Faculty of Electrical Engineering, Precision Engineering, Information Technology

The degree program Media Engineering B.Eng.

Bachelor Thesis written by

Josefina F r i t z

Matriculation-Nr.: 357 6377

Beginner-Friendly Transformation of an Old Tablet into a Remotely Controlled Robot

Winter semester 2024/25

First Examiner: Prof. Dr. Ralph Lano

Second Examiner: Prof. Dr. rer. nat. Matthias Hopf

Technische Hochschule Nürnberg

Notice: This declaration must be securely bound in all copies of the final thesis. (No spiral binding)

Declaration by the student in accord with examination rules and regulations

Personal information of the student:

Family name: Fritz Given name: Josefina Student ID number: 3576377

Faculty: Elektrotechnik Feinwerktechnik Informationstechnik Degree programme: Media Engineering B. Eng.

Semester: WiSe 2024/25

Title of the final thesis:

Beginner-Friendly Transformation of an Old Tablet into a Remotely Controlled Robot

I hereby declare that this work is my own and has not been submitted for examination purposes in any other context. I have indicated and acknowledged all sources, aids, and quotations used in its production.

Nürnberg, 07.10.2024 Josefina Fritz

City, Date, Signature of the student

Declaration regarding the publication of the thesis named above

The decision to wholly or partially publish the thesis is, on principle, first and foremost the sole responsibility of the student author. According to the Copyright Act (UrhG), when a student produces a thesis, the author of the thesis acquires sole copyright and, in principle, also the resulting rights of use, such as first publication (§ 12 UrhG), distribution (§ 17 UrhG), reproduction (§ 16 UrhG), online use, etc., i.e., all rights pertaining to non-commercial or commercial exploitation.

The university and its employees will not publish theses or parts thereof without the agreement of the student author, in particular, it will not be placed in the publicly accessible section of the university library.

I hereby ☒ authorize, if and insofar as no conflicting agreements with third parties exist,

☐ do not authorize

that the above-named thesis may be made publicly accessible by the Technische Hochschule Nürnberg Georg Simon Ohm; if applicable, if indicated by an embargo annotation on the thesis, this will occur after the expiry of any publication embargo of

0 years (0 - 5 years after the date the thesis was submitted).

If authorization is granted, it is irrevocable; the thesis submission includes a pdf formatted version on a data storage medium for this purpose. Provisions in the respectively applicable Study and Examination Regulations about the type and scope of copies and materials that must be submitted as part of the thesis shall not be affected by the submission of the pdf format for publication.

Nürnberg, 07.10.2024 Josefina Fritz

City, Date, Signature of the student

Data protection: The personal data you provide during your application is stored and processed by the Technische Hochschule Nürnberg Georg Simon Ohm. Further information on how the Technische Hochschule Nürnberg handles your personal data can be found at: https://www.th-nuernberg.de/datenschutz/

Kurzdarstellung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung von selbstgemachten Robotoren, die für Einsteiger geeignet sind. Dabei sollen vorallem Kindern auf spielerische Weise grundlegende Programmierkenntnisse und praktische Erfahrungen im Bau einfacher Schaltungen sammeln. Durch die Verknüpfung von Software- und Hardware-Komponenten lernen Kinder einen Roboter zu bauen und steuern. Zudem ist ein wichtiger Aspekt des Projekts die Nachhaltigkeit. Dafür werden alte Tablets und Smartphones wiederverwendet und der 3D-Druck für die Herstellung von Roboterteilen genutzt.

Die ersten beiden Applikationen sind entwickelt mit MIT App Inventor und richten sich an Jugendliche. Die Plattform ermöglichte die Steuerung vom Roboter, über Bluetooth. Die zweite App, erstellt mit Android Studio, ist für jüngere Kinder konzipiert und bietet eine intuitive Oberfläche sowie, Sensorfunktionen wie einen Lichtsensor zur Steuerung von Schleifen.

Die Arbeit umfasst einen Usability-Test zur Bewertung der zweiten App und zeigt einen Vergleich ähnlicher Bildungswerkzeuge an. Zudem werden mögliche Verbesserungen, wie die Integration von Arduino und künstlicher Intelligenz, aufgezeigt.

Abstract

This bachelors thesis focuses on the development of two beginner-friendly educational mobile applications aimed at teaching children fundamental programming concepts and hands-on experience in building simple circuits. These applications guide children through assembling basic electronic components, while also creating and programming apps. By combining software and hardware, the project offers a good introduction to system building, empowering children to construct and control their own robots.

One of the key aspects of the project is its emphasis on sustainability. It promotes the reuse of outdated tablets and smartphones, reduces electronic waste, and provides affordable educational tools. Children are also introduced to 3D printing, which allows them to design and create robot parts while assembling circuits, allowing them to understand how hardware components interact.

The first application, built using MIT App Inventor, is for older children and teenagers. It enables users to invent and operate Bluetooth-connected robot control systems. The second application, developed using Android Studio, is intended for younger children around the age of 10 years. It is equipped with a simple user-friendly interface that enables them to create and execute a program and see their results in real-time. The app also has interactive sensor features like a light sensor that alter loops based on the surrounding.

This thesis also includes an usability test to evaluate the efficiency and the overall experience of the second app. Moreover, it presents a comparative analysis of similar educational tools and explores potential future improvements, such as the integration of Arduino and artificial intelligence.

Contents

List of Figures

Symbols and Abbreviations

Symbol/Abbreviation	Meaning/Unit
STEM	science, technology, engineering and mathematics
MIT	Massachusetts Institute of Technology
LDR	Light Dependent Resistor
UI	user interface
UX	user experience
APK	Android Application Package
FPV	first person view
SDK	Software Development Kit
MOSFET	metal-oxide semiconductor field-effect transistor
IRF540N, IRF520N	a N-Channel MOSFET
DC	direct current
R_{DS}	drain-source resistor
R_1	gate resistor
R_2	voltage divider resistor
R_{LDR}	resistance of the LDR
$k\Omega$	Kiloohm a metric unit represents a value of electrical resistance
V_{in}	represents the supply voltage of 9V
V_{out}	the output voltage
Li-ion	Lithium-ion
PWM	Pulse Width Modulation

Chapter 1.

Introduction

Artificial intelligence [1] and modern technologies such as robotics [2] are becoming an increasingly fundamental part of our everyday lives. This thesis goal is to introduce children to the basics of programming and other technologies that are shaping the field of robotics. Not only will be programming be discussed in this thesis, but the project will also aim to educate children about key concepts in electronics, including circuits.

The knowledge of circuits is required for understanding how electronic devices function and communicate. By learning about circuits, children gain insight into a wide range of current and future technologies, which may soon be as common as smartphones and tablets are today. [3]

This thesis highlights sustainability and recycling. It shows children how technology can be used in innovative and environmentally responsible ways, for example by creating new gadgets from recycled materials. For instance, old tablets can be transformed into robots capable of performing simple tasks. With tools like 3D printing, children also explore how technology can be creatively used to solve challenges. [4]

Because of the fast advancement of the IT and STEM sectors, it's necessary that children stay up-to-date with these changes. Showing them activities like coding, building robots, and designing circuits not only deepens their understanding of technology but also improves creative use. Through this project, children will develop helpful skills such as creativity, problem-solving and critical thinking. [5]

Chapter 2.

Requirements

This requirements chapter presents the necessary parameters for the successful development and functioning of the Remote-App, the Robot-App, and the RoboKarel-App. It describes the purpose, scope, and main features of each app, focusing on the particular requirements of the target users. The target is to create educational tools that are user-friendly and simple, especially for beginners, so that kids and teens will be able to programming and robotics in a way that is both fun and accessible.

This part also focuses on the technical and non-functional aspects that should be the middle of attention. These are the key factors to guarantee that the apps function properly on Android.

2.1. Target Audience Analysis

In this thesis, the term "beginner-friendly" specifically refers to children, which is why they are the primary target group. Children are constantly in a learning phase where they develop new skills through play and exploration everyday. A user-friendly design that fits their needs enables them to learn programming without prior knowledge. Unlike adult users, children approach technology with curiosity and require therefore clear and visual instructions. The apps developed in this project are simplifying complex programming ideas by having interfaces that are easy to use, so that children gradually build their skills without feeling overwhelmed.

Defining the target audience for the two apps developed in this project is essential. Because of the different approaches used in their development, the two apps have each a specific audiences with unique needs. The first apps, that will be created by the users, is developed with MIT App Inventor, and is intended for teenagers around the age of 16

and above, while the second app, that was developed with Android Studio, is designed for younger children starting from the age of 10. The reason behind this selection will be explained further in below.

The first apps, developed with MIT App Inventor, require a certain level of technical understanding, especially in programming, designing and app functionality. Although MIT App Inventor uses a visual programming language - Blockly, the development process posed challenges even for individuals with prior programming experience. Younger users, especially children and teenagers, may need assistance when programming the app, particularly in more complex tasks such as app initialization or implementing Bluetooth functionalities. User feedback advocate that MIT App Inventor is not always intuitive, making it difficult for those without prior experience to navigate. Therefore, the target audience for this app is not only users aged 16 or older but also those with a technical interest and some familiarity with programming. [6]

In contrast, the second app, developed with Android Studio, is intended for children as young as 10. This app is based on simpler programming principles and requires significantly less prior knowledge. Similar to the Karel project from Stanford University, which introduces programming in a playful manner, this app is intuitive and accessible for younger users. Initial testing demonstrated that children could operate the app with minimal difficulty. While a technical interest is helpful, this app offers a much easier and more engaging introduction to programming compared to the first one.

Both apps are designed to help children and teenagers build a robot independently. However, adult supervision is recommended, particularly when soldering is involved. Teenagers aged 16 and older can learn soldering as a valuable skill under supervision, while younger children can use simpler assembly methods, such as copper tape or breadboards, to make construction easier. The 3D-printed parts for the robot can be produced once and reused as needed. The integration of 3D printing encourages technical creativity, enabling children and teenagers to modify or expand the robot according to their own ideas.

2.2. Requirements Analysis for Remote- and Robot-App

Purpose

This part of the thesis outlines the requirements for two interconnected applications developed using MIT App Inventor: the Remote-App and the Robot-App. Both apps

will be designed by children or rather teenagers, with the goal of introducing them to programming and basic robotics. The Remote-App will control the Robot-App via Bluetooth, allowing users to control the robot.

The Remote-Application will feature a simple control interface, including buttons for Left, Right, Forward and Stop, to move the robot.

Scope

This project centers on the design and development of two key applications: the Remote-App and the Robot-App. The Bluetooth connection is an important requirement of both applications and should be established for proper functioning of the projects. The basic control functions of the Remote-App will include commands such as Left, Right, Forward, and Stop.

Moreover, it can be noted here that this project has also the potential to add such features as Arduino integration or AI functionalities for more intressting robotic course of action in the future. Eventually, development could go as far as including gesture recognition and voice command control. Adding more hardware features to support Arduino sensors and actuators would further increase the scope of possibilities in robotic interactions.

Stakeholders

The main users of this project will be users of the age of 16 or above and trying to learn programming. The apps will be created by these students so they will enhance their skills in programming. The developer's duty includes the creation and maintenance of the apps, making sure they run properly and satisfy the users.

Teachers and parents are also the stakeholders, both of them are responsible for the educational process and the kids get the direction as they interact with the apps.

Definitions and Acronyms

MIT App Inventor: A visual programming tool to create Android applications.
Remote-App: Application with the purpose of controlling the Robot-App through Bluetooth.
Robot-App: Application that receives commands from the Remote-App and carries out the requested actions.
Bluetooth: Standard for wireless technology; exchange of data over short distances.
LDR- Light Dependent Resistor: A sensor used to find out the presence of light;

Functional Requirements

The Remote-App will feature four control buttons: Left, Right, Forward, and Stop. It will create a Bluetooth connection with the Robot-App to send movement orders. Via the input buttons, these commands will direct the robot's motions. For instance, pressing the Left button will activate the left light, while pressing the Right button will illuminate the right light. When the Forward button is pressed, both lights will turn on, and pressing Stop will turn them off.

The Robot-App will receive these commands from the Remote-App and execute them.

Non-Functional Requirements

The system should achieve Bluetooth connectivity within 5 seconds to ensure effective communication between devices. The controls of the Remote-App should behave to the user input with the least delay, thus giving the user a comfortable and precise user experience. In addition, the lights should react instantly if the buttons pressed, providing instant feedback.

The application interfaces will be designed to be user-friendly and intuitive, aiming for users to allow for ease of use. It is flexible across devices as the apps should be available on both Android phones and tablets. Besides, the apps should be battery-efficient, so it will save energy in long operation and avoiding power draining.

System Architecture Overview

Remote-App

This would provide the user interface with four buttons: Left, Right, Forward and Stop. It shall have Bluetooth connectivity with the Robot-App. Logic to drive lights on the button clicks.

Robot-App

Receive Bluetooth commands from Remote-App. Logic to drive movement and light on command.

Assumptions and Dependencies

For this project, it is assumed that users will have Android devices equipped with Bluetooth capabilities. Both the smartphone and tablet need to enable Bluetooth and be in the relatively close distance to each other, preferably stay within the limit of 5 meters. The possibility of interference or brief disconnections in the Bluetooth connection is not

a factor that will affect the overall performance of the system. Moreover, users are required to possess at least a hands-on knowledge of MIT App Inventor, although they can turn to the tutorials if they are not capable of that.

Limitations

Nevertheless, there are some of the project's restrictions. The apps will be designed solely using MIT App Inventor, which in turn is a restrict that no more advanced coding is used. Anyway, the programs will be executed only on Android devices, as for iOS compatibility, it is outside the scope of this project. Finally, the Bluetooth connection should not only be stable but it must also keep the communication within the standard range for the reliable of operation.

Acceptance Criteria

The Remote-App is connect with Bluetooth to the Robot-App which allows fast and proper communication. Commands sent via the Remote-App are accurately analyzed and performed by the Robot-App. For example, if you click the Left or Right button, the light on the respective side will switch on. The Forward button will turn on both lights, and the Stop button will make them go off.

Both apps have a user-oriented and straightforward interface that enables teenagers to use them with added comfort. With the target users, the system is set to pass usability tests, therefore making the interaction more pleasant. Throughout testing, the Bluetooth connection will be stable all the time, thus achieving the desired consistency.

User Stories

As a 16-year-old user, I want to connect my Remote-App to the Robot-App via Bluetooth, so I can control the robot seamlessly. This allows me to operate the robot and its features with ease.

As a developer, I want the commands sent from the Remote-App to the Robot-App to have minimal delay, ensuring a smooth and responsive experience for users. Additionally, I want the robot's lights to respond instantly when a button is pressed, providing immediate feedback to the user.

As an educator, I need the application to be simple and beginner-friendly, ensuring students can learn programming concepts without frustration and in an intuitive manner.

2.3. Requirements Analysis for RoboKarel-App

Purpose

The application is a tool for children in terms of visual programming, which allows them to insert commands and see the resulting visual output. It comes with interactive elements, such as a light sensor that controls the loops according to the conditions of the surrounding space. The program sets out to introduce the kids to the basic concepts of programming like looping and conditional statements in an user-friendly and engaging way.

Scope

Children will be able to give commands like "forward, left, right or stop" and will be able to do so through a simple and easy-to-use interface. The visual output over the display will consist of two rectangles onscreen, which will change their colors accordingly to these commands and a comic like face. On top of that, the project will provide looping functions for example, a checkbox or a loop button can be used to activate a loop based on conditions set by light sensors.The target audience of the project are children around the age of 10, so the interface will be designed to be very simple and easy for them to use, not including unnecessary complexity.

Future Considerations

In further development the project could add more complex sensors or add commands to enhance functionality. There is also an opportunity to employ "sound feedback" and the integration of additional sensors to do more advanced visualizations and interactive features, thus making the user experience richer.

Stakeholders

The stakeholders of this project are mainly children who want to learn programming concepts. The developer, who build the app using Android Studio, plays a key role in making sure that the app keeps being suitable for beginners. At the same time, teachers and parents will be the ones to guide the kids through their learning experience and make sure they engage with the app properly.

Definitions and Acronyms

Light Sensor: A hardware sensor detecting light level around it.

Loop: A program structure that executes a sequence of commands continuously while a condition is true.
Command Input: Commands users can input into the app include "forward", "left", "right", "stop" and "loop"
Front Is Clear: A condition based on the LDR detecting sufficient ambient light.

Functional Requirements
The app will feature an simple interface with action buttons for commands such as "forward," "left," "right," "stop," and a dedicated "loop" button. Users can input commands either through text or by pressing the corresponding buttons. The app will also include a loop functionality, which can be turned on in three ways: by checking a "Loop" checkbox for conditions that occur automatically, by clicking the loop button or typing loop manually into the text field.

The light sensor is activated, the app will now run in a loop always executing commands when "Front Is Clear" is met, this means light is sufficient. The app will also present graphic feedback by showing two rectangles that will indicate which command is running. When the "forward" command is active, then the two rectangles will remain white. For the "left" command, the left rectangle will turn white while the right remains black, and for the "right" command, the right rectangle will turn white while the left stays black. For the "stop" command, both rectangles will turn black.

In case the light sensor detects insufficient light, indicating the "Front Is Not Clear" condition, the app will automatically stop executing looped commands.

Non-Functional Requirements
Quickly responding to user input, the app should guarantee that interaction is smooth and immediate. The design will be simple and intuitive, it will have large buttons, and it will be clear and easy to understand for 10 years old without being distracted. In addition to this, the app will be very light and will be optimized to run well on mid-to-low-end Android devices without performance issues.

The light sensor will work in a constant monitoring mode, with real-time feedback being delivered to the user without noticeable latency. For every user command, the app will clearly visually show the feedback, so the experience will be engaging and easy to follow.

System Architecture Overview

User Interface

This would present a big, operational button for commands and a text entry area for manual input of commands. It shows the command representation through two rectangular areas. Both change color depending on what the user inputs. A loop function can be engaged either with a checkbox that says "Loop" or by a button dedicated to a loop.

Light Sensor Monitoring

The loop function is connected to the light sensor readings. If it detects that the light is below the required value, then the robot stops.

Execution of Commands

"Forward", "left", "right" and "stop" commands are to take control on the screen imagery with respective highlighted visual feedback.

