2019 CSE-40522 Capstone Design Project

Final Report

DriveND: a mobile autonomous driving assistant

Prepared by:

Seth Cattanach, Brendan O'Donnell, Mitchell MacKnight

University of Notre Dame

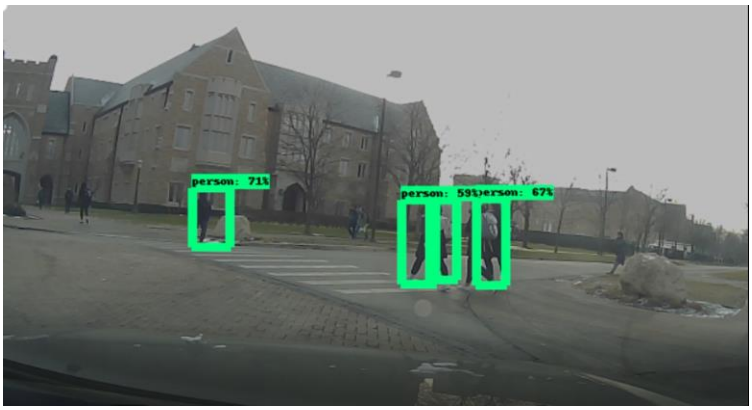
Notre Dame, IN, 46556

On

Friday, December 20, 2019

1. Objectives and Significance

DriveND is a portable, autonomous driving assistant with two primary functionalities: pedestrian detection and lane control. Car manufacturers internationally have adopted autonomous vehicle technologies, but these systems remain largely a luxury as they are tied to expensive models such as the BMW Series 5[1] and the Tesla Model S[2], both of which exceed USD 50,000.



Figures A and B: Screen capture of the DriveND system during development, demonstrating (a) pedestrian detection and (b) lane detection

DriveND lowers the barrier to entry by introducing assistive driving technologies at a significantly lower cost.

The primary features of the system are intended to assist drivers in high-risk driving situations. DriveND actively monitors driving conditions and can alert the driver when a pedestrian is detected (a collision threat) and when the vehicle veers outside the lane boundaries (a symptom of inattentive driving). The system emits visual and auditory signals that can provide the driver with useful, real-time information while allowing the driver to remain fully in control of the vehicle.

Driver safety is an area of clear and increasingly concern, and autonomous vehicle technologies have the potential to decrease vehicle injuries and fatalities. Due to its portability and cost, DriveND aims to change safe driving from a luxury to a common commodity.

[1] https://www.bmwusa.com/vehicles/5-series/sedan/pricing-features.html
[2] https://www.tesla.com/models/design#battery

2. Rationale and Related Background

The Society of Automobile Engineers (SAE) categorizes driving automation into six different levels[3] primarily based on the level to which an automation system (as opposed to the driver) can operate a vehicle. The DriveND system, and many advisory systems such as blind spot detection and backup cameras, fall within Level 1 automation as they have no control over the vehicle's operation itself but rather can assist a driver by providing additional information in real time. **Figure C** (below) details these automation levels in greater detail.

