CSE 237A (Winter 2023) Final Project Report
Raspberry Pi-based Car HUD

Team members:
Luyi Li luyli@ucsd.edu A59015929
Yue Pan yup014@ucsd.edu A59019563

Project Motivation

Head-up Display (HUD) is a transparent display that presents crucial information while driving. The motivation for it stems from a desire to improve driver safety and increase situational awareness. Traditional vehicle instrument panels and infotainment systems require drivers to look away from the road, which can increase the risk of accidents and distracted driving. By projecting information onto the windshield or a transparent reflective glass, a HUD allows drivers to keep their eyes on the road while accessing important information such as speed, navigation, and other car information, as Figure 1 shows. This can help reduce driver distraction and improve reaction times in critical situations. Overall, the motivation behind designing HUDs is to create a safer, more efficient, and more user-friendly driving experience for drivers. By minimizing distractions and increasing situational awareness, HUDs can help improve driver safety and make the driving experience more enjoyable and less stressful.

Figure 1. A Head-up Display (HUD) example

This technology was originally invented for pilots. Nowadays, it has been widely applied on commercial cars. There are also some aftermarket modifications sold by various vendors. However, they usually cost high and lack extensibility. In this project, we aim to design a low-cost and highly customizable DIY HUD module based on Raspberry Pi for cars, RPiHUD. Our system deploys several hardware sensors to collect accurate real-time information from the car and designs a user-friendly UI to present the information. Drivers can also customize the information on the HUD and select his/her preferred display mode. To sum up, our main contributions are as follows:

- RPiHUD provides various useful car information, including speed, engine RPM, coolant temperature, GPS location, interior temperatur, acceleration, compass, etc.

- RPiHUD has a user-friendly UI design and automatically color adjustment based on external brightness level.
- RPiHUD has 6 display modes in total. The driver can change the display mode by pressing the button, which makes it more convenient to control while driving.
- RPiHUD is a fully open-source project, which provides a straightforward and accessible code framework to enable future extensions.

Related work

Commercial car HUD products have existed for a long time. Usually they come integrated into the car as a premium feature. It can be an expensive additional package to a vehicle, limiting their appeal to some buyers. Moreover, the vendors usually only present predefined car information on HUD, which limits any further customization.

There are also some aftermarket products sold by third-party retailers. For the installation, retrofitting a car with an integrated HUD can be challenging, and it may not always be possible to install an integrated HUD in an older car. Some other vendors provide portable OBD-II-based HUD, which reads information through OBD-II scanner and projects the information to a reflective glass, as Figures 2 shows. This is similar to our RPiHUD design method. However, they only show limited information and do not provide open-source libraries to enable further extensions.

Figure 2. An aftermarket HUD example

Overall, RPiHUD distinguishes from existing aftermarket products in several ways: the ability to extend the software and UI according to our preferences and demands, the lower cost of the system, the ability to extend hardware to include sensors such as in-car temperature sensor, GPS module etc. This customization can help drivers to better focus on the information that is most important to them. In addition, our system is based on Raspberry Pi, which means we can add additional processing to the data collected from the car. For example, one could add a camera and use image segmentation and recognition models to automatically detect road signs. In conclusion, our design distinguishes us from existing products in terms of extensibility and future potential.

Hardware Components

Hardware Platform**:** Raspberry Pi 4B

Sensors**:**

- ELM327 OBD-II to USB converter: To collect real-time driving information from the car
- VK-162 GPS module (USB): To receive accurate GPS location

- DHT11 sensor (GPIO): To measure temperature and humidity inside the car
- KY-004 button (GPIO): To select the display mode
- MPU9255 Inertial Measurement Unit (IMU) (I2C): To measure the acceleration of the car and calculate the compass direction the car is heading towards based on the output of the magnetometer.
- TSL2591 light sensor (I2C): To measure the visible brightness level inside the car

Other accessories**:**

- 5-inch LCD display
- A reflective transparent glass allowing some light to pass through and some reflect back, so that users can see projected contents and the road ahead

Hardware Design Choices

OBD-II interface

We had two choices for the OBDII interface: a bluetooth option and a cabled option. Initially, we experimented with the bluetooth options and found several deficiencies: for the python library to work, an additional delay term has to be added to every bluetooth send commands, and the bluetooth module will continue to consume the car's battery power after switching off the ignition. In the end, we switched to a cabled OBDII to USB converter to ensure stable connection and power saving, at the cost of adding an additional wire to the system.

