Informatics

IMU Adapter for Autonomous Race Car

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Technische Informatik

eingereicht von

Fabjan Sukalia
Matrikelnummer 01225753

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Univ.Ass. Dipl.-Ing. Andreas Brandstätter, BSc

Wien, 1. April 2022

Fabjan Sukalia | Radu Grosu

Informatics

IMU Adapter for Autonomous Race Car

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Computer Engineering

by

Fabjan Sukalia
Registration Number 01225753

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Univ.Ass. Dipl.-Ing. Andreas Brandstätter, BSc

Vienna, 1st April, 2022

Fabjan Sukalia

Radu Grosu

Erklärung zur Verfassung der Arbeit

Fabjan Sukalia

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. April 2022

Fabjan Sukalia

Danksagung

Ich möchte mich allen bedanken, die mich bei der Ausarbeitung dieser Arbeit unterstützt haben. Ohne eure Hilfe, wäre diese Arbeit nicht möglich gewesen. Zu allererst möchte ich mich bei Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu und bei Univ.Ass. Dipl.-Ing. Andreas Brandstätter für ihre Betreuung, Verbesserungsvorschläge und Anmerkungen danken. Weiters möchte ich mich bei meiner Familie bedanken, die mich ständig unterstützt hat und mir Kraft und Ausdauer geschenkt hat.

Acknowledgements

I want to thank everybody who helped me with my thesis. Without your support this work would not be possible. First, I want to thank Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu und bei Univ.Ass. Dipl.-Ing. Andreas Brandstätter for their supervision, advises and suggestions for improvements. Furthermore I also want do thank my family for their support and giving me strength and endurance.

Kurzfassung

Die Lehrveranstaltung *Autonomous Racing Cars* benutzt Modellfahrzeuge um Studenten über Algorithmen für autonomes Fahren zu unterrichten. Die Studenten entwickeln hierbei Algorithmen für das autonome Fahren und testen auch ihre Algorithmen mit den Modellfahrzeugen. Die Modellfahrzeuge sind mit Sensoren und einer leistungsfähigen Rechenplattform ausgestattet. Die Entwicklung dieser Algorithmen ist sehr aufwendig, jedoch helfen hier Werkzeuge in Software und Hardware beim Entwicklungsprozess. Die Modellfahrzeuge aus der Lehrveranstaltung besitzen jedoch keine hardware-basierte Hilfswerkzeuge.

In dieser Arbeit werden hardware-basierte Hilfswerkzeuge für die Studenten entwickelt um sie bei der Entwicklung ihrer Algorithmen zu unterstützen. Mit dieser Erweiterung der Modellfahrzeuge können die Studenten einfacher den internen Zustand ihrer Software untersuchen oder erkennen ob ein Fehler aufgetreten ist. Weiters können die Studenten auch so einfacher die Konfiguration oder Verhalten der Software zur Laufzeit ändern.

Abstract

The course *Autonomous Racing Cars* uses model racing cars to teach students about algorithms for autonomous driving. In this course the students have to develop algorithms for autonomous driving and they test their algorithms on model racing cars. The model cars are equipped with the various sensors and a powerful computing platform. Developing algorithms for autonomous driving is challenging and utilising software and hardware tools can help during the development process. The model cars used in the course lack the hardware tools that could help the students.

This work concentrates on extending the hardware to give the students the necessary tools that can help them during the development process. With the hardware extension the software can immediately tell the student of the internal state or if a fault occurred and the student can also change the configuration and behaviour of the software through the hardware extension. The aim is that during runtime it will help the students to faster diagnose errors and help inspecting the parameters of the algorithms to further improve them.

Contents

Kurzfassung **xi**

Abstract **xiii**

Contents **xv**

1 Introduction **1**
1.1 Motivation and Problem Statement 1
1.2 Aim of the Work 2
1.3 Outline 2

2 Related work **5**
2.1 F1TENTH 5
2.2 Previous work on F1TENTH 5

3 Implementation **7**
3.1 Hardware 7
3.2 Software 14

4 Result **21**
4.1 PCB Layout 21
4.2 Power consumption 22

5 Conclusion and Outlook **25**

List of Figures **27**

List of Tables **29**

List of Algorithms **31**

Bibliography **33**

CHAPTER 1

Introduction

1.1 Motivation and Problem Statement

Autonomous driving is a research topic that is actively researched on and gained traction in recent years. Car manufacturers try to equip their cars with autonomous driving capabilities. Also the robotic industries uses algorithms for autonomous driving to enhance their robotic machines. Algorithms for autonomous driving can get complex and also have to handle various different environments or have to be optimized to a specific environment to be more efficient.

The course *Autonomous Racing Cars* teaches students about algorithms for autonomous driving. In the course students use a simulator and model racing cars to implement and test their own algorithms for autonomous driving. Figure 1.1 shows four model cars that are used in the course. The model cars are based on the build documentation of F1TENTH ([f1t]). The students have to develop algorithms for different tasks, i.e. wall following, obstacle avoidance, path following and race track tracking navigation.

The algorithms gets first developed and tested on the simulator and later ported to and tested on the model car. During the porting to the model car the student encounter differences between the simulator and the model car. While the simulator has large set of tools to debug and inspect the errors, the model cars lack such tools.

The model car uses sensors to scan the environment or the movement of the car. Currently it uses a light detection and ranging sensor (*LIDAR*) for scanning the environment and detecting obstacles and a 9-axis inertial measurement unit (*IMU*) for measuring the movement of the car. The IMU is mounted on a prototyping board on top of the computing platform. It could have more sensors and hardware components but having so many components is unpractical for a prototyping board and better suited for a printed circuit board (*PCB*).

Figure 1.1: F1TENTH model cars as used in the course *Autonomous Racing Cars*

Previously the model cars used the Nvidia Jetson TX2 computing platform with a third party carrier board. The Jetson TX2 platform became obsolete and is replaced by the Nvidia Xavier NX devkit. The board where the IMU is mounted does not fit on the new platform and therefore a new board is needed.

1.2 Aim of the Work

The goal of this work is to develop a PCB that hosts the IMU and other components that help the student to debug their software. Also software driver are developed that connect the software with the hardware components.

