SENSATION

Report

Yatharth Thakkar

Website address: `https://github.com/ThaatGuy/Sensation`

SENSATION

Yatharth Thakkar*

December 3, 2022

Abstract

In this project we aim to develop a wearable system to guide the blind around streets, helping them avoid obstacles and get from one place to another. This report outlines a system developed that is strapped onto the users chest. Containing a Jetson Nano that processes the information coming from a camera and a GNSS module. It also provides haptic and audio feedback to guide the user through another extension board that has been developed.
To keep all the components together, a main frame made of acrylic is laser such that the cases and adaptors that are 3D printed can hold the Jetson Nano, battery pack, GNSS module and the camera. Spaces have also been cut such that straps can be attached to go around the chest and neck of the user.
The Jetson Nano uses a camera to capture the the scene and make sense of it to guide the user. To do this, each frame captured by the camera is segmented using cityscapes primarily segmenting out the road and footpaths. Mobilnet-ssd is also used to detect objects such that the user can avoid them. After the implementation the system is able to detect objects well, however is not able to segment the road and the footpath very well as cityscapes was mainly designed to detect the road. A secondary PCB board has also been developed to control 8 vibration motors that attaches to the Jetson Nano, however with the current demo only 3 of them are used and are able to guide the user to move straight, right and left.
To navigate from one location to another the system uses a GNSS module to obtain the users location and the Open-Route-Services API to determine the best route and provide some prior information about the route to optimize the system performance.

*yatharth.thakkar@fau.de

Though the system developed contains a number of problems that are described throughout this report, it serves as good a starting point for further development.

Contents

1 Setup **4**
1.1 Components Used 4
1.2 Casing 5
1.3 Jetson Nano 6
1.4 Navigation and Haptic Belt 6
1.4.1 GNSS Module 7
1.4.2 Vibration motors 8
1.4.3 Power buttons 8
1.4.4 Routing 9
1.4.5 Text To Speech 9

2 Haptic belt **9**
2.1 PCB Design 9
2.2 Controlling the vibration motors 10
2.3 Adding new motors 14

3 Image Segmentation and object detection **14**

4 Navigation **15**

5 Conclusion **19**

1 Setup

This section goes through how the system can be set up. Firstly outlining how the casing used to hold all the components together has been developed and worn by the user. Followed by how the Jetson Nano is set up to perform the segmentation and object detection and how it interacts with the extension board to control the haptic belt. Finally it outlines how to set up the navigation with the GNSS module and the Open-Route-Services API.

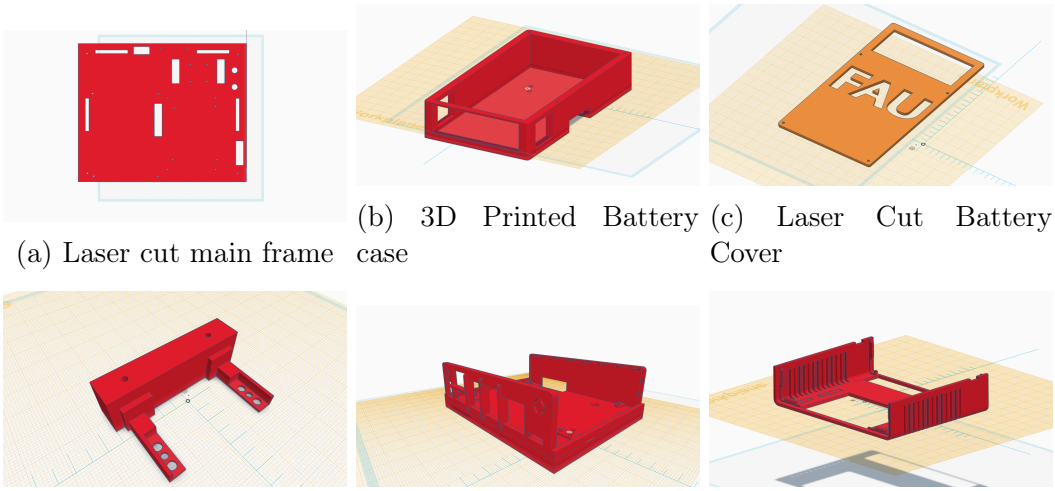
1.1 Components Used

1. Nvidia Jetson Nano developer kit 4GB
2. Intel 8265 Wireless card with antenna
3. 256GB Micro SD card
4. 5V 3A Powerbank 20,000 mAh (INIU BI-B5 20,000 mAh Powerbank)
5. Adafruit PA1010D
6. Logitech C270 camera
7. 3 x 100nF capacitors
8. 1 x 1uF capacitor
9. 1 x 10uF cpacitor
10. 8 x schottky diodes
11. 1 x Bosh BH160b Accelerometer/ Gyroscope IMU
12. 1 x Bosch BMM150 Magnetometer
13. 1 x PCA9685PW PWM controller
14. 7 x 10kΩ resistors
15. 4 x 4.7kΩ resistors
16. 1 x 1kΩ resistor
17. 9 x 2 pin terminal blocks

