CHALMERS

WiFi controlled cars in an obstacle course, a cooperative exercise in problem solving

Bachelor's thesis in Computer Science and Engineering

CARL-HENRIK HULT
JOAKIM SCHMIDT

Department of computer science and engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

DEGREE PROJECT REPORT

WiFi controlled cars in an obstacle course, a cooperative exercise in problem solving

CARL-HENRIK HULT
JOAKIM SCHMIDT

WiFi controlled cars in an obstacle course,
a cooperative exercise in problem solving
CARL-HENRIK HULT
JOAKIM SCHMIDT

Examiner: Peter Lundin

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Gothenburg 2016

WiFi controlled cars in an obstacle course,
a cooperative exercise in problem solving

CARL-HENRIK HULT
JOAKIM SCHMIDT
Department of Computer Science and Engineering,
Chalmers University of Technology

Degree project report

Abstract

This report describes an idea of how to enhance classic toys with concepts borrowed from video games, and how to utilize smartphones in order to construct a cooperative game with WiFi controlled cars in an obstacle course. This report focuses heavily on the implementation details of the different parts in order to provide a guide for anyone interested in further development of the idea. Because this project is about creating a prototype, many of the implementation specifics are deliberately simple, such as the cars having no body, or the lack of focus on aesthetics in the controller application. The Raspberry Pi computer was used as the controlling unit on-board the cars, because of their small size and their ease of use. In the end, three cars, two small and one larger, were constructed. Each car can be controlled from an Android application over WiFi, and it is possible to switch back and forth between them. As a conceptual part of an obstacle course, an electric door, which activates when a number of different activating mechanisms are triggered, was also constructed. In order to help facilitate development and construction of diverse obstacle courses, we also wrote a randomized map generation program.

Keywords: WiFi, cars, gameplay

Sammanfattning

Denna rapporten beskriver en idé för hur man kan kombinera vissa idéer från datorspel med klassiska radiostyrda bilar och moderna smarta telefoner för att skapa en samarbetsövning i spelformat. Spelarna samverkar för att köra tre stycken WiFi-kontrollerade bilar genom en labyrintliknande hinderbana. Rapporten fokuserar mycket på implementationen av de olika delarna i spelet, och tanken är att den skall kunna användas som en sorts guide för någon som är intresserad av att fortsätta utvecklingen. Projektet som genomförts handlade till stor del om att ta fram en prototyp, och vissa delar är därför medvetet simpla och detaljfattiga, som exempelvis bilarnas brist på kaross. Bilarna styrs genom varsin Raspberry Pi-dator, som sitter monterade på dem. Raspberry Pi valdes dels för sin lilla storlek, och dels för att de är enkla att arbeta med. Projektet resulterade i tre stycken bilar i olika storlekar, som alla kan styras genom en androidapplikation, över WiFi. En konceptuell hinderbana utvecklades och byggdes också. Banan har väggar i kartong, och även ett hinder i form av en elektrisk dörr, som bara kan öppnas genom att aktivera ett antal olika tillkopplade aktiveringsmekanismer. Ett datorprogram som kan generera slumpmässiga kartor utvecklades också, i hopp om att öka omspelbarheten och variationen i spelet.

Rapporten är skriven på engelska

Nyckelord: WiFi, bilar, spelmekanik

Acknowledgments

We want to thank Mr. Sakib Sistek who, as our supervisor, has helped us throughout the project with his never ending enthusiasm and willingness to help.

Acronyms

DFS Depth-First Search. 10–12

DRV8835 Pololu DRV8835 Dual Motor Driver Carrier. 7, 8, 14, 15

GPIO general purpose input/output. 5, 7, 10

PTF Path To Finish. 11

PWM pulse-width modulation. 4, 5, 8, 14, 15

RPi Raspberry Pi 3 Model B. 4, 5, 7, 8, 14, 15

RPi Camera Raspberry Pi Camera Module. 5

VC Visit Complete. 11

Innehåll

1 Introduction

1.1 Background

We wanted to create an engaging cooperation exercise for training logical thinking and problem solving, through the combination of some aspects of classic toys and modern video games.

1.2 Goal

The goal is to create an interactive experience in which well known elements, like RC cars, will be used to produce a fun and engaging cooperation exercise.

1.2.1 The game

In the game, three cars, one large and two smaller, must cooperate to solve puzzles in order to advance through an obstacle course. The cars will be quite distinct, with their own unique abilities:

- Cam-car is a small car equipped with a camera, which can stream video to the controller.
- Grab-car is a small car equipped with an electromagnet, with which it can grab certain items.
- The big car is much larger than the other two, and has a small turret with a laser pointer, which it can use to shoot certain targets.

The puzzles consists of various obstacles, such as closed doors, which can only be opened or activated if their corresponding activators, such as buttons, are operated in the correct order. The map itself is a series of walls, guiding the cars forward.

1.2.2 Map creation

To help facilitate interesting and clever maps, a map generation program will also be created. The map generator will have an algorithm that can create new maps that are guaranteed to be solvable. These will then be shown in a graphical interface to help with construction of the map.

1.3 Purpose

Our purpose is to show how some aspects from video games, such as explicit rules and gameplay feedback, can be used to modernize and enhance classic toys. We will also utilize the nowadays very common smartphone, and demonstrate its ability to manage tasks previously handled by specialized hardware.

1.4 Limitations

To limit the project we have decided to create the prototypes with less focus on the aesthetics and more focus on the functionality of the modules, cars and map generator. We have also purposefully limited the scope of the software development in order to have more time to work on the hardware and construction.