As a first step the necessary hardware components are determined together with their use. A circuit is designed for the components and finally a PCB layout is created from the schematic and the dimensions of the devkit. The PCB has to meet some constraints, i.e. the size must match the devkit, mounting holes must be aligned, power draw is limited by the devkit, fan inlet must not be obstructed, IMU must be aligned to the center line of the model car.

1.3 Outline

The next chapter introduces the F1TENTH project and presents some related work done in the field. Chapter 3 presents the hardware and software solution of this work and

chapter 4 evaluates the solution. The last chapter summarizes the work and shows what further work and improvements can be done.

CHAPTER 2

Related work

2.1 F1TENTH

F1TENTH ([f1t]) is an open project and a community that organizes races, publishes documentations for their autonomous model cars, and promotes research for autonomous driving. It was founded in 2016 by the University of Pennsylvania and today it has many research institutes as partners, including TU Wien.

The community regularly organizes racing competitions where teams from different institutes can participate. The documentation contains a building instruction of the model car, installation instruction of the software, a course that teaches different techniques for autonomous driving, and some exercises and lab assignments as teaching materials. At the time of writing this thesis the last racing competitions was held at the *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2021)* in Prague, Czech Republic from 27.9.2021 to 29.9.2021.

2.2 Previous work on F1TENTH

The F1TENTH had some iterations of model cars and some modifications to the hardware were done by researches. In the thesis of [Vaj17] a model race car was designed and built for the competition in 2017. The model car uses a Nvidia Jetson TK1 as the computing platform and runs Ubuntu 14.04 with ROS Indigo. Figure 2.1 shows an overview of the peripherals and how they are connected. All peripherals are connected to the computing platform via USB. The peripherals are an IMU, a LIDAR, a WiFi module, a stereo camera and a microcontroller board. It uses a Teensy 3.2 microcontroller board to generate the pulse-width modulation (PWM) signals for the motor controller and the servo motor for the steering. The Teensy board also receives two PWM signals from a radio frequency (RF) transceiver. The Teensy board can route the PWM signals form the

RF transceiver to the motor control and the servo motor so that the user can manually control the model car via remote control. In the automatic mode the Teensy board does not use the PWM signals from the RF transceiver but generates PWM signals that are controlled by the software on the computing platform.

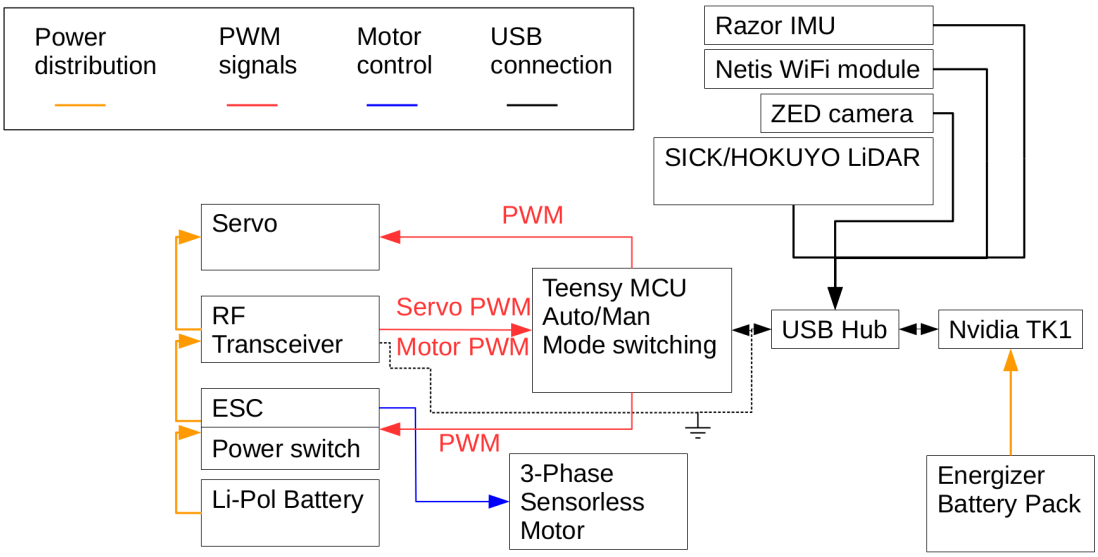


Figure 2.1: An overview of the peripherals used in [Vaj17] and how they are connected. Source: [Vaj17]

Based on the previous work [Dus19] made some necessary modifications on the hardware as some used parts became obsolete. At first the computing platform was replaced with a Nvidia Jetson TX2 and with a Orbitty Carrier from Connect Tech. The stereo camera was replaced with a Intel RealSense Depth Camera D435. The power circuit was also redesigned. A PCB was designed for the power circuit and the Teensy board.

The thesis of [Gue19] develops a testbed for platooning of robotic cars. It uses the F1TENTH model cars and modifies the hardware to include an infrared range finder, an ultrasonic range finder and an automotive radio transceiver that supports IEEE 802.11p. The range finder sensors were placed on a white board to the front of the model car so it can measure the distance to the model car in front of it. The board is white because the infrared range finder need a reflective surface to measure the distance. On the white board are three ultrasonic and four infrared range finder installed. The radio transceiver communicates with the model cars that are close to it.

CHAPTER 3

Implementation

At the heart of the model car is the *Nvidia Xavier NX Devkit*. It controls the model car and runs the software that navigates through the environment. The devkit consists of the compute module, and the carrier board with the power supply and connectors. The extension board will be mounted on top of the carrier board. Figure 3.1 shows the model car with all the equipment that was used previously in the course.

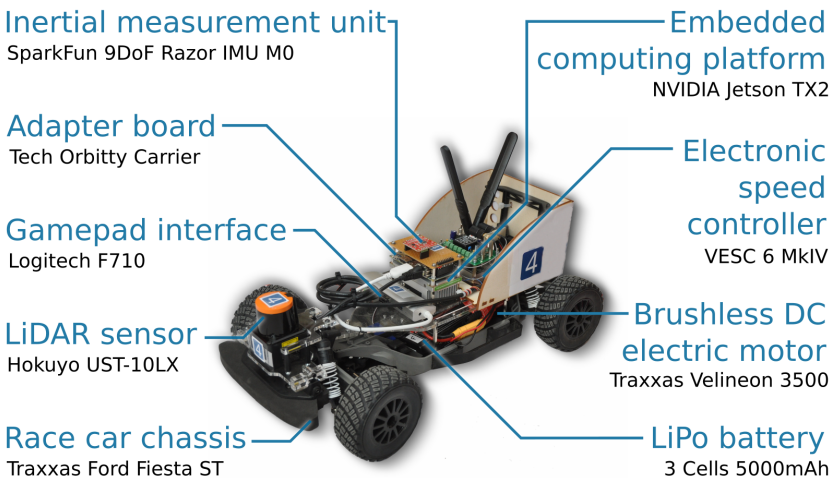


Figure 3.1: Previous model car from the course with all hardware components labeled.

