CAR-BAC

Car Approaching Rearview Bicycle Alert Camera

Steve Milligan/Neil Tender/Robert Kraig/Amy Dexter

Eyes on Edge: tinyML Vision Challenge Project

Table of Contents

1 Introduction

The tinyML Foundation in conjunction with Hackster.io has launched a contest for developers worldwide, challenging them to build advanced applications with low-power machine learning inferencing and computer vision for edge devices. Tiny machine learning (tinyML) is broadly defined as a fast-growing field of machine learning technologies including hardware, algorithms and software capable of performing on-device sensor data analytics at extremely low power, targeting battery-operated devices. This contrasts with typical machine learning solutions that can consume large amounts of resources for data preparation, model training and inference.

Our team for this competition was composed of members of the Bethesda Artificial Intelligence Meetup who expressed an interest in visual recognition systems. We established as our goal the development of a working solution that could be demonstrated and showed some real value or potential for use. It was also hoped that our solution would have a positive impact on society.

We employed an agile way of working that allowed us to rapidly test, discard or postpone, if necessary, various avenues of investigation related to hardware, software, datasets and models. This approach allowed us to focus on building a working model and to avoid excessive focus on the optimization of any individual component.

TinyML Vision Challenge (hackster competition):

https://www.hackster.io/contests/tinyml-vision#challengeNav

2 Project description

Collisions between motor vehicles and bicycles are occurring with increasing frequency as urban spaces become more crowded and more pleasure trips are taken to rural areas post pandemic. Machine Learning enhanced tools may be able to increase cyclists' awareness of approaching vehicles so evasive action can be taken if necessary.

For our project, we built a device that automatically detects vehicles approaching the bicycle from the rear and raises an audible alert to the rider to make him/her aware of the possible danger. This situation is a common source of anxiety for a rider as a fast approaching car that wishes to pass the slower moving bicycle can be unpredictable. The cyclist would like to always know when there is a car in the vicinity so there is time to safely move out of the way. The **Car Approaching Rearview Bicycle Alert Camera** is an apparatus that mounts underneath the saddle (seat) of the bicycle and uses a rear-facing camera to look for the approaching cars at all times. The name of the project, CAR-BAC, is inspired by the tradition among cyclists to warn one another of rear approaching cars by yelling out the phrase “Car back!!!”.

See the links below for a video demonstration and the project GitHub repository with source code, data, photos, and videos.

video – link to Youtube: https://youtu.be/V4aLnohf_z0

GitHub – link to github: https://github.com/Bethesda-AI-Projects-Lab/bicycle_vision

We did not start this project with a specific solution in mind. Instead, the team discussed a wide variety of potential products with the only firm requirement being the use of visual recognition in some manner. The team also wanted to create a product that would benefit society in some way. Experiments began with attempting to implement existing TensorFlow examples on a variety of hardware platforms that included Raspberry Pi, NVIDIA Jetson, Google Coral TPU, open MV Cam, Xilinx FPGA, and a Qualcomm dragonboard (Android phones). We also experimented with microcontrollers like the Raspberry Pi Pico, Arduino Sense BLE and Sony Spresence. In the end, we created a proof-of-concept system that utilized a Raspberry Pi 4 single board computer, Google Coral USB TPU, 5MP camera, buzzer and battery pack and successfully demonstrated the soundness of our idea. This paper documents our efforts to create a low power, machine learning tool that will enhance bicycling safety.

3 Hardware

This section describes the hardware design and components that comprise the CAR-BAC system.

Figure 1 below shows the complete system shown fully mounted on the bicycle. The camera is mounted securely onto the bicycle saddle using an off-the-shelf bicycle mount that was meant for use with a GoPro camera but was repurposed for this project. All of the other components of the system are conveniently stashed away in a saddle bag with only the camera cable protruding out of the bag connecting the camera. The rider of the bicycle does not really notice that CAR-BAC is present because it is lightweight and mounted out of the way -- that is, until it beeps to indicate an approaching car.

Figure 1 Photos of CAR-BAC Mounted on Bicycle

Figure 2 below shows the complete CAR-BAC system removed from the bicycle, showing all of the major components connected together. The following subsections describe the major components and functions and components.

