HTBLuVA Wiener Neustadt
Höhere Lehranstalt für Informatik

DIPLOMARBEIT

Decentralized Autonomous Traffic System

Ausgeführt im Schuljahr 2020/21 von:

Entwicklung und Konstruktion der Roboter sowie entwicklung einer grafischen Benutzeroberfläche
Alexander Brenner 5CHIF-02

Entwicklung der Blockchain und kommunikation zwischen den Robotern
David Fischer 5CHIF-03

Betreuer / Betreuerin:

Dr. Michael Stifter

Wiener Neustadt, am 20. April 2021

Abgabevermerk: Übernommen von:

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Wiener Neustadt, am 20. April 2021

Verfasser / Verfasserinnen:

[illegible] [illegible]

Alexander Brenner David Fischer

Contents

	HÖHERE TECHNISCHE BUNDES- LEHR- UND VERSUCHSANSTALT WIENER NEUSTADT
	Fachrichtung: Informatik

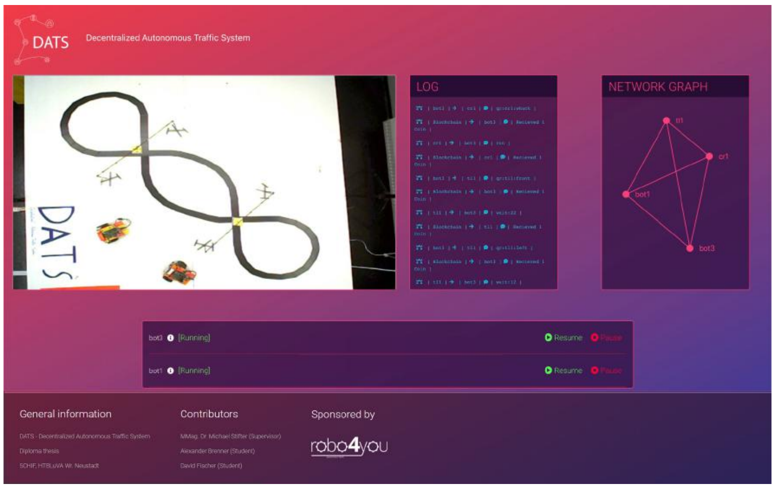
Diplomarbeit Dokumentation

Namen der Verfasser/innen	Alexander Brenner David Fischer
Jahrgang Schuljahr	5CHIF 2020 / 21
Thema der Diplomarbeit	Decentralized Autonomous Traffic System
Kooperationspartner	F-WuTS

Aufgabenstellung	Erstellen eines dezentralen Systems und autonom fahrenden Fahrzeugen, um die Realisierbarkeit solch eines Systems in Echt-Welt-Szenarios zu testen.

Realisierung	Blockchain mithilfe von Python und der Flask Library Fahrzeuge mithilfe des Hedgehog Raspberry Pi Schilds Frontend mithilfe von React.js und Docker

Ergebnisse	Funktionsfähige, effiziente blockchain Funktionsfähiger line follower Funktionsfähiges frontend

Typische Grafik, Foto etc. (mit Erläuterung)	User interface des DATS control panels 

DATS
Decentralized Autonomous Traffic System
LOG
NETWORK GRAPH
tl1
cr1
bot1
bot3
bot3 [Running] Resume Pause
bot1 [Running] Resume Pause
General information
DATS - Decentralized Autonomous Traffic System
Diploma thesis
5CHIF, HTBLuVA Wr. Neustadt
Contributors
MMag. Dr. Michael Stifter (Supervisor)
Alexander Brenner (Student)
David Fischer (Student)
Sponsored by
robo4you

Teilnahme an Wettbewerben, Auszeichnungen	Teilgenommen am ecoplus Innovation Award 2021

Möglichkeiten der Einsichtnahme in die Arbeit	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt

Approbation	Prüfer	Abteilungsvorstand
(Datum, Unterschrift)	MMag. Dr. Michael Stifter	AV DI Felix Schwab

	COLLEGE OF ENGINEERING WIENER NEUSTADT
	Department: Informatik

Diploma Thesis Documentation

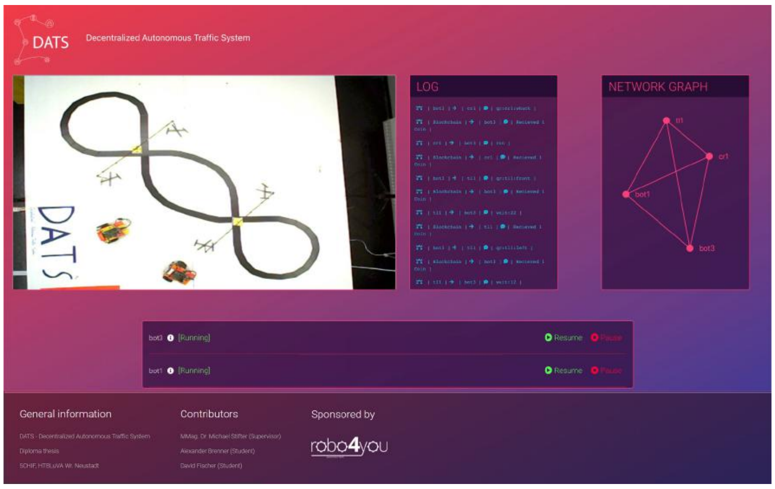
Authors	Alexander Brenner David Fischer
Form Academic Year	5CHIF 2020 / 21
Topic	Decentralized Autonomous Traffic System
Co-operation partners	F-WuTS

Assignment of tasks	Creation of a functioning decentralized system and autonomous vehicles to test the viability such a system in diverse real-world traffic scenarios.

Realization	Blockchain using Python and the Flask Library Vehicle using the Hedgehog Raspberry Pi Shield Frontend using React.js and Docker

Results	Lightweight blockchain Efficient line follower Modern frontend

htl bildung mit zukunft	**COLLEGE OF ENGINEERING WIENER NEUSTADT**
	Department: Informatik

Illustrative graph, photo (incl. explanation)	User interface of the DATS control panel 

DATS Decentralized Autonomous Traffic System

LOG

```
| bot1 | → | cr1 | 💬 | gr:cr1:back |
| Blockchain | → | bot1 | 💬 | Recieved 1 Coin |
| cr1 | → | bot1 | 💬 | [illegible] |
| Blockchain | → | cr1 | 💬 | Recieved 1 Coin |
| bot1 | → | tl1 | 💬 | gr:tl1:front |
| Blockchain | → | bot1 | 💬 | Recieved 1 Coin |
| tl1 | → | bot3 | 💬 | wait:22 |
| Blockchain | → | tl1 | 💬 | Recieved 1 Coin |
| bot1 | → | tl1 | 💬 | gr:tl1:left |
| Blockchain | → | bot1 | 💬 | Recieved 1 Coin |
| tl1 | → | bot3 | 💬 | wait:12 |
```

bot3 [Running] Resume Pause

bot1 [Running] Resume Pause

General information

DATS - Decentralized Autonomous Traffic System

Diploma thesis

5CHIF, HTBLuVA Wr. Neustadt

Contributors

MMag. Dr. Michael Stifter (Supervisor)

Alexander Brenner (Student)

David Fischer (Student)

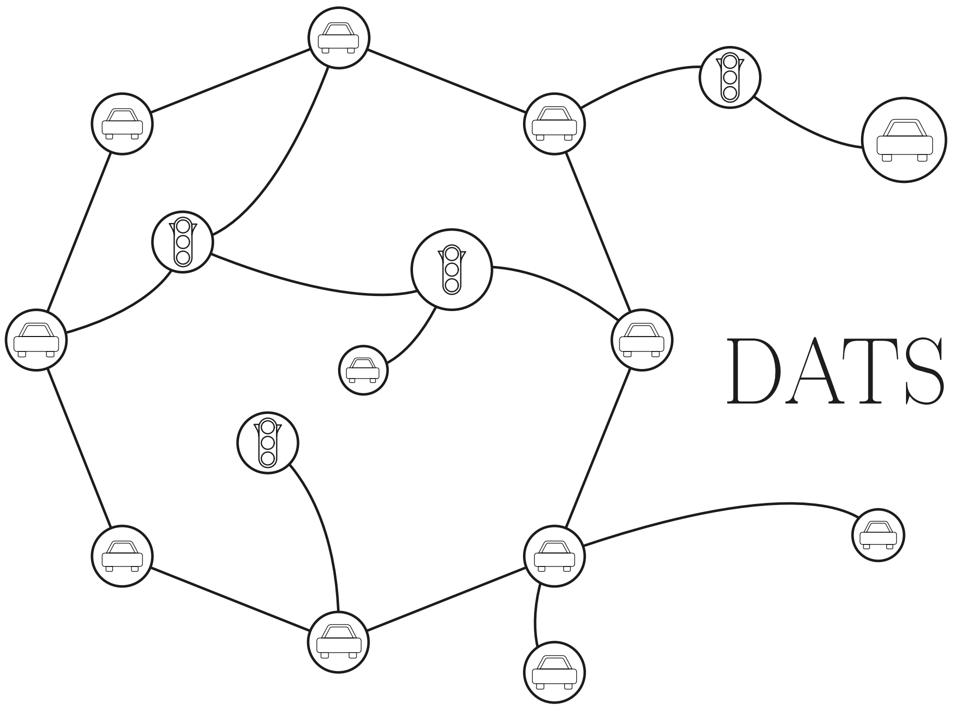
Sponsored by

robo4you

Participation in competitions, Awards	Participated in the ecoplus Innovation Award 2021

Accessibility of diploma thesis	HTBLuVA Wiener Neustadt Dr.-Eckener-Gasse 2 A 2700 Wiener Neustadt

Approval	Examiner	Head of Department
(Date, Sign)	MMag. Dr. Michael Stifter	AV DI Felix Schwab



DATS

Kurzfassung

Das "Internet of Things" und dezentrale Systeme wurden, aufgrund ihres enormen Erfolgs durch die unzähligen Einsatzzwecke, in den letzen Jahren immer bekannter und beliebter. Aufgrund des Höhenflugs des Bitcoin-Kurses und die überall herrschende Präsenz von IoT-Geräten kann man sagen, dass diese Technologien ihren Platz in unserer modernen Gesellschaft gefunden haben.
Der Fokus dieser Diplomarbeit war es diese zwei Technologien zu vereinen um ein blockchainfähiges System zu erschaffen, welches Transaktion zwischen IoT-Geräten effizient abwickeln kann.
Um dieses System einigermaßen praxisnah testen zu können, entschieden wir uns eine Verkehrssimulation aufzubauen. Die Autos fungieren nicht nur als IoT-Devices, sondern auch als Knoten in unserer Blockchain. Das "Decentralized Autonomous Traffic System (DATS)", welches zugleich das Ergebnis dieser Diplomarbeit ist, machte diese Vorstellung zur Realität.
Für die Realisierung eines solch raffinierten Systems, setzten wir auf eine Vielzahl an moderenen Technologien, darunter Docker und Flask. Zusätzlich zu dem IoT und Blockchain Teil des Projekts, entwickelten wir ein Front-End für Steuerung und Überwachung des Systems.

Abstract

Decentralized systems, as well as the Internet of Things, have, thanks to their success, become universally known in the past few years. With Bitcoin's price formally going through the roof and the abundance of IoT devices everywhere, it's safe to say that both technologies have found their place in our modern society.
Thus this diploma thesis' focus was laid on combining these aspects and designing a lightweight Blockchain capable of efficiently handling transactions between low-power IoT devices. To test this Blockchain in a real-world application, we decided on a traffic simulation, in which cars and other elements, are seen as IoT devices, as well as nodes in the Blockchain.
Making this concept a reality was done with the Decentralized Autonomous Traffic System (DATS), which is this diploma thesis' result.
To realize a system as sophisticated as DATS, we relied on some modern technologies, like Docker and Flask. In addition, besides the Blockchain and IoT elements of this project, an intuitive frontend was also developed using React.

Chapter 1

Introduction

Author: David Fischer

Over the last few years, transportation has had a huge leap regarding its technology. Companies have started progressively rolling out software and hardware to their cars, which allow automatic breaking,[1] adaptive cruise control,[2] and lane keeping assistants.[3] While these features are mostly still a luxury and come with a huge markup in the vehicle's price, most modern cars carry them within their standard equipment. A company that stands out is Tesla, which has developed a fully fledged autopilot.[4]
Which leaves the question: What's next? With the evergrowing 5G network, communication between devices has been made faster and easier. The combination of a fully functioning autopilot with communication between vehicles could yield limitless potential in avoiding accidents, controlling traffic lights based on the exhaustion of the road, removal of redundant road signs at intersections, and so much more.

1.1 Goal

This diploma thesis aims to test the viability of a decentralized system on the road. It enables autonomous model cars to communicate with each other and other traffic elements, such as intersections and traffic lights, to avoid accidents, bad behavior by drivers, and create smooth traffic while remaining fast, scalable, and efficient.
To maintain an overview of all of the communication happening between the vehicles, a management platform is hosted on a webserver. A flat design, as well as expertly placed iconography, create a smooth user experience while being easy on the eyes. The platform is also able to create bad behavior, such as forcing vehicles to disobey orders from other cars.
Combined, this stack of software and hardware yields a complete simulation of decentralized traffic, as well as a system to monitor its actions.

[1] savecar.gov, "AEB: Automatic Emergency Braking".
[2] AUDI, "adaptive cruise control with stop and go function".
[3] Mercedes-Benz, "What is: Active Lane Keeping Assist."
[4] Tesla, "Future of Driving".

1.2 Motivation

The F-WuTS robotics laboratory lacks an interactive demonstration of both its student's knowledge as well as its equipment. Simulating real traffic while factoring in decentralization and computer vision makes for a great project, and thanks to its rather small form factor, can be moved to various events.

1.3 Challenges

1.3.1 Viability of Decentralization in a Traffic Simulation

This approach has been theorized multiple times but never realized.[5] Therefore we only have words to back up our project, rather than real examples.
Writing a blockchain, efficient enough to handle fast transactions, and putting it into practice in simulated traffic might not be sufficient.

1.3.2 Stability of Custom Parts

Since most of the robot is 3D printed, the parts might not hold up over time. Wear on the parts, heat generated by both the microcontroller and its battery could harm the components and their structural integrity.

1.3.3 Fragility of Botball Motors

Due to previous experience, we can safely determine that Botball motors do not have a very long life expectancy. They usually wear out after about 12 hours of driving, and either lose strength or stop working altogether. Loss of strength might not seem bad, but as soon as both motors start showing symptoms, replacements are required.

1.4 History of Decentralization and Blockchain

Blockchain, as well as decentralization, saw its first major breakout a few years after the release of Satoshi Nakamotos Bitcoin whitepaper.[6] While decentralization has been around for a long time, its use by the general public has been limited at best.[7] Today, even after Bitcoin's explosion in both hype and price, centralized applications still maintain a monopoly over the web while decentralized applications, like Ethereum DApps,[8] are only finding their footing. While this might be the case today, with the rapid acceleration of technological advancements, decentralized applications have a bright future ahead of them.

1.5 History of the Internet of Things

The Internet of Things is ambiguous in today's society. It is found in homes, workplaces, and almost everywhere else. Most commonly IoT devices include lightbulbs, electrical outlets, webcams, and virtual assistants, such as Google Home[9] or Alexa.[10] However, one of the

[5] Tlig, Buffet, and Simonin, "Decentralized traffic management: A synchronization-based intersection control".
[6] Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System".
[7] Elliott, "A Brief History of Decentralized Computing".
[8] Kaidong Wu, "A First Look at Blockchain-based Decentralized Applications".
[9] Gebhart, "Everything you want to know about Google Home".
[10] Clauser, "What Is Alexa (and Whats the Best Alexa Speaker?"

first IoT devices can be dated back to the early 1980s, where students at the Carnegie Melon University connected to the local Coca Cola machine to check on its beverages and their temperatures.[11] The first webcam is similarly sourced out of laziness, used only to monitor the coffee pot.[12]
IoT devices can be anything including cars, which is why the concept is a central part of this thesis.

[11] Teicher, "The little-known story of the first IoT device".
[12] Kesby, "How the world's first webcam made a coffee pot famous".

Chapter 2

Project Management

Author: David Fischer

2.1 Kanban

Kanban was developed as a system to improve work across human systems, allowing easy identification and handling of system-level bottlenecks. Originally based on the Toyota Production System, Kanban aims to improve the general workflow and simplify tasks. To get a better idea of the concept, Kanban, also known as 看板, means signboard or billboard in Japanese.

2.1.1 Tasks

Tasks are split up and placed on a Lane of the Kanban board. Tasks contain a title, a simple description, and a person assigned to the task. Optionally there are also deadlines, sub-tasks, and a label.

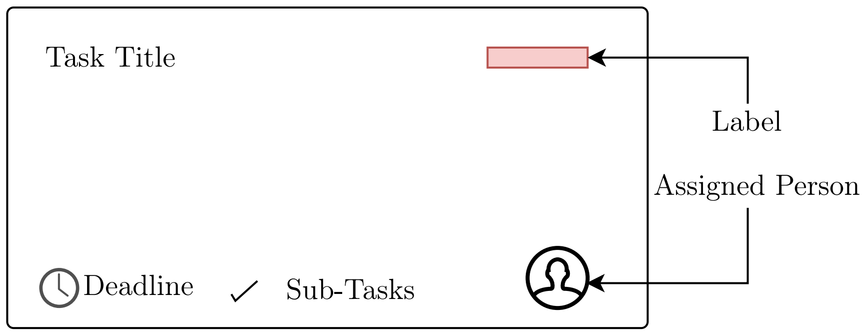


Figure 2.1: Sketch outlining the elements of a Kanban Task

In our case, the labels are used to differentiate sections of the project, for example, blue is used to highlight tasks related to the blockchain, while red tasks are related to the robots.

2.1.2 Lanes

We decided that 5 specific lanes fit our development process best. All of the lanes serve their specific purpose while making it easy to identify tasks stuck in their current position.

Backlog

The backlog generally contains Tasks that have been decided on and are planned for the future. This includes anything from new features to bug fixes.

Todo

Todo, similar to the backlog, includes tasks that are planned for the near future. Having the backlog lane separated from the todo lane makes sense considering the overall visibility of the tasks.

In Progress

As stated in the name, tasks in the "In Progress" lane are currently being worked on. Tasks in this lane usually require one to two weeks to complete.

Review

The "Review" lane makes the development progress easier for everyone involved. Other members of the team inspect the results of the Tasks, making themselves familiar with the changes made. This helps in the long run since everyone has a small amount of knowledge in every subject area.

