UNIVERSITY OF WASHINGTON

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EE 474 Final Project Report

Team
TIANNING LI, LIRUI WANG

Professor
RANIA HUSSEIN

Teaching Assistant
SHUOWEI LI, NIKOLA DANCEJIC, TAVIO GUARINO

UNIVERSITY of WASHINGTON

EE 474 Final Project Report

Tianning Li, Lirui Wang

***Abstract*—Drowsiness of driver is one of the most significant cause of road accident. In this work, we present an embedded system that that simulates a real-time drowsiness detection system based on visual information. We propose three different components that communicate via Bluetooth. Specifically, we use a two-wheel car developed on Tiva LaunchPad to simulate a real autopilot car. A Raspberry Pi is used as a detection system to communicate the eye closure with the car and activate the autonomous mode of the car. Moreover, we provide a LCD display of the current car status to mimic a cloud system that monitors the current car status.**

I. Introduction

During human daily activities, we all can be victims of driving drowsily for a short night sleep or during extended journey. Since drowsiness often denotes decrease of driver vigilance and increase of accident probability, we are to address such problem with a complete embedded system under proper specification to meet the design objective. Formally, we want to design an embedded system that simulates real-world drive mode switches and monitoring of an autopilot car (Fig 1). Such a car provides user control when drivers are aware, and switches to autopilot mode when a potential driver drowsiness is detected by the computer vision system. Additionally, we also upload the current car status to a cloud system that can store this information. The following paper would be composed of detailed explanations of each component in our embedded design. Then we also discuss the application extent and potential issues with this toy example and look forward to future improvements.

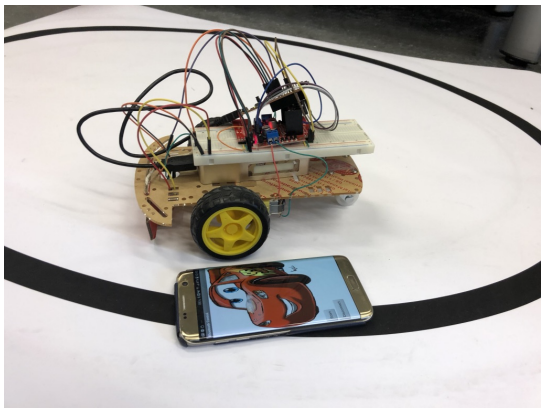
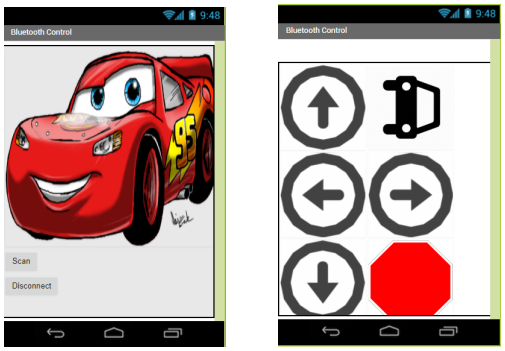


Fig. 1: Autonomous car

II. Front End

The front end app (Fig 2) is based on Android operating system and built with MIT App Inventor. It acts as a user interface for the driver to control the car with Bluetooth connection. On one hand, it calls a Bluetooth client module and sets up Bluetooth connection with the car. Note that the App Inventor does not support direct multi-screen resource sharing, so we build a visibility switch based on the Bluetooth connection. On the other hand, when Bluetooth is connected, the original page (Left) is hidden and the other page (Right) shows up. After communication is set up, the system becomes a button-based control panel for sending commands. The available commands are left, up, right, down, auto mode and stop. The direction and stop commands would control the car to move correspondingly, the auto mode would activate the autodriving mode of the car. In terms of graphics, we select user-friendly icons within a table template. The overall front end system responds swiftly and have barely any unexpected behavior.