3.1 Hardware

The extension board is designed to include all the necessary components and to be placed directly on the Xavier NX board. When placed directly on the Xavier NX board it would obstruct the fan and the heat sink. Figure 3.2 shows the Xavier NX board with the

compute module and the heatsink and fan mounted. In order to not obstruct the air flow of the fan intake and therefore not degrade the cooling capacity, the extension board has a cut-out for the fan and is mounted with 30 mm board standoffs to the Xavier NX board. The board standoffs are also tightened with screws so that it can withstand the forces and vibrations that occur during driving.

Figure 3.2: Nvidia Xavier NX development kit. Source: [nvi]

The Xavier NX board features a 40-pin connector with two power rails, general-purpose input/output pins (GPIO) and different bus systems. The circuit of the extension board is connected to the 40-pin connector. The power supply rails are 3.3 V with a maximum of 100 mA and 5.0 V with a maximum 500 mA. The available current of the power rails imposes a constraint of the circuits on the extension board. All the signals on the connector use 3.3 V power levels.

3.1.1 LED Indicators

The software can show the status with four different colored *light-emitting diodes (LED)* indicators which are replicated and placed on all four edges of the extension board so that they can be seen from anywhere around the model car. The LED indicators are colored in red, yellow, green and blue and every color appears on every edge of the PCB. The meaning of the colors is determined by the software.

The red, yellow and green LEDs have a typical forward voltage of 2.0 V to 2.2 V and the current for each LED is limited by a 300 Ω resistor to approximately 10 mA. The blue

LED has a higher typical forward voltage of 3.3 V and needs a 180 Ω resistor to limit the current to approximately 9.4 mA. In total the current draw of all sixteen LEDs together is around 160 mA.

All LED indicators are connected to the 5 V power rail because the total current draw is higher than the 3.3 V power rail can provide. All four LEDs of the same color are switched on or off by a single N-channel metal–oxide–semiconductor field-effect transistor (MOSFET). This MOSFET is controlled by a GPIO signal from the 40-pin connector. The MOSFET needs a suitable low gate-source threshold voltage so it can be switched on by the 3.3 V signal. Figure 3.3 shows the schematic for one LED color.

The MOSFET that is used, is a *Nexperia BSS138P,215* and it has a gate-source threshold voltage from 0.9 V to 1.5 V which is low enough for the 3.3 V signal. It also has a maximum drain current of 360 mA which is large enough.

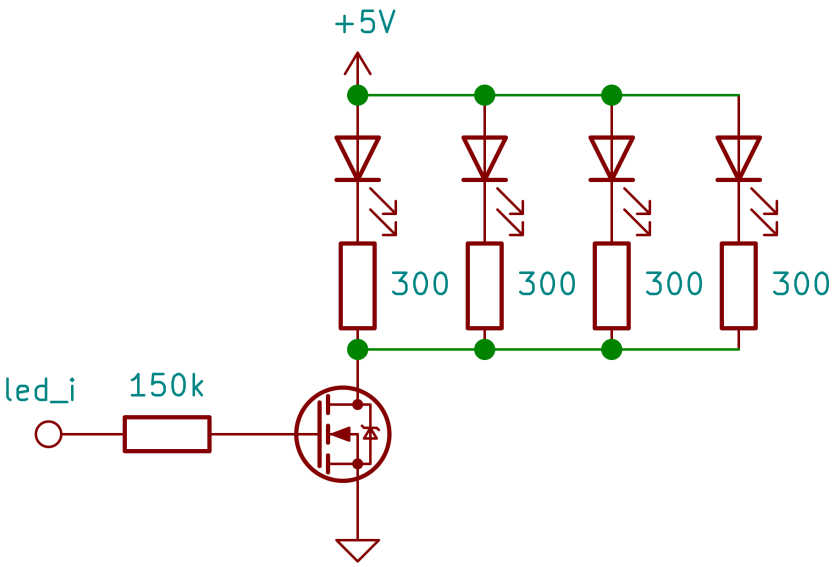


Figure 3.3: Schematics of the LED indicators. This circuit is replicated four times, once for each color.

The LED indicators that are on the rear side of the extension board are split up into two groups of two LEDs each. The cut-out for the fan intake and the mounting holes for the display takes up some space on the rear side so that the LED indicators had to be split.

3.1.2 Buttons and Switches

On the extension board are two buttons and two switches so that the developers can change the behaviour of the software directly on the model car. The buttons and switches are aligned to the sides of the display to have a better integration with the graphical user interface. Next to each button and switch is a LED that lights up when the button is pressed or the switch is set to a certain position.

Both the buttons and switches are internally single poll double throw circuits (SPDT). Also both have the same footprint and can be easily interchanged, e.g. an extension board can only have switches or only buttons or the arrangement can be changed.

Because mechanical buttons and switches do not cleanly close or open the contact a phenomenon called bouncing occurs every time a button is pressed or a switch is switched. A RS latch build from NAND gates can be used as a debouncing mechanism for the switches. The schematic for the debouncing circuit is shown in figure 3.4.

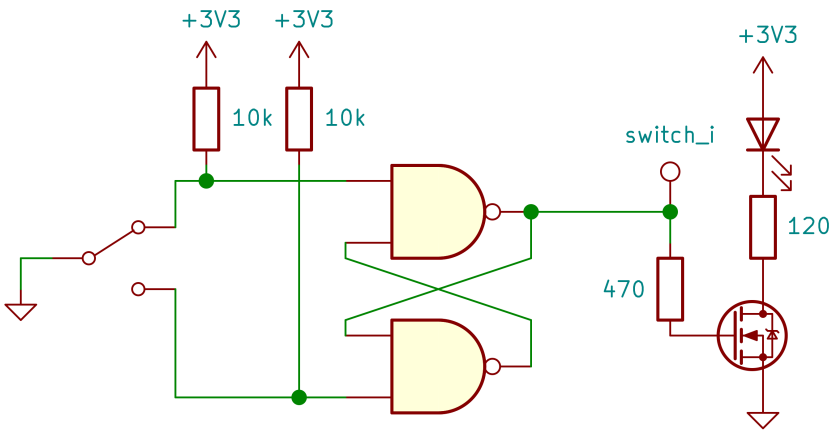


Figure 3.4: Schematic for debouncing a switch or a button and signalling it with an LED.

