ECE 492 - Final Report

Smart AVL (Automatic Vehicle Locator)

Group 9

Client
Lori Pecorilli, President
Latium Fleet Management Inc.
1312–10 Street, Nisku, Alberta, T9E 8K2
780-955-1088

Date
April 10, 2019

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

- Raspberry Pi 3 B+ provided by manufacturer Raspberry Pi Foundation
- PiCAN 2 CAN interface for Raspberry Pi provided by manufacturer SK Pang Electronics
- Python 3 provided by Python Software Foundation
- `python-can` library licensed under GNU Lesser General Public License V3 by Brian Thorne
- Raspberry Pi GPIO emulator provided by Roderick Vella for educational purposes[1]

Signatures:

_____ (Nicholas Hoskins)

_____ (Riley Dixon)

_____ (Adrian Schuldhaus)

[1] https://roderickvella.wordpress.com/2016/06/28/raspberry-pi-gpio-emulator/

Abstract

In this document, we outline the design specifications for a Smart Automatic Vehicle Locator (or AVL), which is able to capture certain information in real-time from a vehicle's communication channels. The device runs on a Raspberry Pi 3 B+, and is attached to the CAN bus, a communications standard found in many vehicles. The Raspberry Pi is able to broadcast the data it receives from the CAN bus to remote users, allowing these users to access data about monitored vehicles in real-time.

The first main objective of the project was to create a device that stores data from the CAN bus in real time. Secondly, we attached a number of peripherals to the device that recorded additional statistics. These devices included an accelerometer and a GPS module. After all our data is recorded locally, the data is wirelessly transmitted to a remote user. The position of our device also needs to be acquired via a GPS module. Finally, although the system works under ideal conditions, more time could be spent evaluating its performance under harsher conditions.

Table of Contents

Functional Requirements

The functional requirements for our project are as follows:

- Able to easily extract and transmit CAN bus messages via OBD-II port
- Able to operate under harsh weather conditions
- Able to analyze generated events and present them practically to users
- Able to support future expansion
- Target exterior dimensions should be kept as small as possible
- GPS module should provide precise point positioning

Design Alternatives

- Cyclone V Hard Processor system instead of Raspberry Pi 3 B+
    - Advantages:
        - Supported in the lab by the course
        - Onboard CAN bus controller
    - Disadvantages:
        - Higher power requirements
        - No onboard bluetooth or wifi interfaces
        - Larger form factor (bulkier)
        - Would require custom enclosure
        - Less flexibility in software options
        - Less code reusability
        - Fewer external software resources

Design & Description of Operation

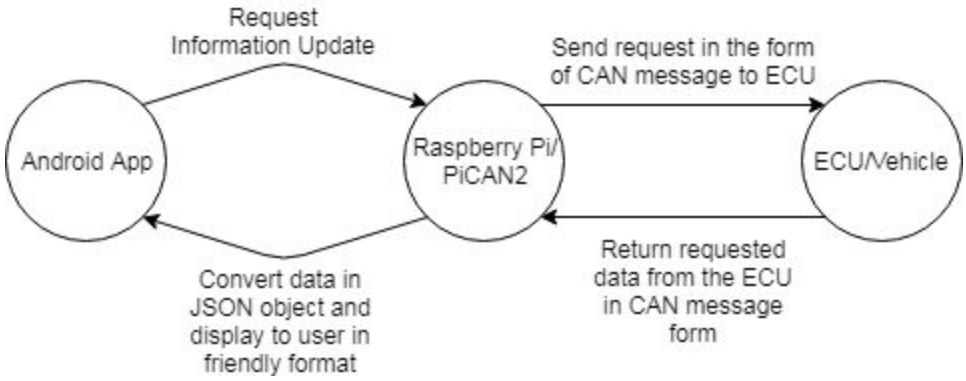
The operation of the device is fairly straightforward. The Raspberry Pi and Android devices should be paired beforehand to allow for easy connecting to start operation. Once the Android device is connected to the Raspberry Pi, communication between the two will occur automatically. The Raspberry Pi periodically polls data from the vehicle's ECU and serializes the data into a string before sending the data via Bluetooth to the Android phone. Once the Android phone receives the string, it parses the string and update the view that is presented to the user. Included data that will be shown to the user is: the current speed, the current RPM, the current uptime of the vehicle, if the check engine light is active, and the current latitude and longitude of the vehicle. The latitude and longitude coordinates are also displayed on a map for the user to track the vehicle.