18. 2 x push buttons
19. 1 x Gopro Chest Strap Mount (Manufacturer: Tseasoon)

1.2 Casing

Tinker CAD was used to develop the components used for this system since it is online and free to use with a simple interface. Figure 1 shows the separate components designed to hold all the components together.
The design of the 5mm thick acrylic main frame can be seen in figure 1a serving as the backbone to this device. The small holes in this frame are 2.8mm in diameter and are used to attach the other components using M3 screws. The wide rectangular cuts have been developed so that one can hide all the wiring behind the frame and it does not cause an obstruction. The thin rectangles have been developed to sew on the belt that is strapped around the users chest and neck in a way that the system does not shake too much while walking. The larger circular holes in the frame are used to hold the buttons for turning the Jetson Nano on and off (top) and executing the guidance and navigation system (bottom).
Figure 1b shows the design for the battery case consisting of one large rectangular hole at the front to insert the cables and two rectangular holes in the sides to activate the battery pack. There is also a gap on the bottom edge allowing us to access the bottom of the main frame and give some space to attach cables to the pins at the back of the Jetson Nano. Figure 1c shows the cover for the the battery that has been cut onto a 1mm thick acrylic sheet with a small window such that the battery status is visible.
Figure 1d shows the design for the camera mount that is 3D printed. To mount this onto the camera, the camera is first disassembled and the two legs of the mount are slid into the slots where the original mount was placed and screwed on. After this, it is screwed into the holes at the top part of the main frame using M3 screws.
To hold the Jetson nano, the case shown in Figure 1e is 3D printed. It contains slots so that cables can be plugged in at the front and one at the back to access the power pins. This is held onto the main frame using four M3 screws. To cover the Jetson Nano the cover shown in Figure 1f is 3D printed and is clipped onto the case. It contains a slot for the heat sink and ventilation, a slot to connect the vibration motors and slots that can be cut out to access pins such as the GPIO pins on the Jetson Nano. After putting everything together the device should look like that shown in figure 2.



(a) Laser cut main frame

(b) 3D Printed Battery case

(c) Laser Cut Battery Cover

(d) 3D Printed Camera Mount

(e) 3D printed Jetson Nano Case

(f) 3D printed Jetson Nano Cover

Figure 1: Individual components of the SENSATION case

1.3 Jetson Nano

The tutorial found in Alarcon (2020) can be used to set up the Jetson Nano and perform object detection in real time. For this application only Moblienet-SSD and FCN-ResNet18-Cityscapes are required when selecting the pre-trained networks, however other networks may also be downloaded. Unlike the tutorial when selecting the PyTourch version make sure it is for Python 3 to insure everything works correctly.
If there are problems when installing Numpy the following commands may be usefull:

```
$ sudo -H python3.7 -m pip uninstall numpy
$ sudo apt purge python3-numpy
$ sudo -H python3.7 -m pip install --upgrade pip
$ sudo -H python3.7 -m pip install numpy
```

1.4 Navigation and Haptic Belt

Navigation in this device is performed using a GNSS module, three vibration motors and speech. All requiring a set of libraries and some changes to the settings:

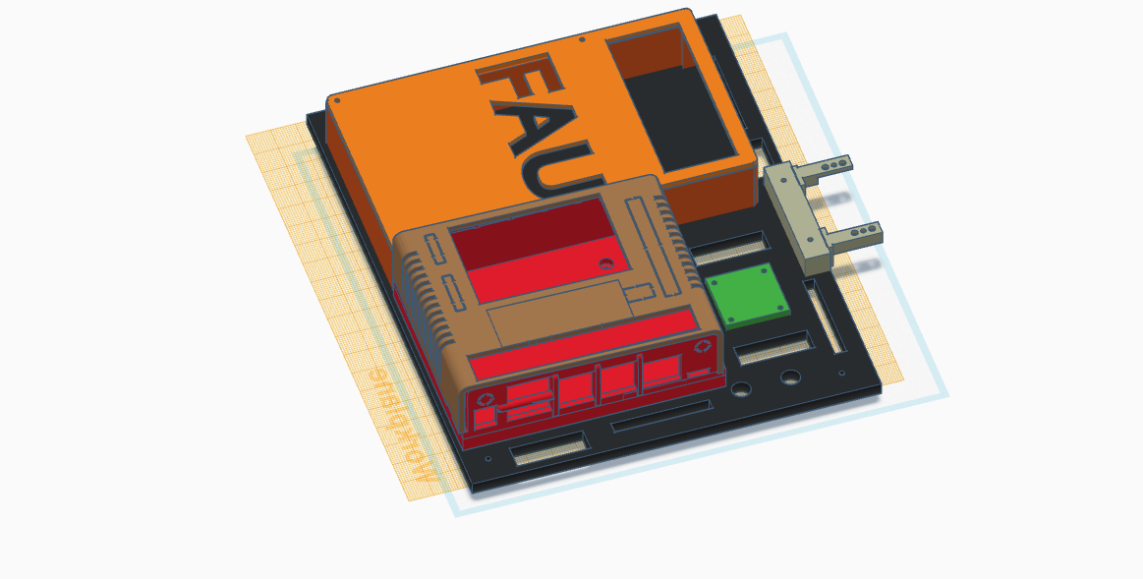


Figure 2: Final Layout of the SENSATION case

1.4.1 GNSS Module

To access location information from the GNSS module it is important to know which port in the Jetson Nano the information is being sent to. This may change and cause the system to fail. Thus it is important to give it a fixed name such that one can get information from it regardless of which port it is connected to. To do this we:

1. Find out what's on ttyUSB:

```
$ dmesg | grep ttyUSB
```

2. List all attributes of the device (if device is ttyUSB1 then x = 1) and note the idVendor and idProduct:

```
$ udevadm info --name=/dev/ttyUSBx --attribute-walk
```

3. Create a file /etc/udev/rules.d/99-usb-serial.rules and enter the following with "1234" and "5678" with the id's found in step 2 :

```
SUBSYSTEM=="tty", ATTRS{idVendor}=="1234", ATTRS{idProduct}=="5678",
SYMLINK+="GPS"
```

4. Load the new rule:

```
$ sudo udevadm trigger
```

5. Verify the changes (the output will tell us which ttyUSB our device name points to):

```
$ ls -l /dev/GPS
```

6. Give access to /dev/GPS without root:

```
$ sudo usermod -a -G dialout $USER
```

7. Log out and back in for the changes to take place.

8. Install the circuit python gps library:

```
$ pip3 install adafruit-circuitpython-gps
```

1.4.2 Vibration motors

To operate the haptic belt, the PCB described under the section haptic belt is placed on top of the Jetson Nano J41 Header such that all the pins are connected to the header. The Jetson Nano cover is closed and the motors are connected to the terminal corresponding to the channel we want to send our commands to. The Jetson Nano sends commands to the PWM chip on the PCB using I2C to drive the motors. To make this easy the pca9685 library needs to be installed using the command:

```
$ sudo pip3 install adafruit-circuitpython-pca9685
```

The vibration motors are attached to the straps going around the chest, one placed behind the main frame (guiding the user to move forward) and two are placed on the two plastic clips that can slide along the belt such that the placement of the right and left motors can be adjusted for comfort and sensitivity to the vibrations .

1.4.3 Power buttons

To turn the Jetson Nano on using a power button the "AUTO ON" and "DISABLE" pins on the J14 connector need to be shorted and the two leads of the power button

need to be connected to the "PWR BTN" and "GND" pins. With this configuration one can turn the Jetson Nano on with a button press and on the second button press it would turn off after 30 seconds. A second button can be used to start and stop the program to guide the user. The leads of this button are connected to the block terminal labeled U8 on the extension board shown in figures 5 and 6 which is described in a later section about the haptic belt.

1.4.4 Routing

To enable routing using Open Route Services an API key is required from the Open Route Service website and the library needs to be installed using:

```
$ pip3 install openrouteservice
```

1.4.5 Text To Speech

```
$ pip3 install pyttsx3
```

2 Haptic belt

The haptic belt is used to provide feedback to the user while they walk on the streets, helping them avoid obstacles and taking them from their current location to the destination.