Assumptions and Dependencies

It is assumed that users have an Android device that comes with lights sensor. Apart from that, the application is made with the presumption that the highly likely target group, which is children, will have basic literacy skills. This will help them to not only read but also understand simple instructions. The creation of the app using Android Studio tools and libraries for sensor data processing as well as visual programming implementation.

Limitations

The application will be fully developed in Android Studio, which may impose certain limitations on functionality compared to other platforms. For the app to function correctly, particularly the loop control feature, it must run on devices equipped with light sensors.

Acceptance Criteria

Users can enter commands to the application either by typing or using button presses. Through these commands, actions such as forward, left, right, and stop are visually represented by changing colors on the following screen. The loop function works as designed, repeating commands when a checkbox or loop button is activated, as long as the light sensor detects enough light. If the light levels drop below a certain threshold, the application will automatically stop executing the looped commands. It is oriented towards the 10-year-old age group, the app is made very clear and engaging for the target

audience, which successfully passed usability testing as result ensuring children's good experience using it.

User Stories

As a 10-year-old user, I want the ability to input commands and immediately see how they run on the screen, allowing me to learn how programs work through real-time feedback. Additionally, I would like to have a simple way to trigger loops, such as using a loop button, so I can observe repeated actions based on changing light conditions, making it easier for me to understand how loops function.

As a parent, I want this application to be intuitive and easy for my child to understand, ensuring they can engage with it without frustration.

As a developer, the light sensor should accurately control the loop function, enabling children to interactively and effectively learn programming concepts based on the real-time changes in light conditions.

Chapter 3.

Software Tools Used in Development

This bachelors thesis relies on several essential software tools to ensure the successful completion of the project. Below is an overview of the key programs, their functionalities, and how they contributed to the development process.

3.1. Figma

The model of the RoboKarel-App was built with Figma, which is a cloud-based design tool mainly used for UI and UX. Figma's real-time collaboration feature made it easy to handle different design iterations and get quick feedback. With this functionality, the project could be worked on by multiple people at the same time, which was a big plus.

The app's interface was created in Figma with an emphasis on usability, so that a neat and simple layout is being ensured. Figma's vector editing, components, and prototyping features, among others, enabled the visualization and refinement of the user flow before moving into development. The app's user interface was built using Figma. Its simplicity and flexibility made it a key instrument for the app's design layout before its development. [7]

3.2. Android Studio

The RoboKarel-App, was created using Android Studio, which is Google's official IDE for Android applications, was the platform used for the development. Android Studio is a cloud-based development platform based on the IntelliJ IDEA framework that offers a full arsenal of tools for developing, debugging, and optimizing Android apps. Java

as well as Kotlin are the two programming languages available, hence the flexibility of programming language has been provisioned. The emulator in the integrated system is one of its differentiating pros. It enables comprehensive forensics on virtual units before deploying them to the hardware. [8]

3.3. MIT App Inventor

Another tool in this project is the MIT App Inventor [9] web-based development environment that was initially created by Google but is now maintained by the MIT. It is designed to help users, especially beginners who want to create mobile applications using a visual programming language. [10]

Namely Blockly - this is a programming language that graphical interface has transformed from complex into the very simple one for its users. They are able to create a program just by joining different blocks that will be automatically turned into code in the background. The visual method brings forth to teach programming logic without having users actually getting to the syntax errors. On the other hand, the MIT App Inventor is somewhat limited. As everything is created and tested on the platform, it can be pretty resource demanding, especially if it is about complex projects. At times the problem might be the platforms over-reliance on online resources which can lead to difficulties like difficulties in exporting APK files. [11]

3.4. Blender

For the creation of the robot's 3D model, Blender, a free and open-source software for 3D modeling, animation, simulation, and rendering, was utilized. Blenders versatility makes it a popular choice among designers, artists, and engineers for creating detailed and functional 3D models.

In this project, Blender was used to design the robot, which was later produced using a 3D printer. Blenders extensive modeling tools made it possible to create precise and aesthetically pleasing designs that met both functional and visual requirements for the project. [12]

3.5. Cura

For preparing the three-dimensional model that was made in the Blender application for printing, Ultimaker's open-source slicing software known as Cura was used. The slicing method consists of 3D models that are distributed within a given number of layers, which are then produced by the 3D printer in a particular order. Cura provides the user with the option to choose from a variety of settings for print parameter management including control over layer thickness, speed of printing, and infill patterns.

Through the utilization of Cura, the 3D model has been modified for printing, which ensures that the correct results are obtained in an efficient manner. According to the software, the optimization of the printing process was done, and thus the successful printed robot was integrated into the project. [13]

Chapter 4.

Competitor Analysis

This chapter compares the project against other existing platforms and robotics solutions, examining similarities and differences to point out both the strengths and the limitations of this project. The analysis sets out to give a clear idea of how the present project is positioned in the competition, emphasizing the main features such as ease of use, educational value, and accessibility, especially to the younger kids.

4.1. Programming

4.1.1. MIT App Inventor vs. Other App Development Platforms for Kids

This section compares MIT App Inventor with several competing platforms that also offer app development tools for younger audiences. The analysis focuses on ease of use, available features, accessibility, and educational value, particularly for platforms designed for children aged 12 and above.

Thunkable

Thunkable is a no-code platform like MIT App Inventor, designed for creating mobile apps for both iOS and Android. A major advantage of Thunkable is its cross-platform compatibility, allowing users to develop apps for both iOS and Android, unlike MIT App Inventors Android-first approach. Thunkable is also a user-friendly package, characterized by a number of features like API integration and real-time app testing and thus, can fit into the demands of advanced users. However, the free version of Thunkable is limited as far as app storage and features are concerned, whereas MIT App Inventor is totally

free of charge for projects basic app-building. Thunkable is a good fit for students who have some experience with block-based programming and are ready for more advanced cross-platform development. [14]

Kodular

Kodular is a platform that provides a block-based interface similar to a MIT App Inventor but with some additional customization options. It offers advanced features like in-app purchases, ads, and a more customizable interface, thus making it more attractive to users who want to develop applications for commercial use. Kodular is open-source software just like MIT App Inventor and has a big supportive community. Besides, its interface may scare the beginners because of its complexity. The students who want to deal with other components which are more complex than just simple app development should use Kodular. [15]

Scratch

Scratch is a well-known platform that teaches coding to children by creating games and animations. By having a block-style interface, this tool in Scratch is simple to use and very attractive for the students and beginners. Scratch is most broadly used in the classrooms for teaching programming logic and is as well widely supported by the online community and a lot of resources are provided for educators. Nonetheless, Scratch does possess some limitations, being a browser-based projects only, and cannot support mobile app development, which is the main difference from MIT App Inventor. Scratch is a great teaching aid for the fundamentals of programming and game design, yet it isn't the best option for developing mobile applications. [16]

Tynker

Tynker is a coding platform that offers a wide range of educational choices from simple block-based coding to more challenging programming languages such as Python and JavaScript. The company promotes active learning through interactive projects in app development, game design, and robotics, providing a more complete coding experience. Tynkers main goal is coding in general, therefore, it is less focused on app development. Tynker would be the ideal choice for those students who want to delve into various coding subjects but may not be the best option for those specifically seeking app development tools. [17]

Code.org
Code.org is a popular platform that provides coding tutorials and activities to beginners. Its interface is block-based and is specifically developed to help students learn programming logic by following simple lessons. Code.org provides rich options and lessons for teachers and students, which is a great plus for the students who have just started learning. On the other hand, it places more emphasis on the learning of general coding skills than the applications of programming. However, the main difference between the two is that Code.org does not offer a wide range of resources for the deployment of real apps. This is indeed a nice entry point to the basics, but not really the best option for those who want to learn how to build mobile apps. [18]

4.2. Robot

4.2.1. Competitor Analysis of the DIY Robot

After studying the DIY Smartphone Robot project by VoltPaperScissors, there were a few of its fundamental concepts that were modified for this bachelor project while the rest were original elements that were incorporated in order to create a unique approach. Both projects emphasize the application of light sensors for robotic control; however, this project applied the concept of a tablet instead of a smartphone and also two custom-developed apps that are tailored to specific goals. In the same way as the DIY Smartphone Robot, this project aims at making low-cost, accessible robotics that are available to people and thus making them get involved with technology both in education and in the interactive way. By enhancing the original idea and incorporating changes, this project presents a one-of-a-kind input that still preserves the main framework of light sensors for the movement control. Both projects have such a commitment to simplicity and innovation that they are so much the same and they bridge the gap between education and playful experimentation.

For further exploration of the DIY mobile phone robotics frontier, a competitor analysis was made on a parallel way. The project Mobbob by Cevinius, OpenBot by Intel Intelligent Systems Lab, and RoverBot (FPV Rover) which are all competing in the robotics education field and have their peculiar features and uses are also included. [19]

Mobbob by Cevinius

Mobbob is a robot which can be controlled by mobile phones. It can move, dance and interact with the environment. The device features a camera for visual input that makes it possible to program more complex behaviors using open-source software. One of Mobbob's main advantages is its flexibility since the addition of visual input and environmental interaction makes it more functional than those that only light sensors. Furthermore, the modular design of Mobbob enables the people to increase its abilities easily, so it is a great base for customization. It can be a flexible platform.

However, along with this enhancement of functions there is also the addition of more complexity. Indeed, Mobbob requires a higher degree of technical skills to build and program, which makes it less accessible to younger users or beginners. However, Sustainability is promoted as the users are allowed to utilize the robot for a long time by upgrading and reusing the robot through the open-source nature of Mobbob. [20]

OpenBot by Intel Intelligent Systems Lab

Taking a more advanced way, OpenBot uses the smartphone on Android as a robot's "brain." The smartphone has control and computing power, which are needed for navigation and obstacle avoidances. OpenBot can be improved with artificial intelligence AI to perform more complicated tasks suited for simpler robots, which are controlled by light sensors.

The scalability of OpenBot is a major strength, as its open-source platform allows users to keep adding new features. On the other hand, it is very expensive due to its use of high-end smartphones and lots of other gadgets, which might scare off the ordinary DIYers. Even so, OpenBot is a green initiative because it gives old phones a new purpose. Yet, advanced hardware may require more frequent replacements if not handled properly. [21]

RoverBot (FPV Rover)

RoverBot, or FPV Rover as it is also called, is the robot based on Arduino that is controlled by a smartphone via Wi-Fi or Bluetooth. This robot is designed for speed and agility and comes with an FPV camera, so you can have a first-person look at it as the robot moves.Its design is highly customizable, providing a platform for hobbyists who want to add more functionality to their robot.

While RoverBots FPV feature enhances interactivity, its focus on agility and customization may not align with educational goals aimed at beginners. The technical skills required for constructing and programming RoverBot are moderately advanced, thus rendering it less accessible to those who are just getting started with robotics. Nevertheless, the fact that it is cheaply priced and is made of readily available components is a big plus for sustainability. As such, it is surely a DIY solution for enthusiasts who require flexibility in design. [22]

4.3. Karel-Apps

4.3.1. Competitor Analysis of the RoboKarel-App

The RoboKarel-App targeted at kids around the age of 10 with the intention of teaching them basic programming and robotics in a fun and engaging way. The App is inspired by the classic "Karel the Robot". Richard E. Pattis of Stanford University developed Karel to teach programming skills to students. The program was still in operation. The program now has a more streamlined visual interface and includes such technologies as light sensors, which users can utilize to control loops depending on environmental conditions. This method puts together the Karel's simplicity and the latest technology to invent a more engaging learning experience. [23]

Karel the Robot (Apple Store)
The Karel the Robot app on the Apple Store stays true to Richard E. Pattis' original design. It is developed by an educational software team, this edition facilitates kids and teenagers to programming logic without the need of any coding knowledge.

One of the application's major advantages, is an easy-to-use visual interface for new users, which is meant to attract kids. It has interactive tasks and challenges that help learners to learn through play. As such, it is a valuable aid to beginners in programming. Not to mention, the software works smoothly with Apple's other devices allowing users to use it without any hassle and gain extra functionalities through using Apple's ecosystem.

However, the apps simplicity, while beneficial for beginners, can become a limitation over time. As users become more comfortable with programming concepts, they may find the app less challenging. Another drawback is its exclusivity to iOS, leaving Android users

unable to access it. Moreover, the apps basic command set restricts creative freedom for those who may wish to tackle more complex projects. [24]

Karel 1981 (Google Play)
Karel 1981 retro version of the original Karel the Robot, was developed by Andreas Schilling. With its 80s inspirations, the application is fully in line with the original Karel concept and adorned with a nostalgic design featuring simple programming tasks. This application serves as a perfect introduction to robotics and programming.

The apps nostalgic, retro design provides a unique user experience, particularly appealing to those who appreciate the simplicity of older interfaces. Available on Android, Karel 1981 reaches a broader audience than the iOS-only Karel the Robot. It offers basic programming tasks that are perfect for beginners, allowing users to grasp foundational programming logic without overwhelming complexity.

Yet its old-fashioned interface may not be as attractive to today's kids, who are used to fancier-looking websites. Besides, Karel 1981 is not only behind in innovative features as compared to other newer apps, but also its limited functionality may confine the interaction and creativity of the users who seek more complex challenges. [25]

Swift Playgrounds (Apple)
Swift Playgrounds is an application developed by Apple which is aimed at children and teenagers to teach them the Swift programming language through of interactive, visual tasks. Unlike Karel-based apps, Swift Playgrounds directly introduces users to a professional programming language that paves the way for more advanced study.

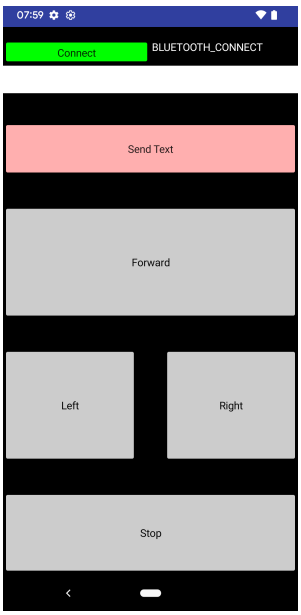
One of Swift Playgrounds' key advantages is that it immerses users into Swift, a programming language with a really high level of expertise required widely used in professional development. Its interactive learning platform presents increasingly more difficult tasks, thus enabling the choice of the most appropriate way to learn and generate fun throughout the whole process. Besides that, its deep integration with the Apple ecosystem ensures a seamless user experience across iOS devices.

Yet, Swift Playgrounds may be too difficult for younger kids as it includes a programming language. Though, it is an essential tool for those who want to learn professional coding. It is available only on iOS devices, limiting its accessibility. For absolute beginners, especially younger children, the app might require more time and effort to become familiar with the language compared to simpler apps like RoboKarel. [26]

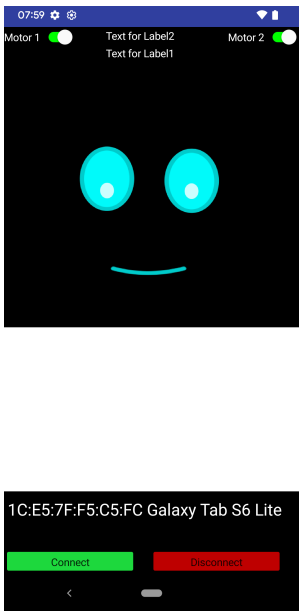
Chapter 5.

Development of the First Apps

The core idea of the development of these apps, which allow children to use them and program them, all by themselves, is to act as an introduction to programming. The first step asks to start off with two linkable apps the Remote-App and a Robot-App. The development of both apps was achieved using MIT App Inventor and "Blockly" the visuelle programming language which is based on the building block principle. The project aims not just to allow children to use the apps but also to learn programming through developing them.



(a) The Remote-App; Source: Author's own Screenshot



(b) The Robot-App; Source: Author's own Screenshot, Face used in App is from VoltPaperScissors project

Figure 5.1.: Remote- and Robot-App created with MIT App Inventor

The Remote-App controls a number of tasks via Bluetooth connectivity. It has five buttons of which "Forward", "Left", "Right", "Stop" are included, and a list for pairing Bluetooth devices is also presented. In this arrangement, the Robot-App is the "Client" and the Remote-App is the "Server".

The Robot-App includes a button to establish the connection and two switches, located in the top left and right corners, which manually control the lights and therefore the motors. The interface also features a face displayed at the top and two large rectangular fields at the bottom, which are initially black and change color based on the commands from the Remote-App. [27]

5.1. The Design Interface of MIT App Inventor

The design interface of MIT App Inventor allows users to visually design the apps frontend by selecting necessary elements. Once the design is set, users can move to the programming interface to assign specific functions to buttons and other UI components. The interface is highly intuitive and suitable for older children. [28]

One of the drawbacks is the fact that adding buttons is simple to do but, as the program gets more complicated structuring is of utmost importance. There are doubts regarding whether kids will be able to stick to this structure while they are making more complex programming tasks.

Another issue is the initially awkward web-based display of the apps frontend. However, MIT App Inventor offers a solution by enabling users to simulate the app directly on a smartphone or tablet via a QR-code, allowing real-time visualization of changes made to the frontend on the mobile device. [29]

With regard to the age, these tools are said to be the most appropriate for children of 16 years and above who are mostly able to solve simple research tasks on their own. The MIT App Inventor large community is also the reason why the tool is a great help in case of questions and gets you help quickly. These platforms, some of which are operated by people who serve as moderators, make sure that answers to questions are given in time and with the highest accuracy. [30]

5.2. The Programming Interface of MIT App Inventor

The programming interface of MIT App Inventor is straightforward. Users may choose buttons from a list on the left and allocate functionalities to them. A broad selection of Blockly blocks can be used, which lets you create different programs. Yet, this is not true for beginners, who may find this variety intimidating.

Since this project assumes no prior programming experience, the appendix in this thesis includes a guide to help children work with Blockly. Some tasks include prewritten code solutions to assist in the learning process.

During discussions with the supervising professor, it was identified that handling Bluetooth permission requests on newer devices is particularly challenging. For this reason, it is anticipated that this section of the code will be prewritten, while simpler tasks will be left for the children to program. Permission handling on newer devices, especially from Android 11 onwards, requires additional code blocks due to stricter security protocols, making it more complex to establish the desired connection.

