IOT Lab 1: IoT Devices

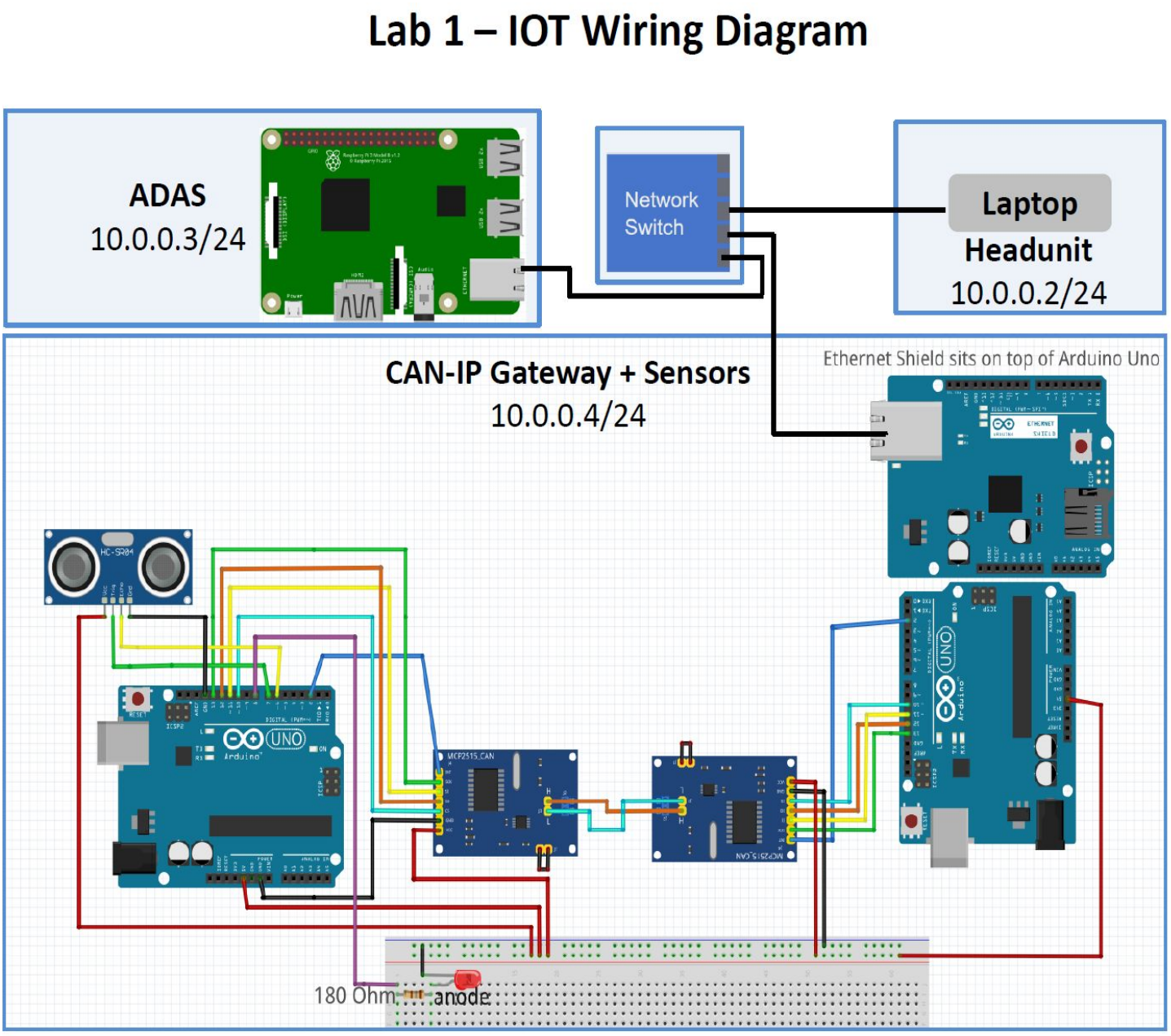
Team: iot-westcoast-group

Team members:

- Matt Pendergraft, mjp10
- Rosie Yu, weiy6
- Isaiah Leonard, ihl3
- Giri Reddy, girir2
- Jie Zheng, jzheng5

Giri Reddy netID: girir2	Isaiah Leonard netID: ihl3	Jie Zheng netID: jzheng5	Matt Pendergraft netID: mjp10	Rosie Yu netID: weiy6

Device Wiring



Video

The demonstration video can be viewed here.

