Mobile robot control using Bluetooth Low Energy

Till Riemer

Thesis for the Degree Program
B.Sc. European Computer Science

at Turku University of Applied Sciences
&
Hochschule für Angewandte Wissenschaften Hamburg

June 25, 2012

Instructor:
Jari-Pekka Paalassalo, Lic.Sc. (Tech.), Principal Lecturer

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

Abstract

University: Turku University of Applied Sciences (TUAS), Finland

Degree Program: Information Technology (Embedded Systems)

Author: Till Riemer

Title: Mobile robot control using Bluetooth Low Energy

Instructor: Jari-Pekka Paalassalo, Lic.Sc. (Tech.), Principal Lecturer

Date: June 25, 2012

Total number of pages: 35

Summary:

This thesis gives a working example on how to design and implement a remotely controllable embedded system consisting of two subsystems who are communicating with each other using Bluetooth Low Energy. The subsystems are a movable peripheral based on the Parallax Sumobot development kit, an Atmel AVR Butterfly, the Texas Instruments CC2540 development kit and a user input interface using the Apple iPhone 4S.

The first part is describing the fundamentals of the technologies and devices used in this project scope, with a focus on Bluetooth technology, in order to equip the reader with the background information necessary to understand the further proceedings in the thesis.

The main part of the thesis are the chapters describing the implementation of the system, beginning with working out an application concept and its requirements in chapter three, deriving the overall system architecture out of the requirements and doing and evaluating design decisions in chapter four, and assembling the hardware parts and implementing the system in chapter five.

The final chapter is evaluating the running system based on the requirements defined previously, giving an overview of the advancements made in the thesis project and providing ideas for future works and extensions to the application.

Keywords:
Bluetooth Low Energy, Embedded, AVR Butterfly, Sumobot, CC2540, iphone 4S

For my mother, who told me to never give up doing what you believe in.

Acknowledgments

This thesis would not have been possible to complete without the advice and guidance of my supervisor, Jari-Pekka Paalassalo, who instructed me through the whole process and always found time to answer my emails. I also want to send out my huge appreciation to Davide Berdin for the long, but nevertheless diverting hours working besides and pushing each other towards the goal, to Lukas Kern, being an amazing conversation partner all over the year and especially motivating me to push on towards the end, to Lola Brigitte Duprat, inspiring me to this thesis topic and enlighten my evenings with joint jam sessions, and to Veronika Karsai for repeatedly cheering me up and inspiring me through her thoughts - you have become very good friends during the year and will surely be across all national borders. A big thank you goes to all my friends and fellow students, in Germany and now thanks to the Erasmus program all over the world, for supporting and encouraging me on my way.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Turku, June 25, 2012 ____________________

Till Riemer

Contents

List of Figures

Chapter 1

Introduction

“Standards are always out of date. That’s what makes them standards.”

– Alan Bennett (British playwright and author)

The spread of personal computers and the establishment of the *World Wide Web* have led to a revolution of the way humanity is communicating and sharing information. Anyone is able to make his thoughts available to the whole world or participate in decision-making using social platforms such as *Facebook* and *Twitter*. The new technology increasingly influences the everyday life of a majority of people, reaching from ordering a pizza to online collaborations in music or movie creation. Yet, in the last decade there can also another innovation be observed, which is happening on a far more local level. PCs more and more get replaced by mobile devices such as Smartphones and tablets, and also in industry or public infrastructure a fast increase of intelligent mobile systems taking responsibility for small tasks can be observed. These devices are sometimes also communicating among each other, transmitting global position data, user authentication information or whole movies. Having the global climate change in mind, awareness for environmental-friendly solutions raised along with the establishment of mobile embedded systems. In addition, costs for energy supply have exploded during the last years, caused by the insecure situation in many mayor energy exporters in middle-east, the increasing amount of nature impacts and the global economy crisis. So-called *Green IT* products not only sell better because of their reputation; they also have a big positive impact concerning energy costs and their mobility in areas where no constant power supply can be guaranteed.

Bluetooth Low Energy is one of the new technologies feeding this emerging market. It provides the tools and abilities to design and implement environmental-friendly and low-cost applications which can operate and communicate in mobile environments, in the optimal case using only a single chip. It is resistant to interferences from other radio signals and can go through other objects, along with its support for encrypted data packages it is more than just comparable to previously existing standards such as Infrared Waves or IEEE-802.11 WiFi and a meaningful extension to the Bluetooth standard.

The purpose of this thesis is to develop an embedded system based on two subsystems, which are communicating with each other using Bluetooth Low Energy - one of them a remotely controlled moving device, the other one functioning as a remote control. The user can connect and disconnect to the device and enter simple movement commands using the remote control, which get executed instantly.

Based on this thematic introduction, following up a short outline of the thesis structure is given.

The following chapter is providing the necessary background information to understand the decisions taken later on. It is focusing on introducing the Bluetooth 4.0 standard and its extension Bluetooth Low Energy, explaining the basic functionality and protocols using in this project. In addition to that, the hardware used in this project is given a short introduction.
The main part of the thesis is focusing on three parts: Concept, design and implementation.
In chapter three, thoughts about possible real-world application scenarios were done, leading to a concept for a general application which might be extended to fulfill tasks for these scenarios in future. This concept is described non-formally and formally, describing requirements on concrete functions and nonfunctional abilities that the system must fulfill. The requirements then get derived into an according application model in the next chapter, comparing two possible solution approaches, determining system components and the way they are working together and making design decisions. Chapter five then concentrates on the evaluation part, first giving a short explanation of the assembly of the used hardware and installation of the developer tools, then explaining in detail some important aspects of the implemented code. The last chapter is validating the fulfillment of the concept and requirements and giving a conclusion of the work achieved in this thesis. It then provides an outlook on possible future improvements or extensions based on this project.

The thesis will mainly focus on the Bluetooth communication behavior and less on the Sumobot device, which has been concentrated on in the past, in a study project by myself as well as in other theses[1]. In order to understand the contents of the thesis, it is expected that the reader has some basic knowledge of Embedded Systems and Networks application development. This includes basic C programming, basic knowledge of embedded systems and networks development and the ability to install, configure and work with standard IDEs. For understanding the iPhone application logic, some knowledge of object-oriented programming and Objective-C in particular might be beneficial.

The final implementation code of all subsystems can be found in the appendix section, it is well readable and documented, although for understanding in detail and being able to make extensions further reading, especially in the datasheets of the devices, is recommended.

The whole source code of the final implementation, including all the project files, graphics and the latex files of this thesis document, can be downloaded [2] and distributed freely when mentioning the author.

[1]one example from the same department and supervisor as this thesis, going more into detail concerning the assembly of the Sumobot and the connection with the AVR Butterfly, can be found at https://publications.theseus.fi/handle/10024/15164

[2]http://code.google.com/p/sumobot-bsc-thesis-tillriemer/

Chapter 2

Background

The following chapter is designated to give a short introduction into Bluetooth Technology for readers unfamiliar with the topic. Following up, the connection establishment process as well as the Bluetooth Low Energy stack and its most important underlying layers of the Bluetooth stack get explained. Furthermore, a short explanation of the hardware devices used in this project is given.

