University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

FHNW

School of Engineering

Department of Computer Science. iCompetence

Supervisor Prof. Dr. Dominik Gruntz

Customer Explore-IT

Robot Programming App

React-Native

IP 5

Borbely Sabina, Semester 7 and

Ott Roman, Semester 7

E-Mail sabina.borbely@students.fhnw.ch, roman.ott@students.fhnw.ch

Windisch, 20. January 2020

Abstract

The children taking part in the Explore IT events already enjoyed using the current Robby programming mobile app. Using this React Native mobile app, the 12-year children can program the movement of a differential drive robot. They can input the left and right wheel speed values for a given number of “robot steps”. The current mobile app user interface is more appropriate for a low number of different “robot steps”.

In this project we have set out to identify two possible approaches for extending the current mobile app with a new user interface. We have worked on the Blockly based approach and the Trainly approach, each approach having its own unique features which may work for or against it.

The Blockly based approach aimed to take advantage of the simplicity and elegance which are part of the power of Blockly. This approach has involved integrating Blockly, a web library into a React Native app, which is usually not the best decision when it comes to the performance of a React Native app. However, the rewarding part of this experience is that the end users can build a program out of colourful custom blocks. When the blocks fit together, the end user gets a clear audible verification "click". When the program is done and saved in the application, hundreds of “robot steps” can be uploaded to the robot.

The Trainly approach aimed to keep the existing speed input style but extend the functionality of the app by saving the speed inputs as blocks. The blocks can be linked together to form a new program or to create even more blocks. This linking of the blocks reminds of a train where the wagons are also linked together. With these options, the blocks can be created in a variety of ways.

Table of Contents

1 Introduction

1.1 Document Structure

This Document describes the way and the result of the Project Robot Programming App.

The Background, the existing Application, the Project goals and Constraints are explained in the Introduction. The chapter Interaction Design covers the theoretical aspects of the Interaction Design for the Robby app.

Then the Document is split in two design approaches Blockly and Trainly. Both chapters explain the UI design and the decisions behind the design. They continue with presenting the final UI design. Both parts explain their respective implementations. The structure of these chapters is not the same, because each design approach involved different challenges. Both chapters finish with an evaluation and possible improvements or extensions.

Finally, there is the Review and Reflection chapter.

1.2 Explore-IT Robot

Explore-IT organizes different technology related teaching events for children in the attempt to increase children's interest in technology. They developed a small robot that can be steered using a React Native mobile application. Our project goal is to extend the existing application Robby App with a new user interface.

The robot is a differential drive robot, meaning it is "a mobile robot whose movement is based on two separately driven wheels, placed on either side of the robot body. It can thus change its direction by varying the relative rate of rotation of its wheels and hence does not require an additional steering motion." (Wikipedia, 2019)

ExploreIT Org, The robot Robotik1

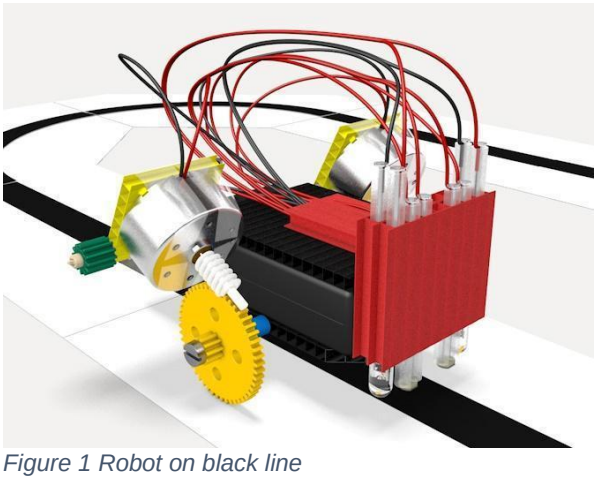


Figure 1 Robot on black line

ExploreIT Org, Robotik2

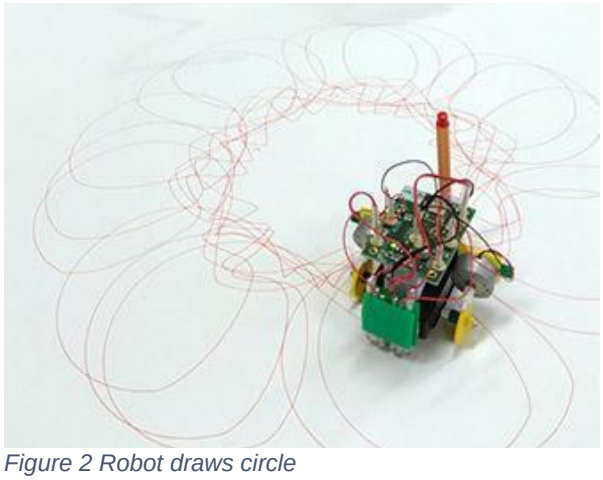


Figure 2 Robot draws circle

Julian, Robin (beide 12) und Amy (10) haben den Mini-Computer aus «Robotik 2» auf den Spurensucher aufgebaut und ihn mit Hilfe der explore-it-Applikation auf dem Tablet programmiert.

Source: https://www.explore-it.org/de/robotik-1/spurensucher-erforsche

The Robot can drive following a black line. It uses two light sensors to measure the reflected light on the surface. The light sensors change the electrical resistance and the motors turn faster or slower.

When both wheels turn at the same speed, the robot moves in a straight line. When the wheels move at different speeds, the robot turns. Thus, controlling the movement of this robot comes down to properly controlling the rates at which each of these two wheels turn. The robot repeats the steps x times per second as per the interval setting on the Settings screen. While other differential drive robots can move backwards, Explore IT robot cannot move backwards.

1.3 Robot Programming App

The existing React Native application allows users to input a list of speeds into the number fields on the Programming screen as in the screenshot below. Users could also add or delete a row/, which represents a step of the robot to or from the existing number of robot steps. Rows of speeds can also be swapped. A new row can be inserted at a given index. The user can also select rows and remove them.

Screenshots of the existing Robby App are available in the Appendix.

Communication between the robot and the mobile application is via BLE. Bluetooth Low Energy is designed to consume less power than Bluetooth Classic.

Before starting to steer the robot, the app has to be connected to the robot. We reuse the already available code for the connection and communication of the mobile application with the robot.

By clicking on the upload button, the speeds can be sent to the robot. This overrides the existing speeds on the robot's computer chip.

The play button lets the robot drive his saved speeds once. Then he stops and can be started again on clicking the play button.

With the record button the robot starts to drive a predefined time. It then uses the light sensors to follow the black lines on the surface.

By clicking the download button, the speeds saved on the robot's computer chip can be downloaded to the application. This overrides the existing speeds in the application.

1.4 Project Requirements

Our task is to research, evaluate possible versions and provide a solution for the following:

- An intuitive GUI easy to use for primary school children. Age 12+
- Extend the GUI with new screens
- A new user interface for allowing the user to input the robot speeds using basic programming blocks and if the time allows it, to provide the user with the choice of saving code blocks for future reuse.
- Basic programming sequences or blocks
- Programming blocks should contain loops

Implementation options: Blockly as a basis for the GUI or our own GUI version

1.5 Implementation Constraints

React Native Version to be used in the implementation 0.60.5 along with the library react-native-ble-plx necessary for the Bluetooth communication with the robot.

```
"react-native": "0.60.5",
   "react-native-ble-plx": "^1.1.0"
```

Implementing new code to steer the small robot focuses on the user interface implementation and state management. The mobile application performance is influenced among other things by the underlying communication protocol with the robot. We used the already implemented code for the communication with the robot, which uses an already defined protocol, Protocol Version VER 003.

The commands sent to the robot can be tested with Serial Bluetooth Terminal available in Google Play Store. (Google Play. Serial Bluetooth Terminal, 2020)

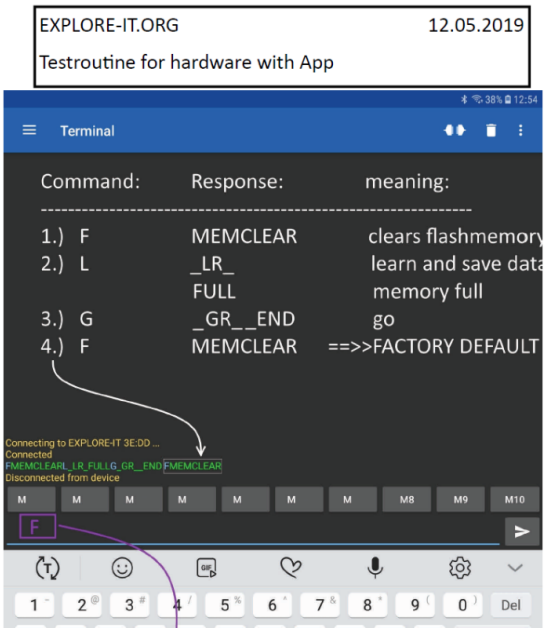


Figure 3 Serial Bluetooth Terminal

The protocol version used in our project, the commands sent to the robot and the responses from the robot can be checked out in the log messages when debugging with RN Debugger - Google Chrome Developer Tools. See Appendix 2 for some examples.

Robby App GitHub Project
https://github.com/explore-it-org/robby-app

FHNW GitLab Project
https://gitlab.fhnw.ch/roboter-programmier-app/robby-app-ip5
the branch develop3_sabina - Blockly Based Implementation
the branch develop2 Trainly- Trainly Based Implementation

1.6 Target Platforms

We needed to add new React Native screens and new features. The specific steps are different depending on what platform you're targeting IOs or Android.

The **project target platform** is Android while IOs is an optional platform. React Native supports IOs, too. IOS related documentation will be provided in this

document for future reference, however testing on IOS devices is not in the project scope. The Robby App applications runs on Android 4.3.x /Jellybean Android devices and above. We tested the mobile app on Android 9 devices.

Android Minimum SDK = 18, Android Target SDK=28

```
buildscript {
    ext {
        buildToolsVersion = "28.0.3"
        minSdkVersion = 18
        compileSdkVersion = 28
        targetSdkVersion = 28
        supportLibVersion = "28.0.0"
    }
    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath("com.android.tools.build:gradle:3.4.1")
```

Figure 4 Android Target SDK

Below is a list of our mobile devices used for testing the Android mobile application, Robby App:

- Samsung Galaxy A70, OS: Android 9.0
- Sony Xperia XA2, OS: Android 9.0
- Google Pixel 3a, OS: Android 9.0

2 Interaction Design

This chapter explains the different design approaches for the speed input. It explains the ideas behind each approach and why the approach is useful or unsuitable for the application.

2.1 Turn with an angle

- This input design approach uses turn commands to turn left or right.
- But there are more values necessary to steer the robot how to turn, like:
  - an angle value,
  - a speed value,
  - a radius value is needed in order to specify how big the turn should be

There are some problems with this approach

- The input has to be converted into speeds for the left and the right wheel separately.
- Every robot has a unique driving curve. This leads to different conversions for every robot.
- The approach is too complicated for the target group. It does not match our project target.

For these reasons, we decided to drop this approach.

2.2 Wheel speed as integer value

This is the same approach as the one used for the speed input in the already existing mobile app.

Number values from 0 to 100 define the left or the right wheel speeds. The robot is programmed to use these values and turn the wheel according the given speed value. Because each robot has different driving profiles, every input will have different physical effects on the robot. It will not be possible to use the same program on different robots and expect exactly the same physical drive path. But this approach has been already tested and easy to understand for the target group. That is why we decided to keep this approach in every further interaction design approach.

3 Blockly

After introducing Blockly, this chapter explains the customized blocks created with the Blockly Library. It explains the ideas behind the custom blocks and whether they were appropriate for the Robby App mobile application.

The block-based approach to programming is an effective way to engage young learners in programming. Anyone who has ever played with Scratch knows the appeal of snapping together a few simple blocks to make something creative and even functional. The Blockly library provides that.

Blockly is an extensible software library to build visual programming editors. It is a Google project, being open source under the Apache 2.0 License, and is delivered with a few premade visual blocks. Google Developers. Blockly (2019) is a website where you can try out Blockly, check out the ready-made blocks and find links to applications built with Blockly.

We needed to generate custom blocks that fit the needs of the Robby mobile app.

3.1 Blockly Based UI Design Experiments

Blocks can be grouped into categories. The categories Loops, Math and movement are relevant for Robby App.

Custom Blockly blocks can be generated with Blockly Factory. I tried different designs of the blocks, considering all possible movement related blocks as well as blocks for loops and input of the left and right speed values.

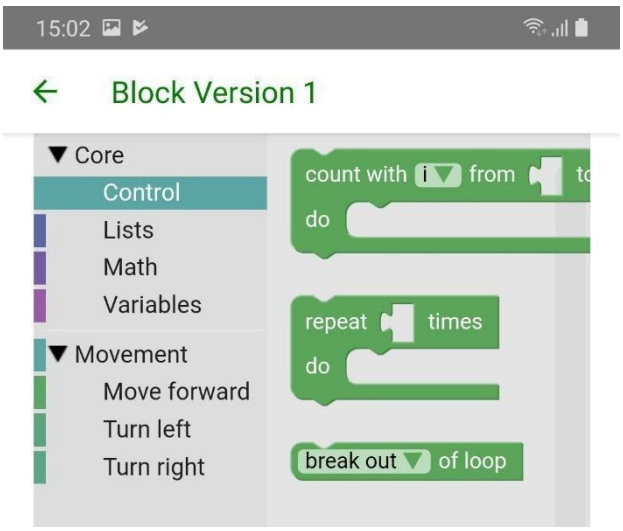


Figure 5 Loop Blocks

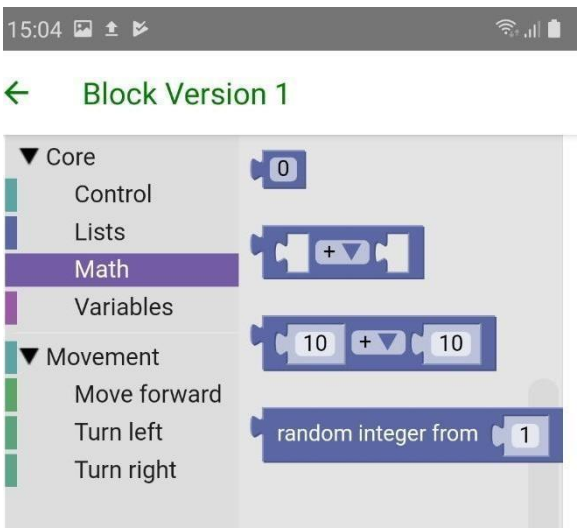


Figure 6 Math Blocks

The above screenshots show **ready-made loop blocks** (the screenshot on the left) which use as input the math values provided as simple number or random number

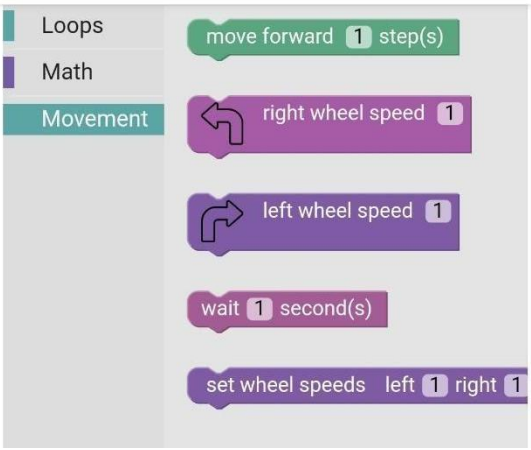
blocks (the screenshot on the right). The client considered it is better to have just simple number edit fields without having to drag and drop a *number* block and fit it into the space for the input value of a loop or other block. Consequently, we generated a custom loop block (see below)

Figure 7 Custom Loop Block

All the other blocks should be in line with the client's request, when it comes to providing input values.

Blocks Version 1. Available Blocks:

- Loops (repeat x times, do)
- Move forward x steps. Change right/left speed by a value
- Set left/right speed to x (dropdown with the list of variables: left and right speed)
- Turn right with the left wheel speed of 18, right wheel speed is 0,
- Turn left - with the right wheel speed of 18,
- Wait x seconds (this would translate to left and right wheel speeds={0, 0})



The screenshot on the left shows the version 1 custom blocks for the movement category. They have been generated with Blockly Factory and integrated into the Blockly web app.

See the screenshots in Appendix 4 for an example of a basic sequence of robot steps, created by putting together the respective blocks.

Figure 8 Movement Blocks

All these blocks can be translated into robot speeds (except turn right and turn left which are more problematic). However these blocks seemed to be too complex for the simple capabilities of the robot and didn't meet the expections of some of the mobile app end users.

Version 2 Custom Blockly Blocks.

The Blockly blocks are grouped into the same categories except a newer category "My Blocks" supposed to include blocks for different saved block names.

The Movement Category:

- Move forward & Set left and right wheel speeds.
- Wait x second(s) – this implies a “step” with the left wheel speed value: 0 and the right wheel speed value: 0
- Turn left. - this implies a number of steps with left wheel speed: 0, right wheel speed > 35.
- Turn right – this implies a number of steps with right wheel speed: 0, left wheel speed > 35

The number of steps needed for an obvious turn left can be hard to determine. After several trials, we decided to give up the idea of the 2 block types: turn right and turn left.

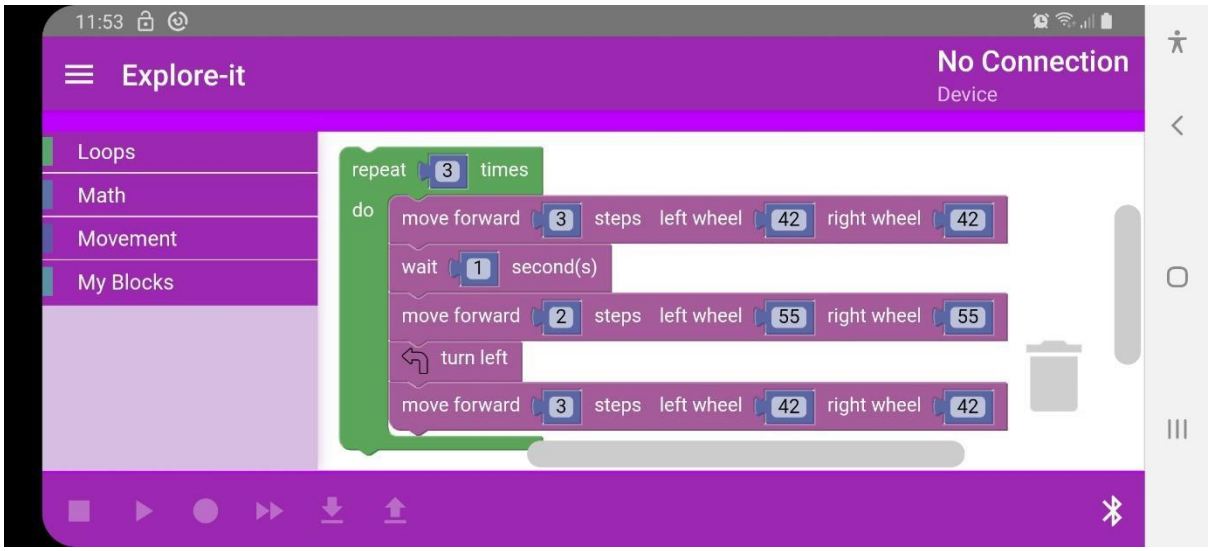


Figure 9 Custom Blocks. Version 2

See Appendix 5 for more Version 2 screenshots.

We thought we could have a category called “My Blocks” or “Saved” for the saved block names. However, the management of these blocks, inserting a program as a saved block into a new program and setting the name of a block should belong to the React Native part, because more complex state management is better done in the RN part. Consequently, we gave up on the idea of using block inserting blocks.

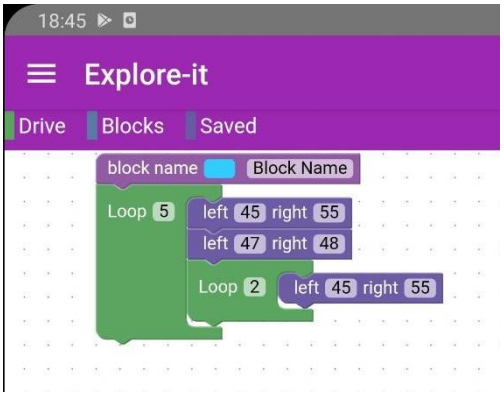


Figure 10 Block Name- Define a block. Saved Blocks

3.2 Final Blockly Based UI

Custom Blockly Blocks

Based on the feedback on the previous block versions, we decided it is better to have only block types: loops and the speed inputs for the sake of simplicity and quick understanding of the app behaviour relative to the robot.

These simple blocks allow the blocks to be combined without any screen real estate problems in vertical mode. The drop downs show the available blocks that can be added to the workspace by drag and drop.

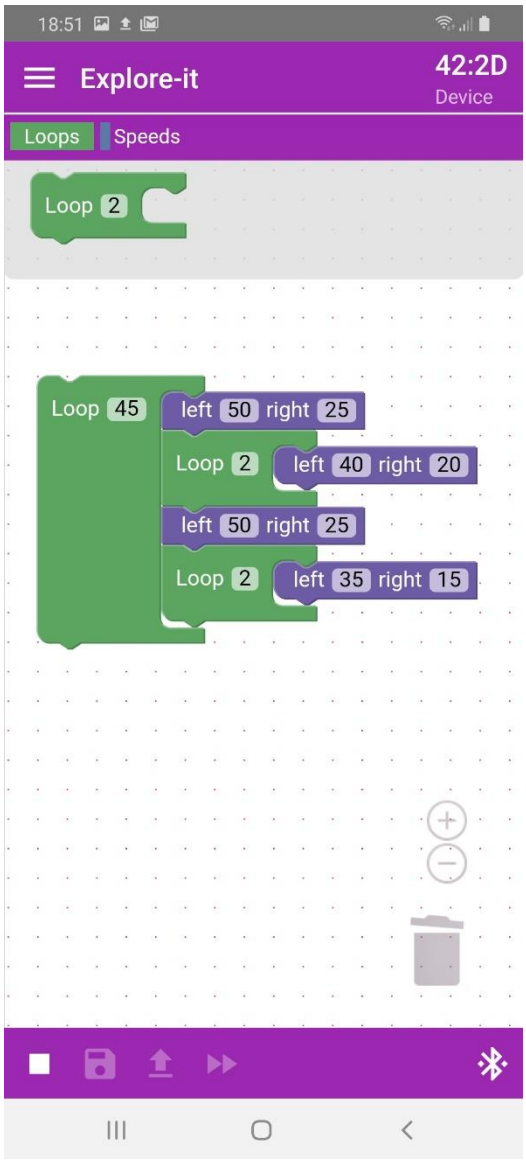


Figure 11 Final Design. Loop Block

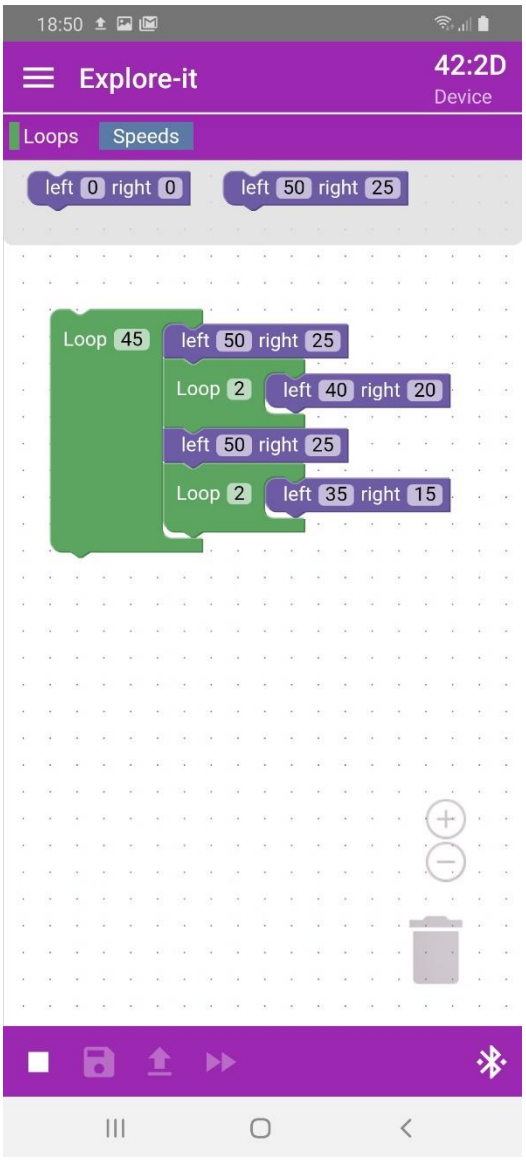


Figure 12 Final Design Speed Blocks

App Navigation

We extended the already existing navigation drawer. See Appendix 6 for a screenshot of the navigation drawer.

3.2 Project Git Repositories

Robby App GitHub Project https://github.com/explore-it-org/robby-app FHNW GitLab Project https://gitlab.fhnw.ch/roboter-programmier-app/robby-app-ip5 the branch develop3_sabina - Blockly Based Implementation the branch develop2 Trainly- Trainly Based Implementation

3.3 Blockly Based Implementation

3.3.1 Technology Choice

Blockly

The block-based approach to programming is an effective way to engage young learners in programming.

Along with the Blockly download, there is a program called the Blockly Factory that assists you in the creation of your new Blockly language and blocks. And, all of this (including languages you develop) runs in an ordinary browser using standard JavaScript. Thus, custom blocks could be easily generated and integrated into our app.

3.3.2 Implementation

Blockly is a web application and communicating between RN and the Blockly web application could get too complicated. We think it is better to exploit the benefits of using Blockly such as building a program to steer the robot by using customized colourful blocks that click together, but React Native UI components are more appropriate for the implementation of the block management related UI, that is supposed to support the end user for tasks like naming and saving several blocks for future reuse

Combining programs is something left for future implementation because the number of steps could get very large when there are more nested loops, and this implies using a newer and faster version of the communication protocol. We tested the app for 10000 steps with a step rate of 2 and uploading the speeds to the robot took

longer than 20 minutes with the current protocol Version 003. We stopped the upload and the app crashed when trying to send the command “GO” or “play forward” (in terms of the icons in the bottom toolbar) for what had been already uploaded (more than half of the steps).

Block Programming Screen Implementation.

This is covered in the folder blockpr. You can see below the folder structure

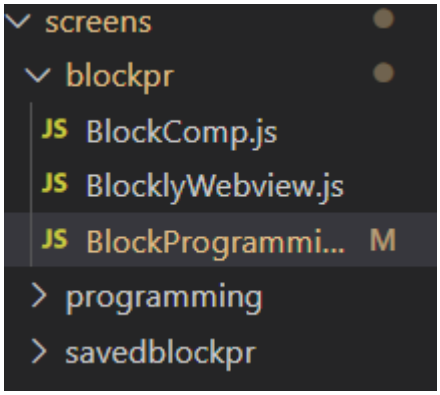


Figure 13 Blockly Screen. Code Folder Structure

Block Programming is the parent component passing down properties to its child components: BlockComp.js

The component Block Programming represents the component that can access state and also displays the bottom tollbar with the respective icons for saving, uploading speeds and playing fast forward.

BlockComp.js is the component representing the current component with its state and passes down properties to its child BlocklyWebview.js Below is a screenshot of the render() of BlockComp.js :

```
render() {
    return (
      <BlocklyWebview ref={r => (this.webviewref = r)}
        receiveCodeAsString={this.receiveCodeAsString} block_xml={th
is.state.block_xml} />
    );

  }
```

Integrating Blockly into the RN App. BlocklyWebview.js Component

BlocklyWebView is the component that uses a React Native WebView component to display the Blockly content.

React Native Community. React Native WebView

React Native WebView is a modern, „well-supported, and cross-platform WebView for React Native. It is intended to be a replacement for the built-in WebView (which will be removed from core).“

GitHub Repo https://github.com/react-native-community/react-native-webview

React Native WebView supported platforms are both Android and IOS. WebView in ReactNative is the common component used in both Android & iOS applications in React native to embed or to load web content in a native view.

See the details on adding WebView to the project, namely the code of the BlocklyWebView component in Appendix 7.

Communicating between JS and Native

Communication between the WebView and the Blockly app implies sending messages to the web page loaded by the webview and also receiving messages back from that web page.

To accomplish this, we used the following 3 React Native WebView properties:

1. React Native -> Web: The `injectedJavaScript` prop
2. React Native -> Web: The `injectJavaScript` method
3. Web -> React Native: The `postMessage` method and `onMessage` prop

More details on the available React Native WebView properties are available in the online documentation. (React Native WebView Guide, 2019)

As already mentioned, the communication is a 2-way communication.

For sending a request from BlockProgramming to BlocklyWebView.js to pass data to RN we used refs. This is an unusual situation where the use of refs instead of properties was appropriate as the parent sends a request to a child. See the relevant code handleSaveClicked() in BlockProgramming.js below:

```
handleSaveClicked() {
  const get_workspacedata = `sendGeneratedCodetoRN(); `;
  this.blocklycomp.webviewref.webref.injectJavaScript(get_workspacedata);
  console.log("event sent to the web app");
}
```

The code below is a part of the return() of BlockProgramming.js

```
<BlockComp block_name="" block_steps={[{ left: 0, right: 0 }]}
            ref={r => (this.blocklycomp = r)}
            block_xml="" updateSpeedsInStore={this.updateSpeedsInSto
re} saveBlock={this.saveBlock}
          />
```

The React Native Webview passes back data by calling onMessage

```
onMessage={event => {
            const { data } = event.nativeEvent;
            { receiveCodeAsString(data) };
            console.log("code from the web app :" + data);
          }}
```

Data is here the generated code as string and also the XML code representing the current workspace. receiveCodeAsString is a property passed down from BlockComp.js. Thus the handler code for receiving the data belongs to BlockComp.

State management

The difficult part is the fact that web and native mobile are different experiences, which means that the shape of the application state will be different. It has to be different; otherwise, you would just be porting from one platform to the other.

The application works OK as far as the speeds are concerned without Redux only by using the code in BlockSpeedsStore.js. The commit SHA for the version before actually using Redux is 014163fe5292260778ddb28b55f395254b400a61.

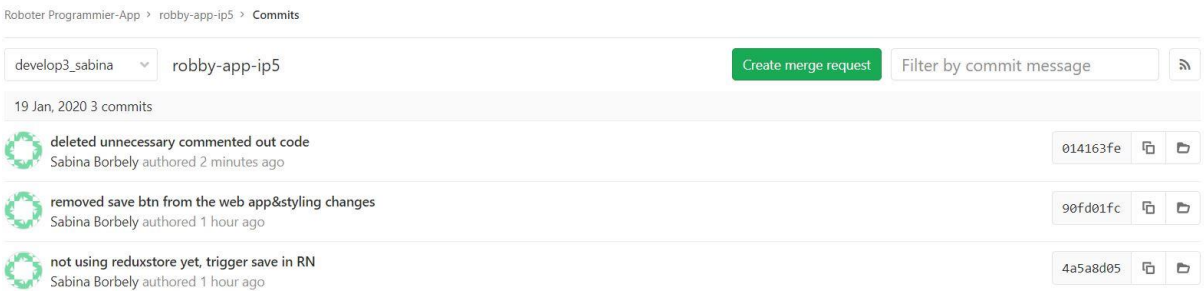


Figure 15 Last Commit Before a Redux Store is actually used. 014163fe

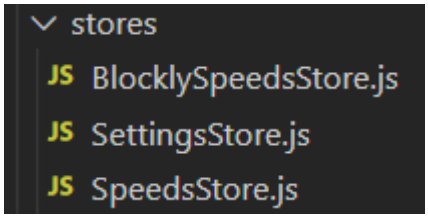


Figure 14 BlocklySpeedsStore.js

State management with Redux

The code is in the folder blockly_reduxstore. This was need more for saving blocks.

In Redux, you divide up application state by slices. We have 2 reducers one for blocks and one for speeds.

Folder Structure

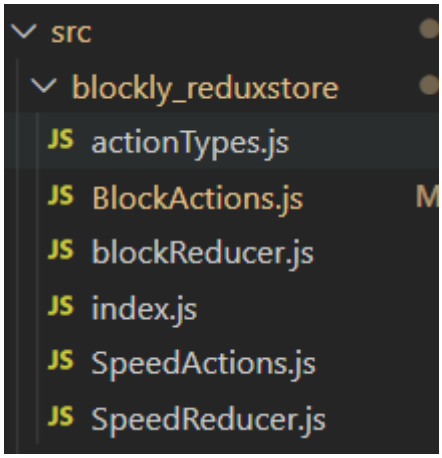


Figure 16 blockly_reduxstore

The BlockComp class has the current state of the current block but also embeds a view for displaying the Blockly app.

More information is available in the comments added to the code on Gitlab. (Branch develop3_sabina)

3.3.3 Evaluation

One possibility of integrating Blockly into RobbyApp is to use WebViews inside simple native wrapper applications. I tried this and I think it's not a bad idea considering the advantage of using the expressiveness of the Blockly blocks. I noticed some performance issues in comparison with other mobile native applications; the app was slower than other native apps on the same Android phone.

Below is some feedback on the outcome of embedding a web app into a React Native app using WebView:

While the implementation didn't provide the performance we wanted, this approach is flexible enough and comes with the ability to take advantage of Blockly and its simplicity. However, because all the rendering is done using web technologies, the app doesn't provide a truly native user experience. The current React Native WebView is more performant compared to previous versions of the WebView.

A more recent article referring to the newer WebView version:

"The WebView element is a powerful API through you can do magic things even like a mini Web Browser embedded inside your React Native app. It's a channel that will connect your apps in different platforms (mobile, Web)." (Blog WebView The Bridge that connects React Native with the Web, 2019)

3.3.4 Possible Improvements and Extensions

An Alternative to Using React Native Webview

React Native with Typescript might be used for integrating Blockly into a React Native app using the npm Blockly module for Typescript. However, we need to research more on the topic.

Improvements

Using a database for saving blocks is something we didn't have time to implement. It was planned if the time allowed it. Now only a few blocks can be saved in in the redux store. Redux Persist should be working in the final app and the redux store can be saved on refresh.

One area that would benefit from further future work is the proper performance optimization in terms of the communication protocol. As far as we know there is already a newer version of the communication protocol, which is faster.

4 Trainly

This chapter explains the Trainly approach. It starts with the original Design Idea. It continues with the User Interface Implementation. In the chapter “Example Program Trainly” it is explained how to program a cross. It uses the final UI Implementation to do so. This chapter then explains the store and programming implementations. Then comes a list of used RN Libraries. After that comes an evaluation and last possible Improvements and Extensions.

4.1 Design Idea

This chapter explains the originally design idea for the Trainly approach. It should explain the design of the Trainly approach, why we made those decision and explains where the application could go. This is not the final User Interface.

4.1.1 Trainly Based UI

This design reminds of a train with many wagons. The Train can have one or many wagons. Each wagon can be linked together. But instead of wagons the program uses blocks. Inside the blocks are the speeds for the robot. After the blocks are created and filled with speeds, they can be linked together to create the program.

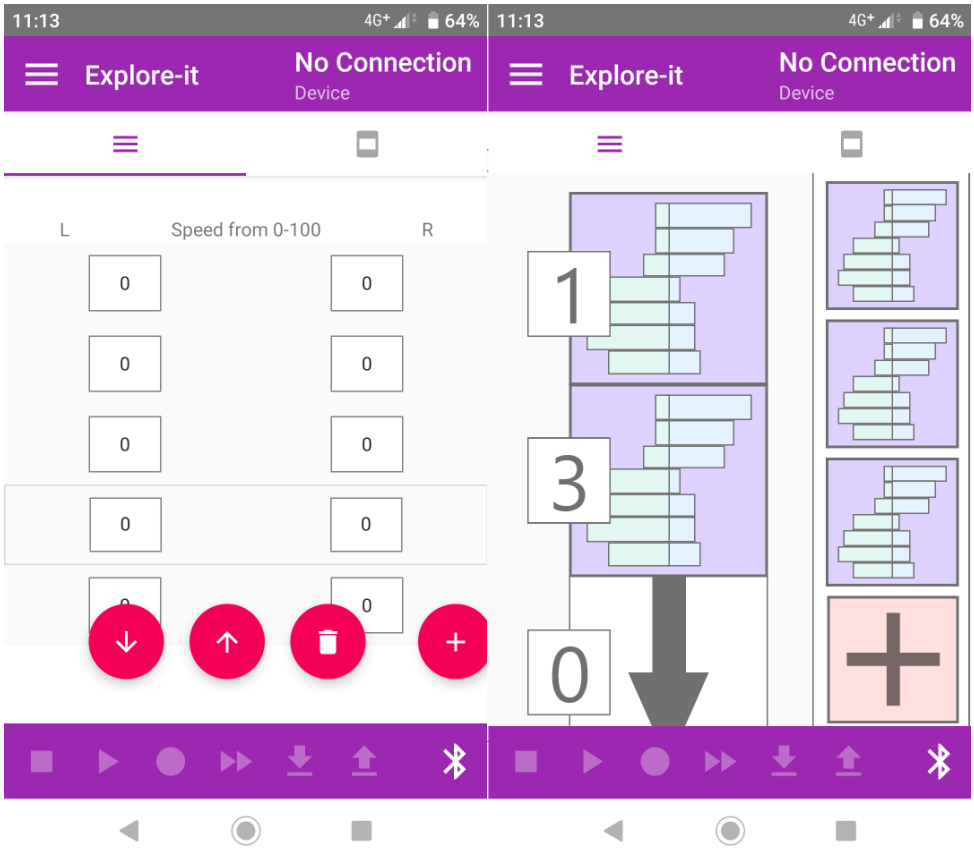


Figure 17 Speed Input

Figure 18 Blockprogramming Design

In the existing application the speed input for the left and the right wheel can be added by clicking the number fields in the “**Fehler! Verweisquelle konnte nicht gefunden werden.**”. It is also possible to upload the collected speeds from the robot to the application.

Because this approach was a successful method to program the robot and the children used it very intuitive. We decided to keep this screen and use it in the Trainly approach.

The Trainly approach let the user create new blocks “Figure 18 Blockprogramming Design”. These blocks consist of the speeds which come from the speed input screen “Figure 17 Speed Input”. These blocks are saved on the application and can be edited by clicking on an existing block. The collection of the existing blocks is on the right side of the screen on “Figure 18 Blockprogramming Design”. To use a block, drag and drop it from the collection to the vertical programming field on the left “Figure 18 Blockprogramming Design”. This creates a copy of the block on the programming side. The user can stack multiple blocks behind each other to generate a “Train”. The idea is that the blocks are run through one after another, from the top to the bottom. There is also a number input field to adjust how many times the block should be repeated till the next block is executed.

The program can be saved in an own collection of programs. This is not shown on the pictures.

A further possibility is to generate a new block out of many blocks. This could be done after the user finished stacking the blocks. This would be also a method to save the program by generating a new block out of it. This new generated block consists only of speeds for the wheels exactly like all other blocks.

The Idea for the Icons of the blocks is to generate them through the speeds which are saved inside the blocks. With this the user can directly tell which block is faster or slower and if the robot will turn left, right or move straight.

An easier way is to let the user choose a name for the block.

Most important for the design of the icons is to differentiate the different blocks from one another and to easy tell what the block does by looking at the icon.

The direction of the train is vertical because of the screen format from smartphones. It gives a better overview of the program and is more intuitive to scroll vertical than horizontal.

4.2 UI Implementation

This Chapter explains the different Implementation and design decisions for single UI elements. First it is explained how and why we used the lists for the blocks and Second it is explained how and why we used the control buttons.

4.2.1 Blocklist and Programminglist

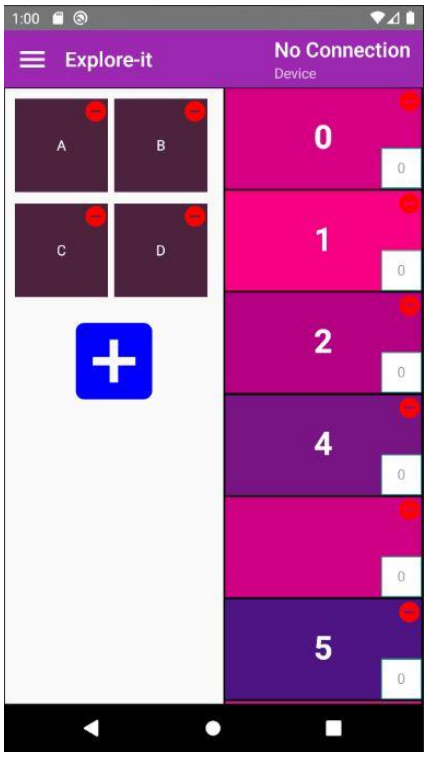


Figure 19 Lists

For the UI we want to create two lists. One list to collect the existing blocks. We call it Blocklist. And the second list to program with those blocks from the Blocklist. We call it Programminglist. The lists are shown in “Figure 19 Lists”

In the “Figure 19 Lists” the Blocklist is on the left side and the Programminglist is on the right side. We wanted to drag and drop the blocks from the Blocklist to the Programminglist. This way we could add a copy from the block in the Programminglist and continue to use the block there.

Research for a library where we could drag and drop between many lists gave no result. We found out, that there is a big need for such a library.

We used a “Flatlist” instead to create the Blocklist.

For our second approach to drag and drop we used a library called “draggable”

We tried to use the library “draggable” to drag and drop the elements in the Blocklist. But we did not

manage to drag the elements outside the “Flatlist” shown in “Figure 20 Draggable failure”.

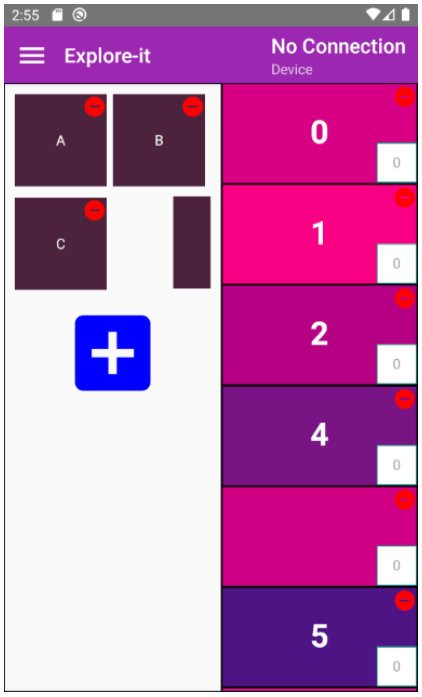


Figure 20 Draggable failure

For the Programminglist we used another library called “draggable-flatlist” it allows to create a list and drag and drop its elements inside the list. The “draggable-flatlist” is a good way to move the blocks, but because it only allows to move the blocks in itself, we needed another way to fill the Programminglist with the blocks from the Blocklist.

4.2.2 Control Buttons

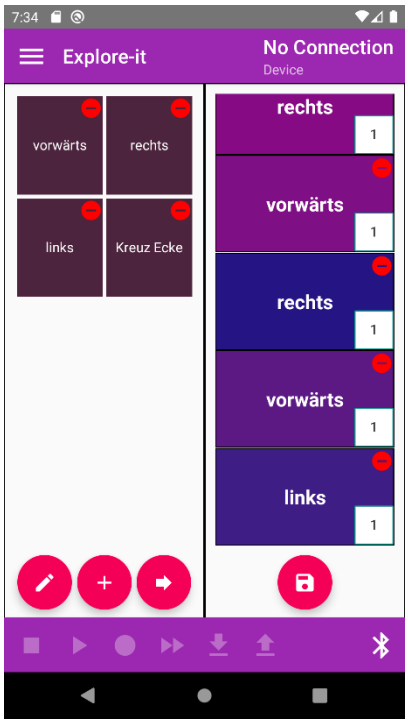


Figure 21 Control Buttons

To solve the problem of filling the Programminglist, we used a “FAB” element. Shown in “Figure 21 Control Buttons”, the circle with the arrow. The “FAB” elements work just like Buttons.

To actually fill the Programminglist the block has to be chosen by a click on it. Then the block can be added at the bottom of the Programminglist by clicking the “Right-Arrow” button.

After a Block is chosen in the Blocklist it can be filled with Speeds by clicking the “edit” button.

The “add” creates another block in the blocklist. Finally, the only button in the Programminglist is the “save” button. It saves the program as block and adds a new block to the Blocklist.

“Figure 21 Control Buttons” also shows the final design of the UI. An example of a program can be seen in the chapter “Review”.

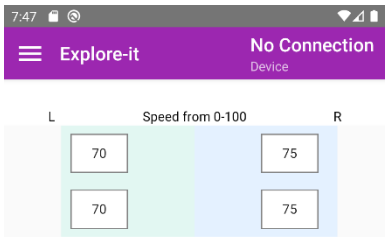
4.3 Example Program Trainly

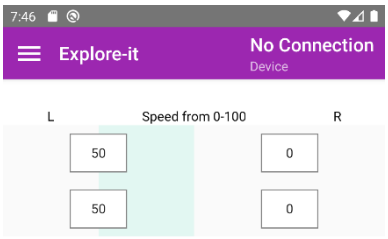
This chapter gives an example of creating a program with the Trainly approach. This is a good way to show the final UI.

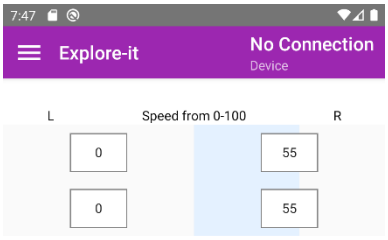
The Robot should drive a cross. The cross should look like “Figure 22 Cross”.

Figure 22 Cross

We start by creating three blocks. One with a right turn, one with a left turn and one who drives just forward.







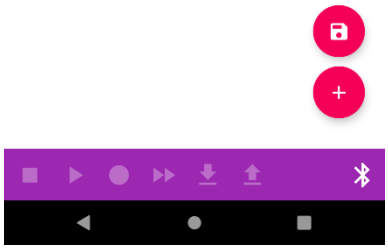


Figure 25 forward

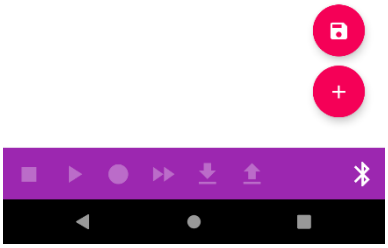


Figure 24 right

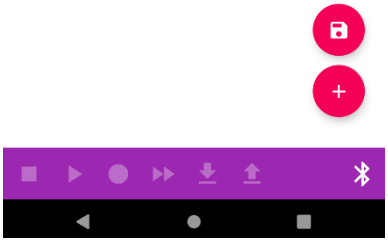


Figure 23 left

The “forward” block becomes the input numbers. For the left wheel: 70 and for the right wheel: 75. As shown in “Figure 25 forward”. The speeds are different because of the possibility that the wheels have different speed curves.

The “right” block becomes the input numbers. For the left wheel: 50 and for the right wheel: 0. As shown in “Figure 24 right”.

The “left” block becomes the input numbers. For the left wheel: 0 and for the right wheel: 75. As shown in “Figure 23 left”.

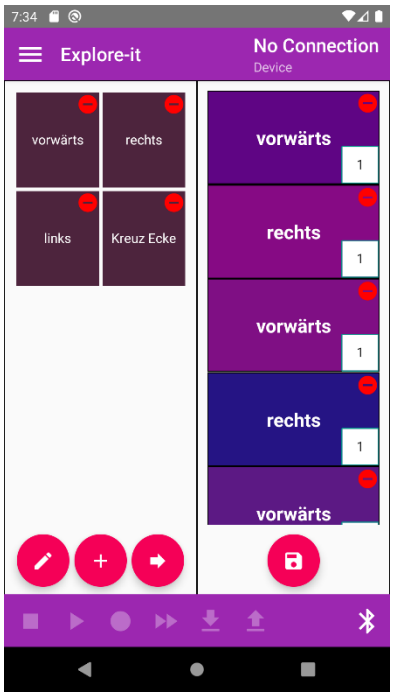


Figure 28 cross corner top

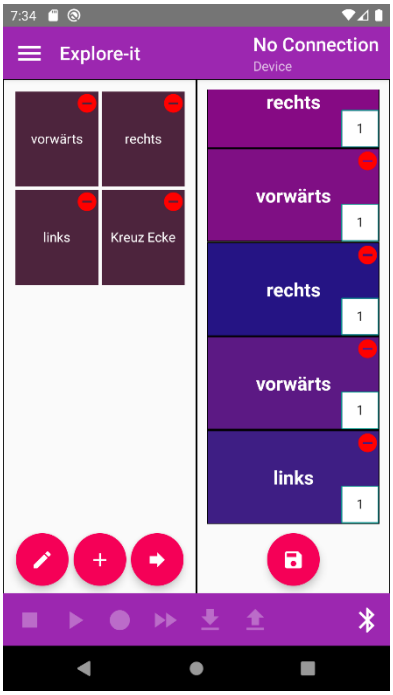


Figure 26 cross corner bottom

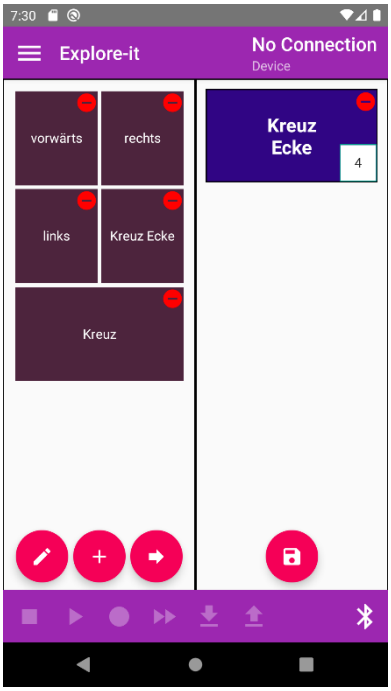


Figure 27 full cross

With the blocks created they can be added to the Programminglist as shown in “Figure 28 cross corner top” and “Figure 26 cross corner bottom”. It needs the following order: forward, right, forward, right, forward, left.

After this the program can be saved as a block. We call the block “cross corner” as shown in “Figure 28 cross corner top” and “Figure 26 cross corner bottom”.

Finally the block “cross corner” can be added to the Programminglist and repeated 4 times as shown in “Figure 27 full cross”.

The program builds now a full cross. It can be uploaded to the robot and be saved as a block for later use.

4.4 Programming implementation

This chapter explains the implementation for the programming for a block sequence. It describes the logic behind the Programminglist after the list is filled with blocks.

As soon as the upload button is pressed the “programming” function creates an array with the speeds which is send to the robot. The same array is created when the program is saved as a block.

The creation of the array works like this: Each block consists of an array of speeds. These arrays are put together to create a new big array. The order is the same like the index order in the “draggable-flatlist”(Programminglist). The number of repeats is used to repeat the arrays from the blocks within the correct order.

comment: At the moment of the delivery the programming did not work correctly.

4.5 Storing

This Chapter describes the storing of the Speeds, the blocks and the program.

To store blocks, we use the functions of the existing SpeedStore.js as template. It uses AsyncStorage from react-native.

With SpeedStore.js we created three new Javascript files called. BlockStore.js, BlockSpeedStore.js and ProgramStore.js. Each of the files has its own functions to save the data. This gives us a good subdivision for saving speeds, blocks and the program.

For the Database the app use “Realm”. The Scheme and the Actions were adopted from the existing Code.

comment: At the moment of the delivery the storing did not work correctly.

4.6 Extern RN Libraries

This chapter consist of a list of the used RN Libraries

Draggable-flatlist: https://www.npmjs.com/package/react-native-draggable-flatlist

FAB: https://www.npmjs.com/package/react-native-fab

Navigation-stack: https://reactnavigation.org/docs/en/stack-navigator.html

The RN Libraries that were used during the development but later removed

Draggable: https://www.npmjs.com/package/react-native-draggable

4.7 Possible Improvements and Extensions

This chapter explains the possible improvements for the FAB Buttons, the Drag and Drop function and the autogenerated Icons.

4.7.1 Disable for “FAB” Buttons

The “FAB” Buttons are bugged in react-native version 0.60.5. The “FAB” Buttons are not clickable after they were disabled once. They are not usable till the end of their lifecycle. This bug appears also in the existing part of the application.

We tested the “FAB” buttons in react-native version 0.61.5. In this version the bug disappeared.

One way to solve this problem is to render a new “FAB” Button after it was disabled. The Other solution is to upgrade the react-native version.

For this project we omitted the disable function, because it is not necessary on a functional level. An upgrade to version 0.61.5 is not possible because the Bluetooth Low Energy module is only working with version 0.60.5.

4.7.2 Drag and Drop function

As already described in chapter “Blocklist and Programminglist”. The Drag and Drop function does not work as intended. An Improvement would be to create a component or a library which can handle drag and drop between two lists. This would be more intuitive for the filling of the Programminglist.

4.7.3 Autogenerated Icons

As described in the chapter UI Design the blocks could have automatic generated icons. They would include the speeds inside the rows. The advantage would be to see the functionality of the block without open them. The disadvantage is for bigger blocks with a lot of rows of speeds. It would be hard to tell what they were built for.

5 Review and Reflection

We think we managed to deliver a user interface that addresses the specific needs we intended to meet, namely the needs of 12-year-old children when steering the Explore IT robot. Of course, there is place for improvements as already mentioned.

Below is an example for programming the same sequence of steps using both approaches:

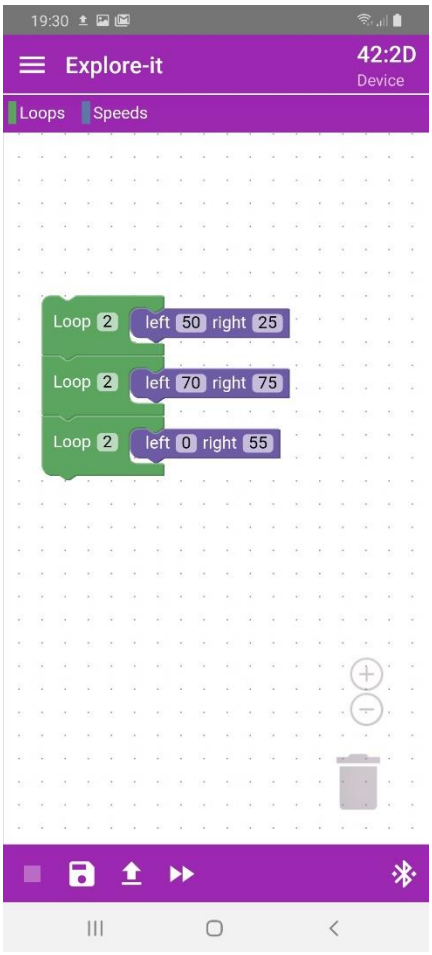


Figure 29 Sequence Example Blockly Based Approach

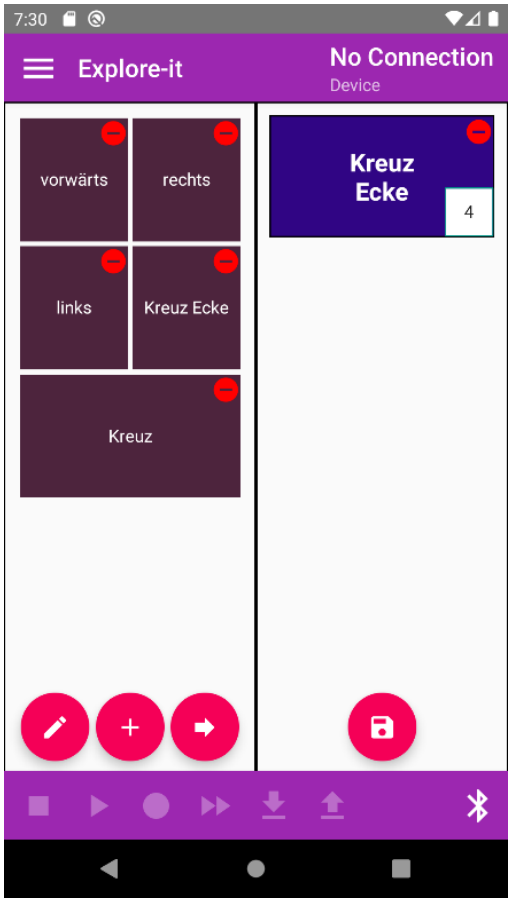


Figure 30 Sequence Example Trainly Approach

See a screenshot of the xml representing the Blockly workspace for this example in appendix 8.

See a screenshot with the code generated by Blockly for this example sequence in Appendix 9.

This project has turned out to be challenging in many ways. The implementation stage has presented its own problems to be overcome.

An area which has proved to be straightforward is finding information on different React Native libraries. However, the diversity of the RN libraries is very large, and in

some cases, there are a number of libraries providing similar or same functionalities. Some of the RN modules are not rated well and some don't provide all the described functionality. For example, we locked the landscape orientation for the Blockly screen using the following module `"react-native-orientation-locker": "^1.1.7"`. The landscape orientation was supposed to be unlocked when switching back to the first screen of the Robby App (`Orientation.unlockAllOrientations();`) but that didn't work. Later we found a comment on a forum complaining about the same issue.
Thus, in some cases, there have been too many library choices, on the Internet, requiring the identification of the most reliable libraries for the needs of the project.

We also need to ensure that the positive as well as negative lessons learned during the project are not forgotten. The most valuable lesson is the value of trying out different implementation versions on different branches. This forced us to review old code and notice where things could be done better.

Separating the project implementation into 2 distinct versions came as a surprise but it was the best decision from the supervisor's side in terms of making the most of the available project time.

6 References

6.1 Pictures

6.2 Literature

Nader Dabit (2019), React Native in Action, *Developing iOS and Android applications with JavaScript*, Shelter Island, NY: Manning Publications Co, available as pdf

6.3 Internet

ExploreIT Org (2019), Material Pakete Robotik1, [online], available at https://www.explore-it.org/de/lernen/lernlaesse/robotik-1

Google Play, Serial Bluetooth Terminal (2020), Serial Bluetooth Terminal [online], [Viewed 19 January 2020], available at https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en

Wikipedia (2019), Differential Wheeled Robot, [online], [Viewed 15 December 2019], available at https://en.wikipedia.org/wiki/Differential_wheeled_robot

Google Developers. Blockly (2019), [online], [Viewed 15 December 2019], available at https://developers.google.com/blockly

Blog WebView The Bridge that connects React Native with the Web, published, 1 May 2019, available at https://blog.bitsrc.io/webview-the-bridge-that-connects-react-native-with-the-web-95a0d5aaa31a

https://developers.google.com/blockly

React Native WebView Getting Started Guide, https://github.com/react-native-community/react-native-webview/blob/master/docs/Getting-Started.md

6.4 API References

React Native Web View API Reference

https://github.com/react-native-community/react-native-webview/blob/master/docs/Reference.md#javascriptenabled

https://github.com/react-native-community/react-native-webview/blob/master/docs/Reference.md

https://github.com/react-native-community/react-native-webview/blob/master/docs/Guide.md

React Native WebView Guide, 2019, Communicating with JS, https://github.com/react-native-community/react-native-webview/blob/master/docs/Guide.md#communicating-between-js-and-native

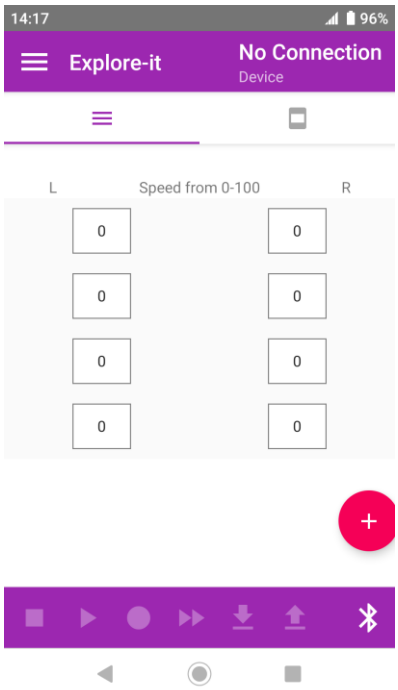
react-native-svg, https://github.com/react-native-community/react-native-svg

React Native Vector Icons, oblador/react-native-vector-icons, https://github.com/oblador/react-native-vector-icons

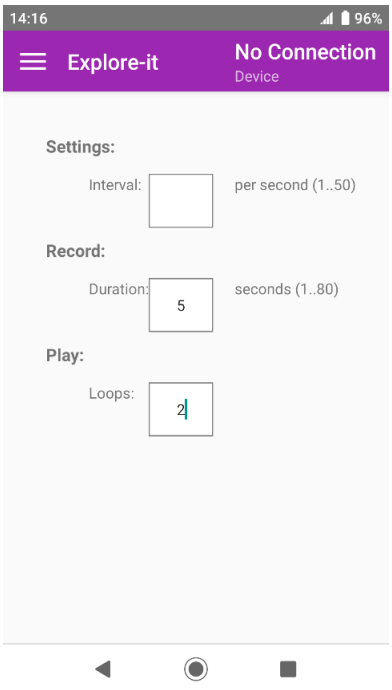
7 Appendices

7.1 Appendix 1

Programming Screen



Settings Screen



7.2 Appendix 2.

Debugging with the RN Debugger. Chrome Developer Tools.

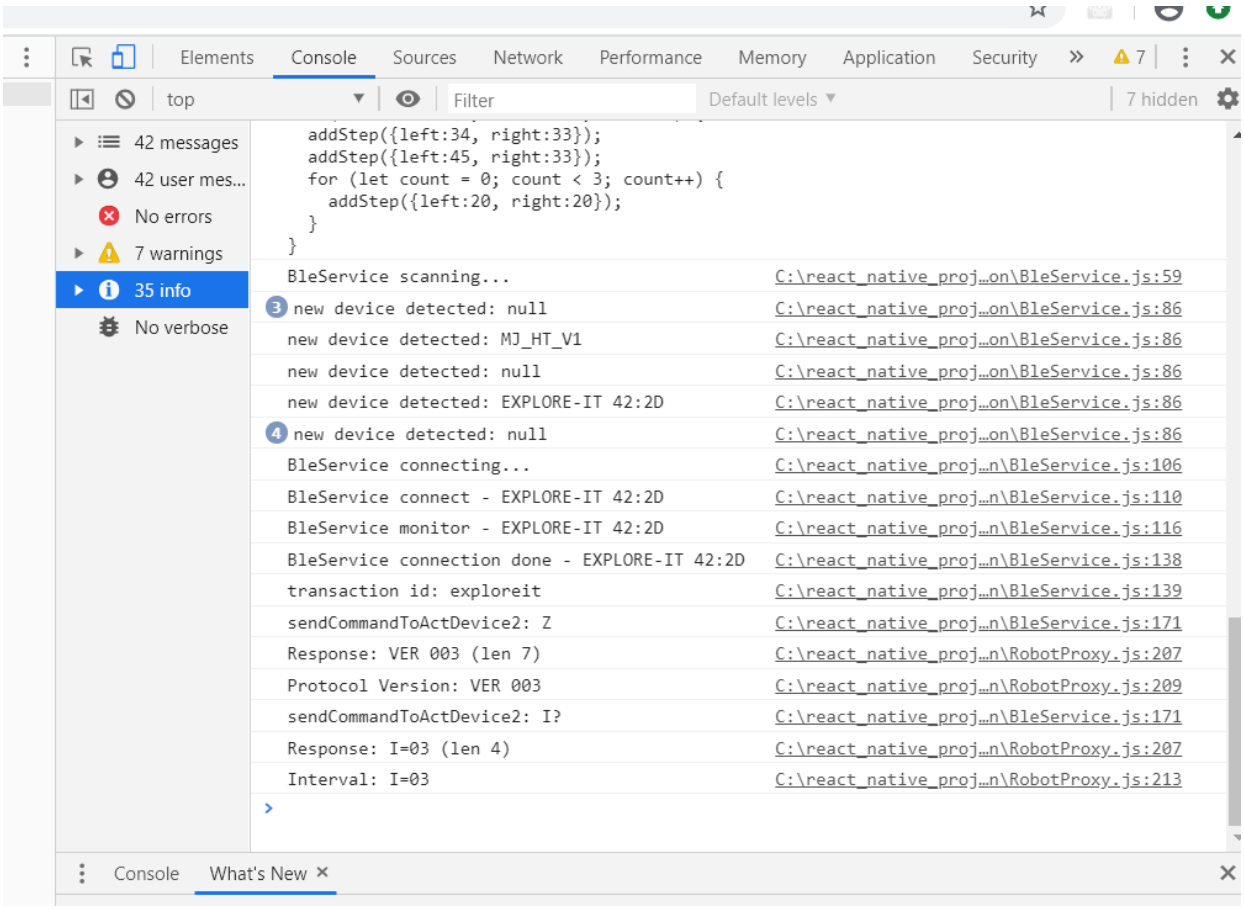


Figure 31 Debugging with the RN Debugger. Chrome Developer Tools.

Figure 32 Connecting to the robot

7.3 Appendix 3

Adding WebView to the project

1.Add react-native-webview to your dependencies

```
$ yarn add react-native-webview
```

(or)

For npm use

```
$ npm install --save react-native-webview
```

2. Link native dependencies

From react-native 0.60 autolinking will take care of the link step but don't forget to run `pod install` React Native modules that include native Objective-C, Swift, Java, or Kotlin code have to be "linked" so that the compiler knows to include them in the app.

```
$ react-native link react-native-webview
```

iOS:

If using cocoapods in the `ios/` directory run

```
$ pod install
```

NOTE: If you ever need to uninstall React Native WebView, run `react-native unlink react-native-webview` to unlink it.

3. Import the webview into your component

```
import { WebView } from 'react-native-webview';
```

Adding the web app to the RN project:

1. Copy the static html and its resources to be loaded in the web view to the following paths:

   Android: <react-native-project>/android/app/src/main/assets/

   iOS: <react-native-project>/ios/<new group as folder>/ (**Note**: “New group as folder” will ensure all its contents are bundled into the IPA file.)

2. WebView should have the uri prop for iOS and Android as shown below:

```
const isAndroid = Platform.OS==='android'..... ....
```

```
const blocklywebapp = {
  link: isAndroid ? 'file:///android_asset/blocksv/index.html'
    : './blocksv/index.html'
};
```

We needed to add the assets of the Blockly web app along with the html in the native resource directory for Android as explained below:

Android

Create assets folder and add the blocks to the assets folder, declare a constant for the link to the URI.

```
const blocklywebapp = {
  link: isAndroid ? 'file:///android_asset/blocksv/index.html'
    : './blocksv/index.html'
};
```

Use the constant to reference the assets as below:

```
<WebView source={{ uri: blocklywebapp.link }} allowFileAccess={true}
```

How to do it for IOS

Go to xcode-> main app folder -> add new group with folder called ‘external’ -> add the blocks web app files

The code for external web app

<WebView source={{uri:'./external/blocks/index.html'}} scalesPageToFit/>

7.4 Appendix 4

Blocks Version 1. Available Blocks: Loops (repeat x times, do)
Move forward x steps. Change right/left speed by a value
Set left/right speed to x (dropdown with the list of variables: left and right speed)
Turn right with the left wheel speed of 18, right wheel speed is 0,
Turn left - with the right wheel speed of 18,

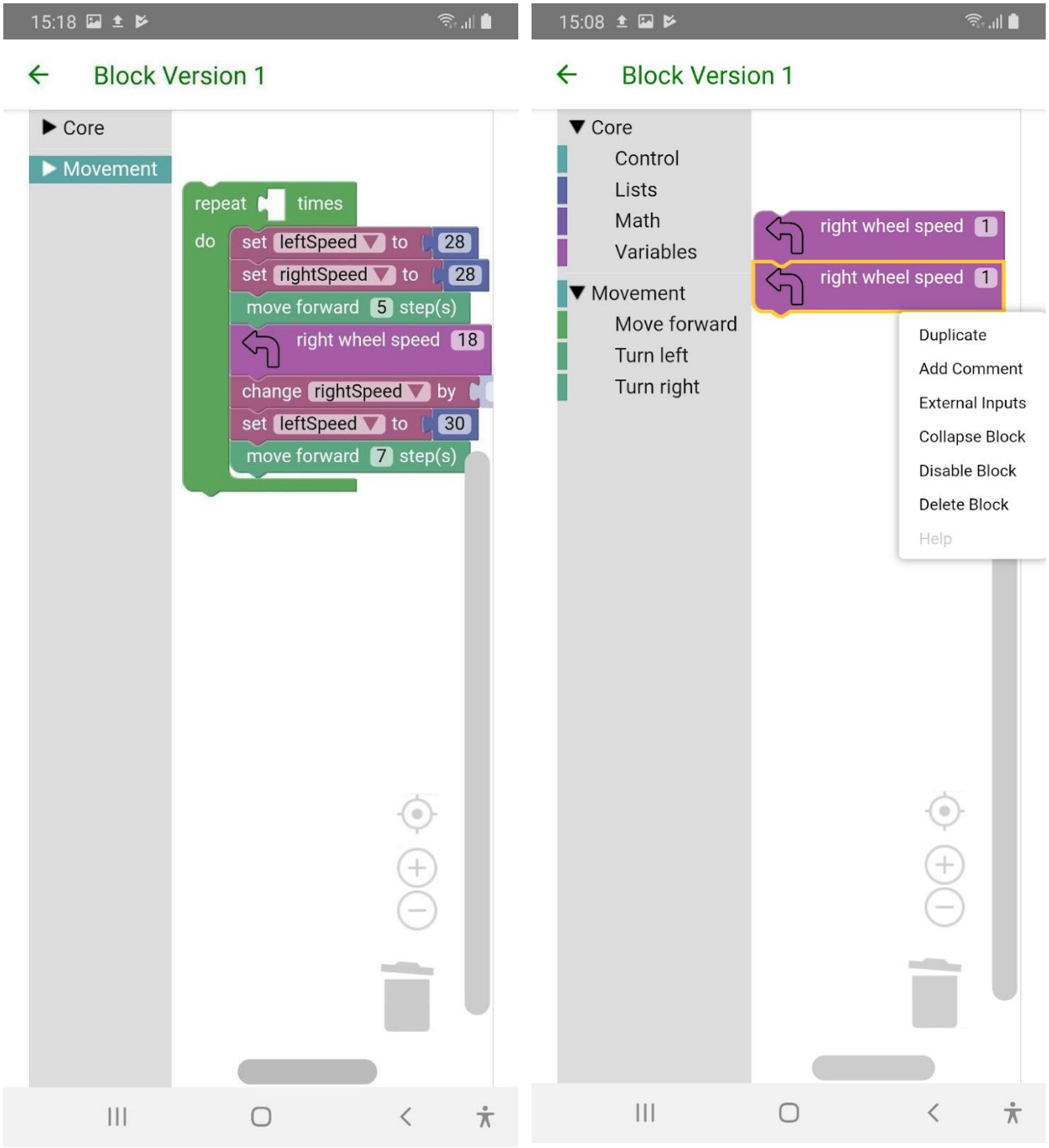


Figure 33 Blocks Version1 Sequence Example

7.5 Appendix 5

Blocks Version 2. Available Blocks:

Move forward & Set left and right wheel speeds.

Turn left. Turn right

Wait x second(s)

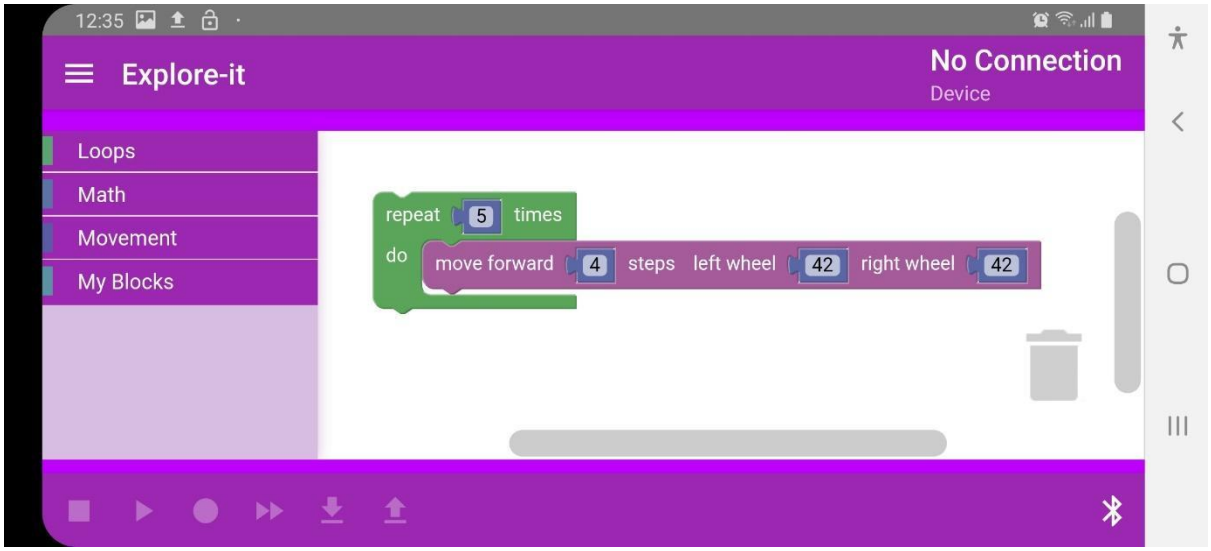


Figure 35 Blocks Version 2

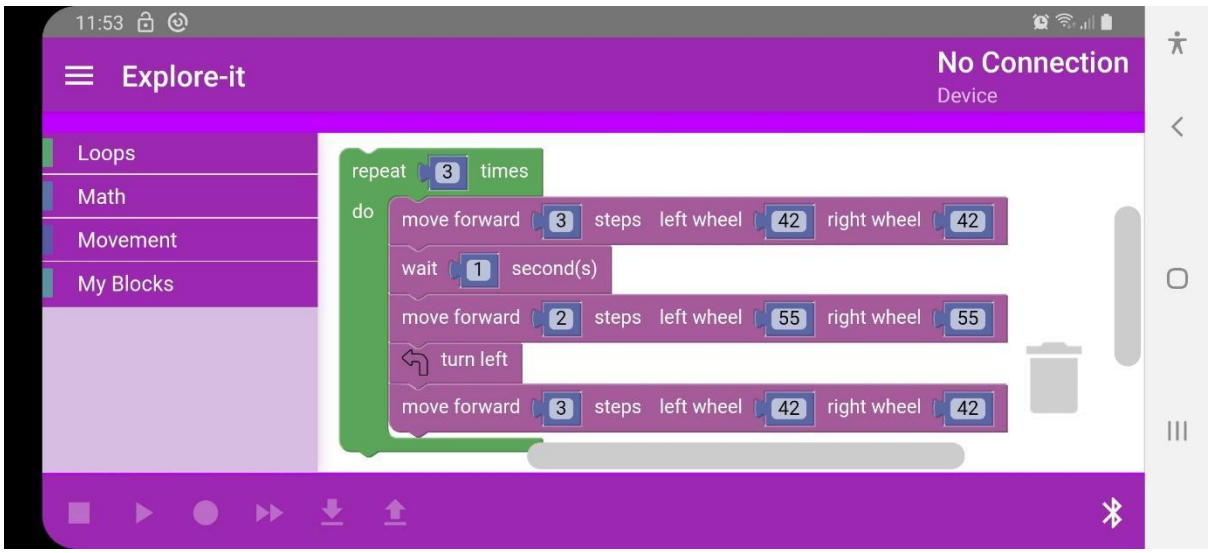


Figure 34 Blocks Version 2. Sample sequence

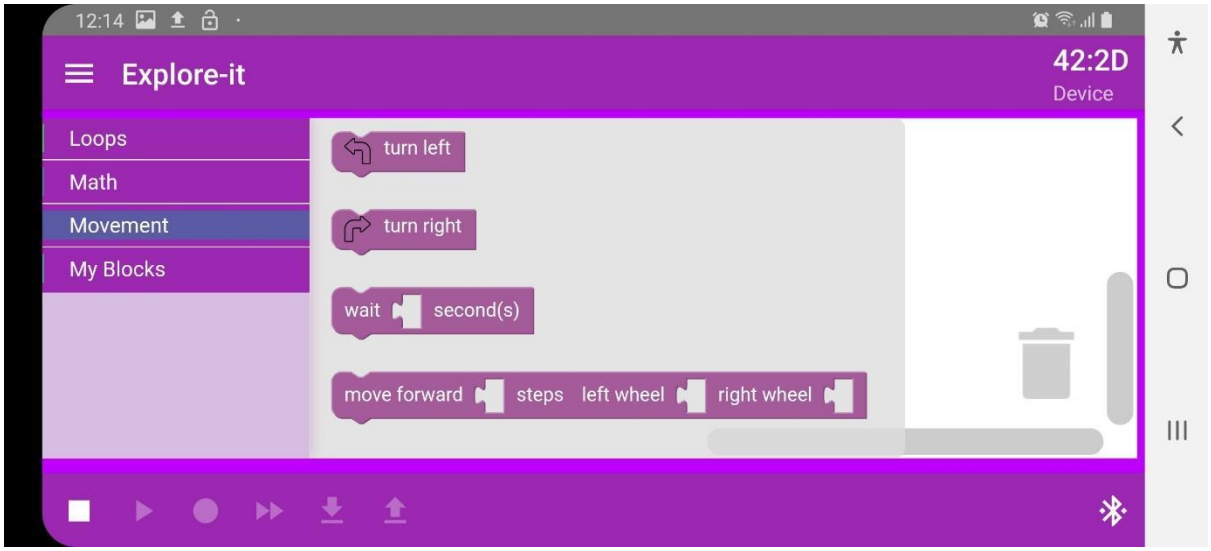


Figure 36 Blocks Version2 Movement Blocks

7.6 Appendix 6

Robby App Navigation

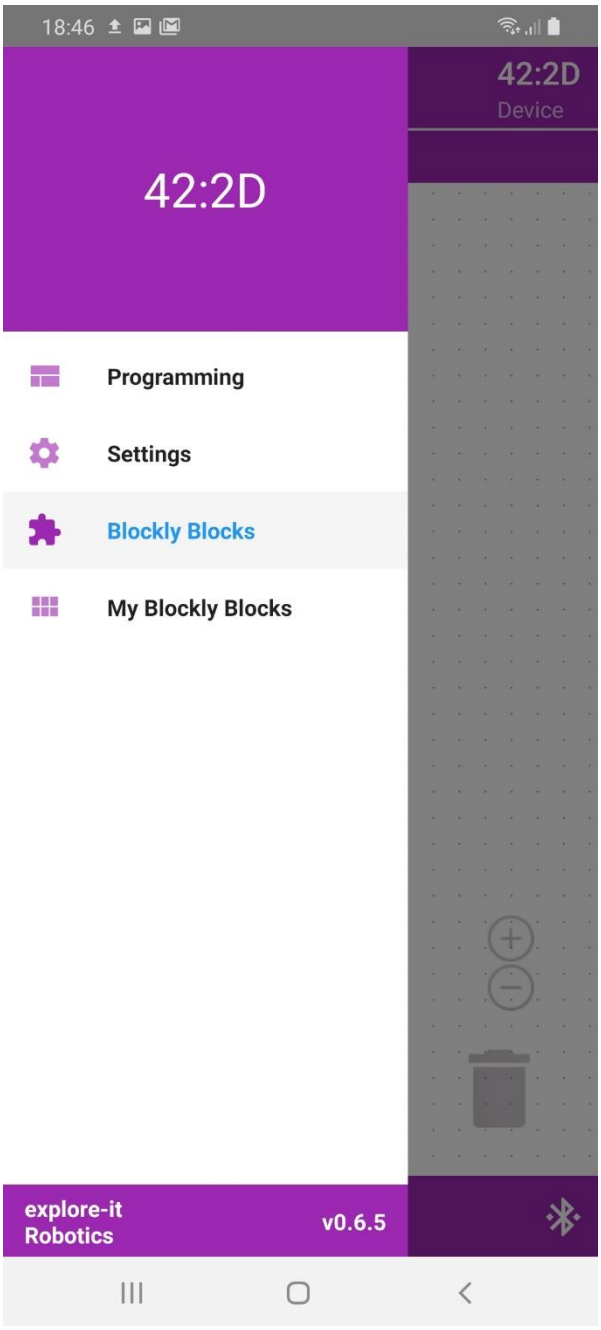


Figure 37 Drawer Navigation

7.7 Appendix 7

BlocklyWebView.js

```
import React, { Component } from 'react'
```

```
import { View, Dimensions, ActivityIndicator, StyleSheet } from 'rea
ct-native';
import { WebView, LOAD_NO_CACHE, LOAD_CACHE_ONLY } from 'react-
native-webview';

const isAndroid = Platform.OS === 'android'
const window = Dimensions.get("window");
const LoadingIndicatorView = () => (
  <ActivityIndicator
    color="#009b88"
    size="large"
    style={styles.ActivityIndicatorStyle}
  />
);
const blocklywebapp = {
  link: isAndroid ? 'file:///android_asset/blocksv/index.html'
    : './blocksv/index.html'
};

export default class BlocklyWebView extends React.Component {

  componentDidMount() {

  }

  render() {
    const { block_xml, receiveCodeAsString } = this.props;
    const runFirst = block_xml.length > 2
      ?
      `window.isNativeApp = true;
    `
      :
      `window.isNativeApp = true;
    window.onload = function(block_xml) {
    Blockly.mainWorkspace.clear();
    let textToDom = Blockly.Xml.textToDom(block_xm});
    Blockly.Xml.domToWorkspace(textToDom, Blockly.mainWorkspace);
  }
  `;


    return (
```

```
      <View style={styles.container}>
        <WebView source={{ uri: blocklywebapp.link }} allowFileAcces
s={true}
          ref={r => (this.webref = r)}
          style={{ marginBottom: 30 }} textZoom={100}
          scalesPageToFit={true}
          renderLoading={LoadingIndicatorView}
          startInLoadingState={true}
          cacheMode={LOAD_CACHE_ONLY}
          javaScriptEnabledAndroid={true}
          injectedJavaScript={runFirst}
          onMessage={event => {
            const { data } = event.nativeEvent;
            { receiveCodeAsString(data) };
            console.log("code from the web app :" + data);
          }}
        />
      </View>
    )
  }
}

const styles = StyleSheet.create({
  ActivityIndicatorStyle: {
    flex: 1,
    flexDirection: 'row',
    justifyContent: 'center',

  },
  container: {
    flex: 1,
    flexDirection: 'row',
    backgroundColor: '#fff',
    alignItems: 'stretch',
    justifyContent: 'center'
  },
  text: {
    color: '#fff',
    textAlign: 'center'
  },

});
```

7.8 Appendix 8

XML code for the spped sequence given in the example

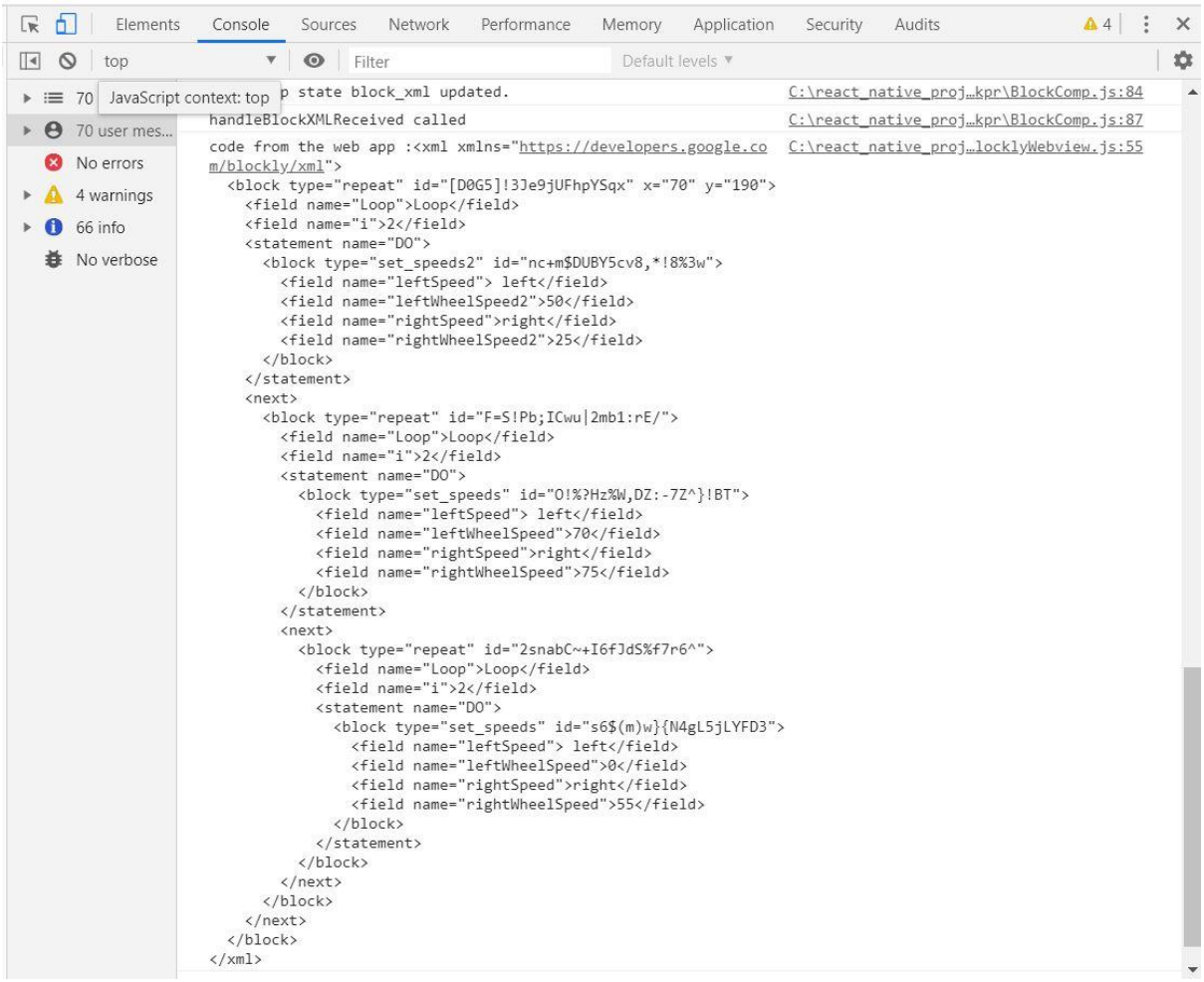


Figure 38 XML - the Blockly content

7.9 Appendix 9

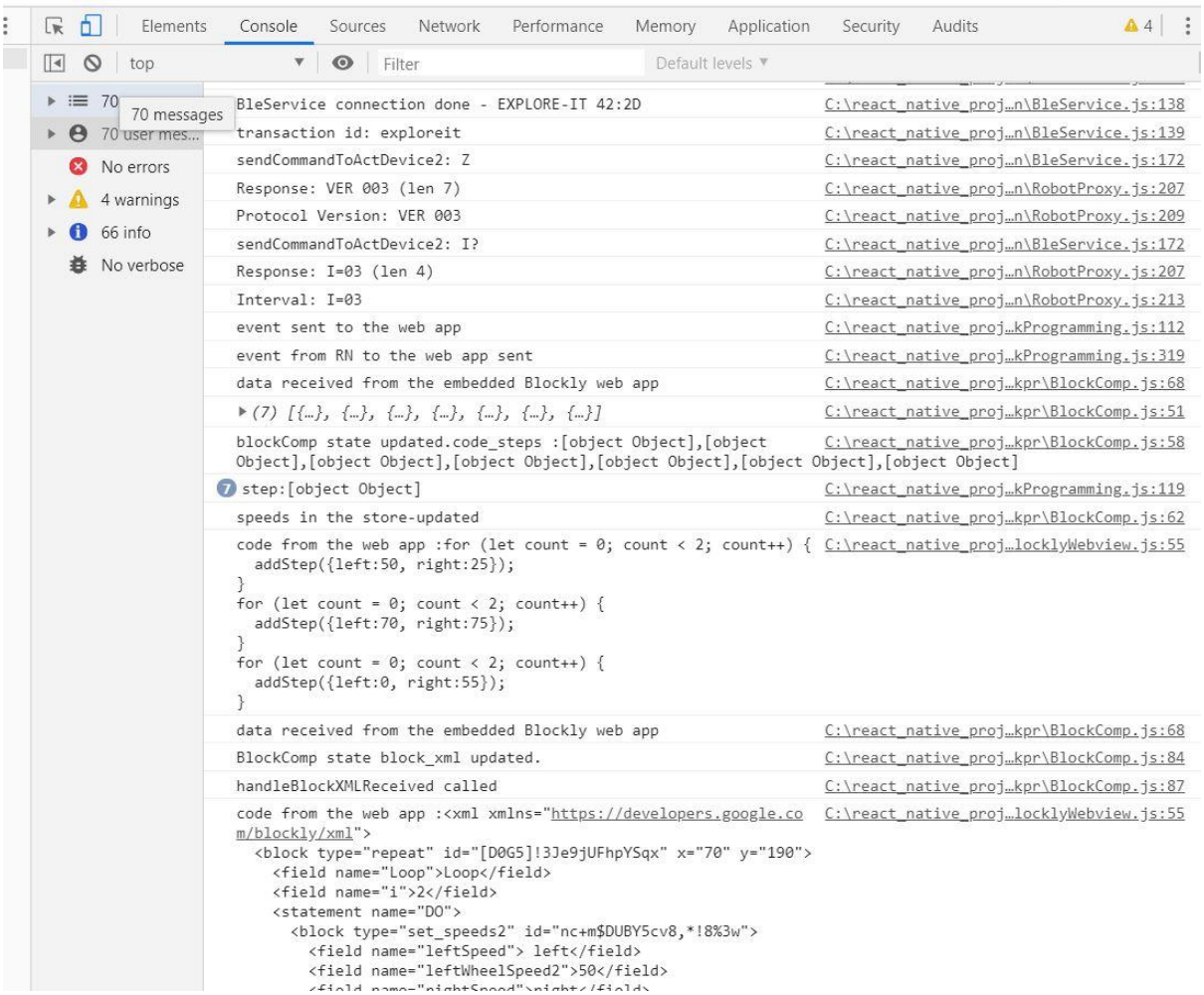


Figure 39 Code generated by Blockly as we defined it

We confirm that this is our own work and we have documented all the sources and material used.

Brugg, 20 January 2020

Sabina Borbely

Roman Ott