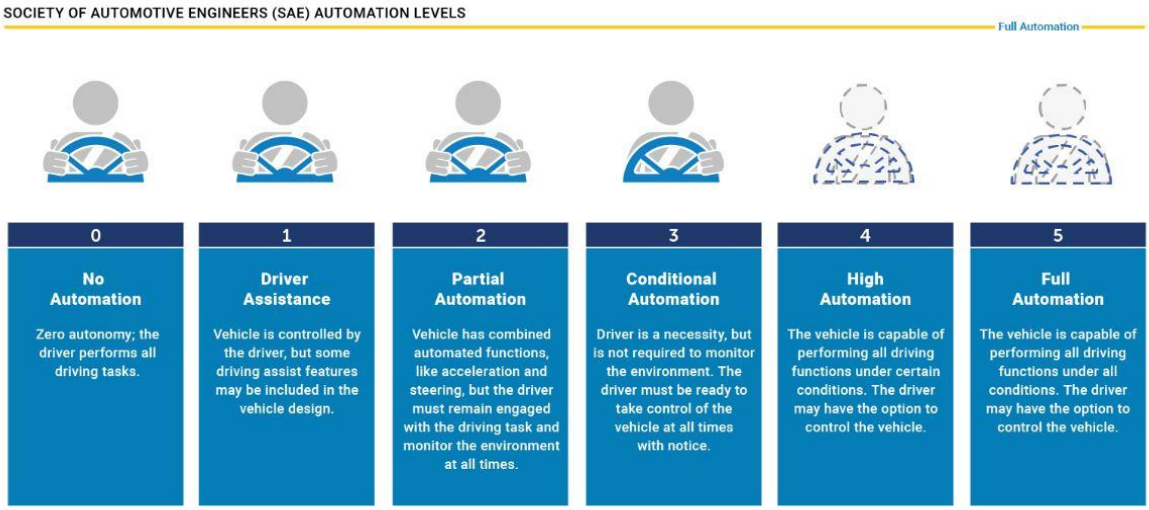


Figure C: SAE Automation Levels

The DriveND system's inability to directly control vehicle operation is a limitation that results from its portability requirement: the system would have to be integrated with the vehicle itself and thus could not be portable. However, this is also an advantage, as a false positive or false negative in a Level 1 system such as DriveND does not adversely affect vehicle operation (that is, if the DriveND system misidentifies a lane marking due to weather or light conditions, it will not cause the vehicle to unexpectedly swerve).

Due to its portability and low cost, this system could potentially be applicable in any driving situation and thus has an extremely broad target market (the U.S. automotive manufacturing industry was worth over **$750 billion** in 2018[4], and our system serves as a complementary product with few existing competitors). Any patents filed would likely have to account for the

[3] https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[4] BEA. (April 19, 2019). U.S. motor vehicle and parts manufacturing gross output from 2005 to 2018 (in billion U.S. dollars) [Graph]. In *Statista*. Retrieved September 19, 2019, from https://www.statista.com/statistics/258075/us-motor-vehicle-and-parts-manufacturing-gross-output/

integration of both the hardware and software components, as there are existing software systems (developed by Tesla, Audi, etc.) with a similar purpose; furthermore, the hardware used is proprietary technology (our system relies on existing commercial hardware including a Raspberry Pi).

3. Approach

Our system consists of both software (object recognition, lane detection scripts) and hardware (LEDs, piezo, push-button) components, so we developed each of these in parallel until each component was integrated into the final program on the Pi. The functionality and design of the final product is shown in **Figure D.**

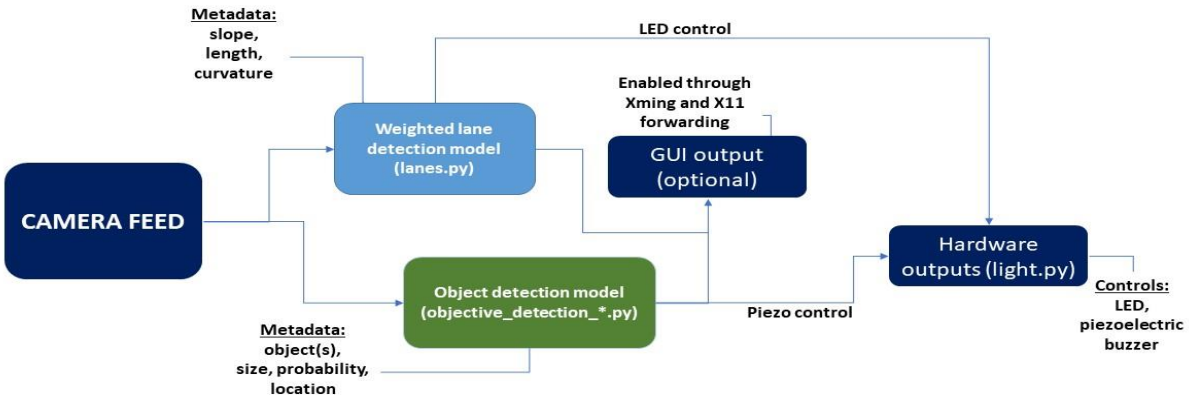


Figure D: Design Components and Input/Output

For the object recognition and lane detection functionalities, we adapted open-source code (TensorFlow) and optimized the resulting models for computational efficiency. Our hardware controls received input from our 2.0-megapixel USB camera at 30 frames per second (hardware failure on the 6.0 MP, 60 FPS camera prevented our team from integrating a stereoscopic video feed as originally planned).