2.1 PCB Design

To enable this an extension board is developed that can be placed on top of the Jetson Nano interfacing with the Jetson Nano J41 Header which is illustrated in Figures 3 and 4. Here, I2C pins 4 and 5 of the Jetson Nano are connected to the PCA9685PW to control how fast the motor vibrates and also which motor gets turned on. The default address for the PCA9685 is set to 0x40, but can be changed by shorting solder joints 1 to 6 (SJ1-6) labeled in Figure 3. After a command is sent to the Jetson Nano to engage a motor a PWM signal is generated by the PCA9685PW to engage the motor. During the initial testing this signal was sent to the gate of a transistor such that the motors are driven by an external power source to decrease the load on the Jetson Nano. However it was found that whenever the motors are engaged

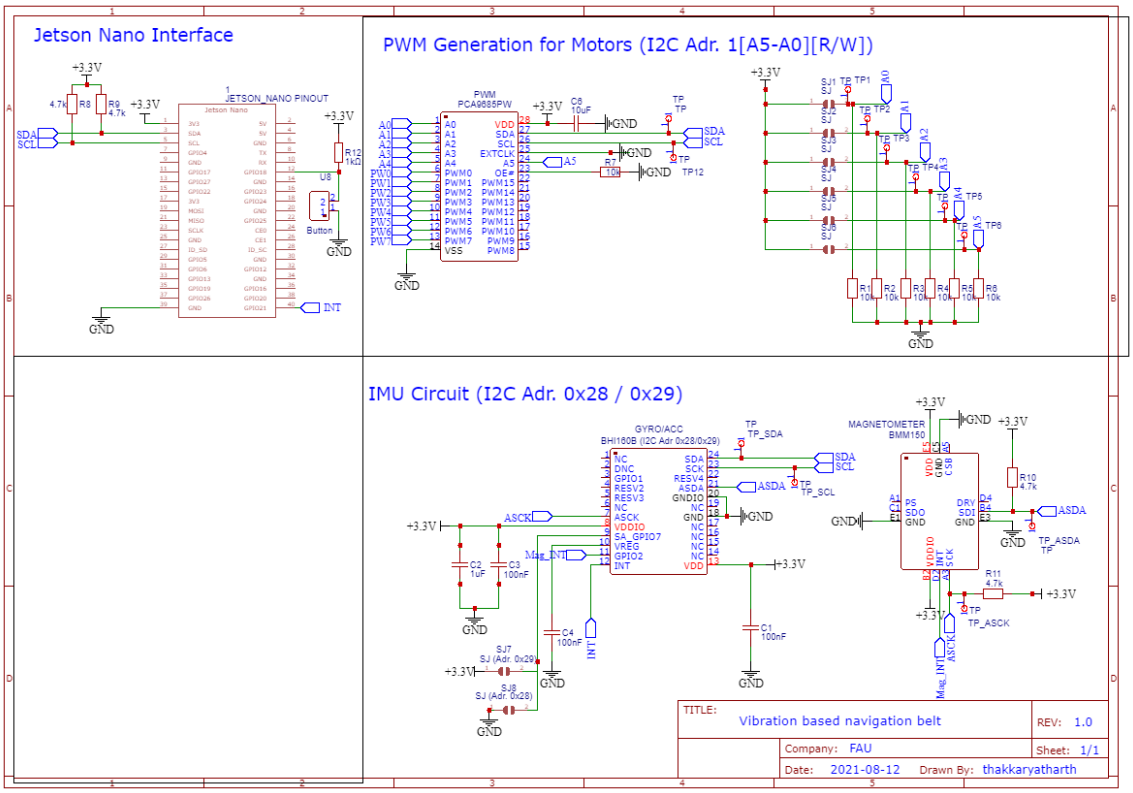


Figure 3: Jetson Nano Extension board schmetic

with this scheme the Jetson Nano goes into recovery mode and turns off due to an instability in the power supply.

The PCB design also includes an internal measurement unit (IMU) that is the Bosch BHI160b which includes a Gyroscope and Accelerometer and a Bosch BMM150 that is a Magnetometer. These have been placed to be used with the GNSS module to improve the position estimation of the user and determine where North is, both enabling better navigation. However, the IMUs were not implemented due to a chip shortage caused by the COVID-19.

2.2 Controlling the vibration motors

The vibration motors are controlled using the motors class found in Thakkar (2022) that uses the pca9685 library to control the PWM. It is initialised by selecting which