2.1 Bluetooth

Bluetooth is a protocol for wireless device communication, which is intended and optimized for short distances [6, 1]. The standard is described in detail in the Bluetooth Specification[1] and maintained by the *Bluetooth Special Interest Group (SIG)*. The most recent major version is 4.0, which introduced a few changes with important impact on the Bluetooth protocol and its use. One major change was the newly featured ”Low Energy” protocol stack (henceforth called by the more regular used name *Bluetooth Low Energy* or simply *BLE*). Besides the BLE stack, the ”normal” Bluetooth also experienced several improvements in version 4.0 - furthermore, the standard is also downwards compatible to all predecessors. [14]
Classic Bluetooth connections operate on the unlicensed 2.4 GHz ISM band and allow relatively high data rates - the most common data rate modes at Bluetooth devices are Basic Rate (up to 721.2 kbit/s) and Enhanced Data Rate (up to 2.1 Mbit/s) [5]. Additional changes in Bluetooth 4.0 were made at the protocol layer through the introduction of the Generic Attribute Profile (GATT, see below) and Security Manager (SM), as well as through a new support of AES-encrypted data transmission and improved error correction [21].

2.1.1 Bluetooth Low Energy

Bluetooth Low Energy (BLE) is an extension to the Bluetooth 4.0 standard. It got introduced by the SIG in late 2009 [15] and is optimized ”specifically (for) for small battery-operated devices (...) that require almost no power” [7]. Devices supporting BLE communication are certified as *Bluetooth Smart Devices* by SIG. They operate on the same ISM band as classic Bluetooth devices, which is divided into 40 channels, 3 for advertising and 37 for data transmission.
A big advantage to classic Bluetooth is the much lesser power consumption of BLE devices, which is made possible through a much simplified device discovery and connection establishment process, as well as the short activity window of BLE devices, usually only sending small data packets every several seconds and going to sleep mode in between. At

[1]available at https://www.bluetooth.org/Technical/Specifications/adopted.htm

the beginning of a connection, the slave synchronizes its clock with the master and thus only needs to wake up periodically in order to send data. The time between two active circles is called *connection interval*.
Data packets are usually way smaller than classic Bluetooth packets - the maximum BLE packet size is 2971 bits. They are transmitted at 1 Mbit/s (over the air), which allows an active transmission window of only a couple of microseconds. [5]
BLE devices also have their limitations though - one is the reduced application data throughput (0.26 Mbit/s compared to 0.7-2.1 Mbit/s at classic Bluetooth), another important one the limitation of the signal range to a maximum of 50m (with a limitation of 10m for good signal quality in usual working environment like offices), half of the maximum possible signal range of normal Bluetooth 4.0. [22] The most suitable technology thus depends very much on the targeted product, its use and the constraints of the surroundings.

BLE devices act on a certain network concept called *Piconet* and must adopt certain roles which define their actions and abilities within the communication. These roles are given by the GAP and GATT profiles, which are introduced later, but as the roles are essentially for understanding the concept of Bluetooth respectively BLE, they are already described here.
Firstly, devices must choose between acting as a client or a server. Like the Client/Server model which is known from TCP/IP, a server is passively offering services to a client, but in a Piconet this does not necessarily relate to the way the network connection is established. Thus, the device must also adopt either the role of a master (in Texas Instruments terms *Central*) or a slave (*Peripheral*), whereas a master can actively connect to one or more slaves. Each master and its connected slaves form one Piconet together and work on the same radio frequency. Two or more Piconets can also form a *Scatternet*, where devices can have the role of a Master in one Piconet and simultaneously a Slave in another [21].
The master is responsible for establishing a network connection, the slave on the other hand is constantly waiting in advertising mode until receiving discovery requests. In order to establish a connection between two (or more) devices, the master must send a discover message, which gets sent over broadcast to all other devices within reach, repeatedly running through all Bluetooth frequencies. Slaves with inquiry mode enabled listen for incoming discovery messages on their assigned frequency within a defined interval and time window, and on receiving send a reply to the master, containing their device address and class, as well as (Bluetooth 2.1 upwards) additional commonly requested data. The master may then, automatically or on user input, send a connection request destined to the specific slave, who, if it is configured to accept connection requests, answers with a connection reply, if else refuses the connection attempt. [6, 8, 36ff.]
After the master received a positive connection reply, the connection is established successfully on both sides.

2.1.2 BLE layers

The Bluetooth standard itself is not a single protocol, but provides a stack with lots of different layers acting similar to the TCP/IP network stack. On the higher-abstraction levels, these layers are often also called profiles. There is a wide range of existing layers and profiles specifically design for certain uses for example in health or sports applications, hence the following listing just shows up the ones that are important for the scope of this thesis.

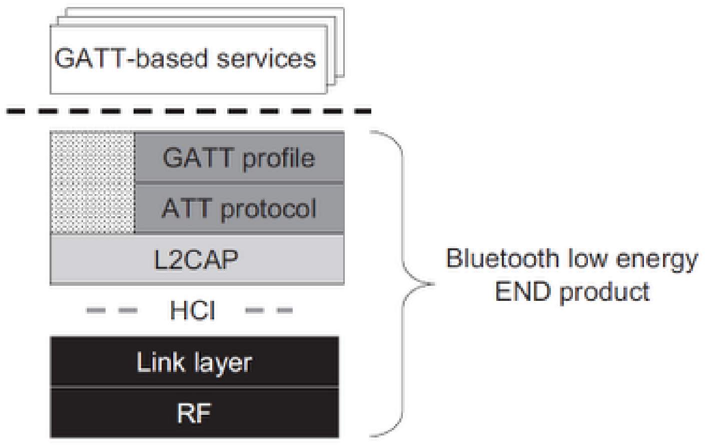


Figure 2.1: *The Bluetooth Low Energy protocol stack (taken from [12])*

The Bluetooth Low Energy network stack can be divided into two subparts, the controller stack, which is containing the interface dealing with the radio transmission, and the host stack, which is working on more high level data. All these layers are directly taken from or based on layers of the Standard Bluetooth protocol stack.

controller stack

The *Low Energy Link Layer (LE LL)* is the BLE equivalent to the classic Bluetooth *Link Manager Protocol (LMP)*. It "manages advertisement, scanning, connection and security from a low-level, close to the hardware point of view" [23] and is operating on the baseband controller hardware. In a typical Bluetooth application, it is not called directly, but either using GAP commands or over HCI in dual mode.

The *Host-Controller Interface (HCI)* is "a standardized Bluetooth interface for sending commands, receiving events, and for sending and receiving data" [17, 10]. HCI specifically allows the use of a Bluetooth device in dual mode with communication over a UART or USB interface. It is consisting of several types of packets: Commands, which are sent from the host to the controller and affect the controller's behavior, events, which are sent vice versa and notify the host about the current state of the controller (including settings, execution statuses of commands, incoming connection requests and so on), as well as synchronous (not supported on TI CC2540 device) and asynchronous data packets.