Hardware Requirements, Bill of Materials, Reusable Design Units

- Raspberry Pi 3 B+ (link)
    - Description: Single board computer with GPIO interface and onboard bluetooth and wifi capabilities. Is a reusable design unit.
    - Cost: CAD$58.92
    - Supplier: https://www.digikey.ca/product-detail/en/raspberry-pi/RASPBERRY-PI-3-MODEL-B/1690-1025-ND/8571724
    - Datasheet: See “Specifications” pages [1], [7] in References for datasheets.
- Pi-CAN 2 interface for Raspberry Pi
    - Description: CAN bus interface board for the Raspberry Pi series of boards.
    - Cost: USD$49.95
    - Supplier: https://copperhilltech.com/pican-2-can-interface-for-raspberry-pi-2-3/
    - Datasheet: See reference [2] for datasheet.
- Plastic Enclosure for Pi-CAN 2 and Raspberry Pi 3
    - Description: Plastic housing for a Raspberry Pi 3 with attached Pi-CAN 2 board.
    - Cost: USD$25.95
    - Supplier: https://copperhilltech.com/plastic-enclosure-for-pican2-and-raspberry-pi-2-3/
- OBD-II to DB-9 connector cable
    - Description: 100 cm 9-Pin DB9 female to 16-pin male OBD-II
    - Cost: CAD$14.03
    - Supplier: https://www.amazon.ca/OBD2-16Pin-Serial-Adapter-Cable/dp/B007UKTW8E
- 16 GB micro-SD Card
    - Description: 16 GB micro-SD card for storage on Raspberry Pi. Is a reusable design unit.
    - Cost: CAD$5.00
- Adafruit Ultimate GPS Breakout
    - Description: GPS module for the Raspberry Pi. Is a reusable design unit.
    - Cost: CAD$39.95
    - Supplier: https://www.adafruit.com/product/746
    - Datasheets: See Reference [3] for datasheet.
- Adafruit USB to TTL Serial Cable
    - Description: Connects to the Raspberry Pi’s UART from a USB port.
    - Cost: CAD$14.44
    - Supplier: https://www.digikey.ca/product-detail/en/adafruit-industries-llc/954/1528-2128-ND/7064488

Total cost of preferred parts is approximately CAD$255 after currency conversion.

I/O Signals

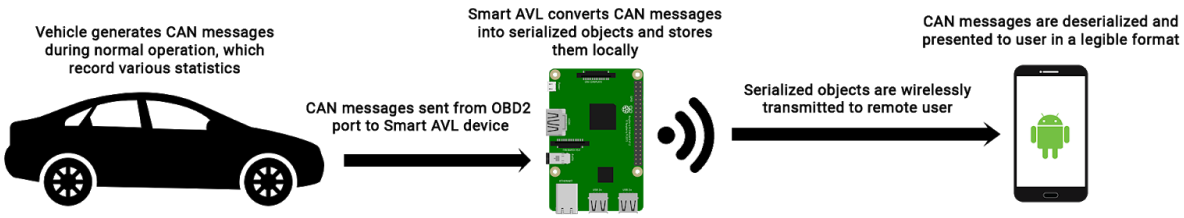


The input/output of the system can be broken down into 4 main components. The Raspberry Pi will push periodic updates to the Android phone over Bluetooth. This update is done every 0.5 seconds. For each update, the Pi sends several OBD-II requests to the ECU, with each request formed as a single CAN message. The Pi then receives the OBD-II responses back, and translates these responses into a single JSON string. This JSON string is then transmitted over Bluetooth to the Android phone, where it is parsed and displayed in a user-friendly format.