Inertial measurement unit

At first we used MPU6050, which consists of a 3-axis accelerometer and 3-axis gyroscope inside it. However, its lack of magnetic information limits us to calculate the correct compass direction. Therefore we turn to MPU9255. It includes a 3-axis magnetometer, which allows for more accurate heading information and magnetic field detection. MPU9255 also has a higher sampling rate and faster communication protocols compared to the MPU6050. This means that it can process and transmit data more quickly and accurately, making it more suitable for applications that require real-time and high-frequency data.

Software Design Choice

GUI Development Frameworks

There are plenty of available GUI application development tools such as Qt, Tk, and web apps. We chose to use Qt for a few reasons. First, the framework needs to be powerful enough to allow some critical features: the text displayed needs to be the horizontally symmetric counterpart of the normal text, so that normal text can be seen from the HUD glass. As far as we know, Tk does not support this feature, while this is achievable through custom paint events on Qt and Cascade Style Sheet (CSS) transformations in any web app. Second, considering Raspberry Pi is an embedded platform with limited compute resources, we avoided using web apps due to their high energy and memory consumption. Therefore, Qt became our choice of design tool.

Software Components and Frameworks

- Programming language: Python 3.10.6
- PyQt 5 (5.16.5): User Interface design

- GPSD-client (1.3.2): GPS information retrieval and NMEA string decode
- Python-OBD library (0.7.1): To communicate with car via OBD-II interface. Note: some manual adjustment to dependencies is required to run this library with Python 3.10+
- Smbus package (1.16): To construct I2C bus-based communication
- RPi.GPIO package (0.7.1): To control GPIO ports and handle interrupts from button
- IMUsensor package (1.0.1): To calibrate and collect data from IMU sensor
- Adafruit sensor packages (8.15.2): To collect data from temperature and humidity sensor and light sensor

Hardware and Software Integration

In brief, real-time information updates from sensors are passed to the Raspberry Pi through their own communication interfaces (GPIO, I2C, USB). These updates are fed into the User Interface (UI) program to update their values displayed on the HUD, or change the HUD behavior in terms of information layout, and color.

Hardware

The various sensors communicate with the Raspberry Pi through various kinds of interfaces. In our project, sensor updates are handled by the `SensorReader` class, which retains the newest information from sensors as member variables. Upon UI request, `SensorReader` can supply the values with minimal delays.

Button: the button pressing is connected to an interrupt to avoid constantly sampling the GPIO pin. The interrupt handler will assert high the mode change field in the UI, which will be sampled when the UI refreshes and set low after changes are done.

GPS: the GPS connects to the Raspberry Pi through the USB interface. It sends NMEA (National Marine Electronics Association) strings containing positioning information. The GPSD-client library is used to receive and decode the string to extract longitude, latitude, and fix mode.

IMU sensor: The MPU9255 IMU sensor can be connected to a Raspberry Pi using the I2C bus protocol. The Raspberry Pi 4B has I2C pins on its board, GPIO2 (SDL) and GPIO3 (SCL), which can be used to establish communication with the sensor. The smbus package[2] is commonly used in Python to implement the I2C communication protocol. Once communication with the sensor is established, data from the accelerometer and magnetometer can be read and processed. The IMUsensor package[3] provides a user-friendly library that can be used to calibrate the magnetometer and calculate the pitch, roll, and yaw angles based on the sensor's output. This information can then be used to calculate the total acceleration in the xy-plane, which can be used as the car's acceleration. The yaw angle can also be used as the heading angle of the compass. For further details on the specific algorithm used for processing the sensor data, please refer to Appendix A.2 in [6].

Light sensor: The TSL2591 light sensor also communicates with the board via I2C bus. For data processing, Adafruit provides a library [7] to allow users to easily read light information from the sensor, including lux value, visible light level, infrared light level, etc. In our project, only visible light level is

used to detect the ambient visible light conditions in the surroundings and adjust the display color of the HUD accordingly.