5.3. Bluetooth Communication and Control in the Remote-App

This next part explains the code of the Remote-App briefly, this also can be found in the appendix.

In the initialization phase, several important functions are loaded. The app first looks for Bluetooth on the device and if it is not powered then the notification appears with the exact message. The app additionally verifies if the device runs Android 11 or higher by checking its API level because newer versions utilize explicit permission for Bluetooth usage. The app records the Bluetooth permissions status in TinyDB as a way for it to keep track of the required permissions that has been granted.

On the right side of the initialization block, the Permission Granted block is also there. This block is responsible for the connection query. The permission is granted and it is ensured that the connection is established correctly, and the permissions for scanning and connecting to Bluetooth devices are processed.

With the block below the initialization that assigns the Connect button the function of allowing the Bluetooth server to accept incoming connections. This indicates that the server is ready to accept connection requests from another device.

Further down are blocks that assign the Forward, Left, Right, and Stop buttons the function of sending specific commands via Bluetooth when clicked. Each button is programmed to send a prearranged number: "1" for Forward, "2" for Left, "3" for Right, and "4" for Stop. These numbers control the movements of the connected device.

The system is set to process commands with a short delay and in the proper order. The control of kids is therefore not only seamless but also free of errors, due to the fact that signals are not mixed when multiple buttons are pressed but commands are processed sequentially. Consequently, the younger generations interacting with the app find easy and intuitive to use the device that they do not even need to worry about any technical problems.

5.4. Bluetooth Communication and Control in the Robot-App

The code for the Robot-App. Similar to the Remote-App, begins with an initialization process that checks the device's API level and sets up the ListView for server discovery. Since the robot functions as a client in this scenario, the Bluetooth addresses and names are loaded into ListView1. Additionally, the status of the Bluetooth permissions is stored in a TinyDB, allowing the app to verify whether the necessary permissions have been granted.

Like the Remote-App, the Permission Granted block is the one that has a major impact on the whole system. It works as a guarantee that new devices would be registered in the system and the Bluetooth functionality works properly. The app is asking for permissions to both the scan for Bluetooth devices and connect them. Once permission is granted, these permissions are kept in TinyDB, and thus, no other permission requests will be needed in the future.

The green Connect Button represented a Bluetooth connection to another device via Bluetooth. After pressing this button, ListView1 is now visible on the screen, and it shows us the devices that can be chosen. Addition of a red button is optional for disconnection,

but it is not mandatory, as the Bluetooth connection can be terminated by closing the Bluetooth panel on the device.

Once the device is selected the ListView1.AfterPicking function is triggered. This function shows the device you have selected and at the same time, Label1 updates to inform that the connection is successful. Moreover, the system data sent by the server (i.e., the Remote-App) is also processed and displayed in Label1.

The Timers block on this feature, on the other hand, tracks the Bluetooth connection and checks that data is getting received continuously. The received data is kept in the global variable name. Based on the incoming data, for example, "1" to indicate Forward, the colors of the rectangular fields in TextBox1 and TextBox2 change respectively. In this case, if "1" is received, both of the rectangular fields are set to white as can be seen in the code.

Another way of improving this code may be by using more explicit naming of the components, e.g. Label1, Label2, TextBox1, and TextBox2, which can help improve the overall readability and maintainability of the code. Through this, the purpose of each component would be more visible.

5.5. Code Explanation for Older Devices

This section describes the code for the Robot-App which is similar to the Remote-App and it has an initialization process that verifies the device's API level and prepares the ListView for server discovery. Since older devices are considered as part of a recycling initiative, it is essential to point out that these devices often require fewer permissions - making the initialization process more streamlined. For example, the permission block - Screen1.PermissionGranted is mandatory for Bluetooth connections on newer devices are not as important on older devices. The code has been modified by taking out the permission blocks and the older devices adjustment has been included.

5.6. In addition

Moreover, in the future, thereafter an Arduino would be used to enhance it further, the idea was set aside as it was seen to add complexity to MIT App Inventor which was already quite challenging in itself. Nevertheless, Arduino with MIT App Inventor is easy

compatible, therefore it could be used to raise the projects difficulty level in the upcoming projects. Apart from ChatGPT which can be integrated to the application through community-supported extensions, MIT App Inventor also enables other advanced features. Beginners can utilize these functions in the early stages of usage. However one potential concern raised during research is the longevity of MIT App Inventors build servers. The APK build process currently relies on a web-based infrastructure that serves millions of users and projects. Although MIT continues to enhance its backend systems, including iOS builds and Android SDK updates, occasional issues arise due to resource limits, particularly when building large APKs. Questions have been raised about the long-term sustainability of these servers, as more complex projects and higher demand may push the infrastructure to its limits in the future. [31]

Chapter 6.

Development of RoboKarel-App

After weighing up the possible complications of having a remote control and robot functioning in the application particularly for young children, it was decided to remove the remote control feature and focus on teaching basic programming concepts only. The developed app is a visual programming environment designed specifically for kids, with which they can input and run simple commands. The commands like "forward", "left", "right" and "stop" give different visual effects. For example, if the "forward" command is given, then both rectangular fields will turn white, while the "left" and "right" commands will make one field light up and the other one black. The "stop" command completes the movement by turning the two fields black.

As soon as the "Play" button is pressed, commands run in a sequential manner and visual effects display on the second screen. Every command has a fixed time: "left" and "right" are almost two seconds long, while "forward" is around five seconds, thus movements are clearly identified. When all commands are completed or when the light environment changes, the app is reverting to the coding screen which lets users modify or enter new commands for a continuous learning loop.

This app is designed to teach programming skills in an engaging way, eliminating the need for children to manually program external tools. Instead, they learn by writing code directly within the app. Inspired by Stanfords "Karel the Robot," the app helps children understand basic programming principles by allowing them to input commands and immediately see the results through a visual interface. [32]

The app features two main screens. On the first, children can input their code in a text field. A short delay follows, allowing time to place the tablet inside a robot before the code execution begins, controlling the robots movements based on the entered commands.

A loop function adds complexity to the app. Using the devices light sensor, the loop checks whether the "Front Is Clear," indicating sufficient light, and repeats actions as long as this condition is met. The loop can be initiated either by entering the "loop" command or selecting a checkbox. While active, the light sensor continuously monitors the environment, and if the surroundings become dark ("Front Is Not Clear"), the loop stops and the app returns to the coding screen.

The primary goal of the app is to create an interactive learning experience where children can control a robot using simple commands, which can be entered either through buttons or typed directly. The app reinforces understanding by providing immediate visual feedback, helping children grasp core programming concepts.

The overall goal of developing this app was to create an interactive tool that engages children in learning programming. By integrating visual feedback and sensor-based controls, the app provides a dynamic learning environment that aligns with real-world programming applications. The combination of immediate feedback and robot-controlled actions reinforces key concepts in an accessible format.

6.1. Prototyps

6.1.1. Prototyp 1

The first prototypes were made in Figma. The first version, as shown in the figure 6.1, allowed the users to enter code using common punctuation marks, such as brackets and semicolons. The original idea was that, if an error occurred, a message would appear below the input field that would identify the issue, for instance, a missing bracket or a semicolon. After consulting with Professor Lano, the idea was slightly changed. Concerns were raised that the app's main user group, the children, might be overwhelmed or discouraged by the syntax requirements. Since even university students usually make mistakes in correct syntax, it was very important to make coding easy and to avoid learners' frustration.

The application is easily navigable and follows a three-step process. The first screen features a main text box where users enter their code, and the prompt, "Write your code here" appears. Here is a green button with a play symbol that lets you execute the code. The second screen shows the code you entered along with any error messages concerning the syntax mistakes. Users can also execute the code by clicking the play symbol.

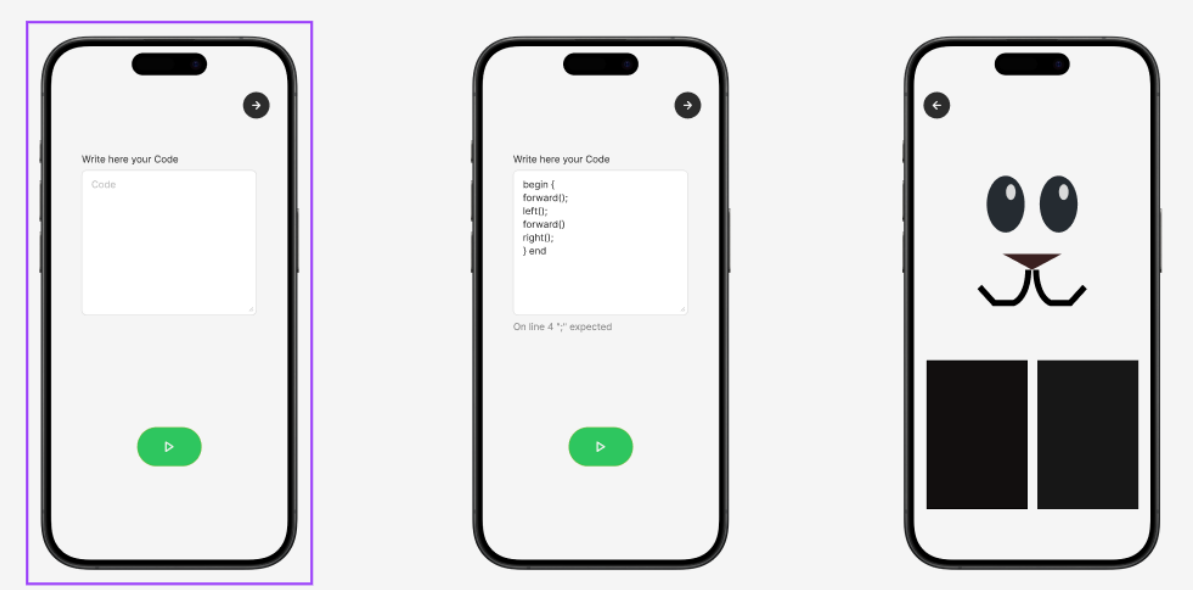


Figure 6.1.: The first Prototype made with Figma; Source: https://www.figma.com/design/

On the last screen of figure 6.1, the application shows the output of the code, with two black squares. The output has also an image of a friendly smiling face, this feature adds a child-friendly character to the application.

After the child inputs the code in the given text box and clicks the green play button, there are two options: if the code contains an error, it shows an error message at the bottom of the screen; if the code is right, the next screen is displayed. A 10-second time frame is then given to the child to put the tablet in the holder. After that, the code executes, making the robot move according to the commands. The execution is done, so the app goes back to the initial screen. Each light or code command is meant to be on for five seconds, although this can be changed in the future versions.

6.1.2. Prototyp 2

The second prototype's interface has a text field but it does not allow users to type text directly. Rather, text is generated using the button options which are displayed underneath the text field. The reason for this is to make programming easier for kids, making it possible to compose a program with a single button push. The buttons initially included artificial parentheses and semicolons but after further conversations, this idea was simplified.

Figure 6.2.: The second Prototype made with Figma; Source: https://www.figma.com/design/

This version of the application features a minimalistic, child-friendly design across two screens.

The first screen in figure 6.2 presents a text box in the upper half of the interface, with the prompt "Click the Buttons for your Code." Under the text box are four rectangular buttons named "Forward", "Left", "Right" and "Stop", which enable users to input commands with a single click. A prominent green button is provided for executing the commands, and a circular navigation button with a right-pointing arrow at the top of the screen allows users to proceed to the next screen.

The second screen graphically shows the commands' outcomes. In the upper part of the display, there is a funny, cartoonish-looking, big-eyed dog/cat with a tiny nose and whiskers, who makes the interface seem cheerful. The face is followed by two big black rectangles that are used as the visual indicators of the robot's movement. A left-pointing navigation button allows users to return to the previous screen.

Such a design actually eliminates the complexity of coding for kids and emphasizes the process of making a code by buttons. So, it is like the development of the information

provided in the first prototype. However, the process of interaction has been made more linear to ensure better understanding from the user.

6.1.3. Prototyp 3

Taking advantage of insights gained from the first two prototypes, the third prototype is the combination of the key elements of the two designs. In this version the error indicators, semicolons, and parentheses are completely deleted. Users are offered a text field where they can input code either by using their own keyboard or by pressing the buttons below the text field. Through this hybrid method, the advantages of both are enhanced while the previous prototypes remain unaltered in terms of simplicity and structure such that the coding process remains easily accessible and user-friendly for a child.

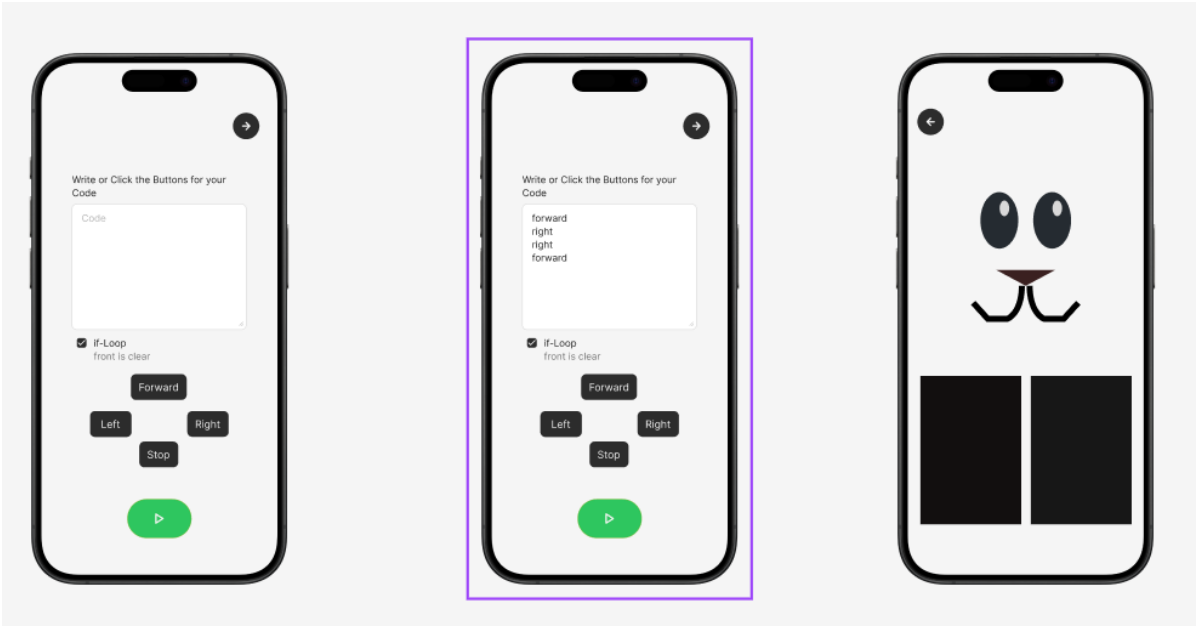


Figure 6.3.: The third Prototype made with Figma; Source: https://www.figma.com/design/

After the user has put in the code, he can press the play button, wait for 10 seconds, and the system will execute the code. This new input style is more flexible than the previous one which has the same simplicity as the original for kids. However, it is more intuitive as the earlier prototypes.

6.1.4. Final Version

A small change brought forth the new prototype whose name is the RoboKarel-App and this is the final version officially programmed. Introduction of a new functionality such as the addition of a loop checkbox is the main distinction of this prototype from Prototype 3. The program will keep on running as long as the front is clear if the checkbox is selected or the "Loop" term is included in the code. So the loop will be repeated until an object is detected in front of the tablet, introducing a dynamic character of code execution process.

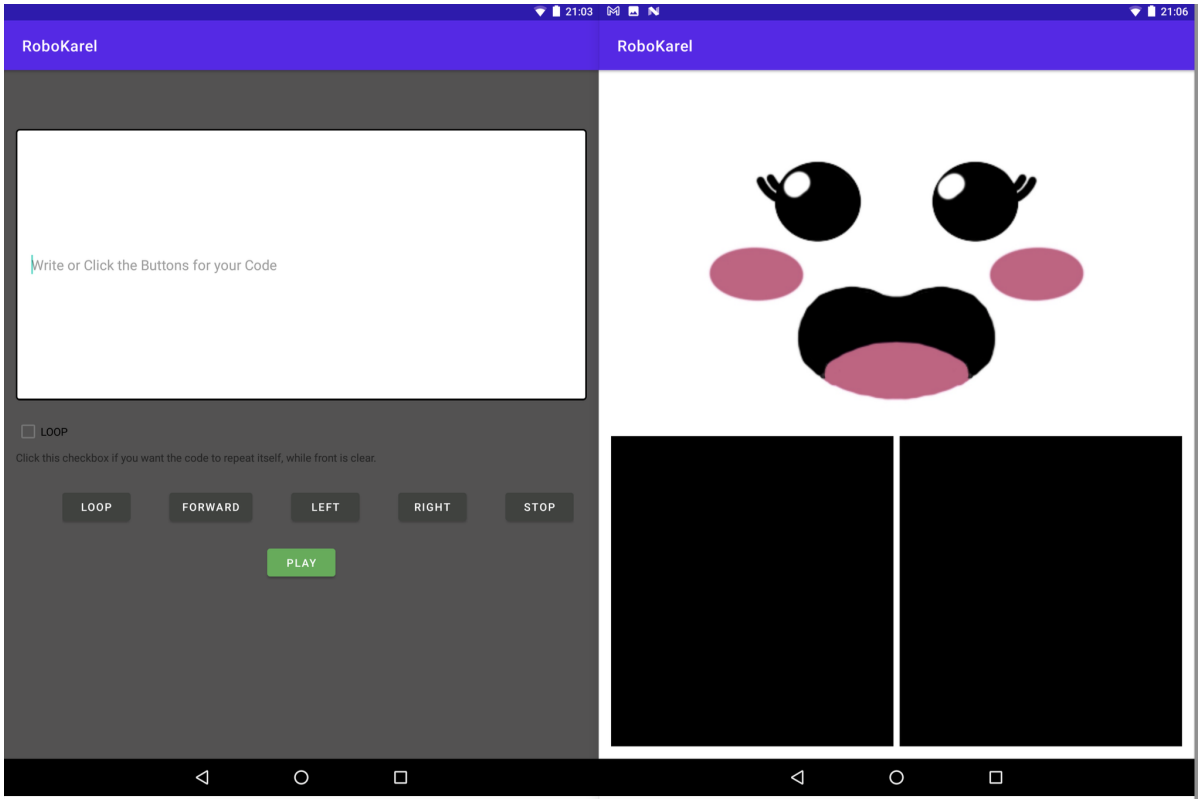


Figure 6.4.: The final App on the the left side is screen 1 and on the right the execution screen 2; Source: Author's own screenshot

The initial screen of figure 6.4, now darker in design, features a centrally positioned text field with the placeholder "Write or Click the Buttons for your Code." A row of horizontally displayed buttons offers the following options: "Loop", "Forward", "Left", "Right" and "Stop." These buttons help users to insert commands using the text field without any manual typing, manual input is possible still. Below the buttons, a checkbox labeled "If-Loop" is accompanied by the condition "Front is Clear." At the bottom of the

screen, a prominently positioned green button gives users the power to execute the code.