Background Research

The CAN Bus messaging service is a standard that was originally created by Bosch and is now in use in a variety of vehicles. This messaging service is used in land vehicles, airplanes, and also some military vehicles. Using the university library services, we were able to access a copy of the older versions of the ISO 11898 standard to use as reference material when developing the code to read the messages being sent from the vehicle. [4]

Software Design



There are two major components to the software involved in the Smart AVL project:

1. The first part of the software was designed to run on the Raspberry Pi. Its function is to collect the CAN messages generated by a vehicle, and to obtain geographic location data from the GPS. These messages are then stored locally and serialized into a JSON string to be sent to the Android phone.

2. The second part of the software is an Android application which wirelessly receives the serialized CAN messages, and geographic location data from the Raspberry Pi. The application then deserializes these messages and presents them to the user in a legible format.

Test Plan & Testbenches

Software

- Python 3's built in `unittest` module was be used to ensure proper functionality of the software.
- Messages from the CAN bus were emulated with a virtual CAN busi when hardware testing is not convenient or possible.
- Use of the Android Studio emulator and testing on a physical Android device can be employed to test the functionality and user interface of the Android components of the system.
- A simple, two-way communication testing Android app, along with the accompanying server on the Raspberry Pi, will be constructed to test and build Bluetooth communication, before being integrated into the SmartAVL system.

Hardware

- The PiCAN board, which attaches to the Raspberry Pi, contains several pins intended for diagnostic purposes. When troubleshooting the project, these pins were probed with a multimeter to ensure the CAN bus was behaving as expected.
- Additionally, many field tests of the device were performed by connecting the Smart AVL to an actual vehicle.
- The GPS was tested in isolation before being integrated into the SmartAVL system. This included its own software testbench and was contained in a separate thread to isolate any potential issues.

Safety

- According to the Raspberry Pi Foundation, the recommended voltage for the Raspberry Pi is 5 V. The expected range of voltage operation for the Raspberry Pi is roughly 3.3 - 6.0 V [5]. The vehicle supplies the power from a USB port located from the cabin of the vehicle.
    - One of the objectives of the design proposal is that the device should be capable of operating between 10V and 24V. If no USB port is present, a cigarette car receptacle to USB port converter can be used to provide 5V power to the Raspberry Pi.

- One of the objectives outlined in the design proposal is that the Smart AVL should be able to operate under harsh weather conditions. The Raspberry Pi is an ideal choice to meet this criterion, as it has been tested to work at temperatures below -70^{o}C. [6]

Regulatory and Society

- Given the right set of circumstances, a malicious actor could remotely connect to the Smart AVL and send arbitrary messages to the vehicle through the CAN bus. This event could have disastrous consequences.
    - At this point in time, the Smart AVL device does not read any data from the Android app. As a result, an attacker would need to directly gain access to the Raspberry Pi. To mitigate the risk of tampering, the Raspberry Pi could disallow any non-Bluetooth connections.

Environmental Impact

A goal in the design of the project was to minimize the amount of hazardous materials used in the project. If the project were to be lost or discarded, the amount of electronic waste created would be small. However, the possibility of the project being accidentally lost is small, given that it will be inside of, and attached to, a vehicle during its normal operation.

The Raspberry Pi and the Adafruit Breakout GPS are both RoHS (Restriction of Hazardous Substances) compliant. No information was found pertaining to the RoHS compliance of the PiCAN 2 interface. Additionally, the Raspberry Pi, GPS, and microSD card are all reusable design units that can be repurposed in other projects. Therefore, we conclude that the environmental impact of the SmartAVL system is low.