3.1.3 Display

To show a wide range of information to the developer a display is also integrated on the extension board. The display itself is part of another PCB that has some circuit on it to help power the display. The display board is the *SparkFun TFT LCD Breakout - 1.8"*. The display itself has a resolution of 128x160 pixel with a diagonal size of 45 mm. It can display colors with a color depth of up to 18 bit, where every color channel has 6 bit. The back-light can be controlled by a PWM signal.

The display board is mounted with board standoff on the extension board and is located above the cut-out for the fan inlet. The gap between the display board and the extension board is large enough to not obstruct the fan inlet.

Data can be send to the display via the seril peripheral interface bus (SPI). Both the display board and the Xavier NX board use the *TI TXB0108* as level shifter. Because the level shifters on the Xavier NX board and on the display are designed for bidirectional busses the output drive strength of the level shifters is weak and it can not drive the input of another bidirectional level shifter. To mitigate this issue a bus transceiver like the *74LVC245* can be connected between the two level shifters. This solution works because the signals in the SPI bus are unidirectional and the display board has no output signal for the SPI bus.

3.1.4 Buzzer

The buzzer is used as an acoustic alarm to immediately notify the developer of a severe situation. The software decides what a severe situation is, e.g. when the car is about to hit a wall, the software a state it should not have reached, the car lost the wireless connection, etc. But it can be also used to play simple tones. The buzzer is controlled by the software and by the battery monitor. The software can use the PWM functionality of the GPIO and output square wave tones of a single frequency. The battery monitor sends a 2 kHz square wave tone when the battery voltage is too low.

The signal from the 40-pin connector and the signal from the battery monitor each are connected to a MOSFET. The drain output of the MOSFETs are connected in parallel to drive the buzzer, similar to a wired-or configuration. Figure 3.5 shows the schematic with the two parallel MOSFETS. The disadvantage is that when one signal permanently switches the MOSFET on then the other signal is ignored. To not block the other signal the MOSFETs must be turned off when idling.

We use a piezoelectronic buzzer that is not self-oscillating. So the software can define the frequency of the tone. A piezoelectronic buzzer can induce a voltage once they are vibrating and if the switching MOSFET is turned off it can damage the MOSFET. In order to protect the MOSFET a resistor must be connected parallel to the buzzer. The resistor dissipates the induced voltage of the buzzer. The manufacturer of the buzzer recommends a resistor of about 1 kΩ which is also used in this circuit.

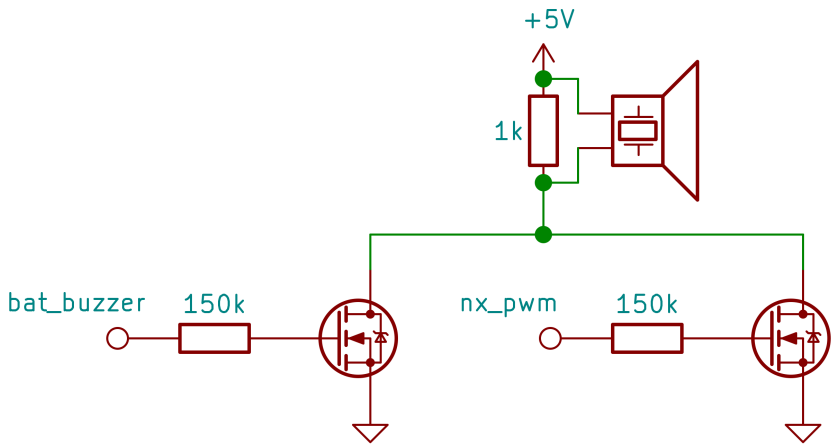


Figure 3.5: Schematic of the buzzer circuit. With the parallel configuration it can be controlled by two different signals.

3.1.5 Battery monitor

The main power source for the model car is a 3-cell lithium polymer battery with 5000 mA h capacity. The nominal voltage of the battery is 11.1 V, when the battery is fully charged it is around 12.6 V and when it is discharged it is approximately 9.0 V.

The battery monitor starts an alarm when the battery voltage gets below a certain threshold. This safety measure is necessary because a fully discharged battery can get damaged. If the battery gets too damaged it can become a fire hazard. The alarm consists of an LED that blinks twice a second and a buzzer that beeps with a 2 kHz tone. The alarm is turned off once the battery voltage is above a certain threshold.

The two thresholds for starting the alarm and stopping the alarm can be manually adjusted via two potentiometers. The threshold for turning the alarm off should be higher than the threshold for turning the alarm on in order to have a hysteresis and to prevent that the alarm gets turned on and off in a oscillating manner. The potentiometers are part of a voltage divider to translate the voltage of the battery to a lower voltage that can be safely used with electronic components that operate with 3.3 V. The schematic of the voltage divider is shown in figure 3.6. With the resistor values in figure 3.6 the voltage range for the upper and lower threshold is from 2.3625 V to 3.675 V for a full battery at 12.6 V and from 1.6875 V to 2.625 V for a nearly empty battery at 9.0 V.