(a) Front End Screen 1 (b) Front End Screen 2

Fig. 2: FrontEnd Display

III. Autonomous Car

The autonomous car is equipped with two motors. Since the TIVA board can only support maximum of 8 mA current, which is not enough to drive the motors, we use L298N to drag up the power(Fig 3). Specifically, the left motor is connected to PA2 and PA3. The right motor is connected to PA4 and PA5. These ports will be connected to the motors and thus determine the movement of the car. The car will receive signals from the Bluetooth module and will give different highs and lows to the GPIO ports.

There are two Bluetooth on the TIVA board. One is using UART1 which is connected through PB0 and PB1. The other is using UART 3 which is connect through PC6 and PC7. The UART1 will receive the signal from the Android app. As mentioned earlier, Android app will send FORWARD,

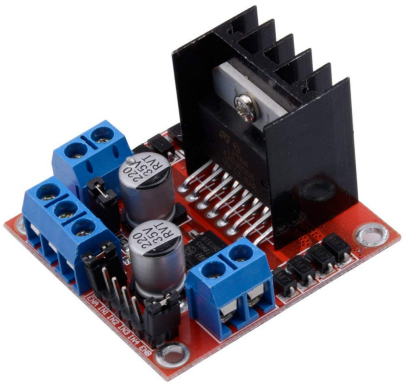


Fig. 3: L298N [1]

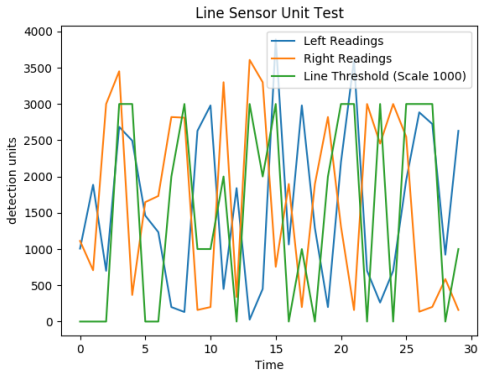


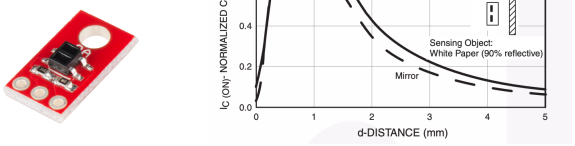
Fig. 4: Line Sensor Unit Test

BACKWARD, LEFT, RIGHT, AUTOPILOT and STOP signal to the TIVA board. At the same time, UART3 receives command from the Raspberry Pi. Raspberry pi can only send AUTOPILOT signal to the TIVA board when continuous eye closure is detected. From the car's perspective, it would echo the commands from the phone and from the Raspberry Pi, which further maps to LCD display.

When the car is on AUTOPILOT mode, it will track the black line in white space. The AUTOPILOT mode will turn off when the user hit any button on the phone. There are two QRE1113 line sensors to detect black and white region. The line sensors will return a value between range 0 to 4095, similar to the built-in temperature sensor. Based on our experiment, a reading below 500 denotes a white region and a reading above 2500 denotes a black region. Based on these thresholds, we can easily determine what's the current direction of the car.

$$V_{result} = \frac{X * 5V}{4095}$$

From the formula, for instance, when sensor hit a white region, it will output 0.6 V. As a unit test for our threshold, in Fig 4, we show the sensor readings and the corresponding case of the car. For simplicity, we use 0 to denote both left and right detect black line, 1 to denote only left sensor detects black line, 2 to denote only right sensor and 3 denotes neither does. Then, we can base our line tracking algorithm on the readings. Note that it's noisy since these are analog readings, and we scale the case constant by 1000 to fit into the same figure. Overall, the behavior of the green line from this unit test confirms our predefined threshold above.
In a nutshell, these two line sensors, which are mounted under the car will control the direction of the car to follow the black line. It will turn right when the left sensor detects the white region and turn left when the right sensor hit the black region. When both the sensor detect black, it will move forward. When both the sensor detect white region (off the road), it will keep turning left in circle until it detect the black line again, which acts as a recovery method. The pseudo code is shown below:



(a) QRE1113 (b) Distance-current graph

Fig. 5: QRE1113

RW: Right Wheel
LW: Left Wheel

Line Tracking Algorithm

```
While Auto Mode:
    if (Left in black and Right in black):
        Both Wheel Forward
    if (Left in white and Right in white):
        Turn left (RW move forward, LW stop)
    elif (Left in black and Right in white):
        Turn right (RW stop, LW move forward)
    elif (Left in white and Right in black):
        Turn left (RW move forward, LW stop)
```

IV. Detection

At most cases, a driver drowsiness can match with their continuous eye closure, which is a catalyst in daily traffic. Therefore, we formulate the problem and choose a eye closure detection algorithm to discover the drowsiness. The detection module is a standalone Raspberry Pi Model 3B+ with a night vision camera. Our 5MP OV5647 Webcam has adjustable focus with extra infrared LED sensor and therefore support night vision with almost no light. This camera simulates the real-world driving scenario better and provides more robust performance at night.

As the real-time frames are streamed from the camera, our drowsiness detection system starts to process the data.

The detection system consists of two parts: Face Detection [3] and Eye Aspect Ratio [4] Computation. The face detection system uses an ensemble of regression trees to estimate the faces landmark positions directly from a sparse subset of pixel intensities. This Ensemble of Regression Trees (ERT) method utilizes simple and fast feature and subsequently refined with an iterative process done by a cascade of regressors. The algorithm is blazing fast, in fact it takes about 13ms. The advantage of the detection pipeline is that it runs in real time and suitable for embedded system. We download a pretrained model from [5]. A few parameters closed associated with the model is that tree depth – the depth of the tree used in each cascade and denote the capacity of the model. Nu, as the regularization parameter that determines the ability of the model to generalize and learn patterns. Oversampling amount denotes the number of randomly selected initial starting points sampled for each training example. In test time, we fix these number nu=0.05, tree depth=3, oversampling amount=300, which are the same as the pretrained model. The face landmarks and detected faces from this method is then fed to compute the eye aspect ratio $EAR = \frac{p_2 - p_6 + p_3 - p_5}{2p_1 - p_4}$, where denotes Euclidean norm. This formula follows the intuition that an eye closure can be denoted as an eye with low aspect ratio.

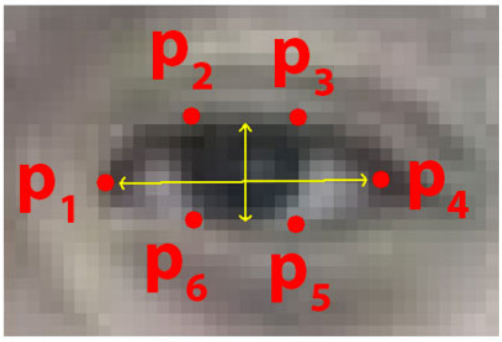


Fig. 6: LandMark

Therefore, we use such information as a key feature to detect the driver's drowsiness. We compute the average aspect ratio of the first 30 detected frames to be the user-specific information u. $0.8u$ would be used as a threshold for drowsy eye. At test time, we count consecutive low eye aspect ratio frames and reset the counts if such pattern ends. We mark one full eye drowsiness a continuous 10 frames eye closure as we do not want to overreact to eye blinks. Based on the detected landmarks, we draw a convex hull around the user's eyes and also put the eye aspect ratio the top right corner of the image. Whenever drowsiness is detected, we would send the Bluetooth command, through Pyserial, to activate the autopilot mode of the car. This simple machine learning algorithm has a drawback that as a traditional feature extraction pipeline, it's not robust enough in all scenarios. For instance, partner's eyebrows are often wrongly classified as eyes.