Design Considerations

ADAS (Raspberry Pi programming)

The ADAS provides the main computation and logic controlling the "vehicle's" object detection and avoidance system. Our ADAS is housed in a Raspberry Pi, and an outline of its main processing loop, along with design considerations for applicable portions, is as follows:

1) Capture image via PiCamera
    a) The two main considerations here are how to capture the images continuously and what resolution to use. For the former, we found a significant performance improvement after profiling showed that we were spending much more time simply capturing images than seemed reasonable. By following the official PiCamera guide for rapid capture and keeping the camera running at 30 FPS and then capturing stills from the video feed, we managed to reduce the image capture time significantly. As for the image resolution, we knew that we wanted reduced resolution images in order to improve detection speed, and after some experimentation found that 640x480 provided a good balance between a relatively high (>1FPS) throughput and reasonable image quality (for reference, we achieved 1.4 FPS at 640x480 and 1.6 at 320x240).
2) Pass image into tensorflow session for inference
    a) We used the ssdlite_mobilenet_v2_coco_2018_05_09 model and adhered closely to the Tensorflow Object Detection Tutorial for inference. This model was chosen because it is widely regarded as the best for relatively high performance on computation-constrained machines. Since the lab assignment stated that quantized models no longer work without TFlite, and therefore could not run on the Raspberry Pi 3 running the ADAS, we did not use a quantized model.
3) Use tensorflow results to determine detections scores
4) Get distance via a simple request to the CAN
    a) We could potentially realize some performance improvement by moving this out of the main loop, but profiling revealed it to be a minimal part of the time required for each iteration.
5) Decide how to set brake
    a) Here we used a static threshold for both object distance and the detection score for when to signal the brake because those have seemed to work well in our testing. Obviously we don't have a real vehicle to test this on, but if we did, then

we could easily adjust this behavior accordingly. For example, we could make the score threshold based on the object distance, or store the last distance measurement or two (using variables on our TensorCamera class) to determine whether and how fast an object was approaching. In addition, we used a relatively low detection score threshold (30%) because (as discussed in the framerate vs detection accuracy question) this application seems like one in which a false negative is much worse than a false positive

6) Convert image to png
7) Send png and brake/distance to dashboard

The Raspberry Pi runs `app.py` as its main entry point. This initialized a websocket on port 9000. When a connection is made the Tensorflow, camera and sensor module are initialized and the main loop above starts.

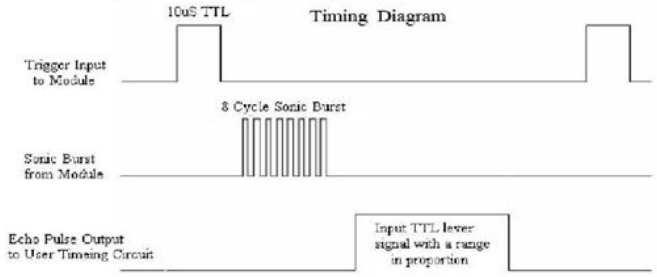
Microcontroller (Arduino programming)

We use Arduino to control the ultrasonic sensor and the LED light. It receives distance data from the ultrasonic sensor and then sends it to the CAN to IP gateway when ADAS requests it. The Arduino also listens to a message passed on by the CAN-to-IP gateway about whether ADAS decides to brake to control the LED light. It communicates directly with CAN-to-IP gateway and both the distance data and brake light status are transmitted through the CAN bus. The ultrasonic sensor works as below.

1. According to the ultrasonic HS-RC04 datasheet, the trigger signal sent out from arduino to the ultrasonic sensor should be a 10 us pulse. Then the HS-RC04 will send out an 8-cycle burst of ultrasound at 40kHz, lasting 200 us (8 x T = 8 x 1/f = 8 x 1/(40 kHz) = 200 us).
2. According to the ultrasonic HS-RC04 datasheet, A minimum 60ms measurement cycle is suggested to prevent receiving multi-reflection echos during the next trigger event. So we add the delay of 60ms per measurement cycle.