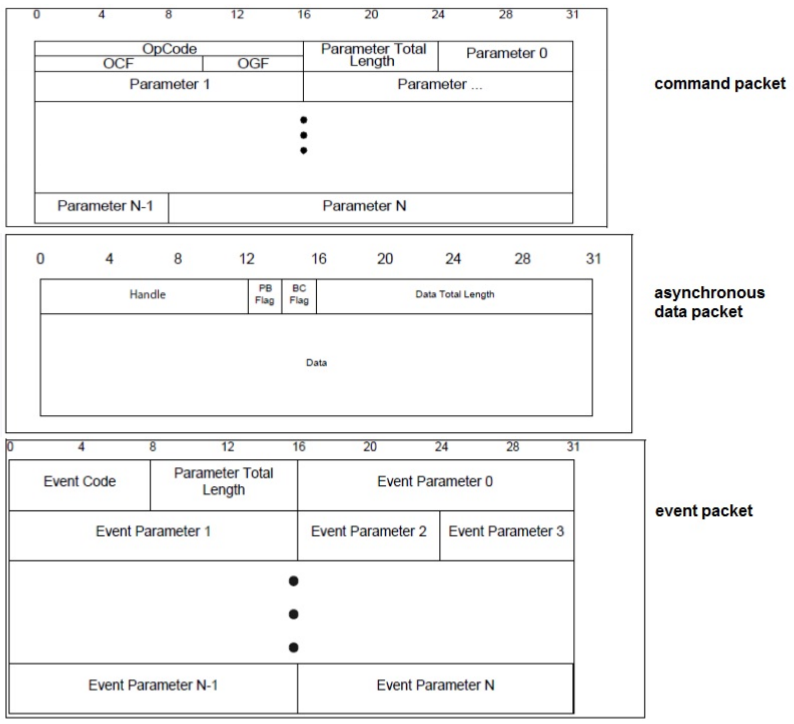


Figure 2.2: *HCI packet formats supported by TI CC2540 (taken from [17, 11ff.])*

[17, 12ff.] provides a list with all HCI commands and events supported by the BLE stack, as well as TI-defined commands and events.

host stack

The *Logical Link Control and Adaptation Protocol (L2CAP)* "provide(s) a frame-oriented transport for asynchronous and isochronous user data" [16]. Its functions include Multiplexing and segmentation of data packets, quality of service parameter management for higher layer protocols such as GATT. In dual mode, it passes packets on to HCI - thus, in this thesis project it only needs to be kept in mind as the underlying layer of data handling.

The *Generic Access Profile (GAP)* provides the basis for all other profiles. It describes how devices connect to each other by specifying devices as either a Master or a Slave device (compare above). Another two roles, which cannot be used for devices actively participating on the Piconet, but for testing and observing, can be given: A broadcaster, which is sending out advertising data but not connectable, and an observer, which is passively looking for devices in range without the possibility to send a connection request.

The *Generic ATTribute Profile (GATT)* is based on and a simplified adaption of the Bluetooth-given *ATTribute Profile (ATT)*, dividing the devices into clients and servers, as described above, whereas a server is passively offering higher-level data profiles and application entities called *services* to clients. These services can include several subservices or several *characteristics*, which again may have several *descriptors* - hence, application entities can be described at several levels of abstraction, making them useful for a broad range of applications without the necessity of sending custom-defined datasets using asynchronize data packets. Services, characteristics and descriptors can all be separated by their custom UUID.

2.2 Parallax Sumobot

The *Parallax SumoBot Robot Kit* is a development kit mainly intended for the construction of robots participating in *Northwest Robot Mini-Sumo Tournament* contests. It is developed and distributed by Parallax Inc. and contains all the necessary parts for developing a full-functioning robot vehicle, but can easily be extended by additional parts.

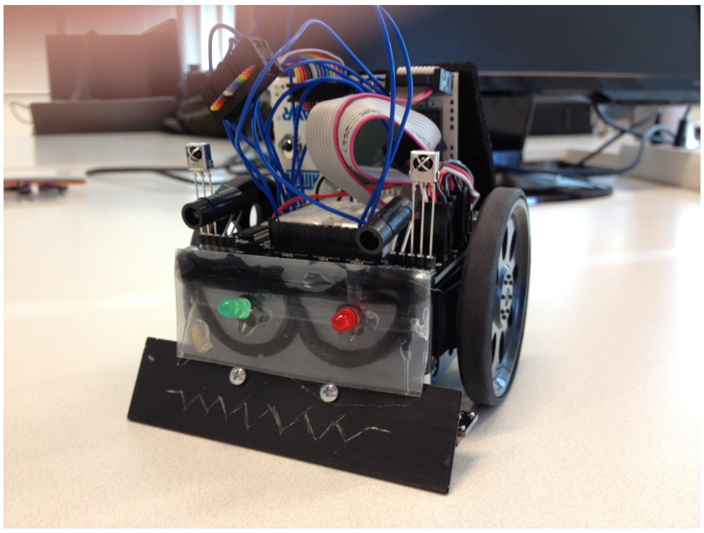


Figure 2.3: *assembled Sumobot with minor modifications on the design, as used in the thesis project*

The contents of the development kit are as follows [10]:

- (1) SumoBot Board with surface-mounted BS2
- (2) QTI Sensor
- (1) Parallax Screwdriver
- (1) Chassis, SumoBot
- (1) Front Scoop, SumoBot”
- (2) Wheel, Plastic, 2.58 Dia, .3 W
- (4) Rubber Band Tire
- (1) Battery holder, 4cell, AA, leads
- (1) SumoBot Manual
- (2) Res, CF, 5%, 1/4W, 470 Ohm
- (1) LED-GREEN-T 3/4
- (2) LED-Infrared - T1 3/4
- (1) LED-Red - T1 3/4
- (2) IR Receiver
- (2) LED Standoff
- (2) LED Light Shield
- (1) Serial Cable
- (1) 3 inch Jumper Wires (1 Bag of 10)
- (2) Servo Extension Cable (10 inches)
- (1) Piezo Sound Generators

- (2) Continuous Rotation Servo (Futaba)
- Assortment of screws, washers, and standoffs

The Parallax Sumobot development board already includes a *BASIC2 Stamp microcontroller module* and is programmable using the language PBASIC, which is developed by Parallax itself, and which is principally providing all the functionality needed to control the robot movements and make use of other peripherals [9]. As the PBASIC programming language is not commonly used in Embedded programming, regarding possible further extension I decided to use a separate microcontroller which can be programmed using the quasi standard language C, the *AVR Butterfly*. I already used this microcontroller in the study project when taking part at the Sumobot contest, and thus was already confident with it's characteristics.