Software

The user interface is written in PyQt5, a Python implementation of the popular Qt framework for designing GUI applications. An Qt application usually contains a Main Window, on top of which are numerous Widgets placed at different locations. Our project is no exception to this pattern. We designed 4 widgets:

- **`RPMWidget`**: displays engine RPM in revolutions per minute and a bar that indicates the percentage of maximum revolution (8000RPM). It also changes color based on the current RPM. Under 3000, the bar is green, then under 4500, the bar is orange, and greater than 4500, the bar turns red. The update interval of this widget is every 10 milliseconds.
- **`GPSWidget`**: displays the GPS fix status (whether GPS has acquired current position), latitude, and longitude. If the GPS is acquiring position from satellites, the border of GPS FIX text will blink to indicate such status. Update to this widget is done every second.
- **`SpeedWidget`**: displays the vehicle speed in kilometers per hour or miles per hour, depending on user selection. The update of this widget is done every 10 milliseconds.
- **`SensorWidget`**: placed on the right of the screen, containing information about various sensors, including interior temperature, acceleration, magnetic heading and direction (NEWS), and engine coolant temperature. The update of this widget is done every 50 milliseconds.

To update the UI with real-time sensor data, our PyQt program is multithreaded. The main thread is responsible for the UI loop. The second thread processes GPS information, including acquiring such information from the sensor, and decoding the received NMEA string to acquire longitude and latitude information. The third thread is responsible for reading from the various sensors on board, excluding OBD. The forth thread is implicitly declared. The python-OBD library uses asynchronous IO to avoid blocking when the UI program queries updates. The multithreaded design allows the UI to be smooth and comfortable to look at. On the contrary, if multithreading is not applied, the user will notice significant lag to UI updates.

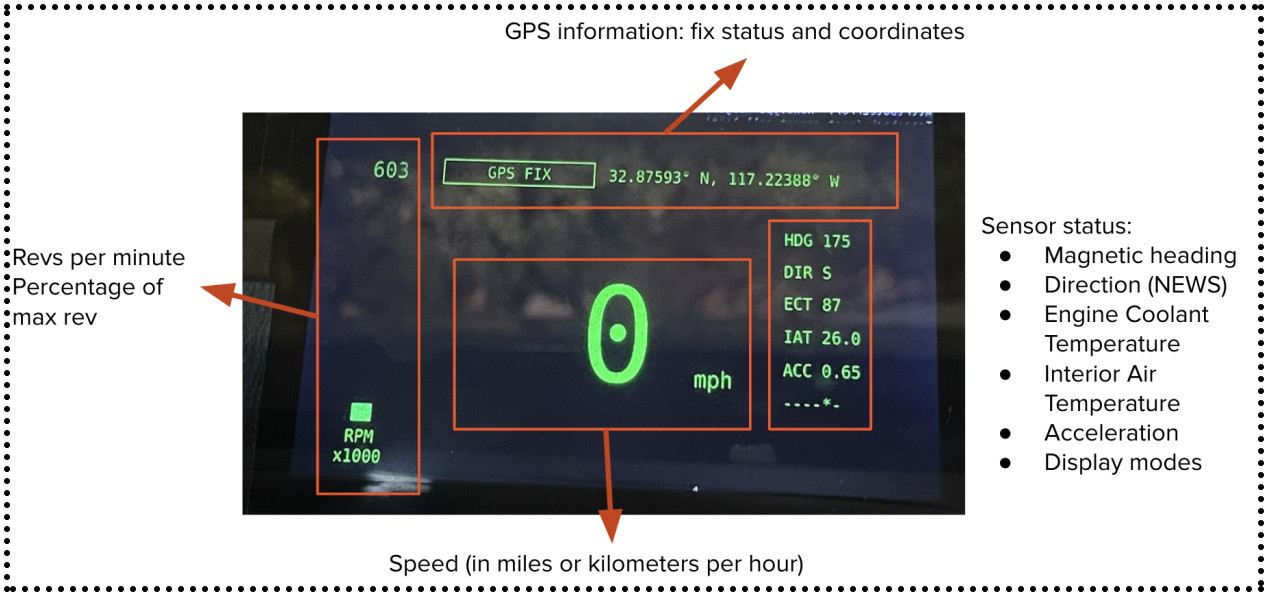

Figure 3: Layout of different widgets on the main window

Aside from updating values, the UI will respond to sensor inputs in additional ways. The UI changes color to green if the light sensor detects high intensity lights, otherwise the UI is displayed in amber. The RPM color also changes depending on the current RPM. The button input allows the user to switch between 6 predefined display modes: “Normal”, “Zen”, and “Full” in both imperial and metric units, as shown in the picture below.