The second screen displays the results of the inputted code. It features two black squares at the bottom and a simple, friendly face with large eyes and lashes at the top, giving the application a playful and approachable character likely to appeal to children.

An example of how this feature works in practice is as follows: First, the loop checkbox is selected, and the commands "Right" and "Left" are input via the designated buttons. The user then activates the playback function, with a 10-second window provided to attach the tablet to the holder. Once the application begins, the right square is illuminated, and the robot moves to the right for five seconds, making a circular turn. After five seconds, the left light activates, the right light turns off, and the robot rotates to the left for five seconds. This sequence of right and left movements continues until an object is placed in front of the tablet, causing the code to stop as the "Front is Clear" condition is no longer true.

The additional functionality ensures that if the loop checkbox is selected, the code will continue repeating until an obstacle is detected in front of the camera. Such obstacles could include a piece of paper, a leg, or a hand. This feature also addresses an issue that was not initially anticipated.

The initial code configuration relied on the proximity sensor [33], a common feature in most smartphones. The proximity sensor detects nearby objects, such as when a phone screen turns off during a call or when placed face down. However, it was later realized that the project would primarily use tablets, which often lack proximity sensors, particularly older models [34]. Once this issue was identified, a solution was implemented.

Through online research, it became apparent that a light sensor could serve as an effective alternative. Light sensors [35] integrated into tablets and smartphones can be used to achieve the same "Front is Clear" condition. By programming the system to stop code execution when a certain light threshold, which is measured in lux, is exceeded. The limitations of the proximity sensor were addressed, and the desired functionality was successfully implemented.

6.1.5. Code

The apppendix to this thesis contains the complete source code for the application.

The MainActivity.java file is the primary interface of the application, where users can input commands manually or add them using predefined buttons. After entering the commands, they are transferred to the ResultActivity for execution.

The design for MainActivity is contained in activity_main.xml, which contains elements that control the application. The user interface components used in this application include a text field, a checkbox, and buttons for application control.

To be more specific, the MainActivity.java file has an EditText field that is labeled codeInput, and it is this way, users can directly enter commands. There is also a checkbox, if activated, which enables loop monitoring using the light sensor. Several buttons provide users with predefined commands, such as "forward", "left", "right" and "stop", which are inserted into the codeInput field. As a last point, a "Play" button sends the commands that were entered along with the status of the ifLoopCheckbox to ResultActivity for execution.

The ResultActivity.java file is in charge of responding to commands given by MainActivity. It regulates two rectangular areas on the screen that change color according to the command executed. Additionally, ResultActivity keeps an eye on the light sensor is to check if the "Front Is Clear" condition is met.

In the ResultActivity class, the executeCommands method ensures that the commands are handled in a prescribed order. Corresponding to each command's execution, the rectangular fields dynamically modify their colors to indicate the respective actions: left, right, front, or stop. The brief color change for the left and the right commands is followed by the longer duration of forward command color change. Loop control is initiated if either the ifLoopCheckbox is selected or a loop command is present. The loop ends when the light sensor detects that the "Front Is Clear" condition is no longer met. The sensor continuously monitors ambient light, and if the light level drops below a specified threshold, the loop terminates, and the application returns to MainActivity.

The ResultActivity layout in the activity_result.xml file includes two rectangular fields and an image. The fields change color according to the commands (left, right, forward, stop), but the image serves as a visual starting point for the activity.

Chapter 7.

Usability Testing

The usability testings purpose [36] is to analyze the RoboKarel-App in order to identify its user-friendliness and functionality with a focus on kids. The aim is to collect information about the applications intuitive design, probable problems, and how much the applications features work as expected.

Defining the usability test where children aged 10 and above are instructed to assess the clarity and usability of the app's core functions. The test studies how children use the app and measures the degree of intuitiveness of the robot's controls.

The first task requires the robot to go forward which assesses the clarity and comprehension of basic movement commands. The second task, which is involving the robot steering around an obstacle, is a test of kids skill to avoid the obstacles while using the control mechanisms of the app. The third task stops the robot using the light sensor, checking the robot's responsiveness to surrounding conditions. This action enables testing the app's sensor technology as well as the robot's capability to react to variations.

Performing such usability testing is really important to make sure that the app is used by the correct target audience and users interaction with the technology is smooth. The gained data from the tests deliver precious information to the developer to the further optimization of the app's usability. The complete usability test sheet is included in the attachments.

7.1. Subject

Fardos, aged 10 years, currently in the fourth grade, has been chosen as the person to test the RoboKarel application because of her interest in technology. Despite her age,

Fardos enjoys to use digital gadgets such as smartphones and tablets. She can quickly and easily navigate through different social networks. Youtube is the application that she uses the most on her device and she is fully informed about how to use it. Her comfort with technological difficulties makes her a good testing candidate for the RoboKarel-App. Furthermore, Fardos practises creative leasure activities, she engages in dancing, singing, and drawing.

7.2. Evaluations of the test

The usability test of RoboKarel-App was conducted on 10-year-old Fardos. She is well versed with digital devices but this was her first experience with programming applications. Hobbies mirror her creative thinking that have an affect on her programming skills, which will be a new experience for her.

At the outset of the test, a concise overview of the app's functionality was provided, as the user interface is in English. Explaining the terms like "forward" and "loop" to Fardos, who is not yet proficient in English, was necessary in order for her to pick the right buttons. The first command was to make the robot "play" and then she saw that the "Forward" button displays the command in the code window and when you click "Play", the robot starts to operate.

For the very first task that required the robot to move forward, Fardos chose the "Forward" button. It was an effective way to accomplish the task as it could be seen in the code window. The robot then started to go forward after the "Play" button was pressed and the robot was placed. She was visibly excited in the entire process of the robot following her command without any issues.

The second task proved to be more difficult. The goal was to move the robot around an object. Fardos was initially not very good at controlling the controls. To make the robot turn the way she wanted she had to push many buttons among them the ones labeled "Left" and "Right." The first time she tried, the robot was stuck and hit the object. Then help was given by the explanation of how to change the command order to get more control. After learning this, she was able to successfully move the robot around the object.

The third task which testing a light sensor was the one that caused a little bit of a problem. Fardos had to make the robot move for a while before the robot could stopped

itself. She pushed the Forward button and fixed the tablet into the specified slot. The robot started to move but when she put her hand in front of it, the robot did not stop at first. After a second trial, the light sensor test was successful, and the robot stopped. Adjusting the position of the tablet in the holder was the main task of this process, which required some tweaks.

Overall, the test was successful. Fardos displayed significant interest and enjoyment during the app testing, despite the initial difficulty in controlling the robot.

Chapter 8.

Circuit structure

At the beginning of the circuit development process, inspiration was drawn from Volt-PaperScissors setup [37]. The fundamental configuration was relatively straightforward, consisting of two transistors, two motors, two LDRs, two button-cell batteries, and copper tape to link the components and establish a continuous circuit.

Initially, the circuit was tested using materials provided by Georg-Simon-Ohm Technical University, which included an Arduino kit. The kit had componets like LDRs, transistors, and a motor. An extra motor was also obtained for testing. The circuit was first built on a so-called breadboard where it was tested for a short time to see if it worked. The circuit can be seen below, in figure 8.1.

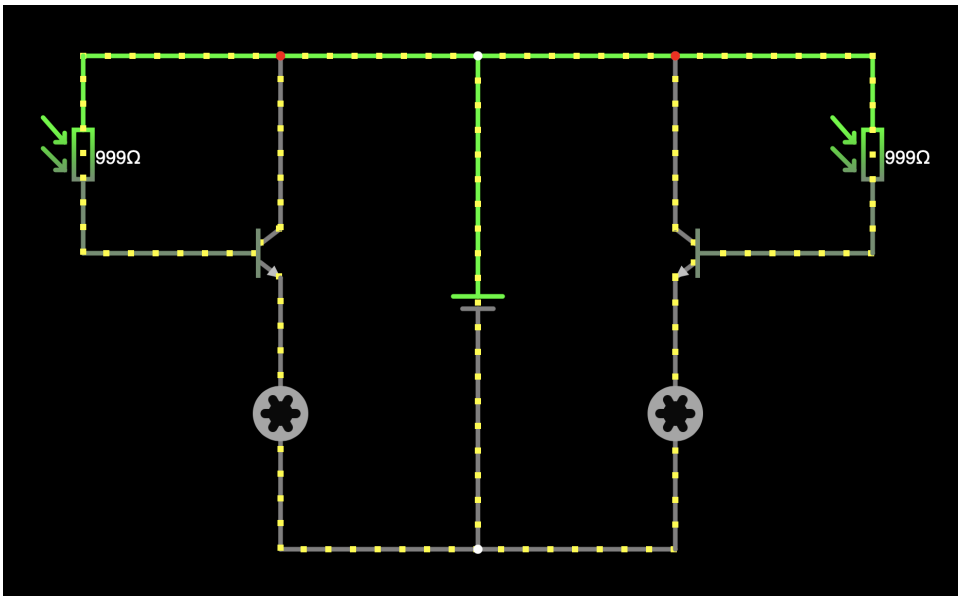


Figure 8.1.: The first circuit made virtually with falstad; Source: `https://www.falstad.com/circuit/circuitjs.html`

But, this step introduced a few mistakes. For instance, the transistor was wrongly wired because of a mistake in the proper ordering of the base, emitter, and collector. Also, the 3V button-cell batteries were not enough to drive the motor, so it had to be changed with the 9V batteries that came with the Arduino kit. In order to carry out the project more accurately, the components borrowed from the Arduino kit were finally swapped out with materials that were similar to those used in the DIY Smartphone Robot project, from VoltPaperScissors tutorial [37], and then the circuit was built again. While the circuit was correctly configured in this case, another problem occurred: the transistor generated too much heat which might endanger the safety of the kids. Consequently, the option to recreate the circuit in a virtually environment was taken.

During the virtual simulation, an oscilloscope was used to measure the current flowing through the transistors, revealing that the current was significantly higher than expected. Natural degradation of the motors resistance system when exceeded by the frictiones (e.g. contact with the ground) was the reason behind which the motors drew a higher current. This high current was also carried through the transistors that weren't meant to deal with such currents. [38]

To address this issue, a modified circuit was developed, which included a base resistor to protect the transistor from the excessive current required for motor operation, see figure 8.2.

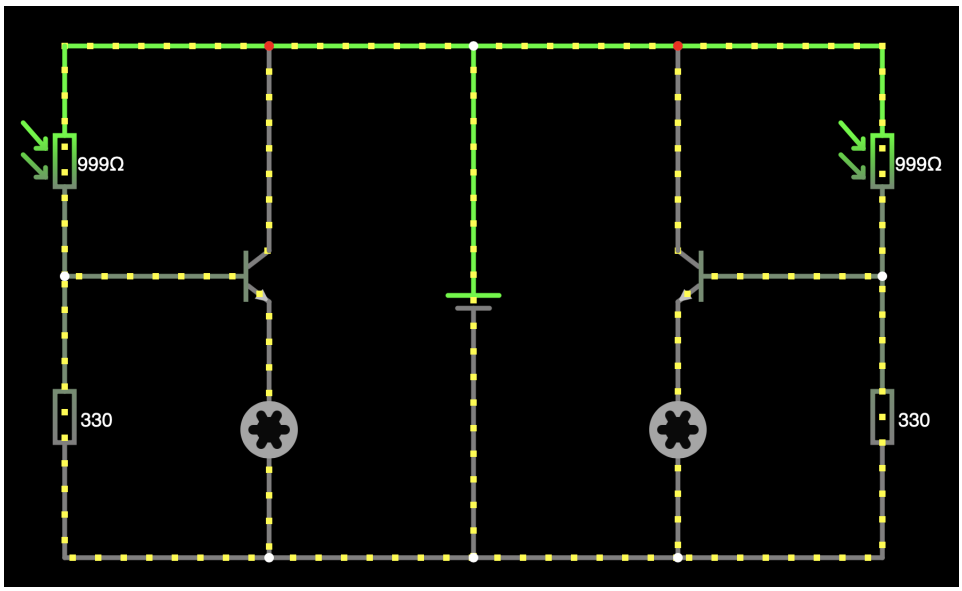


Figure 8.2.: The second circuit made virtually with falstad; Source: `https://www.falstad.com/circuit/circuitjs.html`

The circuit was modified and it was performing its function in the simulations as it was intended and the results were encouraging for its final application. Yet, the actual test revealed the fact that the motor was not supplied with sufficient current, and as a result, it stopped to run. But, the fact that the transistors were no longer having the problem of overheating, was a positive change.

8.1. Final circuit

Further research suggested using a MOSFET in place of a transistor. A new circuit was then devised, incorporating the following elements that can be seen in this figure 8.3 [39]:

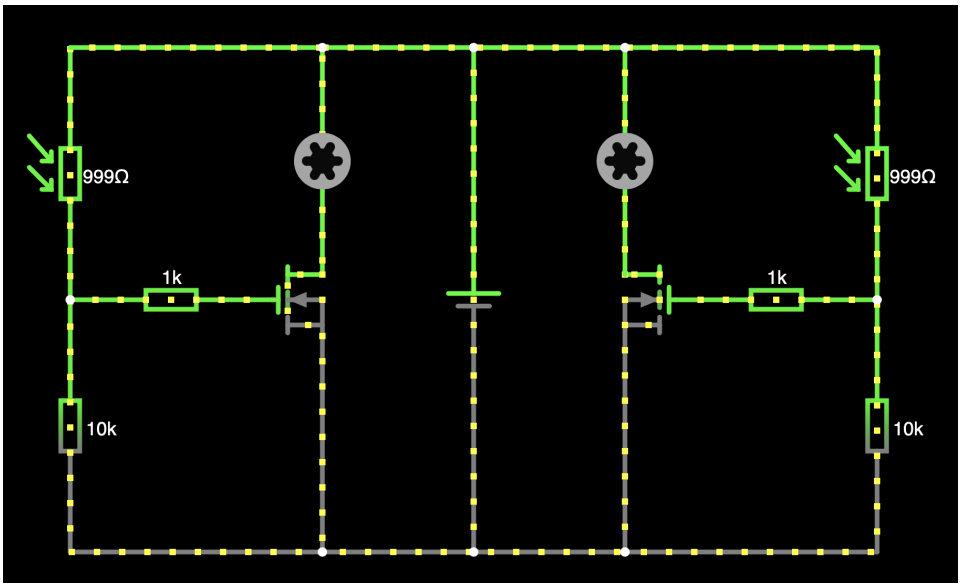


Figure 8.3.: The final circuit made virtually with falstad; Source: `https://www.falstad.com/circuit/circuitjs.html`

Solution: Switching to MOSFETs

For the avoidance of overload, the circuit was reconfigured to adopt higher effectivity MOSFETs. These elements are the most appropriate for the case of quick switching of higher currents and operate more efficiently due to their low resistance in saturation mode (fully on).

Why MOSFETs?

The major edge that the MOSFETs [40] have over the typical transistors is the fact that they can carry a higher amount of current. They are able to switch to high currents making them suitable for applications that involve motors under load. For instance, the MOSFET IRF540N can take currents of up to 33A which is far more than what the circuit needs. In addition, MOSFETs have the characteristic of having a very low resistance (i.e. $R_{DS}(\text{on})$) when fully on, which results in the voltage drops being minimal and the power losses being reduced. The other advantage of the logic-level MOSFETs is that they can fully switch on at lower gate voltages (typically 2-4V) which was also the case in this circuit with the LDRs and voltage dividers.

Limitation on MOSFET Choice

The university has only IRF520N MOSFETs in stock and so this device was the only one used in the circuit. Although the IRF520N is reasonable for most applications, it is not as good as the IRF540N which can deal with a maximum current of 33A. The IRF540N is the recommended for those who are making this circuit replica, simply due to the fact that it has a higher efficiency and better performance. [41]

Final Circuit: Components and Design

The final circuit consists of two MOSFETs, two LDRs, resistors to create a voltage divider, and a 9V battery to control two motors. The selection and designing of each component is to ensure that the current and voltage demanded of the motors is met.

The circuit is powered by a 9V battery, the two DC motors are controlled by N-channel MOSFETs (IRF540N), which adjust to the light intensity detected by the LDRs. The LDRs measures the ambient light and adapt the voltage as per the MOSFET gate. A $1\,\text{k}\Omega$ gate resistor regulates the current flow to the MOSFET gate, while the $10\,\text{k}\Omega$ voltage divider resistor, in combination with the LDRs, determines the gate voltage.

Circuit Calculations

The calculation for the resistors $R_1 = 1\,\text{k}\Omega$ and $R_2 = 10\,\text{k}\Omega$ is based on the requirement that the MOSFET must receive a sufficient voltage at the gate to switch the motor on when the light is bright enough. The resistors, together with the LDR, form a voltage divider that converts the ambient light into a control voltage for the MOSFET's gate.

To understand this in more detail, let's go through the step-by-step calculation of the resistors. The MOSFET, for example, an IRF540N, requires a gate-source voltage of at

least 4V to reliably turn on the motor. This means that the voltage divider must be configured in such a way that the voltage at the gate reaches at least 4V when the light is bright enough.

The voltage divider follows a basic formula: the output voltage V_{out} at the gate of the MOSFET depends on the resistance values of the voltage divider. The formula is:

$$V_{out} = V_{in} \times \frac{R_{LDR}}{R_{LDR} + R_2} \tag{8.1}$$

Equation 8.1: Voltage divider formula for LDR and resistor R_2.