Done

Tasks, which are completed and reviewed are moved to the "Done" lane.

Chapter 3

Methodologies

3.1 Choosing means of Communication

Author: David Fischer

Achieving secure communication in the Internet of Things has been significant ever since its inception. Over the years more and more security flaws have been found in IoT systems, which prompted many people to look for options to secure their devices.[1] Blockchain is hailed as one of the solutions to this problem. The focus of this section lies in exploring this assumption and the blockchain's many applications.

3.1.1 Other Means of Communication

Other communication methods include centralized servers, web requests, and IoT hubs, which communicate using radio frequencies. All of these options, unfortunately, have too many downsides, which outweigh the positive aspects.

Centralized Servers

Centralized Servers, while reducing the IoT device's computing requirements, are still able to be exploited. IoT devices are mostly placed in homes, where data like temperatures, who currently is home, various video feeds, and other data is recorded. When this data is stored on the companies servers it is subject to be leaked. This has happened to one of the most prestigious tech companies, Google.[2]

Web Requests

Unsecured web requests on IoT devices provide easy access for people with devious intents. Most IoT devices either ship with default credentials or don't have any at all.

IoT Central Hubs

IoT Hubs usually feature technology that broadcasts low-rate wireless personal area networks, or LR-WPANs, which devices like lightbulbs or smart electrical outlets connect to. ZigBee uses the IEEE 802.15.4[3] standard to communicate. Potential interference and its maximum ten-meter broadcast length limit this approach's potential.

[1]Mandrita Banerjee, "A blockchain future for internet of things security: a position paper".
[2]Tilley, "How Hackers Could Use A Nest Thermostat As An Entry Point Into Your Home".
[3]IEEE, "IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks".

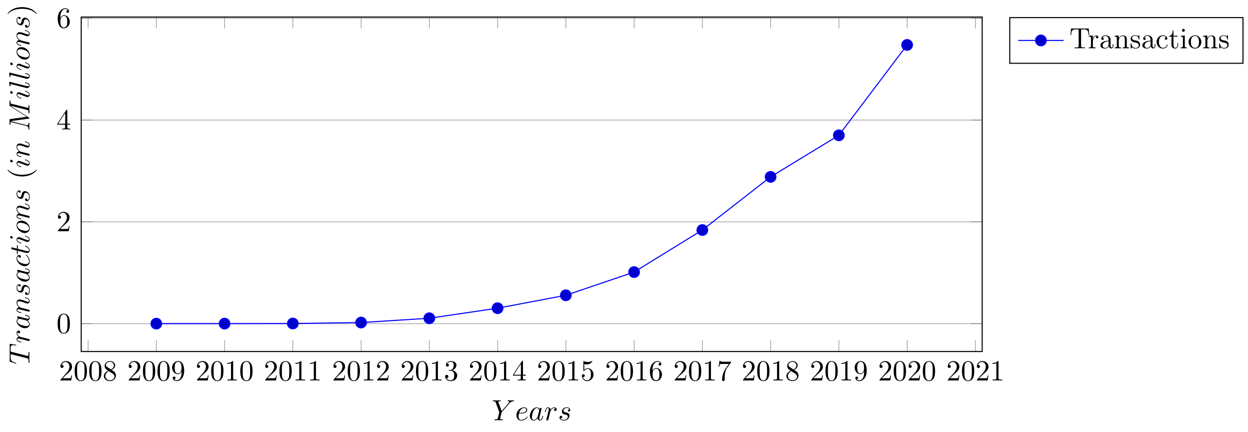
3.1.2 Why Blockchain

Blockchain is a prime example of decentralized architectures and systems. Blockchain can be generalized as a data storage solution, which is shared among every individual node in the Blockchain. Blocks store records, which can be any type of information, for example, a message. Records are verified by the network, after which they are appended to a Block. Blocks are then tied together using hashes, which is a layer of security and welds the blocks together, forming a Blockchain. A visual explanation of this can be found online.[4]
The main difference to centralization is the independence of multiple nodes, which is only achieved in decentralized systems. In centralized systems information is controlled by a single entity, which can lead to problems. Decentralized systems have the advantage of information being spread out, which makes it very hard for a bad player to manipulate data.
This extra layer of security in our system, which is also scalable, made us confident, that a system like this could be viable in the future.

3.1.3 Bitcoin Explosion

Ever since the rapid growth of Bitcoin's[5] value[6], there has been a definite hype around cryptocurrencies and Blockchain. Bitcoin's price can be compared to stock prices, so in a speculative market, price is determined by supply and demand. There are many theories as to why Bitcoins worth skyrocketed, one of them being market manipulation.[7]

Statistic 3.1: Total number of transactions on the Bitcoin network



3.2 Comparing Options

The goal of this section is to find a lightweight Blockchain which is capable of running on microcontrollers, which is the central component of our robots.[8] Finding the optimal blend between efficiency and scalability is key.

[4]Matthew Weber, "A REUTERS VISUAL GUIDE: Blockchain explained".
[5]Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System".
[6]blockchain.com, *Total Number of Transactions on the Blockchain*.
[7]John M. Griffin, "Is Bitcoin Really Un-Tethered?"
[8]Koza, "Welcome to Hedgehog!"

3.2.1 Ethereum DAPPS

Ethereum DAPPS[9] are web applications, supported by Ethereum's smart contracts. Instead of relying on a single server to host the application, DAPPS can use the Blockchain for its program logic.[10]
Generally, DAPPS offers the same kind of service, which is offered by cloud providers such as AWS. Thanks to Ethereums decentralized approach, data still belongs to the user and not the corporation the storage is rented from. Unfortunately, as seen in Table 3.1, storage, and transaction costs on the Ethereum network exceed traditional providers almost a hundredfold.[11] That is due to Ethereum still being based on a ”Proof-of-Work” system, which is an algorithm that confirms transactions and creates new blocks on the chain.[12]

Platform	**Write Speed**	**Sync speed**	**Storage cost**	**Write cost**
ETH DAPPs	15.6 tps	6.2 KB/s	\$460,800 / GB	\$60,000 / 1M tx
AWS	34,888 tps	5800 KB/s	\$0.023 / GB / month	\$4.25 / 1M tx

Table 3.1: Ethereum DAPPs compared to the AWS Berkley storage solution

3.2.2 Bitcoin Lightning

Bitcoin Lightning[13] aims to fix one of Bitcoin's biggest downsides. The Bitcoin Blockchain is only able to process around seven transactions per second, which means that it is not efficient.

Network	**Transactions per second (capacity)**
Visa	65,000 tps[14]
Bitcoin	7 tps

Table 3.2: Average Transactions per second

Bitcoin Lightning is a second layer routing network, which allows participants to transfer bitcoin without making their transactions on the Blockchain. This is achieved with so-called payment channels, which are ledger entries on the Blockchain.

[9]Kaidong Wu, “A First Look at Blockchain-based Decentralized Applications”.
[10]ethereum.org, “Use Ethereum Applications”.
[11]Visa, “How inefficient are dapps?”
[12]Madeira, “Ethereum 2.0 Likely to Affect DeFi and DApps With PoS Introduction”.
[13]Joseph Poon, “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”.

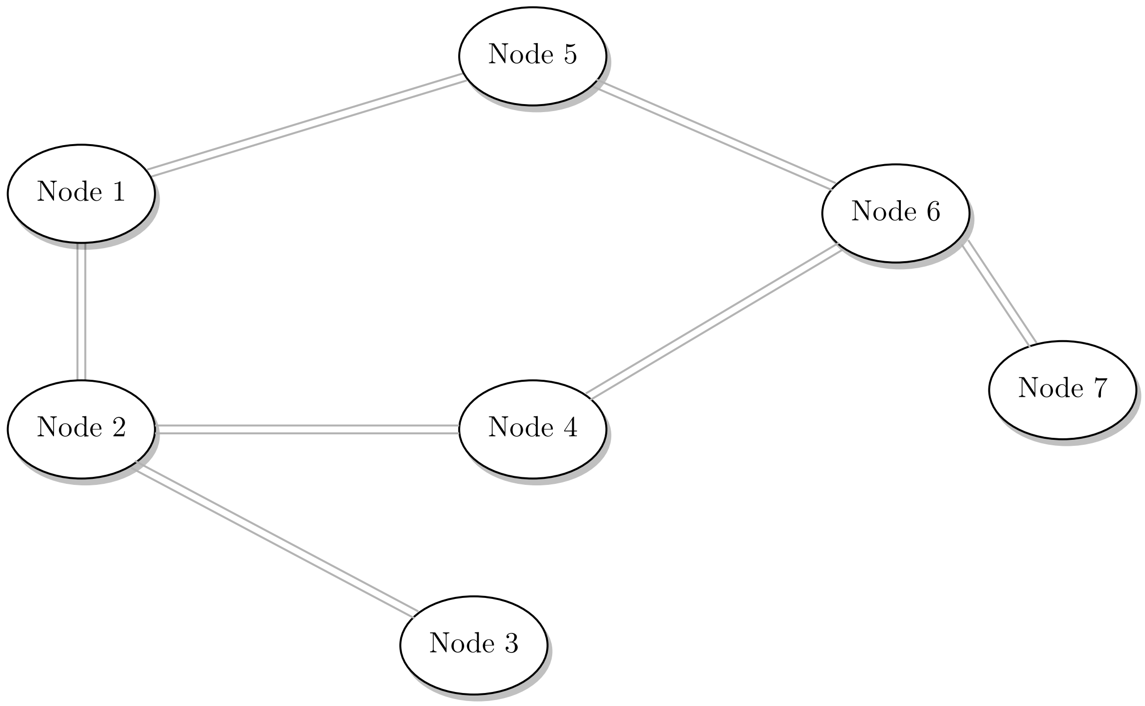


Figure 3.1: Example of a Bitcoin Lightning network

Unfortunately, Bitcoin Lightning is still not proven to be as good as its maximum theoretical efficiency. It would also imply opening a large number of lightning channels on our nodes, which is costly. Therefore Bitcoin Lightning is not an option for our system.

3.2.3 DIY Blockchain

Writing a custom blockchain helps eliminate most of the bulk and grants greater control. With the help of lightweight libraries, as well as efficient code, a fast blockchain delivers results, which pre-existing Blockchains and their respective interfaces for custom applications can't achieve. While custom made blockchains come with other disadvantages, the ability to personalize certain aspects and being able to change every part outweigh the negative aspects.

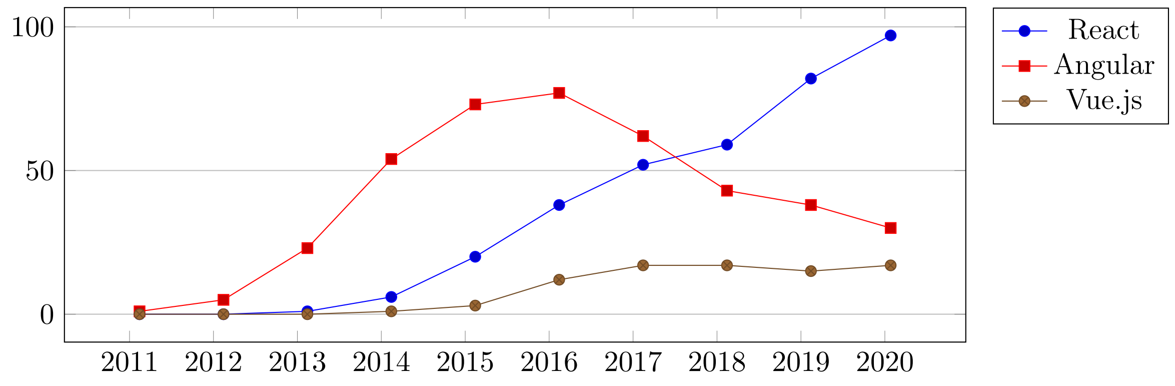
3.3 Application to Monitor and Control Aspects of the Blockchain

Author: Alexander Brenner
To create a simple and centralized solution to control and monitor the system we decided on a web application. The platforms primary objective is displaying the blockchain's transactions, as well as other details. Furthermore, the web app should be able to give an overview via a livestream. To aid the development of the whole system individual actions should be controllable using the web interface. To sum up: the application is not only meant for monitoring and controlling the system, but it's also a key component during debugging.

3.3.1 Comparing Web Frameworks

Since the User-Interface should be platform-independent we decided on a web application, since virtually any device can access a website, provided a web browser is installed. Due to the abundance of modern web frameworks[15], we narrowed it down to three possibilities. Their most important aspects being their dependencies, the documentation, the actuality of the framework, and exclusive features.

Statistic 3.2: Google Trends Statistic of Framework Popluarity



3.3.2 Angular

Angular[16], a framework developed by Google, was first published in 2016 and has received frequent updates. Angular, among other things, provides a great number of premade components meant for building web applications. Angular's biggest advantages are its high speed and the huge selection of responsive templates. The framework's high performance can be credited to its use of "Code Splitting" as well as "Code Generation". Angulars primary downside are its dependencies, which take up a lot of storage space. Nevertheless, Angular is a well-documented Framework and has one of the largest online communities.

3.3.3 Vue.js

Vue.js[17] was created by developer Evan You and first published in 2014. The framework's most important aspects are its minimalistic structure and flexible deployment. Due to its modular nature, Vue.js can be built upon easily. Thanks to the frameworks detailed documentation and simple structure, it is easy to learn and a great fit for beginners.
Vue.js applies the "Model View View Model" (MVVM[18]) design pattern, which provides a separation between the program logic and the user interface. MVVM could also be described as an efficient variant of the "Model View Controller" (MVC) design pattern. Summed up, Vue.js' small size combined with its implementation of MVVM makes for fast and efficient web applications.

3.3.4 React

With React[19] being released in 2013, it is the oldest framework we took into consideration, but it still managed to outperform our other choices. Backed by Facebook, React is still one of the most popular frameworks. One of Reacts best aspects is its portability. A new React project, including dependencies, is about 450MB in size, while an Angular project is about 750MB. React is very well documented, but specific issues might require some

[15]Google, "Google Trends".
[16]Google, "One framework. Mobile and desktop."
[17]Evan, "The Progressive JavaScript Framework".
[18]John, "MVVM as Design Pattern".
[19]Facebook, "A JavaScript library for building user interfaces".

research. React might not be the most beginner-friendly framework, but given enough prior knowledge, it is the right fit for our use case. React excels thanks to its numerous features, like the Component Lifecycle, Component States, and Conditional Rendering.

Component Lifecycle

Every React component has a so-called ”Lifecycle”, which refers to the components state at a certain point in time. The Lifecycle's primary objective is creating an efficient order of initialization during the webpages load time. In our case, this method could be used to fetch a list of the connected robots and initialize a connection to the livestream endpoint. To better illustrate the Component Lifecycle, Wojciech Maj has created an interactive graphic.[20]

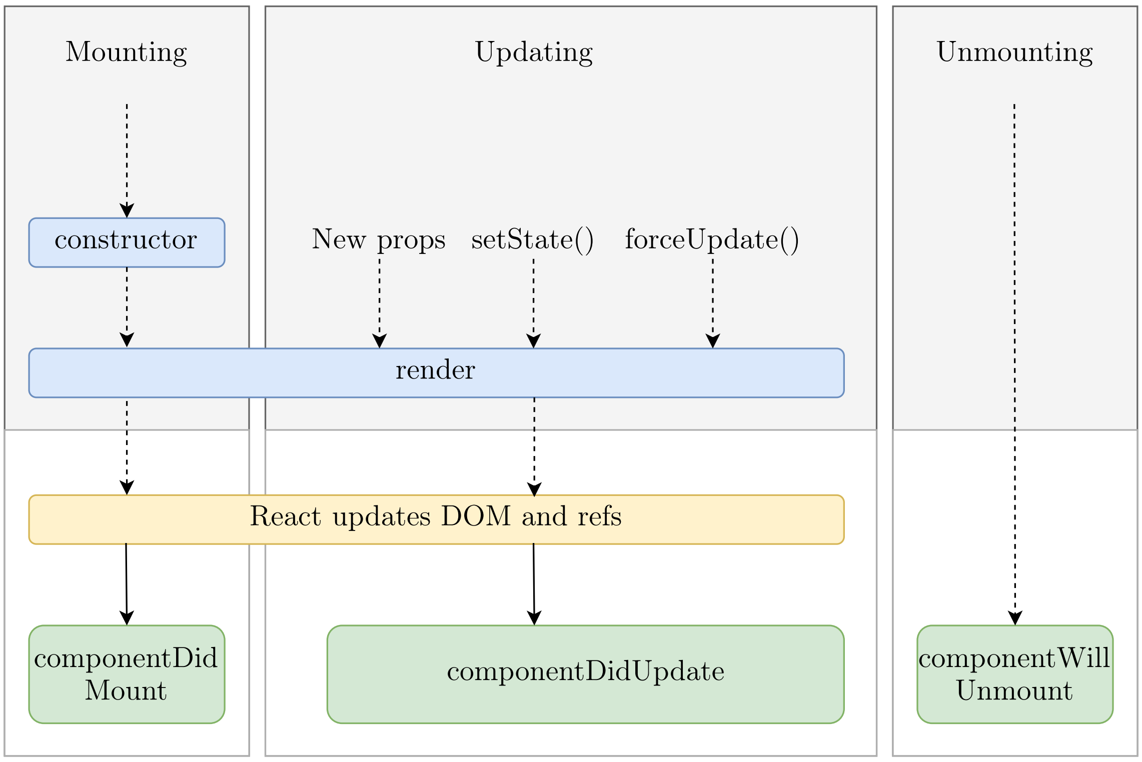


Figure 3.2: Reacts Lifecycle

Component States

React components feature states, which is local storage within the component. As soon as the state changes, the component is refreshed, which means that information changing does not force the entire page to reload.[21] For a single page application, states are the optimal choice.

Conditional Rendering

Conditional rendering provides an elegant way of dynamically loading components, based on information.[22] There is a great number of applications for Conditional Rendering, in

[20]Maj, “Interactive React Lifecycle Methods diagram”.
[21]Facebook, “React: State and Lifecycle”.
[22]Facebook, “React: Conditional Rendering”.

our case, this would be dynamically displaying nodes in a list. Conditional Rendering is also available in other frameworks, however, we find Reacts implementation to be the most polished.