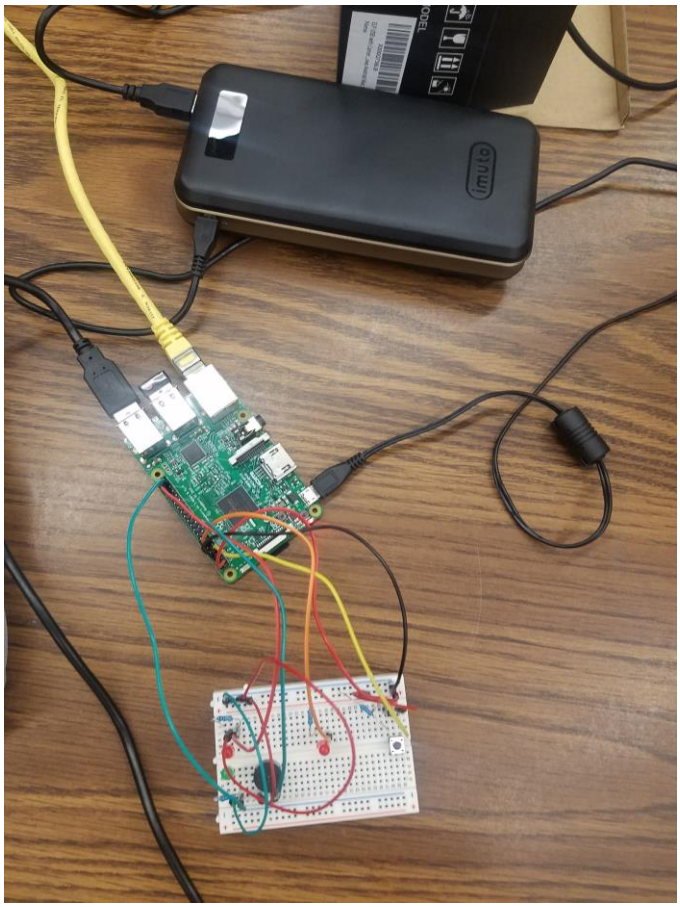
After installing the TensorFlow and cv2 packages on our Raspberry Pi, we adapted our models to ensure compatibility with the Raspbian (Debian) environment using Python 2. We integrated our hardware controls to provide sensory outputs through the LEDs and piezoelectric buzzers at

this time. Using a GUI for debugging purposes (to display the system's video feed), we fine-tuned our system parameters to optimize for both speed and accuracy. When both speed and accuracy fell within our established baseline (a delay of less than 1000ms in both models, and a recall of at least 95% in our object detection model), we configured the Pi to run the program immediately via startup scripts.

4. Project Outcome

The final DriveND system can detect pedestrians and alerting the driver of their presence by sounding a piezo buzzer. Additionally, it can detect the lanes surrounding the car, and turn on a red LED when it determines that the car is outside these lane boundaries. The program will run continuously when the Pi is powered on; it shines a red LED to indicate that the program has begun importing files and packages, and a green LED to indicate that the main loop of the program has begun. A physical control button can end the program at any time. It is powered using a high-capacity power bank, which can be seen along with the other hardware components previously described in **Figure E** (the system could be easily adapted to draw power from an integrated car charger, assuming the power adapter maintained a 5V supply). An in-depth diagram of the system's circuits is also shown in **Figure 1** in the appendix**.**

Figure E: Final setup of the DriveND system



This system works by capturing a video feed through a USB camera and passing each image of this feed to customized TensorFlow model. Each image is also processed by a lane detection algorithm, which identifies lane locations by first identifying candidates (straight lines) with a weighted slope calculation and then calculating the two longest segment candidates (which are visualized on the GUI's video feed in white, when enabled). This model determines whether the car is outside the detected lanes by checking the coordinates of either lines' bottom pixels

against a threshold value; this value was manually set based on manual observation but could be optimized further by incorporating a feedback loop to identify false positives (instances where the system incorrectly flagged a car as drifting outside the lane) and false negatives (instances where the system failed to alert the driver) in a future design.

A video demonstration of the DriveND system in production can be viewed here: https://youtu.be/KGLO_7UrrXw

5. Unexpected Events

We encountered difficulties getting the Pi to recognize our ArduCam (high-resolution camera), and ultimately decided to move the project forward using only the lower-resolution USB camera. As a result, we ultimately did not use a stereoscopic video feed as an input to our system, but we were still able to maintain the core system functionalities (namely, lane detection and obstacle detection/avoidance) outlined in our proposal.

A significant barrier arose while configuring dependencies on the Pi itself (to install packages necessary for image processing, such as OpenCV). The vast majority of TensorFlow distributions are developed for 64-bit operating systems, and Raspbian is a 32-bit operating system. As a result, we had to build and deploy TensorFlow from an open source PyWheel distribution. The workaround allowed us to deploy an older version of TensorFlow (v1.10), but created additional dependency problems - namely, this version of TensorFlow required Python 2. As a result, we downgraded our Pi to run Python 2.7 by default, which required rebasing our code from Python 3.