Hardware Documentation

Overall System

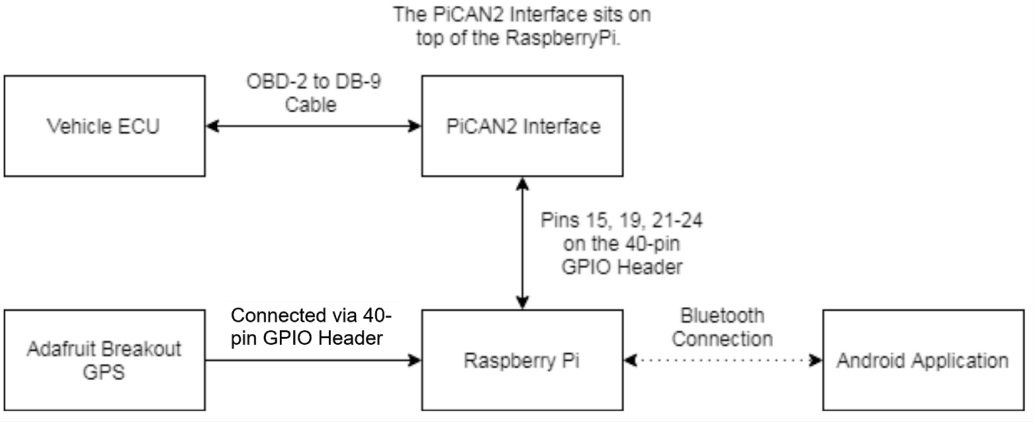


Figure 1: An overview of the various system components and how they are connected to one another.

The overall hardware is straightforward in how it is all connected together at a high level. With the Raspberry Pi at the center of the project, all communication is done through it by some means. The PiCAN2 interface sits on top of the Raspberry Pi as a HAT (Hardware Attached on Top) and as such connects directly into the 40-pin GPIO header, even though it only uses 6 of the GPIO pins. The PiCAN2 then connects to the ECU of the vehicle through the OBD-2 port on the vehicle. The Adafruit GPS connects to the Raspberry Pi's UART over a USB connection. Lastly the Raspberry Pi and the Android application communicate via a Bluetooth connection.

Adafruit GPS

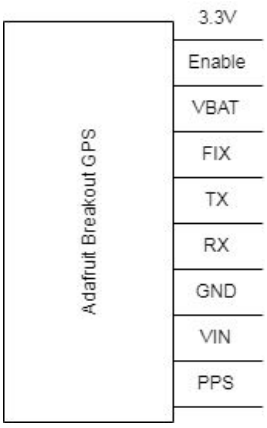


Figure 2: The pin layout for the Adafruit Breakout GPS

Shown to the left is the layout for the Adafruit Breakout GPS. Since we are only using the UART to read data from the GPS, the only pins in use are: VIN (3.3V), GND, RX, and the TX pins. The other pins would be used if the GPS is to be connected via I2C or SPI.

PiCAN 2

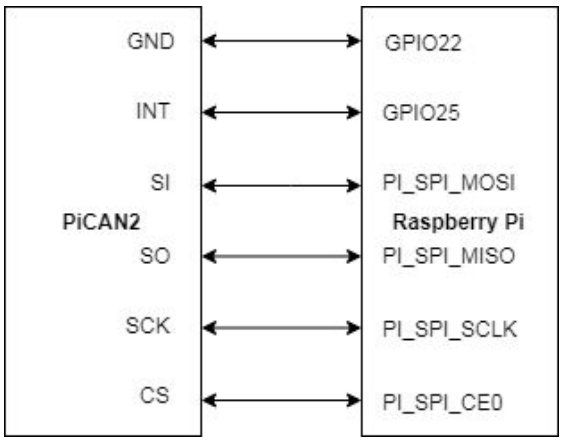


Figure 3: The pin layout and connection of the PiCAN 2 interface with the Raspberry Pi

As mentioned previously, the PiCAN 2 is a HAT and as such sits directly on top of the Raspberry Pi. It also connects to the entire GPIO header even though only 6 of the pins are used. The interface controls the Raspberry Pi's master-slave interface as it transfers a large amount of data to and from the vehicle's ECU.