Chapter 4

Designing and Constructing a Robot

4.1 Introduction

Author: Alexander Brenner
Since the robots are supposed to represent real cars, we decided to construct them in that manner. To make the construction as simple as possible, the robot was split up into an underbody, a tire fixation, and space for the robot's electric components. This general construction approach was inspired by the "Botball" Competition[1], which requires robots to be as simple as possible while remaining efficient.

4.2 Concept of the Botball Competition

Botball is an international competition in which students can demonstrate their knowledge of robotics. The tournament had its origins in 1997 and was started by the KISS Institute for Practical Robotics in the United States of America. As of 2012, Botball is also represented in Europe, thanks to PRIA[2], the Practical Robotics Institute Austria.
The Botball competition is based on a so-called "Game Table", which has tasks that are autonomously completed by up to two robots. The table is split up into multiple sections, each containing different tasks. Each year a new Table is introduced with different tasks and sections. The tasks mostly involve moving objects into other zones to gain points. Furthermore, tasks are mostly based on scenarios, most recently a moonbase.[3] Allowed components, as well as their quantity, are predefined, these include LEGO bricks[4], special metal parts[5], IGUS components[6], motors, servos, sensors[7] and a microcontroller, currently the Wombat[8], developed by KIPR. Over the years students around the world have come up with innovative solutions and complex constructions, despite the restrictions. Thankfully this project is not bound to these rules and a restrictive solution is not needed. In the Botball youth competition, the "Hedgehog"[9] microcontroller is commonly used, often featuring simpler overall construction, which is why they serve as the base for the robots.

[1]KIPR, "botball".
[2]Koppensteiner, "PRIA".
[3]Goodgame, "2020 Botball Game Review".
[4]KIPR, "2020 Botball Lego Parts".
[5]KIPR, "2020 Botball Metal Parts".
[6]KIPR, "2020 IGUS Parts".
[7]KIPR, "2020 Botball Electronics Kit".
[8]KIPR, "2020 Botball Wombat Controller".
[9]Koza, "Educational Robotics Controller: Hedgehog".

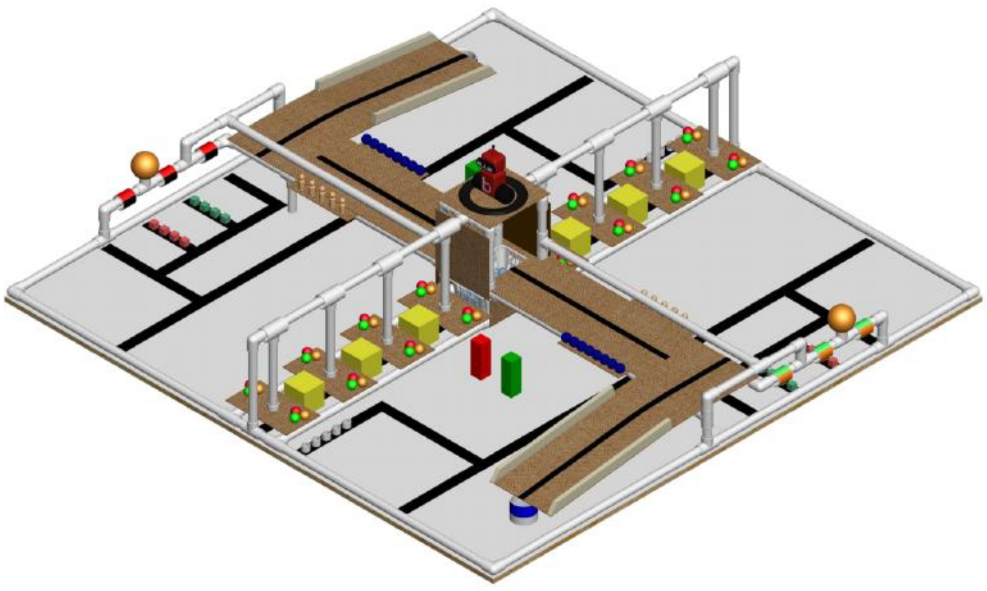


Figure 4.1: The official 2020 Botball Game Table

4.3 Hedgehog Controller

The Hedgehog's basic concepts stem from an early idea from as soon as 2012. This first version of the Hedgehog features an android phone as the controller. It featured an application as an integrated development environment which was later changed to a web-based environment. The Hedgehog's intent is, primarily, giving young students a platform to learn mobile robotics. The Hedgehog uses a Raspberry Pi as its microcontroller with the "Hedgehog Shield" attachment, which is plugged into the GPIO pins. The shield features multiple ports for diverse attachments like motors, servos, and sensors. Since the Hedgehog has multiple USB ports, it is also ideal for USB devices like cameras. As has been pointed out, the Hedgehog has a simple to use, web-based, development platform. Thanks to the number of convenient features and the Hedgehog's overall simplicity, it is perfect for this project.[10]

4.4 General Construction

As mentioned in the introduction, the construction is supposed to be kept as simple as possible. The chassis from the Botball kit fits this purpose pretty well, but there are some minor adjustments to be made. Furthermore, due to the chassis price, combined with import costs, we opted for a cheaper solution.[11] Thanks to the fantastic equipment, provided by F-WuTS, we can 3D print our parts. Thanks to these options, fine-tuning parts is a cheap and easy process. Achieving locomotion is done with two DC motors, with LEGO wheels, which are mounted on both sides of the robot. To stabilize the construction a "Caster Wheel", a marble fixed inside of a plastic case with its bottom exposed, is mounted on the back. The identification of obstacles is done with a camera, mounted on the front of the robot. Finally, the Hedgehog is screwed into the center of the underbody.

[10]Alexander Brenner, "Hedgehog vs Wallaby - Pros and Cons".
[11]KISS, "Chassis Bracket (for Wombat)".

4.5 Tire Fixation

4.5.1 Shaping the Tire Fixation

To mount the motors, a rectangular construction, similar to the ones found in the Botball metal kit[12], is required. Due to the vast number of available CAD shaping programs, we decided to use Cinema 4D[13], due to prior knowledge. Cinema 4D, developed by Maxon, features a user-friendly design, while still maintaining a great amount of functionality. To ensure a secure mounting point on the underbody, screws make for the best option. Other options would include adhesives, which would not hold up between the PLA filament over long periods of time.

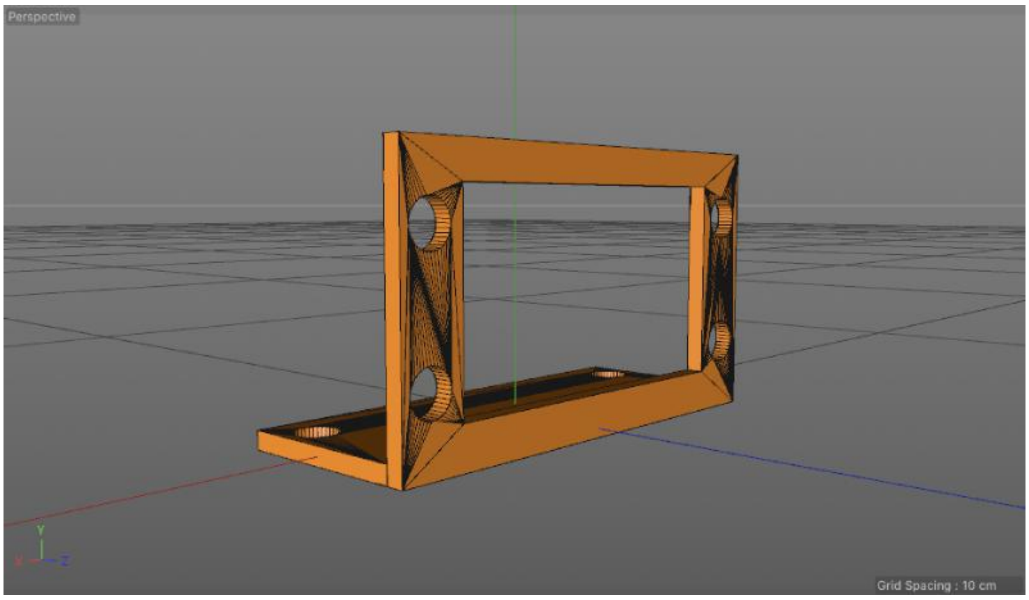
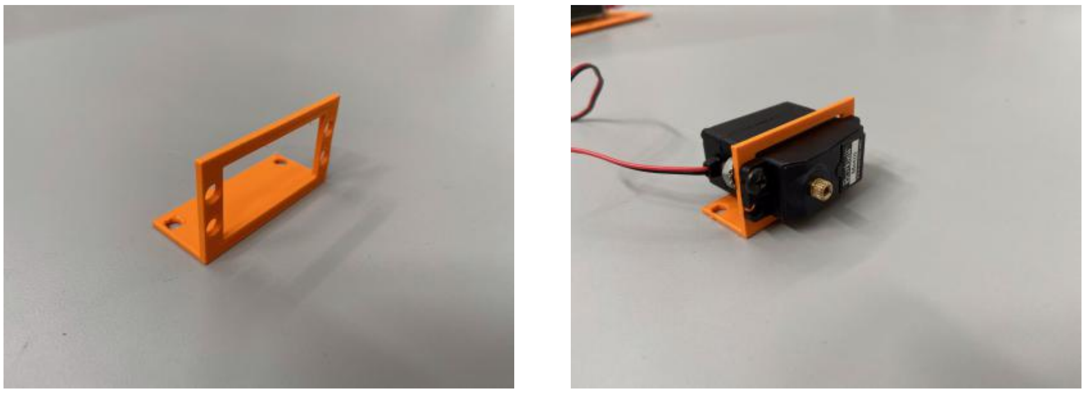


Figure 4.2: CAD Model of the tire fixation

4.5.2 Printing the Tire Fixation

The Cinema 4D model is exported as a standard triangle language file, or short stl. These stereolithography files are then sliced in Ultimaker Cura[14], a software that splits up stl files into layers and exports instructions for the 3D printer. To save time, up to four fixtures are printed at the same time. As mentioned, we were supplied with great equipment, including a Creality Ender Series 5[15], on which the parts are printed.

[12]KIPR, "2020 Botball Metal Parts".
[13]MAXON, "Why Cinema 4D?"
[14]Ultimaker, "Ultimaker Cura".
[15]Creality, "Ender 5 Pro 3D".



(a) Printed Tire Fixation **(b)** Printed Tire Fixation with Botball Motor

Figure 4.3: Printed Tire Fixations

4.6 Underbody

4.6.1 Shaping the Underbody

Similar to the tire fixation, the underbody was also shaped in Cinema 4D. The model being a simple rectangle with a slight elevation. The holes, allowing diverse components to be mounted on the robot, are placed precisely, using a bool object. This object allows one to subtract one object from the other, allowing easy placement of the holes. Using cylinders, scaled correctly to fit real-world screws, the underbody was shaped to our liking.

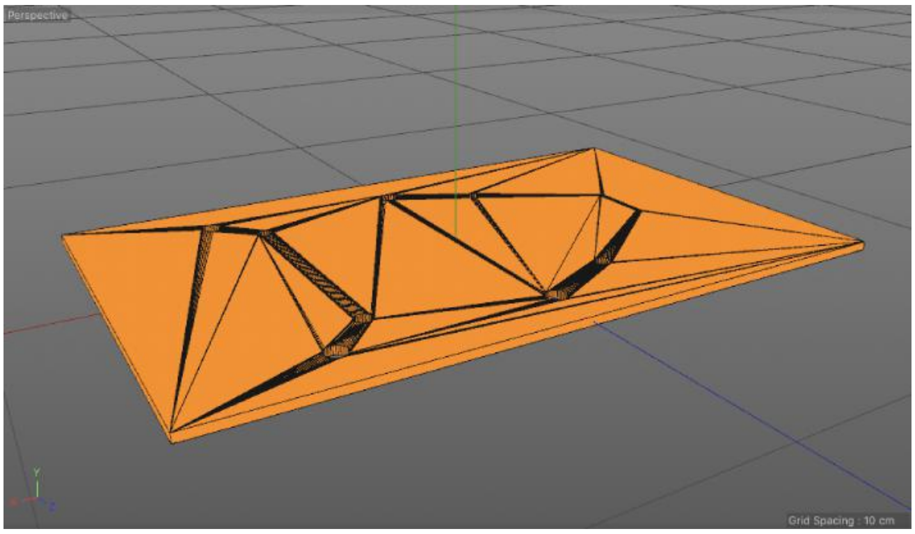


Figure 4.4: Underbody CAD Model

4.6.2 Printing the Underbody

As mentioned, Ultimaker Cura is used to convert the files exported from Cinema 4D to gcode files. Cinema 4D, by default, exports the objects on a cm-scale, while Ultimaker Cura uses an mm scale. Therefore the objects had to be scaled up tenfold to correctly measure up in the real world.

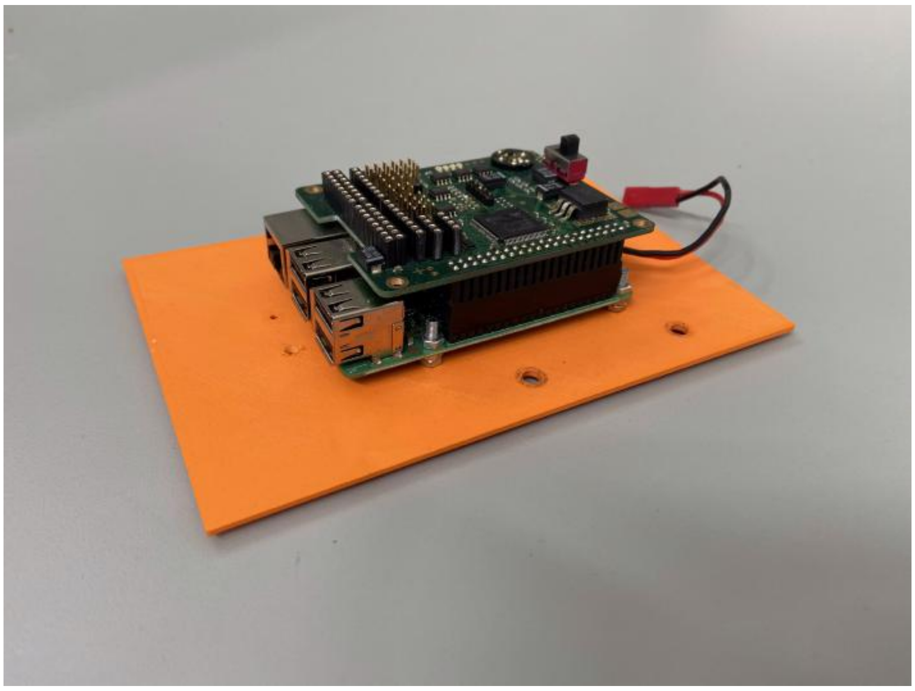


Figure 4.5: Printed Underbody with a Hedgehog Controller mounted on top

4.7 Assembling the Robot

After printing all of the parts, assembly is simple. As mentioned, the holes in the underbody serve as mounting points for both the Hedgehog microcontroller as well as the tire fixations. The caster wheel is mounted on the back using two small holes. The camera is mounted on the front. Lastly, all the cables are hooked up and the robot's construction is finished.

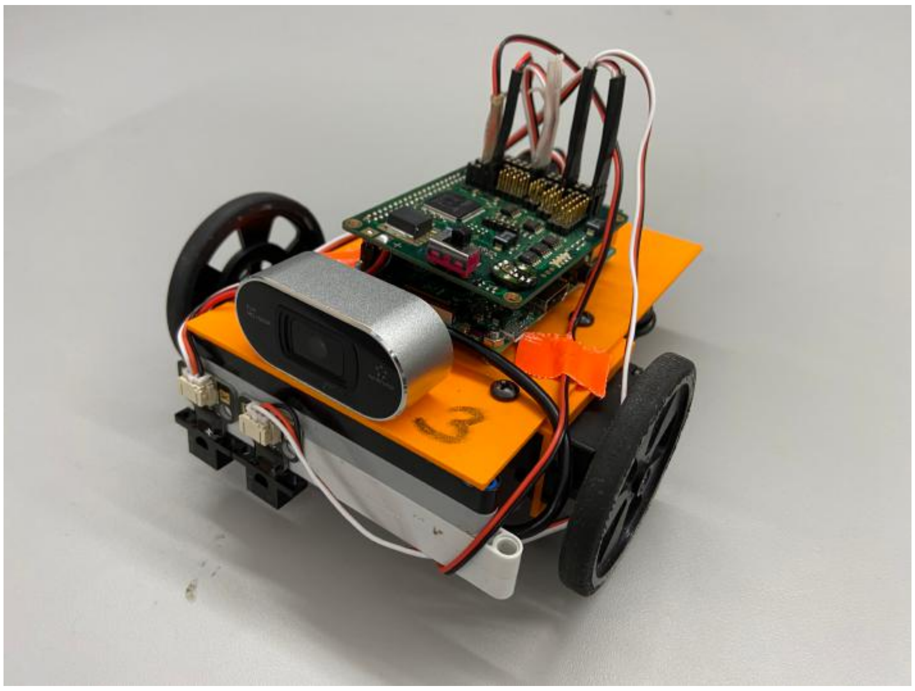


Figure 4.6: Assembled Robot, complete with a webcam

4.8 gcode

4.8.1 Introduction

After shaping the models they had to be converted into a format readable for a 3D printer. The aforementioned software Ultimaker Cura not only for quick adjustments to be made but exports complete models as .gcode files, adjusted for the printer. This format contains information relevant to the 3D printer, including the printer's nozzles movements. Gcode overall is a very important concept in the modern, robotized, and automized industry, as well as for hobbyists.

4.8.2 General Information

Gcode comes in various forms, all bound by one standard. The most widespread form of gcode is known as Computer Numerical Control, also known as CNC. Generally, gcode is used to control diverse machines, in our case a 3D printer, by specifying different tasks. Our school also utilizes a form of gcode, more specifically our engineering department, which controls its diverse CNC machines using a form of gcode.

4.8.3 The gcode File Structure

As mentioned, gcode contains information relevant to diverse devices, which allows them to complete certain human-defined operations. In our case, the 3D printer is fed its next position until the print is complete. The main disadvantage of this system is that not every machine is the same, therefore gcode files need to be configured to fit the machine's specific needs. Similar to other languages, gcode includes language-specific instructions. One of the utmost important instructions in our case is $G1$, which controls the printer's Nozzle, meaning it manipulates its current coordinates (x, y, and z). Another command, E, controls the Extruder, more specifically how much, and when the filament is discharged. These commands can be used in conjunction, achieving, for example, a line of filament. More precise printing is achieved with the F command, which controls the next movement's speed.[16]

[16]Soares, "3D Printer G-CODES".