2.2.1 AVR Butterfly

Figure 2.4: *promotion picture of the AVR Butterfly, by Atmel Corp.*

The *AVR Butterfly* is a re-programmable embedded development and evaluation board by Atmel Corporation, based on the ATmega169V AVR microcontroller. It features a 32kHz crystal, a Piezo sound element, a 120-segment LCD display, RS-232 UART and more [2]. Its main advantages are the small stock price, the communication or peripheral I/O possibilities using UART, ADC or digital I/O ports and the big AVR developer community, offering guidelines and support. The board can be programmed and debugged using the included hardware debugger and the programs AVR Tools and AVR Studio. Several project templates and example projects, which allow a quick start into programming the Butterfly, are available on the websites of Atmel.

2.2.2 Texas Instruments CC2540 Mini Development Kit

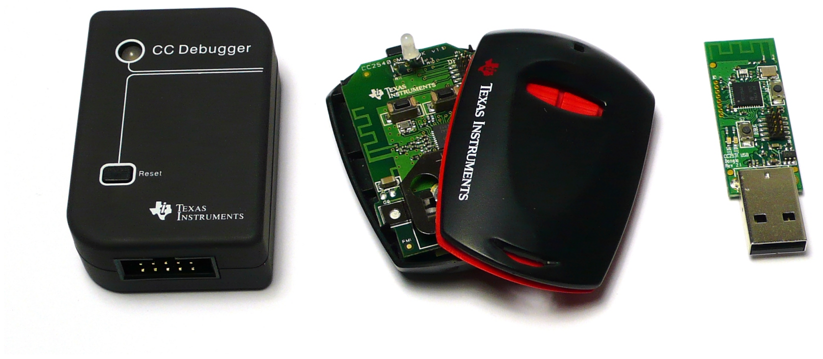


Figure 2.5: *promotion picture of the TI CC2540 Mini Development Kit, by Texas Instruments Inc.*

The *Texas Instruments (TI) CC2540 Mini Development Kit* (furthermore referenced as CC2540) "is a cost-effective, low-power, true system-on-chip (SoC) for Bluetooth low energy applications" [18]. Included in the kit are two Chips, the keyfob and the USB dongle. Both are using the 8051 MCU and utilizing the whole Bluetooth Low Energy stack, supporting both master and slave implementations. They can easily be extended for projects using other I/O operations by the two included USART ports on the keyfob respectively USB and one USART port on the dongle. TI provides plenty of useful documents and tutorials for the CC2540 such as a software developers manual and Bluetooth command descriptions, as well as a web community where it is possible to discuss with other BLE developers. Furthermore, the supplied code and the APIs are well written and documented and the TI-supplied project templates for applications with various purposes hugely reduce the work effort needed to realize most applications using BLE. Deployment and debugging is usually done using the supplied hardware debugger (*CC Debugger*) and the Deployment Software SmartRF Flash Programmer. The compiler vendor IAR offers with its commercial solution *IAR Embedded Workbench 8051* a powerful IDE for developing and debugging CC2540 applications.

2.3 Apple iPhone 4S

Figure 2.6: *promotion picture of the Apple iPhone 4S, by Apple Inc.*

The *iPhone 4S* (furthermore also simply referred to as *iPhone*) is a popular smartphone, developed and published in 2011 by Apple Inc. It is currently one of the best-selling mobile phones in the world, based on the ARM Cortex-A9 MCU and featuring among other things a multi-touch input system, two inbuilt cameras, a 3-axis acceleration sensor and voice recognition. It was also the first released mobile phone utilizing Bluetooth 4.0 and thus BLE [3]
The iPhone 4S is using the proprietary iOS5, an Operating System also developed by Apple. User applications are running in a sandbox and are usually called *apps*, a word that became a general term for mobile phone applications since the release of the first iPhone generation. The iPhone Development Tools provide an abstraction layer for easy access to features of peripherals - e.g. the internal Bluetooth chip can be controlled using the Core.Bluetooth package. iPhone apps have to be deployed using Apple's Mac OS-X operating system with installed XCode (the standard Mac OS-X IDE) and iPhone Developer Tools.

Chapter 3

Application scope and concept

The chapter ahead is dealing with the creation of a feasible application concept, based on the already discussed background concepts and possible real-world usage scenarios, which will be talked about next. These scenarios are leading first into a general, non-formal description of the application concept, and following up into resulting functional and non-functional requirements on the system, which allow a final evaluation after the realization.

3.1 Real-world usage scenarios

Short-range wireless controlled robots are useful and desired in various environments, in consumer market as well as when dealing with special situations. The following list shows a selection of possible real-world scenarios, whereas the limitations of wireless communication protocols, especially concerning the maximum signal range and data transmission rate, have to be kept in mind.

- control of agricultural or cleaning vehicles, especially in environments where the vertical space is limited below the height of a human-controlled vehicle
- consumer electronics, e.g. remote-controlled cars or helicopters with the purpose of entertaining
- controlling robots through tiny tunnels or other environments where it won't be possible or too dangerous for humans to pass, such as dealing with exploration in archeology, rescue operations or defusing bombs

In all these scenarios and in mobile systems in general, saving energy became one of the most important design goals, as possibilities for recharging batteries are limited in open environment and the awareness for the need of environment-friendly, so called "Green IT" products increased rapidly during the last years. Security and safety issues, especially in a use-case scenario such as rescue or S.W.A.T. operations, oblige application systems to encrypt their communication traffic, react reliable to any possible user input and ensure that no interference with other devices or radio signals is occurring.

3.2 Concept and requirements

The various real-world application scenarios require a system that is as general and scalable as possible. Products for the consumer market also require that the devices are available on stock - in addition, the devices and the user interface to be implemented should be easily controllable by non-IT experts, as the range of possible customers differs highly.

Besides of that, the application concept is orientated on three main factors, which are, ordered by their importance in this scope, *reliability*, *reaction speed* and *power saving*. It might seem odd that after describing the increasing importance of saving power this factor is now graded less important for the project, but this aspect largely depends on the implemented user application and can not be influenced much besides of the use of Bluetooth Low Energy instead of standard Bluetooth. In all usage scenarios, the aspect of reliability should be of utmost importance, as not occurring or false reactions of moving vehicles on user input can lead to serious situations and accidents, up to hazarding humans. This also includes an adequate reaction speed on user input, which is also a large influence on the user experience of a product - significant delays arguably worse the overall impression of the application and may raise the customer's temper if occurring on a regular basis.

The resulting system thus consists of a mobile input device and a peripheral device which is an embedded system able to move at least in 2D space (of course, extension to 3D for devices such as drones are possible). The iPhone 4S, one of the first mobile devices equipped with BLE and in large amounts available on stock as well as intuitively usable, has been chosen as input device. The peripheral device chosen is a custom-built robot based on the Parallax Sumobot development kit, which has been proving its suitability for many hobbyist projects in the past, and the Texas Instruments CC2540 microcontroller, one of the first and by now best documented BLE chips. The project should not be seen as a ready-to-distribute application, but as an example demonstration of low energy short-distance communication between mobile devices, as the concrete abilities of the implemented application largely depend on the stake background.