2 Method

Since this project is quite large and broad, we have decided to split it into four main parts. We hope that this will help us tackle each part and its problems separately, which should be easier than everything at once.

2.1 Controller

The controller will be one of our main focus points at the start of the project. Our hope is to get all the basic functionality, including WiFi, in place early on, so we can quickly move on to other parts.

2.2 Cars

After the controller has basic functionality, we will focus our efforts on mounting Raspberry Pi cards on the cars, and program the software needed to control the cars with the Raspberry Pis and, by extension, the controller.

2.3 Map

This part is about developing a generalized way to build different modules for the puzzles of the map. These modules can be quite diverse, from doors to elevators, and require a common ground in both hardware and software, so that any future models are easy to build and program. We will also build a set of simple walls from cardboard. These walls do not move or interact with the cars in any way, and exist solely to prevent cheating from the player.

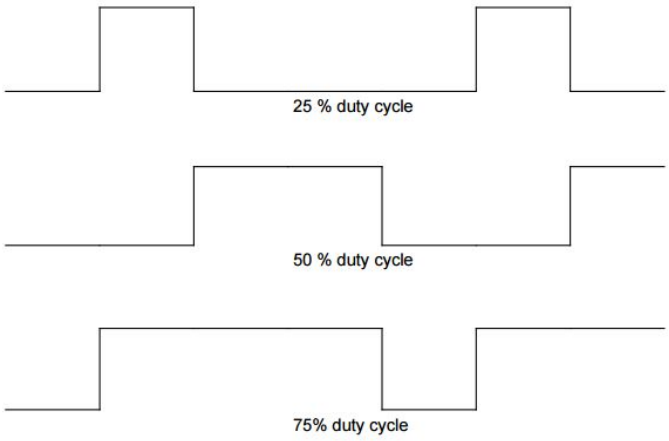
2.4 Map generator

The map generator is pure software, and can as such be worked on at any time during the project. We have some time set aside for it during the later stages of the project, but we hope to do the bulk of the programming during lulls in the other parts.

3 Technical background

3.1 PWM and duty cycles

The motors used in this project are simple DC-motors. They, along with the servos, have no concept of, and give no feedback about, their current position or velocity. Because of this, there is no way to simply tell a DC-motor to run at, for example, 50% of maximum speed clockwise. Pulse-width modulation (PWM) is a technique used to supply the motors with power in such a way as to simulate different velocities. By making use of the fact that motors cannot accelerate to maximum velocity instantly, it is possible to control them by rapidly turning the power on and off. A duty cycle is a time period, during which the signal shifts between low (no pulse) and high (pulse)[1]. As long as the signal is high, the motor is powered and accelerates. When the signal shifts to low, the power to the motor is cut, and it immediately starts to decelerate. By setting a high enough frequency on the duty cycle, for example 50 Hz, and then varying the width of the pulse, it is possible to very finely control the motors velocity. This technique is also used to control the internal motor in the servos. In essence, a duty cycle is a percentage of the maximum velocity of the motor. With the help of a programming library, this can be abstracted to a simple interface, with which one can tell the motor driver to run the motor at, for example, 50% of maximum speed clockwise. In this project, the WiringPi2-Python library[2] was used to achieve this abstraction.



Figur 1: A visual representation of duty cycles

3.2 Raspberry Pi 3 Model B

The Raspberry Pi 3 Model B (RPi) is a single-board computer developed by the Raspberry Pi Foundation[3]. It is the first of the RPi computers to feature integrated Bluetooth and WiFi. Like all the other Raspberry Pi models, it has most of the features that a normal PC has, including USB ports and HDMI for monitor connection, as well as a specialized connector for the Raspberry Pi Camera Module. With a 4GB micro-SDHC card, it is possible to install one of many different Linux distributions that support the RPi. For this project, the Arch Linux ARM[4] distribution is used, mostly because it comes without any bloatware, not even a window manager. The Arch Linux package manager, pacman, is also very good, both in terms of speed and at handling dependencies.

Raspberry Pi Camera Module The Raspberry Pi Camera Module (RPi Camera) is the official camera for the Raspberry Pi, featuring a 5-megapixel camera and both picture and video capabilities. It connects to the RPi via a special connector, and is designed to be very easy to use.

3.3 Voltage regulation

Many electrical components require a very specific input voltage to operate normally. Failure to supply the specific voltage can result in malfunction or even damage to the component. A voltage regulator can help with supplying the desired voltage, by transforming some other input voltage, which may be higher or lower than the output voltage. There are many kinds of voltage regulators, and they operate by many different techniques. The ones used in this paper are switching buck-type regulators, and are capable of transforming an input between 6V and 38V down to 5V. Buck regulators rely heavily on the law of inductance, and are explained in detail here [5]. A simplified explanation is provided below for convenience.

The first component in a buck regulator is a transistor, which switches on and off rapidly. When it is on, or closed, both the input and output voltage is connected to an inductor. This causes a current through the inductor. During this time, a capacitor, connected in parallel to the output, is charged. When the transistor is off, or open, the input voltage is no longer connected to the inductor, which would cause no current to flow through it. However, in accordance with the laws of inductance, the current in the inductor can not change instantly. Since the resistance on the output is constant, and the current does not disappear, Ohm's law forces the voltage over the output to be non-zero. During this time, the capacitor discharges, to help drive the current through the output up. The transistor switches on and off so quickly that the output voltage is, in practice, stable.

3.4 DRV8835 Dual Motor Driver Carrier

This component provides bidirectional control of two different motors. One side supports input signals and the other drives the motors. On the driver side are connectors for connecting a power supply to the motors. The motors are connected in parallel, so whatever voltage is supplied on the input will also be connected to the motors, meaning that care has to be taken to ensure that both motors can handle the voltage. The signal input side has its own power connectors, and requires a small enough current that the RPi can run it with its general purpose input/output (GPIO)-pins. There are two connectors for each motor, with mirroring connectors on the output side. One of the input connectors control the motors speed by receiving a PWM signal, and the other controls the rotational direction by receiving either a high or a low signal. For a more in depth description, see [6].

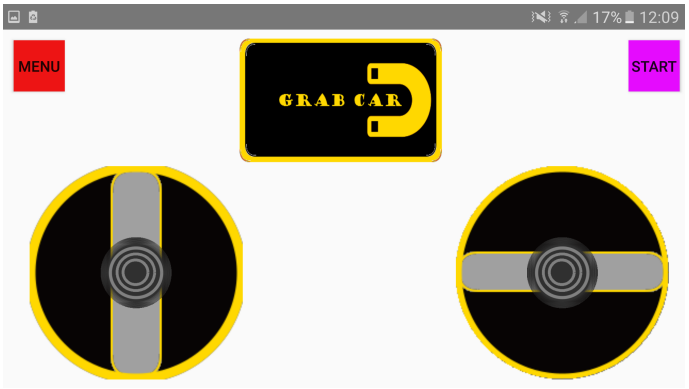
4 Implementation

The schematics for all hardware in this section can be found in Appendix B.

4.1 Controller

Basic design of the interface The controller was written for Android in Java. A basic graphical interface for swapping between the cars was first created. The interface contained buttons for the different cars. The design of the different cars' controllers were close to identical, apart from the name which were shown with a different graphic for each car. To do the dynamic changing of graphical components, Android provides a feature called Fragments. A Fragment can contain many different parts, like buttons and images, and by switching between Fragments, different parts of the application can be made visible to the user. The standard way of switching between Fragments is quite awkward, so one of the very first features implemented was an abstraction of this procedure, to streamline the procedure for future use. Different Fragments for each car, and for the game menu, were then designed and created.

Joysticks On the screen are two joysticks (see figure 2). Each joystick was built the same, with the key difference of what direction they could move. It consisted of a static background that acted as the base of the joystick. The joysticks could only move within the area of their respective backgrounds. The actual stick that you move is being drawn on screen dynamically as you move it around. Its default position is in the middle of the underlay, much as a real joystick would snap back to the middle. When the joystick is moved from the middle, a value is calculated. That value represents the distance from the middle that the joystick currently is positioned. To differ from left, right, up and down the values were negative on one side, and positive on the other.



Figur 2: The controller with joysticks

Implementing WiFi In order to control the cars, the controller should be able to send data over WiFi. Some attempts to use Bluetooth instead were made, but were soon scrapped due to WiFi being much simpler to implement. Java has built-in sockets that can be used to send data streams over WiFi. In order to make sure the communication, or any lag in the connection, does not interfere with the rest of the application, all WiFi code is run on a parallel thread. By using what java calls a Deque — a circular array — as a buffer for the data coming from the joysticks, the WiFi-handler is almost completely decoupled from the rest of the application, and does not in any way interfere with its execution. In other words, the joysticks input their data in one end of the queue, and the WiFi thread reads data from the other end. A simple protocol for sending different commands was conceived and implemented (see Appendix A). With this protocol, data representing a value between 0 and 100 could be sent, along with a value that instructed the car how to use the data. Since the size of the data sent could vary, a single byte of data, telling the cars how much data is incoming, is always sent before each data packet.

Special abilities and their activation When the cars could be controlled from the application, the next step was initiated. The controller was supposed to have more capabilities, such as controlling the different special abilities of the cars. For the big car the controller had to be able to control two additional servos, in addition to the motor and the servo used for steering. This was not true for the small cars, since their special abilities do not use any extra servos or require any additional steering. So the controller had to behave differently depending on which car you were driving.

A simple solution for controlling the two additional servos on the big car was implemented. This used the same joysticks as for the steering of the car, but attached a flag to the data, signaling that the data is to be used for the special ability. This was easy to implement, but problems emerged. When driving the car, steering left and right, the joysticks would go back to their default position when released that made the cars stop. This behavior was desired when steering, but not when aiming the turret, where it is more desirable to have the joysticks stick to their last known position any time the player stops moving them. The joysticks were thus changed somewhat to allow for both types of movement.

4.2 Cars

4.2.1 Hardware

The decision to have three cars, slightly different from each other, was made before the project even started. As work progressed, new ideas for how they could differ kept cropping up, and while many were dismissed for various reasons, enough of them were worth keeping to change the cars into three quite distinct gameplay elements. An RPi[3] was mounted on each car in order to interface with the rest of the hardware, and provide WiFi access. Because the RPi can only output up to 50mA total on its GPIO pins[7], powering the motors and servos would require some additional hardware. The Pololu DRV8835 Dual Motor Driver Carrier (DRV8835)[6] is made specifically to drive motors, but can be used for servos as well. It is also very small and cheap, and would therefore be perfect to solve this problem.

The RPi does not have an on-board voltage regulator, and must therefore be supplied a steady input of 5V. In fixed installations this is not a problem, because it can be powered by a normal micro-USB cell phone charger. In RC cars that move around a lot however, it can become a problem. A lot of time was spent looking for suitable batteries, or powerbanks, that could power

the RPi without having to invest in voltage regulators. Since the RPi can draw up to about 1.2A, and up to 1.5A[8] with a camera installed as well, a powerbank with a maximum output current of about 2A was necessary. Due to the quite rigid size constraints on the small cars, this turned out to be rather hard, if not impossible, to find. In the end, it was deemed simpler to install a 9V battery and a voltage regulator for the RPi, and power the DRV8835 with two 1.5V AA batteries in series (for a total of 3V).

The DRV8835 was used to control both the motors and servos. At first it got tried out on a breadboard, to test it with the motor and servo of one of the small cars. The WiringPi2-Python library[2] was used to send signals from the RPi to the DRV8835. The WiringPi2-Python library was originally made for another version of this motor driver, that was specifically made for the RPi. Fortunately, after brief inspection of the code in the library, it was fairly easy to translate this and use it for the DRV8835. The important thing the library provided was an easy way of creating the PWM signals needed for the motor driver.

A simpler solution for the big car Because the big car has both motor and servos that should be supplied 5V, the issues that existed with the small cars were not as prevalent on the big car. The already present 9.6V battery would be capable of powering all hardware simultaneously, after being transformed down to 5V. The servos on the big car are also more advanced, with a third connector for PWM-signals, which means the RPi is able to take care of the steering directly.

What should be noted is that when using that library for the DRV8835 the modepin had to be handled in the code as well, since the library assumes you are using the driver made for the RPi. That driver has a different default setting than the one used in this project.

4.2.2 Software

The small cars and the big car has some key differences. The small cars use servos that only requires a fixed current flowing through it, with it being negative and positive depending on which way it should turn. This made the software for the small cars slightly different from the software installed in the big car, as the servos in the big car receives different values depending on how many degrees the servos should turn.

The code was divided into two main parts: one responsible for the network connections and one responsible for the car and its functionality. When an Android device connects to the car and sends, for example, a steering signal, that data is saved in the queue that the car reads from. It then interprets the data, and sends it to the right function. A steering signal would for example be sent to the turn-method of the code. Python's deque-class was used as the queue, in a way that mirrors that of the controller's.

To read this signal correctly the program had to know how much data it was supposed to read at every iteration, since the data varied in size. This was managed by first reading a single byte of data that contained information about how much data that the car should read. The data received consisted of the 16-bit register (Appendix A), where different bits meant different actions for the car. The code between the cars also differ in that they have different special abilities and they also have different kinds of feedback to send to the controller.

Cam-car The Cam-car had to be able to send video feedback to the controller when the camera becomes active. When the controller sent data with the special ability flag set, the Cam-car would activate the camera, and start sending video back to the phone. It can still be driven while doing this. When the controller sends data without the special ability flag set, it deactivates the camera.

The big car When activating the big car's ability, the car will stop and the controller's joysticks will instead control the turret. As long as the special ability is active, the car cannot be moved, but it is quick and easy to switch back and forth between controlling the turret and controlling the car.

4.3 Map

A map is a maze-like configuration of walls and puzzles. It consists of a 2-dimensional grid of cells, where each cell is $32 \cdot 32$ cm. The size was decided because the big car is $20 \cdot 30$ cm, and it is also easy to subdivide if that would ever become necessary. Since the aim was to make a working prototype, the walls were constructed out of cardboard and are fairly easy to mass produce, even by hand. Each wall is $32 \cdot 8 \cdot 4$ cm ($L \cdot H \cdot W$). A puzzle is a combination of a single module, and one or more activators. They were implemented to make the game more interesting and difficult.

4.3.1 Modules

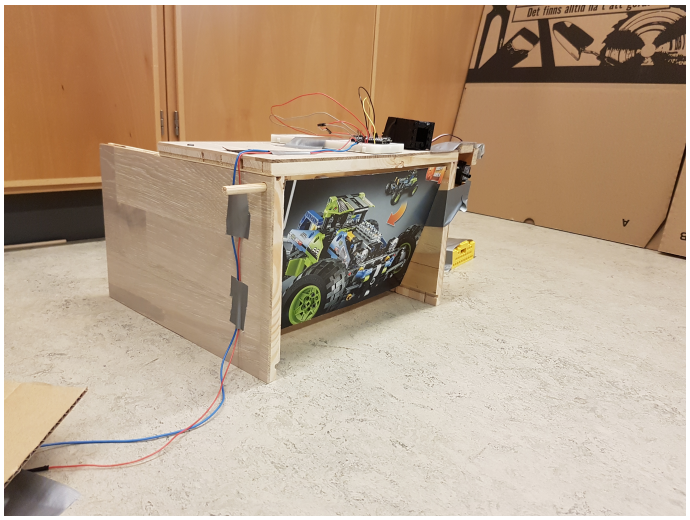
Modules are in most cases some sort of obstruction on the map, and prevents one or more cars to proceed. In order to get rid of the obstruction, the module's activators need to be reached and triggered.

Door Module The door module is a basic door, and was the only module constructed. It was made out of wood instead of cardboard, and is basically a tunnel, with a garage port-like door. The door is operated by a servo and an Arduino MICRO[9].

4.3.2 Activators

Activators are objects on the map, which the cars can interact with in some way, in order to trigger modules. They can be anything from buttons to targets for the big car's laser cannon, and can require either a single or continuous interfacing.

Button The button was built out of cardboard glued to an electrical switch. The cardboard was used to create a bigger button so that the small cars could open a door module by driving over the cardboard and subsequently press the button. When the button is pushed in, a signal goes to the connected module, that tells it that it has been activated.



Figur 3: The door module

Laser target The laser target is a photo resistor that triggers when a concentrated light source is shone directly on it, such as the laser from the big car. When the laser hits the laser target, the photo resistor's resistance lowers, the current increases, which leads to the micro controller registering it as a signal.

4.3.3 Software

The software in the modules is quite simple. Objects representing the activators are created and loaded into a list. Since all activators basically do the same thing (allows a current to flow through them) there is no need for specialized classes for the different types of activators. An activator can be set to require a continuous triggering in order to remain active, or it can be a one-time trigger. The list of activators is then traversed in a loop, every 20 ms, and checked whether they are active. If all activators are active, the module triggers. There is no software limitation to the amount of activators a module can have, it is only limited by the number of GPIO-pins the micro controller has.

4.4 Map generator

4.4.1 Generator algorithm

It was decided early on that the map generator should be written in Python. This was because it is quicker and easier to realize a working prototype in Python than in some other, lower-level languages, such as C, and there was really no need for a faster or more complex language. The map itself was early on visualized as a maze, and thus some time went into the research of different techniques with which mazes can be constructed. It was decided that the algorithm should implement a Depth-First Search (DFS)[10] with wall-removal, since it seemed to be the technique most easily extendable with special rules, which would be necessary for the algorithm to also place puzzles throughout the map.

Since the maze's corridors were intended to be quite narrow, the big car would soon run into problems navigating around turns and corners. To counter this problem, the generator was extended

with a new algorithm, Path To Finish (PTF), to find and create the shortest and straightest path from the starting point to the finish point. When generating a new map PTF would run first, and then DFS would complete the map, making all the other parts in the map reachable. Together the two algorithms created mazes that would suit the needs of the game, but the way they, and the rest of the map generator, were set up made it very complicated to place puzzles in the maze in such a way as to guarantee a solvable map.

The solution to this problem was the introduction of the concept of branches. A branch in this instance is a list of cells, where each cell is a neighbor to the one before and the one after it in the list (except for the first and last cells, which only have one neighbor each). The branch also has a list of other branches, which are connected to it via a single cell. Together, they make up a tree-like structure, in which all cells in the whole map are held. The DFS algorithm was remade into the Visit Complete (VC) algorithm, which is similar to DFS, but works with branches.

Creation and placement of puzzles Since the maps can become quite complex, but must always be solvable, puzzles need to be placed with care, and according to rules. A puzzle is defined as a single module, with one or more activators connected to it. One can quickly conclude that, for the puzzle to be solvable, an activator must never be placed behind the module it is connected to. Drawing on this simple conclusion, three rules were defined to guide the puzzle making process:

1. An activator may never be placed behind the module it is connected to.
2. An activator may never be placed in any of the branches that are located behind the module it is connected to.
3. For every branch found when checking rule 2, if there are other activators in those branches, a check for rule 2 must be run on the modules these other activators are connected to, and the new module may not be placed in any of those branches either.

Elevation levels In the hope of creating maps that always had exactly one solution, it was decided that the puzzle making algorithm should make sure that all three cars always had a certain place to be in the map. To help facilitate this behaviour, a level system was created. The level system splits certain cells into four equal parts, that may then be elevated. The elevation can be one of three defined heights, and any combination of these parts, within some limits, can be elevated. With this system, some branches are restricted to only allow the small cars, and they effectively become two branches, which in turn creates more options for puzzle placement.

4.4.2 Solution finder and proving the algorithm

The first working version of the map generator had an algorithm capable of creating 2-dimensional mazes, with a start point and a finish point. To prove that the generated mazes were always solvable, a recursive solution finder[11] was also implemented at this stage. The solution finder was able to find the solution to all of several thousand generated mazes. A simple test suite was written to help in making sure this continued to be true as development of the generator continued.

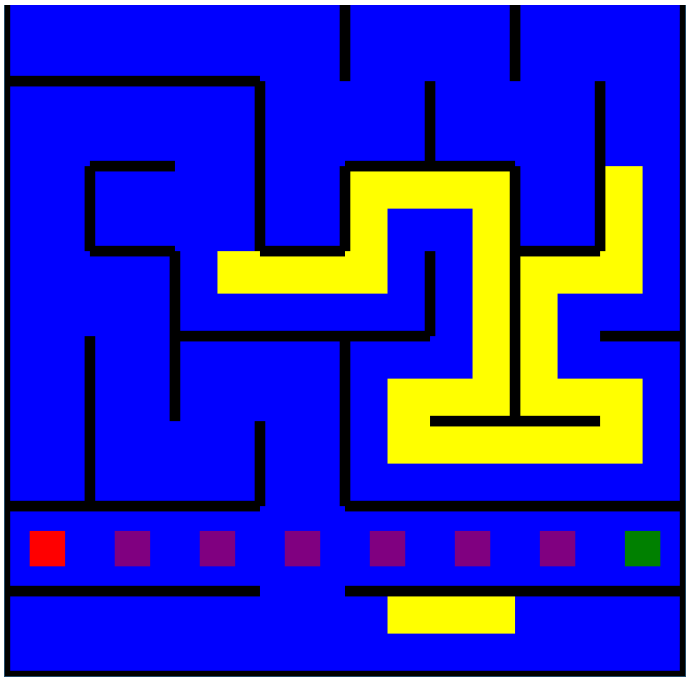
As the generator was layered with more algorithms, to create more complex mazes, the test suite was updated with more tests and more options. To simplify the usage of the test suite, a command-line arguments parser was added. The parser allowed the tester to run only specific tests with

specific arguments, thus shortening the time spent on running tests in the future. For example, if the tester wanted to generate and solve five thousand maps with only the DFS algorithm, and the same map size for every map, the program could be run like this (in a Linux terminal):

```
$ python testcases.py sot 5000 dfs false
```

4.4.3 Graphical representation and visualization

Since it was very hard to visualize how the map would look based on the data the generator produced, a simple graphical renderer was created. The renderer could show the map as a grid of colorful squares with black bars acting as walls between them. At the time levels were implemented, the renderer was updated to support them. The levels were rendered as simple boxes, with different colors depending on their height. The first stage was yellow, the second orange, and the third brown (see figure 4).



Figur 4: Result of the map renderer, with elevations. Red is start, green is the finish line, and purple represents the path that the solution finder has found.

5 Results

In accordance with the goals, the three cars are quite different from each other, and they all have distinct tasks to do in the game; the Cam-car has a camera with which it can stream video, although only to a computer, not the controller. While the Grab-car does not have the electromagnet envisioned, it instead has a way of pressing buttons installed inside walls, that the other cars cannot reach. In contrast, the big car works as intended: it has a small turret which can pan and tilt, and a controllable laser pointer on top of it which can be used to shoot at certain targets. The cars are all controlled via WiFi by an Android application, in which you can swap between them seamlessly and activate their special abilities.

A door module was built with three corresponding activators. It works with the cars and each activator can only be triggered by one of the cars. There is a laser target for the big car, a hidden button for the Cam-car, and a button hidden inside a wall that only the Grab-car can reach. When all these activators are triggered, the door opens as it should.

The map generator program is capable of generating solvable maps of any size. Random parts of the map are also elevated, so as to help produce maps in which exactly one solution exists. However, the program is not capable of generating puzzles in the map, although the groundwork for this part has been laid out, and most of the rules that would concern and guide this process have been formulated.

Despite what we achieved, we do not feel the idea of enhancing classic toys with certain video game aspects has been explored nearly enough. However, we did manage to show how the smartphone can be used in yet another area previously handled by specialized hardware.

6 Discussion

Here we will present some thoughts about the project, both on the idea as a whole and the work behind it, as well as some ideas for further development. Most of the ideas for the future are based on what we initially wanted to achieve, but couldn't for various reasons, mostly due to a lack of time.

6.1 What was accomplished

The idea behind the project was mostly about the cooperative exercise involved in playing the game. In order to force cooperation and a feeling of equal contribution towards the goal for the players, the three cars were required to be different enough that they could not replace each other in the puzzles, but alike enough that none of them was objectively more engaging than the others. By having two different sizes, and different abilities, and then tailoring the puzzles of the map to engage all of them in different ways, we feel that we at least partially accomplished our goal. Of course, to really prove this would require more development and the construction of more modules and activators.

A large part of the work involved in this project was modifying the two small RC cars to work with RPis instead of their built-in hardware. This process involved some new hardware and concepts, like PWM, that had to be understood and incorporated. This was partially successful; the cameras are not working according to plan, but the cars works quite well, and move as expected.

6.2 What was not accomplished

Camera As stated above, the cameras do not work as intended. This is mostly because work on them started later than planned, and it was much more complicated than anticipated. The Android side of things was especially troublesome, because the built-in class for watching videos, "VideoView", is apparently primarily made for watching video files, but not video streams. This caused some issues that were not easy to foresee, such as the video player being unable to determine how much to buffer, since the video has no predetermined end.

Electromagnet From the beginning, the plan was to equip the Grab-car with an electromagnet that could be controlled by the player, and used to pick up and move different objects. This ability would lead to some interesting puzzles, such as having more than three activators connected to a module, where one or more had to be activated by putting objects on them. Unfortunately this was not achieved, mostly because the magnet would require another set of connectors on the DRV8835, which we did not consider when they were decided upon.

6.3 Critical discussion

Looking back on the project there are several things we would have done differently if we were to redo this project. These are discussed in detail below.

Camera Work on the camera and streaming would have started much earlier in the project, as we now know how difficult this would prove to be. In the beginning we thought this would be a fairly easy task that could be finished in about a week, but in retrospect this might have been a part that would have needed work constantly throughout the project.

Electromagnet We never implemented a magnet due to poor planning. Much like with the camera we started working on it too late. There was also an issue with powering the magnet since we did not consider this problem in the early phase of the project. It would have been necessary to rethink the construction of the car, a task we did not have time for.

Breadth of the project While work on the project has overall gone quite well, looking back we feel that it was maybe too broad, and not deep enough. A lot of different technologies were used, most of them new to us, and that prevented us from moving forward at a steady pace. Working with Raspberry Pi was, while quite easy and straightforward, still a new experience. Both Linux and Python was completely new to one of us, and PWM-signal handling was something neither of us had ever worked with before. These new ideas and concepts all add up to a lot of work in gaining a deep enough, but still very shallow, understanding to be able to integrate them into the project. In hindsight, it would have been better to try and minimize some of these ideas and concepts, such as using a DRV8835 built for integration with the RPi, which Pololu also supplies, in order to gain a deeper understanding about a few of them.

Planning Overall we should have put more time on the controller, the cars, and their abilities, and less time on some other things, like the map generator.

6.4 Further development

As this project has many different parts, their possible expansions will be discussed individually in the paragraphs below.

Cars The small cars could be improved to be more energy efficient. As of now, the small cars run on one non-rechargeable 9V-battery and two non-rechargeable AA-batteries. Switching to rechargeable batteries in both cases would help to reduce the negative impact on the environment somewhat. Even better would be to replace all of them with a single rechargeable battery, since having two different types of batteries, each with its own voltage is less than optimal. The big car already has this setup, and it feels significantly simpler to handle. This would be especially true if this project was to be commercialized in some way, since the success of this kind of product is highly dependent on the ease-of-use factor.

Map The map never became all of what we had envisioned, with many different types of modules and activators, that each contributed to the game in some unique way. This is mostly due to a lack of time, but also because the work to create, and especially physically construct, the parts is very time consuming and monotonous. Ideally, the parts should be manufactured in an automatic factory, because they are all very much alike. To help facilitate any future development of this part, we here present two lists of our envisioned modules and activators.

Modules

- Ramp - this would allow access to an elevated platform, one level higher than the current, once active.
- Elevator - an elevator could provide access to multiple levels for one car, but has to be operated by another car.
- Bridge - a rotating bridge that could connect two or more elevated platforms.
- Moving wall - these differ from doors in that they always provide access to some part of the map, no matter their current position.

Activators

- Seesaw - an unbalanced seesaw, that has to be correctly balanced by two cars to activate.
- Broken wire - a cut wire, whose ends are almost touching, that must be repaired with a resistor that can be found somewhere in the map.
- Lever and string - the lever can be pulled by picking up the string and tugging on it. This could be made a team activity by making the string so long that it must be pulled around another car.

Controller The controller could be improved in a major way. Our vision of the controller was to show off all the things you could do when you have your smartphone as the controller of your RC-car. This is already shown by the use of the feedback from the cars, for example how the controller can show the video feed from the cars and also easily changing between the cars. This could be improved further by also making the phone reacting to feedback. It was discussed early in the project, that the cars would try to complete the map whilst being timed, so that you could compete with others and yourself. This would also result in an increase in the difficulty, by having teamwork under time pressure. The controller could work as a timekeeper for this purpose, and also keep track of the teams previous attempts to clear the maze. This could be further developed into a kind of high-score list.

The system for switching between cars should also be improved if this goes into further development. To make the whole experience more multiplayer friendly, the application should allow two players to easily swap cars with each other. The controllers could talk to each other, and create an interface for the switch to happen.

Beyond adding functionality, the controller could also use a remake on the aesthetic side, creating an application that is more user friendly and also adding a finish that is pleasant to look at.

Map generator The map generator is still missing some fundamental parts of the algorithm, chief among them the ability to place puzzles in the maze. This is mostly due to a lack of time and other parts of the project having higher priority, rather than it being very difficult to implement. This means that the rules of puzzle making, as described in the implementation part, have not been tested and verified, but we feel confident about their correctness. Once that part is implemented, the map generator should work for generating random puzzles. Ideally the generation should also adhere to the following rules:

- There should only be one possible solution to the map. This is not so much for gameplay purposes, but rather to make it easier to verify that the map is in fact solvable.
- One more rule to make it easier to verify the generated maps, is that modules should be placed so that they block entrance to entire branches. This is because the algorithm right now has an easier time handling entire branches than cut-off branches.

- For a better gameplay experience, any map generated should have puzzles with both breadth and depth. This means that at any point in the game, there should be multiple possible routes to take (breadth), and choosing one route forces the players to stick with it for as long as possible before it is implicitly revealed to them whether the choice was correct or not (depth).
- Special care may be needed to ensure that puzzles are not created in such a way that it requires three players to solve them, since the game is supposed to be playable by just one or two players. Note that this does not mean they should be solvable by two cars, but rather that they don't somehow require some parallel input that only three players are capable of achieving. This is just a reflection, no example proving the point has ever been constructed.

In addition to this, we have some ideas of how it could be improved in order to really feel like a part of the game. A way for players to save and share their favorite maps over the internet would be fairly easy to implement, and would allow players to challenge each other. If a timing feature was introduced to the game, so that players could get a score based on how fast they went through a certain map, a high score list could be set up on a server. We also thought that in addition to showing a picture of the complete map, one might want to add a sort of instruction manual creation algorithm, that can be used to create step-by-step instructions on how to build the map, and what parts are needed.

Referenser

[1] S. F. Barrett and D. J. Pack, *Microcontrollers Fundamentals for Engineers and Scientists*. Morgan and Claypool, 2006, ".

[2] G. Henderson and P. Howard, "Wiringpi2-python," May 2016. [Online]. Available: https://github.com/Gadgetoid/WiringPi2-Python

[3] R. P. Foundation, "Raspberry pi 3 model b," April 2016. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[4] "Arch linux arm," May 2016. [Online]. Available: https://archlinuxarm.org/

[5] T. Instruments, "Switching regulators," May 2016. [Online]. Available: http://www.ti.com/lit/an/snva559/snva559.pdf

[6] P. Corporation, "Drv8835 dual motor driver carrier," April 2016. [Online]. Available: https://www.pololu.com/product/2135

[7] eLinux, "Rpi low-level peripherals," April 2016. [Online]. Available: http://elinux.org/RPi_Low-level_peripherals#Power_pins

[8] R. P. Foundation, "Power supply," May 2016. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/power/README.md

[9] "Arduino micro," June 2016. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardMicro

[10] Wikipedia, "Depth-first search," Mars 2016. [Online]. Available: https://en.wikipedia.org/wiki/Depth-first_search

[11] W. D. Pullen, "Maze solving algorithms," Mars 2016. [Online]. Available: http://www.astrolog.org/labyrnth/algrithm.htm#solve

A Controller protocol

When data is sent from the controller to the cars, the data is a 16-bit register. The different bits are reserved for different things, which will be explained in detail below.

Bit 0-6 The first 7 bits (0-127) are reserved for data, in this case percentages.

Bit 7 Reserved for flag that specifies where the percentages should go, for example if its for the engine or the servo.

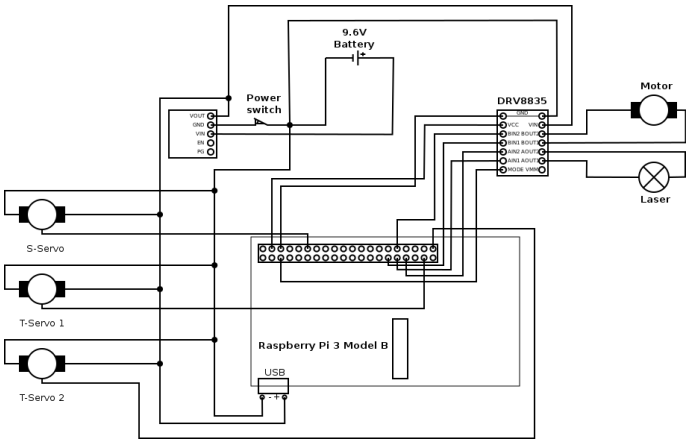
Bit 8 Toggles what way the device decided by the 8th bit should be used, for example be driven forward or backwards and left or right.

Bit 9 This is a flag that specifies whether the special ability should be active.

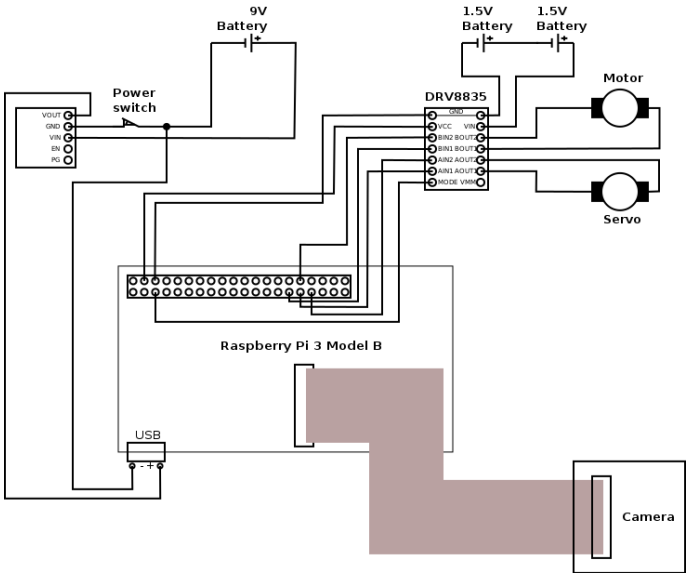
A visual representation of this is included below.

Figur 5: The bit register

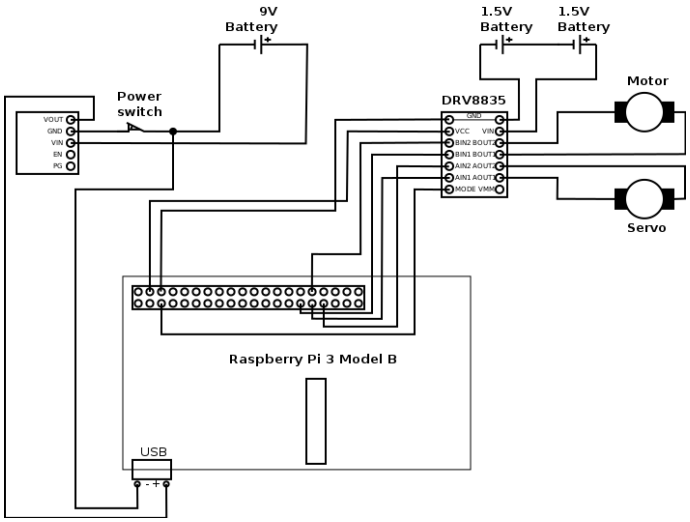
B Schematics



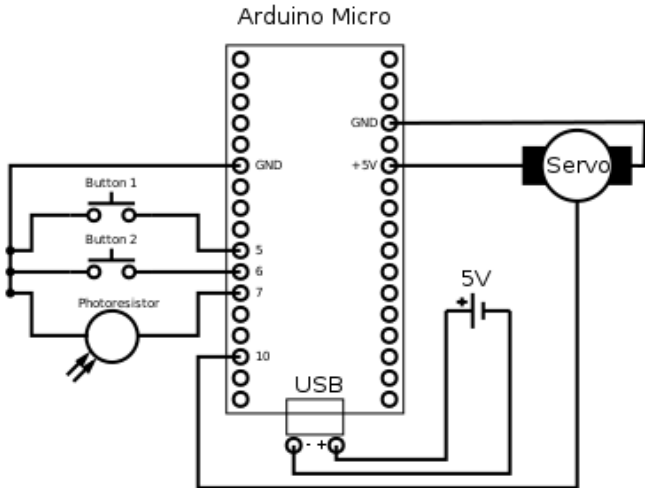
Figur 6: Schematic for the big car



Figur 7: Schematic for the Cam-car



Figur 8: Schematic for the Grab-car



Figur 9: Schematic for the door module