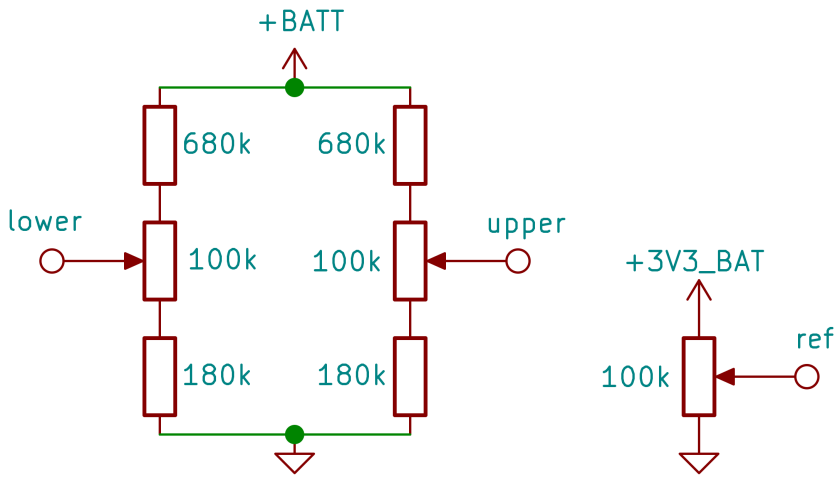


Figure 3.6: Schematic of battery threshold trimmer

The output voltages of the potentiometers are fed to two comparators that compare these voltages against a reference voltage. This reference voltage is constant and does not change when the battery voltage changes. The output signals of the comparators indicate if the battery voltage is above or below the given thresholds. The outputs of the comparators are then fed to a RS latch that memorizes if the alarm is on or off. The RS latch is build out of NAND gates like in section 3.1.2. The comparator circuit with the RS latch is shown in figure 3.7.

The signal that indicates if the battery is low is connected to a GPIO pin of the 40-pin connector and it also controls two oscillators that are serially connected. A single stage of the oscillator cascade is shown in figure 3.8. The first oscillator generates a 2 Hz signal which lets the alarm LED blink and it also enables the second oscillator. The second oscillator generates a 2 kHz signal that is connected to the second MOSFET that drives

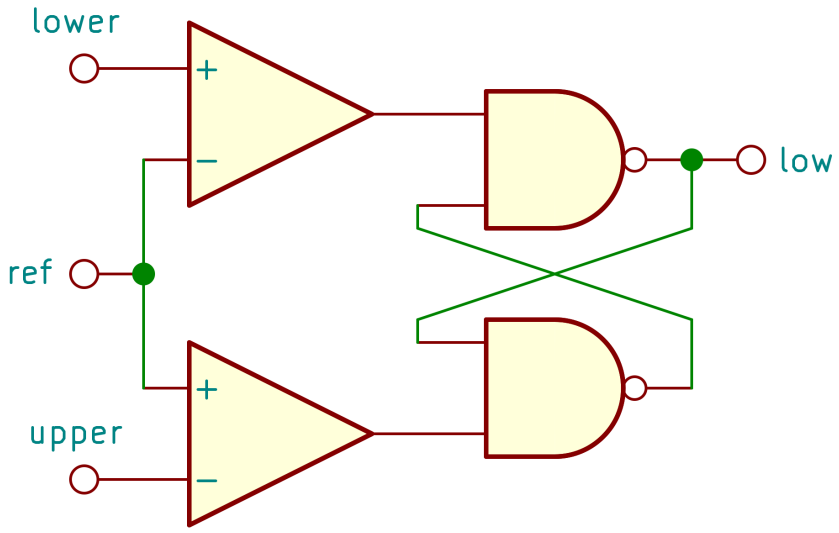


Figure 3.7: Schematic of the threshold comparator with RS latch

the buzzer, as depicted in figure 3.5. The frequency of each oscillator is determined by the values of the resistor and capacitor that are marked as R and C in figure 3.8. The 2 Hz frequency can be achieved by setting R to 100 kΩ and C to 3.6 µF and the 2 kHz frequency can be achieved with the values 10 kΩ and 36 nF.

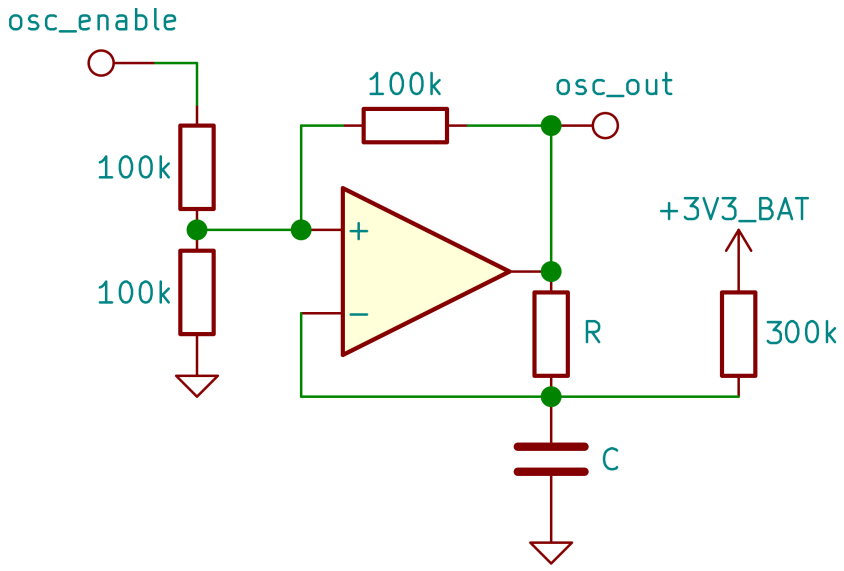


Figure 3.8: Schematic of a single stage oscillator with enable input. The values of R and C define the frequency of the oscillator.

Additionally the battery monitor circuits has an analog-digital-converter (ADC) to measure the battery voltage. The battery voltage is translated to a quarter of its voltage

via a voltage divider made out of resistors, so that the battery voltage can be safely be measured by the ADC. To stabilize the measurement and to filter out noise a capacitor is attached to the output of the voltage divider.

The ADC has a resolution of 10 bit and is connected to the Xavier NX board via the I²C bus. The reference voltage is the same as the supply voltage which is 3.3 V. This means that a difference in the least significant bit of the conversion is a difference of approximately 3.22 mV. By applying the factor of the voltage divider the ADC can therefore measure the battery voltage in steps of 12.89 mV. For the range of the battery voltage this means that the ADC can distinguish the voltage in approximately 279 steps.

The battery monitor circuit except the ADC are powered by the battery instead of the Xavier NX board so that it can operate even when the computing platform is turned off. The 3.3 V is generated with a linear regulator. The circuit for the power supply is shown in figure 3.9 and also shows the capacitor on the input and output of the linear regulator which are recommended by the manufacturer of the linear regulator. The output voltage of the linear regulator also powers a LED to notify the user that the battery is connected.