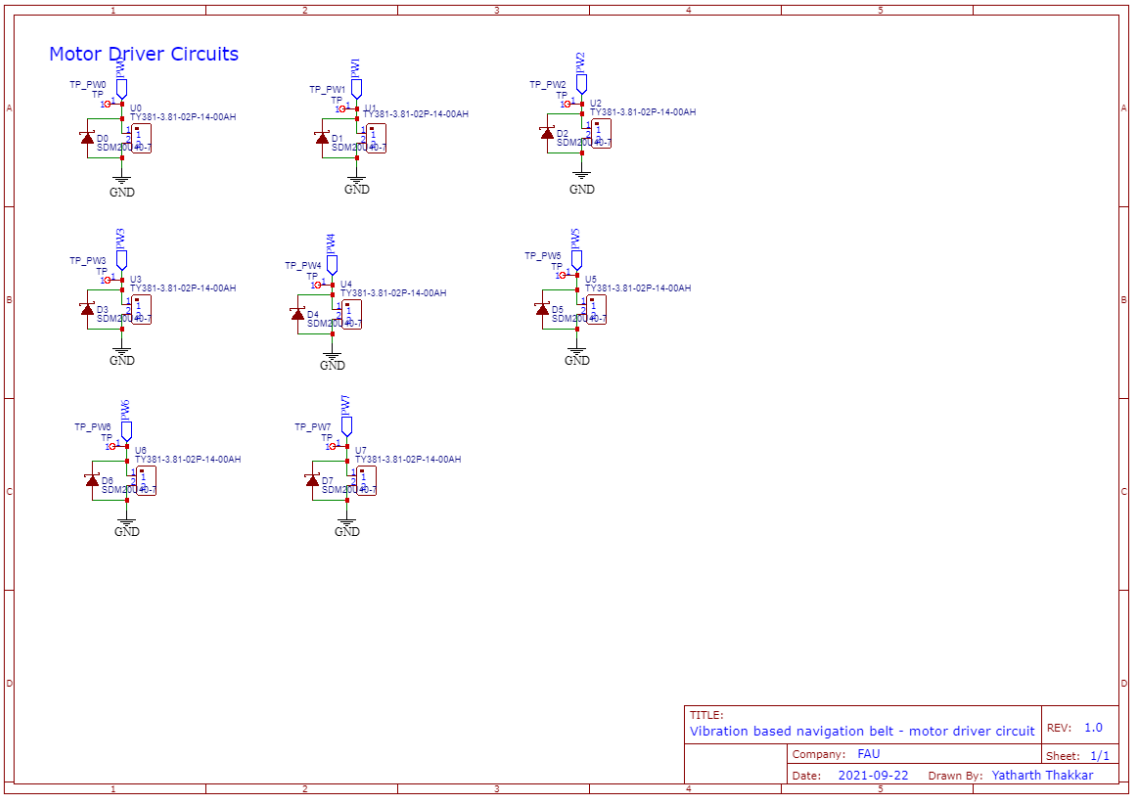


Figure 4: Jetson Nano Extension board schmetic

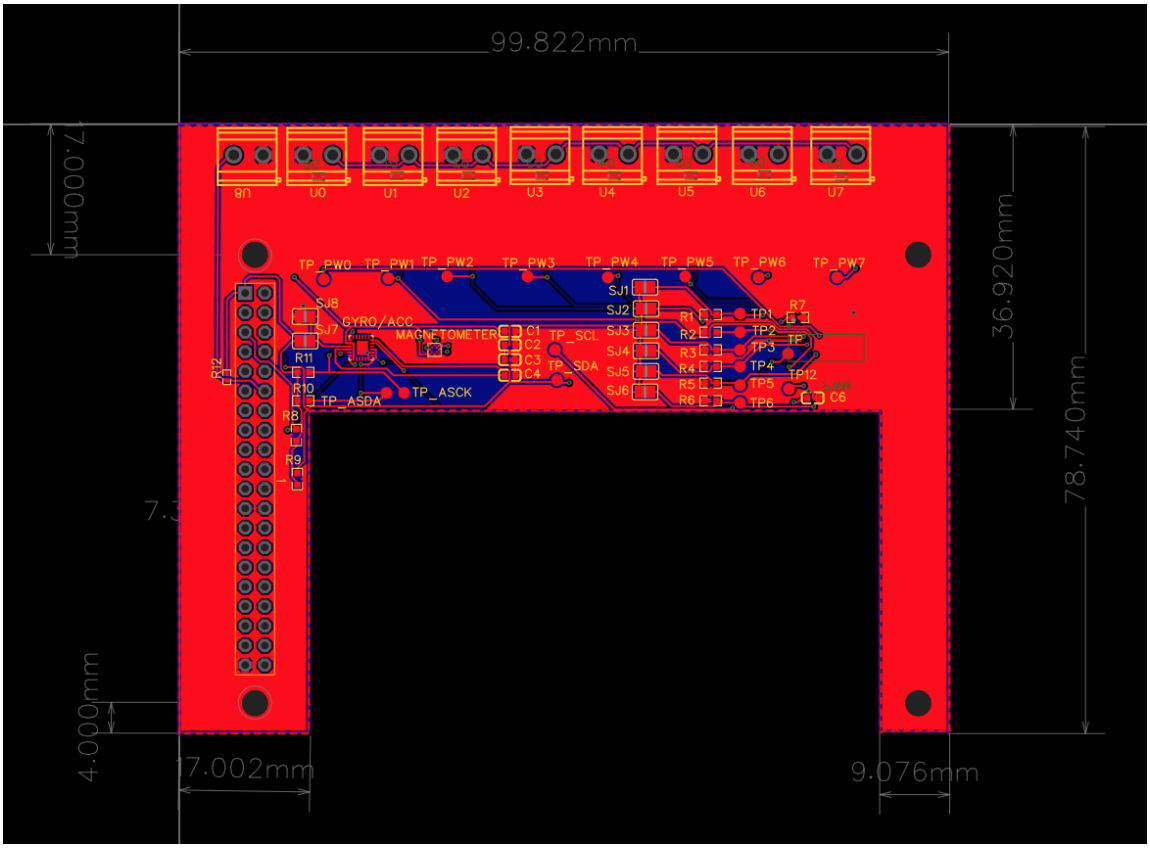


Figure 5: Jetson Nano Extension board pcb front

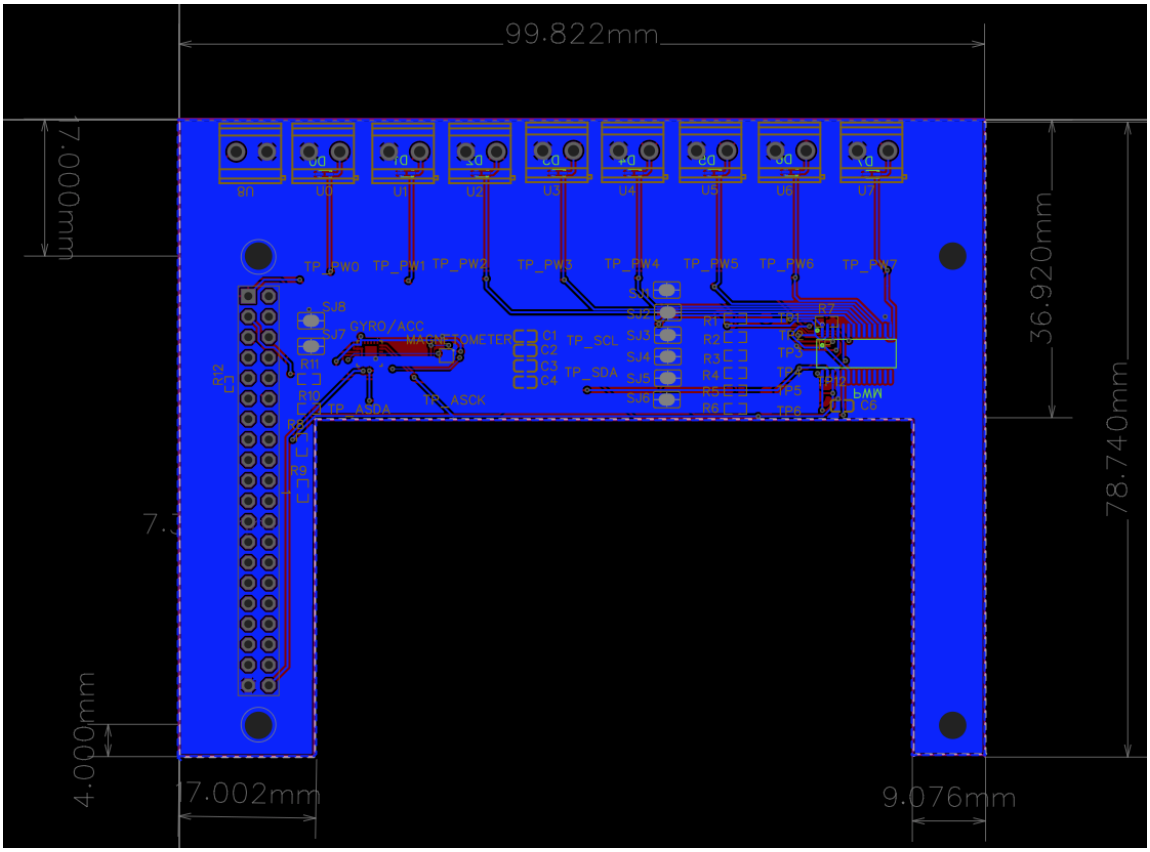


Figure 6: Jetson Nano Extension board pcb back

motors correspond to which channel. The motors library takes in the motors we want to control (the channel), the intensity as a percentage (integer between 0 and 100) and the instruction as parameters to engage a motor/sequence of motors. The motor sequence for a particular instruction can be modified in the engageMotor method in the Motor class.