The full data schematic of the PiCAN 2 board (including microcontrollers) can be viewed on the supplier's website under the "Documents" header. [6]

Source Code

All project source code is maintained in a GitHub repository. This includes a directory for the code running on the Raspberry Pi, as well as another directory containing the source code for the Android app. The Raspberry Pi components are primarily written in Python, while the Android app is written in Java.

The Android App is completed for the scope of the project. It displays some core vehicle operating information, the location on a Google Maps activity, and a data log in a simple, user-friendly manner. As most of the communication handling happens on the Raspberry Pi, the app consists mostly of presenting the data to the user.
The Raspberry Pi has the program that interfaces with the Android phone over threaded Bluetooth sending/receiving. The program also handles the sending/receiving of messages to the CAN bus onboard the attached vehicle, as well as the interface with the GPS chip to obtain the real-time location of the device. This code running on the Pi meets all its requirements laid out as the scope of this project.

The GitHub repository for the project can be found at: https://github.com/aschuldhaus/SmartAVL

Conclusions

The project is completed and able to meet virtually all the requirements of our design, specified above. When connected to the vehicle, the prototype is able to retrieve speed, rpm, engine runtime, and check-engine light status from the CAN bus. The GPS chip provides the longitude and latitude of the unit. This information is then relayed and displayed on the Android

companion app connected to the device over Bluetooth. This information can be easily interpreted by the end-user.
The device maintains a minimal form factor, while protected in the PiCAN enclosure. Ideally, the GPS chip would have also been included inside this enclosure, but due to issues accessing the extra GPIO pins while using the PiCAN board, it is connected via an external USB port and thus sits outside the enclosure.
The only requirement we were unable to test was its ability to to operate under harsh weather conditions. This was due to lack of time and access to a suitable testing environment. However, we have no reason to believe that this would fail in testing, as the components we used are rated for sufficiently harsh operating conditions.

Future Work

There is a lot that can be done beyond the scope of this project to improve or expand on the Smart AVL device. A couple ideas are listed here:

- Make device interface over cellular networks rather than Bluetooth to retrieve real-time vehicle data from nearly anywhere in the world.
- Have the device report to a web-server for logging and tracking of an entire fleet of vehicles simultaneously.
- Power the device from the vehicle's OBD-II port instead of using a USB power cable.
- Improve the user interface features to include more vehicle information, track/plot value history, or provide more intelligent and useful insights into this data.

References

- Raspberry Pi 3 B+
    - Specifications: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf
    - Mechanical Drawing: https://github.com/raspberrypi/documentation/raw/master/hardware/raspberrypi/mechanical/rpi_MECH_3bplus.pdf
    - Schematic Diagram: https://github.com/raspberrypi/documentation/raw/master/hardware/raspberrypi/mechanical/rpi_MECH_3bplus.pdf
- PiCAN 2
    - User Guide: http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2UG13.pdf
    - Schematic: http://skpang.co.uk/catalog/images/raspberrypi/pi_2/pican2_rev_B.pdf
- Adafruit Ultimate GPS Breakout
    - Overview Guide: https://learn.adafruit.com/adafruit-ultimate-gps

Appendix

Start-Up Manual

Setting up Raspbian

Raspbian is the recommended operating system for the Raspberry Pi, and as such, Raspbian is the operating system that is supported by our SmartAVL project. It can be installed as follows:

1) Connect a microSD card with a storage capacity of 16 gigabytes or more to your computer.
2) Install NOOBS on the microSD card.
3) Plug the microSD card into the Raspberry Pi, then connect a mouse, keyboard, and display to the Raspberry Pi.
4) Power up the Raspberry Pi and follow the onscreen prompts to install Raspbian.