Chapter 5

Designing and Constructing a Test Environment

5.1 Introduction

Author: Alexander Brenner
The project's findings are tested, as well as measured, on a custom-made table. We, again, took inspiration from the Botball Competition. The general idea was a two by two-meter area, covered with lines to guide the robots. Additionally, a construction is required to enable a birds-eye view of the table, using a camera mounted above.

5.2 Table-Layout Conception and Structure

Since a substantial part of the project engages with intersections and traffic lights, these situations are featured on the table. Therefore a parkour layout, featuring a simple design, was chosen to allow robots to travel between the two landmarks quickly.

5.2.1 Intersections and Traffic Lights

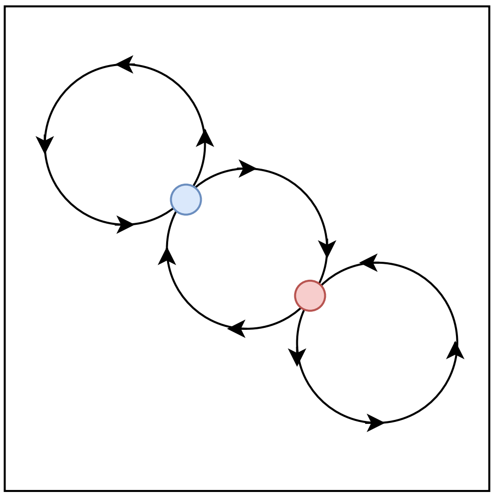


Figure 5.1: Sketch outlining the general outline of paths on the table

Both intersections are marked using either red or blue. The intersection marked blue is controlled using a regular traffic light system. The robots are presented with two options, green or red, which is derived from real traffic situations. The red junction is controlled using a system similar to the system used in the United States of America, whoever comes first, drives first. However, in this project's case, a queue is stored in the blockchain, and bots are told to slow down or speed up, depending on current traffic. The three intersecting circles allow for streamlined movement, while maintaining key traffic situations. This allowed us to focus on the line follower to a greater extent, without having to worry about the robots colliding.

5.2.2 Table Materials

The blockchain nodes responsible for controlling the intersection and traffic light are mounted on one of the sides of the table, with cables managed cleanly through holes drilled in the table. Therefore a slim wooden plate, painted white, is used. The white coating is important since the light sensors, mounted on the robots, are better at differentiating very light shades from very dark shades.

5.3 Construction

5.3.1 Intersections and Traffic Lights

After taping our representation of roads on the table, the technical components still need a place to be fixed upon.

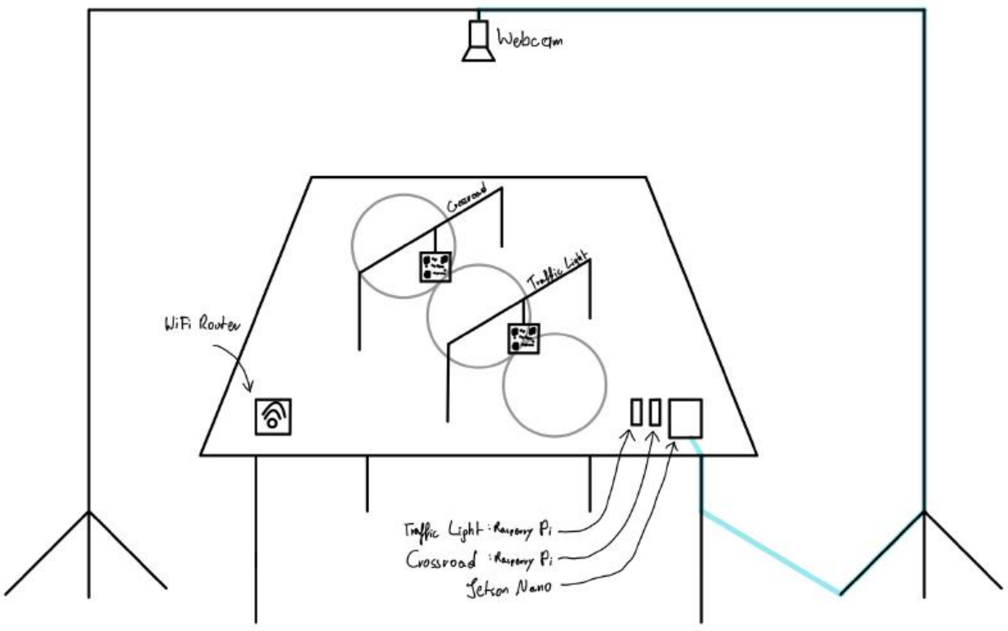


Figure 5.2: Sketch yielding a general overview of the tables construction

The wooden plate is drilled onto a table, which leaves space for cables to be run through the plate. The camera is placed at a specific height to ensure that only the 2 by 2-meter area is visible.

5.4 Overhead Camera

To realize the overhead camera, we used simple scaffolding and taped a USB camera to the top. The camera was connected to an NVIDIA Jetson Nano Developer Kit.[1] We chose a powerful microcomputer to handle a continuous stream of high-resolution video. Thanks to our previous experience in working with the NVIDIA Jetson Nano, more specifically its object identification abilities, the implementation of the existing QR code detection OpenCV code from Section 6.9 was quick.

5.4.1 NVIDIA Jetson Nano

The NVIDIA Jetson Nano is a powerful microcomputer specifically designed to handle applications like image classification, as well as object detection. Thanks to its onboard Ethernet port, it met our requirements in being able to process high-resolution video and being able to stream it to multiple clients. The frontend application explained in chapter 7 is therefore installed on the Jetson Nano, since it operates independently of the blockchain while sharing the same network. The NVIDIA Jetson Nano comes with a premade operating system based on Ubuntu.[2] The image includes multiple pre-installed packages, most notably an optimized version of OpenCV.

5.5 Final Result

After ruling out plenty of different layouts, we settled on the final design, as seen in figure 5.3. As stated, this road layout allows us to focus on the primary goals of this thesis, without having to account for potential collisions. Crude LEGO structures were used to mount QR-Codes, representing both the intersection and crossroad. The QR-Codes are explained thoroughly in Section 6.10. This approach allowed us to easily replace or remove the road hazards during testing.

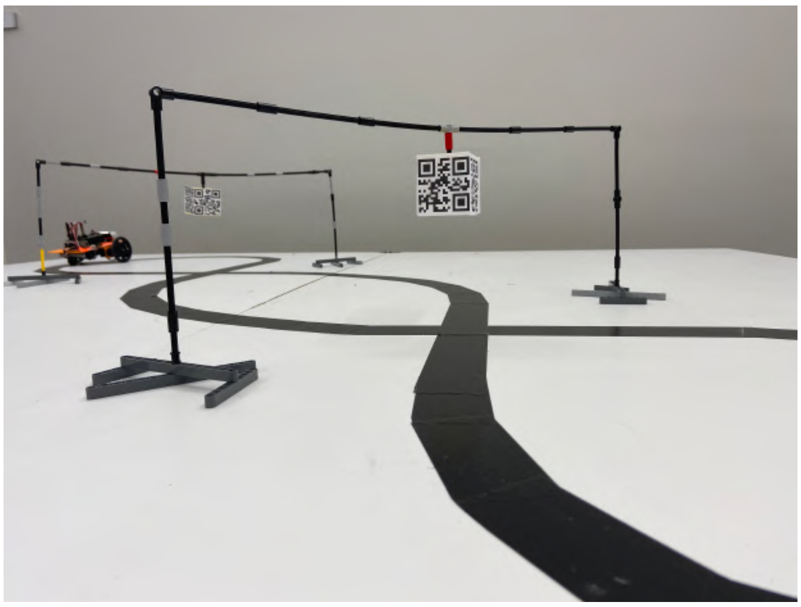


Figure 5.3: Side view of the finished Table

[1]NVIDIA, "Jetson Nano Developer Kit".
[2]NVIDIA, "Jetson Download Center".

Chapter 6

Writing a Blockchain

6.1 Introduction

Author: David Fischer

As mentioned, the usage of lightweight libraries to achieve excellent performance is essential. Since the robots, as well as nodes, are Raspberry Pis, which come with the Raspbian operating system, which has Python[1] pre-installed, the entire Blockchain is written in Python. The Flask[2] HTTP library features fast performance, thanks to its small size, and thus is used as the means of communication. Since Python dictionaries offer great compatibility with the JSON format, representational state transfer, also known as REST, is utilized as the communication protocol. This also grants the web management platform easier interfacing with the blockchain.

6.2 General Blockchain Structure

6.2.1 Theory

As stated, blockchains are assembled using three key elements. Blocks, connected using hash values, containing transactions. A valid blockchain is identified by matching hash values between the blocks. These are usually calculated with a so-called "Proof of Work"[3] (PoW) algorithm. PoW algorithms hand the service requester a task to complete, which hinders attacks, invalid transactions, and other forms of service disruption.

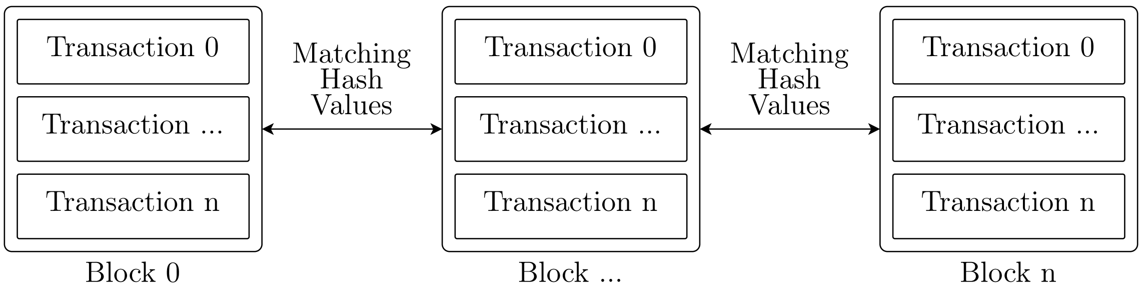


Figure 6.1: Simplified Blockchain Structure

[1]Foundation, "Welcome to Python.org".
[2]Pallets, "Flask".
[3]Markus Jakobsson, "Proofs of Work and Bread Pudding Protocols".

This decentralized approach to data storage is virtually infinitely scalable, while still remaining secure and efficient.

6.2.2 Transactions

Transactions include a sender, a recipient, and a message. This keeps transactions small and to the point, while still carrying meaningful data.

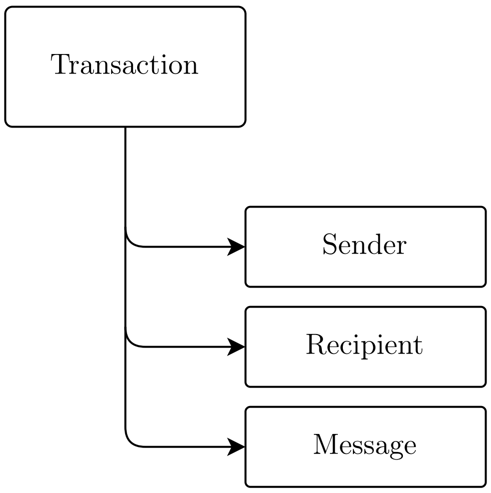


Figure 6.2: Contents of a Transaction

Sender

Regular blockchains, in the case of this example Bitcoin, specify the sender's wallet address, also known as an identifier, in this field. This project utilizes the hostname of the node to differentiate between transaction senders as well as recipients.

Recipient

Similar to the sender, the receiving node's hostname is used. This approach makes it possible to make transactions readable by humans.

Message

A regular blockchain's approach is, again, very different. Usually, this field is not a message, but instead, an amount of currency transferred. This project utilizes strings as messages, which makes it possible to transfer virtually any message.

6.3 Code

6.3.1 Initialization

Since blockchain nodes need to be independent, the same code is running on every host. Thanks to Python's class structure, writing expandable and modular code is very simple.

```
 1 class Blockchain:
 2     def __init__(self):
 3         self.current_transactions = []
 4         self.chain = []
 5         self.nodes = []
 6         self.hostnames = {}
 7 
 8         # Create the genesis block
 9         self.new_block(previous_hash='1', proof=100)
10 
```

Program 6.1: Python Blockchain Class Code Snippet

This code segment shows the initialization process of the Blockchain class, which is called every time an object of the Blockchain type is created. The variable *current_transactions* stores transactions made on the node, which are added to the next block, as soon as it is mined. The *chain* variable, stores all of the blocks which have been mined. The nodes and hostnames variables are both used for communication with other nodes.

Creating the Genesis Block

As seen on line 9 of the code snippet 6.1, a genesis block is created using a class function. The genesis block is the first block ever mined, therefore referred to as "genesis."[4] Since there is no previous block, its hash is set as one. For efficiencies sake, the initial block is not mined and the proof is set to 100.

6.3.2 Mining a Block

Proof of Work

To create a new block, containing the current transactions, the proof of work algorithm is executed. The proof of work algorithm requires proof from the last block. The simple approach to proof of work we chose, iterates a number p until hash(pp') contains four leading zeroes, with p' being the last hash. Hashing is done with the python library hashlib[5] and SHA-2 (Secure Hash Algorithm 2), more specifically SHA-256. The SHA-256 algorithm generates an almost-unique, fixed-size 256-bit hash. Hashing is a so-called one-way-function, producing a signature for a set of data, which can not be reversed to its original state. This method of proving work is simple, but still provides a certain level of security.

[4]Tardi, "Genesis Block".
[5]docs.python.org, "hashlib — Secure hashes and message digests".

```
def proof_of_work(self, last_proof):
    proof = 0
    while self.valid_proof(last_proof, proof) is False:
        proof += 1

    return proof

def valid_proof(self, last_proof, proof):
    guess_encode = f'{last_proof}{proof}'.encode()
    guess_hash = hashlib.sha256(guess_encode).hexdigest()
    return guess_hash[:4] == "0000"

```

Program 6.2: Proof of Work Algorithm

Hashing Blocks

To hash the previous block, the same SHA-256 algorithm is used. First, the block is dumped into a string, then hashed. This hash signature is then returned.

```
def hash(self, block):
    block_comp = json.dumps(block, sort_keys=True).encode()
    return hashlib.sha256(block_comp).hexdigest()

```

Program 6.3: Hash Function

6.4 Consensus Algorithm

As per definition, a consensus algorithm's task is to achieve agreement on a set of data among distributed systems.[6] To save processing time, which is especially important on microcomputers, our approach is kept as simple as possible. The node is tasked to iterate through its neighbors and fetch their chains. If a neighbors chain is longer, their chain is validated and the original nodes chain is replaced. If no longer chains are found, the node calls the consensus algorithm on other nodes in its list of neighbors.

6.4.1 Validating Chains

As stated, other chains need to be validated, to ensure their validity and successful transfer. A function iterates through the chain and compares the blocks given previous hash to the result of hashing the current block, as seen in snippet 6.3. If an error occurs during this process or hash signatures dont match up, the chain is marked as invalid.

[6] meet97_patel, "Consensus Algorithms in Blockchain".

6.5 REST API

6.5.1 Introduction

*RE*presentational *S*tate *T*ransfer, also commonly known as REST, is an architectural style, developed by Roy Fielding.[7] Defined by multiple guiding principles, almost any piece of information is defined as a resource. These resources shall be self-descriptive, always being transferred with a media-type, which in our case would be *JSON*. A REST API should offer multiple interfaces, allowing diverse operations like gathering information (*GET*), updating information (*PUT*), creating new information (*POST*), as well as deleting information (*DELETE*). These interfaces are all part of HTTP, and should not be confused with interfaces offered by REST, but since we are using REST behind an HTTP server, we will be using these terms from here on out.

6.5.2 Flask

Flask is a micro web framework written in Python.[8] It offers fast speed while maintaining a small size and a great amount of functionality.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

app.run(port=5000)

```

Program 6.4: A simple Flask server, offering one endpoint

The snippet 6.4, creates a Flask server and specifies a single route, returning "Hello, World!" in plaintext. The app is then started on port 5000, meaning if http://localhost:5000/ is accessed, the *hello_world*() function is called.

6.5.3 The JSON Data Format

JSON, also known as *J*ava*S*cript *O*bject *N*otation, is a syntax used for storing and exchanging data. JSON is usually served in text and then converted into objects. Python has a library for parsing JSON built-in, with its dictionary object type looking exactly like a JSON. It is therefore simple to work with JSON objects in any context.

```python
{'key': 'value', 'another_key': 'value'}

```

Program 6.5: Example JSON Object

Using JSON allows us to exchange data fast while making processing it simpler.

[7] Fielding, "Representational State Transfer (REST)".
[8] Pallets, "Flask".

6.5.4 Blockchain Endpoints

Interfacing with the blockchain would be impossible without endpoints being opened up. This means every blockchain function has been opened up using a Flask web server. The following sections serve a closer look at these interfaces.

Adding a Transaction

Since the general blockchain structure has already been outlined, these sections are simple. To add a transaction a *POST* request is sent to the node under */transactions/new*. The code then simply creates a new dictionary containing the information given in the requests parameters and mines a new block, to correspond to our 1 transaction per block limit.

```python
@app.route('/transactions/new', methods=['POST'])
def new_transaction():
    # Get Parameters
    values = request.get_json()

    # Create a new Transaction
    index = blockchain.new_transaction(values['sender'], values['recipient'], values['
     amount'])

    # Mine the Block
    r = requests.get('http://localhost:5000/mine')

    # Create and Send a Response
    response = {'message': 'Transaction will be added to Block {}'.format(index)}
    return jsonify(response), 201


```

Program 6.6: Simplified Code Snippet outlining a Transaction being added

The snippet 6.6 has been stripped of diverse error handling. Normally, the request parameters would be validated.

Mining a Block

The endpoint to mine new blocks is available using a GET request at */mine*. The current block is mined by calling the proof of work algorithm, giving a reward to the node which mined the block, validating the chain, and creating a new block. After these operations are complete, the consensus algorithm is called to propagate the new chain through the network.

Registering other Nodes

The endpoint to register new chains is available using a POST request at */nodes/register*, specifying the neighbor nodes IP address as well as its identifier. These values are stored in a dictionary in the Blockchain class.

6.5.5 Various Monitoring Endpoints

While the aforementioned endpoints offer a great amount of the functionality of the blockchain, a few minor routes are still required.

Fetching the Current Blockchain

Fetching the blockchain is as simple as it gets. The chain is fetched from the Blockchain class and placed in a response JSON.