Timing diagram

The Timing diagram is shown below. You only need to supply a short 10uS pulse to the trigger input to start the ranging, and then the module will send out an 8 cycle burst of ultrasound at 40 kHz and raise its echo. The Echo is a distance object that is pulse width and the range in proportion .You can calculate the range through the time interval between sending trigger signal and receiving echo signal. Formula: uS / 58 = centimeters or uS / 148 =inch; or: the range = high level time * velocity (340M/S) / 2; we suggest to use over 60ms measurement cycle, in order to prevent trigger signal to the echo signal.



HCSR04.pdf

3. The pulseIn() function of the arduino is used to measure the pulse duration from the echo pin in microsecond, i.e., us. Factoring in that the speed of sound is 343 m/s, i.e., 0.0343 cm/us and that the pulse duration is for a round trip, the distance can be derived as:

   L =v *t = 0.0 343 cm/us * pulseIn() us /2 = 0.01715 * pulseIn() cm

4. According to the HS-RC04 datasheet, the measurement range of the ultrasonic is **2cm-4m** with a measurement angle of **15** degrees. So for real world application, we could build a mechanical stop or case to recess the ultrasonic sensor by 2cm to eliminate errors generated by objects closer than 2cm. We could also add more ultrasonic sensors for sensing objects from different angles in the future.

Electric Parameter

Working Voltage	DC 5 V
Working Current	15mA
Working Frequency	40Hz
Max Range	4m
Min Range	2cm
MeasuringAngle	15 degree
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	Input TTL lever signal and the range in proportion
Dimension	45*20*15mm

HCSR04.pdf

For the LED circuit, according to the Arduino Uno R3 documentation, each digital pin outputs **5V** with a maximum current output of 40mA, while a red LED typically **drops** around **1.7 to 2.0** volts and 20 mA is common for many small LEDs.

(5 V -1.7 V) / 20 mA= 165 Ohm

So we just choose a standard 5% 180 Ohm axial resistor as the current limiting resistor.

> Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(),digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.

CAN to IP gateway

The CAN-to-IP gateway is an Arduino Uno board connected with an Ethernet shield, which can communicate with the microcontroller through CAN bus and also talk to ADAS through Ethernet. It is a bridge translating between the CAN message and IP packets.
Our design considerations are listed below.

1. TCP protocol is used for Ethernet communication between Raspberry Pi and Arduino which works as the IP-CAN Gateway, because with relatively low overhead on ethernet it provides reliability by its nature which is one of the most important qualities any automobile should have. The TCP vs UDP decision will be discussed further below The CAN-to-IP gateway serves as the server and Pi serves as the client.
2. We structure both IP and CAN messages as the same for simplicity and consistency. There are 3 types of messages:
    a. distance data pull requests from Pi to gateway Aruidno,
    b. distance values from microcontroller Arduino through gateway Arduino to Pi,
    c. and LED status from Pi through gateway Arduino to microcontroller Arduino.

    They are all 3 bytes. The first byte (an unsigned char) indicates whether it is about distance(“1”) or LED (“0”). It is followed by 2 bytes of data (an unsigned short). For distance requests, the two bytes are unused. For distance value messages, the unsigned short is the distance value in cm. For LED messages, “0” means “OFF”, “1” means “ON”.
3. The HC-SR04 has a ranging accuracy of 3mm and a max measurement range of 4m. 2 bytes are enough for the distance data (4m/3mm~1334).

Dashboard

The dashboard is fairly simple. It’s an HTML page on a computer at 10.0.0.2 that establishes a websocket connection to 10.0.0.3 (the raspberry pi). It updates some HTML with the messages

it receives over a websocket. That data consists of the detection image (with classification boxes) encoded as a png and a message encoded as json that contains brake and distance.

A websocket was selected to remove the overhead of polling an endpoint from the dashboard. An initial handshake is needed with the ADAS component and then the ADAS is able to send data, as it has, to the client. The functions that update the dashboard are triggered by the message type (an ArrayBuffer object or a string to encode in a map) and are registered against the onmessage trigger of the received data.

This would *not* scale to multiple dashboards as it is written now as websocket is TCP-based and doesn't have a native broadcast method. This could be adjusted by changing the design to send a message to all connected clients on update but would require iteratively sending such a message and tracking client connections (which is a common design).

Further improvements could be made by rendering more and more on the dashboard. However, in profiling ADAS performance the two biggest improvements (sending binary directly without a BGR2RGB + PNG encode & drawing detection labels by sending the bounding box + label) would have yielded less than a small gain gain due to the vast majority of the time budget being spent by image inference (see hardware acceleration answer for additional detail on profiling).

Performance

We were able to achieve framerates (as measured at the dashboard) of 0.90 frames per second with an image resolution of 640x480 and 1.1 frames per second with a resolution of 320x240 on the Raspberry Pi 3. On the Raspberry Pi 4 we observed 1.4 FPS and 1.6 FPS at each respective resolution.

The detection table below is done using a Raspberry Pi 4 with resolution 640x480.
Threshold = 0.3, i.e. when confidence score <= 0.30, the detection is negative and when confidence score > 0.3, the detection is positive.

Scenario 1: People moving towards the camera - Wei

	Video Frame Rate	Confidence Score	Detection Accuracy
1	1.43	0.81	1
2	1.43	0.76	1
3	1.43	0.77	1
4	1.43	0.92	1

5	1.46	0.92	1
Average	1.44		100%

Scenario 2: People moving away the camera

	Video Frame Rate	Confidence Score	Detection Accuracy
1	1.56	0.85	1
2	1.42	0.49	1
3	1.43	0.86	1
4	1.47	0.39	1
5	1.42	0.83	1
Average	1.46		100%

Scenario 3: People moving side to side

	Video Frame Rate	Confidence Score	Detection Accuracy
1	1.55	0.87	1
2	1.56	0.84	1
3	1.47	0.87	1
4	1.47	0.72	1
5	1.43	0.60	1
Average	1.50		100%

Scenario 4: People jumping

	Video Frame Rate	Confidence Score	Detection Accuracy
1	1.43	0.60	1

2	1.46	0.86	1
3	1.47	0.52	1
4	1.43	0.66	1
5	1.41	0.60	1
Average	1.44		100%

Answers

Question 1: Quantized models

Why are quantized models better for resource-constrained devices?

Answer:
Quantized models can be smaller and faster in exchange for trading off some of their classification ability. If a device is resource-constrained where it can't talk to the network or needs to make decisions faster than introducing network overhead would allow, limited by component cost, etc. quantizing a model is one possible design tradeoff. As discussed below in the frame-rate vs. detection accuracy question, this application is one in which a lower detection accuracy (and in particular classification accuracy) is more than acceptable in exchange for faster throughput.

A quantized model projects one precision such as float32 to another, generally smaller, such as an int8. This saves model space and rounds of the edges decreasing the accuracy. It's also possible there are hardware acceleration gains by the resource-constrained device through optimized instructions such as faster integer instructions over floats.

Note that since the lab assignment stated that quantized models no longer work without TFlite, and therefore could not run on the Raspberry Pi 3 running the ADAS, so unfortunately we did not use a quantized model for this application.

Question 2: Hardware acceleration

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

Yes, hardware acceleration would help in image processing. PiCamera takes advantage of the camera and the GPU and uses the image signal processor to speed up several operations (described here). Image processing happens on the GPU after the RTOS on the GPU gets the

image in memory and is then passed off to an encoding block if appropriate for that format. We made use of the capture method on the video port with “rgb”. PiCamera selects an encoder which in this case is a raw format that does not pass through a hardware-accelerated encoder.

There is hardware for a jpeg encoding. If we used that instead of a png it might be possible to speed up our encode but it would require marshalling our working format (tensorflow uses the numpy array) to something that could be consumed by the encoder and passed in leveraging MMAL or directly.

Improvements in this would speed up our call to capture. In the crude profiling we performed we saw roughly the following on the ADAS (does not include network overhead):

Image Capture 0.07 seconds
Image Inference 0.4 seconds
Encoding + Other 0.1 seconds

Which could be reduced.

Another place where hardware acceleration is frequently used for image processing is in the machine learning libraries around them. However, the gains here are mostly in training neural nets or in performing inference with very large models. In our case, we are using a pre-trained model and the model we chose is quite small. In addition, for our application, the Raspberry Pi’s GPU does not support CUDA, which TF 1 requires. So while it is possible that hardware acceleration would improve the performance of object detection in some cases, for our specific application it does not seem promising.