Setting up PiCAN

The PiCAN board is an attachment to the Raspberry Pi. It can be installed by following this guide.[7] When soldering the board, ensure that the termination resistor is activated, and that the DB-9 connector is configured for an OBD-II cable.

Setting up Python-CAN

The Smart AVL device uses Python, and relies on the Python-CAN library, which is not installed by default. It can be installed and configured as follows:

1) Open a new terminal window on the Raspberry Pi.
2) Execute the command **`pip3 install python-can`**
3) Execute the command **`nano ~/can.conf`**
    a. This command opens an editor window. Paste the following into the editor window, then save your changes:

```
[default]
interface = socketcan
channel   = can0
bitrate   = 500000
```

Setting up Bluetooth

There are a number of steps to properly setup Bluetooth capabilities for the Raspberry Pi. This guide will explain how to setup the official Raspberry Pi Bluetooth library for python. We also strongly recommend setting up the GUI for managing Bluetooth devices and pairing with the Raspberry Pi. All of these instructions should be run on the Raspberry Pi.

1) Open up a new terminal window.

2) Execute **`sudo apt-get install bluetooth bluez blueman pi-bluetooth python-bluez`**. This will install the GUI as well as the underlying Bluetooth libraries that we will be using.
3) Reboot the Raspberry Pi.
4) Ensure Bluetooth has been started with **`systemctl status Bluetooth`**. If Bluetooth is not active, turn it on with **`sudo systemctl start Bluetooth`**.
5) The next following steps 6-12 detail how to resolve a permission error if an “errno 13: Permission Error” is encountered during operation. We recommend this step is done now as a precaution. Credit to “dlech” for solving this error.[8]
6) In **`/etc/systemd/system/dbus-org.bluez.service`** edit the “ExecStart” line by appending the flag **`“--compat”`** on that line.
7) Add the “pi” user to the Bluetooth group with **`sudo usermod -G Bluetooth –a pi`**. If this fails, add the Bluetooth group with **`sudo newgrp Bluetooth`** and try the above command again.
8) The next three steps are for having these changes persist on boot.
9) Create a new file **`/etc/systemd/system/var-run-sdp.path`** and add the following text:
   [Unit]
   Description=Monitor /var/run/sdp
   [Install]
   WantedBy=dbus-org.bluez.service
   [Path]
   PathExists=/var/run/sdp
   Unit=var-run-sdp.service
10) Create a new file **`/etc/systemd/system/var-run-sdp.service`** and add the following text:
    [Unit]
    Description=Set permission of /var/run/sdp
    [Install]
    RequiredBy=var-run-sdp.path
    [Service]
    Type=simple
    ExecStart=/bin/chgrp bluetooth /var/run/sdp

11) Start the new daemon with the following commands:
    a. Run **`sudo systemctl daemon-reload`**.
    b. Run **`sudo systemctl enable var-run-sdp.path`**.
    c. Run **`sudo systemctl enable var-run-sdp.service`**.
    d. Run **`sudo systemctl start var-run-sdp.path`**.
12) Reboot the Raspberry Pi and Bluetooth will now be setup.

Setting up the Adafruit Breakout GPS

To take advantage of Adafruit’s libraries that they package with their hardware peripherals, multiple steps will have to be taken to configure the Raspberry Pi. This involves setting up I2C,

SPI and the GPIO pins and installing Adafruit's CircuitPython library. This guide is taken from the Adafruit CircuitPython tutorial.[9]