2.3 Adding new motors

To add a new motor one must attach it to an empty terminal on the extension board and define it in the initialisation of the Motor class with the correct channel.

3 Image Segmentation and object detection

To perform the image segmentation and object detection the FCN-ResNet18-Cityscapes and Mobilenet-SSD models were used which were installed when setting up the Jetson Nano. Both models have been used to help guide the user when walking, avoiding obstacles and keeping them on the sidewalk. While Mobilenet-SSD was able to detect obstacles well, Cityscapes was not able to segment the road and sidewalk very well. Meaning we are able to trigger the motors correctly to avoid the detected obstacles, however are unable to help the user avoid walking into the road. This can be seen when looking at the statistics of the scene like in Figures 7 and 8. Figure 7a shows a scene captured while walking in the city where there is no vegetation separating the sidewalk and the road and Figure 7b shows the statistics counting what is being segmented where 54% of the image is being classified as building, 47% as road and 3% as sidewalk. Looking at both of these it can be seen that only the sidewalk across the road is segmented correctly whereas the sidewalk he user is walking on is segmented as a road. Something similar is seen in figure 8, here the sidewalk is between some vegetation and represents a more residential area and no sidewalk is detected by the model.
This problem can be mitigated by training a model that is specifically for sidewalks as FCN-ResNet18-Cityscapes is trained for cars, thus for roads. With this we should be able to correctly discriminate between road and sidewalk and guide the user to stay on the side walk and help in the decision of avoiding obstacles.

To avoid objects in the path of the user, two small regions on the center of the frame going from the bottom of the frame to 1/4th of the way up the frame are observed. One on the left side and another on the right. If an object overlaps one of these regions the motor corresponding to the opposite side will turn on with the

(a) Scene captured by the camera

```
grid size:   16x8
num classes: 21
----------------------------------------
 ID  class name        count     %
----------------------------------------
  0  void                0   0.000000
  1  ego_vehicle         0   0.000000
  2  ground              8   0.062500
  3  road               47   0.367188
  4  sidewalk            3   0.023438
  5  building           54   0.421875
  6  wall                0   0.000000
  7  fence               0   0.000000
  8  pole                1   0.007812
  9  traffic_light       0   0.000000
 10  traffic_sign        1   0.007812
 11  vegetation          2   0.015625
 12  terrain             1   0.007812
 13  sky                 7   0.054688
 14  person              0   0.000000
 15  car                 0   0.000000
 16  truck               0   0.000000
 17  bus                 0   0.000000
 18  train               0   0.000000
 19  motorcycle          0   0.000000
 20  bicycle             4   0.031250
nvbuf_utils: dmabuf_fd 1281 mapped entry NOT found
nvbuf_utils: NvReleaseFd Failed... Exiting...
nvbuf_utils: dmabuf_fd 1283 mapped entry NOT found
```

(b) Statistics of the scene captured by the camera

Figure 7: Scene captured when walking in a city

center motor. Telling the user to move slightly in that direction. So if an object is detected on the left region then the center and right motor will be engaged. If the object overlaps both then all the motors would turn on alarming the user. In the ideal case the user would be directed to move away from the road.
This method works however because the distance to the object is unknown the system is prone to false positives where an object is overlapping the region of interest but in reality is far away. To overcome this a stereo camera such as an Oak-D Lite could be used which has a similar form factor as the one currently used.

4 Navigation

The navigation in this system is based on the open route service directions API which can be found on ORS (2022). The directions API gives us the route between two points given the start and end co-ordinates and the path parameter which is for this case is "foot-walking". It returns an updated route with the current instruction for the user, the distance to the next turn, the steepness, surface type, way type (e.g. street) and other interesting information about the road which may be useful to support navigation and segmentation.

(a) Scene captured by the camera

```
num classes: 21
----------------------------------------
 ID  class name        count     %
----------------------------------------
  0  void                  0   0.000000
  1  ego_vehicle           1   0.007812
  2  ground                2   0.015625
  3  road                 44   0.343750
  4  sidewalk              0   0.000000
  5  building              1   0.007812
  6  wall                  1   0.007812
  7  fence                 0   0.000000
  8  pole                  3   0.023438
  9  traffic_light         0   0.000000
 10  traffic_sign          1   0.007812
 11  vegetation           52   0.406250
 12  terrain              12   0.093750
 13  sky                   5   0.039062
 14  person                0   0.000000
nvbuf_utils: dmabuf_fd 1287 mapped entry NOT found
nvbuf_utils: NvReleaseFd Failed... Exiting...
 15  car                   6   0.046875
 16  truck                 0   0.000000
 17  bus                   0   0.000000
```