In this equation, V_{in} represents the supply voltage of 9V provided by a battery, R_{LDR} is the resistance of the LDR, which varies depending on the brightness, and R_2 is the fixed resistor, set at 10 kΩ in this case.

Now, let's consider the circuit assuming that the LDR has a resistance of 500 Ω in bright light. The goal is to achieve an output voltage of at least 4V to ensure that the MOSFET switches. The calculation is as follows:

$$V_{out} = 9V \times \frac{500\Omega}{500\Omega + 10k\Omega} = 0.43V \tag{8.2}$$

Equation 8.2: Example of a voltage divider calculation with $R_{LDR} = 500\Omega$ and $R_2 = 10k\Omega$.

As we can see, the voltage is only 0.43V, which is far below the required 4V.

To increase the voltage, the value of R_2 needs to be adjusted. By reducing R_2 to 1 kΩ to achieve a higher voltage, the result is:

$$V_{out} = 9V \times \frac{500\Omega}{500\Omega + 1k\Omega} = 3V \tag{8.3}$$

Equation 8.3: Example of a voltage divider calculation with $R_{LDR} = 500\Omega$ and $R_2 = 1k\Omega$.

The voltage at the gate is now 3V, which is close enough to the threshold to switch the MOSFET when the light is bright enough. With even more light, the resistance of the LDR could decrease further, increasing the voltage at the gate.

In the final circuit, the resistors were selected as follows: $R_1 = 1\,\text{k}\Omega$ was chosen to limit the current flow to the MOSFET's gate while providing enough voltage to trigger it. $R_2 = 10\,\text{k}\Omega$ ensures that, in lower light, the voltage at the gate drops so the MOSFET turns off again. This combination of resistors effectively controls the motor based on the light conditions.

Test Results and Functionality

Upon the construction of the MOSFETs circuit, the motors had a functioning power in the case of enough light detection. The previous calculations proved that the MOSFETs are completely switched on allowing the motors to draw enough power to operate normally.

Tests were done using a 9V battery, and the circuit worked as expected. Yet, the battery's lifespan was short under conditions of long-term motor load. Besides, it was noticed that the IRF520N MOSFET stayed cool and only a little voltage drop was registered across the motor.

Conclusion and Optimization Suggestions

Effectively, the differencing of the MOSFET technology and the original circuit using transistors showed the latter to be more efficient. MOSFET solved the problems of overheating and excessive voltage drop that were with transistors, which allowed the motors to run at full power and the light-control system to work correctly.

For future improvements, the circuit could be linked to more stable power supply such as Li-ion batteries, which can last longer. If a PWM controller, which would allow for more precise control of motor speed, could be added that will be another possible improvement, which would also help the circuit become more efficient.

Improving Circuit Assembly

Initially, the circuit assembly design was the use of copper tape, which was a very accessible material for younger users.

However, the conductivity of the copper tape proved inadequate, prompting the search for an alternative solution. The next phase involved soldering all connections, but the

result was neither aesthetically pleasing nor functional. Furthermore, a short circuit caused by a defective resistor in either the first or second assembly required the circuit to be disassembled and rebuilt multiple times.

It became evident that this approach was impractical for children due to the fact that it involved considerable adult intervention and failed to meet the visual standards. In general, circuitry is first tested on a prototype using a breadboard to come up with the idea of constructing the entire circuit on the breadboard directly. This technique has benefits such as simplifying the component replacement, providing a more organized and visually attractive layout, and making it easier to incorporate necessary changes. In the end, we arrived at that decision, which was the most effective and the most visually appealing one. The final version can be seen in figure 8.4.

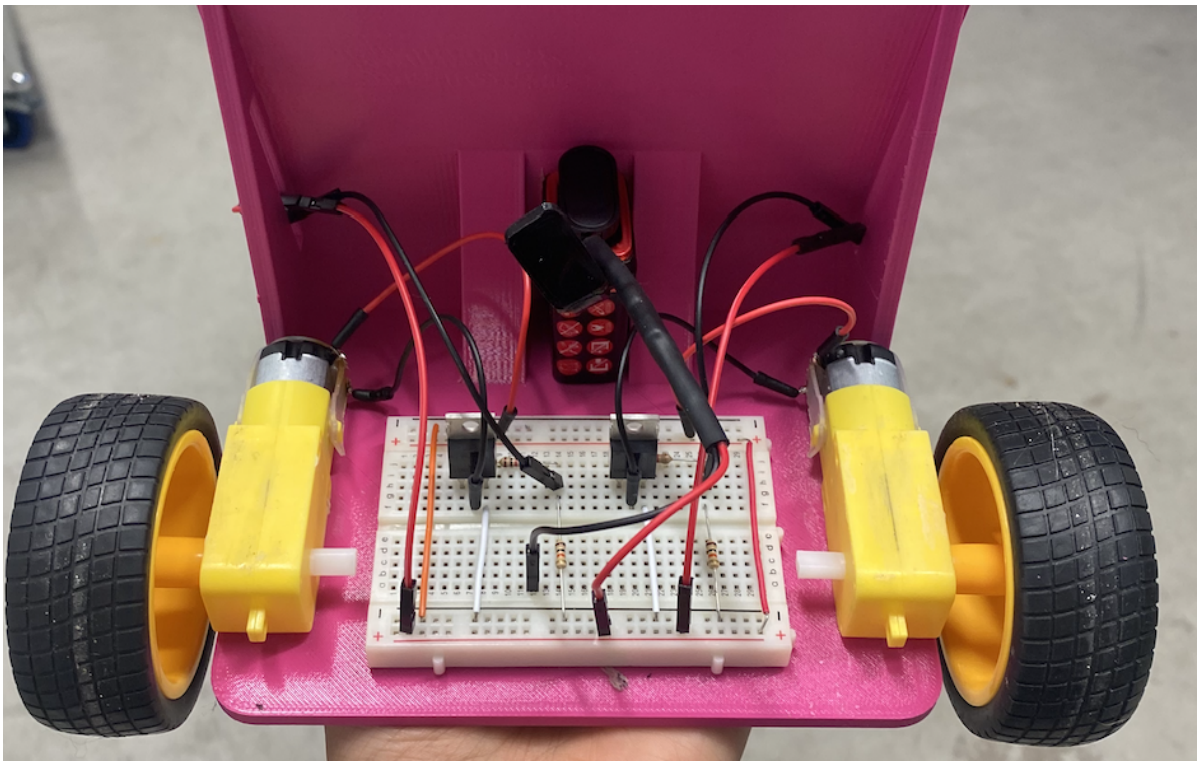
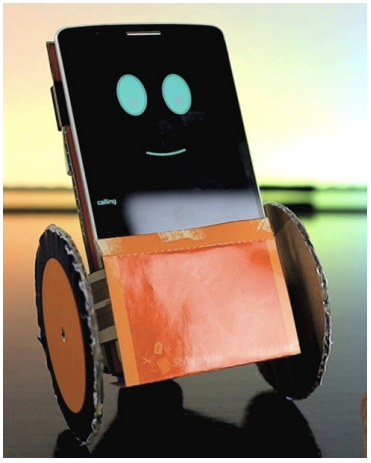


Figure 8.4.: The final circuit assembly using a breadboard; Source: Author's own image

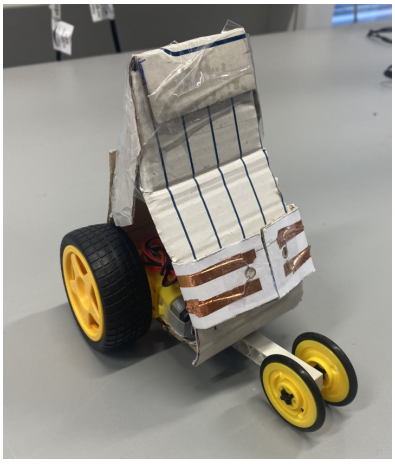
Chapter 9.

3D-Modell

The 3D-Modell was developed in a structured manner. At the first stage, a cardboard model was constructed using templates provided by VoltPaperScissors [37], see figure 9.1a. The image was fixed on a durable cardboard sheet, then it was bent and attached at the designated points to get the desired figure. While the model was successfully assembled, it lacked the stability necessary to support a smartphone. To address this, an additional wheel was incorporated to enhance stability. Illustration can be found under figure 9.1b.



(a) Image of the robot created by VoltPaperScissors; Source: `https://www.VoltPaperScissors.com/diy-smartphone-robot`



(b) Image of the first model made with cardboard and an extra wheel; Source: Author's own image

Figure 9.1.: Comparison of two models: VoltPaperScissors robot and the author's cardboard model

Since the project was designed for tablets rather than smartphones, the cardboard model proved insufficiently stable, and maintaining the tablets balance was challenging. This

resulted in the custom 3D-printed tablet holder. A search on platforms such as Thingiverse yielded a suitable tablet holder model [42], which was modified to meet the specific requirements. Among the problems encountered, one was about using two motors to provide the movement. To boost the stability of the tablet, two additional wheels were included on the four motor-driven wheels. The wheels were taken from Thingiverse [43] and modified to fit the prototype.

After consulting with Mr. Baer it was obvious that the original idea of interconnected wheels was not practical. The two wheels were driven by the same axle, which applied the same torque, thereby forcing the wheels to have the same difficulty in turns. In a left turn, the outer wheel should have a higher angular velocity than the inner wheel, whereas, the rigid axle prevented this from happening. Mr. Baer, with his 3D modeling expertise, also pointed out that the holder does not have enough stability to hold the tablets weight. Therefore, a better option was introduced and a caster wheel is being used instead. This meant that no more wheels, axles, or screws had to be printed. The hole was was kept in the holder so the caster wheel can be attached through it using a nut.

Originally the plan was to 3D print the caster wheel, but it was finally decided to repurpose an existing caster wheel provided by the university. The second prototype was printed successfully, though there was a minor issue with the upper part of the holder not printing correctly. However, the prototype was still able to fulfill its purpose during the early stages of development.

To improve the design more modifications were made. A cardboard model initially used to mount the LDRs was considered aesthetically unsatisfactory, prompting the development of a more refined alternative. Additionally, the first prototype lacked rounded edges at the front, leading to an imbalance with the rear. In the third prototype, the edges were rounded, the holders length was reduced, and the width was increased to better accommodate the tablet. A compartment for the battery was also incorporated into the rear of the holder, as the overall stability had been significantly improved. For visual representation look at figure 9.2.

Figure 9.2.: The final 3D-Model printed an assembled together; Source: Author's own image

Chapter 10.

Future Outlook

Over the years, there will be plenty of options for the project to expand with new features. The basic app, which is developed via the MIT App Inventor platform, enables to connect with various networks. Imagine, for instance, the use of artificial intelligence or the provision of a connection to an Arduino which can be easily implemented. Such a development would enable a child, who starts with the creation of an app in this project, to get involved in further experimentation and will so in the development of more complex applications. This would enhance their knowledge about programming and robotics.

RoboKarel-App has the potential to be developed on a larger scale. More integration of the tablets sensors could give birth to new features and fun elements - for instance, the loop function. For example, one could imagine a robot looking for a hidden magnet, called "Beeper," which is placed in front of the robot. The robot can carry out various movements according to the magnetic field values that were measured with the tablet using a built-in magnetic field sensor. It would stop once a specific threshold is reached, signaling the location of the "Beeper" [44]. This kind of interactive activity not only enhances the level of the students' engagement but also the knowledge of the applications of the sensors and environmental data analysis.

The 3D model, while currently free from significant deficiencies, offers potential for future optimization. If the integration of additional components becomes necessary, the front and rear plates of the model could be enlarged to provide extra space or improved stability.

The circuit is flexible primarily for the reason of a breadboard being used that allows an easy modification and expansion when required. Let's say for instance adding an Arduino board to introduce new features. Besides, more sensors can be added and modern versions can be upgraded, of the existing components. The modular structure of the circuit is a

guarantee of compatibility for the technology that will come in the future with all its requirements.

These potential expansions increase the project's capacity for growth, making it a more flexible and versatile learning tool. This enables both beginners and advanced users to explore robotics and programming in novel and engaging ways.

Chapter 11.

Conclusion

The objective of this thesis was to develop a robot that is both beginner-friendly and cost-effective, making it suitable for beginners and young learners in the fields of robotics and programming. Alternatively, by combining simple control mechanisms, an easy-to-use software application, and modular hardware components, it was possible to create a system that effectively conveys fundamental principles of robotics in an engaging and essential way. During the development phase, different technological methods were considered and applied to confirm the functionality and flexibility of the project.

At the heart of the project were three key applications: the Remote-App and Robot-App, both developed using MIT App Inventor, and the RoboKarel-App, created with Android Studio. These applications allow for seamless integration between software and hardware, enabling users to intuitively control the robot while learning fundamental programming concepts. MIT App Inventor facilitated the rapid and straightforward development of the apps, while Android Studio provided greater flexibility and complexity for the RoboKarel-App, designed specifically for younger users.

The modularity of the project is an advantage that gives the opportunity to constantly extend and upgrade both the hardware and software. The frame made of 3D printing and the circuit based on breadboard constitute non-mechanical foundations that can easily be customized software-wise to fit the needs of the robot. The flexible structure of a breadboard refers to this feature to easily enlarge the circuit by connecting new modules like sensors or microcontrollers. Plus, it serves as a flexible environment, where learners can try to add new functions themselves.

One of the key elements of this project is the use of light sensors for controlling the robot's movements. The robot receives feedback via the sensors enabling it to react to the surrounding and hence, the learners get a real-time response to the commands they

have programmed. The addition of the loop feature in the RoboKarel-App gives the programm a certain complexity that is analogous to the application of logical programming structures in real life [45]. It adds to the learners' knowledge of programming concepts and at the same time, it is the introduction of more complicated mechanisms like loops and conditions.

The prototypes developed in this thesis demonstrate the robot's potential as an effective educational tool for children and adolescents. Utilizing tablets as the control interface gives a common, friendly, and cost-effective way to implement the solution, thus, increasing the project's accessibility. Further, the use of 3D-printed components has made the product even more cost-efficient and allows for customization and future upgrades.

Looking forward, several options are lying ahead for the extension of both the hardware and software parts of this project. The incorporation of artificial intelligence would be a valuable improvement since it would give the robot the capability to work without the guidance of a human being and carry out tasks that are more complicated. The magnetic field or infrared sensors could be added to make it easier for the robot to interact with others. From the software side, features like voice, and gesture control could be added for the user to have a better experience.

All in all, this project is a solid foundation for beginners for gaining or improving their knowledge in robotics and programming. By employing straightforward tools such as MIT App Inventor and Android Studio, a working system was composed that allows students to participate in robotics in a creative and self-directed way. The system's flexibility and scalability add its value as an educational tool with the possibility of future innovation and adjustment. This thesis demonstrates that simple technologies and modular concepts can be leveraged to develop cost-effective yet powerful educational tools, facilitating an easy introduction to robotics while leaving room for advanced experimentation. Everything that was made in this project will be available as open-source. [46]

Bibliography

[1] Stanford University Standing Committee. *Artificial Intelligence and Life in 2030.* Accessed: August 22, 2024. Stanford University, Published 2016. URL: https://tinyurl.com/2ldtcluo.

[2] Ben Nancholas. *Modern Robotics: How Robots Are Transforming Industries and Everyday Life.* Accessed: September 12, 2024. Published 2023. URL: https://online.keele.ac.uk/modern-robotics-how-robots-are-transforming-industries-and-everyday-life/.

[3] Goethe University Frankfurt. "New curriculum improves students' understanding of electric circuits in schools". In: *ScienceDaily* (Published 2020). Accessed: September 29, 2024. URL: https://www.sciencedaily.com/releases/2020/12/201218112500.htm.

[4] Myint Swe Khine Othman Abu Khurma Nagla Ali. "Exploring the impact of 3D printing integration on STEM attitudes in elementary schools". In: (Published 2023). Accessed: August 20, 2024. URL: https://files.eric.ed.gov/fulltext/EJ1406908.pdf.

[5] Alessandra Sciutti. "Educational Robotics and Robot Creativity: An Interdisciplinary Dialogue". In: *Frontiers in Robotics and AI* (Published 2021). Accessed: September 18, 2024. URL: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2021.662030/full.

[6] Aubrey Joyce Colter. "Evaluating and improving the Usability of MIT App Inventor". Accessed: September 4, 2024. PhD thesis. Massachusetts Institute of Technology, Published 2016. URL: https://dspace.mit.edu/handle/1721.1/106027.

[7] Toptal. "The Power of Figma as a Design Tool". In: (Published 2023). Accessed: September 18, 2024. URL: https://www.toptal.com/designers/ui/figma-design-tool.

[8] Android Developers. *Download Android Studio & App Tools.* Accessed: September 18, 2024. Published 2024. URL: https://developer.android.com/studio.

[9] MIT App Inventor Team. *MIT App Inventor*. Accessed: September 18, 2024. Published 2024. URL: https://appinventor.mit.edu/.

[10] Evan W. Patton; Mike Tissenbaum; Farzeen Harunani. "MIT App Inventor: Objectives, Design, and Development". In: (May 2019). Accessed: September 18, 2024. URL: https://www.researchgate.net/publication/337361811_MIT_App_Inventor_Objectives_Design_and_Development#:~:text=MIT%20App%20Inventor%20is%20an,language%20to%20program%20application%20behavior..

[11] Google Developers. *Blockly: A Visual Programming Tool*. Accessed: September 26, 2024. Published n.d. URL: https://developers.google.com/blockly?hl=de.

[12] Blender Foundation. *Blender: Free and Open 3D Creation Software*. Accessed: September 20, 2024. Published 2024. URL: https://www.blender.org/.

[13] UltiMaker. *UltiMaker Cura: Free, Easy-to-Use 3D Printing Software*. Accessed: September 20, 2024. Published 2024. URL: https://ultimaker.com/software/ultimaker-cura.

[14] Jotform. *App Inventor vs Thunkable*. Accessed: September 1, 2024. Published 2024. URL: https://www.jotform.com/blog/thunkable-vs-app-inventor/.

[15] Kodular. *Kodular Creator: Build Android Apps without Coding*. Accessed: September 1, 2024. Published 2024. URL: https://www.kodular.io/creator/.