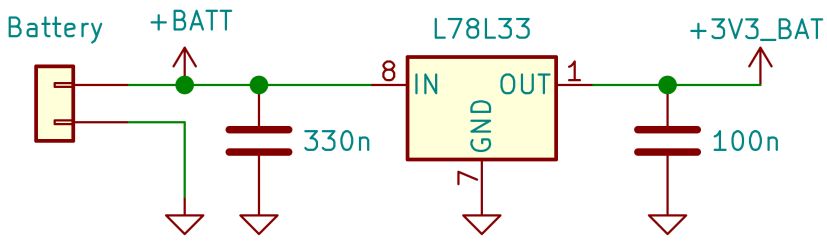


Figure 3.9: Schematic of the battery power supply for the monitor circuit

3.1.6 Inertial Measurement Unit

An important sensor that helps the software is the inertial measurement unit (IMU). The IMU on the extension board is the *SparkFun OpenLog Artemis*. It communicates with the Xavier NX board via UART. It is also mounted with board standoff onto the extension board and gives more room for other components on the extension board. It is placed so that the IMU sensor is on the center line of the extension board and therefor also on the center line of the model car. This sensor does not need additional circuits and the signals can be directly connected to the computing platform.

3.2 Software

3.2.1 Robotic Operating System

The F1Tenth project uses Robotic Operating System (ROS), a software framework that is designed for robotic applications. It has an ecosystem of drivers and algorithms which can be used for various robotic platforms and use cases. One of the intended use cases is autonomous driving. In a ROS environment multiple processes communicate with each

other. Typically a process is also a ROS node. The communication between nodes is mainly done by publishing and subscribing to a topic and exchanging messages or by requesting and replying to a service. Because of this abstraction the nodes do not know with which node they communicate and so the nodes can be exchanged to operate with a different algorithm but with the same communication interface ([ros]).

3.2.2 Operating system

The manufacturer of the Xavier NX development board provides a fully functional operating system that includes all necessary drivers and firmwares. This operating system is *JetPack 4.6* and is based on *Ubuntu 18.04*. This release includes the *L4T 32.6.1* drivers and libraries.

In the default configuration the operating system does not include some necessary kernel modules and Python packages that are needed to access the hardware on the extension board. The missing kernel modules are *spidev* and *i2c-dev*. Both modules are needed to give user-space programs, like ROS nodes, access to the SPI and I^2C busses. These modules were installed by downloading the kernel source code and compiling and installing the missing modules.

Some of the pins on the 40-pin header have additional functions which are per default disabled. The tool *Jetson-IO* configures the pins and activates the additional functions by generating a *device tree overlay* which will be loaded by the operating system during start-up. The configuration needed for the extension board can be seen in figure 3.10.

All the nodes are written in Python and they need some libraries to access the hardware or gain additional functionality. All nodes control the extension board via GPIO pins. The manufacturer provides the library *Jetson.GPIO* for easy access of the GPIO pins and some additional hardware like PWM output. The display node uses the *Pillow* library for drawing the graphical user interface. Additionally, it uses the *spidev* library to access the SPI bus that is provided by the kernel module mentioned previously. The battery monitor node has to access the I^2C bus which it does with the help of the *smbus2* library. All these needed libraries are installed with the tool *pip*.

3.2.3 LED indicators

The led node represents the LED indicators on the extension board. Nodes can send a message to the leds node in order to change the state of the four colors. The state of this node can be represented by four boolean value, one value for each of the four colors. Algorithm 3.1 shows the message format with the full state. Other nodes can send a message to the leds node and change the state. When a message is received then the GPIO pins that control the LEDs are set according to the new state.

```
=== Jetson Expansion Header Tool ==

      3.3V (  1) .. (  2) 5V
      i2c9 (  3) .. (  4) 5V
      i2c9 (  5) .. (  6) GND
    unused (  7) .. (  8) uarta
       GND (  9) .. ( 10) uarta
    unused ( 11) .. ( 12) unused
  spi3_sck ( 13) .. ( 14) GND
    unused ( 15) .. ( 16) spi3_cs1
      3.3V ( 17) .. ( 18) spi3_cs0
 spi1_dout ( 19) .. ( 20) GND
  spi1_din ( 21) .. ( 22) spi3_din
  spi1_sck ( 23) .. ( 24) spi1_cs0
       GND ( 25) .. ( 26) spi1_cs1
      i2c2 ( 27) .. ( 28) i2c2
    unused ( 29) .. ( 30) GND
    unused ( 31) .. ( 32) pwm8
      pwm1 ( 33) .. ( 34) GND
    unused ( 35) .. ( 36) unused
 spi3_dout ( 37) .. ( 38) unused
       GND ( 39) .. ( 40) unused
```

Figure 3.10: The tool *Jetson-IO* showing the special function configuration for the 40-pin header

Algorithm 3.1: LED message format

```
bool red
bool yellow
bool green
bool blue
```

3.2.4 Buttons

The button node periodically reads the button states and sends a message to the subscribed nodes. The receiving nodes can then detect a button press or release or the switch position. The button node does the readout and message publishing 20 times per second. This rate is fast enough to reliably detect state changes. The message format for the button state is shown in algorithm 3.2 and uses a boolean value for each button and switch. Because buttons and switches are electrically the same there is no need to differentiate between them in the message format.

Algorithm 3.2: Button message format

```
bool top_left
bool bot_left
bool top_right
bool bot_right
```

3.2.5 Display

The display node is the only node for the extension board that has two message formats. One message format is needed for controlling the backlight and the other message format is for the content of the display. These message formats are separate because usually the backlight is changed infrequently compared to the display content.

The backlight message, as seen in algorithm 3.3, only contains a floating point number between 0.0 and 1.0 for the brightness, where 0.0 means no backlight and 1.0 means full brightness. Numbers outside this range are clamped to the range. The backlight LEDs of the display are controlled by a PWM signal.

Algorithm 3.3: Backlight message format

```
float32 pwm_level
```

The display content is divided into five areas, where each area shows text information. With the second message format the content of four of the five areas can be modified. The fifth area holds information about the system, i.e. network addresses, time and date. Figure 3.11 shows the five areas with each area having a different background color. The area in the center holds the system information. The other four areas are placed so that their position is next to a button or switch. The content of the display is send to the display via the SPI bus.