```
@app.route('/chain', methods=['GET'])
def full_chain():
    response = {
        'chain': blockchain.chain,
        'length': len(blockchain.chain),
    }
    return jsonify(response), 200

```

Program 6.7: Returning the current Blockchain

To get an idea of this routes response please see 6.8.

```
{
    "chain": [
        {
        "index": 1,
        "previous_hash": "1",
        "proof": 100,
        "timestamp": 1605879307.5103903,
        "transactions": [
            // No transactions since this is the Genesis Block
        ]
        }
    ],
    "length": 1
    }

```

Program 6.8: Blockchain containing only the Genesis Block

Listing Node neighbors recursively

To create a network graph, mentioned in section 7.4.3, singular nodes need to be able to recurse through the network and fetch every node's respective neighbors. This is done using two routes "/nodes/frontend/list" and "/nodes/frontend/list_recursively". "/list" simply returns the nodes current neighbors, while "/list_recursive" is called to work its way through the network, leaving out every node, which already contributed to the list. This way the initial node ends up with a complete list of every node and their respective neighbors.

6.6 Discovering Other Nodes

The only thing the blockchain is missing are other nodes to exchange their chains. To achieve this, a separate class is put in place, containing a diverse set of networking Python packages. Every few minutes the node's network is scanned, ideally discovering other nodes.

If another node is discovered, it is automatically added to the blockchains list of neighbors. This is done by utilizing Python's built-in threading module.

6.6.1 Python Threading

A thread, by definition, is a separate flow of execution, running in parallel to the primary task. We utilize this, to concurrently run the Blockchain as well as several other tasks.

6.6.2 NodeDiscovery Class

To avoid manual additions of neighbors, we decided to write a class to automate this progress. Since our nodes are connected to the same WIFI network, within the same subnet range, the implementation was pretty simple. This of course would have to be expanded to work with the 5G towers.
To make automatic detection of nodes work, multiple hosts on the subnet are concurrently pinged,[9] ranging from the first to the last. Once a ping is successful, a *GET* request is sent to the host on port 5000. This is done to check if the host is a blockchain node since other devices could also be within the network. The detected blockchain nodes are added to the Blockchain class.
This process simplifies our testing workflow, which is also why it is run in parallel to the blockchain every 5 minutes, to detect nodes, which have just joined the network.

6.7 Blockchain Event Handler

Since every part of the blockchain is now functional including the chain itself, neighbor nodes, and automatic detection of other nodes, the next step is automatic processing of new transactions. This is done using a new Python class, called *EventHandler*, run in a thread, therefore concurrently to the Blockchain, similar to *NodeDiscovery*.

6.7.1 EventHandler Class

The EventHandler class is created in the main section of the blockchain. It is responsible for handling incoming transactions, meant for the node. Once a new transaction is added to the current block, the EventHandler is tasked to figure out if the transaction is meant for the node it is running on and how to react. During the creation process, the EventHandler is given the type of the node, either robot, traffic light, or crossroads. This makes it possible to achieve uniform communication between the nodes.

[9] linux.die.net, "ping(8) - Linux man page".

```python
def handle(self, block):
    transaction = block['transactions'][0]
    if(transaction['recipient'] == self.hostname):
        print('[*] recieved transaction meant for me. instructions: ' + str(
    transaction['amount']))

        if(self.entity == "BOT"): # Robot
            self.send_instructions(str(transaction['amount']))
        elif(self.entity == "TL"): # traffic light
            self.traffic_light_queue(transaction['sender'], str(transaction['amount'])
    )
        elif(self.entity == "CR"): # crossroads
            self.crossroad_light_queue(transaction['sender'])
    else:
        print('[*] transaction ignored')

```

Program 6.9: EventHandler handle function

The snippet 6.9 gives a simple overview of what decision is made, based on simple conditions.

Robot Handling

Robots receive instructions from traffic lights and crossroads. As explained in section 8.5.1 these are held simple, condensing down to either "continue driving" or wait for a certain amount of time.

Traffic Light Handling

Since our traffic lights should compare to the current system in place on roads today, the light cycles are not dynamic. A 15-second delay is placed between every cycle. The requesting robots are simply given the time left on the current cycle. The cycles are changed using a thread within the EventHandler class, responsible for waiting 15 seconds, then changing a variable to be either "left:right" or "front:back" to correspond to the current green lights.

Crossroad Handling

Crossroads are controlled using a queue system. Once a robot sends a request, it is added to the queue. If no other robots came before the current one, it is told to continue driving, otherwise, it halts. The delay until the robot is allowed to continue driving is calculated using the current amount of robots in the queue and multiplying it by how many seconds a robot takes to cross over. If multiple robots in the same lane are halted at the same time, the queue prioritizes those robots to keep the traffic flowing.

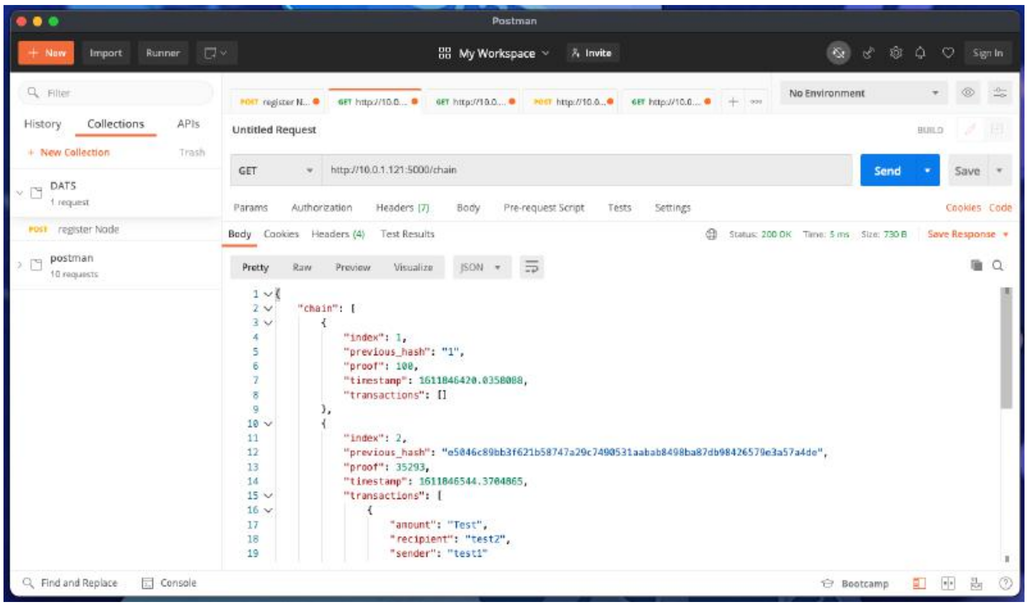
6.8 Testing Blockchain Routes using Postman

Postman[10] is a collaboration platform, developed specifically for API development. Thanks to Postman, we were able to streamline our development process, since this project's primary communication methods all rely on requests.

[10] Postman, "Postman | The Collaboration Platform for API Development".

6.8.1 Usage during Blockchain Development

Postman was especially useful during the initial development of the blockchain API.

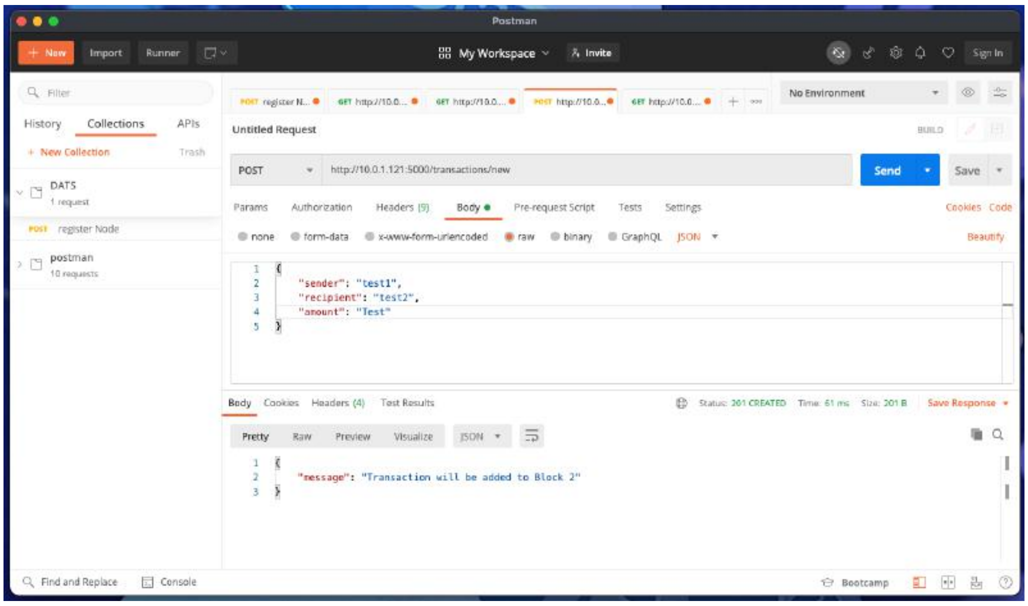


```
GET http://10.0.1.121:5000/chain
```

```
{
    "chain": [
        {
            "index": 1,
            "previous_hash": "1",
            "proof": 100,
            "timestamp": 1611846420.0358088,
            "transactions": []
        },
        {
            "index": 2,
            "previous_hash": "e5046c89bb3f621b58747a29c7490531aabab8498ba87db98426579e3a57a4de",
            "proof": 35293,
            "timestamp": 1611846544.3704865,
            "transactions": [
                {
                    "amount": "Test",
                    "recipient": "test2",
                    "sender": "test1"
```

Figure 6.3: Requesting the current blockchain using Postman

Figure 6.3 shows the Postman applications interface. Multiple request methods can be chosen, we only had to rely on *GET* and *POST*. Once the blue send button is pressed, Postman sends a request to the specified address using the chosen method. The Body column displays the APIs returned value.



```
POST http://10.0.1.121:5000/transactions/new
```

```
{
    "sender": "test1",
    "recipient": "test2",
    "amount": "Test"
}
```

```
{
    "message": "Transaction will be added to Block 2"
}
```

Figure 6.4: Adding a transaction to the blockchain using Postman

Figure 6.4 includes a request body, set to the JSON data type.

6.9 Reading QR Codes using OpenCV

6.9.1 Introduction

Since, at any given time, the robots do not know where they are on the table, an indicator had to be put in place to signal which recipient the node has to specify in its next transaction. We decided on QR codes since they are easy to print, universally used across many applications, and feature great flexibility in their usage. In this project's case, QR codes are used to label Robots as well as the diverse road hazards, such as traffic lights and intersections.

Figure 6.5: Example QR Code

QR codes can be used to transfer virtually any information, with their only restriction being the maximum data limit of 3Kb.[11] To limit the error margin on partially destroyed codes, QR codes feature error correction.[12]

6.9.2 OpenCV

OpenCV,[13] also known as Open Computer Vision Library, is an open-source machine learning library. Built to support multiple kinds of devices, while retaining speed, OpenCV is a perfect match for this project. With its over 2500 optimized algorithms, OpenCV boasts one of the broadest catalogs in open-source computer vision projects. Since OpenCV is licensed under the BDS clause, third party additions are not only welcome but also supported by the community.

6.9.3 ZBar

ZBar[14] is a fully-featured open-source software suite, built for reading bar codes, as well as QR Codes. ZBar can be used with its graphical user interface or integrated into other programs using its vast suite of libraries.

[11] Taylor, "How Much Data Can A QR Code Store?"
[12] qrstuff.com, "QR Code Error Correction".
[13] team, "OpenCV (Open Source Computer Vision Library)".
[14] Brown, "ZBar bar code reader".

6.9.4 Implementing OpenCV and ZBar

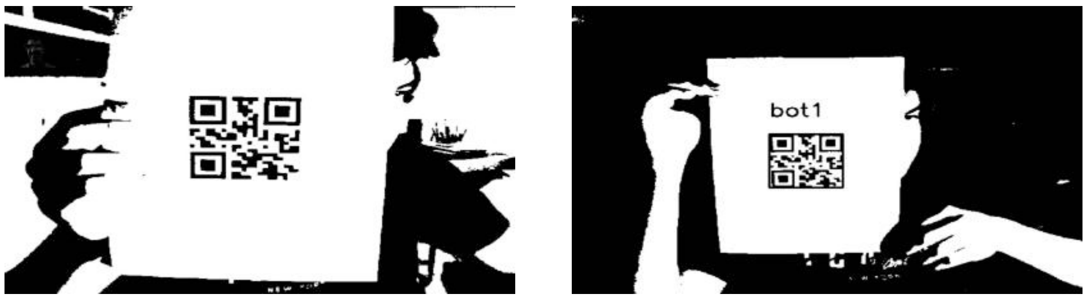
To use both OpenCV and ZBar in a Python script, separate libraries are required. Pyzbar[15] is a python wrapper, written to support the ZBar C library in Python. OpenCV ships with a pre-compiled Python library, so no additional packages have to be installed.

```
import cv2
import numpy as np
import pyzbar.pyzbar as pyzbar

```

Program 6.10: Importing the Required Libraries

6.9.5 Preprocessing Captured Frames

Since on a microcomputer, like the Raspberry Pi, there is no space for wasted performance, image preprocessing is an important step during the detection of QR Codes.[16] The approach we chose to achieve a more efficient framerate was converting and cropping the image, which made the delay during the ZBar decoding process smaller.



(a) OpenCV feed without label **(b)** OpenCV feed label

Figure 6.6: Comparison of OpenCV feed outputs

Compared to the entire frame, with regular RGB coloring, this increased the frames produced per second from 0.2 to 3. The code snippet 6.11 used to achieve this has a relatively small profile, showing that the OpenCV library is powerful while remaining simple.

```
cap = cv2.VideoCapture(0, cv2.CAP_V4L2)

_, frame = cap.read()
frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(frame, (5, 5), 0)
ret, bw_im = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)