[16] Modulo. *Scratch Coding for Kids: A Review by an Experienced Educator*. Accessed: September 1, 2024. Published 2024. URL: https://www.modulo.app/scratch-coding-review/.

[17] The EduTech Post. *Coding for Kids with Tynker Game-based Learning*. Accessed: September 1, 2024. Published 2024. URL: https://www.edutechpost.com/tynker-game-based-learning/.

[18] Code.org. *Learn Computer Science | Code.org*. Accessed: September 13, 2024. Published 2024. URL: https://code.org/.

[19] Michael Scheuerl. *DIY Smartphone Robot - The Award-Winning STEM Challenge*. Accessed: Juli 2, 2024. Published 2024. URL: https://www.voltpaperscissors.com.

[20] Cevinius. *MobBob | Smart Phone Controlled Robot*. Accessed: Juli 15, 2024. Published 2024. URL: https://www.cevinius.com.

[21] Intel Intelligent Systems Lab. *OpenBot: Smartphone Powered Robot*. Accessed: Juli 16, 2024. Published 2024. URL: https://www.openbot.org.

[22] Instructables. *DIY Phone Controlled FPV Rover (Fast & Agile)*. Accessed: Juli 16, 2024. Published 2024. URL: https://www.instructables.com/DIY-Phone-Controlled-FPV-Rover/.

[23] Richard E. Pattis. *Karel the Robot: A Gentle Introduction to the Art of Programming*. A foundational text for introducing programming concepts. New York: Wiley, Published 1981.

[24] s. r. o. CloudMakers. *Karel the Robot on the App Store*. Accessed: August 4, 2024. Published 2024. URL: https://apps.apple.com/us/app/karel-the-robot/id923005645.

[25] Andreas Schilling. *Robot Karel 1981 for Android and iOS*. Accessed: August 4, 2024. Published 2024. URL: https://karel1981.com.

[26] Apple-Inc. *Swift Playgrounds - Apple Developer*. Accessed: August 17, 2024. Published 2024. URL: https://developer.apple.com/swift-playgrounds/.

[27] MIT App Inventor. *MIT App Inventor: Create Apps*. Accessed: Juli 2, 2024. Published 2024. URL: http://appinventor.mit.edu/.

[28] MIT App Inventor. *Designer and Blocks Editor in MIT App Inventor*. Accessed: October 3, 2024. Published n.d. URL: https://appinventor.mit.edu/explore/designer-blocks.

[29] MIT App Inventor. *Setup Device with MIT App Inventor over Wifi*. Accessed: September 5, 2024. Published n.d. URL: https://appinventor.mit.edu/explore/ai2/setup-device-wifi.html.

[30] ewpatton. *Welcome to the MIT App Inventor Community*. Accessed: August 22, 2024. Published n.d. URL: https://community.appinventor.mit.edu/t/welcome-to-the-mit-app-inventor-community/7.

[31] TIMAI2. *Resolved: Build Servers Are Having Problems*. Accessed: September 12, 2024. Published n.d. URL: https://community.appinventor.mit.edu/t/resolved-build-servers-are-having-problems/70114.

[32] Eric Roberts. *Karel the Robot Learns Java*. Accessed: September 5, 2024. Published n.d. URL: https://cs.stanford.edu/people/eroberts/karel-the-robot-learns-java.pdf.

[33] Android Developers. *Positionssensoren in Android*. Accessed: September 12, 2024. Published n.d. URL: https://developer.android.com/develop/sensors-and-location/sensors/sensors_position?hl=de.

[34] XDA Forums. *There is an LED on the Nexus 9*. Accessed: September 18, 2024. Published n.d. URL: https://xdaforums.com/t/there-is-an-led-on-the-nexus-9.2929379/.

[35] Android Developers. *Umgebungssensoren in Android*. Accessed: September 12, 2024. Published n.d. URL: https://developer.android.com/develop/sensors-and-location/sensors/sensors_environment?hl=de.

[36] Kate Moran. *Usability Testing 101*. Accessed: October 3, 2024. Published 2019. URL: https://www.nngroup.com/articles/usability-testing-101/.

[37] Michael Scheuerl. *Easy DIY Smartphone Robot*. Accessed: September 26, 2024. Published n.d. URL: https://www.voltpaperscissors.com/easyrobot.

[38] Paul Falstad. *CircuitJS - Electronic Circuit Simulator*. Accessed: September 29, 2024. Published 2024. URL: https://www.falstad.com/circuit/circuitjs.html.

[39] Luis Llamas. *Controlling Large Loads with Arduino and MOSFET Transistor*. Accessed: September 28, 2024. Published 2024. URL: https://www.luisllamas.es/en/arduino-mosfet-transistor/.

[40] Heinen Elektronik. *MOSFET - Schnelles Schalten bei geringen Spannungen*. Accessed: September 29, 2024. Published n.d. URL: https://tinyurl.com/yq5jt63t.

[41] EE-Diary. *IRF540N: The Ultimate Guide for Arduino Users*. Accessed: September 28, 2024. Published 2024. URL: https://www.ee-diary.com/2022/06/irf540n-ultimate-guide-arduino-users.html.

[42] Printables User. *Tablet Holder*. Accessed: September 29, 2024. Published 2024. URL: https://www.printables.com/model/402907-tablet-holder/files?lang=de.

[43] Thingiverse User: EnforcerZhukov. *Tablet Holder for 7" and 8" tablets*. Accessed: September 29, 2024. Published 2018. URL: https://www.thingiverse.com/thing:3072345.

[44] Yongping Pan. "Magnetic-Field-Inspired Navigation for Robots in Complex and Unknown Environments". In: *Frontiers in Robotics and AI* (Published 2022). Accessed: September 18, 2024. URL: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/frobt.2022.834177/full.

[45] Shubham Gautam. *Time Complexity Analysis of Loop in Programming*. Accessed: October 3, 2024. Published n.d. URL: https://www.enjoyalgorithms.com/blog/time-complexity-analysis-of-loop-in-programming.

[46] Josefina Fritz. *RoboKarel App Repository*. Accessed: October 6, 2024. Published 2024. URL: `https://github.com/Josefinafr/RoboKarel/tree/main`.

Appendices

Appendix Content

List of Listings

Supplemental Information

A. Instructions for Using MIT App Inventor to Program the Remote-App

For those who have never built anything with MIT App Inventor before, it is recommended to start here: `https://appinventor.mit.edu/explore/sites/all/files/hourofcode/AppInventorTutorials.pdf`

If you want to connect two devices using Bluetooth in MIT App Inventor, you will need to use both the Designer and the Blocks Editor. Follow these simple steps to create your Bluetooth connection and send commands!

A.1. Adding Components in the Designer

In the Designer section of MIT App Inventor, you can add both visible and non-visible components to your app. Here is what you need to include:

- **Buttons**: These will control movement. Add four buttons:
  - Forward
  - Left
  - Right
  - Stop

  Additionally, add a **Connect** button, which will establish the Bluetooth connection between the devices.
- **Label**: A label will display whether the devices are connected or not. In this example, the label is called `connectAbfrage` and will show the connection status.

- **BluetoothServer**: This is a non-visible component that enables the app to send and receive data via Bluetooth. You can find it under the *Connectivity* section in the Designer.

- **TinyDB**: Another non-visible component. TinyDB stores important data, such as whether Bluetooth is allowed to connect.

A.2. Setting Up the Apps Functionality with Blocks

Next, switch to the **Blocks Editor** to program the logic of your app. Here is how each block works:

A.2.1. App Initialization and Permissions

- **Screen1.Initialize**: When the app starts, this block checks if Bluetooth is turned on. If it is not, the app prompts the user to enable Bluetooth. It also asks for Bluetooth permissions (essential for newer Android versions).

- **Permission Handling**: The app will request Bluetooth scanning and connection permissions. If the user grants them, the app stores this information in TinyDB, so it will not ask again the next time the app is opened.

A.2.2. Establishing the Connection

- **Connect.Click**: When the user presses the *Connect* button, the app instructs the Bluetooth server to start accepting connections. The `connectAbfrage` label will update to *Ready to receive connections*.

- **BluetoothServer1.ConnectionAccepted**: Once a connection is accepted, the label changes to *Connection accepted*.

A.2.3. Sending Commands

Each control button (*Forward*, *Left*, *Right*, *Stop*) sends a different command when pressed:

- *Forward* sends “1”
- *Left* sends “2”
- *Right* sends “3”
- *Stop* sends “4”

These commands are transmitted via Bluetooth and received by the other device to control motors or other connected components.

A.3. Connecting the Blocks

The blocks are linked together in such a way that the app first checks for Bluetooth availability, then waits for the user to initiate a connection. Once connected, the control buttons send commands through Bluetooth, allowing the other device to respond accordingly.

B. Instructions for Using MIT App Inventor to Program the Robot-App

If you want to connect two devices through Bluetooth using MIT App Inventor, you will need to add important components in both the Designer and the Blocks Editor. Follow these simple steps to build an app that lets you connect and interact with Bluetooth devices!

B.1. Adding Components in the Designer

First, open the **Designer** section of MIT App Inventor and add the following components:

- **ListView**: This will show a list of available Bluetooth devices. The user can pick a device to connect to from this list.
- **Labels**: Add two labels, called `Label1` and `Label2`, to display the selected device name and status messages about the connection process.
- **TextBoxes**: Add two textboxes to display the data received from the connected device. These textboxes will also change colors based on the data received.
- **BluetoothClient**: A non-visible component that handles the Bluetooth connection. It lets the app send and receive data between devices. You can find it in the "Connectivity" section.
- **Clock**: Another non-visible component that runs on a timer, used to check for new data coming in from the connected Bluetooth device.

B.2. Setting Up the App's Functionality with Blocks

Now, switch to the **Blocks Editor** to program how the app works. Here is what each block does:

B.2.1. App Initialization and Permissions

- **Screen1.Initialize**: When the app starts, it checks if your device is running Android version 31 or higher. If it is, the app asks for Bluetooth permission to scan for devices. If permission is granted, the available Bluetooth devices will be shown in the ListView.

- **TinyDB**: The app remembers if permission has already been granted by storing this information in TinyDB, so the user will not be asked again next time they open the app.

B.2.2. Connecting to a Device

- **Connect.Click**: When the user taps the `Connect` button, the app scans for Bluetooth devices and displays them in the ListView.

- **ListView.AfterPicking**: When a device is picked from the list, the app tries to connect to it. The connection status is updated (shown in the label), and the `Clock` component is activated, allowing the app to start checking for new data.

B.2.3. Receiving and Handling Data

- **Clock1.Timer**: The timer checks at regular intervals to see if any new data has come in from the connected device. Based on the data received, the background colors of the two TextBoxes change:

  - If the received data is "1", **both TextBoxes turn white**.

  - If the received data is "2", **the left TextBox turns white** and **the right TextBox turns black**.

  - If the received data is "3", **the right TextBox turns white** and **the left TextBox turns black**.

  - If the received data is "4", **both TextBoxes turn black**.

B.2.4. Disconnecting

- **Disconnect.Click**: When the user clicks the `Disconnect` button, the Bluetooth connection is cut off, and the ListView is hidden again to show that the connection is no longer active.

B.3. How the Blocks Are Connected

The blocks work together to ensure everything functions smoothly:

- The app asks for permission to use Bluetooth before doing anything else.
- The ListView shows available devices, and once a device is selected, the app connects to it.
- The `Clock` keeps checking for new data, and when new data arrives, the TextBoxes change color based on the received values.

With this setup, your app will be able to connect to Bluetooth devices, send and receive data, and display the data in real time with color-coded feedback!

C. Instructions for Using the RoboKarel-App

The RoboKarel app helps you learn how to program a robot. You can type in code or press buttons to create commands, and the robot will follow your instructions once you start it. You can also use a loop function to make the robot repeat actions as long as the path is clear.

How to Use the App

1. Programming Movements

You can type commands into the text field or use the buttons to create the robots movements:

- Use **Forward** to make the robot move forward.
- Use **Left** and **Right** to turn the robot.
- Use **Stop** to make the robot stop.

Combine these commands in any order to create a program for your robot.

2. Using the Loop Function

You can activate a loop to make the robot repeat commands until something blocks its path. The loop can be activated in three ways:

1. **Checkbox**: Check the box labeled *Loop*.
2. **Loop Button**: Press the Loop button on the screen.
3. **Text Field**: Type the loop command directly into the text field.

3. Running the Program

Once your program is ready, press the **green Play button** to start the robot. After pressing Play, you will have **10 seconds** to place the tablet into the robots holder. The robot will then follow the commands you have programmed.

Tips

- Make sure to place the tablet into the holder within 10 seconds after pressing the Play button.
- Try different combinations of commands and loops to explore how the robot behaves.

D. Usability Testing Instructions for the RoboKarel-App

1. **Let the robot move forward:**
   Press the *Forward* button in the app so you can see it in the text field. Afterwards, press the **green Play button** and place the tablet into the robot's holder. Did the robot move?
2. **The robot should move around an object:**
   Place a small obstacle (such as a bottle or toy) in front of the robot. Try to steer the robot around the obstacle. Did it work?
3. **Let the robot drive until you stop it with your hand:**
   Try the *Loop* button with a movement of your choice. After pressing the **Play** button and putting the tablet in place, put your hand in front of the robot. Does it stop when your hand is in front of it? (This tests the light sensor).

Code

A. RoboKarel-App Code

Listing 1: MainActivity.java

```
1  package com.example.robokarel;
2
3  import android.content.Intent;
4  import android.os.Bundle;
5  import android.widget.Button;
6  import android.widget.CheckBox;
7  import android.widget.EditText;
8  import androidx.appcompat.app.AppCompatActivity;
9
10 /**
11  * MainActivity
12  *
13  * The MainActivity.java file is the primary interface of the
        application,
14  * where users can input commands manually or add them using
        predefined buttons
15  *
16  *
17  * @author Josefina Fritz
18  */
19
20 public class MainActivity extends AppCompatActivity {
21
22     private EditText codeInput;
23     private CheckBox ifLoopCheckbox;
24     private Button forwardButton;
```

```
25      private Button leftButton;
26      private Button rightButton;
27      private Button stopButton;
28      private Button playButton;
29      private Button loopButton;
30
31      @Override
32      protected void onCreate(Bundle savedInstanceState) {
33          super.onCreate(savedInstanceState);
34          setContentView(R.layout.activity_main);
35
36          codeInput = findViewById(R.id.codeInput);
37          ifLoopCheckbox = findViewById(R.id.ifLoopCheckbox);
38          forwardButton = findViewById(R.id.forwardButton);
39          leftButton = findViewById(R.id.leftButton);
40          rightButton = findViewById(R.id.rightButton);
41          stopButton = findViewById(R.id.stopButton);
42          playButton = findViewById(R.id.playButton);
43          loopButton = findViewById(R.id.loopButton);
44
45
46          forwardButton.setOnClickListener(v ->
                codeInput.append("forward\n"));
47          leftButton.setOnClickListener(v ->
                codeInput.append("left\n"));
48          rightButton.setOnClickListener(v ->
                codeInput.append("right\n"));
49          stopButton.setOnClickListener(v ->
                codeInput.append("stop\n"));
50
51
52          loopButton.setOnClickListener(v -> {
53              codeInput.append("loop\n");
54          });
55
56          playButton.setOnClickListener(v -> {
```

```
57              Intent intent = new Intent(MainActivity.this,
                    ResultActivity.class);
58              intent.putExtra("code",
                    codeInput.getText().toString());
59              intent.putExtra("ifLoop",
                    ifLoopCheckbox.isChecked());
60              startActivity(intent);
61          });
62      }
63  }
```