Question 3: Multithreading

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

Answer:
In general, multithreading can improve performance over a single threaded application when the application in question has operations that could easily be parallelized or where some ‘heavy’ operations are not CPU bound (and therefore need not block computation while they are waiting. A classic example is when an application is largely split between I/O and computation, and using multithreading can ensure that the I/O part of the application does not block the computation-bound part of the program simply waiting on other resources. However there are limits of multiple threads, mainly resource contention like L2 / L3 caches, TLB evictions, pre-emptive locks and I/O bound operations - contention in any of these subsystems will cause the individual threads to wait on resources being available. Throwing more threads and cores is not an optimal solution, as proven by Amdal’s law.

In the case of our program, we would not see significant gains in performance unless we were able to make the tensorflow inference faster by itself because that inference so completely dominates the time for each processing cycle (outlined in the ADAS section). Indeed, when we profiled our application, nearly 80% of each loop was spent in image inference, with most of the remainder spent encoding and decoding the image (which are also compute-bound). Moreover, given Python's known issues with the GIL and multithreading, it seems unlikely that significant improvements could be made to this application with multithreading, though it could perhaps help slightly if we were to remove the polling from the raspberry pi (socket.recv) for distance and use shared state.

Question 4: Trade-off between frame rate and detection accuracy

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

Answer:
Making a decision like this is highly dependent on the application. In this case, for avoiding obstacles, it seems like our priority should be a high frame rate over optimal detection accuracy (within reason, of course) because of how quickly cars move and how bad it would be to hit a person (or other object). At 60 mph, a car is moving at 88 feet per second; at that speed, taking extra time to increase detection accuracy, and in particular for detection classification accuracy, does not seem prudent. Indeed, taking an extra two seconds to process an image means that the car would have travelled nearly 200 feet before it has the object detection results. At that point, the detection results are likely not even relevant any more. In addition, for this use case, we know that we generally want to be pretty conservative in terms of avoiding false negatives (i.e. not detecting an obstacle and stopping) because that is likely much worse than a false positive (i.e. detecting something that turns out not to be there and therefore beginning to brake when it isn't actually necessary), so we can have a slightly lower detection accuracy and simply brake even when the detection score is relatively low (in our current configuration, we have that threshold score set to 30%). As a result, we felt that we wanted to encourage as high a frame rate as possible while maintaining reasonable detection accuracy and image resolution (for the headunit to display) instead of increasing detection accuracy at the expense of framerate.

We manually tuned object distance and detection percentages to use less than 100 cm and a detection with score at least 30% in the frame. Further tuning of these parameters could potentially find a better fit for a real world scenario, but would require real world data in order to do so. For example, either the distance or detection score thresholds could be adjusted based on the other, or we could calculate 'approaching velocity' instead of using a static 100 cm limit.

Question 5: TCP or UDP

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

Answer:
We have chosen TCP for a few reasons, most notably:

1. Foremost *we* wanted to focus on writing an application instead of worrying about error checks and recovery from errors due to hardware issues (Ethernet port getting wedged) or software issues (application failure or networking stack errors). We realized that only applications need to check for errors and have recovery mechanisms in place to recover from communication infrastructure failures. While a production-level car control system might merit writing customized UDP logic to be more fault-tolerant, for a prototype system like this, the increased 'usability' of TCP is a large factor (similarly, Toyota would likely develop an ASIC or other custom hardware instead of a Raspberry Pi and breadboard).
2. Since TCP provides guarantees over UDP to recover from frame loss or peer resets (hardware or application failure) we were willing to live with the extra overhead with TCP. We wanted to leverage TCPs inbuilt reliability, which is critical for a car safety system. Moreover, while UDP is frequently used for real-time, streaming applications (such as video streams) in which missing a frame isn't a big deal, there are scenarios for car control systems where out of order or dropped packets can make a huge difference. For example, if our braking algorithm considered whether an object was approaching the sensor, instead of a static distance, then obviously packet order and delivery becomes crucially important. Perhaps the brake signal alone could be changed to use UDP to increase throughput, but the application overhead given the relatively small distance and quick connection through TCP is minimal.
3. Third, since we are using ethernet, the overhead in TCP is relatively small which barely reduces the responsiveness as the communication is a local area network where the nodes are one hop away from each other. Since the TCP connection is such a tiny part of the application timing anyway, it seems that the overhead of writing custom UDP logic for this lab did not merit the potential (very small) speed increase.
4. We also considered the overhead of (3-way) TCP connection setup and teardown, which is negligible given the other overheads in application complexity (error recovery and sequencing of frames).