Having discussed the project concept, the following functional and non-functional requirements can be applied to a practical implementation:

3.2.1 Functional requirements

F0 The system consists of two subsystems with independent applications: the Sumobot with Butterfly and CC2540, and the iPhone.

F1 For this example demonstration, the robot must be able to move to every target in 2-dimensional space using two servo motors.

F2 Possible movements include several predefined speed stages as well as for- and backward movement and the possibility to stop.

F3 The movements are distinct and reliable, meaning that only one defined movement can be executed at a time and a pressed button will always result into the desired movement, given an ongoing Bluetooth connection.

F4 The peripheral does not take action by itself, all movement control comes from the user input of the iPhone application, assuming that an active Bluetooth connection is existing.

F5 The iPhone application is organized in two tabs, the first one responsible for connection establishment and the second for the control of the peripheral. The user can move from the first to the second tab with a swipe gesture from the right to the left and come back using a Back-button.

F6 The user can connect to the first found peripheral and disconnect at any time using a button on the first tab of the iPhone application. A disconnect always sends a command to stop the peripheral first.

F7 The second tab provides buttons for the desired directions, which trigger the sending of the command to the peripheral when being pressed.

F8 One iPhone can control at maximum one peripheral at a time.

3.2.2 Non-Functional requirements

N0 The peripheral application should be designed in a way that is energy-efficient in order to allow possible power-saving application extensions or projects working with the BLE chip application.

N1 The peripheral application code should be flexible and easily portable on other embedded systems, thus the application logic should be strictly separated from the hardware access.

N2 The iPhone application interface should be intuitive to operate and intended for general purpose use.

N3 Between the user input and the actual robot movement should be no delay visible for the human eye.

The term *peripheral application* refers to both the Butterfly and the CC2540 application. N0 and N1 don't apply to the iPhone application due to its high-level abstraction and the small possible influence on the characteristics of the underlying OS. In addition, the technical abilities of the Sumobot can rely on the given defaults by Parallax, because the focus of this project does not lie on the creation of a robot with an actually useful purpose, but on the practical demonstration of the BLE communication of two devices.

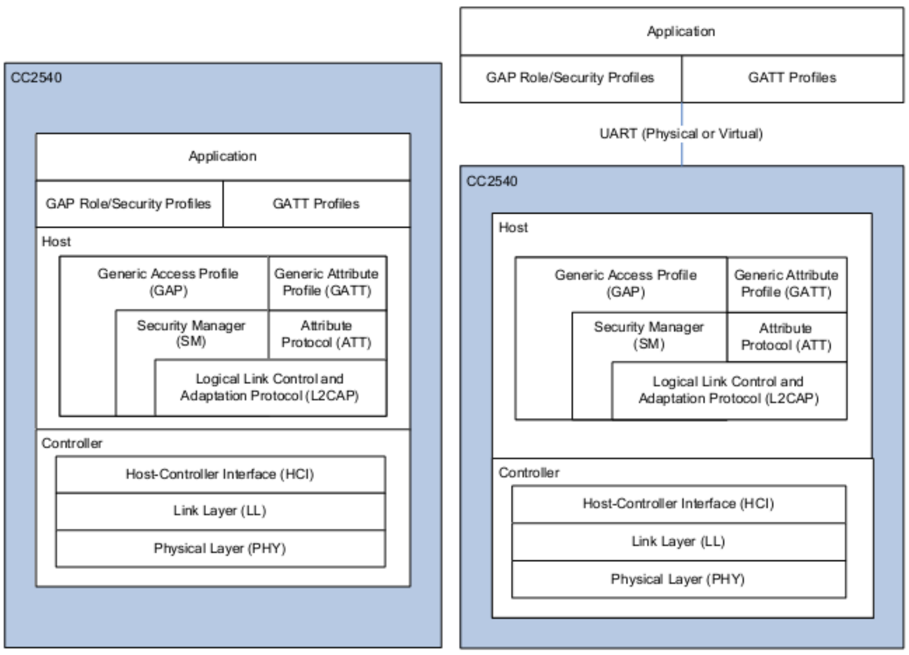
Chapter 4

Application design

Based on the determined requirements, the following chapter is specifying the overall software architecture and behavior of the single components as well as their intercommunication. The two subsystems of the requirements specification can be split up further in terms of implementation into three system components. These are specified as the input device, in this case an Apple iPhone 4S, the robot Bluetooth controller (a Texas Instruments CC2540 development board) and the robot Bluetooth host (an AVR Butterfly development board). These subsystems and the communication flow between them are now furthermore explained.

4.1 Bluetooth communication

Bluetooth, like many other communication standards, provides different operational strategies. Application developers have to decide between implementing the *single mode* or *dual mode* strategy (often also called Single-CPU and Multi-CPU). In single mode, the Bluetooth stack shares a single CPU or microchip with the application logic. When utilizing dual mode instead, the Bluetooth network operations and the application logic run on separate microchips - the application can control the BLE communication by accessing the HCI interface of the Bluetooth stack, typically using a UART or simulated UART connection between the two devices.



CC2540
Application
GAP Role/Security Profiles
GATT Profiles
Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager (SM)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Controller
Host-Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)
UART (Physical or Virtual)

Figure 4.1: *BLE protocol layers of the TI CC2540, left: single mode [20, 7], right: dual mode [20, 8]*

The single mode strategy has a clear advantage concerning costs, especially in mass production (only one chip necessary for Bluetooth and application logic), and can also have a positive effect on the system performance, because the application does not have to do I/O operations on an external peripheral for Bluetooth communication. If acting as a slave, it might also require much less power than the dual mode way, because the system can go to sleep mode as long as no event is incoming and the device has nothing else to do. Still, it can also have a negative impact, if the application requires more CPU power than what is supplied by the BLE microchip - therefore, the better solution in terms of performance and power consumption should be checked on a case by case basis.
There may also be other limitations concerning the amount of I/O ports, memory or the support of an operating system - for example, the CC2540 provides only a rudimentary OSAL without the support of real-time operations [20, 9]. In case that the intended device is non-trivial or that it's functionality might be extended in future, it is therefore better to use the dual mode strategy up from the beginning and separate the BLE functionality in terms of hardware.

The CC2540 is supporting both of those operational strategies. My first approach was to use a dual mode strategy for the BLE communication between the subsystems and thus use the CC2540 as Bluetooth controller and the Butterfly as Bluetooth host. The main reason for this decision was the limited possibilities in future extension of the project due to the hardware limits of the used CC2540 chip, especially concerning the amount of available I/O ports. The same applies for the limited power of the CPU and the lack of support for a "real" RTOS.
The other reason was that I already had a working robot code from my study project, running on the AVR Butterfly with working servo control, which I just had to extend about the communication with the CC2540. In case the the single mode strategy would be chosen,the code would have to be ported onto the CC2540.
Unfortunately, implementing the dual mode strategy turned out to be more difficult than expected in my case. This is mainly due to the lack of a template for dual mode peripheral devices on the CC2540 side and to the rather restrictive BLE API of iOS. This chapter is first introducing the dual mode approach and discussing its problems, then explaining the single mode approach, which was actually implemented in the ending.

