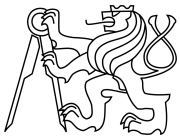
CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY



ASSIGNMENT OF BACHELOR'S THESIS

Title:	Implementation of client and server for control and interpretation of robot Karel
Student:	Stefan Ćirić
Supervisor:	Ing. Jan Trávníček
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2018/19

Instructions

Study the Karel programming language.
Design a client -- server communication over HTTP supporting sending complete Karel programs and individual Karel commands.
Design a Karel server and implement it on the ESP8266 SOC computer.
Test the implementation of the Karel server.

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague March 1, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Software Engineering

Bachelor's thesis

Implementation of client and server for control and interpretation of robot Karel

Stefan Ćirić

Supervisor: Ing. Jan Trávníček

16th May 2017

Acknowledgements

I would like to thank my mentor Ing. Jan Trávníček, for helping me in finding an appropriate topic for my bachelor's thesis, his continued support during the work of the thesis and my studies at the Czech Technical University. Secondly, I would like express my honest gratitude towards my teachers, who work with such passion, have the most patience and understanding, and from who I have learned so much. Last but not least, I am grateful to my family and loved ones for their constant encouragement and support throughout the entire period of my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 16th May 2017 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Ćirić, Stefan. *Implementation of client and server for control and interpretation of robot Karel.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Obsahem této práce je implementace systému klient-server pro robota Karla. Systém bude implementován na čipu Arduino ESP8266 WiFi, na který se uživatelé připojí pomocí WiFi a budou tak schopni komunikovat s robotem prostřednictvím poskytované webové stránky. Práce podrobně popisuje čip Arduino ESP8266 WiFi, na kterém je systém vyvíjen, zároveň plně vysvětluje pojmy programovací jazyk Karel, robot Karel a ukazuje, co je robotovým úkolem a čeho může dosáhnout. Druhá část se zaměřuje na návrh a implementační techniky vybrané pro splnění práce a předvádí vnitřní strukturu a architekturu projektu.

Klíčová slova implementace, uživatel, server, karel programování, c++ programování, arduino IDE, HTTP, HTML, ESP8266 WiFi čip

Abstract

The content of this thesis is the implementation of client and server system for Karel robot. The system will be implemented on an Arduino ESP8266 WiFi chip, through which users will connnect over a WiFi connection and be able to interact with the robot through the web page provided. It describes in detail the WiFi chip on which the system is developed, as well as fully explaining

the concepts of the Karel language, Karel robot, showing what the robot's task is, and what it can accomplish. The second part focuses on the design and implementation techniques chosen to accomplish the task, going over the internal structure and architecture of the project.

Keywords implementation, client, server, karel programming, c++ programming, arduino IDE, HTTP, HTML, ESP8266 WiFi chip

Contents

List of Figures

Introduction

The concept of mazes and maze-solving is familiar to everyone. Since our early days, we are offered by our elders all sorts of tasks, such as puzzles and riddles which we give our best to solve. Solving mazes can be a very exciting experience. We are set with a problem which has a clear goal, and if we manage to accomplish it we are rewarded for our efforts. Mazes can be implemented in all sorts and forms. A good example of using mazes in educational purposes, especially in programming, is done by the robot Karel. Karel is a programming language created with the purpose of displaying programming concepts to interested minds through a simple game. Karel is a robot that exists in a two dimensional world where he is tasked with finding a path from point A to point B. To achieve this, he must bypass all of the obstacles in his path, before he can finally reach his destination. With the help of the Arduino ESP8266 WiFi chip, I have created a maze-solving puzzle with the concepts of Karel in mind. The ESP8266 WiFi chip is one of the many offered by the Arduino company, and it provides an open-source platform for many kinds of hardware and software. This allows for a wide range of software tools and documentation for any independent project. There is a world-wide community that works with Arduino, constantly evolving and upgrading the libraries and tools available, while also sharing their work and experience. In my case it was the development of a Karel robot that exists on a WiFi server through which clients can connect and interact with Karel.

Motivation

Ever since I was a young boy I have had a deep interest in solving puzzles, finding hidden meaning and context, and creating things. Whether it be paper wallets that my childhood friend and I have made by dozens, a castle made out of LEGO's, a medieval shield and sword from pieces of wood from the backyard, a tree house in our local park - I have constantly been creating something. This has probably led me to choosing the field of programming

as my future career. Once I have heard of the Arduino company and projects I was interested in doing one myself. With my aforementioned interest in puzzles, the Karel Robot project seemed like an excellent fit. As mentioned, the Karel itself was created with an educational purpose in mind to teach those interested in programming, and Arduino being a platform that allows experimentation and development of interesting projects for students, hobbyists and professionals, I wanted to use this opportunity to create something new, while also testing my abilities to work with new technologies, concepts and environments.

Description

This thesis can be split in 5 main parts, excluding the introduction. Each of the parts covers a different topic of the project, and they are information about the Arduino WiFi chip, the Karel programming language, my design and implementation, testing of the application and my thoughts and conclusions. The first part is very technical, and features a hardware overview for the chip. As all chips are different, and there are many groups and families of chips, it goes into detail about the ESP8266, its capabilities, functions, elements, available libraries and file system. The second part covers the Karel robot and Karel programming language. It briefly mentions the history of Karel, its origins and touches the topic of robot languages as well. It then goes into more detail regarding the main objective of Karel, what it's supposed to achieve and how. It also describes the idea of the robot and the world it lives in. The third part consists of my chosen projects' architecture and my way of solving the task. It goes over the developed architecture on the chip into detail, mentioning the classes involved, and later it dissects each class separately, explaining its purpose and goal in the scope of the project, with some examples. The fourth part regarding testing shows how the application performs. The last part, being the conclusion, goes over my experience working on the project, an overview of the process, results and expectations.

Chapter 1

Aim of the thesis

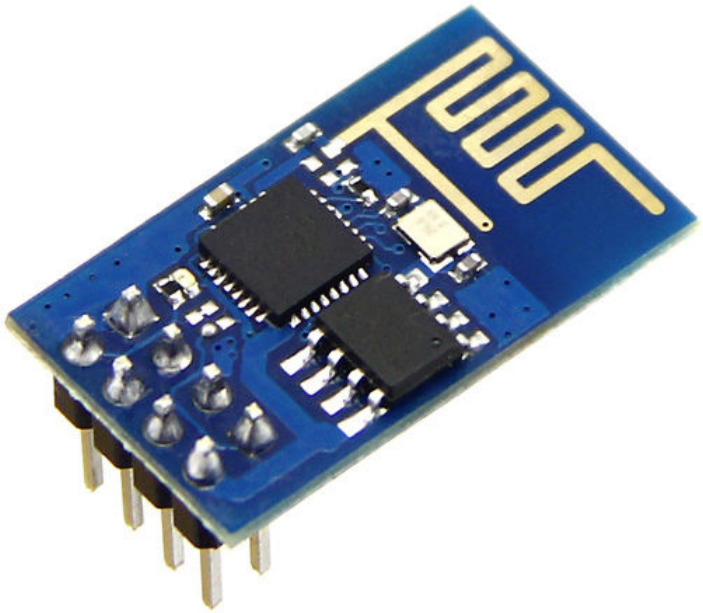
The aim of the thesis is to develop and to implement the client and server for the Karel robot with the help of the Arduino ESP8266 WiFi chip. The thesis will serve as documentation for my project, for myself or anyone else that would wish to expand in the future. It records all of the steps that were taken to complete it, along with the proper description of components used to create it. It also shows my ability to work on a larger scale project by incorporating multiple technologies together to achieve an end result. The other goal was to gain experience and get used to embarking upon a new project and trying my best to adapt to different circumstances, eventually conquering them. Finally it is to show my ability to successfully create something that I can proudly claim to to have accomplished on my own.

Chapter 2

Arduino ESP8266

The ESP8266 WiFi Module (Figure:2.1) is a self contained SOC with integrated TCP/IP protocol stack, allowing any micro controller access to a WiFi network[1]. Each individual chip comes with a pre-programmed set of AT commands, which allows for easy usage right out of the box. It can set up HTTP, mDNS, SSDP and DNS servers, do OTA updates, as well as use a file system in its flash memory and work with SD cards.

Figure 2.1: The ESP8266 chip design



It has a very low power consumption of 3.3V and a small processor on board, allowing the chip to work completely autonomously. Along with its low price of approximately 200 Czech crowns makes it one of the more desirable chips for home projects. Compared to the alternative which is a regular Arduino chip and WiFi module that cost a lot more. Although the ESP8266 has only 2MiB of flash memory, as do most of the chips in their category, and the number of GPI0 pins are limited to 8, it still provides plenty of functionality.

2.1 Technical Overview

The ESP8266 contains complete WiFi networking solution, meaning it can be used to either host or offload WiFi networking functions from any other processor. In the case of hosting an application, it boots up directly from an external flash, while the integrated cache improves the performance of the system[2]. Alternatively, we can use it as a WiFi adapter. It is one of the most integrated WiFi chip's in the industry. Besides it's integrated antenna switch, RF balun, power amplifier and the likes, it also integrates an enhanced version of Tensilica's L106 Diamond series 32-bit processor, with on-chip SRAM. It is also often used with other applications through external sensors through it's GPIO pins.

The chip has been designed for mobile use, while aiming for the lowest power consumption possible, with a combination of several techniques[3]. It's power saving architecture operates in 3 modes: active mode, sleep mode and deep sleep mode. It consumes about 60 nanoA in deep sleep mode, and less than 1.0 mA or less than 0.5mA to stay connected to the access point. It can be programmed to wake up at any required interval, or when a specific condition is met. This feature allows to remain in low-power standby mode until WiFi is needed.

2.2 Hardware Overview

ESP8266 is embedded with Tensilica L106 32-bit micro controller[2]. It features extra low power consumption and a 16-bit RSIC. The CPU clock speed is 80MHz, with its maximum value of 160MHz. It allows for 80% for user application programming and development. There are 3 interfaces available to connect to the embedded MCU. All of them can be visited by request, while the memory arbiter will decide running the sequence according to the time when they are received by the processor. The interfaces are:

1. Programmable RAM/ROM interface (iBus) which can be connected with a memory controller

2. Data RAM interface (dBus)

3. AHB interface, to visit the register

As mentioned, it supports OTA which is a process of uploading firmware to the chip using a WiFi connection instead of a serial port[1]. To perform this the Arduino IDE is most oftenly used in the process of development, while a Web Browser or HTTP Server are better options after deployment. The basic requirements for the OTA is that the flash chip size should be able to support both the old and new program at the same time. its external SPI flash can theoretically support up to 16 Mbyte of memory. Also, the minimum flash memory requirement is dependent on whether OTA is disabled or enabled. If disabled the minimum is 512 kByte, or 1 Mbyte alternatively.

When performing the OTA the module has to be exposed wirelessly so it can be uploaded with a new sketch. This means that there is a chance of it being hacked or violently loaded with a different program. To prevent the likelihood of being hacked it is good practice protecting your uploads with a password and selecting a certain OTA port.

Some protection functionalities are built in and do not require any additional coding by the developers. It uses the Digest-MD5 to authenticate the uploads. It is verified on the ESP side using an MD5 check sum.

2.3 Firmware Overview

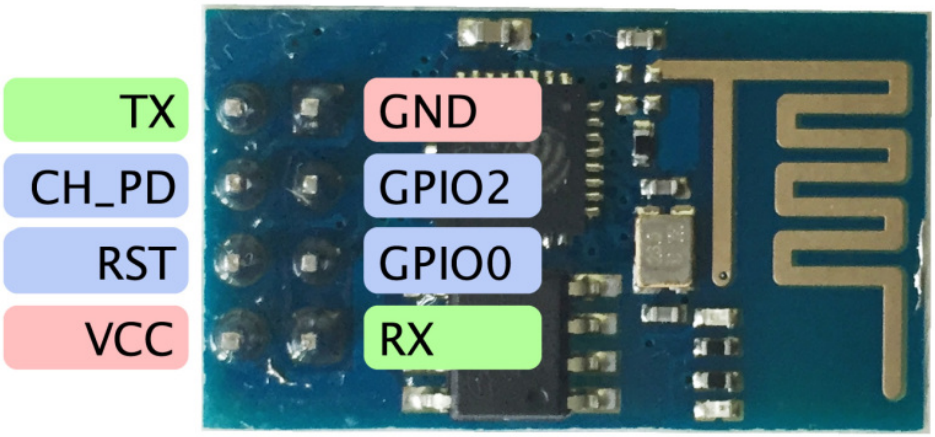
The chips' ROM and SRAM load instructions during wake-up through an SDIO interface from the external flash, which is where the application and firmware is executed. It implements the full 802.11 b/g/n/e/i WLAN MAC protocol and WiFi Direct specification[1]. Not only does it support basic service set, but also P2P group operation compliant with the latest WiFi P2P protocol. Protocols which are handled automatically by the ESP8266 are:

1. RTS/CTS
2. acknowledgment
3. fragmentation and defragmentation
4. aggregation
5. frame encapsulation
6. P2P WiFi direct

2.4 Communication

Although the many benefits of the ESP8266 chip, being the smallest module it has the most limited I/O pins. At first, all of the pins are used for programming, leaving us with very little.

Figure 2.2: Naming and ordering of pins on the ESP8266



The pins it has are numbered from 1 to 8 and are TX, CHPD, RST, VCC, GND, GIO2, GPIO0 and RX Figure:2.2. Of these, VCC, GND, RST and CHPD are not I/O pins, but are necessary for the operation of the module. This leaves us with GPIO0, GPIO2, TX and RX as I/O pins. However, even these have predefined functions. The GPIO pins determining what mode the module starts up, while the TX/RX are used to program the module and for Serial I/O, most notably used for debugging[4].

As mentioned another way of interacting with the chip is directly through the Serial port. The Arduino IDE supports this feature and allows to set the baud rate among other things. The form of communication is done through the AT instruction set [5]. Often the chip has predefined commands, however user-defined commands are supported. Commands are sent by typing the AT prefix plus the command, like: AT+[command name] Most often used commands are AT and AT+RST to see if the board is working and to restart it.

2.5 File system

The file system as well along with the program are stored on the same chip, however, programming a new project will not change the contents of the file system. This means that we can use the file system to store program data, configuration files or content for a Web server. However, this imposes some limitations[2].

The design used to accommodate the implementations and constraints on the chip is SPIFFS, because it was designed for small system, although at the cost of some simplifications.

SPIFFS does not support directories, it just stores files. The good thing

is that since it is not a traditional file system it allows for the '/' character to be used in names, thus allowing the usage of directory listing when using the functions. Another possible problem is that is only supports 32 character for file names, with one being reserved for the terminating character. It is suggested to not nest files too much and keep naming conventions short, as problems can arise if the limit is reached.

2.6 ESP8266WiFi library

The ESP8266 WiFi library was developed based on the ESP8266 SDK, using all of the naming conventions[1]. However, over time the features developed outgrew the Arduino WiFi library. The chip can work in different modes, namely it can work as a station, that is a device that connects to a WiFi network that is provided through an access point. The access point is then usually integrated with a router to provide access from a WiFi network to the internet. Each such access point is recognized by its SSID.

Alternatively, it can perform the functionality of a soft access point, to establish its own WiFi network and have others connect to it. In fact, it can simultaneously perform both roles, which provides the ability to build mesh networks. The library provides a wide variety of C++ methods and properties to configure these modes.

In the mode of a soft access point the ESP8266 can provide a WiFi network for others, except for the fact that it has no interface to a wired network. This is the reason we distinguish between access point and soft access point. The main purpose of this mode is to be used as an intermediate step before connecting the ESP to a WiFi in station mode. Most notably when the SSID and password are not known to the chip upfront, it boots soft access point mode, which allows us to connect to it through mobile and then provide it with the data to a network. Of course, it can support its own server and offer its connections and features. This is the mode that is going to be used in this project for the implementation of the server for the Karel robot. To just quickly note its other modes of operations they are: Station, Scan, Client, Client Secure, and Server.

The main functionality of this class is setting up a WiFi network for users to connect through an SSID and password. Managing connections and/or shutting them down.

2.7 ESP8266WebServer library

The ESP8266 WebServer library offers functionality of having a server on the chip[1]. It holds its IP address, header values and argument values sent to it. This allows the chip to process GET and POST HTTP requests. Since it does not offer an interface, we must use the IP address to connect. Once

the connection is established it serves us the page. It holds the HTML page as a String in a variable which is then sent through a HTTP request back to the client. This is done by the "send" command. It requires the code, type of content we are sending and the HTML page in String form. To respond to the client is done by the "on" command which allows to act accordingly based on the requested page. To process the command we can retrieve it in String form from the arg variable.

For ease of use, it has an extensive "handleClient" method which deals with clients. It periodically checks for connections. Upon an establishment of a connection it waits for data from the client to be available, and then processes it. There is a limit, i.e. a timeout for which it waits for the request to be processed. If the timeout is reached it closes the connection and tries again. If the connection is successful, it retrieves the data and closes the active connection, after which the process is repeated.

Chapter 3

Karel

Karel programming language was invented by Richard E. Pattis. It was first introduced in his book "Karel the Robot: A Gentle Introduction to the Art of Programming. The prefix Karel comes from a Czech writer, Karel Capek who introduced the word robot.

Richard E. Pattis was a senior lecturer of computer science at Stanford University in California[6], and he created the Karel language with the sole purpose of showing the concepts of programming to people who wanted to learn, as well as expand on already existing concepts.

Karel Capek among being a writer had multiple roles such as being a publisher, literary reviewer, photographer and art critic[7]. However, one of his best known works is his novel and play "War with the Newts" and R.U.R. (Rossum's Universal Robots)[8]. In this play he introduced the word robot for the first time, and it has been accepted world wide ever since.

3.1 History of robot languages

While the history of robot programming languages is very complicated. Some would say that it essentially began with the introduction of automata[9]. Automata were not very flexible, as they were purely mechanical and most often than not had only one program. To re-program it, you would need to redesign the whole machine. Non the less they are grandparents of modern technology and worth mentioning. Now it is generally assumed that programming languages are sorted into three main categories, being:

1. Programming by teaching

2. Robot-oriented programming

3. World modeling and task level programming

The first generation of languages operated at a very primitive level, mostly suited for highly repeated tasks. They were based on already existing languages that had been developed in the 1950's. The second generation experienced a boom in robot languages, as the interest and need grew rapidly. Among the second generation is the Karel programming language. They offered more control and flexibility, to some extent even limited artificial intelligence. The third generation no longer cares about the specific language you are using, yet more about the programming ideas and methods at hand. We could say that it is an ever-evolving field, with many features still at the research stage. Aspiring to self-learning robots, where they could teach themselves, essentially what we know today as artificial intelligence.

3.2 Karel the Robot

Karel is a robot who lives in a two-dimensional world made up of horizontal and vertical streets which he can explore[10]. Although this world may not be interesting by today's standards, as there are no museums, universities and theatres, it is still sufficient for him to perform basic commands, such as walking, turning and picking up items.

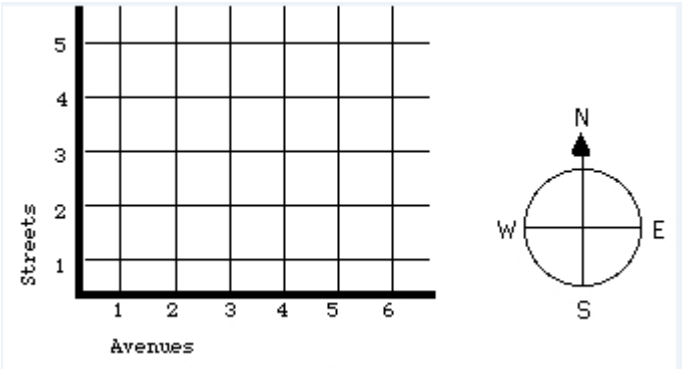
Essentially Karel came with a set of a few basic commands, such as MOVE, LEFT, PICK UP and TURN OFF in the depiction of Richard E. Pattis. Although over the years the Karel robot has expanded and can offer more functionality, even learn new custom commands, this is the crux of the robot. Some models of the robot affect his world, introducing new elements with which he can interact. The reason for these extensions was the desire to include Karel in object-oriented design, and allow for object-oriented concepts to be explained.

3.3 Karel's world

Informally put, the robot world is a grid of streets that the robot can traverse[11]. Figure:3.1 illustrates the structure of the world with standard North, South, East and West compass points. While this is the basic depiction of Karel's world, there are versions in which he can go through the walls. This moves the beginning of the world into a negative number, which is not very suitable for this implementation. Since we are using two-dimensional arrays to represent Karel's world it marks the beginning or origin of the world at (0,0) coordinates.

Both streets and avenues are numbered from 0 to N. The default size supports Karel's world of 35x35, which places the centre at the position on coordinates (17,17) where the flag is located, meaning that is the block he must reach. All positions can assume relative and absolute paths. By giving a concrete position such as (1,1) or (3,5) or saying simply something is 3 blocks

Figure 3.1: Depiction of Karel World with compass.



west and 2 blocks south. Since all blocks are of the same size it would indicate that something is 3 steps left and 2 steps down. We assume streets are vertical and avenues are horizontal. The robot moves according to its direction, so this is where the compass comes in play. Since robot can only move in the direction it is facing, is it important to keep track of it.

While there are main 4 walls at the end of each room, many versions of the Karel make the robot go through a maze. This is done by occupying blocks on grid by walls. By connecting multiple blocks of walls together we can create a maze for the Karel as can be seen of Figure:3.2. Like on the edges of the world he cannot pass through these walls, so he must bypass them.

3.4 Karel language

The Karel syntax was created for educational purposes. Each Karel robot has specified default commands which it understands[10]. However, since this is often not enough the language allows for creativity and expansion. The basic syntax of a procedure has the following form:

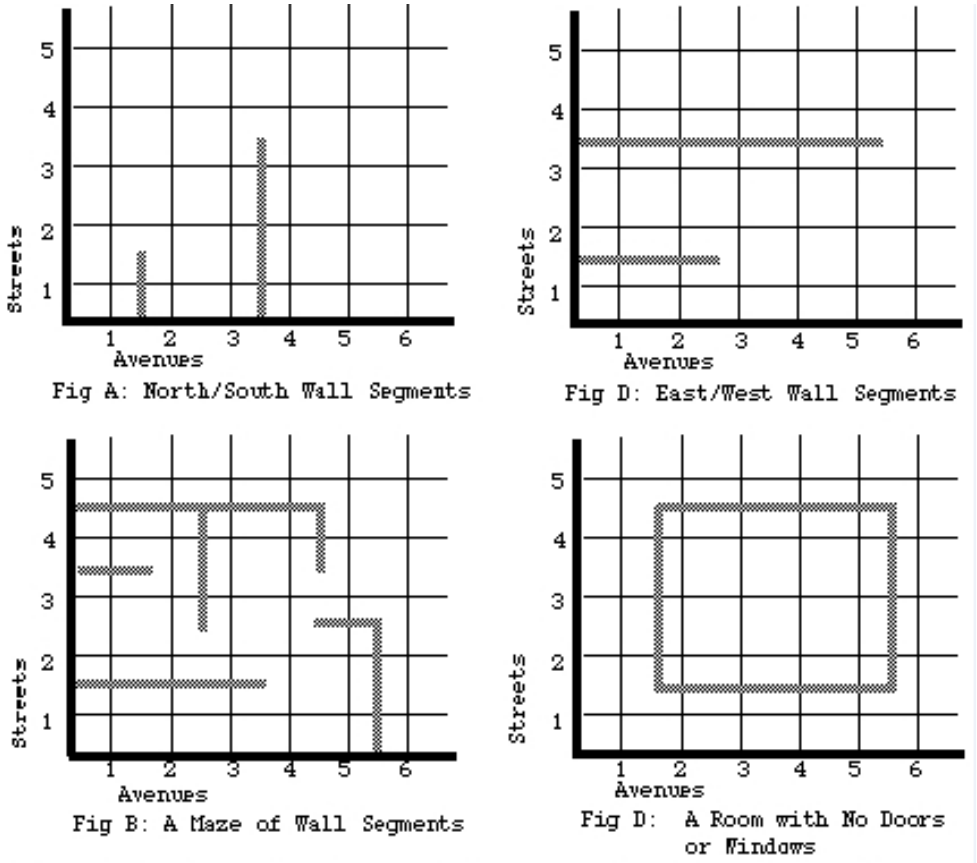
```
new_command_name

BEGIN

    list_of:
        CUSTOM_INSTRUCTION
        or
        BUILT_IN_INSTRUCTION
        or
        KEYWORDS
END
```

The syntactical elements are:

Figure 3.2: Examples of Karel wall creation.



- Statements: BEGIN, END, ITERATE, IF
- Keywords: STEP, LEFT, PICK UP
- Numbers: 1,2,3 ... N

3.5 Karel abilities

Karel is a small robot which lives in the world we create. He is a nice companion and he always listens to his owner. He can turn left in his block, make a step forward, or use his arm to pick up a flag from the block he is standing and read it to his owner. His eyes are special cameras with which he can observe his surrounding and check if there are things around him. This is essential when moving around to avoid hitting a wall.

At first Karel might not be the most impressive robot, but he always wants to improve, and for this he needs help. His owner can teach Karel new commands by adding onto what Karel already knows. By using the Karel

language the owner creates a special keyword and a list of commands for Karel. Karel carefully reads the new task and remembers it. From then on, if the user tries to ask the Karel to perform the new task, he would do it in the way it was defined by the user.

3.6 Karel's task

There have been many variations and problems defined for Karel over the years. In essence, despite the additions on his path, Karel has always had a role of a finder, that is he's main problem was finding a path to his desired item. This version tasks Karel with finding a flag in his world, which will always be in the center of the map, picking it up and reading the hidden message to his owner.

When Karel wakes up, he forgot where he fell asleep. The only thing he knows is that he needs to find the map, whose location is familiar to him, and that he can do this only with the owners help. On his path, there could be obstacles which he needs to avoid, so the owner needs to pay attention as well. Karel will not walk through an obstacle, yet he will merely remain in the same block. To find the direction he is facing, by turning left in his block we can deduce the new direction he is facing and proceed to find the best path to our flag. Since he is a robot he will not get tired if we don't find the shortest path.

Chapter 4

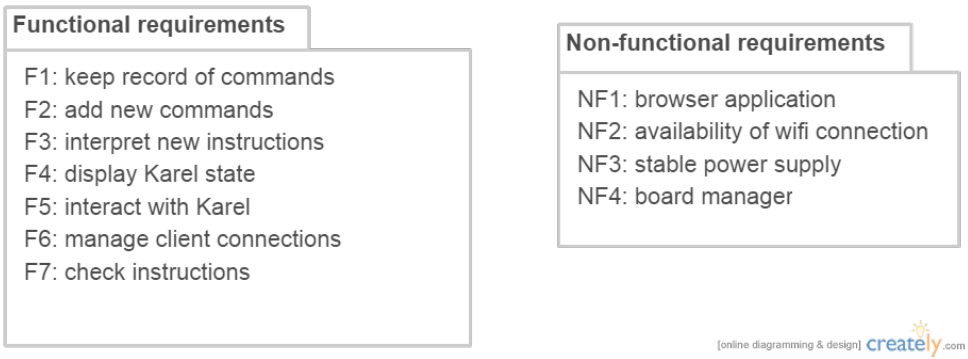
Design and implementation

In this chapter I will go over the design, design choices and implementation of the project. The technologies used are the Arduinos framework suited for C++ development, such as the Wifi and Server libraries that extend on its ability to create a server and further manage it. Due to the limited memory capacity there is a custom-made template by me for vectors. The transfer of data between the server and the client is done by HTTP GET method, passed as a string.

4.1 Application requirements

The project has a set of requirements it needs to fulfill to operate properly. They can be divided into two groups: functional and non-functional requirements (Figure:4.1)

Figure 4.1: Application requirements



4.1.1 Functional requirements

- **F1: keep record of commands** - The application needs to have all of the available commands stored, making them available when used by the user.

- **F2: add new commands** - The user can create his custom command in the Karel language. The system needs to be able to store the command among the default ones, making it available for later use.

- **F3: interpret new instructions** - The system needs to be able to interpret the custom created commands, performing them according to the defined syntax of Karel.

- **F4: display Karel state** - The system needs to be able to represent the internal changes from Karel and his world visually to the user on the web page, in the form of sentences. Showing Karel current coordinates, and the direction he is facing.

- **F5: interact with Karel** - The system needs to allow the user to submit commands to the Karel robot through the server, allowing the user to move the robot around the world.

- **F6: manage client connections** - The system needs to be able to allows users to connect to the server, manage the connection while by processing requests,

- **F7: check instructions** - When the user inputs a new command, the system needs to check the correctness of it, and only accept it if it follows the proper syntax.

4.1.2 Non-functional requirements

- **NF1: browser application** - The application is developed for use through a web browser on a computer or phone.

- **NF2: availability of WiFi connection** - To access the web page the user needs to have a device that is able of establishing a WiFi connection in order to connect to the Arduino ESP8266.

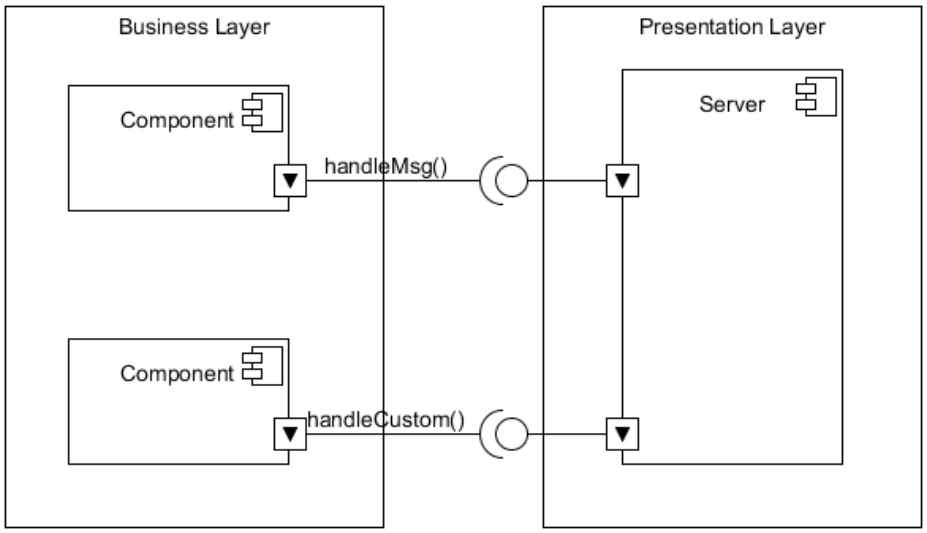
- **NF3: stable power supply** - The Arduino ESP8266 WiFi chip requires a stable power supply of 3.3V to function properly.

- **NF4: board manager** - In the case a user has an Arduino board manager at hand, he is able to connect the ESP8266 WiFi chip to the board manager and provide the power supply through a USB port.

4.2 Architecture

I have decided for a two-layered architecture (Figure:4.2) for this project, composing of the business and presentation layer only, excluding the data layer. There will be no option of saving any custom made commands after the termination of the application, thus the data later is not necessary. The lowest

Figure 4.2: Project architecture



layer, in this case the business layer, is tasked with accepting and processing data sent from the server, as well as performing the tasks that are passed on by the server through the presentation layer. It offers an interface to the server through which they interact. There is no direct contact between the members of the KarelRobot and KarelHandbook class. This is done with the encapsulation principle in mind, to limit the influence of the outside users to what the interface offers. The presentation layer deals with the clients through the browser, by looking for new connections, managing existing ones and closing them. It keeps the barrier between the front-end and back-end of the application. It is mainly tasked with transferring data between the client and Karel robot. It also allows the client to see all of the information changes as displayed on the web page.

4.3 Classes

The project essentially consists of 14 classes in total, them being: KarelRobot, KarelHandbook, Vector, Server, WiFi, IPAddress, Statement, Step, Left, PickUp, StatmList, Iterate, If and While respectively. They are split into two

main groups: management and functional classes. The management classes (Figure:4.3) can be separated once more, between the built-in Arduino classes, Server, WiFi, IPAddress, and the classes I have developed being KarelRobot, KarelHandbook, Vector, Server and WiFi. They have more functionality and responsibility than the functional classes. They take care of all of the organization and processing inside the application.

The functional classes (Figure:4.4) are all subclasses from an abstract class Statement. They are used to express the functionality of the Robot's default commands, along with the new commands it will eventually learn. They are essentially very simple, and don't perform more than 3 or 5 tasks each. I will now go into details regarding each class.

4.3.1 Arduino classes

Arduino offers a lot of libraries for its projects. As mentioned before, for my task I am using the WebServer and Wifi library which have the Server, WiFi and IPAddress class. The WiFi class is used to create a wireless soft access point through which clients can connect to the Arduino ESP8266. It requires two string parameters, an SSID and password for the hot-spot. After initialization, it starts broadcasting its connection around and waits. It has a static IP address through which users can connect through the browser once they are on the network. The IP addresses are actually objects of the class IPAddress, so to retrieve it from the WiFi access point we need to store it into an object of class IPAddress. The Server class is used as the base class for all Ethernet server based calls. It is not called directly, but invoked whenever you use a function that relies on it. Once the server is activated, and the users are connected and try to access the IP address the Server class invokes the "on()" method which serves the user with a web page. The web page is saved in String format, which allows for easy manipulation from the programmers side when in need of presenting internal values and variables to the user, just by adding the variables to the String. On each user request the "on()" method is used to process the request. Each request has a function assigned to it, which deals with the request, or a single function perfroms a specific task based on the request. Once the functions is finished, the server uses the "send()" method to serve the user with the new page and changes. It sends the HTTP code and new page as a String. To deal with clients the server has a extensive "handleClient()" function, which is part of the internal library for the server. It starts off by waiting for connections. When a connection is found, it checks it before proceeding. On each user request it is invoked again. Upon finishing it closes the connection, and goes back to its original state of waiting for new connections.

4.3.2 KarelRobot

KarelRobot is one of the two main classes of the program. It represents the basic robot and its functionality. It holds information of the essential robot commands, his coordinates in the world, as well as the position of the obstacles that he faces. The robot starts at a random free location in the world, turned to the North. It knows how to move, turn left, check his surroundings and pick up a flag from the ground. To get around the world he knows how to check his surroundings by checking each vertically and horizontally adjacent block next to him for an obstacle. This is done by taking into account the direction the robot is facing and then accordingly checking the block in the world to see if it is occupied or not. If there is no obstacle the methods return true, otherwise false. When trying to move, the robot depending on his direction tries to move exactly one block forward. To do this he firstly tests the block, and only after confirming the space is not occupied will he change his position. In the case of an obstacle, he remains in the same spot, waiting for a new command. The turn left method allows the robot to move one direction to the left, while remaining in the same spot. If he was facing North before the left method was executed, it would without checking change his direction from North to West and finish. The pickup command instructs the robot to try and get the flag from the ground, without moving him. In case the robot is not in the centre of the world, the method would simply return to the user that there was nothing there. However, in the case the robot reaches the centre and then executes the pick up command he would successfully return the secret text written on the flag.

4.3.3 KarelHandbook

While the KarelRobot offers the basic and essential functionality of Karel, KarelHandbook can be thought of as an extension class of Karel. Wherever Karel goes he carries along with him a book in which he writes down things he learns on his journey. However, every time he falls asleep, the writings are mysteriously gone. The KarelHandbook class stores information of all the commands Karel knows so far, including the basic commands as well as the new custom ones. It holds all of the keywords Karel can recognize and interpret, and it has variables which hold temporary information such as the new desired command to be learned and some counters for easier handling. The main functionality is in the analyzing and interpreting a string sent as a new list of statements that Karel should learn as a new command. It uses a couple of methods to achieve this. First of all, when Karel is supposed to learn a new command from the user the "getUserInput()" method is called which just copies the string into the Handbook as a string of regular words. The process of analyzing and interpreting begins with the "createCommand()" function which recursively constructs the new command. For easier parsing the method

"readNextWord()" will read the next available word from the user input and retrieve it. Then the word is checked whether it is a keyword, default command or custom command. Depending on the word processed an appropriate method (eg. createIterate()) is called and a new object is constructed which is then added to the list of statements from which the new command is supposed to consist of. If somewhere along the way the process is disrupted, eg. by not following the correct template for new commands, it is simply dumped and deleted. It returns a String informing the user that the process was not successful and finishes. The methods createIterate(),createStatmList() etc. deal with the required syntax of each of the keywords. If we were trying to create a new Iterate object we would need to provide it first with the "ITERATE" keyword, our desired number of repetitions and a command we would like repeated. So something that would be accepted by the parser and would create a new command looks something like this:

```
RIGHT

BEGIN
    ITERATE 3 LEFT
END
```

This small procedure follows the syntax of the "createIterate()" method and it would successfully create and add the command "RIGHT" to Karel's arsenal. This is done similarly for all other such methods. If the syntax is not met, the method stops, deletes and returns a NULL pointer.

4.3.4 Statement

The Statement class is the parent class of all functionality classes, and it provides the necessary interface for them. It allows for polymorphism later down the line. It offers an interface for the "execute()" command, which depending on its subclasses performs the tasks associated with the semantics of the invoked class.

4.3.5 Step, Left and PickUp

The Step,Left and PickUp classes are the most plain and simple classes derived from Statement. They work as wrapper classes for Karels' default commands. Since all new commands are developed as "StatmList"s (statement lists) which hold an array of commands to be executed as Statement pointers, they are essential for proper functionality.

4.3.6 Iterate

The Iterate class handles the creation of all new iteration objects. its member variables are a Statement pointer which will hold the new command it will

iterate, and an integer representing the number of iterations. Since it holds a " Statement* " it can accept any of its brother and sister classes. This allows for iterations of simple procedures such as just turning left, or more compound procedures which are done by the StatmList class. The required syntax for Iterate is:

```
ITERATE number_of_times STATEMENT
```

4.3.7 StatmList

The StatmList class represents compound commands learned by the Karel. Essentially each user request for a new command is a StatmList. It holds a vector of Statement pointers which are executed one by one. It starts off with 5 reserved spaces for each command, which can be extended if necessary. It is created by the "createStatmList()" method of KarelHandbook which deals with the parsing of the syntax for StatmList. StatmList uses the keywords BEGIN and END instead of brackets ([ ]) for an indication of a block or group of commands that should be executed independently. Nesting of StatmList's is possible due to the polymorphism allowed from the deriving from the Statement class. If there is an error in following the syntax for StatmList creation in the "createStatmList()" method, the process is interrupted, and a null pointer is returned. The template for creating a StatmList object is:

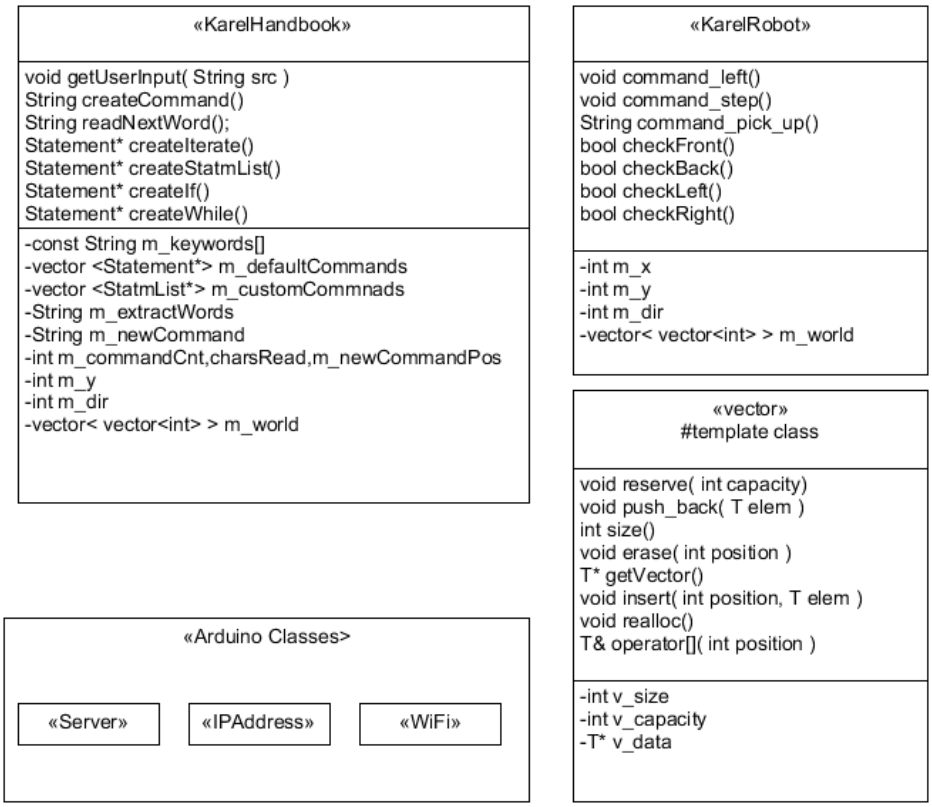
```
BEGIN
    KEYWORDS
    or
    COMMANDS
END
```

4.3.8 While

The While class is used to deal with an execution of some desired commands while a certain condition holds. As mentioned in the If class description, the conditions are defined by the KarelRobot class. The While class deals with a repetitive execution of commands while a certain condition is met. It is somewhat similar to the Iterate class in the sense of repeating a single or list of commands, however while the Iterate class will perform the repetitions exactly equal to the given number, the While class will do so until the condition returns false. The While class is very important for the development of Karel, as it allows the user to create AI-like procedures for Karel in which he try and reach the centre of the world and ultimately read the flag. The expected statement can be a StatmList, meaning it can do more than just simple commands. The required syntax for the While class is:

```
WHILE condition STATEMENT
```

Figure 4.3: Representation of management classes



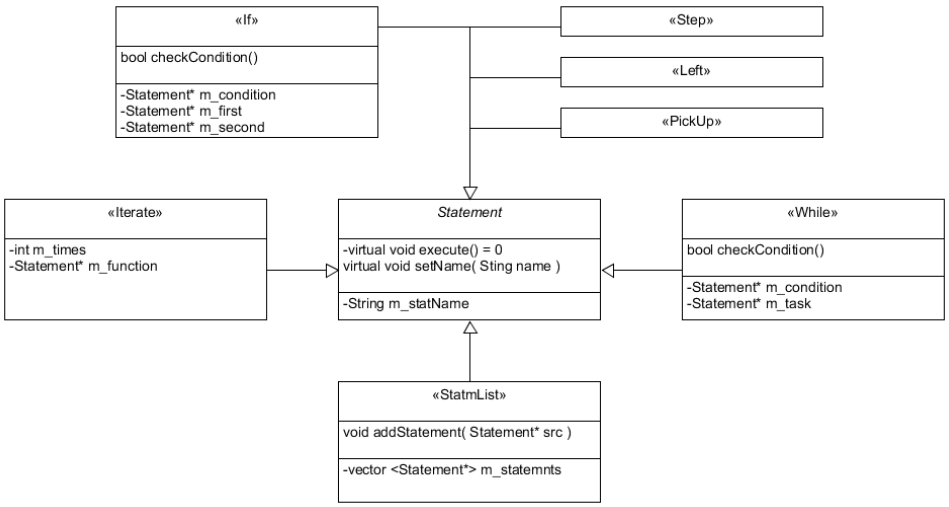
4.3.9 If

The If class is used to represent branching or decision making in the the program based on a certain condition. The conditions are defined from the KarelRobot class as checkFront(), checkBack(), checkLeft() and checkRight(). The If class requires a condition and one or two statements. If two statements are given to the If class when creating an instance, upon checking the validity of the condition it would execute one of the two statements. If it is true the first statement would be executed, otherwise the second one. However, if there is only one statement passed as an argument, it would only perform that statement if the condition is met, otherwise after checking the condition it would just stop. The syntax for the If class is:

```
    IF condition STATEMENT_1

or
    IF condition STATEMENT_1 STATEMENT_2
```

Figure 4.4: Representation of functional classes



4.4 Use cases

The main actors of the application will be the client and server. All communication and actions are invoked by the client through the server. The client performs two main actions. He either sends a command to the robot, or creates a new command for him that he can later choose to use. He sends his request to the server, which invokes the classes KarelRobot or KarelHandbook depending on the given task.

4.4.1 Communication Model

When the client sends the command through the form on the web page. The server sends the request to the handleMsg() function (Figure:4.6). The message converts the request to a String, and checks if the String is a known command in Karel's or KarelHandbook's arsenal. When it finds a match, in this case for command "STEP" it would be a default command from KarelRobot, it calls the instance in the code of KarelRobot to execute the appropriate method for the command. KarelRobot calls "command_step()" which inside performs the check if the block in front of Karel is empty. Once it confirms, it updates the coordinates of Karel on the board state. Once KarelRobot is finished, "handleMsg" constructs the updated web page according to the commands passed by KarelRobot. Once it is done, the server calls the "send()" command which takes the updated web page as a String and displays it to the user as a web page.

Figure 4.5: Model of use cases

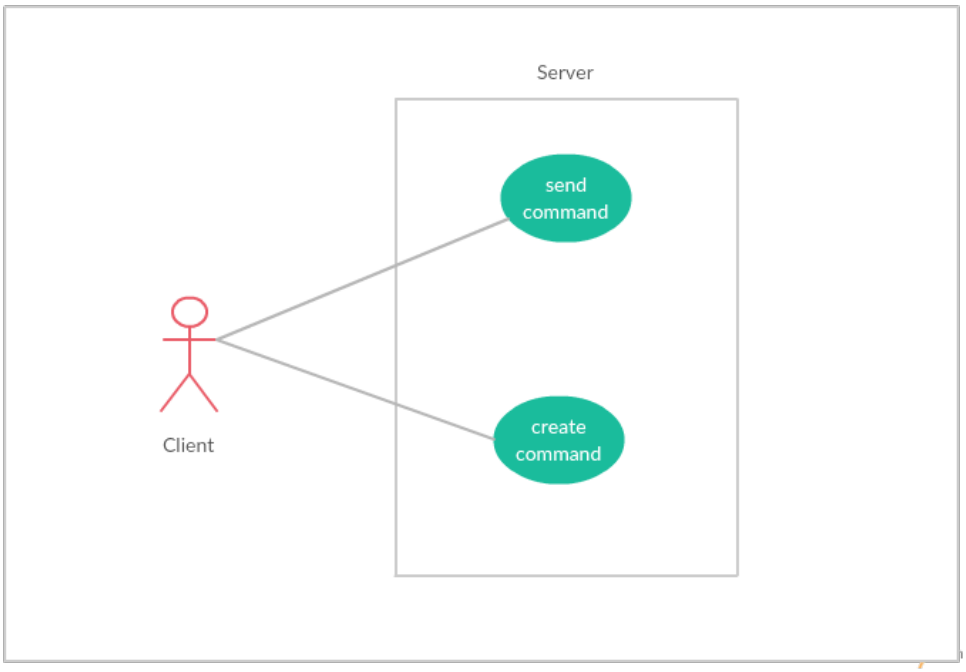
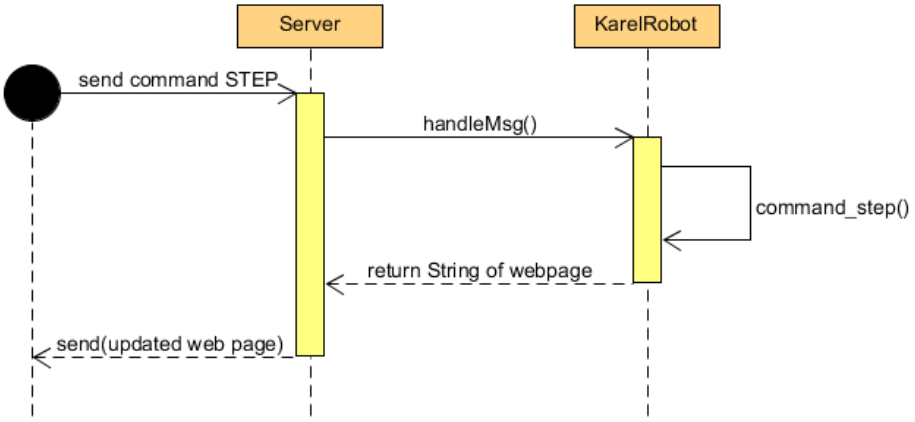


Figure 4.6: Sequence model of sending command "STEP"



Chapter 5

Tests

To check the execution of the application, I tested it on predefined settings to see how it operates, and recorded my results. To confirm the process of the system being able to receive and understand new commands, I have tried two tests. A rather simple test checking whether a simple command can be understood, and a more advance test using multiple custom commands.

5.1 Simple commands

By the default setting of the application, if we would want to turn the robot to the right from the direction he is facing, we would need to send the "LEFT" command three times. Since this can be tedious, we can create a custom command of our own called "RIGHT", which would automatically turn the robot to the right when called by calling the "LEFT" command three times. There are two ways to do this.

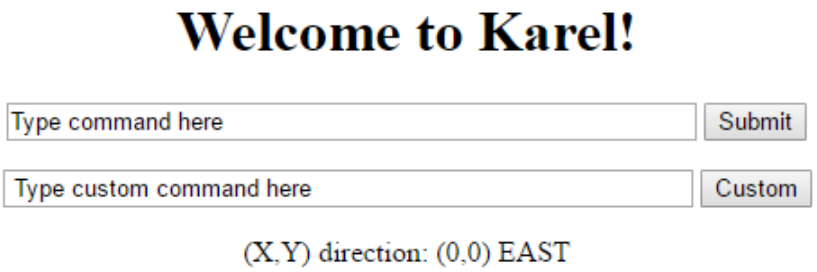
We could create the "RIGHT" command as a compound command of simply executing "LEFT" three times. This would be a compound command of default commands being executed sequentially to get our desired results. The syntax for such a command would look something like this:

```
RIGHT
    BEGIN
        LEFT
        LEFT
        LEFT
    END
```

To submit it to the Karel we click the "Custom" button and wait for our command to be processed (Figure:5.1).

Figure 5.1: Submitting of new command RIGHT

Figure 5.2: Output of RIGHT



To send it, we type in our new command "RIGHT" and click "Submit". The Karel learned our new command, and the new direction the robot is facing is now East, since he was facing North at the beginning (Figure:5.2).

We could also take a different approach, by using the "ITERATE" command from the Karel syntax. This would allow us to give a number of repetitions we would want, and the command which we want to repeat. In this case "RIGHT". To accomplish this the command would look like this:

```
RIGHT_2
    BEGIN
        LEFT
        LEFT
        LEFT
    END
```

The same way like before, we input the command to the Karel (Figure:5.3. After creating the command, we submit it to the Karel to execute it. The Karel correctly understands the command (Figure:5.4, and now changes his direction to South, since he was facing East previously. The Karel is able to understand and execute simple commands that abide by the syntax.

Figure 5.3: Submitting of new command RIGHT_2

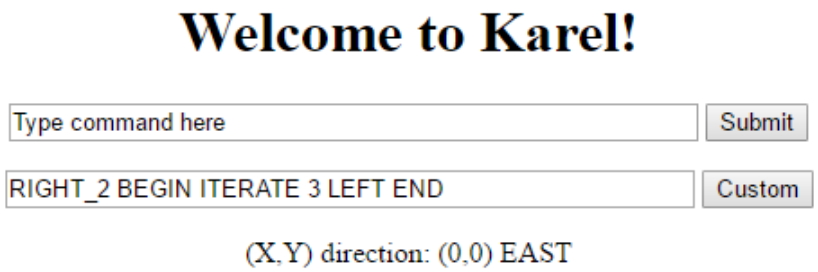
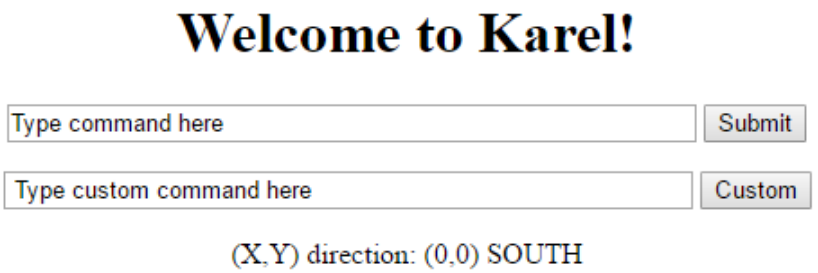


Figure 5.4: Output of RIGHT_2



5.2 Complex commands

To show that Karel can understand even more complex commands, and combine them together we will try to create a command that will resemble a program in which Karel will have to walk to the flag, which will be placed in a predefined location, and pick it up to read the text. The predefined conditions are: The size of the world will be 3x3 ranging from 0 to 3. The flag will be placed on the position of (2,2). Karel start from the position (0,0) facing North. A command that could lead us to the flag could look like this:

```
getFlag
    BEGIN
        WHILE CHECK_FRONT STEP
        RIGHT
        WHILE CHECK_FRONT STEP
        IF CHECK_FLAG PICK_UP
    END
```

This command instructs the robot to move North until he reaches a wall by using the "WHILE" command to execute the command "STEP" while there is an empty block in front of him. Once he reaches it, we will use the

Figure 5.5: Execution of complex command getFlag

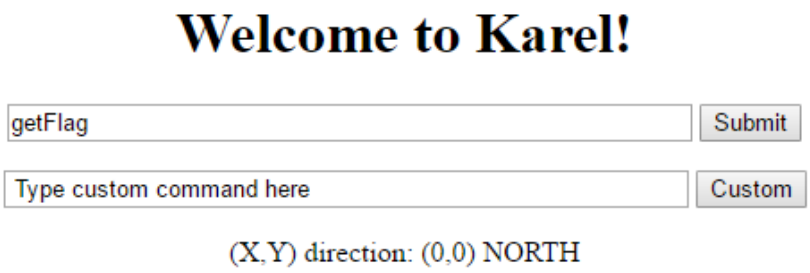
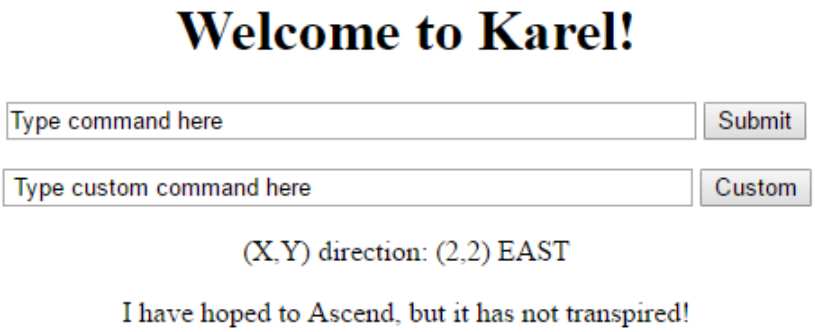


Figure 5.6: Output of getFlag



previously defined command "RIGHT" to turn him facing the flag. Again, we will make him walk until he reaches the wall. Once there, he will check if he is on the position of the flag, and try to pick it up. If he is successful, the server will display the secret message for us. We submit our newly defined command and wait for the output (Figure:5.5).

The robot has successfully interpreted our command getFlag, and has moved as expected, picking up the flag in the end. He reads the secret text on the flag which is displayed on the web page (Figure:5.6).

Conclusion

In summation, the goal of the thesis was to implement and develop a Karel robot that exists on a server provided by the ESP8266 WiFi chip, with whom clients can interact. The clients communicate with the robot by giving it commands which the Karel executes. Along with the default commands, clients can create custom ones in the form of specified Karel language. The written part of the thesis shows the capabilities of the chip, Karel's language and the design of the application.

Results

Working with the Arduino ESP8266 WiFi chip and the libraries provided, I have to acknowledge that the tools offered at the developers disposal are easy to use and understand if their is some prior knowledge regarding the basic concepts of programming exists. They enable the development of a wide variety of projects, and leave a lot of space for experimentation. However, since the libraries are open source, a lot of the documentation is not properly described, making it sometimes hard to understand. Working with the chip itself can be difficult, luckily there are a lot of guides provided by community members, showing how the connections on the chip should be made, with the procedures explained step by step. The application on the chip works consistently. However, there is an issue regarding computers with a strong anti-virus connecting to the chip, because the defence system sometimes disables it. Phones on the other hand have no problems establishing and keeping the connection.

Recommendations

I would definitely recommend the Arduino chips and boards provided to anyone interested in developing their own project. There is a vast number of options, and the ESP8266 is not necessarily perfect fit for just any project,

so carefully going through all the options would be a good idea. Despite its drawbacks, it served the purpose of this project really well. I am looking forward to exploring more about the Arduino universe, perhaps developing a new project on a different chip. The examples provided by the Arduino IDE are a solid starting point even for those not familiar with programming, as they slowly reveal the possibilities of what the chips can do. Another great place to visit would be the Arduino forums where anyone can get involved in the community and ask for help or contribute.

Bibliography

[1] Arduino ESP8266 documentation. Available from: https://github.com/esp8266/Arduino

[2] Team, E. S. I. ESP8266EX Datasheet. 2015. Available from: http://download.arduino.org/products/UNOWIFI/0A-ESP8266-Datasheet-EN-v4.3.pdf

[3] ESP8266 overview. Available from: https://espressif.com/en/products/hardware/esp8266ex/overview

[4] 14core Editor. FLASHING UPGRADE EPS8266 V1 WITH ESPRESSIF FLASH TOOL. Available from: http://www.14core.com/flashing-upgrade-eps8266-v1-firmware-with-espressif-flash-tool/

[5] AT instruction set. Available from: https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

[6] Richard E Pattis biography. Available from: https://www.ics.uci.edu/~pattis/

[7] Karel Capek biography. Available from: https://www.britannica.com/biography/Karel-Capek

[8] CAPEK, K. *Rossum's Universal Robots*. Createspace Independent Publishing Platform, ISBN 9781542552073.

[9] Owen-Hill, A. A History of Robot Programming Languages. May 2016]. Available from: http://blog.robotiq.com/the-history-of-robot-programming-languages

[10] Karel programming language documentation. Available from: http://mormegil.wz.cz/prog/karel/prog_doc.htm

[11] Joseph Bergin, J. R., Mark Stehlik; Pattis, R. A Gentle Introduction to the Art of Object-Oriented Programming in Java. In *Karel J Robot*, 2013.

Appendix A

Acronyms

WiFi Wireless Local Area Networking

SOC System on chip

TCP/IP Transmission Control Protocol/Internet Protocol

HTTP Hypertext Transfer Protocol

mDNS multicast Domain Name System

SSDP Simple Service Discovery Protocol

DNS Domain Name System

OTA Over The Air

SD Standard Definition

RF Radio Frequency

SRAM Static Random Access Memory

GPIO General Purpose Input/Output

MCU Micro Controller Unit

RSIC Resilient Sound Isolation Chip

CPU Central Processing Unit

AHB Advanced High-performance Bus

IDE Integrated Development Environment

SPI Stateful Packet Inspection

MD5 Message Digest 5

SDIO Secure Digital Input/Output

802.11 b/y/n/e/i WLAN MAC Wireless standard defined by IEEE

P2P Peer to Peer

RTS/CTS Request to Send / Clear to Send

VCC Power Supply Pin

RST Reset pin

GND Ground pin

CHPD Chip Select pin

TX Transmitter

RX Receiver

SPIFFS File System Object

STA Station

AP Access Point

SSID Service Set Identifier

USB Universal Serial Bus

HTML Hypertext Mark-up Language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
  - `karel` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `BP_Stefan_Ćirić_2017.pdf` ..... the thesis text in PDF format
  - `BP_Stefan_Ćirić_2017.ps` ..... the thesis text in PS format