Listing 2: ResultActivity.java

```
1  package com.example.robokarel;
2
3  import android.graphics.Color;
4  import android.hardware.Sensor;
5  import android.hardware.SensorEvent;
6  import android.hardware.SensorEventListener;
7  import android.hardware.SensorManager;
8  import android.os.Bundle;
9  import android.os.Handler;
10 import android.os.Looper;
11 import android.view.View;
12 import android.widget.ImageView;
13 import android.widget.Toast;
14 import androidx.appcompat.app.AppCompatActivity;
15
16 /**
17  * ResultActivity
18  *
19  * The ResultActivity.java file is in charge of responding to
        commands given by MainActivity
20  *
21  * @author Josefina Fritz
22  */
23
24 public class ResultActivity extends AppCompatActivity {
25
26     private static final int INITIAL_DELAY_MS = 4000;
27     private static final int COMMAND_DELAY_FORWARD_MS = 4000;
28     private static final int COMMAND_DELAY_LEFT_RIGHT_MS =
           1000;
29     private static final int RETURN_DELAY_MS = 100;
30     private static final float LIGHT_THRESHOLD = 10;
31
32     private View leftField;
33     private View rightField;
```

```
34      private ImageView faceImage;
35      private SensorManager sensorManager;
36      private Sensor lightSensor;
37      private boolean isFrontClear = true;
38      private Handler handler = new
           Handler(Looper.getMainLooper());
39      private Runnable commandRunnable;
40
41      @Override
42      protected void onCreate(Bundle savedInstanceState) {
43          super.onCreate(savedInstanceState);
44          setContentView(R.layout.activity_result);
45
46          leftField = findViewById(R.id.leftField);
47          rightField = findViewById(R.id.rightField);
48          faceImage = findViewById(R.id.faceImage);
49
50          faceImage.setImageResource(R.drawable.face);
51
52          sensorManager = (SensorManager)
              getSystemService(SENSOR_SERVICE);
53          lightSensor =
              sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
54
55          if (lightSensor != null) {
56           sensorManager.registerListener(lightSensorListener,
                lightSensor, SensorManager.SENSOR_DELAY_NORMAL);
57          } else {
58           Toast.makeText(this, "Lichtsensor␣nicht␣verfuegbar",
                Toast.LENGTH_SHORT).show();
59          }
60
61          String code = getIntent().getStringExtra("code");
62          boolean ifLoop = getIntent().getBooleanExtra("ifLoop",
              false);
63
```

```
64          if (code != null) {
65              handler.postDelayed(() ->
                    executeCommands(code.split("\n"), ifLoop),
                    INITIAL_DELAY_MS);
66          } else {
67              Toast.makeText(this, "No code provided",
                    Toast.LENGTH_SHORT).show();
68          }
69      }
70
71      private final SensorEventListener lightSensorListener =
            new SensorEventListener() {
72          @Override
73          public void onSensorChanged(SensorEvent event) {
74              if (event.sensor.getType() == Sensor.TYPE_LIGHT) {
75                  float lightLevel = event.values[0];
76                  isFrontClear = lightLevel > LIGHT_THRESHOLD;
77                  if (!isFrontClear) {
78                      return;
79                  }
80              }
81          }
82
83          @Override
84          public void onAccuracyChanged(Sensor sensor, int
                accuracy) {
85
86          }
87      };
88
89      private void executeCommands(String[] commands, boolean
            ifLoop) {
90          commandRunnable = new Runnable() {
91              int index = 0;
92              boolean isLooping = ifLoop;
93
```

```
94              @Override
95              public void run() {
96                  if (isLooping && !isFrontClear) {
97                      returnToCodeScreenImmediately();
98                      return;
99                  }
100
101                 if (index < commands.length) {
102                     String command = commands[index].trim();
103
104                     if (command.equals("loop")) {
105                         isLooping = true;
106                         index++;
107                         handler.post(this);
108                         return;
109                     }
110
111                     if (!command.isEmpty()) {
112                         executeCommand(command);
113                     }
114
115
116                     int delay = COMMAND_DELAY_FORWARD_MS;
117                     if (command.equals("left") ||
                            command.equals("right")) {
118                         delay = COMMAND_DELAY_LEFT_RIGHT_MS;
119                     }
120
121                     index++;
122                     handler.postDelayed(this, delay);
123                 } else if (isLooping && isFrontClear) {
124                     index = 0;
125                     handler.postDelayed(this,
                            COMMAND_DELAY_FORWARD_MS);
126                 } else {
127                     returnToCodeScreen();
```

```
128                 }
129             }
130         };
131         handler.post(commandRunnable);
132     }
133
134     private void executeCommand(String command) {
135         switch (command) {
136             case "forward":
137                 leftField.setBackgroundColor(Color.WHITE);
138                 rightField.setBackgroundColor(Color.WHITE);
139                 break;
140             case "left":
141                 leftField.setBackgroundColor(Color.WHITE);
142                 rightField.setBackgroundColor(Color.BLACK);
143                 break;
144             case "right":
145                 leftField.setBackgroundColor(Color.BLACK);
146                 rightField.setBackgroundColor(Color.WHITE);
147                 break;
148             case "stop":
149                 leftField.setBackgroundColor(Color.BLACK);
150                 rightField.setBackgroundColor(Color.BLACK);
151                 break;
152         }
153     }
154
155     private void returnToCodeScreenImmediately() {
156         handler.removeCallbacks(commandRunnable);
157         finish();
158     }
159
160     private void returnToCodeScreen() {
161         handler.postDelayed(() -> finish(), RETURN_DELAY_MS);
162     }
163
```

```
164     @Override
165     protected void onDestroy() {
166       super.onDestroy();
167       if (lightSensor != null) {
168        sensorManager.unregisterListener(lightSensorListener);
169        }
170        handler.removeCallbacks(commandRunnable); }
171 }
```

Listing 3: activity_main.xml

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:background="#535252">

    <EditText
        android:id="@+id/codeInput"
        android:layout_marginTop="60dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:minHeight="350dp"
        android:hint="Write or Click the Buttons for your Code"
        android:background="@drawable/edit_text_border"
        android:padding="20dp"/>

    <CheckBox
        android:id="@+id/ifLoopCheckbox"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="LOOP"
        android:layout_marginTop="16dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click this checkbox if you want the code
            to repeat itself , while front is clear." />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
```

```
35          android:layout_marginTop="30dp"
36          android:layout_marginLeft="60dp">
37
38          <Button
39              android:id="@+id/loopButton"
40              android:layout_width="wrap_content"
41              android:layout_height="wrap_content"
42              android:text="Loop"
43              android:backgroundTint="#3F423F" />
44
45          <Button
46              android:id="@+id/forwardButton"
47              android:layout_width="wrap_content"
48              android:layout_height="wrap_content"
49              android:text="Forward"
50              android:layout_marginStart="50dp"
51              android:backgroundTint="#3F423F" />
52
53          <Button
54              android:id="@+id/leftButton"
55              android:layout_width="wrap_content"
56              android:layout_height="wrap_content"
57              android:text="Left"
58              android:layout_marginStart="50dp"
59              android:backgroundTint="#3F423F" />
60
61          <Button
62              android:id="@+id/rightButton"
63              android:layout_width="wrap_content"
64              android:layout_height="wrap_content"
65              android:text="Right"
66              android:layout_marginStart="50dp"
67              android:backgroundTint="#3F423F" />
68
69          <Button
70              android:id="@+id/stopButton"
```

```
71              android:layout_width="wrap_content"
72              android:layout_height="wrap_content"
73              android:text="Stop"
74              android:layout_marginStart="50dp"
75              android:backgroundTint="#3F423F" />
76      </LinearLayout>
77
78      <Button
79          android:id="@+id/playButton"
80          android:layout_width="wrap_content"
81          android:layout_height="wrap_content"
82          android:text="Play"
83          android:layout_gravity="center_horizontal"
84          android:layout_marginTop="24dp"
85          android:backgroundTint="#4CAF50" />
86
87  </LinearLayout>
```

Listing 4: activity_result.xml

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:background="#FFFFFF">

    <ImageView
        android:id="@+id/faceImage"
        android:layout_width="500dp"
        android:layout_height="500dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="1dp"
        android:layout_marginBottom="24dp"
        android:backgroundTint="#FFFFFF"
        android:src="@drawable/face"


    <LinearLayout
        android:id="@+id/controlFields"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_alignParentBottom="true"
        android:gravity="center">

        <View
            android:id="@+id/leftField"
            android:layout_width="0dp"
            android:layout_height="400dp"
            android:layout_weight="1"
            android:background="#000000" />

        <View
            android:id="@+id/rightField"
```

```
36                android:layout_width="0dp"
37                android:layout_height="400dp"
38                android:layout_weight="1"
39                android:background="#000000"
40                android:layout_marginStart="8dp"/>
41        </LinearLayout>
42    </RelativeLayout>
```

Illustrations

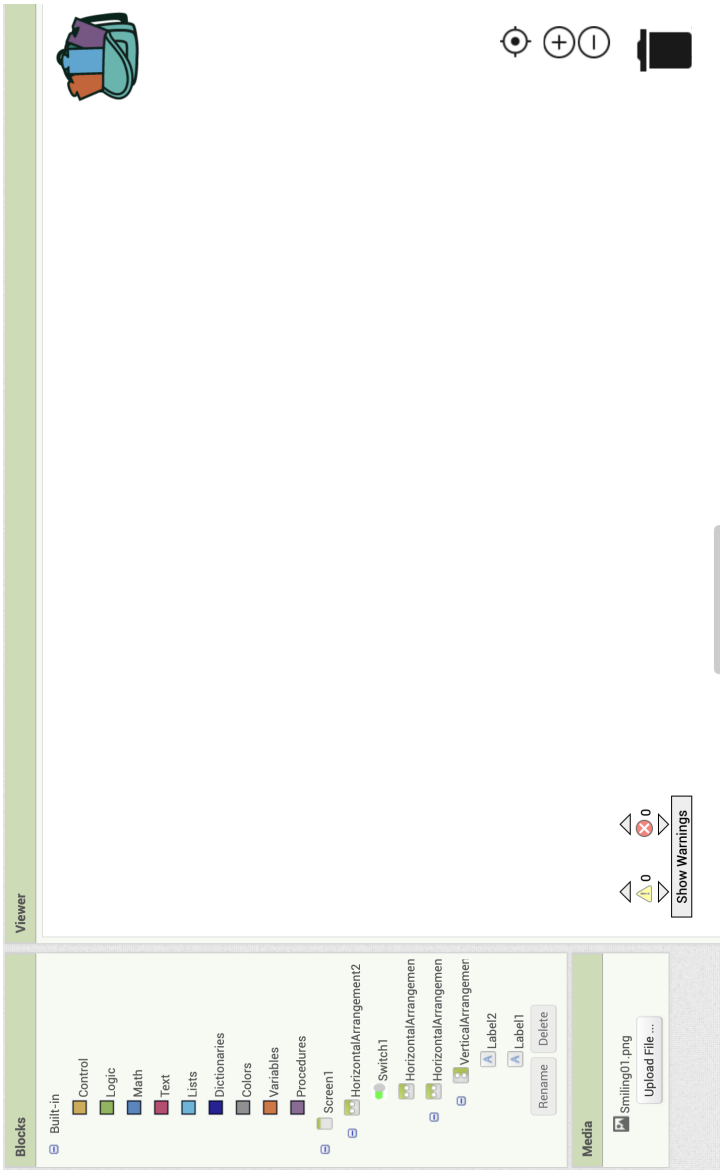


Figure 1.: The MIT App Inventor programming interface without any code; Source: `https://code.appinventor.mit.edu/`

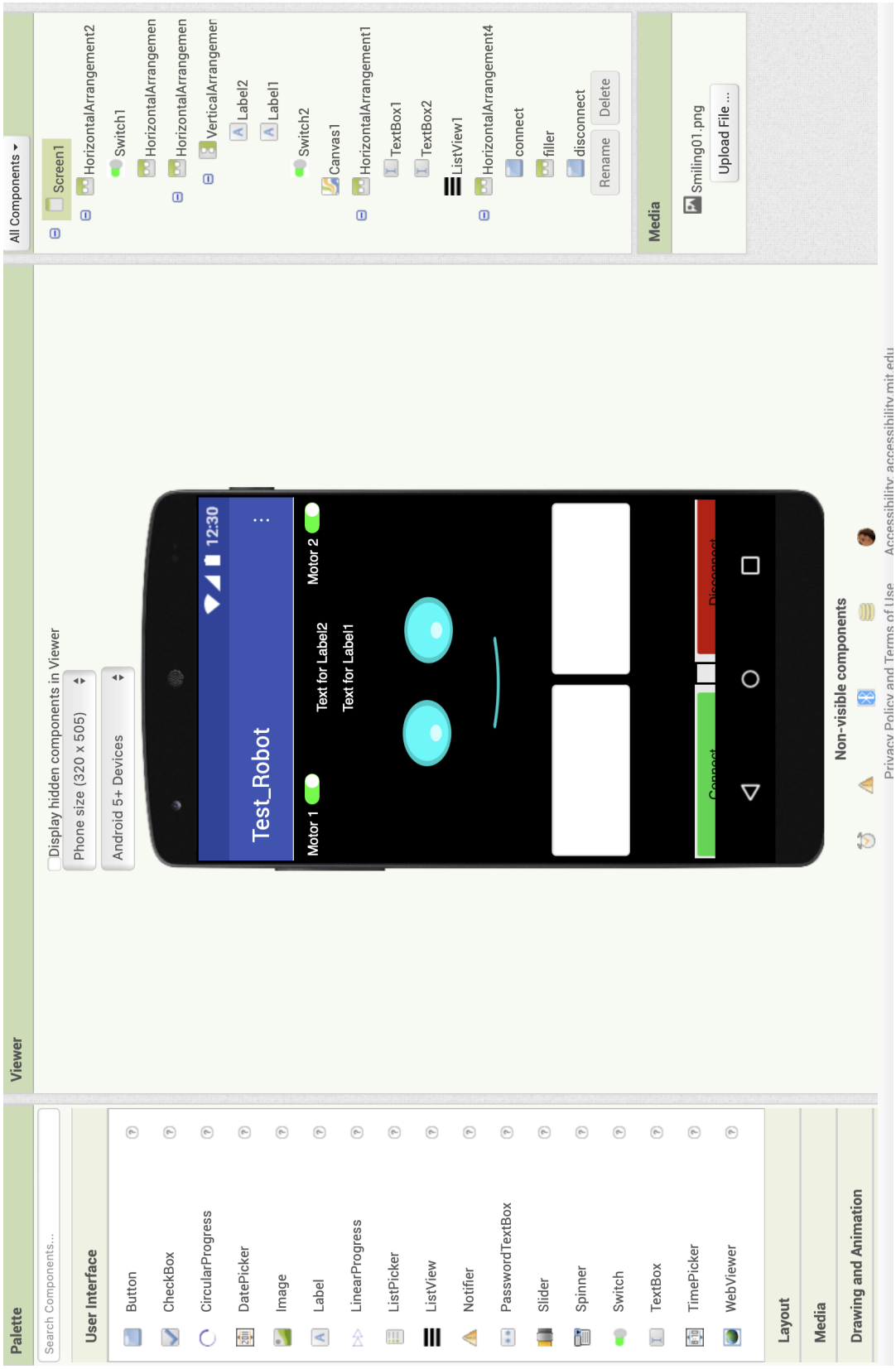


Figure 2.: The MIT App Inventor design interface with the design of the Robot-App; Source: `https://code.appinventor.mit.edu/`

Figure 3.: The MIT App Inventor design interface with the design of the Remote-App; Source: `https://code.appinventor.mit.edu/`

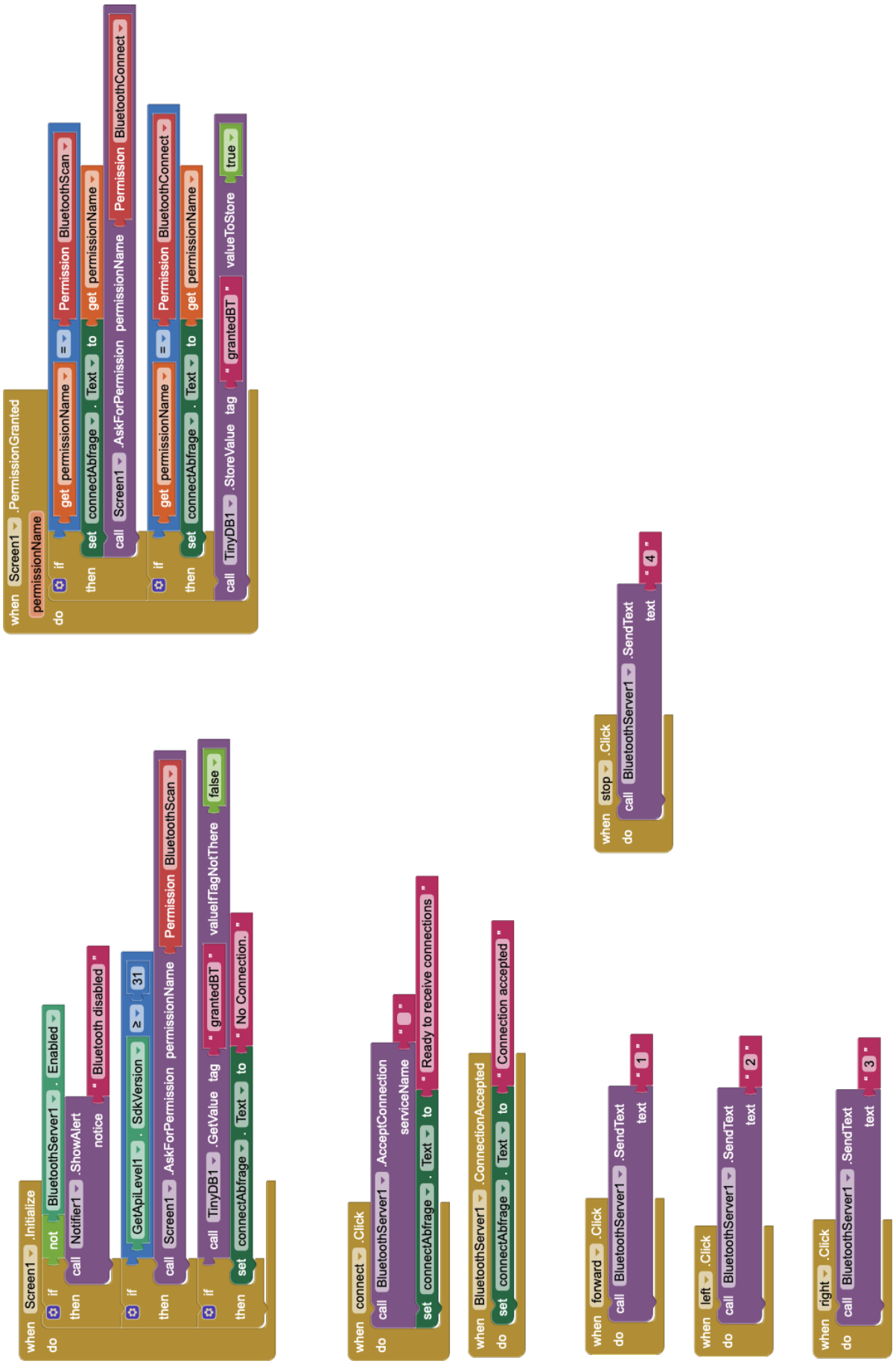


Figure 4.: The MIT App Inventor blockly interface with the code of the Remote-App; Source: `https://code.appinventor.mit.edu/`