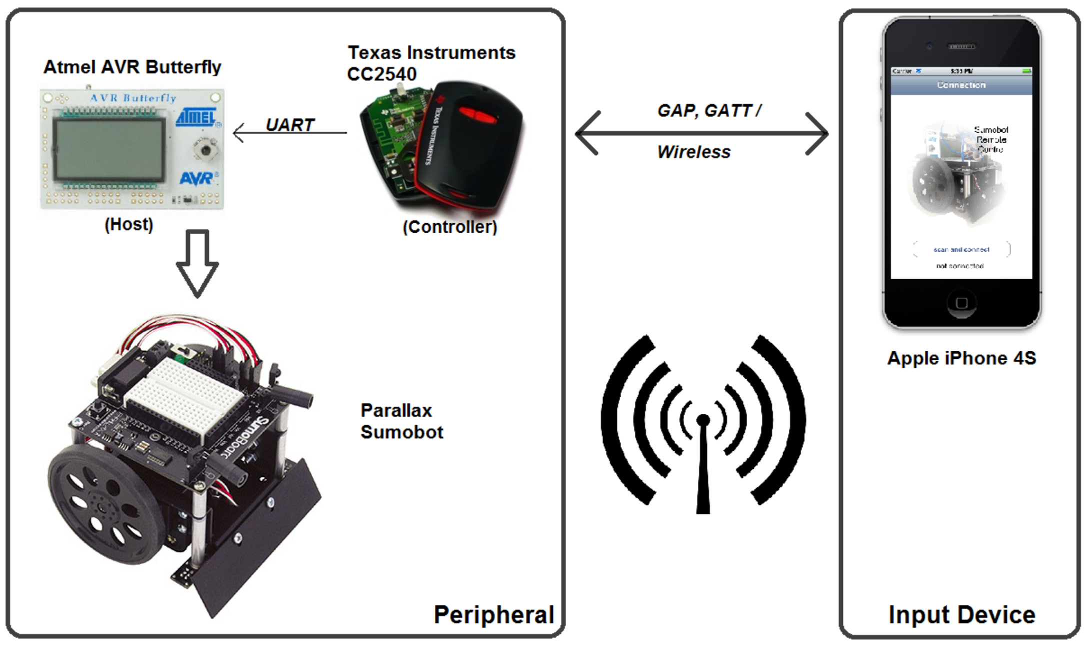


Figure 4.2: *Overall system architecture showing the communication between the different subsystems. Refers to single mode strategy, in dual mode, the arrow for the UART communication between Butterfly and CC2540 would go in both directions, because the CC2540 is set to Advertising by the Butterfly.*

As it would be very difficult to manually select a feasible input device device onside the robot, the iPhone should work as the central device in order to make the user able to select a device to connect to, and the robot controller, respectively the CC2540, should be set as a peripheral device in terms of connection initialization. Concerning the established connection, the most plausible and easy to implement solution is to set the iPhone as a client and the CC2540 as a server. As the CC2540 application is basically just transmitting incoming commands over UART (more or less in both operational strategies), the Butterfly application simply needs to listen and answer to incoming requests on the UART interface, which is very close to the way a server is supposed to act in Bluetooth communication. In order to send the movement orders requested by the user, the iPhone is waiting for user input and then needs to actively communicate with the robot on the network and therefore it is most reasonable to make it the client.

4.1.1 connection interval

The connection interval should be as high as possible in order to save energy, without causing a visible delay between the user input and the robot movement. It has to be set on both edges of the Bluetooth communication in order to ensure a reliable data transmission. According to [13, 24], the human brain can retain an individual image for one-fifteenth of a second and therefore believes an event to be continuous if it receives a second image within that time. Thus, the delay between the iPhone input and the Sumobot movement should be no more than $\frac{1}{15} = 0.0\overline{6}$ seconds. Practice testing brought me to choose a connection interval of 0.055 seconds with an overhead of $11.\overline{6}$ms for data transmission and processing on both sides. The unit for the interval is set to 1.25ms by default, thus the actual value written is $\frac{55}{1.25} = 44$.

The slave latency of the CC2540, determining how often the peripheral may ignore connection intervals in order to save energy, should be disabled (set to 0) in order to ensure that the slave is continuously forwarding all received movement commands from the master device. This can be a source of higher power supply, but it is necessary because the reliability is more important than the power supply in this project.

4.2 Sumobot application

The following section is dealing with the Sumobot application, in detail the AVR Butterfly, the TI CC2540 and the communication between those two subsystems.

4.2.1 dual mode strategy

When choosing the dual-CPU strategy, the purpose of the CC2540 software is just to translate incoming HCI commands to BLE stack calls, and vice versa, sending equivalent HCI commands to messages received from the BLE stack. Fortunately, TI already provided a simple application with the BLE stack and tools package for PC, which does exactly the same thing, but is mainly designed for using the master configuration and this needs additional configuration - details in the Realization chapter.
The whole process of initializing the CC2540 and connection establishment is visualized in the following figure.

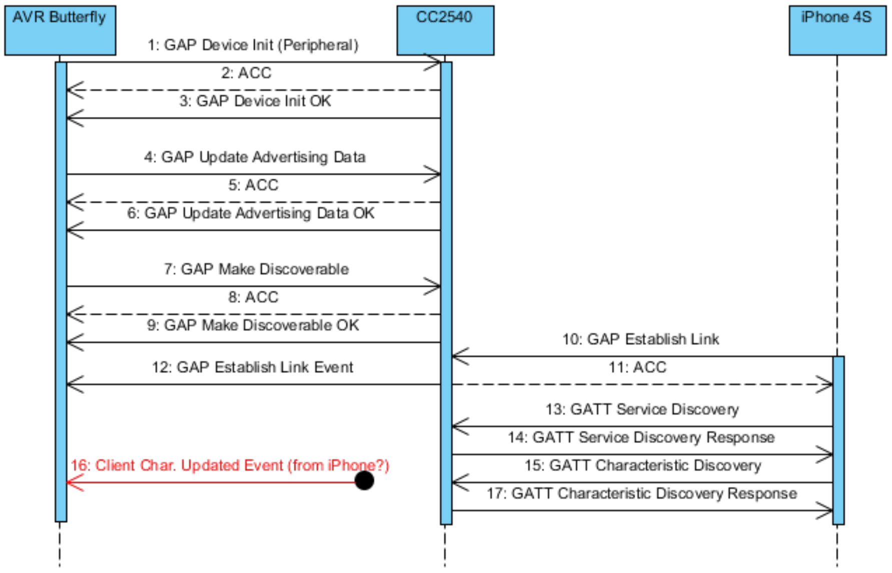

AVR Butterfly
CC2540
iPhone 4S
1: GAP Device Init (Peripheral)
2: ACC
3: GAP Device Init OK
4: GAP Update Advertising Data
5: ACC
6: GAP Update Advertising Data OK
7: GAP Make Discoverable
8: ACC
9: GAP Make Discoverable OK
10: GAP Establish Link
11: ACC
12: GAP Establish Link Event
13: GATT Service Discovery
14: GATT Service Discovery Response
15: GATT Characteristic Discovery
16: Client Char. Updated Event (from iPhone?)
17: GATT Characteristic Discovery Response

Figure 4.3: *Sequence diagram of initialization and connection establishment process in dual mode.*

The host is receiving the client requests via HCI events and sending the replies as HCI commands to the controller (see figure below). This is not the most common use of the HCI protocol, as in most cases the master device, for example a PC, is using the dual mode strategy when working with a Bluetooth adapter (see typical use at [6, 33]) - still, it is possible and wanted in this project, when the dual mode strategy is chosen for the slave device.