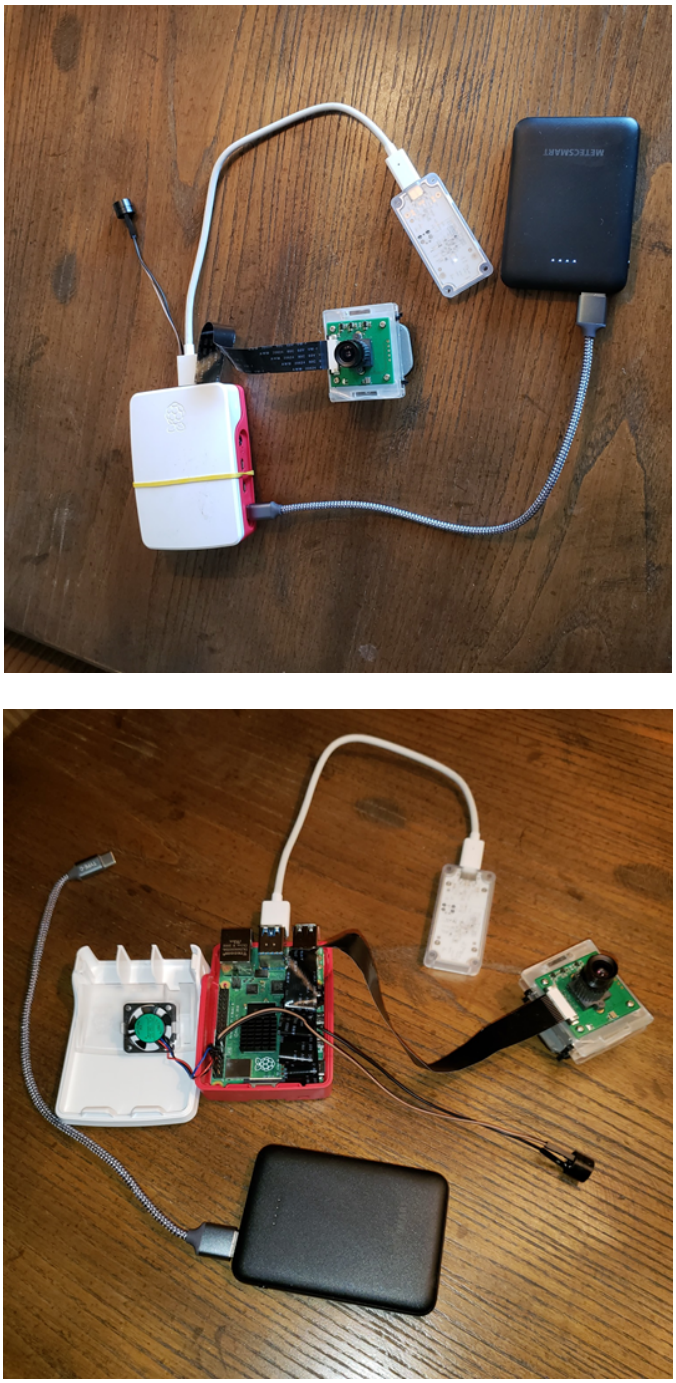


Figure 2 Photos of CAR-BAC Assembled Showing all Components

3.1 Components

3.1.1 Raspberry Pi 4 - Master Controller

The Raspberry Pi 4 functions as the master controller for CAR-BAC. A Python script that implements the algorithms for detection and alerting of the approaching vehicles is executed on the ARM processor within the Pi. Figure 3 depicts the Pi mounted in the official Pi foundation case.

3.1.2 Arducam for Raspberry Pi - Camera

The camera performs the essential function of capturing live video facing the rear of the bicycle. The Arducam model that we are using has the ability to adjust the lens to focus at the desired distance from the camera. The camera is connected to the Pi via an extra long cable so that it may reach from the saddle-mounted camera into the saddle bag and Pi. Figure 4 shows the Arducam attached to GoPro Mount.

3.1.3 Google Coral TPU, USB - CNN Accelerator

The TPU performs the machine learning inference of the object detection. It contains a special microchip optimized for accelerating convolutional neural network inference, allowing greater speed and lower power than if it were to run on the Pi. The TPU connects to the Pi via a USB port. It is shown below in Figure 5

3.1.4 Battery Pack -- Energy Storage

The battery pack is the type that is commonly used to charge a mobile phone on the go but works quite well for this application. It is small and lightweight and stores enough power to run the CAR-BAC for hours (see power consumption section below). We selected a model that can provide up to 2A output at 5V to ensure that the peak current of the Pi and Coral can be met. The battery connects to the Pi over a USB-to-USB-C type cable. See Figure 6.

3.1.5 Buzzer, Piezo type -- Audible Alert

The simple buzzer runs off of 5V and generates a surprising loud piercing sound for its small size. The buzzer connects to the Pi via a GPIO port. Figure 7 shows the buzzer connected to the Raspberry Pi via a jumper cable attached to the standard GPIO header, along with the power connection for the Pi case fan.