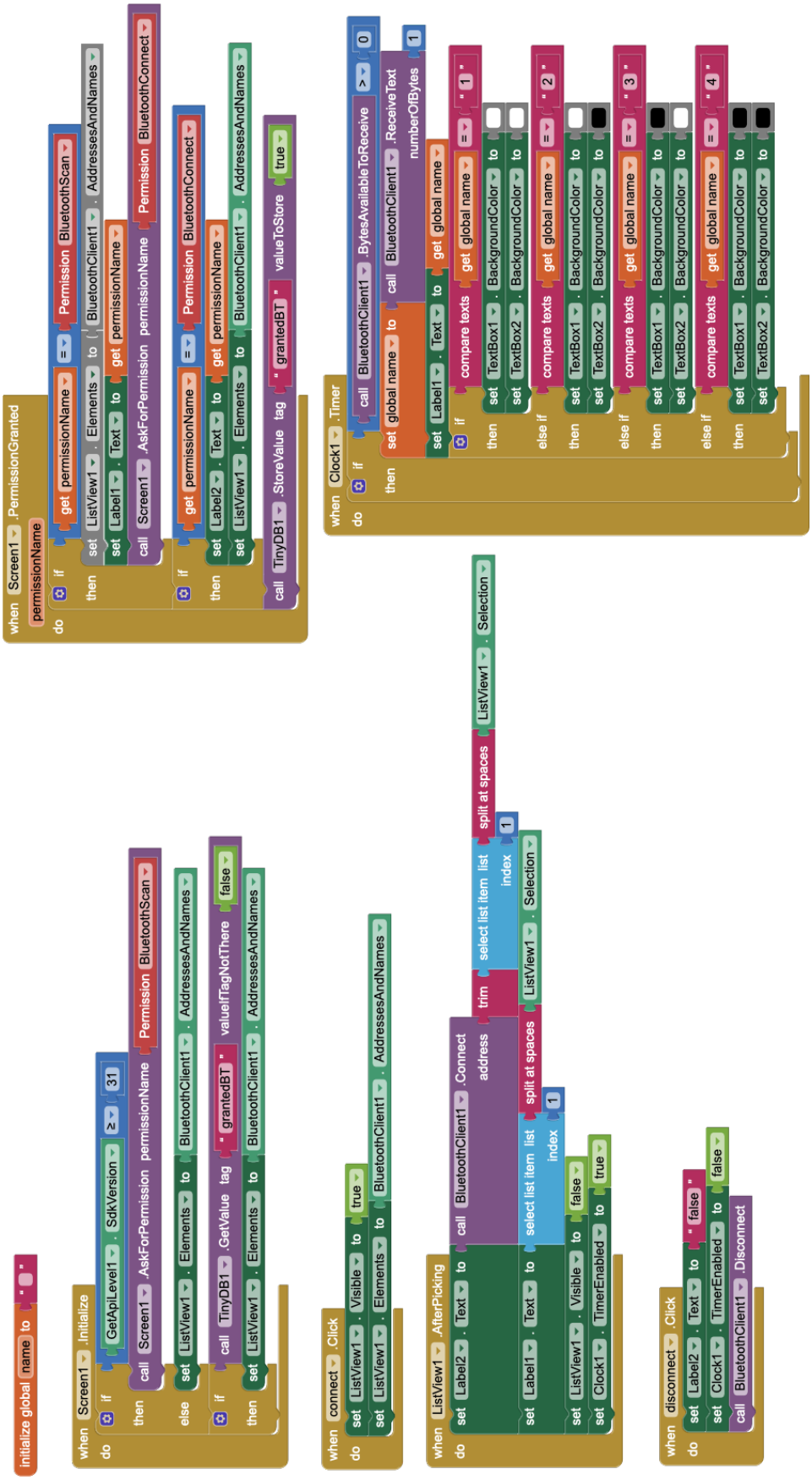


Figure 5.: The MIT App Inventor blockly interface with the design of the Remote-App; Source: `https://code.appinventor.mit.edu/`

Figure 6.: Circuit made with copper tape; Source: Author's own image

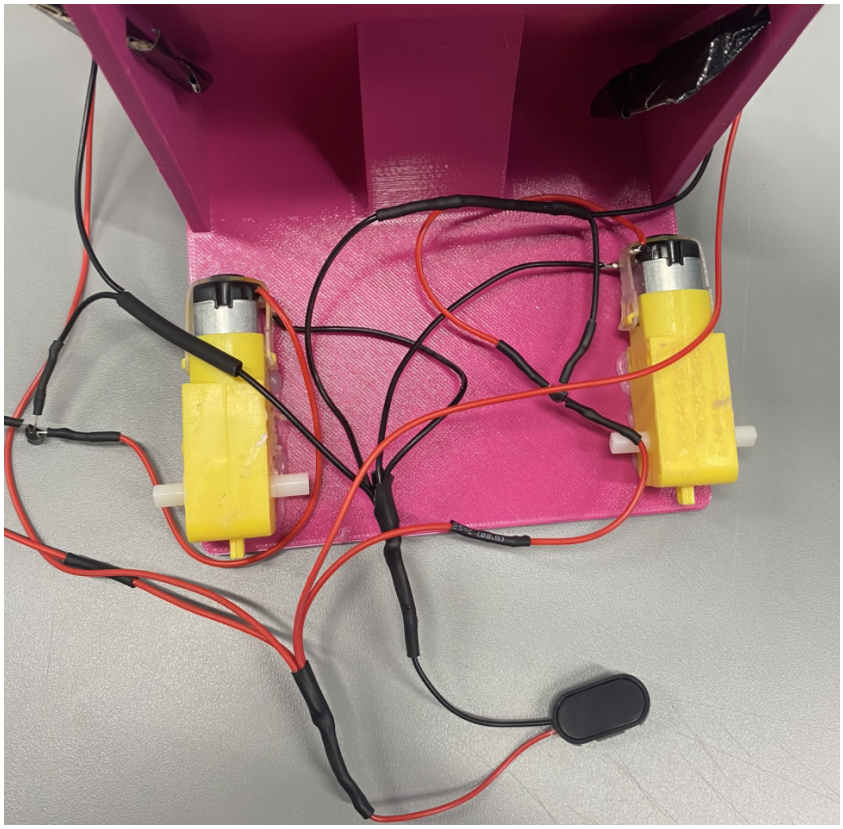


Figure 7.: Circuit that was soldered; Source: Author's own image

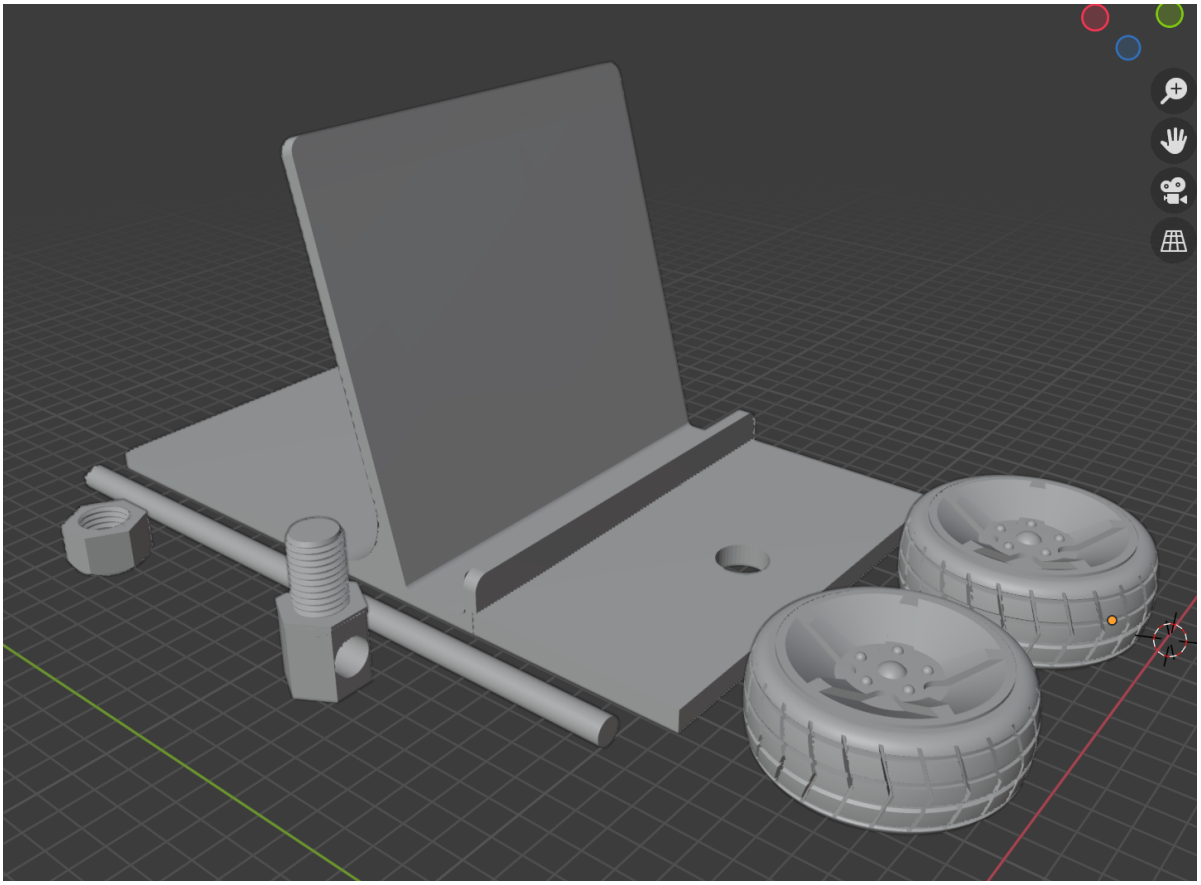


Figure 8.: First prototype created in Blender; Source: Author's own screenshot

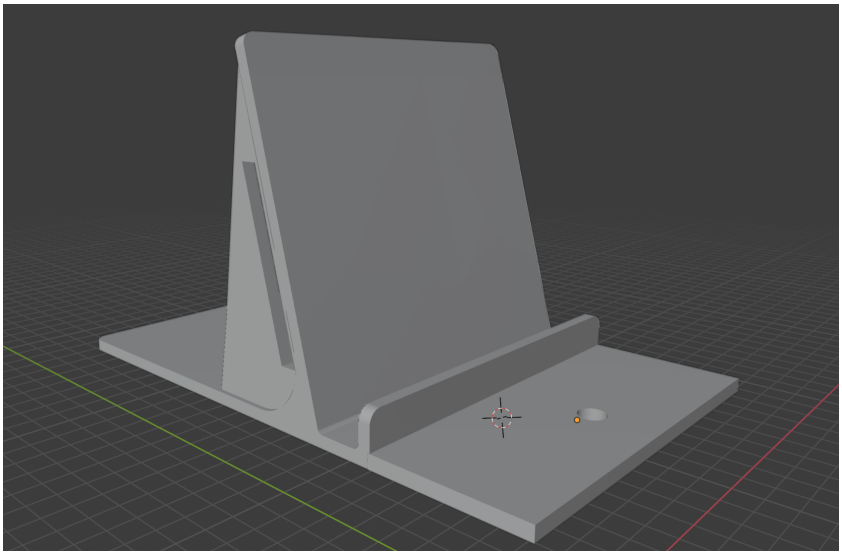


Figure 9.: Second prototyp created in Blender; Source: Author's own screenshot

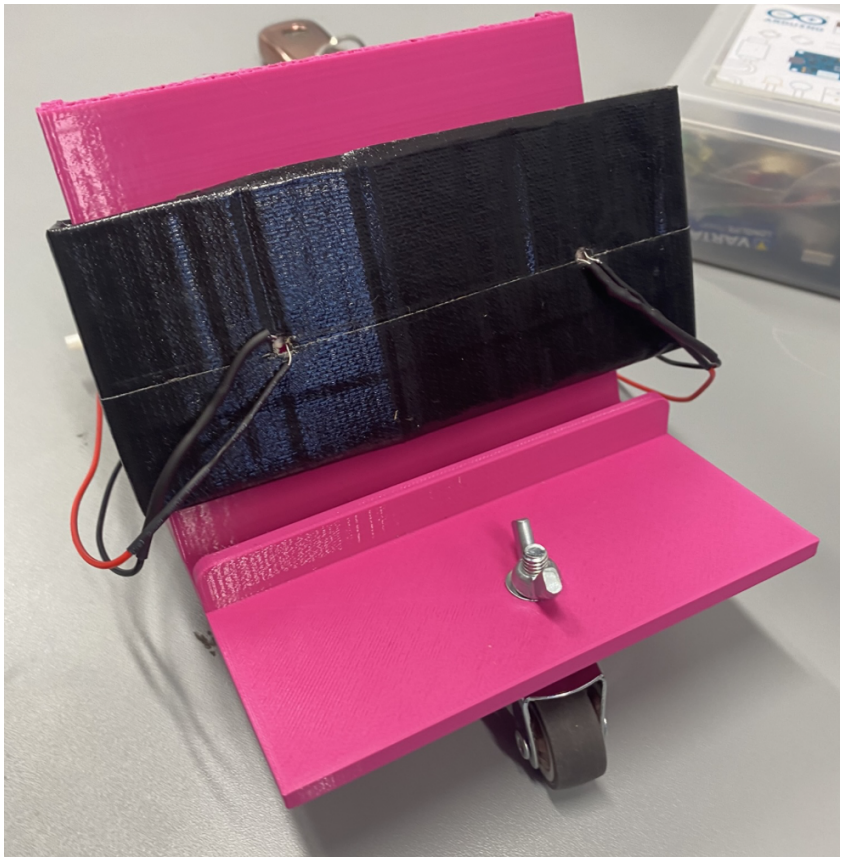


Figure 10.: Second prototyp printed with, cardboard bracket for LDRs; Source: Author

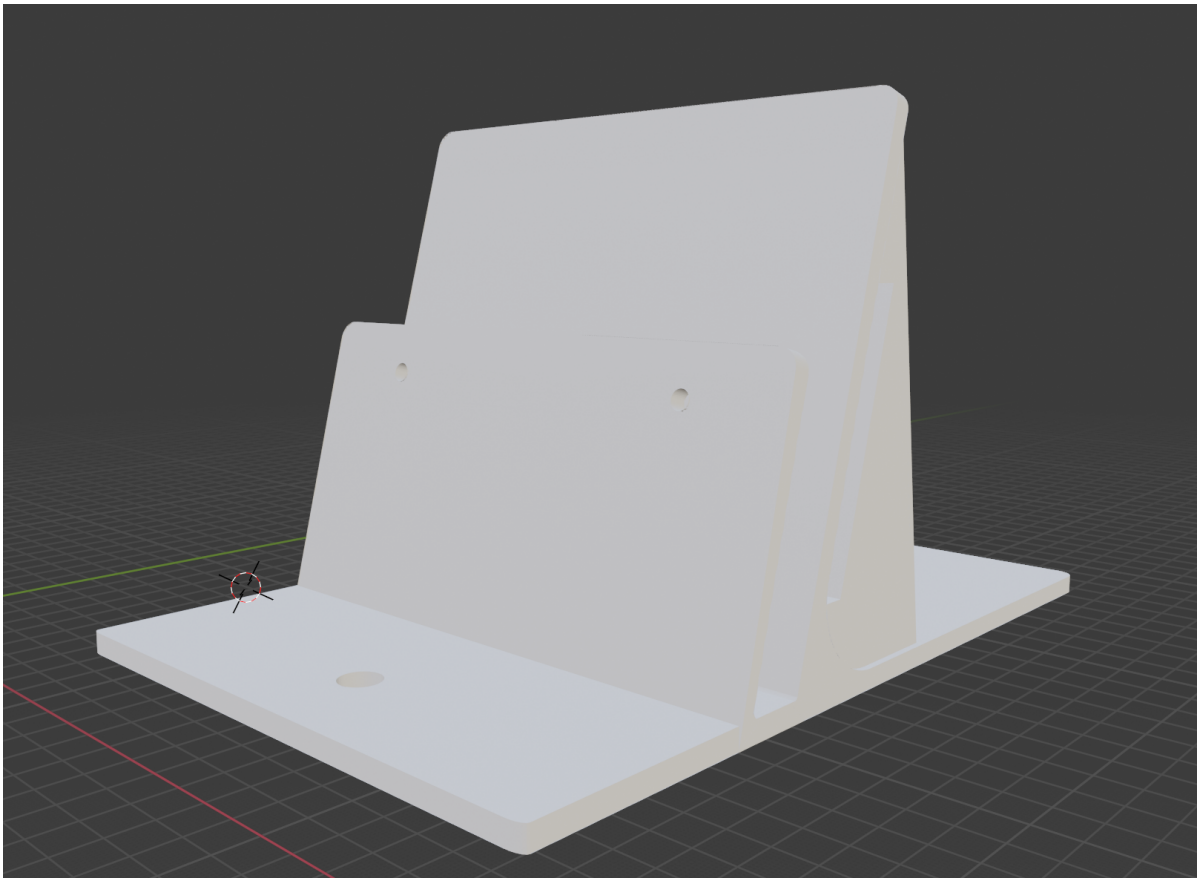


Figure 11.: Front vision of the final 3D-model version; Source: Author's own screenshot

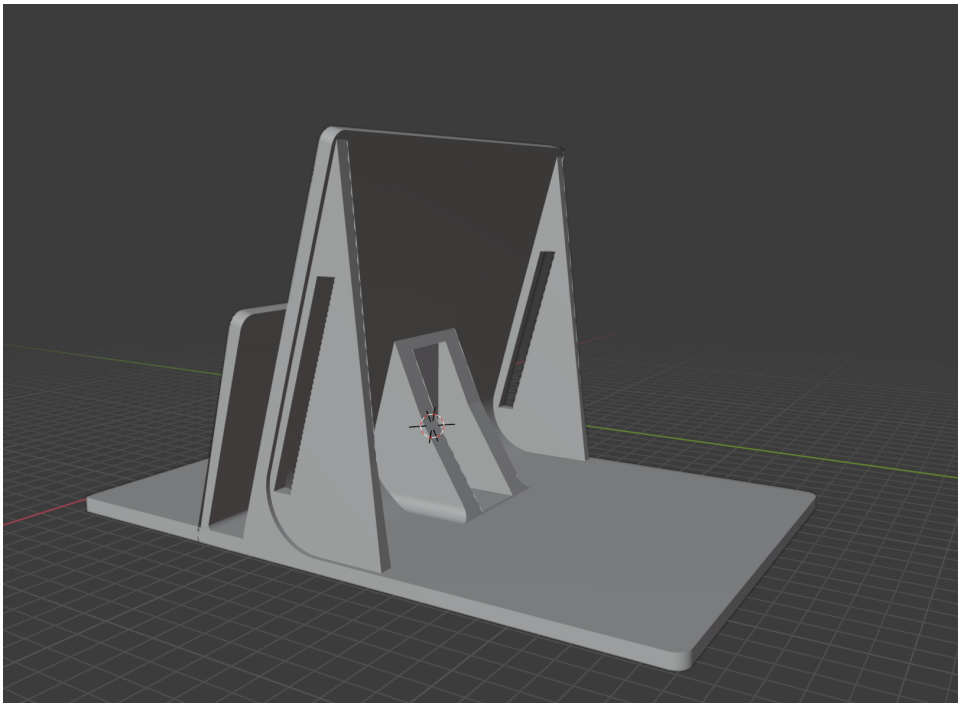


Figure 12.: Back vision of the final 3D-model version; Source: Author's own screenshot