(b) Statistics of the scene captured by the camera

Figure 8: Scene captured when walking in a residential area

In the current system, the end co-ordinate and path parameter are preset by the user and the current co-ordinate updated by the GNSS readings. Because there is no IMU present it requests ORS for the route instructions every 2 seconds and the distance to the next instruction is seen to trigger the vibration motor. Because of this, it is important to have internet connection while walking. Because the GNSS readings are not very accurate the system also has some trouble getting the users location accurately and thus determining how much further the person needs to walk till the next instruction. External factors such weather and the surrounding buildings also play a role in how accurate the localisation is. The inaccuracies can be seen in Figures 9 and 10 showing readings coming from similar routes but Figure 9 better represents the route than Figure 10. By using sensor fusion techniques such as a Kalman filter with an IMU and GNSS receiver one can improve the localisation and also omit the need for internet throughout the journey by just computing the distance the user has travelled in which direction and comparing it with the set of instructions initially provided by ORS.

To improve navigation one can also use a GNSS module with a better antenna. While testing it was found that the PA1010D GNSS module has some difficulty retrieving signals and is very dependent on the surroundings. Not having an magnetometer also makes it difficult to navigate correctly as instructions from ORS included cardinal directions such as "Head north on Hauptstrasse then turn left". Here in this example without knowing where you are going or where north is it is difficult to walk in the correct direction.

Figure 9: Coordinates sampled by the GNSS module when walking around a block under good conditions

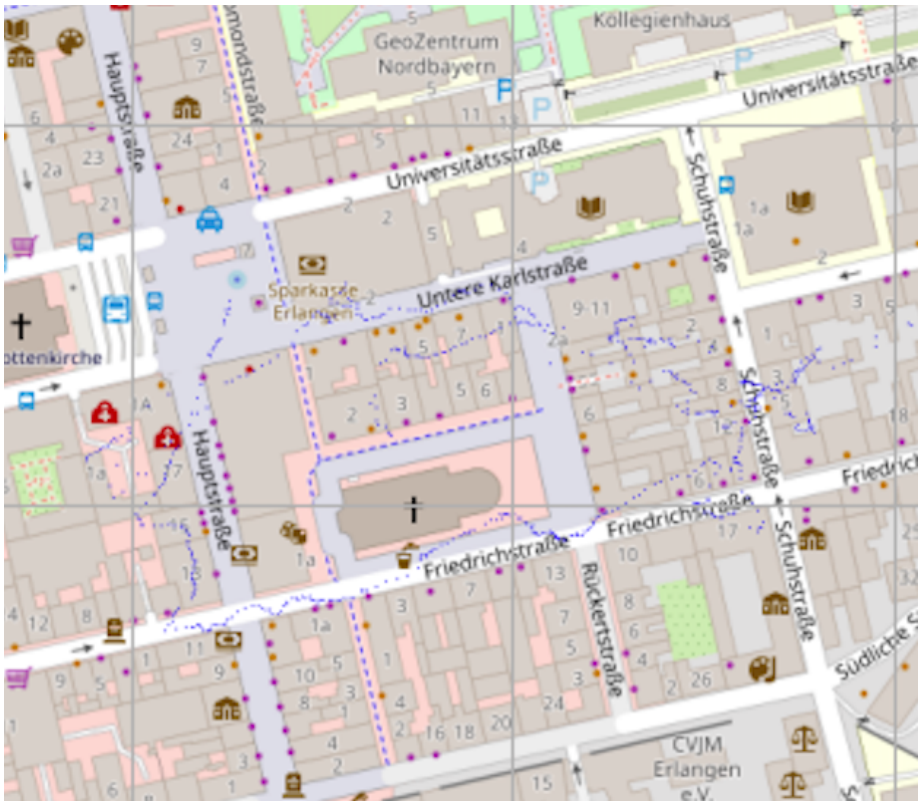


Figure 10: Coordinates sampled by the GNSS module when walking around a block under moderate conditions

5 Conclusion

This report outlines the current state of the SENSATION project. Covering what has been developed and what needs to be improved. This includes the frame that is laser cut and 3D printed to hold all the components, the haptic belt and extension board developed to drive the vibration motors to guide the user, the image segmentation and object detection to understand the users surroundings and the navigation to allow the user to go from one place to another.

References

Alarcon, N. (2020). Real-time object detection in 10 lines of python on jetson nano. *https://developer.nvidia.com/blog/realtime-object-detection-in-10-lines-of-python-on-jetson-nano/*.

ORS (2022). Open route services api interactive examples. *https://openrouteservice.org/dev//api-docs/v2/directions*.

Thakkar, Y. (2022). Sensation. *https://github.com/ThaatGuy/Sensation*.