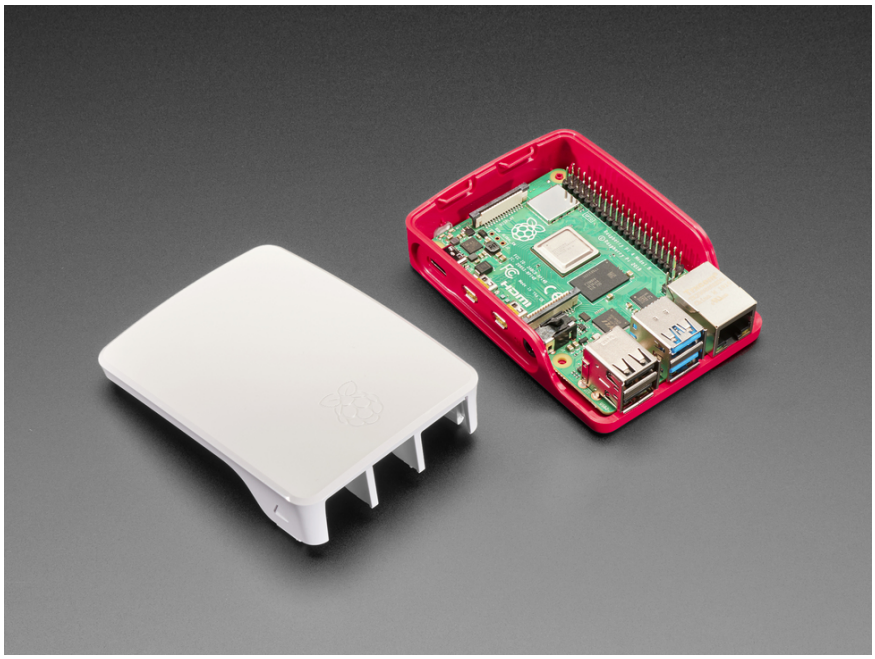


Figure 3 Raspberry Pi 4 in Case

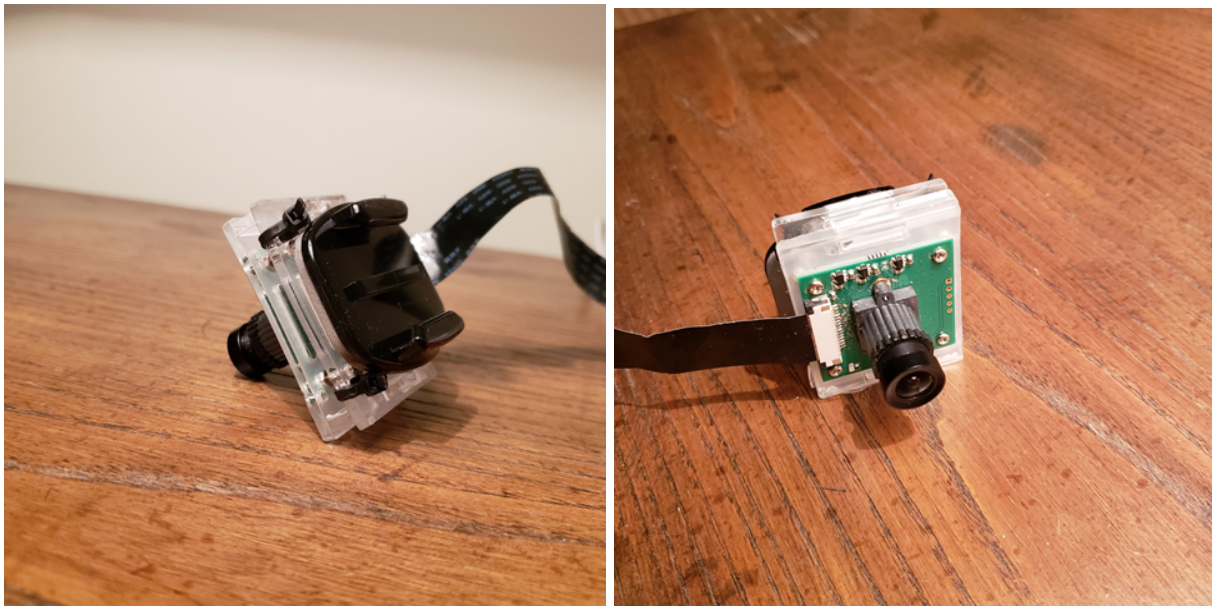


Figure 4 Camera Attached to GoPro Mount

Figure 5 Google Coral Edge TPU

Figure 6 Battery Pack

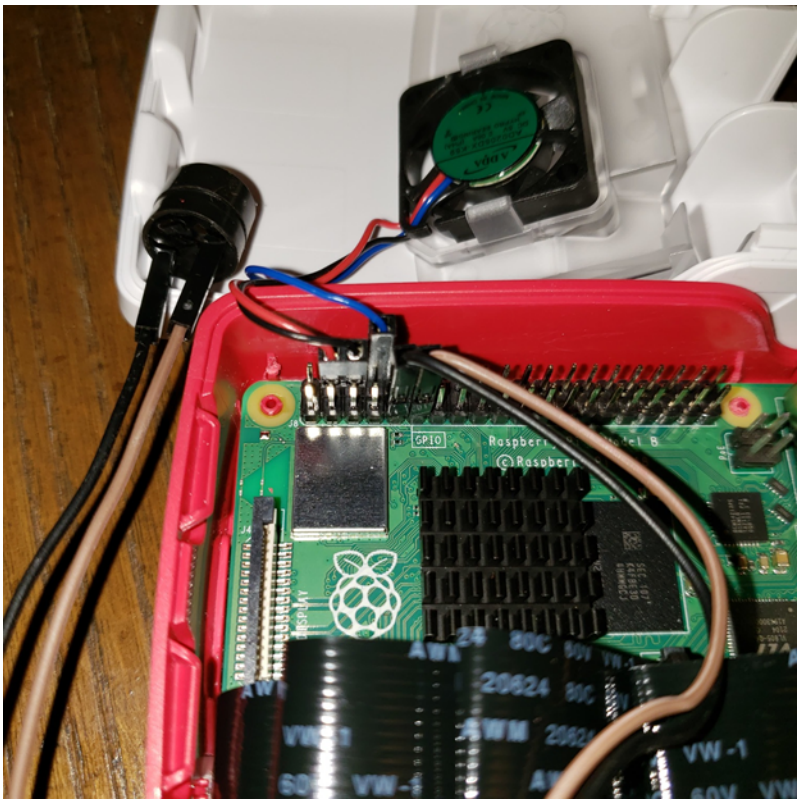
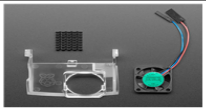


Figure 7 Photo of GPIO Connector for Buzzer and Fan

3.2 Bill of Material

See Table 1 below for the complete itemized bill of materials, including model number, description, cost, quantity, and in some cases supplier.

Table 1 Bill of Materials

Component	Image	Part Number	Quantity	Manufacturer	Price ($)	Notes
ParaPace Bicycle Saddle Rail Camera Mount Bike Seat Mount for GoPro		PP178blk	1	Parapace	$13.99	Purchased from Amazon.com
GoPro Grab Bag (GoPro Official Mount)		AGBAG-002	1	GoPro	$19.00	
Raspberry Pi 4 Model B - with 2GB, or 4GB RAM		4297	1	Raspberry Pi Www.raspberrypi.org/	$35 -55	
Coral USB Accelerator		G950-06809-01	1	https://coral.ai/products/accelerator	$59.99	
Arducam for Raspberry Pi Camera Module with Case, 5MP 1080P		B0031	1	https://www.arducam.com/product/b0033c-arducam-5mp-1080p-camera-module-with-case-for-raspberry-pi-3-3-b-and-more/	$21.99	
Buzzer 5V		1536	1	https://www.adafruit.com/product/1536	$0.95	