decodedObjects = pyzbar.decode(bw_im)
```

Program 6.11: Converting the captured frame into grayscale

[15]Hudson, "Read one-dimensional barcodes and QR codes from Python 2 and 3."
[16]Chowdhury, "A Study on Image Processing to Facilitate Business System by Multiple Barcode Detection".

6.9.6 Handling Scanned QR Codes

Once a QR code is successfully scanned and decoded, one is left with a string, containing the QR code's data. As stated, the scanned QR codes contain the other node's hostname. Using Pythons "requests" module, an HTTP request is sent to the local blockchain. The sender is set to the node's hostname, the recipient is set to the QR codes decoded output, and the message is set to one of the messages specified in the standard.

6.9.7 Monitoring the OpenCV Output

To monitor all of the robot's live camera visions, an interface was put in place. A Python package called "Flask-OpenCV-Streamer."[17] As seen in figure 6.6a, the output is converted to grayscale, with a border drawn around the detected QR code as well as its scanned text, written above. The Python package takes this output and displays it on a website, hosted on the node. This makes it easy to spot potential issues and adjust settings quickly. To reduce potential unnecessary overhead the web-server is optional and is turned off by default.

6.10 QR Code Format

6.10.1 Format Standard

All QR Codes on the table follow the same structure. Hostname followed by a string with general information. The data is separated using a colon. An example of this would be $tl1 : right$, meaning trafficlight1, scanned from the right side.

6.10.2 Traffic Lights

As mentioned, traffic lights are abbreviated with "tl", followed by their identification number. The additional data is used to determine from which orientation the robot is approaching the traffic light. The blockchains "EventHandler" is responsible for the traffic light cycle, swapping between green for the front and the back and green for the left and the right.

(a) tl1:front

(b) tl1:left

Figure 6.7: Examples of traffic light QR Codes

[17]oitsjustjose, "A Python package for easily streaming OpenCV footage".

6.10.3 Crossroads

Crossroads follow the same pattern as traffic lights, with the exception being the empty data field. Since the direction from which a crossroad is approached does not matter in our simulation, unlike the traffic lights. Crossroads are abbreviated as "cr", also followed by their identifier, for example, $cr : 1$.

Chapter 7

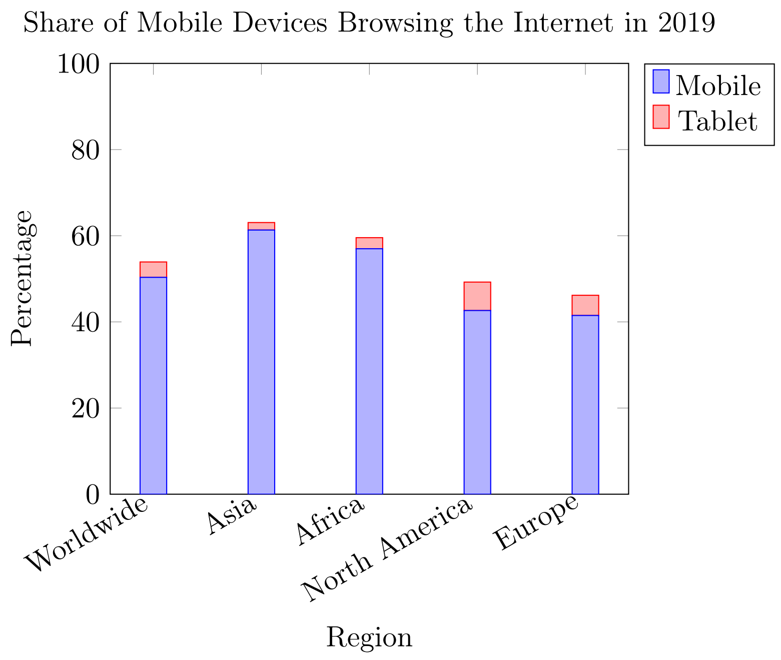
Developing a Graphical User Interface

7.1 Introduction

Author: Alexander Brenner
Since we decided to use the React framework, this section focuses on its many advantages as well as the general development process. In addition to complying with the primary objective of monitoring and controlling the system, the interface is designed as simple as possible with excellent mobile responsiveness. Mobile devices are carried almost all of the time, unlike laptops, which makes them great for quick access to information.

7.2 Mobile Responsiveness and React Bootstrap

Worldwide, more than 50 percent of websites are visited from mobile devices.[1]



[1]statista, "Anteil mobiler Endgeräte an allen Seitenaufrufen nach Regionen weltweit im Jahr 2019".

Due to this intensive use, many frontend frameworks opt to include packages for mobile responsiveness. The Bootstrap[2] library is one of the most reputable ones, even going as far as to state their approach as "Mobile first." This implies that the design is meant for mobile devices, with desktop web browsers coming second. For this particular project, "React Bootstrap,"[3] a version of Bootstrap made for Reacts component system is most ideal. Other changes like the removal of unneeded dependencies were also made. Bootstrap saves time overall and is therefore an important part of the web application.

7.3 Development

To display all of the vital information as clearly as possible, a basic structure is needed. Essentially, the interface displays a video stream of the table, a log window, which displays transactions made on the blockchain, a network graph, showing the connections between the nodes, and a list of nodes with their respective information and specific commands. With this basic principle, we designed a wireframe.[4]

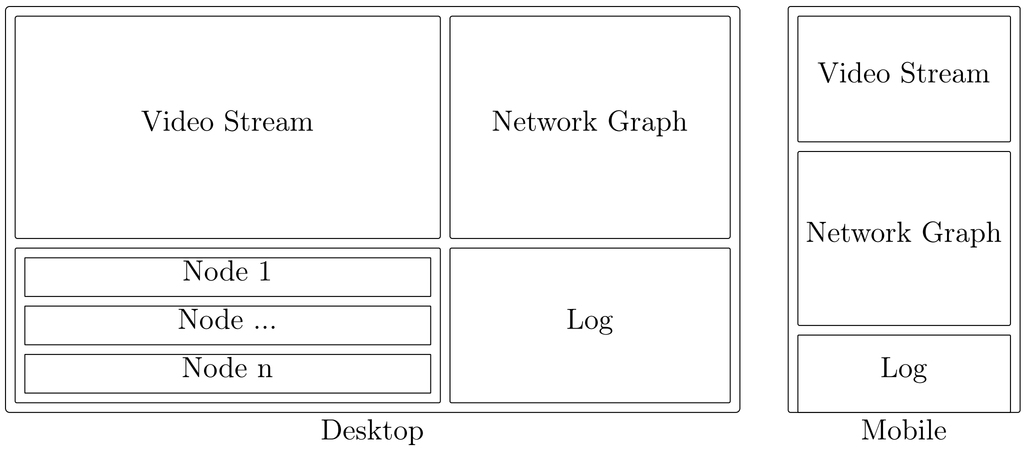


Figure 7.1: Wireframe of the Mobile as well as Desktop interfaces

7.3.1 Component Overview

App.js

In React, the App.js file is the centerpiece, which ties everything together. Usually, all of the main components are featured and drawn based on Reacts conditional rendering. In the case of this project, the video stream, log, node container, and network graph components are all listed. The App.js is also turned into a React component, which yields the opportunity to work with the component states.

Videostream

To observe the events on the table, a camera is mounted above. The cameras feed is preprocessed and exposed over RTSP (Real Time Streaming Protocol). To display the feed, a React component is used, keeping the code structured.

[2] Bootstrap-team, “Build fast, responsive sites with Bootstrap”.
[3] React-Bootstrap, “React Bootstrap”.
[4] UX, “What is wireframing?”

Log

Primarily used for debugging as well as visualizing the blockchain's transactions. This component regularly checks the blockchain for new blocks and transactions and displays them in a visually pleasing manner. This is done, by saving the collected data in the classes state variable, which causes the component to call its render() function, displaying the information.

Network Graph

To get a general overview of the Nodes and their connections to each other, a network graph component is featured. The blockchain offers an interface, which recurses through every node and returns a dictionary with hostnames as keys and a list of their neighbors as values. This information is requested within the component and displayed. To achieve this, a node package called "react-d3-graph" is used.[5] Data is collected from one of the nodes, parsed, and fed into the package. The output is displayed inside a separate component.

Node-List-Container and Bot-Component

An important part of testing the system is being able to interface with the nodes. To achieve this, every node gets its data stored in a node component. This provides the site with greater performance since updates on a node only affect one component and not a list of every node. The separate Node components are then displayed within another component, only serving as a wrapper. The following graphic illustrates this approach.

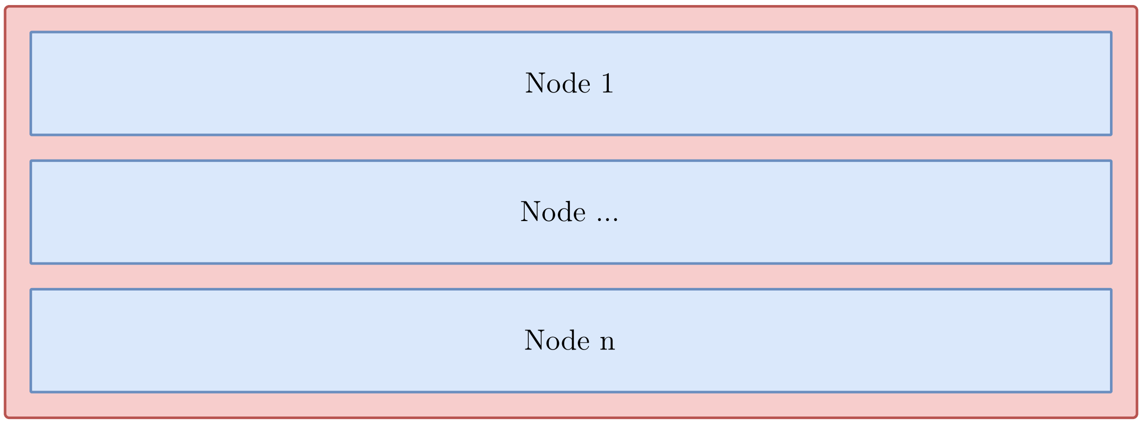


Figure 7.2: Wireframe of the wrapper component in blue and the separate node components in red.

In Figure 7.2 the blue-bordered elements represent the Bot-Component and the red-bordered element represents the Node-List-Container construct.

7.3.2 UI and UX Design

Introduction

An inviting design during the conception of the front end was one of the primary topics. To realize this, multiple modern frameworks and existing websites were studied, during

[5]Caldas, "react-d3-graph".

which advantages and weaknesses were identified. We settled on design strategies that fit the project's need best.

Google Material Design

A modern UI (User Interface), as well as a thought out UX (User Experience), characterize an accomplished website. Therefore multiple companies dedicate their efforts to advance development in the field of design philosophies as well as design frameworks.
Google revolutionized the concept of material design in 2014. Their concept is based on a multitude of guidelines, meant to help create a modern user interface while maintaining a good user experience. One of the pre-eminent guidelines specifies that the z-axis must only be one pixel thick, therefore ruling out 3-dimensional objects. A sense of depth is instead created using drop shadows, yielding an approach to reality. To achieve this, Google analyzed real objects and their shadows to create the material design guidelines.[6]
With all of material designs core concepts pointed out, its guidelines were followed closely during development.

Typography using Google Fonts

A readable font is one of the most significant parts of a modern website. Information has to be displayed as cleanly and uncluttered as possible. The Bootstrap design frameworks include restyled versions of existing HTML tags, following this approach. However, a more common approach is using Google's vast library of free-to-use fonts, known as Google Fonts.[7] Fonts are included on a website with a pre-made CSS file, containing the most important elements. One of the biggest advantages of this approach is the fact that the user does not need any fonts pre-installed on their device.

Iconography using FontAwesome

Visual aids to help identify functions helps users to quickly identify information. Most modern websites include so-called icons, which are usually small simplistic representations of an object. One of the most known and reliable icon libraries is called "FontAwesome."[8] Developed by Dave Gandy and first released in 2012, it features over 8,000 icons. Thanks to FontAwesome's lightweight approach, it can be used with any framework, including React. The icons range from simple variations of real objects to company logos. Using this library is also very easy, thanks to its comprehensive documentation. Regarding our project, diverse icons ranging from cars to arrows are used.

7.4 Implementing Blockchain Monitoring

7.4.1 Birds-Eye View Livestream

Implementing the Livestream is fairly simple, thanks to the hosting approach of the stream, which is done using a separate web server. The web server takes the output of a webcam and creates a continuous stream of encoded jpeg files, which HTML5 interprets as a video. This method, in our case, achieved a maximum framerate of up to 30, which is more than enough.[9] The final output is then simply placed inside of an $< img >$ tag, which is placed

[6]Google, "Build beautiful products, faster."
[7]Google, "Making the web more beautiful, fast, and open through great typography".
[8]Fonticons, "Font Awesome".
[9]BanitalebiDehkordi, "The Effect of Frame Rate on 3D Video Quality and Bitrate".

in the VideoStream component of the frontend. This approach not only saved time overall, but it also increased the performance of the devices displaying the stream, since all the preprocessing is done on the original web server.

```
<img src="http://192.168.1.x:3030/video_feed" alt="livestream"></img>

```

Program 7.1: Image-Tag used to Display a Video Feed

Due to this simple implementation, the VideoStream component had no other complex issues to resolve, so conditional rendering was not necessary.

7.4.2 Log Component

The log component is used to serve as a simple overview of all transactions in the Blockchain. Since the blockchain is propagated to all nodes on the network, any node can be used as an endpoint to fetch the '/chain' route, mentioned in 6.5.5. Before the component is loaded into the DOM, all transactions are stored in a temporary array. The array is then stored in the components state variable.

```
axios.get("http://" + endpoint_ip + ":5000/chain")
    .then(names => {
        // iterating thorugh the blockchain's blocks
        for(var i=1; i < names.data.chain.length; i++){
            // iterating through the single blocks transactions
            for(var j=0; j < names.data.chain[i].transactions.length; j++){
                var temp_arr = [];
                temp_arr[0] = names.data.chain[i].transactions[j].sender;
                temp_arr[1] = names.data.chain[i].transactions[j].recipient;
                temp_arr[2] = names.data.chain[i].transactions[j].amount;
                bot_msgs.push(temp_arr);
            }
        }
        this.setState({ msg: bot_msgs });
    })
}


```

Program 7.2: Fetching all Transactions from the Blockchain

A filled state variable would look like this.

```
var bot_msgs = [["bot1", "cl1", "Scanned QR"], // bot1 arrives at crossroad1
                ["cl1", "bot1", "run"],        // bot1 is instructed to
                                               // contine driving
                ["bot2", "cl1", "Scanned QR"], // bot2 arrives at crossroad1
                ["cl1", "bot2", "wait:7"]      // bot1 has not crossed yet,
                                               // so bot2 is instructed to wait
               ]


```

Program 7.3: Example of a Log Component State Variable

New messages are not loaded instantly, but instead every 10 seconds. Since the frontend is not the main priority, this saves processing power for other requests to the nodes.

7.4.3 Network Graph

As a result of the React-Component-Lifecycle, components can be told when to execute code. Due to the network graph's low priority during the initial load, the component is rendered into the DOM, even though the required data might not be acquired yet. This behavior is achieved with an overload of the already defined React component function $componentDidMount()$. The overloaded function calls a GET request, which asks a blockchain node for its neighbors using $/nodes/frontend/list\_recursive$. The request is made using the Axios[10] HTTP library, which delivers data based on JavaScripts Promises.

```
axios.get("http://192.168.1.x:5000/nodes/frontend/list_recursive")
.then(res => {
    req_data = res.data;
    var parsed_data = this.parseNodes(req_data);
    var conf = this.configureGraphSettings();
    // ...
});

```

Program 7.4: *GET* Request using the Axios JavaScript Library

After the data is received, a function is called to prepare the data for the graph. We decided on $react-d3-graph$[11], which is an open-source NPM package, created to easily display interactive graphs. The aforementioned function takes the received data and reformats it through several operations, to end up with data suitable for the d3-graph. After the new set of data is finished, the state variable of the NetworkGraph component is updated, to permanently store it.

Since the state variable starts out containing nothing, a helper variable $is_loading$ is also stored. This helper variable indicates the current state of the data collection and is therefore set to $True$ while the request is running. With the addition of the graph data to the state variable, the helper variable is set to $False$ to indicate the successful acquisition of data to display. While the helper variable is set to $True$, a loading icon is displayed,

[10]axios, "Promise based HTTP client for the browser and node.js".
[11]Caldas, "Interactive and configurable graphs with react and d3 effortlessly".

otherwise, the graph is presented. This workaround was necessary since the react-d3-graph library does not support rendering an empty set of data. This approach to conditional rendering is again a reminder of why we chose to use React as the web framework.

```
if(this.state.is_loading){
    return([LOADING CONTENT])
}else{
    return ([GRAPH]);
}

```

Program 7.5: Conditional Rendering as by Example of the Network Graph

The react-d3-graph library allows for simple customization of the graph. A helper function *configureGraphSettings*() was created for that purpose, simply filling a JSON object with the desired settings and returning it. In our case, we changed the font size, font color, width, height, as well as colors of the nodes.

```
configureGraphSettings() {
    var graph_settings = {
        height: 400,
        width: 300,
        node : {
        color: "#FC427B",
        fontSize: 18,
        fontColor: "#FC427B"
        },
        link : {
        color : "#FC427B"
        }
    }
    return graph_settings;
}

```

Program 7.6: Configuring the Network Graphs Appearance

This configuration is then passed to the components state variable. The *render*() function then creates a graph object containing the configuration and data.

```
<Graph id="graph-id" data={this.state.graph_data} config={this.state.grap_config} />

```

Program 7.7: Creating the final react-d3-graph

7.4.4 Listing Connected Robots

BotWrapper Component

As mentioned, the robot control panel is separated using two components. The BotWrapper components state variable contains the name and IP address of all robots, information which is required to create the Bot component. Before the component is loaded into the DOM, a GET request is sent to acquire every individual node's IP as well as their identifier. To save processing power on the backend, sorting out nodes other than robots is left to do for the frontend. This is done using a separate function, which simply iterates through the list and removes any entries, which do not include *bot* in their identifier.

```
for(var i=0; i < dataNames.length; i++){
    if(dataNames[i].includes("bot")){
        // ...
    }
}

```

Program 7.8: Filtering out Unwanted Nodes

After the data is cleaned up, a dictionary is created, using the identifiers as keys and the IP addresses as values. This dictionary is then placed in the components state variable.

To display this data in the DOM, the Bot component is required. Every entry in the dictionary is a separate Bot component, allowing efficient handling of user input. To achieve this, the Reacts *map*() conditional rendering function is used. This function iterates through the dictionary and creates one component for every set of data.

```
this.state.bot_data.map((item, i) => {
    return(
        <Bot name={item} ip={this.state.bot_data[item]} />
        )
    })

```

Program 7.9: Reacts map() function in use

The next section explains the now created Bot components.

Bot Component

As seen in the code snippet 7.9, the components construction is handed so-called "props", which are parameters. These are saved in the components state variable, to ensure access at any time and anywhere in the components code. The component features two icons, provided by FontAwesome, to stop and start the bot. The *onClick*() listener placed on these icons calls a function corresponding to the icon. Aforementioned, the Hedgehog code features an HTTP listener to receive instructions from the blockchain. This listener is used to send certain instructions using an Axios POST request.

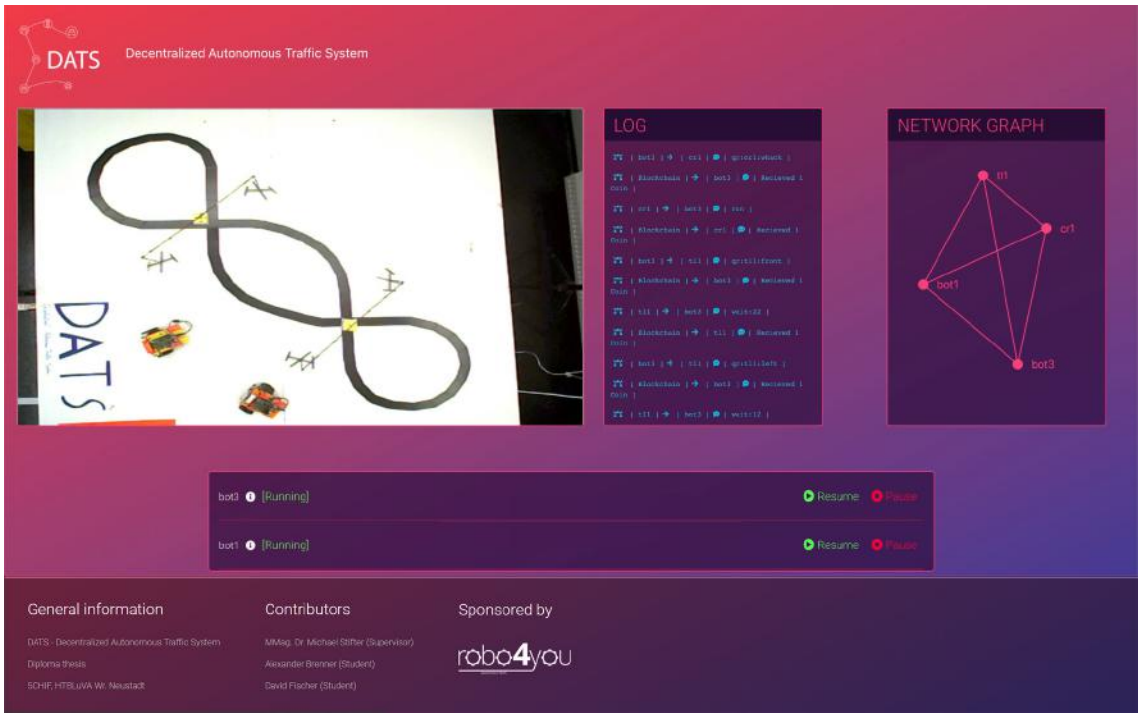
```
axios.post(robot_ip, {
        "msg": "stop"
    });
}

```

Program 7.10: Sending the Stop Instruction to a Robot

Additionally, the robot's current driving status is displayed. This function was not planned, but after rigorous testing, turned out to be quite useful. To identify the robots, the bot component also stores their identifier as well as their IP address. Hovering over the robot's name displays the IP address, which made quick changes to the robots over SSH easier.

7.5 Final Result

An excellent front end, both in code and design, is substantial. Nowadays, users expect a structured and easy to use interface on every website they visit. Even though this platform is primarily used by the project team, we still had to create a presentable result. After using a barebones version of the interface during testing, we decided to rewrite the stylesheet and make major changes. Instead of a black and white color scheme, a vibrant combination of pink and purple was chosen. Additionally, multiple components underwent small technical changes. An example of this would be the bot component, which now features icons, which when hovered over display the robot's IP address. A footer was also added to display general information like contributors and the sponsor, robo4you.