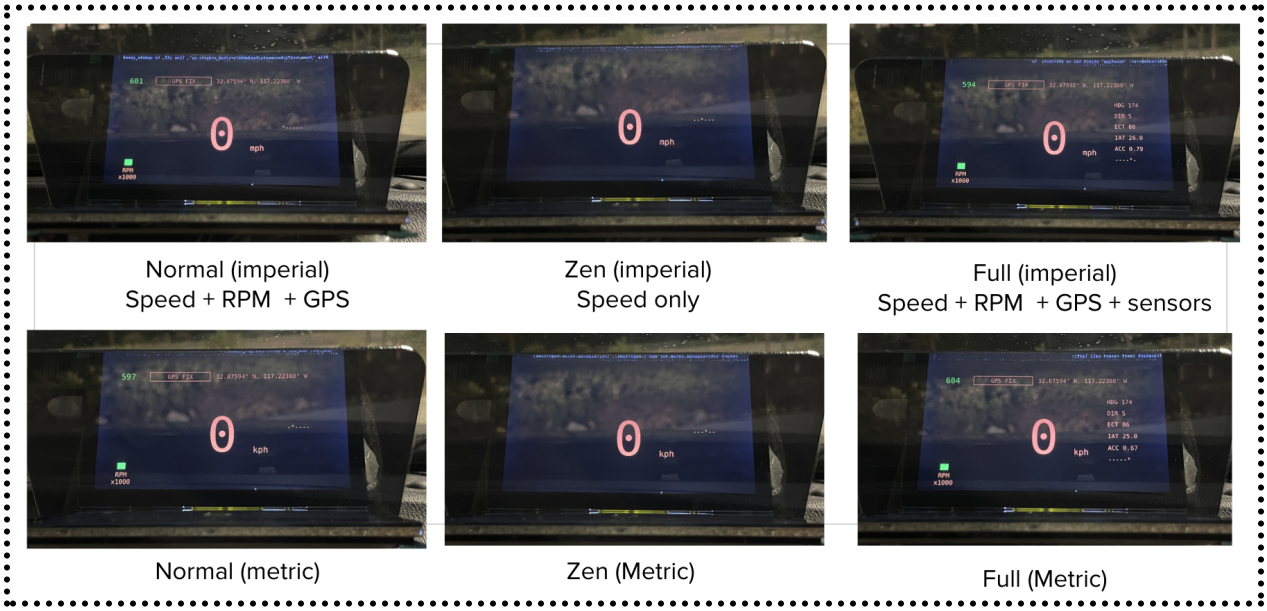

Figure 4: 6 display modes at night

Experiment

We performed a road test with the HUD set up as in the picture below. Designing a car accessory, we are most interested in its accuracy and response rate, and are less interested in other metrics such as power consumption, because sufficient power is provided by the generator and engine of the car. During the test, the OBDII scanner is plugged into the OBDII port located in front of the driver seat. The screen and

reflective glass is placed on the passenger side dashboard.