1) Enable I2C support on the Raspberry Pi.
    a. Install the I2C-Tools utility with **`sudo apt-get install -y python-smbus`** and **`sudo apt-get install -y i2c-tools`**.
    b. Configure the Raspbian Kernel by opening up the configuration menu with **`sudo raspi-config`**.
    c. In the Configuration Tool, navigate to Interfacing Options > I2C. Select yes to enable the "ARM I2C interface" and yes for the "I2C kernel to be loaded by default".
    d. Reboot the Raspberry Pi. Test by running the command **`sudo 12cdetect –y 1`** which should show an address map of active I2C addresses in use. On a new installation it is normal for this to be completely blank.
2) Enable SPI support on the Raspberry Pi.
    a. Configure the Raspbian Kernel by opening up the configuration menu with **`sudo raspi-config`**.
    b. In the Configuration Tool, navigate to Interfacing Option > SPI. Select yes to enable the "SPI interface to be enabled".
    c. Reboot the Raspberry Pi. Test by running the command **`ls -l /dev/spidev*`**, there should be one or two devices shown.
3) Now it is time to configure the device based on how the GPS is being connected to the Raspberry Pi
    a. If connecting directly using GPIO pins follow the following instructions.
        i. Configure the Raspbian Kernel by opening up the configuration menu with **`sudo raspi-config`**.
        ii. In the Configuration Tool, navigate to Interfacing Option > Serial. Select no to "enabling a login shell over serial" and yes to having "the serial port hardware to be enabled".
        iii. Reboot the Raspberry Pi. Ensure the serial monitor has been enabled by confirming that the device at path **`/dev/ttyS0`** is present.
    b. Else if connecting via USB use the following instructions.
        i. Thanks to "Sutekh" from the Ubuntu forums for their guide on setting up udev rules for USB devices.[10] If more devices are to be setup, the developer may find it handy to setup a udev rule to allow each device to have a specific socket just for it.
        ii. Make sure the pi user is part of the user group "dialout". If it is not, run **`sudo adduser pi dialout`**.
        iii. Add the pi user to the "dialout" group by **`sudo usermod –a –G dialout pi`**.
        iv. The USB socket has been configured to connect to the device from the device ID rather than the USB socket itself. This will mitigate potential issues if more USB devices are connected. There is a limitation however as there cannot be more than one of the USB-to-TTL convertors connected at the same time if done through this method.

4) Run the following commands to install the next few packages:
    a. Run **`pip3 install RPI.GPIO`**.
    b. Run **`pip3 install adafruit-blinka`**.
    c. Run **`pip3 install adafruit-circuitpython-gps`**.
5) At this point the GPS is ready to be used. If the connection type changes, the python file at which the GPS was initialized from needs to be updated as well. Failure to do so could prevent the GPS device from being initialized and read from properly.

Setting up VNC Viewer

The Raspbian OS comes preinstalled with VNC Viewer, it just needs to be setup. This will allow a developer or IT to access the Raspberry Pi wirelessly without the need to have an external monitor, keyboard, and mouse. Instructions based off of.[11]

1. Install VNC Connect by running **`sudo apt-get install realvnc-server realvnc-viewer`**.
2. In the Raspberry Pi menu, go to **`Menu > Preferences > Raspberry Pi Configuration > Interfaces`**. Make sure that VNC is enabled.
3. Find the private IP address of the Raspberry Pi. This can be done either through SSH while connected to the Raspberry Pi, or if privileged try to find the Raspberry Pi on the network. Use this IP address in the VNC Viewer application of the computer trying to remote into VNC.
4. At time of publication, the sole-SmartAVL Raspberry Pi has a password of “ece492” to connect to it. This is using a VNC password instead of the UNIX login to login into the computer. We highly recommend that this password is changed should this project be used outside of a development environment. If on a new installation, the default username is **`pi`** and default password is **`raspberry`**. The login can be changed in the VNC menu of the Raspberry Pi by opening **`Menu > Options > Security`** and changing the Authentication type and settings there.

Datasheets

- Raspberry Pi 3 B+
    - Datasheet [11]
- PiCAN 2
    - Datasheet [2]
- Adafruit Ultimate GPS Breakout
    - Datasheet [3]

References

[1]"Raspberry Pi 3 Model B+ - Raspberry Pi", *Raspberry Pi*, 2019. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/. [Accessed: 08- Apr- 2019].