```
| bot1 | → | cr1 | ● | gr:cr1:back |
| Blockchain | → | bot1 | ● | Recieved 1 Coin |
| cr1 | → | bot1 | ● | ok |
| Blockchain | → | cr1 | ● | Recieved 1 Coin |
| bot1 | → | tl1 | ● | gr:tl1:front |
| Blockchain | → | bot1 | ● | Recieved 1 Coin |
| tl1 | → | bot3 | ● | wait:22 |
| Blockchain | → | tl1 | ● | Recieved 1 Coin |
| bot1 | → | tl1 | ● | gr:tl1:left |
| Blockchain | → | bot1 | ● | Recieved 1 Coin |
| tl1 | → | bot3 | ● | wait:12 |
```

bot3 [Running]

bot1 [Running]

General information

DATS - Decentralized Autonomous Traffic System
Diploma thesis
5CHIF, HTBLuVA Wr. Neustadt

Contributors

MMag. Dr. Michael Stifter (Supervisor)
Alexander Brenner (Student)
David Fischer (Student)

Sponsored by

Figure 7.3: Final Version of the Management Interface

7.6 Deployment

Author: David Fischer
To simplify the process of pushing new versions of the web interface to the Jetson Nano, mentioned in 5.4.1, we used Docker[12] in combination with GitHub's built-in CI/CD Pipeline.[13]

7.6.1 GitHub Actions

GitHub Actions[14] are GitHub's version of continous integration and continous deployment workflows.

Continuous Integration

Continuous integration, also known as CI, defines a software development practice in which multiple developers merge code in a central repository multiple times a day. Each commit causes an automated sequence of events to run, usually building and testing the code.

Continuous Deployment

Continuous deployment, while similar to continuous integration, also incorporates the provisioning and deployment of the code.

Stages of a CI/CD Pipeline

A CI/CD pipeline usually consists of common steps. These should be held as essential as possible since the workflow should not take longer than 10 minutes to complete.

Source Triggers the workflow, as soon as a new commit arrives.

Build Involves several steps to build the project. In our case, *docker build* was sufficient.

Test Runs diverse tests on the compiled code, for example, unit or integration tests.

Deploy Replaces the old production build with the newly generated build.

GitHub Actions File

GitHub Actions are defined using the YAML file format. As seen in 7.11, the "runs-on" key is defined as "self-hosted". GitHub provides a program called "Actions Runner," which allows us to pull, build and deploy directly on the Jetson Nano. This sped up our workflow significantly and allowed us to have a working build running in less than 3 minutes after the commit. The steps defined in the GitHub Actions file run every time a commit is detected by the "Actions Runner". First, the new code is cloned using the "actions/checkout@v2" step. After the code is downloaded, the "Build" and "Deploy" steps run, both using Docker. The commands used to run and deploy are explained in section 7.6.2.

[12] Docker, "Get Started with Docker".
[13] Anastasov, "CI/CD Pipeline: A Gentle Introduction".
[14] GitHub, "About continuous integration".

```
name: Build and Deploy Docker Image
on:
    push:
    branches:
        - master
jobs:
    build:
    runs-on: self-hosted
    steps:
        - uses: actions/checkout@v2

        - name: Build
        run: docker build -t DATS_web:latest
        - name: Deploy
        run: docker run -p 3000:3000 -d --restart=always DATS_web:latest

```

Program 7.11: The Github Actions file we used to Build and Deploy code on the Jetson Nano

7.6.2 Docker

Docker enables developers to deliver software in packages called images. Docker runs on OS-Level virtualization, therefore allowing images built on one machine to run on another. Once an image is run, it becomes a container and runs isolated on the host machine. There are multiple ways to interface with containers, the most common option being forwarding ports. Due to our deployment being a web server, we utilized this method.

7.6.3 Building with Docker

To build a project using Docker, a Dockerfile[15] is required. Dockerfiles are a sequence of commands, which are executed in separate containers. These containers are then converted to images, which in turn become layers. The layers are then stacked, to create a final image. Dockerfiles are built using the command "docker build -t image:version ."

```
FROM node as builder
WORKDIR /opt/DATS_WEB
COPY . /opt/DATS_WEB

RUN npm i
RUN npm run build

FROM nginx:1.14.2
COPY --from=builder /opt/DATS_WEB/build /usr/share/nginx/html
ADD nginx.conf /etc/nginx/

CMD ["nginx", "-g", "daemon off;"]

```

Program 7.12: The Dockerfile used to build the web interface

[15] Docker, "Dockerfile reference".

While Dockerfiles might seem complex at first glance, they are quite simple to read. DockerHub is a platform filled with pre-existing images, which are usually made to be built upon. For example, the Dockerfile we used specifies "node" as its builder, which saves the trouble of having to manually install NodeJS. The working directory is set within the image and the contents of the directory in which the Dockerfile resides are copied into the image. Two commands run within the image, the first being "npm i," "npm" meaning "Node Package Manager" and "i" meaning "install," which installs all required packages. The second command builds the NodeJS project. We could stop here and tell the container to run "npm start" to start the project. Since we are limited in processing power, overhead was a problem and we decided on a lightweight solution. Since NodeJS is only used as a builder, we are also able to save storage space, due to the packages and NodeJS itself not being needed. "Nginx" a lightweight webserver is retrieved from DockerHub, the build files from the builder are retrieved and saved in the new image. To finish the image, its entry point is set, being "nginx -g". This concludes the build process and we are left with a small, efficient Docker image, only consisting of 4 layers.

7.6.4 Running a Docker Container

To turn an image into a running container, the command "docker run image:version" is used. As seen in the snippet 7.11, we used multiple parameters.

-d Short for - -daemon, this parameter tells the container to detach from the shell and run in the background.

-p Short for - -port, this parameter forwards container ports to the host.

- -restart Tells the container when to restart. This is useful when the host restarts or the container crashes.

Chapter 8

The Hedgehog Controller

8.1 General Setup

8.1.1 Introduction

Author: David Fischer
To automate the setup process, we wrote a script to do the heavy lifting for us.

8.1.2 Installing required Libraries

Since we rely on libraries, some of which need to be compiled on the Raspberry Pi, having a script to automate the steps saved time. The script is separated into 11 steps, 8 of them install packages and libraries or build them.

8.1.3 Using Hedgehog Libraries outside of the Hedgehog-IDE

The Hedgehog-IDE comes preinstalled on every Hedgehog distribution. It is a simple interface for robotics beginners, only requiring a web browser. This comes with the downside of a separate instance of Python preinstalled on the hedgehog. Therefore the hedgehog package, by default, can not be used when using the system python version. To bypass this, we used Pythons "usrlocal.pth", which allows "site-packages" to be linked from different folders.

```
echo "/home/pi/HedgehogBundle/server/env/lib/python3.5/site-packages" >
        /home/pi/.local/lib/python3.5/site-packages/usrlocal.pth

```

Program 8.1: Linking the HedgehogBundles packages to the system Python install

8.1.4 Crontabs

Cron is a time-based job scheduler, allowing users to run jobs at specific points in time. We are running two separate tabs, both starting in parallel with the Hedgehog. The first tab kills the Hedgehog-IDE, saving a bit of processing power, and leaving the Hedgehogs less exposed on the local network. The second tab automatically starts the Blockchain on startup, making it easy to dynamically scale the network.

```
# Ending the Hedgehog-IDE process to save processing power
@reboot sleep 20 && killall "hedgehog-ide"

# Starting the Blockchain
@reboot sleep20 && python3 /home/pi/DATS/bot_code/blockchain/main.py

```

Program 8.2: Crontabs used on the Hedgehog Controller

8.2 Introduction

Author: Alexander Brenner
Subsequently, we were left with the constructed robots, but no code to drive them with. To guarantee proper line following, we again decided to take parts from the official Botball Kit. The so-called "Top-Hat" sensor yields a number based on how well the light it emits reflects back, this is useful specifically for detecting if the surface is black or white.

8.3 Basic Interfacing

The main tasks of our robot are to move motors and read sensor values. The Hedehog robotics controller offers interfaces for servo motors, regular motors, as well as digital and analog sensors, making the process effortless.[1]

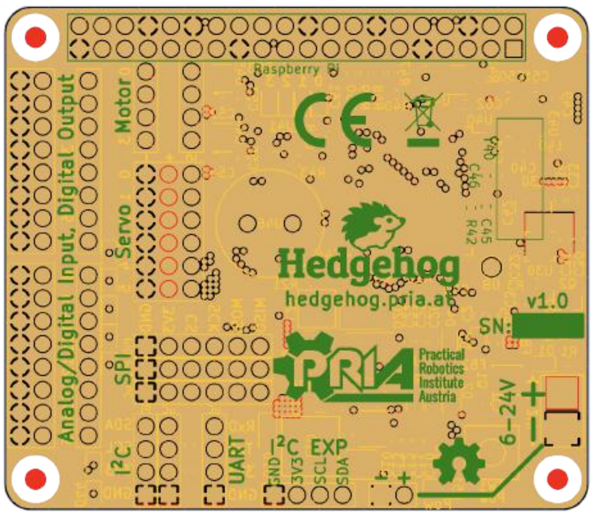


Figure 8.1: Hedgehog Gerber File outlining the available Ports

8.3.1 Motors

Thanks to the Hedgehogs excellent Python library, interfacing with almost all elements of the robot is simple. For example, motors are controlled using one line of code.

[1]PRIArobotics, "Hedgehog PCB".

```python
with connect(emergency=15) as hedgehog: # Initializing the Hedgehog Controller
    MOTOR_PORT = 1
    speed = 500
    hedgehog.move(MOTOR_PORT, speed) # Telling the Motor to move

```

Program 8.3: Interfacing with a Motor

Therefore, controlling multiple motors is correspondingly simple, since multiple calls to the motors can be executed subsequently, without any noticeable delay.

```python
with connect(emergency=15) as hedgehog: # Initializing the Hedgehog Controller
    hedgehog.move(LEFT_MOTOR_PORT, speed)
    hedgehog.move(RIGHT_MOTOR_PORT, speed) # Telling the Motor to move

```

Program 8.4: Interfacing with multiple Motors

8.3.2 Sensors

Sensors, while also easy to interface with, bring bigger obstacles with them. Thanks to our vast experience in working with the Hedgehog and other robotics controllers, we are now able to quickly identify and solve issues. One of these issues is cross-talk, which, in simple terms, is the unintentional manipulation of one sensor value caused by other sensors. This often occurs when sensors are plugged in directly next to each other. Even though the Hedgehog barely has any issues with crosstalk, one port is left empty in between the sensors.
Analog sensors yield values in a certain range, for example, 100 to 2500, while digital sensors are only able to return 0 or 1. Digital sensors, in our case, are used for the initialization process. Since the bot is clueless about its current surroundings and the light level, the button is used to tell the analog top-hat sensors to read out white and black values on the table. This makes it easy to test, without having to manually adjust values in the code.

```python
with connect(emergency=15) as hedgehog: # Initializing the Hedgehog Controller
    ANALOG_SENSOR_PORT = 1
    DIGITAL_SENSOR_PORT = 5 # Defining the sensor ports as variables
    print(hedgehog.get_analog(ANALOG_SENSOR_PORT)
    print(hedgehog.get_digital(DIGITAL_SENSOR_PORT) # Reading and Printing the Sensors
     output

```

Program 8.5: Reading Analog Sensor Values

8.4 PID Based Line Follower

Proportional, Integral, and Derivative, also known as PID, is used to calculate an error value and correct it. In our case, PID makes for a very accurate line follower.[2] To grasp

[2]Sheikih Jibrail, "PID Control of Line Followers".

this concept, understanding all of the PID components is essential.

8.4.1 Proportional

Proportional, describes the robot's approximate location in relation to the Line, demonstrated by a value. If the robot is placed exactly on the line, this value would correspond to 0.

8.4.2 Integral

The Integral value in a PID-based line follower is calculated with every Proportional value measured while the robot is moving.

8.4.3 Derivative

Describes the rate of change of the Proportional value.

8.4.4 Calculating the Error

All of the aforementioned terms are gauges for the errors the robot makes while following the line. The actual PID-Parameters are Kp, Ki, and Kd. These are multiplied with the error to make the robot correct its current error. The error is calculated using the grey color value and the middle tophats value.

$$error = (black + white)/2 * middle\_tophat\_value$$

8.5 Writing a PID Based Line Follower

To ensure functionality under different lighting conditions as well as differing black and white values, the line follower code initializes both values before starting. A button was mounted on the robots to make this setup as simple as possible. Both sensors are placed on the black tape, as soon as the button is pressed, the average of the read values is set as the black value. The same is repeated to read the white value. Thanks to this method we are guaranteed greater flexibility while testing.

```
wait_for_click()
black = (bot.get_analog(m_tophat) + bot.get_analog(r_tophat)) / 2

wait_for_click()
white = (bot.get_analog(m_tophat) + bot.get_analog(r_tophat)) / 2

grey = (black + white) / 2

```

Program 8.6: Reading the Black Tapes and Tables Light Values using both Sensors

The $wait\_for\_click()$ method is a helper function and simply reads the buttons input every few milliseconds.

```python
def wait_for_click():
    while(bot.get_digital(init_button) == 1):
        sleep(0.01)
    return

```

Program 8.7: The wait_for_click() Function

The necessary values are now set and the robot is ready to start driving. After another button press, the line follower is started in an infinite loop.

The $follow_line$ function initiates by setting two base variables: $lspeed$ and $rspeed$. Once the code enters into the while loop, the error is calculated, as explained in 8.4.4. Based on simple decision making, the turn speed is calculated, which, once subtracted and added to the left and right speed values respectively, results in the motors final movement speed.

```python
def follow_line():
    global cond_variable
    lspeed = 0
    rspeed = 0

    while(cond_variable):
        error = grey - bot.get_analog(m_tophat)

        if(error > 1000):
            baseSpeed = 400
            magic_variable = 2
        else:
            baseSpeed = 700
            magic_variable = 1.5

        turnSpeed = Kp * error * magic_variable

        lspeed = int(baseSpeed - turnSpeed)
        rspeed = int(baseSpeed + turnSpeed)

        bot.move(lmotor, lspeed)
        bot.move(rmotor, rspeed)
    return

```

Program 8.8: The Line Follower Function

To demonstrate the PID method, we captured the robot's error value during a two-meter drive along a straight line. Using this list of error values we were able to determine the distance traveled and the sensor's approximate location in relation to the line. This data was then converted to a figure 8.2 to demonstrate how the PID based line follower operates.

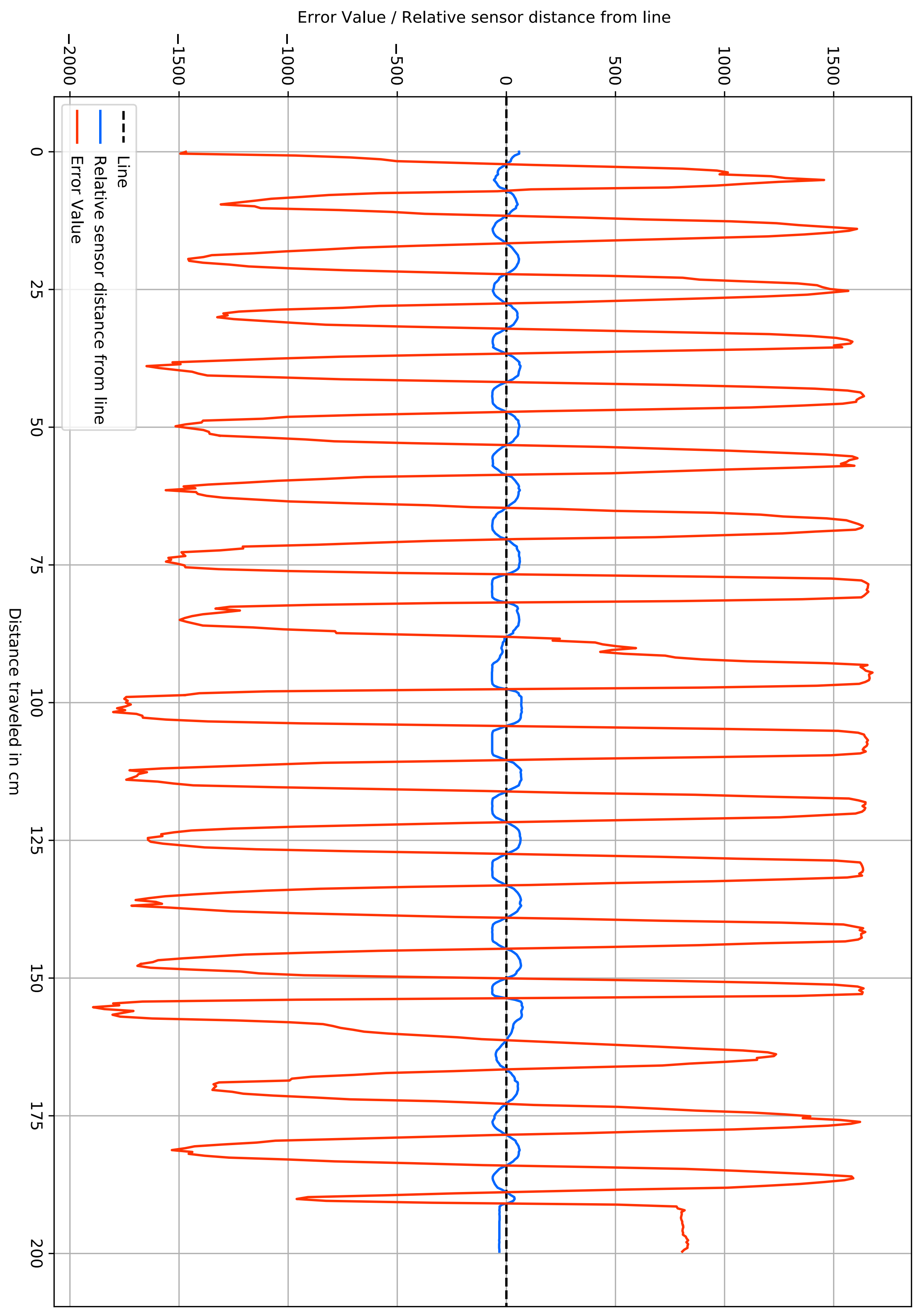


Figure 8.2: The error values changes during the robots drive down a two meter line

8.5.1 Receiving Instructions

Author: David Fischer
Since the robots need to be able to stop on demand, we used another Flask server. The Flask server is started as a thread, in parallel to the line follower. Due to its simple nature, only one route was implemented, which is continually waiting for instructions. The communication between the Flask server and the line follower function was solved without using a thread-safe interface since this specific use case did not need one. Therefore, a global variable is sufficient for our purposes. Due to the unpredictable nature of thread-unsafe interfaces, problems like race conditions are prone to occur. As seen in 8.8, the global variable *cond_variable*, is accessed every few milliseconds, therefore the worst problem that could occur is a slight delay before the robot comes to a stop.