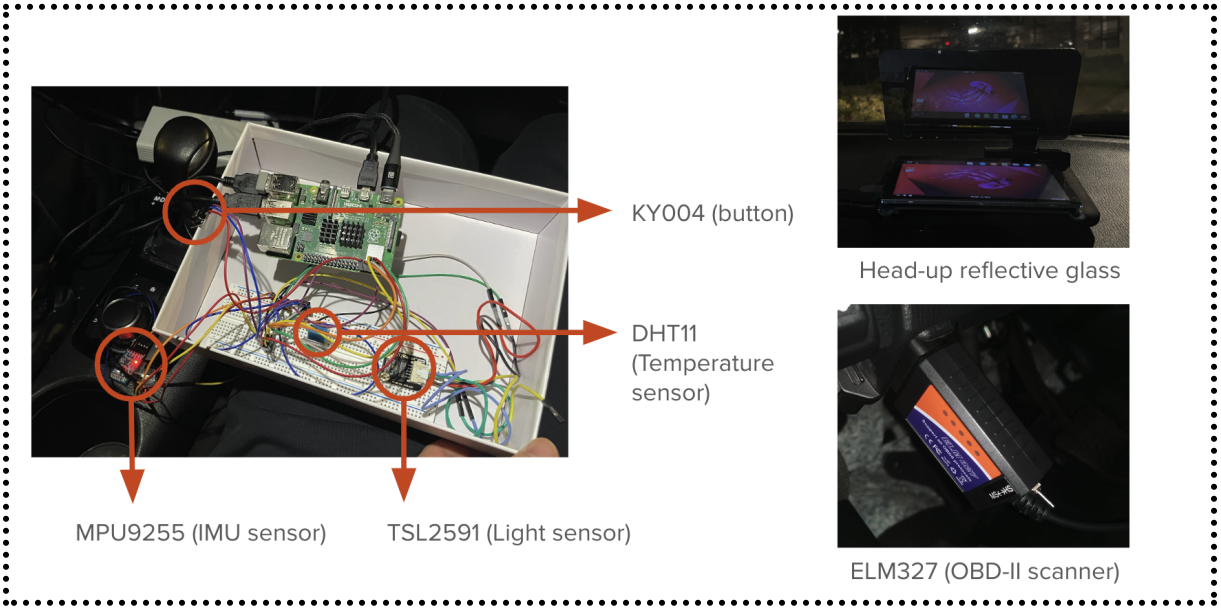

Figure 5: Road test system setup

Update timeliness

We found the updates are very timely, without noticeable lag with regard to the car's instrument. We purposefully revved the engine and observed that our RPM display follows closely to changes to the car's tachometer. The display mode changes response well to button presses, with a maximum delay of 1 second on the GPS widget. This is designed so to reduce CPU load due to frequent GPS refresh and is acceptable considering the purpose of the system. The color of the UI changes in response to the light conditions: turning the interior lights at night results in UI color changing immediately from amber to green.

Accuracy

We tested all sensor data and they appear coherent with other sources. The vehicle data, including RPM and speed, corresponds well to the car's instrument. The GPS information is comparable with that acquired from our smartphone. The IMU sensor, after calibration, provides accurate heading that agrees with the compass from the smartphone, so is the accelerometer.

In conclusion, the test results showed that our system is up to the expectations of the intended use. We consider the design of this process to be successful.

Conclusion

RPiHUD is a low-cost, customizable HUD module for cars based on Raspberry Pi. It collects real-time information using various sensors and displays it on a user-friendly UI. RPiHUD provides valuable information and allows drivers to customize the display according to their needs. It is more affordable, customizable, and extendable than existing aftermarket products. RPiHUD enables drivers to focus on critical information while driving, making it safer and more efficient. It is an open-sourced project that provides a code framework for future extensions. Overall, RPiHUD contributes to the development of low-cost, customizable, and open-source HUDs for cars.

Reference

1. Python-OBD library, https://python-obd.readthedocs.io/en/latest
2. Introduction to Raspberry Pi I2C, https://www.electronicwings.com/raspberry-pi/raspberry-pi-i2c
3. Introduction to IMU and hands-on with RPi and MPU 9250, https://medium.com/@niru5
4. Raspberry Pi GPIO Interrupts Tutorial, https://roboticsbackend.com/raspberry-pi-gpio-interrupts-tutorial/
5. DHT11 temperature and humidity sensor on Raspberry Pi, https://www.freva.com/dht11-temperature-and-humidity-sensor-on-raspberry-pi/
6. Using LSM303DLH for a tilt compensated electronic compass, https://www.sparkfun.com/datasheets/Sensors/Magneto/Tilt%20Compensated%20Compass.pdf
7. Adafruit TSL2591 High Dynamic Range Digital Light Sensor, https://learn.adafruit.com/adafruit-tsl2591/python-circuitpython
8. Interfacing with uBOX series GPS dongle https://github.com/tfeldmann/gpsdclient

Appendix

1. Source code repository: https://github.com/CPA872/MazdaHUD
2. Video demo link (actual road test):
   https://drive.google.com/file/d/1Qfygnc_uhxY4Bcl6cm7bTNLe-e_ucb9p/view?usp=sharing
   https://drive.google.com/file/d/1ObSp-n4MzTZoxrlGZrrSDWMyo3Xi4M4F/view?usp=sharing