gateway will work. We also decided to forgo identifying the device communicating via CAN, and assume that all incoming messages are of size 4B and represent distance. To activate the brake light, the ADAS returns int 1 to activate the brake or int 0 to turn the brake light off. Upon adding more devices to the CAN bus, we will utilize the CAN ID's to decode messages. This decision was made to increase the speed of decoding and reaction.

Part 4 Head Unit

The Head Unit was created using ReactJS given our team's experience using the framework. This also has the additional benefit of making extensions to our dashboard easy. Since we needed to continuously stream annotated images, we decided to use WebSockets to stream image data to the frontend (as opposed to using HTTP requests). This data is then consumed by the Blob Web API and converted to an object URL, so it can be rendered using an `<img/>` HTML tag. Distance information was also streamed to the Head Unit using Web Sockets.

Object Detection Performance / Results

We have 5 data points from 3 different scenarios in the following table:

Scenario (actual)	Object (detected)	Confidence	Detected dist	Actual dist
Keyboard moving closer	Keyboard	0.62	99 cm	100 cm
Keyboard moving closer	Keyboard	0.55	79 cm	80 cm
Keyboard moving closer	Keyboard	0.70	57 cm	60 cm
Keyboard moving closer	Keyboard	0.59	38 cm	40 cm
Keyboard moving closer	Keyboard	0.72	19 cm	20 cm
Person sitting	Person	0.74	100 cm	100 cm
Person sitting	Person	0.69	77 cm	80 cm
Person sitting	Person	0.71	60 cm	60 cm
Person sitting	Person	0.64	46 cm	40 cm
Person sitting	Person	0.61	24 cm	20 cm
Pan	Chair	0.60	100 cm	100 cm
Pan	Chair	0.64	85 cm	80 cm
Pan	Chair	0.57	65 cm	60 cm
Pan	Chair / Toothbrush	0.55	45 cm	40 cm
Pan	Suitcase	0.55	26 cm	20 cm

Performance / results discussion

The object detection test included three objects: a keyboard, a person (sitting), and a cooking pan. All testing was performed with the ultrasonic sensor sitting on top of a plastic box, which was situated on the floor. The Raspberry Pi 4 camera sat right behind the ultrasonic sensor at about the same level as the ultrasonic sensor.

The object detection performance for a keyboard and a person (sitting) was quite good and was steady around more than 60% most of the time. A black cooking pan was also used, but the performance was a little inconsistent. While the distant metric was accurate, the object detection misclassified the pan in all instances. It is possible that the pan was not included in the vocabulary of the pre-trained machine learning model, so it just gave its best guess.

Before using a pan, we also tried to use a person standing, but the ultrasonic sensor gave a lot of false readings for the distance. Probably because the waves passed through the legs.

Team Member Contributions

Berkay
Worked on installing tensorflow on Raspberry Pi, and wrote the code to interact with the Pi camera to convert each image into a numpy array.

Max
Worked on TFLite object detection, setup Pi environment

Ben
Worked on IP to CAN bidirectional communication and CAN bus between Arduinos for ultrasonic sensors.

Daniel
Preparing the pi, object detection

Alpri
Designed Head Unit and developed a method of communicating distance and image data using Web Sockets.

Github Repo: https://github.com/maxweis/fa20-cs498it-lab1