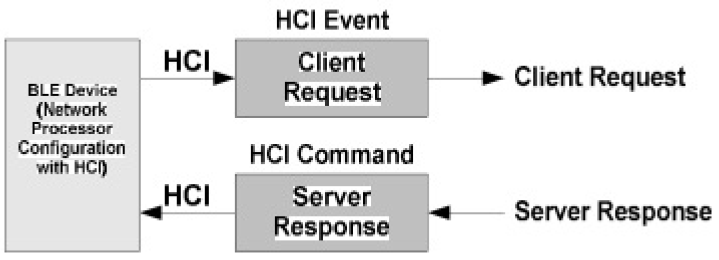


Figure 4.4: *HCI communication scheme for the CC2540 acting as a server, part of [17, 23]*

UART communication

The USART interface of the AVR Butterfly is used for receiving and sending HCI commands, events and data from and to the CC2540. As HCI communication is usually asynchronous [8, 4.1.2], the USART interface is used in asynchronous mode and therefore referred to as *UART* from hereon.
The CC2540 HostTestRelease template is given default UART settings, which are a baudrate of 57600, 8 bit data packages with one stop bit and no parity bit. Initially, hardware (RTS/CTS) flow control is enabled, which is not supported by the AVR Butterfly and therefore should be disabled. Looking at the Butterfly, having the given baudrate in mind there is a 0,0% error rate in data flow when decreasing the system clock speed by a minimal amount to 7372800Hz [1, 176]. This means that before using the UART the system clock has to be calibrated first, which can be done using the internal oscillator of the Butterfly. Because the sent HCI data packages can be relatively big and flow control is switched off, the Butterfly should respond on incoming data using the UART RX complete interrupt and store the content in a message queue in order to quickly make space for the next incoming byte. The message queue should be big enough to store at least one complete HCI event including data content.

HCI command interpretation

The HCI commands necessary for initializing the CC2540, establishing a connection and receiving movement commands have been retrieved via trial and error testing, using the CC2540 USB dongle and sending and analyzing HCI commands over the PC with BTool. The evaluation of the data communication resulted in the following communication protocol for the AVR Butterfly:

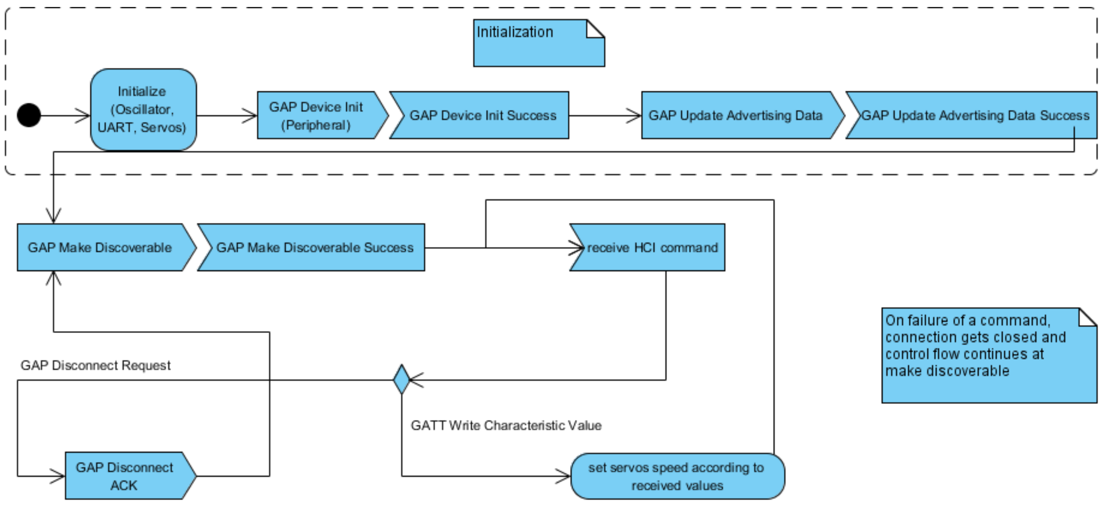

Initialization
Initialize (Oscillator, UART, Servos)
GAP Device Init (Peripheral)
GAP Device Init Success
GAP Update Advertising Data
GAP Update Advertising Data Success
GAP Make Discoverable
GAP Make Discoverable Success
receive HCI command
On failure of a command, connection gets closed and control flow continues at make discoverable
GAP Disconnect Request
GATT Write Characteristic Value
GAP Disconnect ACK
set servos speed according to received values

Figure 4.5: *Activity diagram of the Butterfly application in dual mode and the HCI handling in particular.*

The initialization task consists of the hardware initialization of the Butterfly itself, then setting the CC2540 as peripheral device and setting up advertising data. The advertising process is started with GAP Make Discoverable. Once a connection is established (which is handled by the CC2540), the Butterfly is listening for and interpreting incoming HCI events like described before. HCI commands and events are organized in structs, according to the command / event format seen in the Background chapter. Often-used codes such as *HCI_EVENT* are predefined, less-used statements hard-coded. The current system state is hold in a global variable - it can be INITIALIZE, ADVERTISE or CONNECTED.

problem

The solution approach described in this section has many advantages, as already mentioned, the biggest of them being handing the control of the BLE chip, in this case the CC2540, to the application, without the need to make adjustments on the transmission functionality. Unfortunately, during the application design and implementation process several problems were encountered, which made it impossible to carry on the the dual mode approach and implement a working solution within the time boundaries of this thesis project.
The TI-supplied HostTestRelease project is at the current stage designed for mainly working in master configuration - when using the device as peripheral, there have to be several changes made, as already discussed. In addition to that, during the implementation of the it turned out that the application doesn't actually handle GATT discovery requests by itself as it should, but simply drops them because the GATT services for handling service and characteristic discovery requests don't exist. Though, the HCI host is receiving a client characteristic updated message from the controller at about the same time the discovery requests are triggered (see figure 4.3). Implementing the handling for these requests on the Butterfly side would result into huge additional workload, especially when considering that it should be expandable to future real-world applications. On the other side, at the current time it is unfortunately not possible to just send GATT data to a random characteristic UUID from the iPhone, because the iOS Core.Bluetooth API does require the UUID to be in the list of found characteristics and thus a characteristic discovery has to be executed first.
This design impasse led to the decision to implement the desired system in a different way, by using the single mode strategy and transmitting just the plain movement values over UART.