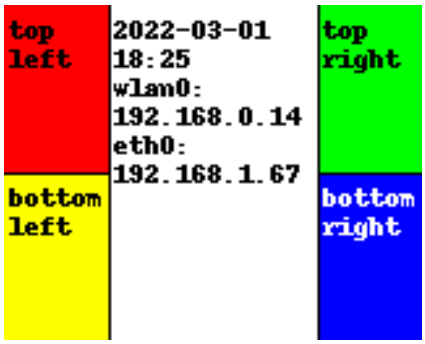


Figure 3.11: The five areas of the display content

Algorithm 3.4 shows the message format for modifying the text areas. The first field *textfield* is the index that selects the text area. Table 3.1 shows the mapping of the index to the text area location. The next field *text* contains the text that should be drawn onto the area. The renderer that draws the texts does not warp the text when it is too wide for the area. The task to properly warp the text is up to the user. Finally, the background color of the whole area can be changed with the three fields *bg_red*, *bg_green* and *bg_blue*. These three values define the color in the red-green-blue color model.

Algorithm 3.4: Textfield message format

```
uint8 textfield
string text
uint8 bg_red
uint8 bg_green
uint8 bg_blue
```

index	location
0	top left
1	bottom left
2	top right
3	bottom right

Table 3.1: Mapping of index to the text field location

3.2.6 Buzzer

The task of the buzzer node is to make an acoustic signal. Because such an acoustic signal can be very distracting when it is not turned off, the message format is designed to mitigate this problem by having a duration field for how long the acoustic signal should be turned on. A second duration field indicates how long afterwards there should be no acoustic signal. This helps in creating patterns of acoustic signals. There is also a field for the frequency of the acoustic signal. The full message format can be seen in algorithm 3.5.

Algorithm 3.5: Buzzer message format

```
uint16 freq
uint16 on_duration_msec
uint16 off_duration_msec
```

3.2.7 Battery monitor

The battery monitor node controls the battery monitor circuit and periodically measures the battery voltage and checks if the battery is below the threshold given by the potentiometers on the extension board. It has to convert the code from the ADC to a voltage by multiplying it with a scaling factor. The formula to convert the ADC code to battery voltage is

$$U_{Battery} = \frac{code_{ADC}}{2^{10}} \cdot 4 \cdot 3.3\,\mathrm{V}$$

The code from the ADC gets first normalized. Because the ADC has a resolution of 10 bit the code is therefore divided by 2^{10}. Afterwards the normalized value is multiplied by 4 because of the resistor divider and finally it gets multiplied by reference voltage of the ADC which is 3.3 V.

Both values, the battery voltage and the battery low signal are send each second and use the message format in algorithm 3.6. Because the low battery signal should be a warning to the user and not an error this information can be send at such a low rate.

Algorithm 3.6: Battery monior message format

```
bool low
float32 voltage
```

3.2.8 Inertial Measurement Unit

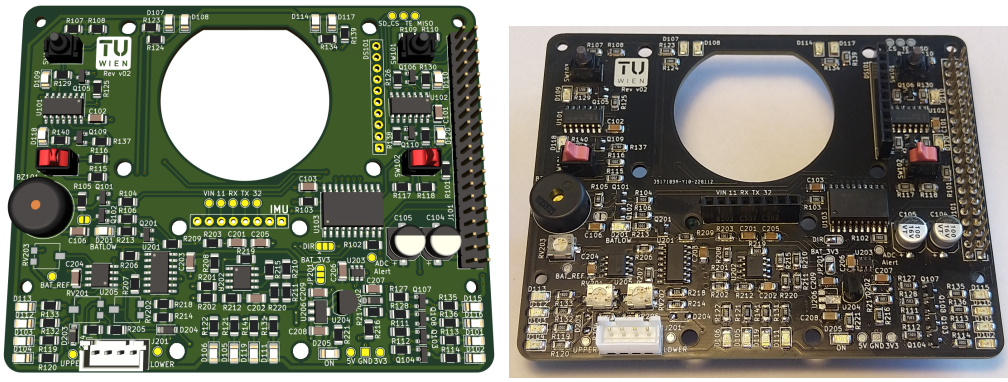
The ROS framework already includes a package with a driver node and firmware for the IMU. The firmware is called *Razor AHRS* (Attitude and Heading Reference System) and combines the measurement of the magnetometer, accelerometer and gyroscope with a sensor fusion algorithm. More information can be found in the ROS wiki page for the package ([raz]).

CHAPTER 4

Result

4.1 PCB Layout

The final PCB with all the hardware components from section 3.1 is shown in figure 4.1. This PCB has the cutout for the fan inlet so that the airflow of the fan is not obstructed. The fan cutout has a diameter of 37 mm and is cropped at the upper edge of the PCB. The crop is necessary so that the strip of material is wide enough to be still stable and to not break. The dimensions of the PCB are the same as the Xavier NX devkit, namely 100 mmx79 mm.



(a) (b)

Figure 4.1: Left: rendered image of the final PCB. Right: final assembled PCB

The preferred package for all electronic components is *1206* which is approximately 3.2 mmx1.6 mm. For the integrated circuits (IC) packages were preferred that have a wider gap between the pins. With this selection of packages the risk of solder bridges and other faults during assembly are reduced.

All the components that are connected to the 5 V power rail were placed close to the edge of the board. The 5 V power rail is then routed as a 0.75 mm thick trace around the board. The 3.3 V rail was routed as a filled polygon on the top layer. The common ground was routed as a filled polygon on the bottom layer of the PCB.