We also failed to fully predict how strongly our system would be impacted by weather conditions. Though we expected the system's accuracy to be impaired by snow, the greater concern was due to brightness and unusual light angles. When the camera was angled into bright sunlight, the resulting video feed was not of sufficient quality to perform lane or object detection. Lane detection, in particular, was greatly affected by adverse light conditions. Future iterations of this system should perform additional preprocessing on the video feed to ensure consistent lighting.

6. Project Performance

Our project fully fit within our proposed budget. We did have some technical difficulties with our Pi and one of the cameras, though we were able to resolve all issues without any additional cost. We resolved these issues by rebasing our code to run using Python 2.7 rather than Python 3, to ensure compatibility with our 32-bit Raspbian operating system and TensorFlow version 1.10, and foregoing stereoscopic vision to ensure an accurate video feed with minimal risk for hardware failure.

Most of our objectives were accomplished by date specified in the timeline in the project proposal. Some unexpected hardware issues and Pi configuration issues resulted in some setbacks. Additionally, integrating the different software components took additional time, largely due to the code rebasing activities. As a result, final production environment testing was delayed by about a week, though it was completed ahead of the final deadline.

While the final DriveND system includes all proposal features, it does have some limitations. When the system operates exclusively in lane detection or object detection mode, the model outputs its predictions in nearly real time (less than our 1000ms baseline), but the video feed has a delay (3000-4000ms) greater than our baseline when using both the object and lane detection features. The system's accuracy is also strongly affected by light conditions, which could perhaps be mitigated by using a higher resolution, specialized camera for outdoor use. In perfect (high visibility) conditions, the system did perform within its expected baseline (greater than 95% recall), but in less-than-ideal road conditions (snow, low and direct sunlight), the system operated below 95% recall.

Appendix

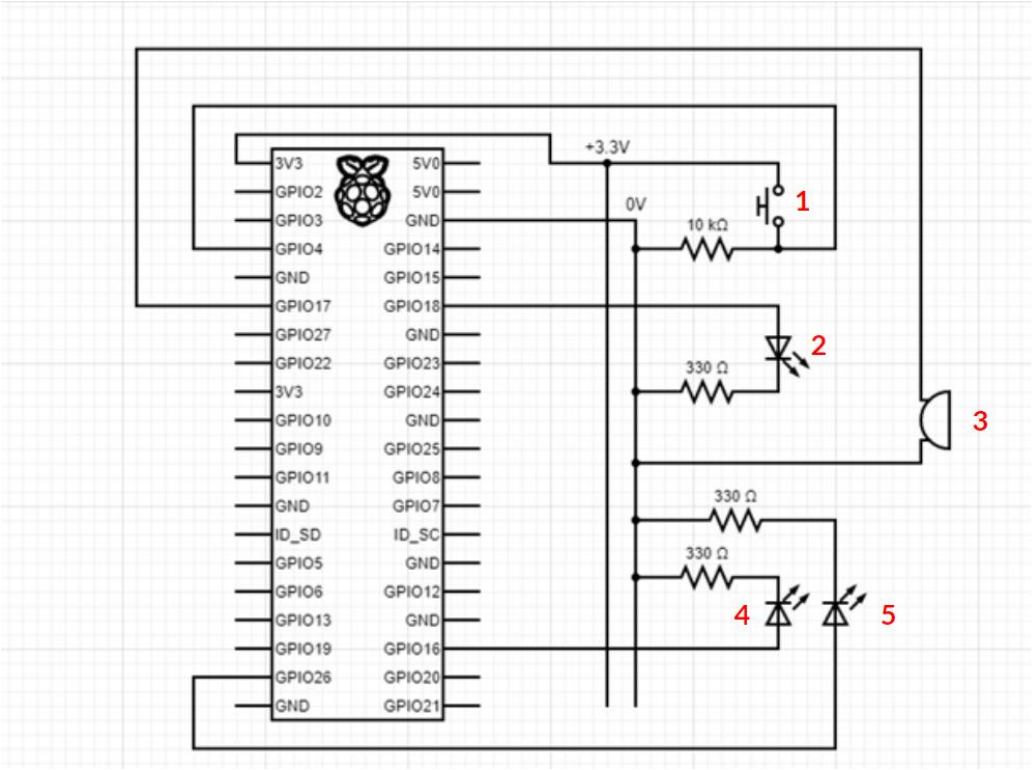


Figure 1

1. Power button
2. Red LED for lane detection
3. Piezo buzzer for object detection
4. Red LED to indicate startup
5. Green LED to indicate program running