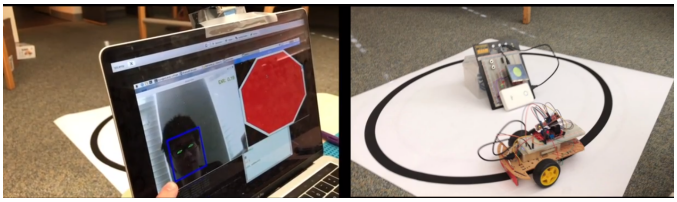


Fig. 7: Detection

V. LCD Display

The LCD display system uses freeRTOS (Real-Time Operating System) to set up a communication task with Bluetooth. This Tiva Board runs a default task scheduler to manipulate the Bluetooth and choose display for LCD. To save implementation redundancy, we chooses another TIVA board and uses UART3 as communication. Even though FreeRTOS can be used to manage more sophisticated tasks and processes, here we only use it to display LCD and respond to user touch. The LCD display will display the current state of the car. It shows direction arrow if the car is moving forward, backward, left and right. It will show red circle when the car stops and yellow circle when the car is autopilot mode. The yellow circle acts as a notification for other drivers and passengers the car is on auto mode. Moreover, we develop the touch screen function of the LCD display and ask it to change color for the next moving state. Overall, This LCD displays the current status of the car and allows, for instance, a cloud system to monitor the driver. Note that it should also be able to mounted on top of the car.

VI. Communication

From a system perspective, we set up a communication network that involves five Bluetooth modules with three mounted on car, one on the front end and the other one on Raspberry Pi. A successful communication is the building block for most modern embedded system. In Driver Mode, a driver sends a direction commend via Bluetooth, which is recieved by the car and echo to the Raspberry Pi (Fig 7). Rasperry Pi would display the sign and send the signal to the Bluetooth on LCD Tiva Board. At the same time, Raspberry Pi, as a backend in our project, can activate the autonomous mode of the self-driving car. The reason why this complicated communication system (Fig 8) comes to play is that we want to simulate real-world scenario where there exists multiple communication server and clients to share the information such as car status and processed commands.

VII. Discussion

Imagine that the little Tiva Car is the future autopilot car, (Tesla, Honda, etc) and the second Tiva Board with LCD display is the cloud system (developed by car producer as a server) for managing driver and car information. The local Raspberry Pi would be a real-time vision system that not only detects drowsiness but also helps driver transit to autopilot mode without notice. This could be useful in a potential traffic accident. A demo video is

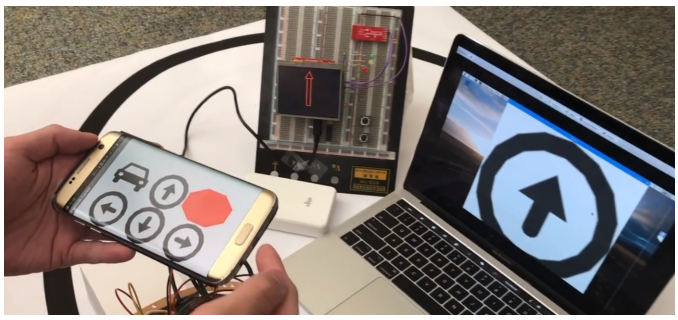


Fig. 8: Communication

attached in the link https://drive.google.com/open?id=1z3Bn2V7WmgyHsC9uP5rhBkZpuAx-xkRn. In the attached video, the driver controlling the car feels asleep and the drowsiness detection manages to send the autopilot command to the car. As a result, the car switches to autodriving without leaving the circle. This in-time respondence and robust performance is what would place a guarantee on the driver's safety.

We would now discuss a few issues and potential improvements in our implementation. First, we rely on the number of Bluetooth. Indeed, with better management, we would need a single Bluetooth on Tiva car to act as an on-board server and client. The map can also be redesigned to be more complicated, as a circle does not reflect the complexity of real-world roadmap. Some issues with the autonomous car is that it sometimes gets stuck and drift out of the circle. In addition to the power and hardware issues, our mechanical design might not be optimal. The sensor readings can depend heavily on the environmental light and map reflection. Similarly, a collision-avoidance system can be added to further close the gap with real scenarios. In terms of the demo, a good way to remove the extra Raspberry Pi in our project is to embed Dlib that runs drowsiness detection into the Android application. Another thing is we are currently using line sensors. The driving is very bumpy especially when the car is turning direction. For future work, we would need a front camera that can capture image in the front and do machine learning analysis to not only predict the path of the front but also turn the direction at the correct time to reduce bumpiness.