4.2 Power consumption

The Jetson Xavier NX devkit provides two power rails to the extension board. Each power rail can provide a limited amount of current. The 5 V power rail can provide up to 500 mA. Table 4.1 lists all components on the 5 V power rail with their current consumption. The total power consumption is 165 mA which is well below the limit.

component	current draw
LED indicator red	40 mA
LED indicator yellow	40 mA
LED indicator green	40 mA
LED indicator blue	40 mA
Buzzer	5 mA

Table 4.1: List of components with their current draw for the 5 V power rail

The 3.3 V power rail has a limit of 100 mA. Table 4.2 lists most of the components with their current draw and the sum of it is about 94 mA. Some components are not listed in the table, either because their current consumption could not be determined or is negligible. The total current draw is very close to the limit and it could also be above the limit for a short time. This cases can be tolerable if there are no other consumers attached to the Xavier NX devkit which are also powered by the 3.3 V power rail.

component	current draw
Button/Switch LED	43.33 mA
ADC	0.14 mA
IMU	20 mA
Display backlight	30 mA

Table 4.2: List of components with their current draw for the 3.3 V power rail

The extension board also has a third power rail for the battery monitor circuit. This power rail is powered by the battery and generates 3.3 V. The linear regulator has a current limit of 100 mA. Table 4.3 lists most components on this power rail. The components that are not listed have a negligible current consumption. The current draw of the battery low LED can be seen as halved because when the LED is active it will blink and would only draw current half the time. The total current draw is 12.5 mA which is reasonably low for a battery powered circuit.

component	current draw
Battery On LED	4.33 mA
Battery Low LED	4.33 mA
Power supply	6 mA

Table 4.3: List of components with their current draw for the battery powered power rail

With a current draw of 12.5 mA the linear regulator would dissipate 136.34 mW which should increase the temperature of the linear regulator by about 27 K. This temperature increase is well below the maximum temperature of the IC and the IC would not overheat.

CHAPTER 5

Conclusion and Outlook

In chapter 3 an extension board together with the software driver were presented which were successfully implemented and tested. The extension board solves or mitigates the problems and the software driver can operate the extensions board and integrate into the programming environment for the students.

The following problems and difficulties were solved or mitigated. This ensures that this work fits the needs for the course *Autonomous Racing Cars* and that it makes operating the racing cars safer.

The battery monitor ensures that the user is aware of the amount of energy that is stored in the battery and that a warning signal, a blinking LED and a beeping sound, warns of a too low battery voltage. Batteries that are discharged too low can be irreversibly damaged. Also there is a green LED that is always turned on when the battery is connected to remind the student to disconnect the battery when the racing car is not in use.

The buzzer, which was mentioned previously, can also be used by the software when the software detects a fault or when it needs the attention of the user. This can be combined with the LED indicators which can further indicate the fault or internal state. Some textual information on the display can further inform the user of the fault or even give some options for further operation. The buttons and switches are a way for the user to give an input to the software and change the configuration.

The layout of the PCB has some requirements that were met. The PCB had to fit on top of the Xavier NX development board. This meant that the fan inlet should not be obstructed and that the mounting holes match those of the development board. The large gap between the extension board and the development board also meant that the 40-pin header had to be connected with stacking connectors. Another requirement that was met was the placement of the IMU on the central axis of the model car.

During development some improvements were identified that would increase the safety, user experience and functionality of the model car. At first the user interface that is shown on the display could be improved and extended with more functions. For now it only displays text but it could also display graphics and diagrams for better visualization of the internal state or of the sensor readings.

The extension board is not protected against external forces, e.g. tools that are falling on it, crash with another model, etc. It could be protected with a cage or a case, which could also protect it against debris. Such an enclosure could be designed to protect the extension board and possibly also the Xavier NX development board.

Currently the extension board has only one sensor that is used for autonomous driving. It could be improved with more sensors that measure the surrounding environment or the movement of the model car. Some example sensor would be, ultrasonic distance measurement, radar or depth cameras. The car also does not have any means to directly communicate with other cars. A possibility might be to use ultra-wide band radio (*UWB*) for wireless communication and also to measure the distance between cars.

The battery monitor connects the battery power lines directly with the power lines of the extension board. The battery is also directly connected to the electrical motor and the motor driver module, which can induce noise on the power lines and can affect sensible sensors, as mentioned in suggestion above, that are connected to it. One solution is to add filters that reduce the unwanted noise signals. Another possibility is to electrically isolate the battery monitor circuit from the other circuits on the extension board. Isolating the battery monitor circuit also has the benefit to protect the other circuits on the extension board from electrical faults that could damage the extension board or even the Xavier NX devkit.

List of Figures

1.1 F1TENTH model cars as used in the course *Autonomous Racing Cars* . . 2

2.1 An overview of the peripherals used in [Vaj17] and how they are connected. Source: [Vaj17] 6

3.1 Previous model car from the course with all hardware components labeled. 7
3.2 Nvidia Xavier NX development kit. Source: [nvi] 8
3.3 Schematics of the LED indicators. This circuit is replicated four times, once for each color. 9
3.4 Schematic for debouncing a switch or a button and signalling it with an LED. 10
3.5 Schematic of the buzzer circuit. With the parallel configuration it can be controlled by two different signals. 11
3.6 Schematic of battery threshold trimmer 12
3.7 Schematic of the threshold comparator with RS latch 13
3.8 Schematic of a single stage oscillator with enable input. The values of R and C define the frequency of the oscillator. 13
3.9 Schematic of the battery power supply for the monitor circuit 14
3.10 The tool *Jetson-IO* showing the special function configuration for the 40-pin header 16
3.11 The five areas of the display content 17

4.1 Rendered and assembled PCB 21

List of Tables

3.1 Mapping of index to the text field location 18

4.1 List of components with their current draw for the 5 V power rail 22
4.2 List of components with their current draw for the 3.3 V power rail 22
4.3 List of components with their current draw for the battery powered power rail 23

List of Algorithms

3.1 LED message format 16
3.2 Button message format 17
3.3 Backlight message format 17
3.4 Textfield message format 18
3.5 Buzzer message format 18
3.6 Battery monior message format 19

Bibliography

[Dus19] Jan Dusil. Slip detection for f1/10 model car. Bachelor's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2019.

[f1t] F1tenth. https://f1tenth.org/. [Online; accessed 25-March-2022].

[Gue19] Nuno Miguel Santos Guedes. A robotic platooning testbed for cooperative its components. Master's thesis, Instituto Superior de Engenharia do Porto, 2019.

[nvi] Jetson xavier nx developer kit. https://developer.nvidia.com/embedded/jetson-xavier-nx-devkit. [Online; accessed 25-March-2022].

[raz] razor_imu_9dof - ros wiki. https://wiki.ros.org/razor_imu_9dof. [Online; accessed 25-March-2022].

[ros] Documentation - ros wiki. http://wiki.ros.org/ROS. [Online; accessed 25-March-2022].

[Vaj17] Martin Vajnar. Model car for the f1/10 autonomous car racing competition. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2017.