[2]”PiCAN2 Interface Datasheet”, *S.K. Pang Electronics*, 2019. [Online]. Available: http://skpang.co.uk/catalog/images/raspberrypi/pi_2/PICAN2DSB.pdf [Accessed: 08-Apr-2019].

[3]"FGPMMOpA6h GPS Standalone Module Datasheet", *GlobalTop Technology Inc.*, 2011. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf. [Accessed: 08- Apr- 2019].

[4]"CAN in Automation (CiA): History of the CAN technology", *Can-cia.org*, 2019. [Online]. Available: https://www.can-cia.org/can-knowledge/can/can-history/. [Accessed: 08- Apr- 2019].

[4]Milliways, "Power requirements of the Pi", *Raspberry Pi Stack Exchange*, 2019. [Online]. Available: https://raspberrypi.stackexchange.com/questions/51615/raspberry-pi-power-limitations. [Accessed: 08- Apr- 2019].

[5]M. Humphries, "Raspberry Pi proven to be stable when submerged in liquid nitrogen - Geek.com", *Geek.com*, 2019. [Online]. Available: https://www.geek.com/chips/raspberry-pi-proven-to-be-stable-when-submerged-in-liquid-nitrogen-1555235/. [Accessed: 08- Apr- 2019].

[6]"PiCAN 2 - CAN Interface for Raspberry Pi 2/3", *Copperhill Technologies*, 2019. [Online]. Available: https://copperhilltech.com/pican-2-can-interface-for-raspberry-pi-2-3/. [Accessed: 08- Apr- 2019].

[7]"PiCAN2 User Guide", *Copperhilltech.com*, 2019. [Online]. Available: https://copperhilltech.com/pican2-controller-area-network-can-interface-for-raspberry-pi/. [Accessed: 08- Apr- 2019].

[8]”dlech”, "Running programs with root permission (or resolving user bluetooth permissions) · Issue #274 · ev3dev/ev3dev", *GitHub*, 2019. [Online]. Available: https://github.com/ev3dev/ev3dev/issues/274. [Accessed: 08- Apr- 2019].

[9]"Installing CircuitPython Libraries on Raspberry Pi", *Learn.adafruit.com*, 2019. [Online]. Available: https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi. [Accessed: 08- Apr- 2019].

[10]"Create your own udev rules to control removable devices", *Ubuntuforums.org*, 2019. [Online]. Available: https://ubuntuforums.org/showthread.php?t=168221. [Accessed: 10- Apr- 2019].

[11]"VNC (Virtual Network Computing) - Raspberry Pi Documentation", *Raspberrypi.org*, 2019. [Online]. Available: https://www.raspberrypi.org/documentation/remote-access/vnc/. [Accessed: 08- Apr- 2019].

[12]"Raspberry-Pi-Model-Bplus-Product-Brief.pdf", *Raspberry Pi*, 2019. [Online]. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf. [Accessed: 08- Apr- 2019].

Declaration of Original Content

The design elements of this project and report are entirely the original work of the authors and have not been submitted for credit in any other course except as follows:

- Raspberry Pi 3 B+ provided by manufacturer Raspberry Pi Foundation
- PiCAN 2 CAN interface for Raspberry Pi provided by manufacturer SK Pang Electronics
- Python 3 provided by Python Software Foundation
- `python-can` library licensed under GNU Lesser General Public License V3 by Brian Thorne
- Raspberry Pi GPIO emulator provided by Roderick Vella for educational purposes[1]

Signatures:

Nicholas Hoskins (Nicholas Hoskins)

Riley Dixon (Riley Dixon)

Adrian Schuldhaus (Adrian Schuldhaus)

[1] https://roderickvella.wordpress.com/2016/06/28/raspberry-pi-gpio-emulator/