Battery Pack		5K01	1	Metecsmart	$19.99	Purchased from Amazon.com
32GB micro–SD Card		SDSQXAF-032G-GN6	1	Sandisk	$10.29	Purchased from Amazon.com
Pi Camera cable, 2 feet		CB013	1	Arducam	$4.99	Purchased from Amazon.com
USB C Cable Short, 0.8ft		4348686024	1	JXMOX	$8.99 for pack of 6	Purchased from Amazon.com
Female to Female Jumper Wires		4330127279	2	GenBasic	$5.99 for 80pcs	Purchased from Amazon.com
Official Raspberry Pi Foundation Raspberry Pi 4 Case - Red		4301		Raspberry Pi Www.raspberrypi.org/	$6.00	
Official Raspberry Pi 4 Case Fan and Heatsink		SC0448		Raspberry Pi Www.raspberrypi.org/	$5.00	

3.3 Power Consumption

The power consumption and battery life of CAR-BAC is naturally very important because a cyclist may be out riding for several hours, so the system needs to be able to operate for at least as long as the ride duration. Also, this is a TinyML project and one of the goals is to design a system that fits in that category, a very low power machine vision application.

To assess the performance of CAR-BAC, we measured the power consumption and then estimated the battery life. The power was measured using a Kill-A-Watt power measuring device while powering the Raspberry Pi from the AC power adapter.