```python
@app.route('/push', methods=['POST'])
def handle_push():
    global cond_variable # using the global cond_variable
    values = request.get_json()
    msg = values['msg']

    if(msg == 'stop'):
        cond_variable = 0

    if(msg == 'run'):
        cond_variable = 1

    if('wait' in msg):
        time = int(msg.split(':')[1])
        sleep(time)
        cond_variable = 1

    return 'recieved'

```

Program 8.9: The /push Route

Stop Instruction As an example: The stop button on the web interface is pressed. A POST request is sent to the robot, including a 'stop' message. The Flask server running on the hedgehog receives the request and parses its contents. As seen in 8.9, there are three options, which the robot can respond to. In this case, 'stop', which simply changes the global variable *cond_variable* to 0, causing the robot to stop. As mentioned, a slight delay could occur, although it was not noticeable in practice.

Run Instruction The opposite of the stop instruction, simply changing the *cond_variable* to 1, causing the robot to continue driving.

Wait Instruction Similar to the run instruction in nature, except for a number of seconds being sent along with the instruction. The seconds are separated from the message with a colon, so the wait instruction would look like this in practice: "$wait : 10$". Before setting the *cond_variable* to 1, the amount of seconds is waited out using the *sleep*() function.

8.6 Testing using the Hedgehog Controller

During the testing phases, the authors came across multiple issues. Ranging from hardware-related problems, to the blockchain not being able to propagate properly over the network. The set of debugging tools helped out immensely in addition to the Hedgehog controllers emergency stop functionalities. Working with hardware always comes with unexpected factors, since simulating reality is nearly impossible. The greatest challenge the authors had to overcome was networking delays since we were restricted to a weak WiFi network.

Chapter 9

Conclusion

Author: David Fischer
The Decentralized Autonomous Traffic System is one of the first approaches to decentralized traffic management with fully autonomous vehicles. This thesis was primarily inspired by the AUDI vehicle-to-infrastructure service[1] as well as current technology trends. The AUDI V2I service informs drivers which speed is required to reach the next traffic light when it cycles to green. This concept was expanded and decentralized. The project underwent vast changes during its development, ranging from changes to the code to scrapping the current vehicle prototype and rebuilding from the ground up. During development, the authors ensured high code quality, to enable future robotics students to continue the development of this project. Moreover, multiple Git repositories were created and publicized, allowing interested students to access our documentation, gcode files, and code.

9.1 Development

One of the most time-intensive tasks while working on this thesis was writing the diverse code. Ranging from complex algorithms to decentralization mechanisms, which had to work on low-power processors.

9.1.1 SSH Keys

One of the greatest assets during development were SSH keys.[2] Due to the thesis' focus on decentralization, multiple microcomputers regularly needed small adjustments. SSH keys are access credentials, similar in nature to a regular user and password authentication. The main difference being that SSH keys allow one to access a host without a password since the private key is used for authentication. Due to the sheer amount of nodes and times new code had to be deployed, being able to quickly log in saved a lot of time.

9.1.2 Hedgehog Raspberry Pi Shield

The Hedgehog Raspberry Pi shield enables easy interchange of parts like sensors or motors. The included library enables simple interfacing with the connected parts. This allowed the authors to quickly identify and replace broken sensors or motors and continue development. The detailed documentation allowed the authors to quickly identify code related issues and solve them.

[1]Schneider, "Audi networks with traffic lights in Europe".
[2]Security, "SSH Keys".

9.1.3 NVIDIA Jetson Nano

Due to the large scope of the thesis, the authors were not able to implement every planned feature, the biggest being a QR code scanner running on the NVIDIA Jetson Nano overhead camera.

9.1.4 Development during the COVID-19 Pandemic

Due to the ongoing COVID-19 pandemic, the authors were limited in both access to the project's hardware elements and time to test and prototype. These restrictions inspired the creation of the web interface's debug features, most notably the webcam view and transaction module. This allowed both of the authors to work on the project while maintaining a safe distance.

9.2 DATS Software

In parallel to the thesis, multiple software modules were written to demonstrate our vision of completely automated traffic. All of the code is modular and expandable to support future students in their endeavors. The authors regret not being able to test more than two robots at once, due to shipping delays. Expanding the project's scale in regards to the number of robots and the size of the table would be interesting for the near future.

9.2.1 DATSChain

A modular, efficient, and lightweight blockchain developed to run on microcomputers like the Raspberry Pi. The blockchain features communication methods, as well as rewards for mining new blocks, therefore it is fit for multiple future applications by interested students.

9.2.2 QR Code Reader

An efficient QR code reader, based on existing libraries with improved image processing and downscaling. Detected codes are propagated through the blockchain in a matter of seconds due to the reader having direct access to the local blockchain.

9.2.3 Responsive Line follower

A line follower, written to be as dynamic as possible. Featuring a web-server, the line follower is able to quickly respond to incoming instructions. The line follower features virtually infinite possibilities for expansion including custom code snippets like a parking procedure that could be run on demand.

9.2.4 DATS Web Interface

A carefully designed web interface with built-in debugging features. Featuring a Livestream of the Table, a network diagram, a blockchain transaction log, and manual controls for the vehicles. After multiple changes to its functionality and design, the authors are pleased with the final result. Debugging transactions or vehicle behavior using the platform was very easy thanks to its intuitive design. Due to the author's modular approach, the web interface is also expandable.

9.3 Outlook

As part of this thesis, the DATSChain was developed, which allows multiple microcomputers to dynamically propagate data over a network connection. DATSChain is split into multiple modules, most notably the "EventHandler", which is responsible for the logic behind traffic lights, crossroads, and vehicle interactions. Due to the ongoing COVID-19 Pandemic and its consequential lockdowns, we were only able to test the basic functionality of the Decentralized Autonomous Traffic System, barring us from expanding it even further. As mentioned, the authors hope for the continuation of this project by future robotics students, as this would mean more complex traffic scenarios, as well as other tasks for the vehicles to complete, could be implemented.

Index

Autoren Index

Literatur Index

Bibliography

Alexander Brenner, David Fischer. “Hedgehog vs Wallaby - Pros and Cons”. In: (2017). URL: https://robo4you.at/publications/Hedgehog_vs_Wallaby.pdf.

Anastasov, Marko. “CI/CD Pipeline: A Gentle Introduction”. In: (2019). URL: https://semaphoreci.com/blog/cicd-pipeline.

AUDI. “adaptive cruise control with stop and go function”. In: (2020). URL: https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/adaptive-cruise-control-with-stop-go-function.

axios. “Promise based HTTP client for the browser and node.js”. In: (2014). URL: https://github.com/axios/axios.

BanitalebiDehkordi, Amin. “The Effect of Frame Rate on 3D Video Quality and Bitrate”. In: (2014). URL: https://link.springer.com/article/10.1007/s13319-014-0034-3.

blockchain.com. *Total Number of Transactions on the Blockchain*. Tech. rep. 2020. URL: https://www.blockchain.com/charts/n-transactions-total/.

Bootstrap-team. “Build fast, responsive sites with Bootstrap”. In: (2020). URL: https://getbootstrap.com/.

Brown, Jeff. “ZBar bar code reader”. In: (2010). URL: http://zbar.sourceforge.net/.

Caldas, Daniel. “Interactive and configurable graphs with react and d3 effortlessly”. In: (2017). URL: https://github.com/danielcaldas/react-d3-graph#readme.

— “react-d3-graph”. In: (2020). URL: https://danielcaldas.github.io/react-d3-graph/docs/.

Chowdhury, Atiqul Islam. “A Study on Image Processing to Facilitate Business System by Multiple Barcode Detection”. In: (2019). URL: https://www.researchgate.net/publication/338479793_A_Study_on_Image_Processing_to_Facilitate_Business_System_by_Multiple_Barcode_Detection.

Clauser, Grant. “What Is Alexa (and Whats the Best Alexa Speaker?” In: (2020). URL: https://www.nytimes.com/wirecutter/reviews/best-alexa-speakers/.

Creality. “Ender 5 Pro 3D”. In: (2020). URL: https://www.creality3dofficial.com/products/ender-5-pro-3d-printer.

Docker. “Dockerfile reference”. In: (2021). URL: https://docs.docker.com/engine/reference/builder/.

— “Get Started with Docker”. In: (2021). URL: https://www.docker.com/.

docs.python.org. “hashlib — Secure hashes and message digests”. In: (2020). URL: https://docs.python.org/3/library/hashlib.html.

Elliott, Eric. “A Brief History of Decentralized Computing”. In: (2019). URL: https://medium.com/the-challenge/a-brief-history-of-decentralized-computing-d0d665783bcf.

ethereum.org. “Use Ethereum Applications”. In: (2020). URL: https://ethereum.org/en/dapps/.

Evan, You. “The Progressive JavaScript Framework”. In: (2020). URL: https://vuejs.org/.

Facebook. “A JavaScript library for building user interfaces”. In: (2020). URL: https://reactjs.org/.

Facebook. “React: Conditional Rendering”. In: (2020). URL: hhttps://reactjs.org/docs/conditional-rendering.html.

— “React: State and Lifecycle”. In: (2020). URL: https://reactjs.org/docs/state-and-lifecycle.html/.

Fielding, Roy. “Representational State Transfer (REST)”. In: (2000). URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

Fonticons. “Font Awesome”. In: (2020). URL: https://fonts.google.com/about.

Foundation, Python Software. “Welcome to Python.org”. In: (2020). URL: https://www.python.org/.

Gebhart, Andrew. “Everything you want to know about Google Home”. In: (2020). URL: https://www.cnet.com/home/smart-home/everything-you-want-to-know-about-google-home/.

GitHub. “About continuous integration”. In: (2021). URL: https://docs.github.com/en/actions/guides/about-continuous-integration.

Goodgame, Steve. “2020 Botball Game Review”. In: (2020). URL: http://webspace.pria.at/ecer2020/2020%20Botball%20Game%20Review%20v1.1%2020200110.PDF.

Google. “Build beautiful products, faster.” In: (2020). URL: https://material.io/.

— “Google Trends”. In: (2020). URL: https://trends.google.com/trends/explore?date=all&q=react%20js,angular%20js,Vue.js.

— “Making the web more beautiful, fast, and open through great typography”. In: (2020). URL: https://fonts.google.com/about.

— “One framework. Mobile and desktop.” In: (2020). URL: https://angular.io/.

Hudson, Lawrence. “Read one-dimensional barcodes and QR codes from Python 2 and 3.” In: (2019). URL: https://pypi.org/project/pyzbar/.

IEEE. “IEEE 802.15.4-2020 - IEEE Standard for Low-Rate Wireless Networks”. In: (2015). URL: https://standards.ieee.org/standard/802_15_4-2020.html.

John M. Griffin, Amin Shams. “Is Bitcoin Really Un-Tethered?” In: (2019). URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3195066.

John, Kouraklis. “MVVM as Design Pattern”. In: Oct. 2016. DOI: 10.1007/978-1-4842-2214-0_1.

Joseph Poon, Thaddeus Dryja. “The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments”. In: (2016). URL: https://lightning.network/lightning-network-paper.pdf.

Kaidong Wu Yun Ma, Gang Huang. “A First Look at Blockchain-based Decentralized Applications”. In: (2019). URL: https://arxiv.org/pdf/1909.00939.pdf.

Kesby, Rebecca. “How the world’s first webcam made a coffee pot famous”. In: (2012). URL: https://www.bbc.com/news/technology-20439301.

KIPR. “2020 Botball Electronics Kit”. In: (2020). URL: http://webspace.pria.at/ecer2020/Parts/2020%20Electronics%20Kit.pdf.

— “2020 Botball Lego Parts”. In: (2020). URL: http://webspace.pria.at/ecer2020/Parts/2020%20Lego%20Parts.pdf.

— “2020 Botball Metal Parts”. In: (2020). URL: http://webspace.pria.at/ecer2020/Parts/2020%20KIPR%20Metal%20Parts.PDF.

— “2020 Botball Wombat Controller”. In: (2020). URL: http://webspace.pria.at/ecer2020/Curriculum/2020%20Botball%20Wombat%20Robot%20Build%20Guide.pdf.

— “2020 IGUS Parts”. In: (2020). URL: http://webspace.pria.at/ecer2020/Parts/2020%20IGUS%20Parts.pdf.

— “botball”. In: (2020). URL: https://www.kipr.org/botball.

KISS. “Chassis Bracket (for Wombat)”. In: (2017). URL: https://botball-swag.myshopify.com/collections/metal-parts/products/chassis-bracket-for-wombat.

Koppensteiner, Dr. Gottfried. “PRIA”. In: (2020). URL: https://pria.at/.

Koza, Clemens. “Educational Robotics Controller: Hedgehog”. In: (2020). URL: https://hedgehog.pria.at/.

— “Welcome to Hedgehog!” In: (2017). URL: https://hedgehog.readthedocs.io/en/latest/.

linux.die.net. “ping(8) - Linux man page”. In: (2000). URL: https://linux.die.net/man/8/ping.

Madeira, Antonio. “Ethereum 2.0 Likely to Affect DeFi and DApps With PoS Introduction”. In: (2020). URL: https://cointelegraph.com/news/ethereum-20-likely-to-affect-defi-and-dapps-with-pos-introduction.

Maj, Wojciech. “Interactive React Lifecycle Methods diagram”. In: (2018). URL: https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/.

Mandrita Banerjee Junghee Lee, Kim-Kwang Raymond Choo. “A blockchain future for internet of things security: a position paper”. In: (2017). URL: https://www.sciencedirect.com/science/article/pii/S2352864817302900.

Markus Jakobsson, Ari Juels. “Proofs of Work and Bread Pudding Protocols”. In: (1999). URL: https://link.springer.com/chapter/10.1007%2F978-0-387-35568-9_18.

Matthew Weber Maryanne Murray, Sarah Slobin. “A REUTERS VISUAL GUIDE: Blockchain explained”. In: (2020). URL: http://graphics.reuters.com/TECHNOLOGY-BLOCKCHAIN/010070P11GN/index.html.

MAXON. “Why Cinema 4D?” In: (2020). URL: https://www.maxon.net/en-us/products/cinema-4d/overview/.

meet97_patel. “Consensus Algorithms in Blockchain”. In: (2020). URL: https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/.

Mercedes-Benz. “What is: Active Lane Keeping Assist.” In: (2020). URL: https://www.mercedes-benzsouthwest.co.uk/about/news-and-events/active-lane-keeping-assist/.

Nakamoto, Satoshi. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: (2008). URL: https://bitcoin.org/bitcoin.pdf.

NVIDIA. “Jetson Download Center”. In: (2020). URL: https://developer.nvidia.com/embedded/downloads.

— “Jetson Nano Developer Kit”. In: (2020). URL: https://developer.nvidia.com/EMBEDDED/jetson-nano-developer-kit.

oitsjustjose. “A Python package for easily streaming OpenCV footage”. In: (2018). URL: https://github.com/oitsjustjose/Flask-OpenCV-Streamer.

Pallets. “Flask”. In: (2010). URL: https://flask.palletsprojects.com/en/1.1.x/.

Postman, Inc. “Postman | The Collaboration Platform for API Development”. In: (2014). URL: https://www.postman.com/.

PRIArobotics. “Hedgehog PCB”. In: (2018). URL: https://github.com/PRIArobotics/Hedgehog_PCB.

qrstuff.com. “QR Code Error Correction”. In: (2011). URL: https://blog.qrstuff.com/2011/12/14/qr-code-error-correction.

React-Bootstrap. “React Bootstrap”. In: (2020). URL: https://react-bootstrap.github.io/.

savecar.gov. “AEB: Automatic Emergency Braking”. In: (2020). URL: https://www.safercar.gov/Vehicle+Shoppers/Safety+Technology/aeb-1/.

Schneider, Tilman. “Audi networks with traffic lights in Europe”. In: (2020). URL: https://www.audi-mediacenter.com/en/press-releases/audi-networks-with-traffic-lights-in-europe-11649.

Security, SSH Communications. “SSH Keys”. In: (2020). URL: https://www.ssh.com/ssh/key/.

Sheikih Jibrail, Rakesh Maharna. “PID Control of Line Followers”. In: (2013). URL: https://core.ac.uk/download/pdf/53189911.pdf.

Soares, Italo. “3D Printer G-CODES”. In: (2016). URL: https://3dprinterchat.com/3d-printer-g-codes/.

statista. “Anteil mobiler Endgeräte an allen Seitenaufrufen nach Regionen weltweit im Jahr 2019”. German. In: (2019). URL: https://de.statista.com/statistik/daten/studie/217457/umfrage/anteil-mobiler-endgeraete-an-allen-seitenaufrufen-weltweit/#professional.

Tardi, Carla. “Genesis Block”. In: (2019). URL: https://www.investopedia.com/terms/g/genesis-block.asp.

Taylor, Lee. “How Much Data Can A QR Code Store?” In: (2017). URL: http://qrcode.meetheed.com/question7.php#:~:text=Standard%20QR%20Codes%20can%20hold,grid%20of%20modules%20(squares).

team, OpenCV. “OpenCV (Open Source Computer Vision Library)”. In: (2020). URL: https://opencv.org/about/.

Teicher, Jordan. “The little-known story of the first IoT device”. In: (2018). URL: https://www.ibm.com/blogs/industries/little-known-story-first-iot-device/.

Tesla. “Future of Driving”. In: (2020). URL: https://www.tesla.com/autopilot.

Tilley, Aaron. “How Hackers Could Use A Nest Thermostat As An Entry Point Into Your Home”. In: (2015). URL: https://www.forbes.com/sites/aarontilley/2015/03/06/nest-thermostat-hack-home-network/#6ea6dc763986.

Tlig, M., O. Buffet, and O. Simonin. “Decentralized traffic management: A synchronization-based intersection control”. In: (2014), pp. 109–114.

Ultimaker. “Ultimaker Cura”. In: (2020). URL: https://ultimaker.com/software/ultimaker-cura.

UX, Experience. “What is wireframing?” In: (2019). URL: https://www.experienceux.co.uk/faqs/what-is-wireframing/.

Visa. “How inefficient are dapps?” In: (2018). URL: https://medium.com/coinmonks/how-inefficient-are-dapps-c18062c80a71.

— “Visa Fact Sheet”. In: (2018). URL: https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf.