



TEI of CRETE

Department of Applied Informatics and
Multimedia



Fachhochschule Düsseldorf

University of Applied Sciences

Thesis

*“Σχεδιασμός και υλοποίηση μιας γενικής απομακρυσμένης ελεγκτικής διεπαφής
για SDK-ρομποτικά συστήματα”*

Author: Minardou Aikaterini

2261

Date: 10/04/2013

Professors: Prof. Dr. rer. nat. Gundula Dörries,

Prof. Dr. Eng. Andreas Vlissidis

A big “Thank you” to those
who believed in me and kept
me going

Abstract

The purpose of this project was to create a control interface for a specific robot, which with some modification can apply and control other robots as well. The robot that was used for this project was an Arexx RP6 equipped with the RP6v2 M256 Wi-Fi expansion module in order to expand the module capabilities to connect with a WI-FI network. Since smartphones and tablets are becoming a part of people's daily life and the expansion of their abilities/use is a constant research subject, developing the control interface as a smart device application was the smart thing to do.

The thesis is divided into two parts, which encompass the programming of the robot and the development of the smartphone application. The robot's microcontrollers (Main Boards ATMEGA32 and the WI-FI module ATMEGA2560) are programmable in C language. The main objective was to enable the robot to listen to every command that was sent via the network, received by the WLAN module and sent to the control interface messages like "Obstacle ahead", "Left(/Right) Bumper was hit" and the values from the light sensors. The controlling interface was developed for Android Devices in Java Android and provides a user interface for interacting with the robot. The Control Interface is quite versatile and with a few modifications other robots, like the humanoid robot NAO or the helicopter AR.Drone, can be programmed and controlled by it.

Abstract (Ελληνικά)

Ο σκοπός αυτής της εργασίας ήταν να δημιουργηθεί ένα περιβάλλον ελέγχου για ένα συγκεκριμένο ρομπότ, το οποίο με μερικές αλλαγές μπορεί να χρησιμοποιηθεί για τον έλεγχο άλλων ρομποτικών συστημάτων. Το ρομπότ που χρησιμοποιήθηκε για την εργασία αυτή ένα Arexx RP6 εξοπλισμένο με μία RP6v2 M256 Wi-Fi μονάδα επέκτασης, προκειμένου να επεκτείνει τις δυνατότητες της μονάδας και να μπορεί να συνδεθεί με ένα δίκτυο μέσω WIFI. Δεδομένου ότι τα έξυπνα τηλέφωνα (smartphones) και οι tablets συσκευές, έχουν γίνει μέρος της καθημερινότητας μας και η συνεχής προσπάθεια επέκτασης των δυνατοτήτων και της χρήσης τους έχει γίνει ένα βασικό θέμα έρευνας, το να αναπτυχθεί αυτό το περιβάλλον ελέγχου για μια τέτοια συσκευή ήταν η καλύτερη δυνατόν κίνηση. Η πτυχιακή χωρίζεται σε δύο μέρη, τα οποία περιλαμβάνουν τον προγραμματισμό του ρομπότ και την ανάπτυξη της smart συσκευής.

Οι Microcontrollers του ρομπότ (βασικής μονάδας ATMEGA32 και του WI-FI module ATMEGA2560) προγραμματίζονται σε C γλώσσα προγραμματισμού. Ο κύριος στόχος ήταν να μπορεί το ρομπότ να ακούσει κάθε εντολή που εστάλη μέσω του δικτύου, η οποία ελήφθη από τη μονάδα WLAN και να «απαντάει» στην συσκευή με μηνύματα όπως «Εμπόδια Μπροστά» , «Χτυπήθηκε ο Αριστερός/Δεξής Αισθητήρας», τις τιμές τους αισθητήρες φωτός καθώς και ενημερώσεις για το υπόλοιπο της μπαταρίας.

Η εφαρμογή αναπτύχθηκε για Android συσκευές σε Java γλώσσα προγραμματισμού και παρέχει στον χρήστη ένα περιβάλλον εργασίας για την επικοινωνία με το ρομπότ. Η εφαρμογή είναι αρκετά ευέλικτη και με μερικές αλλαγές μπορεί να χρησιμοποιηθεί και για άλλα ρομπότ όπως το ανδροϊδές NAO ρομπότ ή το AR Drone τα οποία μπορούν να προγραμματιστούν και να ελεγχθούν από αυτή.

Table of Contents

| | |
|--|-----|
| Abstract | ii |
| Abstract (Ελληνικά) | iii |
| Introduction | 1 |
| Chapter 1: The Robot | 3 |
| 1.1 General for the Robot | 3 |
| 1.2 Robot's Components | 4 |
| 1.3 RobotLoader | 5 |
| 1.4 Base Robot Programming | 6 |
| 1.4.1 Master/Slave Communication Protocol..... | 6 |
| 1.4.2 I ² C Slave Program | 6 |
| 1.4.3 Compile the Source Code..... | 6 |
| 1.4.4 Upload the HEX file..... | 10 |
| Chapter 2: The WIFI Expansion Module | 11 |
| 2.1 General for the WIFI Module..... | 11 |
| 2.2 WIFI Module's Components..... | 12 |
| 2.3 WIFI Configuration..... | 13 |
| 2.4 WIFI Module Programming | 14 |
| 2.4.1 Included libraries | 14 |
| 2.4.2 WiFi Control Function | 14 |
| 2.4.3 Main Function | 16 |

| | |
|---|----|
| 2.5 Compile the Source Code..... | 16 |
| 2.6 Connection and Upload..... | 17 |
| Chapter 3: The Android App..... | 19 |
| 3.1 About the Application..... | 19 |
| 3.2 Few words about Java Android..... | 20 |
| 3.2.2 Development Tools..... | 20 |
| 3.3 Design..... | 21 |
| XML Source Code..... | 21 |
| 3.4 TCP Socket Programming..... | 23 |
| 3.5 Connect Tab..... | 26 |
| 3.6 Main Activity..... | 28 |
| 3.6.1 Different Messages Scenarios..... | 32 |
| Limitations..... | 34 |
| Conclusion..... | 35 |
| Discussion..... | 36 |
| References..... | 38 |
| Bibliography..... | 39 |

| | |
|--|----|
| Figure 1: Design | 2 |
| Figure 2: RP6 Robot Source: http://goo.gl/TruQX | 3 |
| Figure 3: RP6 Robot, Source: http://goo.gl/dVJLc | 4 |
| Figure 4: RP6 Robot Loader | 5 |
| Figure 5: Status box..... | 5 |
| Figure 6: Screenshot from makefile (1)..... | 7 |
| Figure 7: Screenshot from makefile (3)..... | 7 |
| Figure 8: Cygwin Screenshot (1)..... | 8 |
| Figure 9: Cygwin Screenshot (2)..... | 9 |
| Figure 10: Cygwin Screenshot (3)..... | 9 |
| Figure 11: Add/Select Box | 10 |
| Figure 12:M256 WIFI Expansion Module Source: http://goo.gl/o6q5h | 11 |
| Figure 13: Network representation; Source: RP6 M256 WIFI Module manual | 12 |
| Figure 14: WIFI Configuration Dialog..... | 13 |
| Figure 15: RP6M256 Libraries..... | 14 |
| Figure 16: Incoming line | 14 |
| Figure 17:WIFI Control State..... | 15 |
| Figure 18:Main Function's loop | 16 |
| Figure 19:WIFI Remote- Makefile..... | 17 |
| Figure 20: WIFI Loader | 18 |
| Figure 21: Application..... | 19 |
| Figure 22: Eclipse Package Explorer | 21 |
| Figure 23: Connect Dialog | 22 |
| Figure 24: Screenshot from main_activity.xml file..... | 23 |
| Figure 25: Screenshot of the manifest.xml file | 23 |
| Figure 26: Connect to Server Function | 24 |
| Figure 27: Threads run () function | 24 |
| Figure 28: Reading/Writing Code | 25 |
| Figure 29: Try...Catch Block..... | 25 |
| Figure 30: Disconnect From Server | 25 |
| Figure 31: Connect Tab (1) - Source Code | 26 |
| Figure 32: Connect Tab (2) - IP& Port control | 26 |
| Figure 33: Connect Tab (3) - accurate IP function..... | 26 |
| Figure 34: Connect Tab (4) | 27 |
| Figure 35: Connect Task - AsyncTask | 28 |
| Figure 36: onCreate method - Control Panel..... | 28 |
| Figure 37: Handler Messages | 29 |
| Figure 38: Selection a menu item..... | 30 |
| Figure 39: Help Dialog..... | 30 |
| Figure 40: After the user disconnected..... | 31 |
| Figure 41: After Disconnect is pressed | 31 |
| Figure 42: Low Battery message..... | 32 |
| Figure 43: Obstacle detected | 32 |
| Figure 44: Bumper Hit | 33 |

Figure 45: Wrong IP or Port number..... 33
Figure 46: Suggestion..... 36

Introduction

The purpose of this project was to create a control interface for the Arexx RP6 robot, which with some modifications it can be applied and controlled other robots of the same class as well. The base robot doesn't support WIFI connectivity; it is on the other hand possible to control it via a TV Remote control since it can receive IR codes. In order to expand its capabilities an expansion module is attached to the main board, the RP6v2 M256 Wi-Fi module. With a second microcontroller ATMEGA 2560 and a separate 802.11 b/g WLAN module is possible to connect to a WIFI network, receive and transmit commands via the WLAN module while the microcontroller, connected via I²C bus to the main robot and/or other expansion modules, act as a master (from the master/slave communication protocol) and control the slaves (other components.).

The robot's microcontrollers are programmable in C language and the manufacturer (Arexx and Conrad Electronics) provide a big library of functions, for the main robot and the expansion modules, and a software called RobotLoader, a Java platform application that allows the user to upload programs on the microcontroller, either via the PC USB Interface (mainly for the main board) or upload it wirelessly. In company with the previous components examples are provided for the better understanding of the functions. One of them is a WIFI Remote example that controls the robot by sending commands through the WIFI terminal of the RobotLoader.

Since smartphones and tablets are becoming a part of people's daily life and the expansion of their abilities/uses is a constant research subject, developing the control interface as a smart device application is the best approach.

The thesis is divided into two parts, which encompass the programming of the robot and the development of the smartphone application. The main objective was to program the robot to listen to every command that was sent via the network, received by the WLAN module and reply to the control interface messages as "Obstacle ahead", "Left/(Right) Bumper was hit", information about the light density in the area from the light sensors and constantly updates for the battery level.

The controlling interface was developed for Android Devices in Java Android and provides a user interface for interacting with the robot. According to the received messages the application has to be capable to inform the user for every one of them. The Control Interface is quite versatile and with a few modifications other robots, like the humanoid robot NAO or the helicopter AR.Drone, can be programmed and controlled by it.

The WLAN module of the RP6's expansion module is configured to have an IP and a communication Port. In order to receive commands from another device (in this case the Android application) the device has to be programmed as a TCP Socket. *The Transmission Control Protocol (TCP) is one of the two original core protocols of the Internet protocol suite (IP), and is so ubiquitous that the entire suite is often called TCP/IP. TCP provides reliable, ordered, error-checked delivery of a stream of octets between programs running on computers connected to an intranet or the public Internet.*¹ A socket is one end-

¹ Source: http://en.wikipedia.org/wiki/Transmission_Control_Protocol

point of a two-way communication link between two programs running on the network. Socket classes are used to represent the connection between a client program and a server program.²

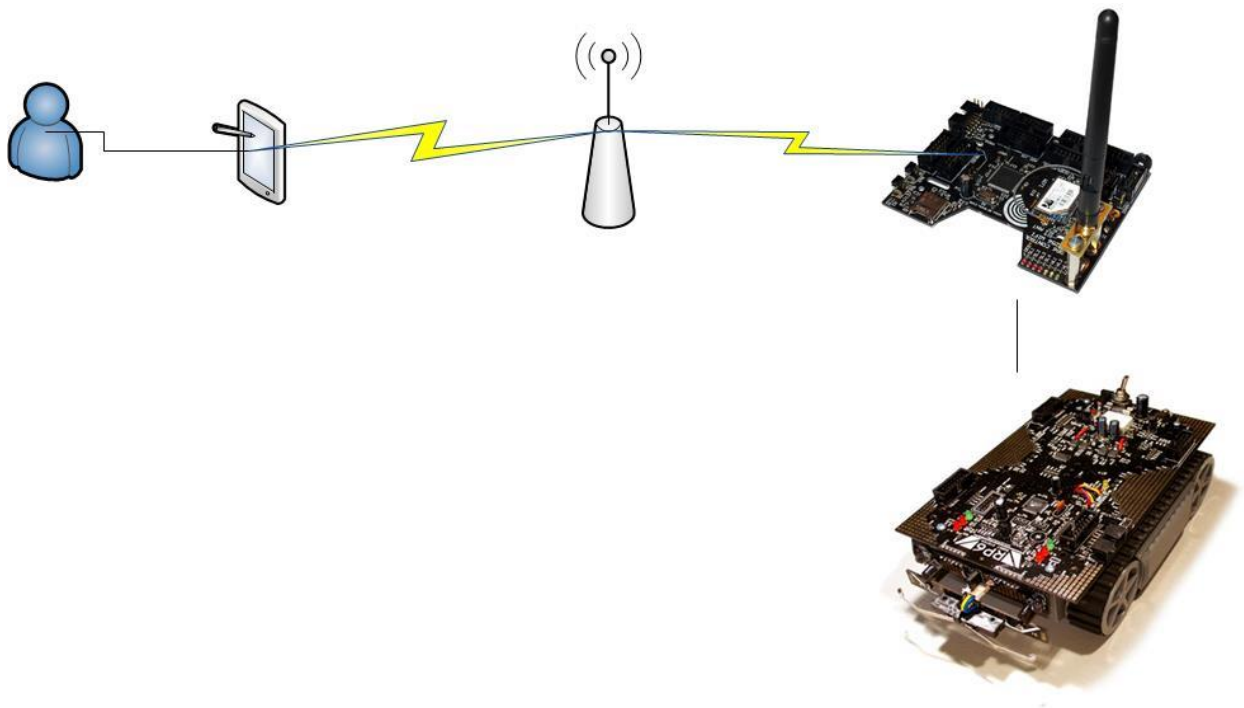


Figure 1: Design

² Source: <http://docs.oracle.com/javase/tutorial/networking/sockets/>

Chapter 1: The Robot

1.1 General for the Robot

The robot that was used in this project is an Arexx RP6 Robot, which was created by Arexx in collaboration with Conrad Electronics. It is a low cost autonomous robot, equipped with an ATMEGA32-8 bit RISC (Reduced instruction set computing) Microcontroller. Programmable in C language, it is accompanied with function libraries, a very extensive manual and lots of example programs, which give the opportunity to the user to better understand the capabilities of the Robot.

Using the I²C (Inter-Integrated Circuit) Bus and master-slave communication protocol-based programs, expansion modules can either control the Robot or be controlled by it.³ It is powered by 6 1.5Volt batteries, stored under the main board, which if are NiHM can be charged via the charger connector.

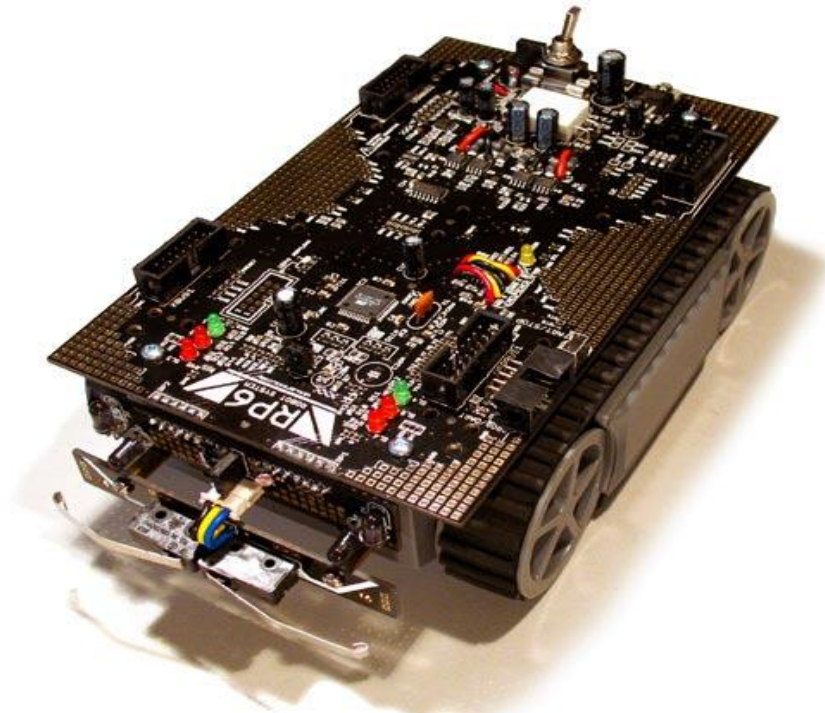


Figure 2: RP6 Robot Source: <http://goo.gl/TruOX>

³ More information on Chapters 1.3 and 2.4

1.2 Robot's Components

As it can be seen in the following figure the Robot offers a big variety of components. More Sensors can be added either by soldering them in the main board or by soldering them on the expansion module (experimental board)⁴.

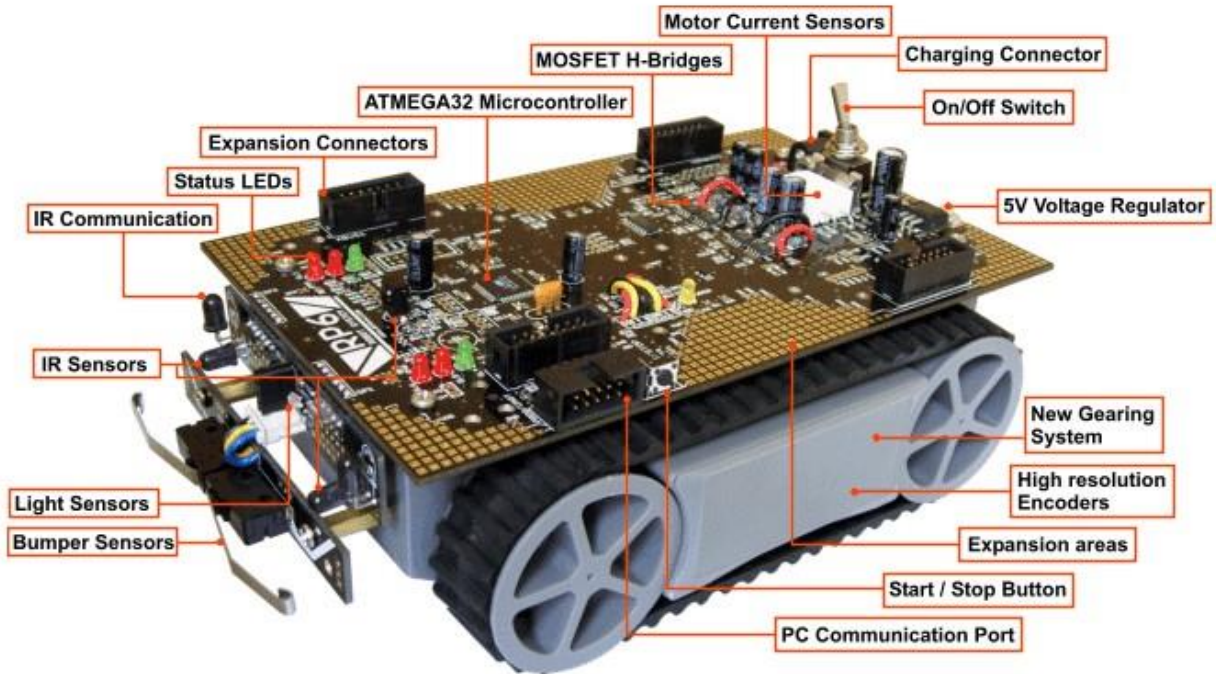


Figure 3: RP6 Robot, Source: <http://goo.gl/dVJLc>

Since the purpose of the project is controlling the robot, the robot has to be programmed to send back feedback regarding its condition; for example the user must be notified if an obstacle was detected or if a bumper was hit. The sensors that are going to be used are:

- * Light Sensors
- * Bumper Sensors
- * IR(InfRared) Sensors
- * Commands to request the Battery Level

⁴ Expansion Board: more info <http://www.arexx.com/rp6/html/en/acc.htm>

1.3 RobotLoader

For uploading the program to the robot the manufacturer provides us with Software called Robot Loader Version 2.4.⁵ The software is developed in Java programming language and it “has been developed to allow easy access and program upload to the robot’s microcontroller and all its expansion boards”⁶. Connect the Robot with the PC via the PC communicator port. When the suite recognizes and displays the robot as a possible connection and we have spotted the USB port that is connected to the connection van be established by pressing the Connect button.

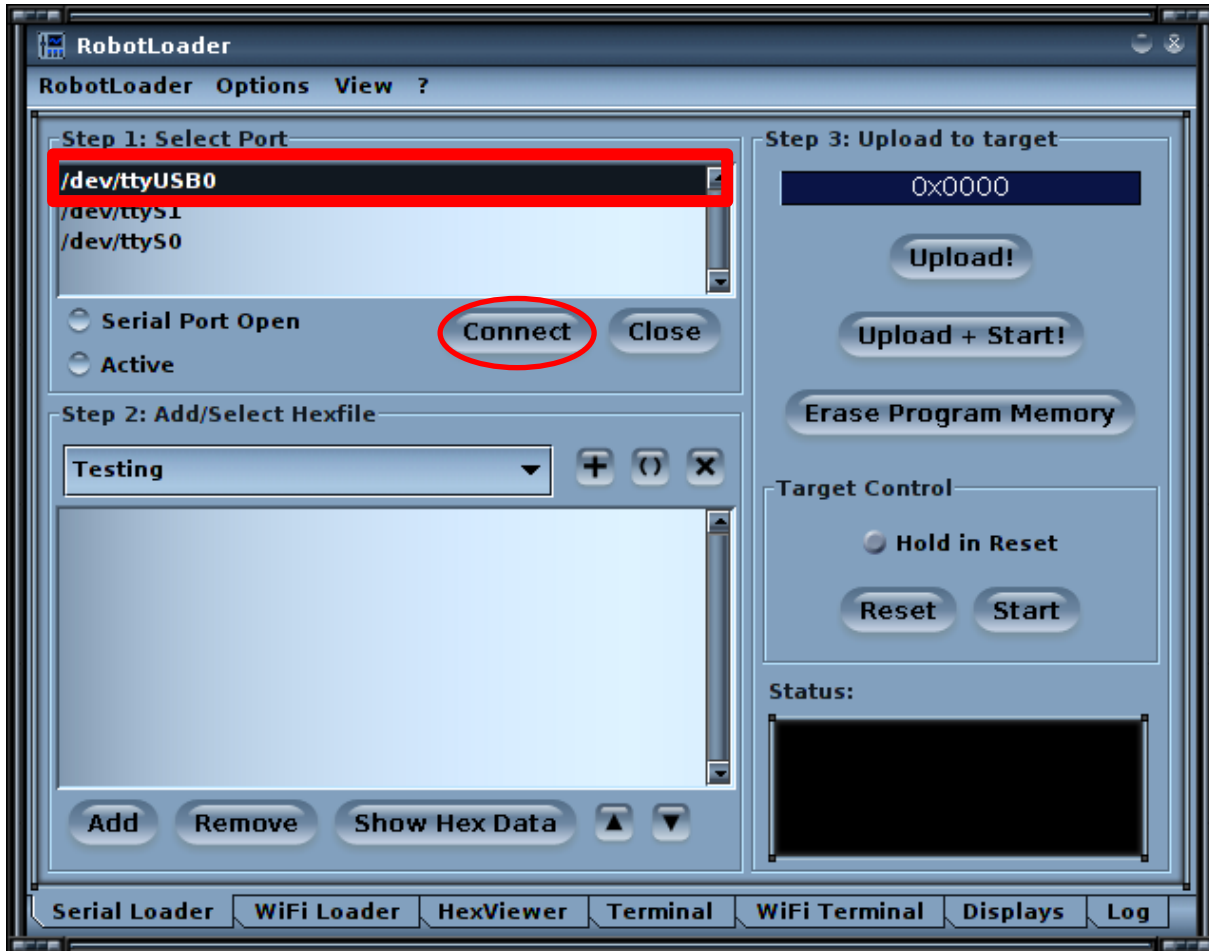


Figure 4: RP6 Robot Loader

When the connection is established a “success” message is displayed in the *Status* box:

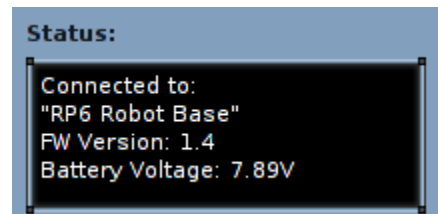


Figure 5: Status box

⁵ Can be downloaded <http://www.arexx.com/rp6/html/en/software.htm>

⁶ Source: <http://goo.gl/kP7Pe>

Since the connection is established we can move on with the program upload.

1.4 Base Robot Programming

Not forgetting the fact that the base robot doesn't have the components to support a wireless communication, the base robot has to be programmed after considering the I²C communication protocol and program it as a slave to respond to the received commands from the master. Such kind of program is offered by the manufacturer.

1.4.1 Master/Slave Communication Protocol

Master/Slave Communication Protocol is a model for a protocol which allows one device (master) to control one or more devices (slaves).⁷ I²C is a protocol that synchronizes the data flow whenever there is a change in the bus. Every slave is associated with a byte sequence. The master sends a byte before the command to all the slaves and only the slave with the same byte sequence receives and translates the data. The other slaves just cancel the communication waiting for another group of data to come. Finally the master issues a stop condition to terminate the communication.⁸

1.4.2 I²C Slave Program

The main idea of this program is: a master device controls the functionality of the robot and sends commands to the slave via the I²C Bus port. For example the master transmits the Command "Move fwd"; the slave receives the message and acts accordingly. Naturally those tasks can be interrupted if for example, an obstacle is detected ahead or a bumper was hit. Furthermore "information" functions are running as well in the same loop, sending the master information that was gathered by the Light Sensor.

1.4.3 Compile the Source Code

The ATMEGA32 Microcontroller that the RP6 has embedded reads only hexadecimal type of files. The programs are written in C using Programmer's Notepad⁹ or Kate Editor¹⁰. Using Cygwin¹¹ (or the

⁷ Source: <http://goo.gl/kP7Pe>

⁸ Embedded C Programming the Microchip Pic von Richard H. Barnett, Larry D. O'Cull, Sarah A. Cox

⁹ Website : <http://goo.gl/GwjW>

¹⁰ Website : <http://goo.gl/KIUyD>

¹¹ Website : <http://goo.gl/MX63>

Terminal if using Linux) we compile and generate the .hex file that the microcontroller reads with the *make* command.

Using the *make* command the compiler generates the files that are instructed in the *makefile*. One of them is the hexadecimal file that we need to upload to the Microcontroller. In the official documentation, it is suggested to use an already created *makefile* for the developing of a new program. The creation of a makefile is too complicated, but using one that is already made and by simply applying some small changes, that we will see next, makes it easier.

According to the name that the .c format *makefile* has, only one thing has to be changed¹²; the Target:

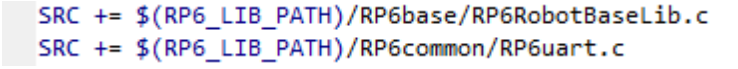


```
TARGET = RP6Base_I2CSlave
```

Figure 6: Screenshot from makefile (1)

The Target has to be changed to the same name as our .c file, for example if our file is called RP6Base_I2CSlave the Target point changes to TARGET = RP6Base_I2CSlave. No Spaces and no extension .c are required, since this command: `SRC = $(TARGET).c` follows.

This part must not change as well:



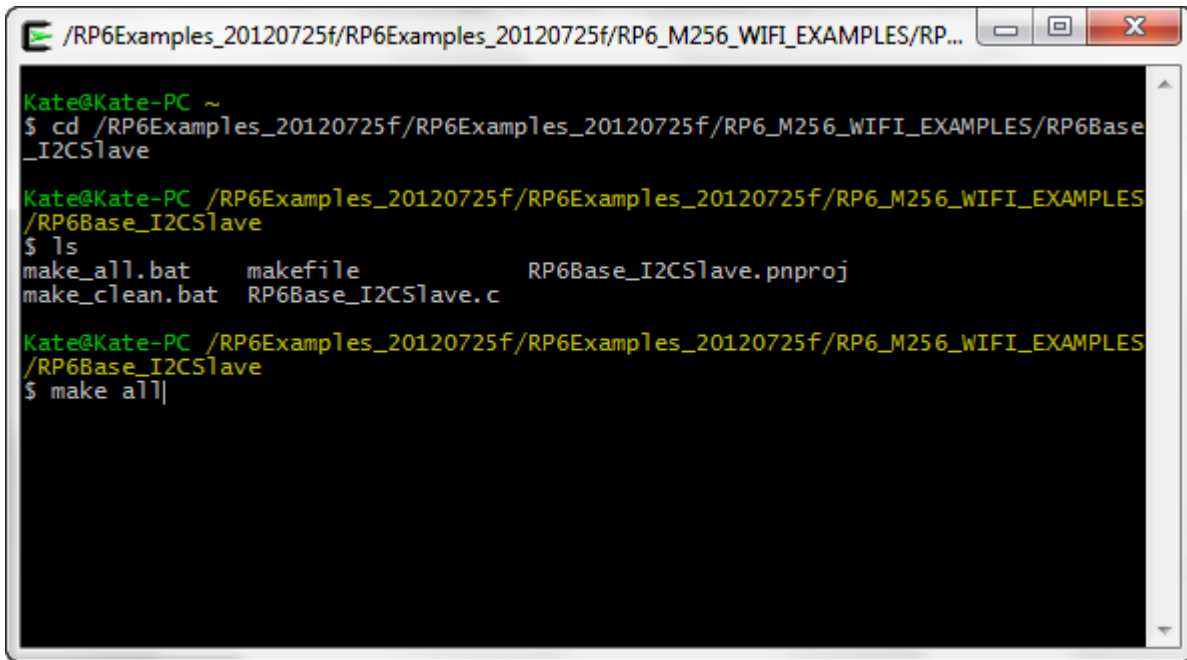
```
SRC += $(RP6_LIB_PATH)/RP6base/RP6RobotBaseLib.c  
SRC += $(RP6_LIB_PATH)/RP6common/RP6uart.c
```

Figure 7: Screenshot from makefile (3)

These lines provide the path for the RP6 Libraries.

After all the steps that are mentioned above are completed, we can open Cygwin (or Linux Terminal) and run the *make all* command:

¹² **Only** and **if** necessary!

A screenshot of a Cygwin terminal window. The window title is "/RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP...". The terminal shows the following commands and output:

```
Kate@Kate-PC ~  
$ cd /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave  
Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave  
$ ls  
make_all.bat      makefile          RP6Base_I2CSlave.pnproj  
make_clean.bat   RP6Base_I2CSlave.c  
Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave  
$ make all|
```

Figure 8: Cygwin Screenshot (1)

Cygwin will inform the user that the compile has finished. If an error occurs during the process, it will point out the line number, the compiling stopped and what is the error.


```

Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave
$ make all

----- begin -----
avr-gcc (WinAVR 20100110) 4.3.3
Copyright (C) 2008 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

Compiling: RP6Base_I2CSlave.c
avr-gcc -c -mmcu=atmega32 -I. -gdwarf-2 -Os -funsigned-char -funsigned-bitfields
-fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=RP6Base_I2C
Slave.lst -I../RP6Lib -I../RP6Lib/RP6base -I../RP6Lib/RP6common -std=gnu99 -MD -MP -MF .dep/RP6Base_I2CSlave.o.d RP6Base_I2CSlave.c -o RP6Base_I2CSlave.o

Linking: RP6Base_I2CSlave.elf
avr-gcc -mmcu=atmega32 -I. -gdwarf-2 -Os -funsigned-char -funsigned-bitfields
-fpack-struct -fshort-enums -Wall -Wstrict-prototypes -Wa,-adhlns=RP6Base_I2CSlave.o -I../RP6Lib -I../RP6Lib/RP6base -I../RP6Lib/RP6common -std=gnu99 -MD -MP -MF .dep/RP6Base_I2CSlave.elf.d RP6Base_I2CSlave.o ../RP6Lib/RP6base/RP6RobotBaseLib.o ../RP6Lib/RP6common/RP6uart.o ../RP6Lib/RP6common/RP6I2CSlaveTWI.o --output RP6Base_I2CSlave.elf -w1,-Map=RP6Base_I2CSlave.map,--cref -lm

Creating load file for Flash: RP6Base_I2CSlave.hex
avr-objcopy -O ihex -R .eeprom RP6Base_I2CSlave.elf RP6Base_I2CSlave.hex

Creating Extended Listing: RP6Base_I2CSlave.lss
avr-objdump -h -S RP6Base_I2CSlave.elf > RP6Base_I2CSlave.lss

Creating Symbol Table: RP6Base_I2CSlave.sym
avr-nm -n RP6Base_I2CSlave.elf > RP6Base_I2CSlave.sym

Size after:
AVR Memory Usage
-----
Device: atmega32

Program: 8650 bytes (26.4% Full)
(.text + .data + .bootloader)

Data: 261 bytes (12.7% Full)
(.data + .bss + .noinit)

----- end -----

Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave
$ ls
make_all.bat          RP6Base_I2CSlave.elf  RP6Base_I2CSlave.map
make_clean.bat       RP6Base_I2CSlave.hex  RP6Base_I2CSlave.o
makefile             RP6Base_I2CSlave.lss  RP6Base_I2CSlave.pnproj
RP6Base_I2CSlave.c  RP6Base_I2CSlave.lst  RP6Base_I2CSlave.sym

```

Figure 9: Cygwin Screenshot (2)

The files that were created can be viewed with the *ls* command.

```

----- end -----

Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES/RP6Base_I2CSlave
$ ls
make_all.bat          RP6Base_I2CSlave.elf  RP6Base_I2CSlave.map
make_clean.bat       RP6Base_I2CSlave.hex  RP6Base_I2CSlave.o
makefile             RP6Base_I2CSlave.lss  RP6Base_I2CSlave.pnproj
RP6Base_I2CSlave.c  RP6Base_I2CSlave.lst  RP6Base_I2CSlave.sym

```

Figure 10: Cygwin Screenshot (3)

The hexadecimal can be located in the results.

1.4.4 Upload the HEX file

Going back to the Robot Loader, in the “Add/Select Hexfile” box we can locate the produced .hex file then select it and press the upload button under the “Upload to Target” box.

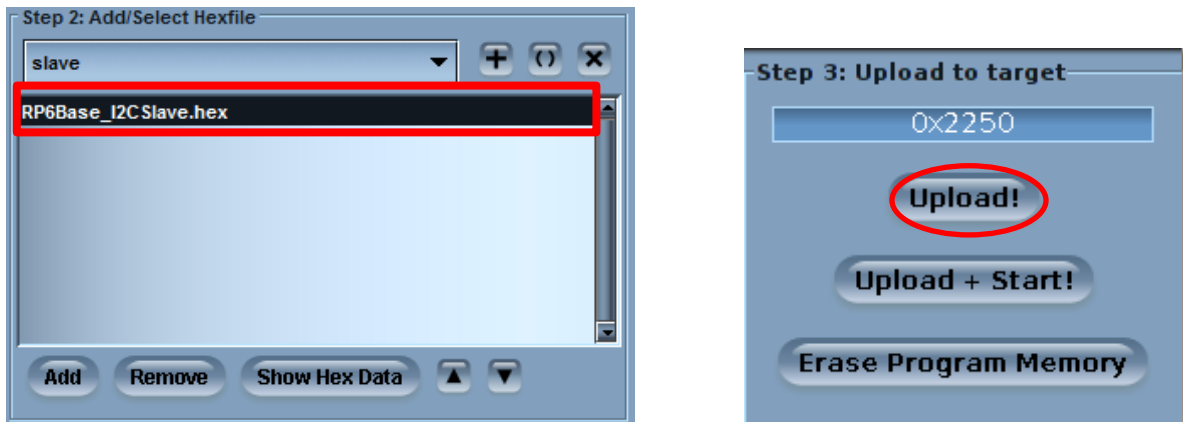


Figure 11: Add/Select Box

When the upload finishes the messages in the status box will change to: “Task Completed successfully”. After that the Robot base can be controlled via any master device we connect to it.

Chapter 2: The WIFI Expansion Module

2.1 General for the WIFI Module



Figure 12:M256 WIFI Expansion Module Source: <http://goo.gl/o6q5h>

The RP6v2 M256 WIFI Expansion Module gives the ability to the users to associate a RP6 robot to a Wireless computer network. Through this expansion module, Robot's remote control scenario and information exchange with another device, will be possible.

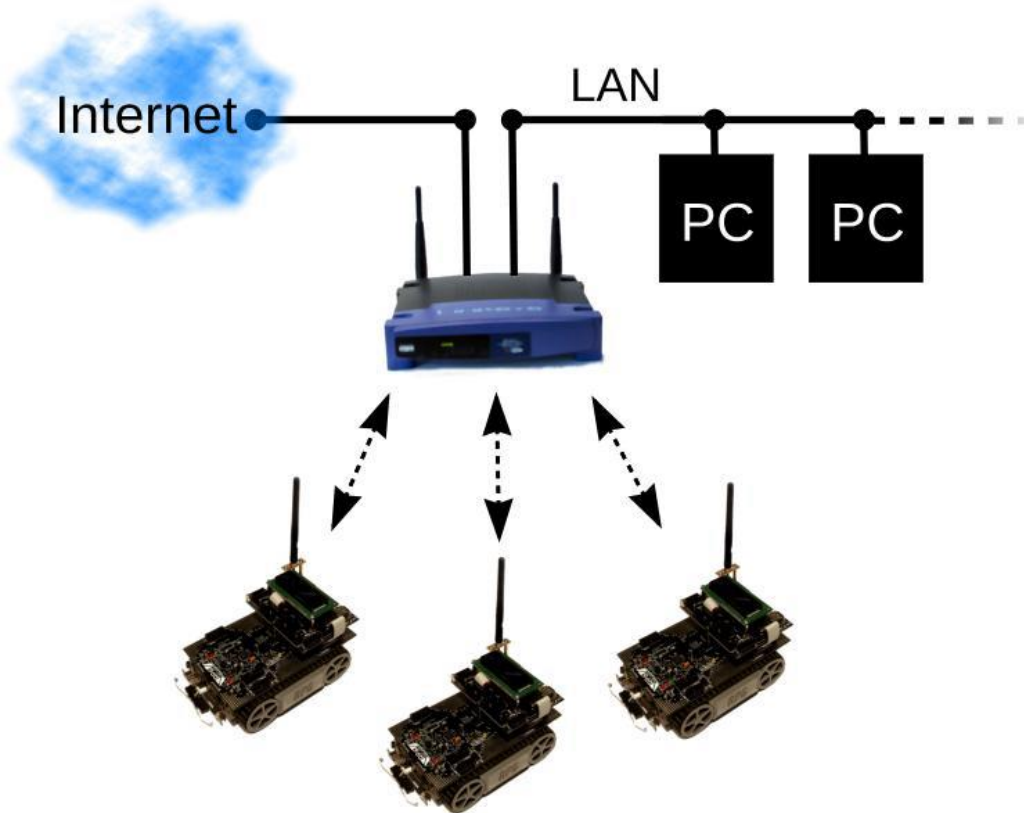


Figure 13: Network representation; Source: RP6 M256 WIFI Module manual

2.2 WIFI Module's Components

Equipped with an ATMEGA2560 Microcontroller, 256 KB flash ROM and 8KB SRAM gives to users the opportunity to construct more complex programs. The WLAN functions are managed by the Roving Networks RN-171 microchip, an energy efficient 802.11b/g WLAN module with built in processor. It also manages the TCP/IP stack and the AES/WAP2 encryption, something that relieves the AVR processor and makes the control for the user simpler. After power on, the module can connect automatically to the network. For storing bigger amount of data, the module provides a microSD card slot, ideal for data logging, mission data or eventually storage of static data that will be retrieved via WLAN. The I²C bus will be used, to connect the module with the main robot and supply it with power. For program upload¹³, the USB PC interface can be used or alternatively the WLAN module can be used for the upload, relieving the user from using cables.¹⁴

¹³ More info Chapter 2.4

¹⁴ Most of the information were collected from: RP6 M256 WIFI Module manual

2.3 WIFI Configuration

As it was mentioned previously, the WLAN module connects to the WIFI network automatically, moments after power on. In order for the connection to take place the WLAN module has to be configured, and that is possible only by using the USB serial interface. *“The settings are stored in the WLAN module and remain there after the module has been switched off”¹⁵*

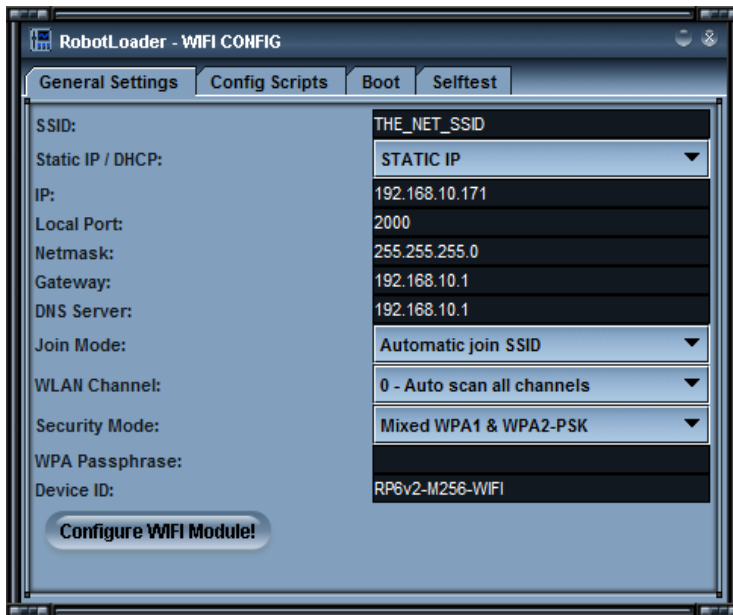


Figure 14: WIFI Configuration Dialog

password

By pressing the **Configure WIFI Module!** button the information is stored and a wireless connection can be established.

By opening the Discover WIFI connections the robot can be located and added to the connection list.

Important Fields:

- SSID: Service Set Identifier or in other words the name of a wireless network
- The robot IP has to be static (and that has to be configure in the router specifications)
- Set the robot's IP address
- Set a communication Port
- Netmask, Gateway and DNS Server have to be set according the routers configuration
- Security Mode: WPA 1 & 2 and WEP are supported but WPA is preferred for security reasons.
- WPA Passphrase: the WLAN's

¹⁵ RP6 M256 WIFI Module Manual

2.4 WIFI Module Programming

2.4.1 Included libraries

The RP6M256Lib.h had to be included so that the functions that give access to the M256 module components could be used. The RP6I2CmasterTWI.h was added due to the fact that it grants access to functions that initialize the TWI (Two Wire Interface of the I²C Bus communication protocol). The RP6M256_I2CMasterLib.h allows the program to control the Robot nearly the same as it was with RP6Lib.

```
#include "RP6M256Lib.h"          // M256 Lib

#include "RP6I2CmasterTWI.h"     // I2C Master Lib

/*****
/*****
// Include the "RP6 Control M256 I2C Master library".
// This allows I2C control of the Robot base in a nearly identical API as
// on the Robot Base.

#include "RP6M256_I2CMasterLib.h"
/***** /
```

Figure 15: RP6M256 Libraries

2.4.2 WiFi Control Function

This function allows the remote control of the robot according the received commands from the WLAN module. The function that checks if an input is received is the following:

```
uint8_t getInputLine_WIFI(void)
{
    if(getBufferLength_WIFI())
        return parseLine_WIFI(readChar_WIFI());
    return 0;
}
```

Figure 16: Incoming line

The control that has the biggest priority is the message that certifies the connection between the User Interface and the Robot. After that the WLAN state of the RP6 M256 is enabled and the communication can start.

```
switch(wifiControl.state)
{
  case IDLE:
    if(getInputLine_WIFI())
    {
      if(strcmp(receiveBuffer_WIFI,"here")==0)
      {
        writeString_P_WIFI("Hello User");
        wifiControl.state = WIFI_ACTIVE;
        break;
      }
    }
    break;
  case WIFI_ACTIVE:
    if(getInputLine_WIFI())
    {
      if( strcmp(receiveBuffer_WIFI,"fwd")==0)
      {
        wifiControl.speed_left = speedl;
        wifiControl.speed_right = speedr;
        writeString_P_WIFI("FWD!");
        wifiControl.dir = FWD;
      }
      else if(strcmp(receiveBuffer_WIFI,"bwd")==0)
      {
        wifiControl.speed_left = speedl;
        wifiControl.speed_right = speedr;
        writeString_P_WIFI("BWD!");
        wifiControl.dir = BWD;
      }
      else if(strcmp(receiveBuffer_WIFI,"left")==0)
      {
        if(speedl/2 >= 30)
        {
          wifiControl.speed_left = speedl / 2;
          wifiControl.speed_right = speedr / 2;
        }
        else if(speedl > 10)
        {
          wifiControl.speed_left = 30;
          wifiControl.speed_right = 30;
        }
      }
    }
  }
}
```

Figure 17:WIFI Control State

As a default parameter we use the option of return the WLAN state back to IDLE to save energy, mainly because that means that the connection is lost.

```

default:
    wifiControl.state = IDLE;
break;

```

2.4.3 Main Function

In the main function the robot initializes the connection ports and after that it starts running the main loop. It runs the same commands continuously:

- Informs the user interface about the battery level
(writeIntegerLength_WIFI adcBat, DEC, 4)
- Waits for 600 milliseconds (mSleep(600))
- Search for obstacles (warnForObs())
- Waits for 600 milliseconds (mSleep(600))
- Adding delay (in 1ms) so the previous commands have the time to complete (task_ckeckINT())
- Waits for 600 milliseconds (mSleep(600))
- Runs the light detection function (light_detection())
- Waits for 600 milliseconds (mSleep(600))
- It needs to be called frequently the task_I2CTWI frequently otherwise the I2C Bus request functions don't work (task_I2CTWI())
- Waits for 600 milliseconds (mSleep(600))
- And run the behavior Controller function. (behaviourController())

```

while(true)
{
    writeIntegerLength_WIFI(adcBat,DEC,4);
    mSleep(600);
    warnForObs();
    mSleep(600);
    task_checkINT();
    msleep(600);
    light_detection();
    mSleep(600);
    task_I2CTWI();
    mSleep(600);
    behaviourController();
}
return 0;

```

Figure 18:Main Function's loop

2.5 Compile the Source Code

Following the same procedure as before¹⁶, we locate the file where the source code is stored, change if needed the Target name in the *makefile* and run the command in Cygwin or Linux Terminal *make all*.

¹⁶ Chapter 1.4.3


```
Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES
/WIFI_REMOTE
$ ls
make_all.bat      WIFI_REMOTE.c    WIFI_REMOTE.map
make_clean.bat   WIFI_REMOTE.hex  WIFI_REMOTE.o
makefile          WIFI_REMOTE.lst  WIFI_REMOTE.pnproj

Kate@Kate-PC /RP6Examples_20120725f/RP6Examples_20120725f/RP6_M256_WIFI_EXAMPLES
/WIFI_REMOTE
$ ..
```

Figure 19:WIFI Remote- Makefile

And the hexadecimal file is produced.

2.6 Connection and Upload

When the WLAN module connects to the network it transmits UDP packets every few seconds; opening the Discover WIFI Devices (Options-> Discover WIFI Devices) we can see that the robot is in that list. Click on the Add button and the information of the robot is added to the *Select Port* box in the second tab of the RobotLoader (v2), the WIFI Loader. Pressing the Connect button under the *Select Port* a communication is established between the RobotLoader and the Robot. Under the *Status* box there is a status bar, it is the RSSI (Received Signal Strength Indicator) status bar. It indicates the signal strength that was currently received in dBm.

“dBm (sometimes dBmW) is an abbreviation for the power ratio in decibels (dB) of the measured power referenced to one milliwatt (mW). It is used in radio, microwave and fiber optic networks as a convenient measure of absolute power because of its capability to express both very large and very small values in a short form. Compare dBW, which is referenced to one watt (1000 mW) sometimes dBmW) is an abbreviation for the power ratio in decibels (dB) of the measured power referenced to one milliwatt (mW). It is used in radio, microwave and fiber optic networks as a convenient measure of absolute power because of its capability to express both very large and very small values in a short form. Compare dBW, which is referenced to one watt (1000 mW).”¹⁷

¹⁷Source: <http://en.wikipedia.org/wiki/DBm>

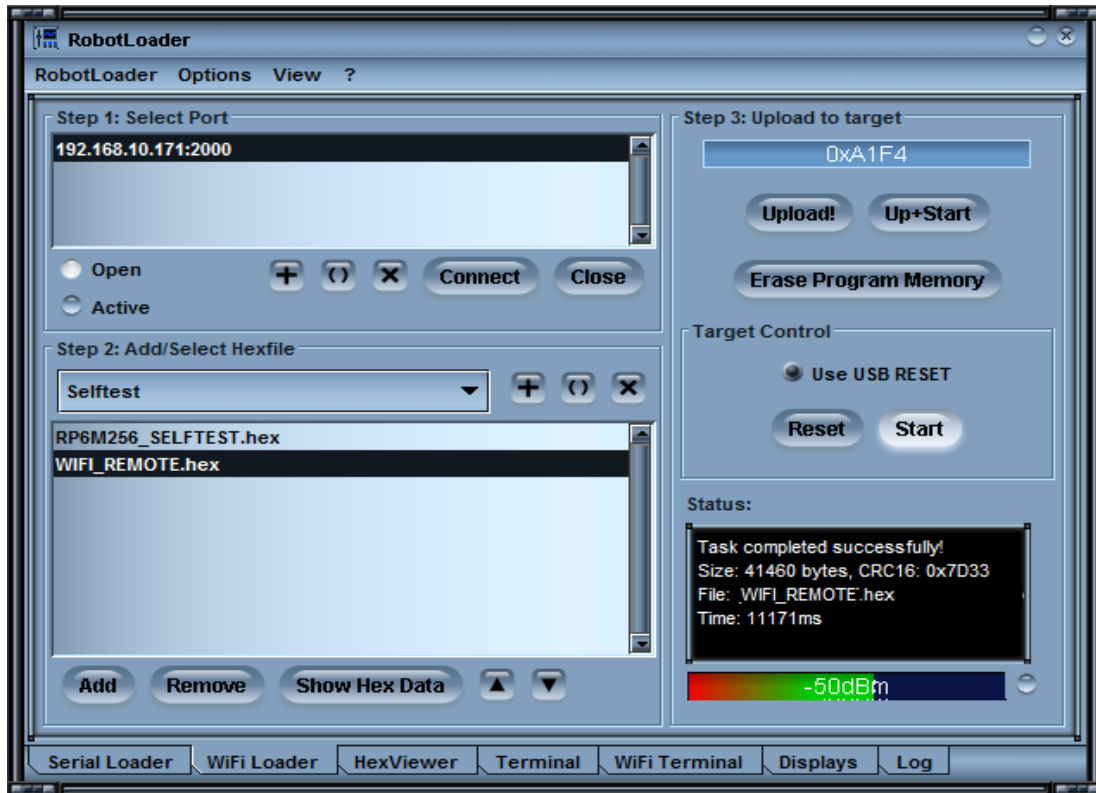


Figure 20: WIFI Loader

With the new version of the RobotLoader the upload of the program is possible via the WLAN connection. By pressing the upload button (if everything works fine) the message in the status box will change to “Task completed...”

Chapter 3: The Android App

3.1 About the Application

What will be created is an Application that gives to the user the needed interface to control the robot. The objects that are needed are:

- Buttons to send the *direction* commands to the robot
- A help dialog that gives the user information on how to use the application
- And last but not least a connection dialog, through which a TCP connection will be established between the Android device and the robot.

The development of the application will be a combination of XML and Java programming. Follows a picture of how the application looks like at the final stage of the development.



Figure 21: Application

3.2 Few words about Java Android

“Android is an operating system based on Linux with a Java programming interface. The Android Software Development Kit (Android SDK) provides all necessary tools to develop Android applications. This includes a compiler, debugger and a device emulator, as well as its own virtual machine to run Android programs.

Android allows background processing, provides a rich user interface library, supports 2-D and 3-D graphics using the OpenGL libraries, access to the file system and provides an embedded SQLite database.

Android applications consist of different components and can re-use components of other applications. This leads to the concept of a task in Android; an application can re-use other Android components to archive a task. For example you can trigger from your application another application which has it registered with the Android system to handle photos. In this other application you select a photo and return to your application to use the selected photo.”¹⁸

3.2.2 Development Tools

Android SDK

“The Android Software Development Kit (SDK) contains the necessary tools to create, compile and package Android application. Most of these tools are command line based. The Android SDK also provides an Android device emulator, so that Android applications can be tested without a real Android phone. You can create Android virtual devices (AVD) via the Android SDK, which run in this emulator. The Android SDK contains the Android debug bridge (adb) tool which allows connecting to an virtual or real Android device.”¹⁹

Android Development Tools

“Google provides the Android Development Tools (ADT) to develop Android applications with Eclipse. ADT is a set of components (plug-ins) which extend the Eclipse IDE with Android development capabilities. ADT contains all required functionalities to create, compile, debug and deploy Android applications from the Eclipse IDE. ADT also allows creating and starting AVDs.

¹⁸ Source: <http://goo.gl/9mCsC>

¹⁹ Source : <http://goo.gl/9mCsC>

The Android Development Tools (ADT) provides specialized editors for resources files, e.g. layout files. These editors allow switching between the XML representation of the file and a richer user interface via tabs on the bottom of the editor.”²⁰

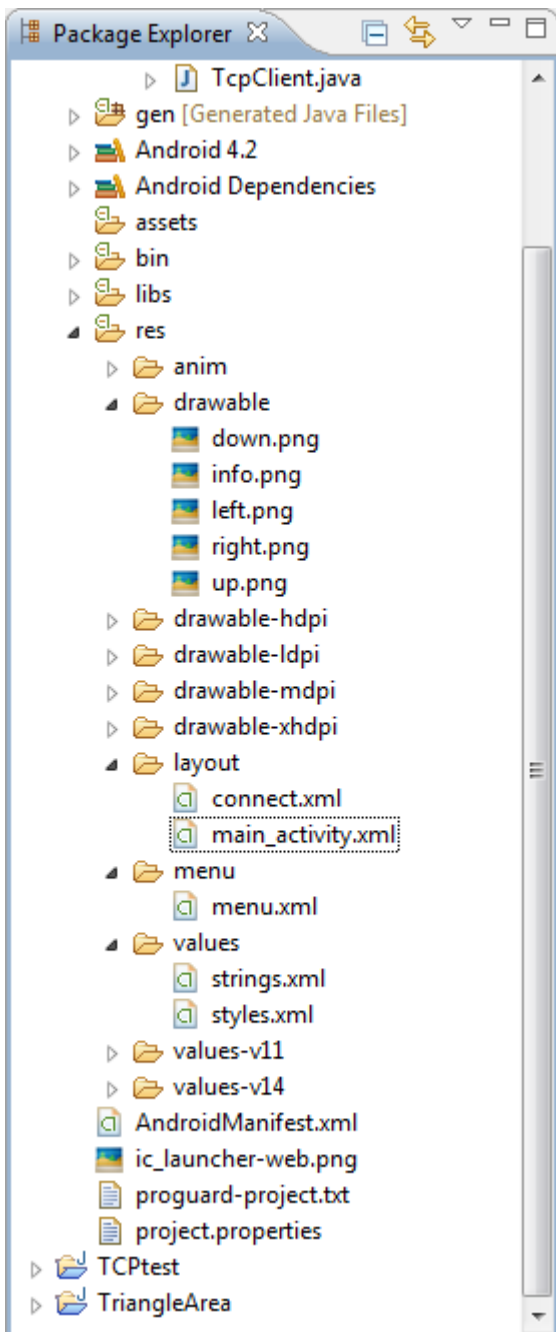


Figure 22: Eclipse Package Explorer

3.3 Design

XML Source Code

The application has two parts, the XML part that is the design (buttons, button’s position, button’s id etc.) and the Java part, which includes functions and classes triggered from the previously created XML components.

As it can be spotted in *Figure 22* in the *res* directory the design XML files are stored. For buttons *ImageButton* were used. For XML to retrieve the image that wanted to apply, the image had to be saved under the *drawable* folder, as a .png (Portable Network Graphics) format.

Under the *values* directory values of strings are stored; for example the string “*Connect*” that want to appear in the menu bar etc. or the values of the background color. They can be retrieved either by XML source code or Java source code.

In the menu directory and more specific in the menu.xml file, the menu items are defined. “*Instead of building a menu in your activity’s code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.*”²¹

For the *Connect* dialog two *EditText* objects were used to act as an input method for the IP and the communication port. Also there are two buttons; the *Connect* and the *Close*.²²

²⁰ Source : <http://goo.gl/9mCsC>

²¹ Source: <http://goo.gl/ZRiA>

²² The functions that are running with the onClick Command are described in Chapter 3.5

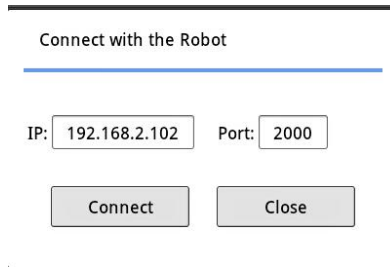


Figure 23: Connect Dialog

By adding the *Close* button, is given has the ability to close the window without connecting to the robot.

In the *main_activity.xml* file the visual structure of the user interface is defined. The color of the background, the layout of the interface (Linear/Grid/Relative), the buttons and their positions are determined in the layout file.

Every component has to have values; most important one is the ID. The id of the component is described by the command: `android:id="@+id/id_of_the_element"`.

Also the background, the style and size are determined the same way. As can be seen the values are changing due to the type of the component. A button cannot have the same values as a Text View.

In Figure 24 a small example of the *main_activity.xml* source code is given.

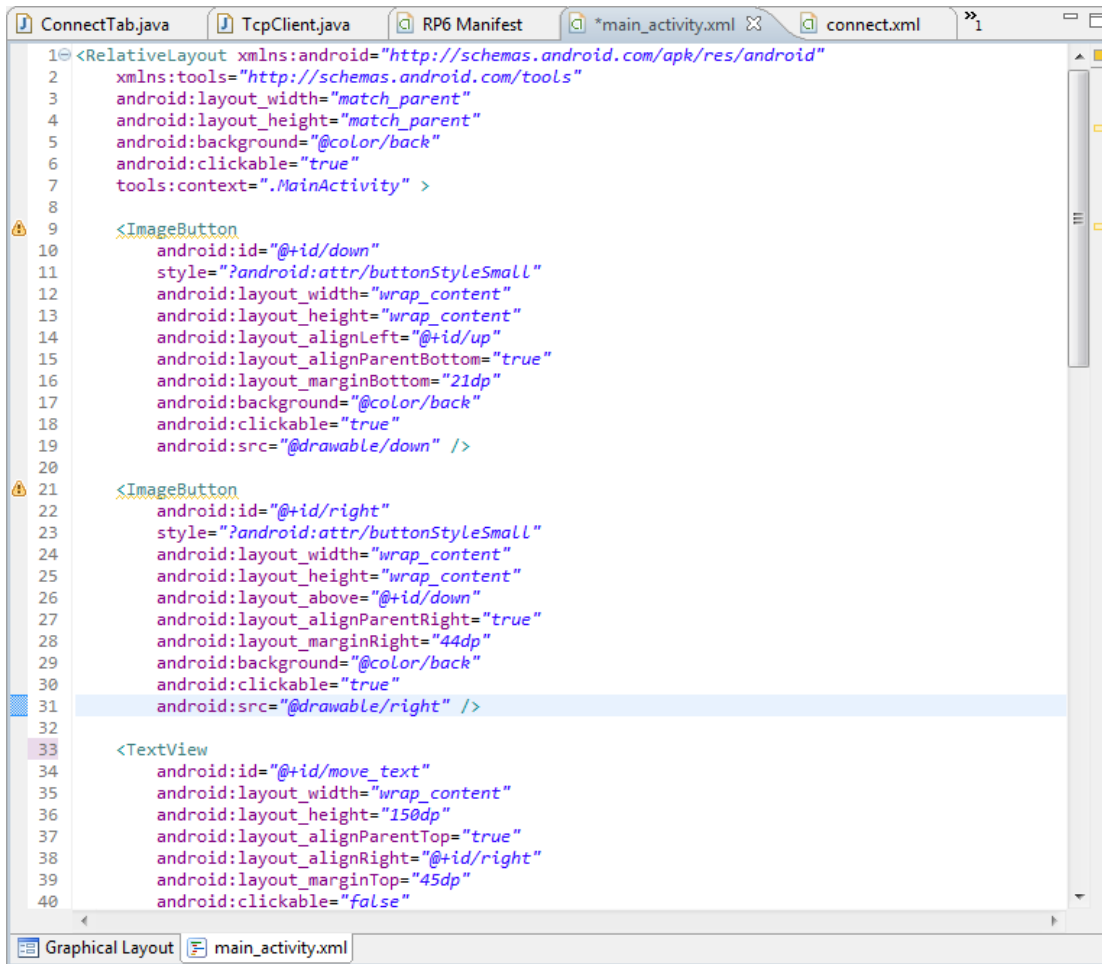


Figure 24: Screenshot from main_activity.xml file

3.4 TCP Socket Programming

To establish a connection between the robot and the Android device, a TCP Socket has to be programmed to run as a Thread in the Android application. In order for the application to grant access to the WIFI module of the device, a *uses-permission* has to be added in the manifest file.

```

36 <uses-permission android:name="android.permission.INTERNET" />
37 <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" >
38 </uses-permission>

```

Figure 25: Screenshot of the manifest.xml file

For the Socket to run as a Thread, the TcpClient class has to extend the Thread class:

```

16 public class TcpClient extends Thread {

```

The reason for implementing the Thread is so that the main activity will be able to run more than two different instances at the same time, each one for specific purposes.²³

To establish the connection the server's IP and communication Port have to be known

```
private void connectToServer() throws IOException{  
  
    if (socket == null || !socket.isConnected()){  
        ConnectTab.msgToMain("", Control_Panel.WAIT);  
        socket = new Socket();  
        socket.connect(new InetSocketAddress(conIp,conPort), 4000);  
    }  
}  
}
```

Figure 26: Connect to Server Function

With the command `socket.connect (new InetSocketAddress (conIp, conPort), 4000);` a request for connection is send to the server where IP = conIP and Port = conPort, with a TIMEOUT set at 4000 milliseconds, just to avoid wrapping the application into an infinite loop waiting for the accept message from the server.

Since the request of the client may fail, it is advised to “catch” the error which occurred.

```
public void run(){  
    try{  
        connectToServer();  
        mRun = socket.isConnected();  
        outBuffer = "";  
        out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(socket.getOutputStream()), true);  
        in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
        if(socket.isConnected()){  
            outBuffer = "here"; //to enable the wifi  
            out.println(outBuffer);  
            out.flush();  
            outBuffer = "";  
        }  
    }  
    catch (UnknownHostException e) {  
        ConnectTab.msgToMain("", Control_Panel.UNKNOWN_HOST);  
    }  
    catch(IOException e){  
        Log.e("IOException","Connect to Server", e);  
    }  
    catch(Exception e){  
        Log.e("Exception Error","Connect to Server", e);  
    }  
}
```

Figure 27: Threads run () function

If errors don't occur the program continues by creating a *reading* and a *writing* buffer to receive and transmit messages from and to the server. The application must be open to receive or transmit messages all the time while the connection is still active. The process can as well through some errors so it has to be surrounded by a *try... catch* block.

²³ Source: <http://goo.gl/sajLo>


```

if( in.ready() ){
    receivedMsg = in.readLine();
    Log.i("Msg from Server", receivedMsg);
    Control_Panel.reMsg = receivedMsg;
    ConnectTab.msgToMain("", Control_Panel.MESSAGE_RECEIVED);
}if( outBuffer != "" ){
    out.println(outBuffer);
    out.flush();
    outBuffer = "";
}

```

Figure 28: Reading/Writing Code

When a message is received, the class sends another message to the handler in the main class, informing him about the received message. The received message runs through control commands and based on the content of the message, the handler displays different kind of messages.

```

    }catch (InterruptedException e) {
        Log.e("InterruptedException", "Receive/Transmit ", e);
    }catch (IOException e) {
        Log.e("IOException", "Receive/Transmit", e);
    } //try

} //method

```

Figure 29: Try...Catch Block

In the *Menu* tab there is the Menu Item *Disconnect*, which when pressed the client disconnects from the server. So this function has to be developed in the *TcpClient* class:

```

133     mRun = false;
134     new Thread(new Runnable() {
135         @Override
136         public void run() {
137             try {
138                 socket.close();
139             } catch (IOException e) {
140             } //try
141         } //run
142     }).run();
143
144 } // method
145

```

Figure 30: Disconnect From Server

3.5 Connect Tab

The Connect Tab class is called from the Control Panel class when the Menu Item *Connect* is pressed. It creates a dialog box that sets as a layout the *connect.xml* layout. After creating all the components of the dialog box the command `connect.show()` is needed so that the user can see it. It also creates a `TcpClient()` instance which will run only if the *Connect* button is pressed.

```
client = new TcpClient();
connect = new Dialog(context);
connect.setContentView(R.layout.connect);
connect.setTitle("Connect with the Robot");
createButtons(context);
connect.show();
```

Figure 31: Connect Tab (1) - Source Code

When either one of the buttons is pressed, control commands are running checking the state of the socket. The possible states are a) equal to null or not and b) connected or not to a server. When the *Connect* button is pressed it proceeds to connect with the server/robot only after making sure that the IP and the Port number are correct (`accurateIp()` and `accuratePort()` methods).

```
if(accurateIp(ipAdd) && accuratePort(conPort)){
    client.setConIp(ipAdd);
    client.setConPort(conPort);
    Log.i("before",String.valueOf(client.mRun));
    new connectTask().execute("");
    //the function after that is not returning here it keeps running!!!
}else {
    msgToMain("", Control_Panel.MESSAGE_ERROR_ADDRESS);
}
});
}
```

Figure 32: Connect Tab (2) - IP& Port control

The function below returns true if the IP has the correct format for a typical IPv4 address.

```
private boolean accurateIp(String ip) {
    String regularExpression = "(\\d+)(\\.)(\\d+)(\\.)(\\d+)(\\.)(\\d+)";
    if (ip.matches(regularExpression)) {
        return true;
    }
    return false;
}
```

Figure 33: Connect Tab (3) - accurate IP function

And this is the output:

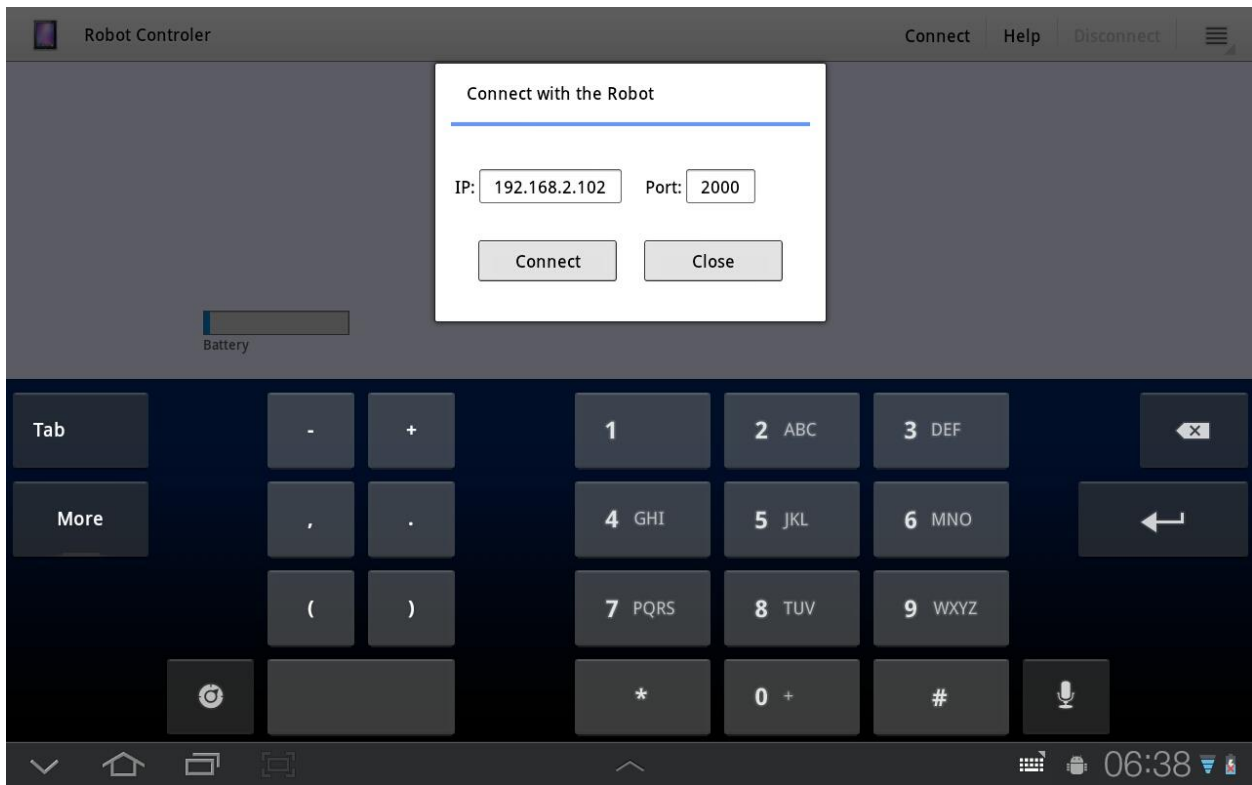


Figure 34: Connect Tab (4)

When the Connect button is pressed and the IP and Port number are correct it will call and execute the `connectTask ()` method. This method extends the `Asynchronous Task` class. *“The `AsyncTask` enables proper and easy use of the UI thread. This class allows performing background operations and publishing results on the UI thread without having to manipulate threads and/or handlers. `AsyncTask` is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework.”*²⁴

²⁴ Source: <http://goo.gl/Kz3Qh>

```

public class connectTask extends AsyncTask<String, String, TcpClient> {

    @Override
    protected TcpClient doInBackground(String... params) {

        // TODO Auto-generated method stub
        Log.i("before in the execute task",String.valueOf(client.mRun));
        client.run();
        return null;
    }
}

```

Figure 35: Connect Task - AsyncTask

3.6 Main Activity

In the Main Activity, or how it's called in this project Control Panel, the main user interface is built using Java programming language. By starting the application the onCreate method is called. *"It is where the normal static set up is done: create views; bind data to lists, etc. This method also provides a Bundle, containing the activity's previously frozen state, if there was one."*²⁵

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
    moveText = (TextView)findViewById(R.id.move_text);
    receiveText = (TextView)findViewById(R.id.receive_text);
    up = (ImageView) findViewById(R.id.up);
    down = (ImageView) findViewById(R.id.down);
    left = (ImageView) findViewById(R.id.left);
    right = (ImageView) findViewById(R.id.right);
    up.setOnClickListener(this);
    down.setOnClickListener(this);
    left.setOnClickListener(this);
    right.setOnClickListener(this);
    batteryLevel = (ProgressBar) findViewById(R.id.battery);
}

```

Figure 36: onCreate method - Control Panel

Before the handler is set, it has to be sure how many messages it will handle and for each and every one of those messages, to answer to the question: *what is it for?*

For that reason the messages are set as integers at the begging of the class

²⁵ Source : <http://goo.gl/jfdL>


```

public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.menu_connect:
            receiveText.setText(CLEAR);
            msgDisplay(CLEAR);
            tab = new ConnectTab(context);
            F = true;
            break;
        case R.id.menu_help:
            Builder builder = new AlertDialog.Builder(this);
            builder.setMessage(R.string.help_text);
            builder.setTitle("Information");
            builder.setCancelable(false);
            builder.setPositiveButton(R.string.ok, new dialogOkClick());
            AlertDialog dialog = builder.create();
            dialog.show();
            break;
        case R.id.menu_disconnect:
            if (tab.client.socket != null) {
                tab.client.disconnectSocket();
                tab.client.socket.equals(null);
                ConnectTab.connect.dismiss();
                tab = null;
                F = false;
            }
            Toast.makeText(context, "You disconnected from the robot/server", Toast.LENGTH_SHORT).show();
            msgDisplay("You disconnected from the robot/server");
            receiveText.setText("You disconnected from the robot/server");
            disconnectMenu.setEnabled(false);
            connectionMenu.setEnabled(true);
            break;
        case R.id.menu_close:
            this.finish();
    }
}

```

Figure 38: Selection a menu item

Choosing the *Help* menu item, creates a dialog, giving instructions for the application

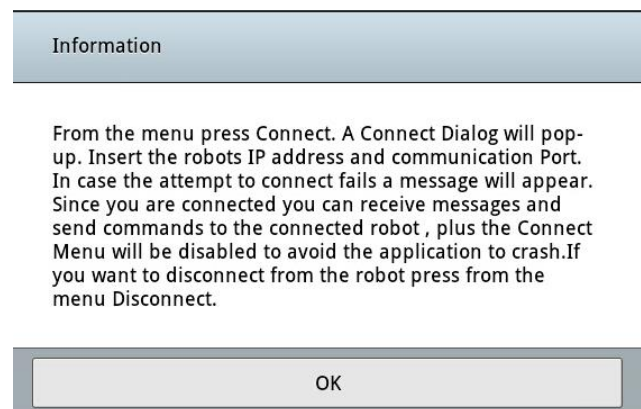


Figure 39: Help Dialog

By pressing the *Disconnect* menu item the method `disconnectSocket()` is called and the user terminates the communication between the client and the server. The application informs the user that he is now disconnected and pressing the moving buttons won't call an action.

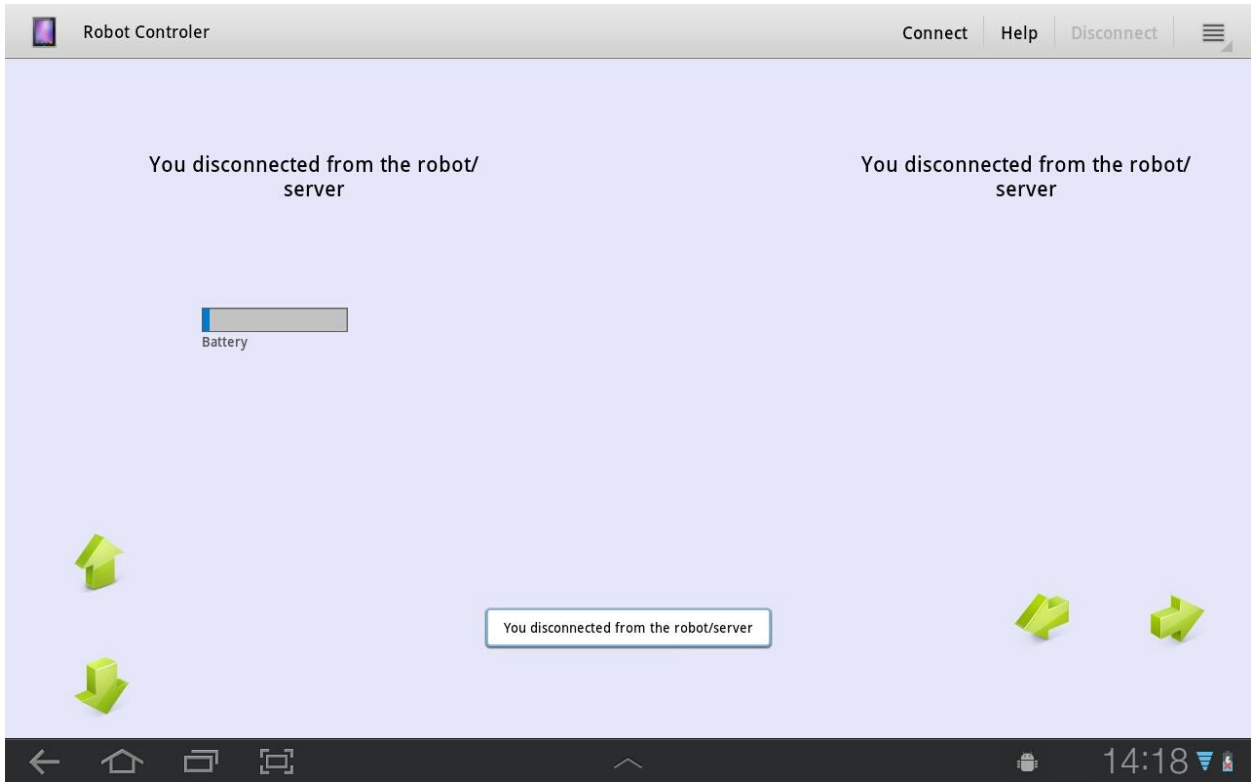


Figure 40: After the user disconnected

The notification dialog which informs the user that he has to connect for the buttons to work is displayed.

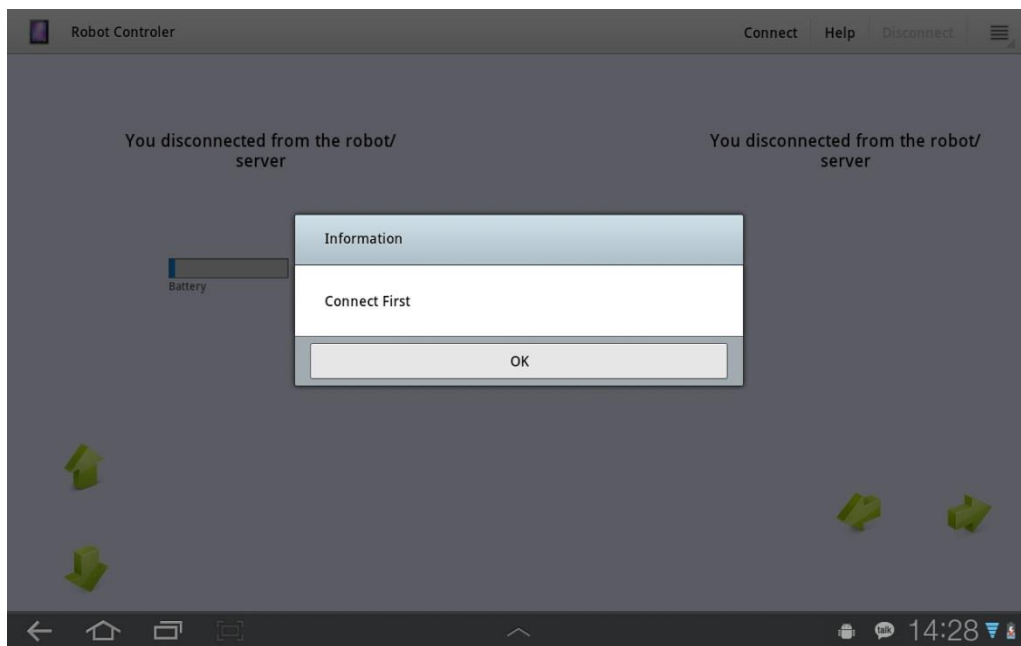


Figure 41: After Disconnect is pressed

3.6.1 Different Messages Scenarios

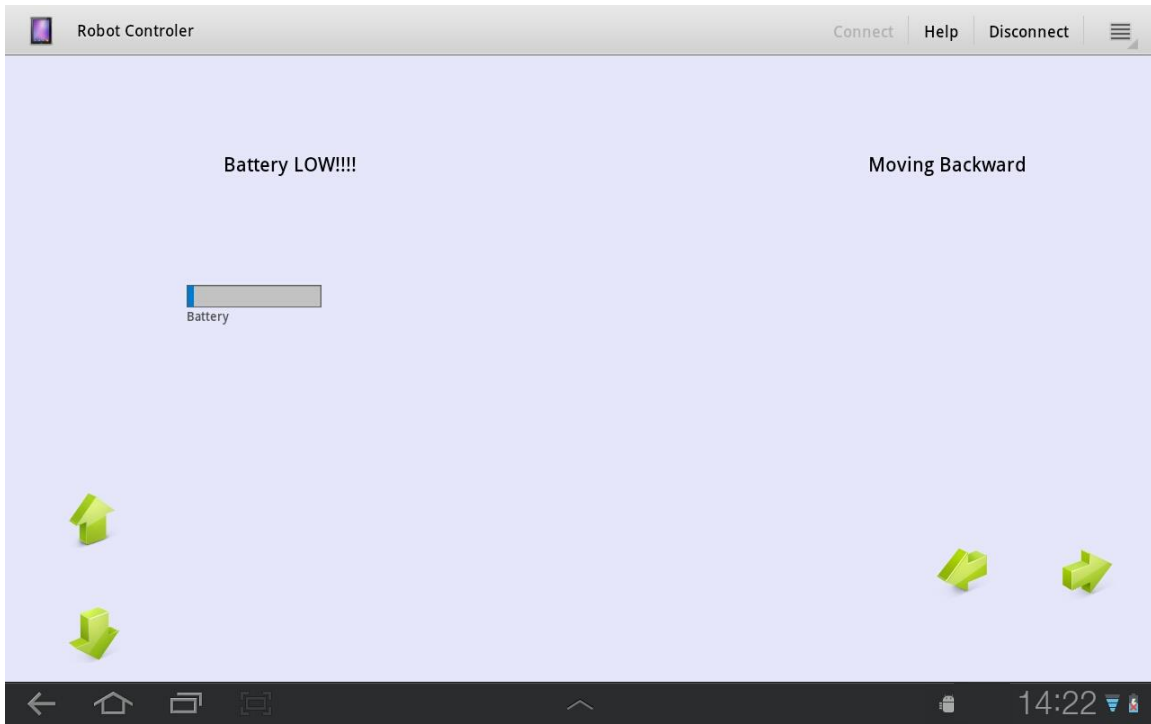


Figure 42: Low Battery message

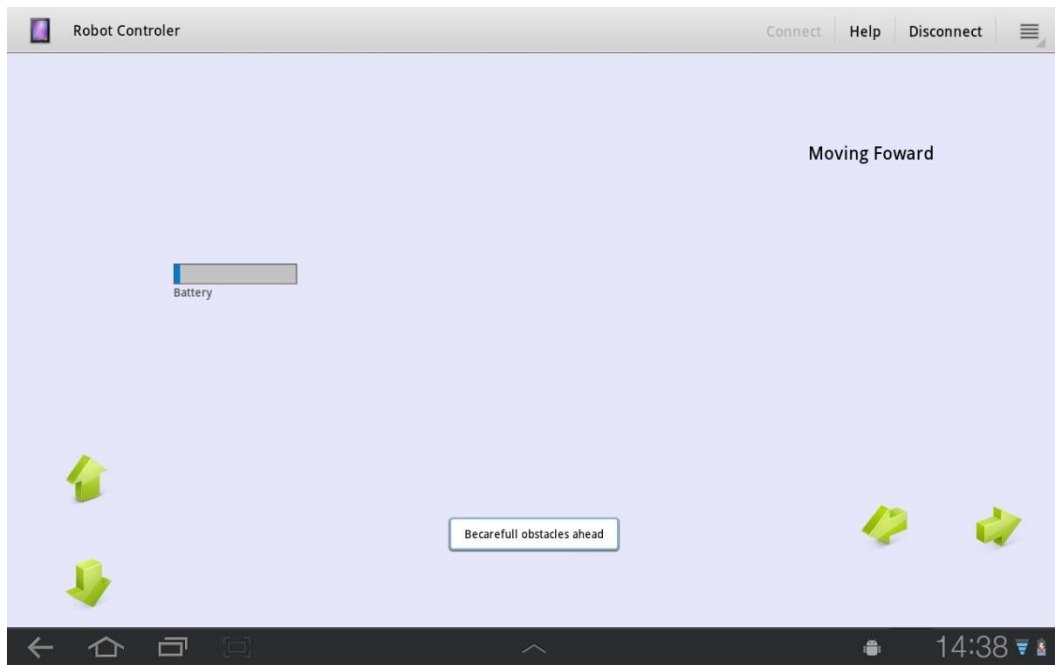


Figure 43: Obstacle detected

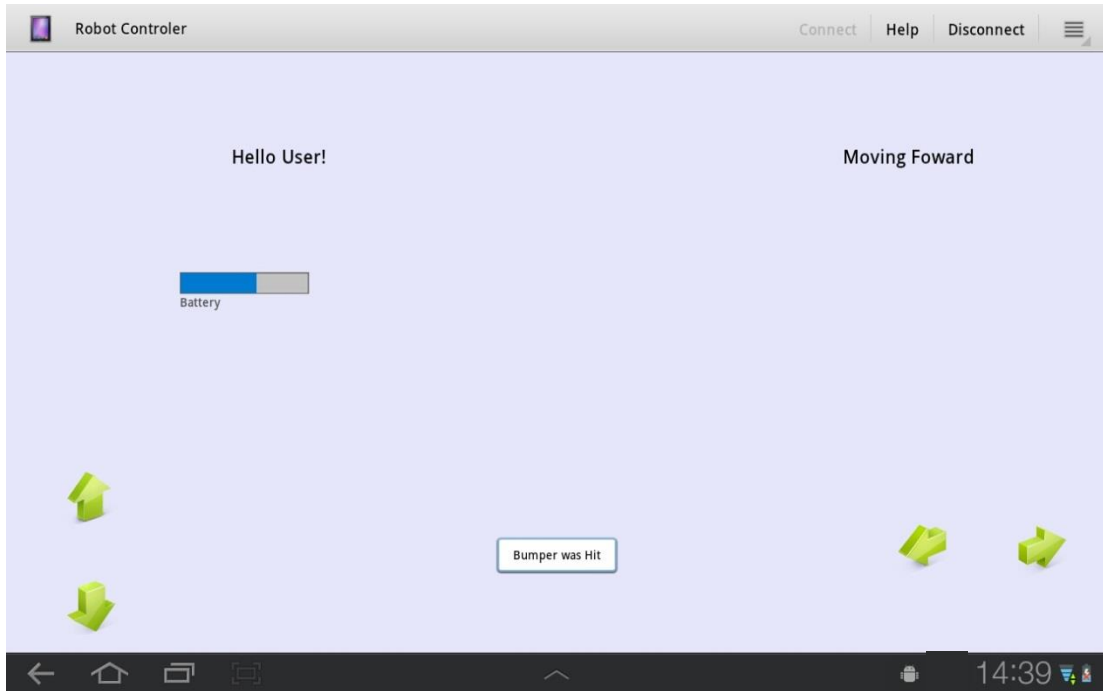


Figure 44: Bumper Hit

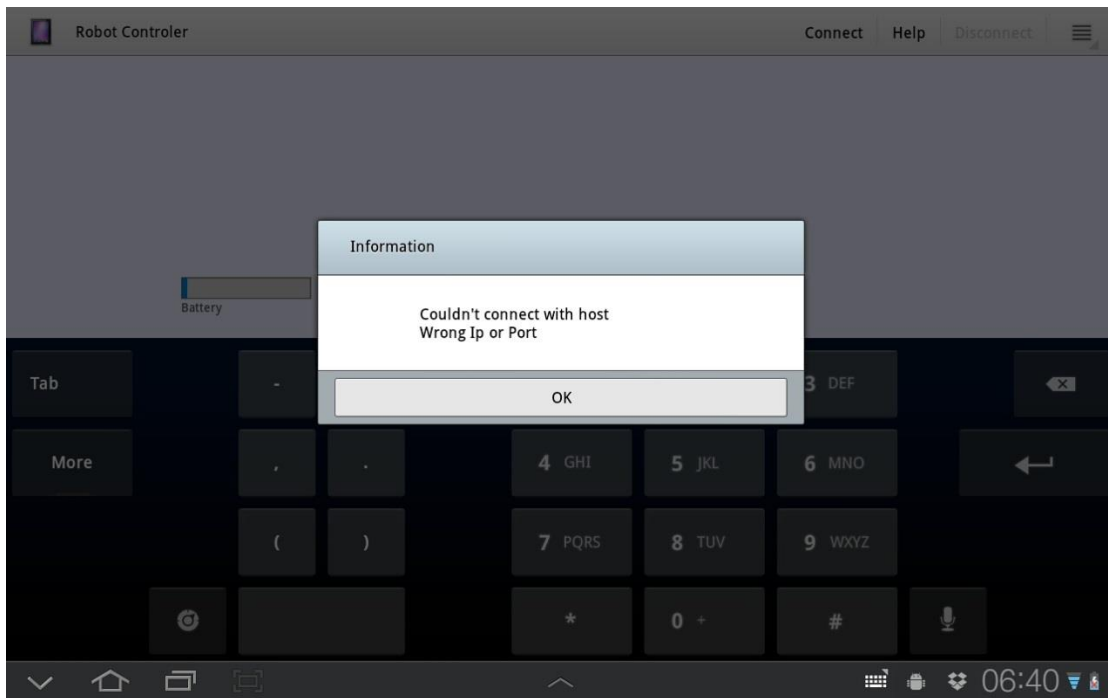


Figure 45: Wrong IP or Port number

Limitations

For the project to be completed the parts of the robot had to be ordered. Unfortunately the RP6v2 M256 WIFI expansion module's microcontroller Bootloader that was delivered wasn't updated to the firmware version 1.4 that was needed, but had the 1.2 version loaded instead. The Bootloader is *a small program that loads the operating system into the computer's memory when the system is booted and also starts the operating system.*²⁶ The 1.2 version supports WIFI communication but cannot communicate with the RobotLoader v.2 that we needed in order to configure the module.

As it was suggested by the support group of Arexx (the company that produces them), the bootloader could be updated via an ISP programmer and the use of *Atmel Studio*²⁷. They provided me with the Bootloader v1.4 and an instruction manual that was followed in detail.

Even though the update was successful the problems continued. . The communication between PC-Robot (and Application-Robot) was established, the WIFI uploads were successful, but the robot wasn't listening to commands that were inserted to the WIFI terminal (fifth tab of the RobotLoader). Since I had limited time to fix the problem I tried my best but I failed. It was suggested the module to return to the manufacturer and be updated properly to the latest version (Bootloader).

²⁶ Source: http://www.webopedia.com/TERM/B/boot_loader.html

²⁷ Source: http://www.atmel.com/microsite/atmel_studio6/

Conclusion

As it was stated in the Introduction this thesis has two parts, the programming of the robot and the developing of the Android application.

The robot's manufacturer provides good function libraries and a big selection of examples written in C programming language, which I took under consideration in order to create my own source code. Even in theory the program seems to work and it is logically correct, the WIFI module's problem didn't make it possible to test it in real action.

Developing the application in Java Android didn't come easy as well. Android developing is a big chapter in the world of developing and it certainly has a lot of choices. The biggest challenge for this project was to predict all the possible states. Without having a robot to connect to, the testing was happening with a server application, which cannot always give us the correct responds. In case the application would be tested with another robot few modification and addition are needed. Even if the robot will be programed to act like the RP6 that we used, it will probably have additional parts that it would best to put in use. In that case the application has to be modified to meet the user's demands.

Discussion

Ideas on how the project can evolve:

- Try to build a WLAN module using an Arduino board and an XBee wireless communication module. Arduinos are programmable in C language if the correct bootloader is uploaded to them. Using the Arduino board as the master device and the XBee as the WLAN module for communicating via a WIFI network, a similar module can be build.
- The Android Application was built to be versatile and adjust to control other robots like the humanoid robot NAO or AR Drone. In the application can be added a *choice* of robots. Instead of programming the other robot to act upon the same commands, the developer can program a second different controlling scenario and the user can choose at the beginning which one he wants.

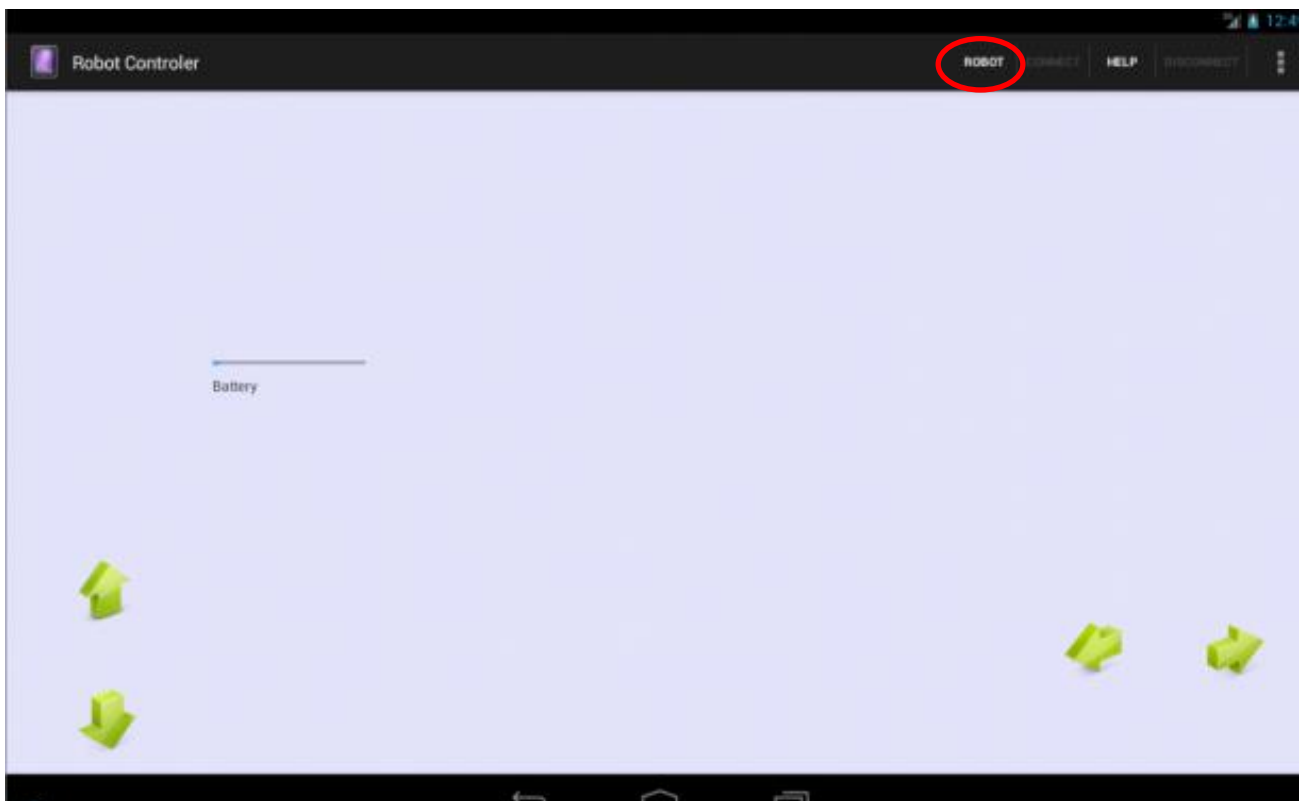


Figure 46: Suggestion

- Buying some more RP6 expansion modules (like robotic arm), using a soldering iron adding more sensors on the main board (or the experimental board) and expand the abilities of the robot and the Android Control Device. (Something that would need the M256 module repaired first).

References

- Lars Voggela Android Tutorials: <http://www.vogella.com/articles/Android/article.html>
- Arexx RP6 Official Website: <http://www.arexx.com/rp6/html/en/index.htm>
- Android for Developers: <http://developer.android.com/develop/index.html>
- Android API References: <http://developer.android.com/reference/packages.html>
- Cygwin official website: <http://www.cygwin.com/>
- Think Android: <http://goo.gl/pL73O>
- Socket Programming Tutorial: <http://goo.gl/SRGbE>
- YouTube Educational Channel
- Tutorials by Jakob Jenkov: <http://goo.gl/HNEfX>
- Edu mobile Tutorials: <http://www.edumobile.org/android/>
- O'Reilly Open Feedback Publishig System: <http://ofps.oreilly.com>
- Oracle Java tutorial : <http://docs.oracle.com/javase/tutorial/index.html> (Milan, 2009)
- Google web sites: <https://sites.google.com/site/microcontrollerprogrammingc/>
- Wikipedia: <http://en.wikipedia.org/wiki/DBm>
- TCP Protocol : http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- Socket: <http://docs.oracle.com/javase/tutorial/networking/sockets/>
- Atmel Studio: http://www.atmel.com/microsite/atmel_studio6/
- Webopedia: http://www.webopedia.com/TERM/B/boot_loader.html

Bibliography

Embedded C Programming the Microchip Pic [Book Section] / auth. Richard H. Barnett Larry D. O'Cull,Sarah A. Cox // Embedded C Programming the Microchip Pic / book auth. Richard H. Barnett Larry D. O'Cull,Sarah A. Cox. - NY : Clifton Park : Thomson/Delmar Learning,, 2004..

Lerning Java, 3rd Edition [Book Section] / auth. Patrick Niemeyer Jonathan Knudsen // Lerning Java, 3rd Edition / book auth. Patrick Niemeyer Jonathan Knudsen. - [s.l.] : O'Reilly, 2005.

PIC Microcontrollers Programming in C [Book Section] / auth. Milan Verle // PIC Microcontrollers Programming in C / book auth. Milan Verle. - [s.l.] : mikroElektronika, 2009.

RP6v2 Control M256 WIFI [Book Section] / auth. Engineering Arexx // Instruction Manual / book auth. Engineering Arexx. - [s.l.] : Arexx , 2012.