The measurements were as follows:

3.6W idle (not running Python script)

5.6W running live

To estimate the battery life, we calculate the run-time based on the capacity of the battery. The Metecsmart Mini Portable Charger is rated at 5000mAh (5V/2A). The Raspberry Pi is powered off of a 5V rail and thus consumes approximately 1.12A on average. The 2A rating should provide adequate margin. The estimated runtime is:

5000mAh / (1.12A * 1000) ~= 4.5 hours

This number may not be very accurate as there are some inefficiencies and other factors we didn't consider, e.g. power may fluctuate, battery capacity may vary over different operating voltages/temperatures. However, 4.5 hours seems like an excellent starting point for the duration of a bicycle ride. If we can later improve the power efficiency of CAR-BAC, the run-time will go up.

3.4 Cost / Availability

The total cost of the components used to build our prototype was $218.96 before taxes. All components were readily available from online suppliers (Amazon, Adafruit, Mouser, etc.) and we believe there are substantial opportunities to lower the platform cost in the future. Examples include using a Raspberry Pi Zero instead of the Pi 4 which would save almost $50, using the Google Coral Dev Board Mini instead of the Pi 4/USB TPU combination ($16 savings) and, using cheaper components to mount the device on the bicycle. We believe a commercial version of this product could ultimately be assembled for less than $100 per unit.

3.5 Size / Weight

With all of the components stacked together (Raspberry Pi in case, battery, and TPU), the approximate size of the hardware components that are stowed away in the saddle bag (i.e. everything but camera) is 2.5in (W) x 4in (H) x 2in (D). The size of the camera on the GoPro mounting clip is approximately 2in (W) x 1 5/8in (H) x 2 in (D).

The approximate weight of all the hardware components combined is 12.66 oz = 0.79 lbs.

4 Software and Algorithms

4.1 Software

The software is implemented as a simple Python script that executes on the Raspberry Pi. The script utilizes open-source libraries for most of the functions: OpenCV, NumPY, PyCoral, and FilterPy.

After initializing the hardware, the script enters a loop in which it processes one frame per iteration. The sequence is fairly straightforward and consists of the following steps:

1. Fetch a frame from the camera
2. Format and resize the frame to 320x320
3. Invoke the Coral TPU to perform object detection
4. Filter out all objects that are not in the classes of interest, i.e. cars/trucks
5. Perform object tracking
6. Use the tracks to perform approaching vehicle detection
7. If an approaching vehicle is detected, turn on the buzzer; otherwise, turn it off

This loop is repeated indefinitely.

4.2 Algorithm Overview

CAR-BAC's algorithm can be understood as a three-step process. First it must detect vehicles in the field of view of each video frame. Second, it must combine the image detections across multiple frames into tracks that characterizes each vehicle's motion. Third, for each track, it must determine whether the track is approaching the camera and is thus a potential threat.

Our solution is a direct implementation of this three-step concept. For vehicle detection, our prototype employs SSDLite MobileDet (3), an object detection model available precompiled from Google Coral (4). To aggregate the detections into tracks, we apply a tracking algorithm called SORT (5). The final classification step is performed in a straightforward way. We fit a linear regression to the time series of bounding box horizontal and vertical sizes found in each vehicle track. The vehicle is classified as approaching if the slope of the linear regression fit indicates that the vehicle track is occupying an increasing number of pixels from one detection to the next.

4.2.1 Algorithm Implementation Details

The object detector implements an architecture known as SSDLite. The architectural basis for SSDLite was first developed and published in 2015 as Single-Shot Detector (SSD) (6). At the time, SSD greatly reduced training and inference speeds. Existing high-performance methods required two neural networks, one to generate region proposals and a separate network to generate detections given the proposals. SSD condensed this process into a single neural network, seeded with a hardcoded set of bounding boxes that would be tweaked by the network while processing each detection. The single-network SSD approach was shown to offer comparable (albeit not matching) performance vs Faster R-CNN (7), as measured by mean average precision (mAP) (8) using COCO (9), the canonical dataset for object detection research. SSDLite (10) replaces some of SSD's standard convolutional layers with depthwise convolutions, providing further reduction in computation and memory requirements, enabling smaller and smaller applications on low-power devices. Our version of

SSDLite, pre-compiled by Google Coral for the Edge TPU, uses quantization-aware training. Thus the inference computations are performed entirely in 8-bit fixed-point arithmetic.

Each object detection returned by the TensorFlow Lite model consists of six output numbers. The first two outputs, id and score, are used to filter detections. The last four outputs, coordinates defining the left, right, top, and bottom edges of bounding boxes for each detection, are used to construct tracks and to determine if the detected object is approaching.

The id indicates which COCO class the model considers to be the best match. Trained for the entire COCO object set, the Google Coral model is capable of recognizing 80 distinct class labels. However, most of these labels are objects that are not relevant to the use case for our product. We filter the model detections, and we retain only those detections from four classes: car, motorcycle, bus, and truck. A fifth class (bicycle) was retained for some testing but it is turned off for our prototype demonstration.

The score returned by SSDLite is a confidence value between zero and one. Coral's implementation permits the end user to optionally filter detections so that it will report only those that exceed a threshold confidence. We use a default detection confidence threshold of 0.55. This can be overwritten at the command line.

The four bounding box numbers returned by SSDLite are the left, right, top, and bottom edges of the bounding box proposed by SSDLite for each detection.

The Simple Online and Realtime Tracker (SORT) (5) is a very basic tracking algorithm. With parameter min_hits = 3, we initialize a track when it contains three matching frames. Kalman Filtering is used for first-order motion estimation to predict the position in subsequent frames. For each new frame, a cost matrix is constructed from the outer product of intersection over unions (IOU) (11). IOU is computed to measure the alignment between bounding boxes of the new detections and the predicted next positions for bounding boxes of existing tracks. The minimum IOU to permit a match is 0.4. Because there are often multiple existing tracks as well as multiple new detections, the cost matrix defines an assignment problem (12) that is solved by the Hungarian algorithm (10). Another parameter, max_age = 3, terminates existing tracks if they fail to match a new detection for three consecutive frames.

With tracks generated, the final step is to classify tracks as approaching vs. not approaching. For this, we consider the most recent 25 frames (approach_tracking_depth = 25). We fit a linear regression for the bounding box heights and widths through these 25 frames. If the regression slope shows bounding box height increasing at a rate that exceeds a threshold amount, then the track is classified as approaching, and the user is alerted. We are using an approach tracking threshold rate of +4.0 pixels squared per frame, determined experimentally.

When a vehicle exceeds the approach tracking threshold, our prototype alerts the rider with an audible alarm.

See Figures 8 and 9 for still frames that illustrate the algorithm in action.

Detected cars have bounding box, red = approaching, green = not approaching

Figure 8 Examples of Processed Video Frames with Car Approaching Detected

(a) one approaching car, one not approaching car, and (b) one approaching car and one bicycle (not detected, excluded class)

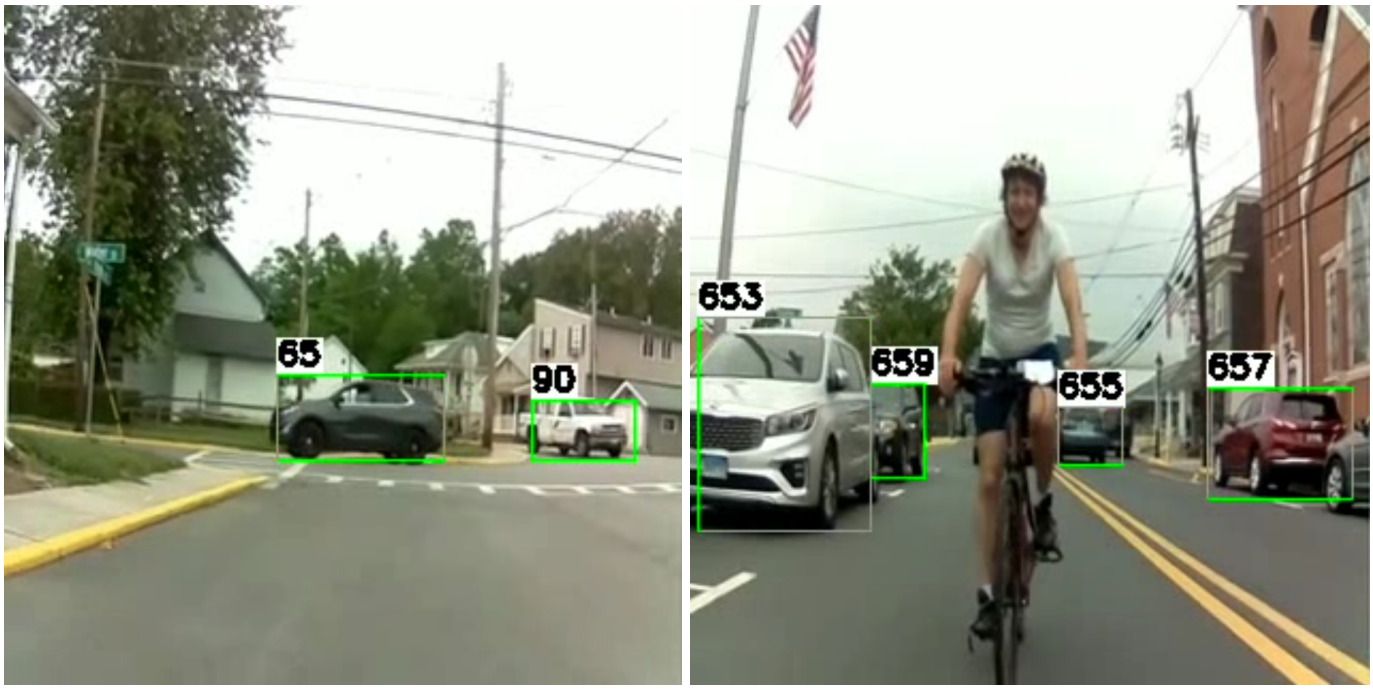


Figure 9 Examples of Processed Video Frames with No Detected Approaching Cars

(a) two cars detected but not approaching - heading in different directions, and (b) numerous parked cars detected but none approaching, bicycle not detected (excluded class)

5 Test Results

In our view, the easiest effective way to obtain a measure of how well CAR-BAC is performing is to simply try it out under real world conditions. So we had an experienced cyclist ride around for a while with the CAR-BAC installed on his bike and to provide feedback as to how well it did at detecting hazardous scenarios. This assessment was subjective as there were no measurements taken or quantitative analysis performed, but rather we relied on the judgement of the rider.

In general, the CAR-BAC system worked very well. It is very reliable at detecting an approaching car or truck and generating an audible alarm that is easy to hear. The rider can often become aware of the car before it is heard.

Of course, there is always room for improvement. Occasionally there were false positives and the buzzer would go off when there was no approaching car. This usually would happen when riding by a string of parked cars and individual cars go in and out of view due to occlusion, confusing the tracking algorithm and producing a positive slope in bounding box sizes that results in the alarm going off. Another area for improvement would be for CAR-BAC to detect a car earlier to give the rider more warning. Sometimes, but not always, the approaching car is not detected until it has already gotten somewhat close behind. We also noticed that lighting is an important factor and that CAR-BAC does not detect objects as well when it starts to get dark out in the evening. Finally, when any improvements are designed and implemented to address any of the above observations, we would also seek to enhance the test assessment process itself by defining a quantitative metric by which alternatives can be compared objectively.

In the end, we want the rider to see CAR-BAC as an automated, transparent way to improve his/her awareness and safety on the road. Of course, the rider must never completely let down his/her guard and rely 100% on a device such as CAR-BAC, no matter how well it may perform. But by augmenting the rider's situational awareness, CAR-BAC could save lives by improving riders' decision-making at the moments when evasive maneuvers come into play. For example, a cyclist experiencing multiple threats (e.g. vehicle approaching from rear plus a dangerous animal or pavement condition in front) may not detect all threats. By providing a different detection mechanism, CAR-BAC could

6 Project evolution/lessons learned

We found that working on the CAR-BAC project was an incredibly fun and inspiring experience. Looking back on it, the most remarkable thing is that the path we ended up taking was unpredictable and that we ended up in a very much different place than we had initially imagined.

When we first started the project, after coming up with the idea, we expected a much more involved process. From the onset, we had assumed that we would have to invent and develop many components of the system. We planned to spend a lot of time collecting data and training models. We searched for publicly available self-driving automobile driving datasets and planned on augmenting that with data that we would collect from our own bicycle riding. This would have undoubtedly required an arduous labeling processIng the end. We didn't have a clue how we would determine whether a detected vehicle is approaching or how we would raise an alert. We figured we'd have to spend a lot of time exploring the design space and experimenting and then fine tuning to optimize our models and algorithms along the way. We assumed that we would be writing a lot of software, too.

Instead, we managed to construct a reasonable prototype without performing most of that work. The software is a Python script, and the components are all off-the shelf: the model and some example code from Google Coral AI, open-source Python libraries for everything else. There was only a small amount of custom code required to combine the pieces and control the system. We are pleased with the outcome - a very useful apparatus that actually works!

6.1 Lessons Learned

The most important lesson we learned while completing this project was to be flexible and to "fail fast". It turns out that failing fast doesn't always lead to failure! Sometimes an idea that seems too simple can work better than expected. In this project, we were surprised by the successful demonstration we achieved with two software components directly off the shelf (the pre-compiled Edge TPU detector and the SORT detector) and a classifier based on linear regression. We learned not to get hung up spending time optimizing one component but rather to advance quickly to find workable solutions with the priority to build a proof-of-concept prototype. The design can always be improved upon at a later point.

7 Ideas for Improvement/Next Steps

As we worked on developing CAR-BAC, we thought of many ideas to make it even better. We were not able to try all of these given the available time to complete the project, but we made a list and will consider revisiting some of these to try to make CAR-BAC even better.