Some extended functions include a speaker system for warning drowsiness, which can be built with a HDMI cable with sound function or an extra speaker. The car system can be rebuilt with four-wheel drive that provides more power to carry the LCD. It would also resolve the issues with omni-direction wheel and stuck situation of the car. In terms of the detection algorithm, this Adaboost pipeline with simple intensity summation as image features can be replaced by a HMM (Hidden Markov Model) or a SVM (Support Vector Machine) model that we expect to extract more robust keypoints with small loss of efficiency. With more computation power, the trending data-driven deep learning method seems also promising in terms of generalization. These improvements and issues are worth trying and not too complicated to implement. The details of our implementation can be found in submission and the overall flowchart is plotted in Fig 9.

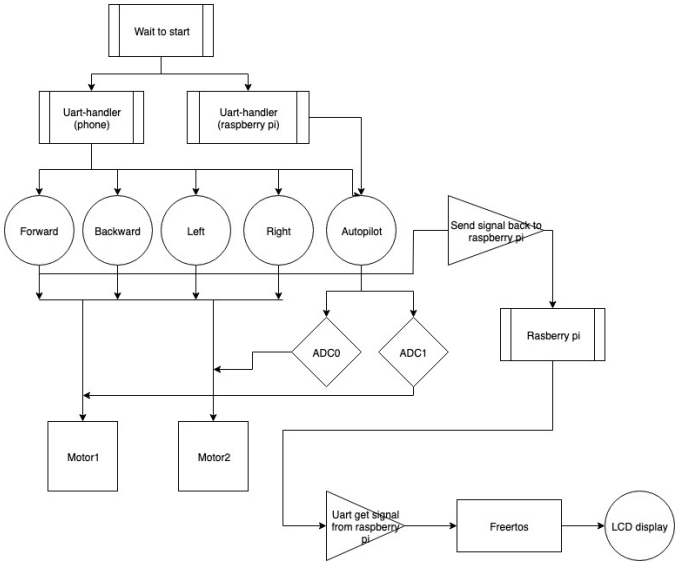


Fig. 9: Flowchart

VIII. Conclusion and Future Work

This functioning embedded system, as a toy image of the real world, presents a working solution to the drowsiness driving problem. In this project, we review the skills of utilizing input/output functions, communication, and modules testing function of the embedded system. We refresh the high-level ideas such as interrupts, RTOS, Bluetooth. We build unit testing and above-and-beyond modules such as a computer vision detection system and a line-tracking algorithm. This project, as a complete functional embedded system, gives us a better understanding of a functional embedded system from software to hardware. Moreover, it's closely related to daily application and can be a bedrock for future works.

REFERENCES

[1] Qunqi L298N Motor Drive Controller Board Module Dual H Bridge DC Stepper For Arduino, Amazon. [Online]. Available: https://www.amazon.com/Qunqi-Controller-Module-Stepper-Arduino/dp/B014KMHSW6/ref=asc_df_B014KMHSW6/?tag=hyprod-20&linkCode=df0&hvadid=167139094796&hvpos=1o1&hvnetw=g&hvrand=7445658577320568149&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9033309&hvtargid=pla-306436938191&psc=1. [Accessed: 11-Mar-2019].

[2] SparkFun Line Sensor Breakout - QRE1113 (Analog), COM-14646 - SparkFun Electronics. [Online]. Available: https://www.sparkfun.com/products/9453. [Accessed: 11-Mar-2019].

[3] V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, 2014, pp. 1867-1874. doi: 10.1109/CVPR.2014.241

[4] Al-Gawwam, Sarmad, Benaissa, Mohammed. "Eye Blink Detection Using Facial Features Tracker". 2016. 27-30. 10.1145/3175587.3175588.

[5] L. Anzalone and L. Anzalone, Training alternative Dlib Shape Predictor models using Python, medium.com, 10-Oct-2018. [Online]. Available: https://medium.com/datadriveninvestor/training-alternative-dlib-shape-predictor-models-using-python-d1d8f8bd9f5c. [Accessed: 11-Mar-2019].