In short, UDP does obviously provide some advantages in terms of having a smaller packet overhead and potentially faster throughput, for this application those potential advantages did not outweigh the reliability and usability of TCP.

Question 6: Full-duplex communication

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

Answer:
No. Since we are limited by the CAN bus which is half-duplex medium it is not possible to have a full-duplex communication path between CAN ECU and Pi that is the first limitation. Also, Arduino runs as a single thread application, which can maintain one active connection. Even if

using TCP or UDP, Arduino single thread can only context switch between a read or a write even though the underlying Ethernet medium supports full duplex.

Question 7: Dashboard update time interval

Can you make the time interval smaller? Also, you should mention how sending data over the network influences the Pi's performance in your report.

Yes, but not significantly we are very close to being constrained by the image inferencing. In profile our main loop we were seeing more than 90% of our update budget in inferencing.

The most significant improvements would come through improving inferencing speed. One possible solution to decrease the speed of the update interval is offload everything from the Pi to the dashboard. Since the dashboard is running on a significantly more powerful computer tensorflow-js for the classification would likely run faster than the 500 - 800ms we see (depending on Pi model). This speed difference wouldn't be observed in a real-world scenario and would ignore any robustness designed into the ADAS.

Another would be to use frame differencing to segment just the changing image to make an inference about future location. This could allow

Finally we could use a transmission medium with less overhead. This wouldn't speed up inference but would cut down on network latency, browser rendering speed, and encode/decode.

The network transmission overhead to the dashboard is minimal. It's handed off to the kernel almost immediately and the functions don't wait for a response message in the main loop.The serialization and encoding of the image prior to transmission has some minimal overhead (never more than 100 ms in our profiling). The distance sensor polling could be much shorter. We implemented it as a request response, but in a life safety system a sensor that's constantly updating a shared state (such memory mapping the sensor to the ADAS directly data) may be an appropriate tradeoff from a more extensible network.

Question 8: Interconnect over Ethernet network

Since we have three Ethernet Devices (the Pi, the CAN to IP gateway, and the Head Unit), how should we interconnect them in the Ethernet network?

Answer:
We have interconnected the Pi, CAN to IP gateway and Head Unit using an ethernet switch.All are directly connected and on the same broadcast domain.

We decided to use the 10.0.0.0/24 subnet and assigned the gateway 10.0.0.4, the Raspberry Pi 10.0.0.3, and the dashboard 10.0.0.2. In this way there is no need to worry about routing to other subnets and the switch maintains the MAC address to IP mapping. 10.0.0.0/8 is a reserved range so we took a smaller slice. We could have gone smaller, aligned to an octet for ease.

Contributions

Note: Rosie Yu (weiy6) dropped the class as we were finalizing the report, but made significant contributions in the organization and content

	Giri Reddy netID: girir2	Isaiah Leonard netID: ihl3	Jie Zheng netID: jzheng5	Matt Pendergraft netID: mjp10	Rosie Yu netID: weiy6
Organizing	5%	5%	5%	70%	15%
System					
ADAS	0%	80%	0%	20%	0%
Micro	0%	0%	70%	0%	30%
Gateway	0%	0%	60%	0%	40%
Dashboard	0%	0%	0%	100%	0%
Integration	70%	0%	0%	30%	0%
Presentation					
Clean code and document	20%	20%	20%	20%	20%
Report	20%	20%	20%	20%	20%
Video	5%	5%	5%	80%	5%

Code

Github repo: https://github-dev.cs.illinois.edu/iot-westcoast-group/fa20-cs498it-lab1/

References and Citations

[1] PiCamera Library (performance documentation in particular
[2] Python Websockets Library
[3] asgiref for sync_to_async decorator
[4] DHCPD Configuration
[5] MDN for CSS grid and websocket events
[6] Camera, GPU, CPU connection
[7] Model Quantization
[8] Tensorflow Object Detection Tutorial
[9] Tensorflow Model Used
[10] Pi setup (i.e. which libraries and packages were needed to get the PiCamera and Tensorflow running) and setting up a debugging pipeline for processed images before the headboard was integrated