The ideas fall into three major categories – (1) finding ways to miniaturize the device to make it smaller/lighter/lower power consumption/cheaper (2) improving the performance – accuracy, battery life, etc…, and (3) new features that can be developed to add more capabilities.

7.1 Miniaturization – making CAR-BAC more "lightweight"

We would like to reduce the size, weight, power consumption, and cost of CAR-BAC as much as possible so that it is useful and accessible to bicycle riders. This can be accomplished by swapping out the various hardware components. Instead of the Raspberry Pi 4 with Coral TPU, for example, we can consider using the Raspberry Pi Zero or the new Coral Dev Board Mini. It may be possible to get away with a smaller and cheaper camera, too, as the Arducam 5MP is certainly an overkill for the application. If the device can be made small enough, it may be packaged into a single unit that could mount nicely under the saddle of a bicycle.

7.2 Improving the performance and accuracy

There are several ideas that came to mind that could potentially improve the performance and accuracy of the CAR-BAC system. The goals would be to improve the sensitivity and false positive rate and improve the advance warning time of the alert (i.e. detect the approaching time earlier)

First, there are several thresholds and parameters that can be tuned to get better performance. For example, the thresholds and other parameters associated with object detection, object tracking, and approach detection.

Next, it may be possible to improve upon the object detection model by re-training it with better data. The SSDLite model used in CAR-BAC is a completely off-the-shelf example for the Coral TPU. We suspect that object detection for this application could be improved if we trained a model using a dataset that is more representative, e.g. if we used just images with cars and trucks on roads rather than the COCO dataset which includes 80 different classes.

As for object tracking, CAR-BAC uses the SORT tracking algorithm, which is very simple but has some significant limitations. In particular, it does not have any mechanism to address occlusion (i.e. when a car temporarily goes behind an object such as another car) and hence the tracker loses the car under these conditions, resulting in false positives in certain scenarios, such as when riding by a string of parked cars. There are better, more accurate tracking algorithms and one could be adopted by CAR-BAC to improve tracking.

7.3 New features

Some examples of ideas for some nifty new features that could be added to CAR-BAC

- Add a handlebar mount with a display that the rider can easily see.
- Add additional sensors to the bicycle to provide more information to the algorithm to make better, more accurate decisions or detect other types of safety hazard conditions.
- Add lights to CAR-BAC. These can be lights to alert drivers that they are approaching the car.

8 Conclusions

Our team set out to learn more about low powered, AI enhanced, visual recognition systems and to create a proof of concept application that could benefit society. We have achieved, and exceeded these goals while completing this project. Along the way, we learned:

- How to create large, labeled data sets for model training. We also learned that no model training or transfer learning were necessary to complete our proof of concept. This could have a large impact on project management for some ML projects by allowing savings on data acquisition and prep in the earliest phases of a project.
- How to combine off the shelf components and open source software to quickly assess the feasibility of potential solutions.
- How to effectively collaborate with a remote team. Two members of our project team were located in southern California and two members were from Washington DC.
- CAR BAC really works! We are excited to continue with new phases of the project.

9 References

1. Project video:

   https://youtu.be/V4aLnohf_z0

2. Project Github repository:

   https://github.com/Bethesda-AI-Projects-Lab/bicycle_vision

3. SSD Lite MobileDet model for the Coral Edge TPU (Tensorflow Lite)

   http://download.tensorflow.org/models/object_detection/ssdlite_mobiledet_edgetpu_320x320_coco_2020_05_19.tar.gz

4. Coral Edge TPU example for object detection

   https://coral.ai/models/object-detection/

5. SIMPLE ONLINE AND REALTIME TRACKING. Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft, Queensland University of Technology, University of Sydney

   https://arxiv.org/abs/1602.00763

6. SSD: Single Shot MultiBox Detector. Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg

   https://arxiv.org/abs/1512.02325

7. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun

   https://arxiv.org/abs/1506.01497

8. Mean Average Precision

   https://en.wikipedia.org/wiki/Evaluation_measures_(information_retrieval)#Mean_average_precision

9. COCO Dataset - Common Objects in Context

   https://cocodataset.org/#home

10. MobileNetV2: Inverted Residuals and Linear Bottlenecks. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, Google Inc.{sandler, howarda, menglong, azhmogin, lcchen}@google.com

    https://arxiv.org/abs/1801.04381

11. IOU - Intersection over Union

    https://en.wikipedia.org/wiki/Jaccard_index

12. Assignment problem

    https://en.wikipedia.org/wiki/Assignment_problem

13. Hungarian Algorithm

    https://en.wikipedia.org/wiki/Hungarian_algorithm