4.2.2 single mode strategy

The use of the single mode strategy makes the need of the HCI protocol obsolete, therefore, the CC2540 application is based on another application template, the SimplePeripheral project, which as the name connotes is specifically designed for peripheral applications. It offers a demonstration GATT profile called Simple Profile, which can be extended to transmit any data written to its characteristics over UART to the Butterfly. Advertising can be enabled initially, which largely reduces the amount of communication needed between the Butterfly and the CC2540 (see following figure); but on the other hand it increases the workload for possible future changes and extensions, as the CC2540 is not forwarding the whole GATT communication, but only the movement commands on the single GATT profile.

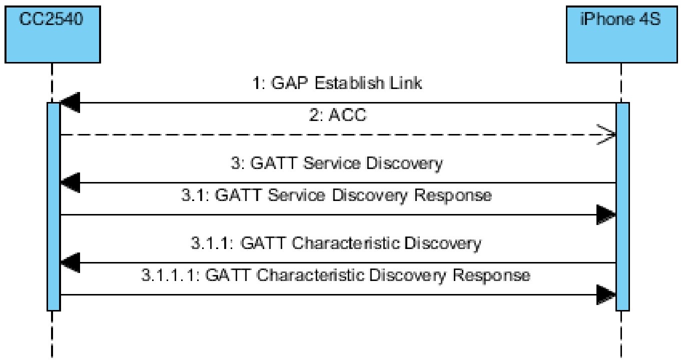

CC2540
iPhone 4S
1: GAP Establish Link
2: ACC
3: GATT Service Discovery
3.1: GATT Service Discovery Response
3.1.1: GATT Characteristic Discovery
3.1.1.1: GATT Characteristic Discovery Response

Figure 4.6: *Sequence diagram of the connection establishment process in single mode.*

Yet, it largely simplifies the Butterfly application, as the movement data fits into a one-byte value and hence no HCI command interpretation is required on the Butterfly. This is well visible comparing the following sequence diagram of the single-mode butterfly application to the one in the dual-mode section:

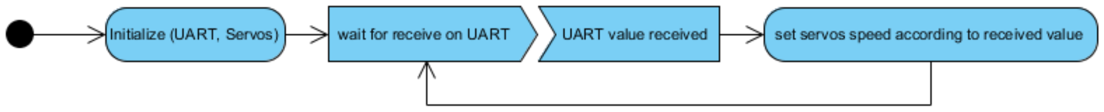

Initialize (UART, Servos)
wait for receive on UART
UART value received
set servos speed according to received value

Figure 4.7: *Activity diagram of the Butterfly application in single mode.*

As the sent UART data consists of just 1-byte values instead of more complex HCI commands and events, there can be several simplifications made on the UART driver. First of all, no interrupt-triggered UART receive or message queue are needed anymore, a poll-and-wait for incoming data is enough. In case an incoming message is missed because of an overwrite, the application should execute the most recent one anyway - as the application logic in this project is reduced to just interpreting the incoming messages into servo movements, there is no delay to expect which might influence the reliability or reaction speed aspects in any way.

4.3 iPhone application

The iPhone application is coherent no matter if using single or dual mode on the CC2540 side. The movement commands are sent to a GATT service and characteristic which is offered by the CC2540 in single mode, in case dual mode is used, the same values can be used as then the whole GATT traffic is getting forwarded to the Butterfly.
The application is based on the Bluetooth Low Energy example application for the CC2540,

supplied by Texas Instruments, which is an Objective-C application accessing the Core. Bluetooth API of iOS and utilizing the Delegate and the MVC patterns, which are considered of been well-known design patterns. The approach taken for this project was to extend the TI example application by changing it to a multi-tab application.

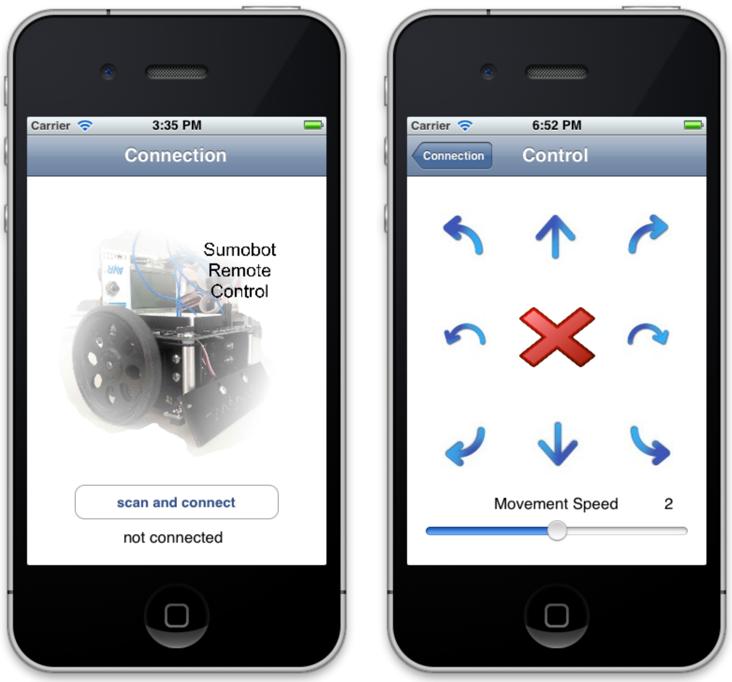


Figure 4.8: *UI concept of the iPhone app - the connection tab (left) and the control tab (right).*

4.3.1 Tab "Connection"

The UI of the connection tab and the code responsible for the connection establishment is mainly based on the TI example project and using the functions of the iOS Core Bluetooth Layer, which allows a very high-level approach.
It is the initial screen of the iPhone application and mainly consists of one button which, when pressed, is searching for advertising devices in range and connecting to the first one found. On successful connection establishment, a GATT service discovery is executed and afterwards a characteristic discovery on each found service. The user then sees a message saying "connected" or "no device found" accordingly.
Unfortunately, due to the restrictive iOS Core Bluetooth API, a Service/Characteristic discovery is necessary in order to send Data over GATT, because the API is checking the desired destination characteristic against the list of known characteristics. The discovery process takes a much larger amount of time than the connection establishment itself. As it would take way too much time to reconnect and rediscover the characteristics at each button press, the connection is established permanently until the user presses the button again. This increases the amount of energy needed and is contra the BLE principle of minimum-time connections, but due to the iOS API restrictions and the nonfunctional requirement of no visible delay, it is necessary in this project.

4.3.2 Tab "Control"

The control tab allows the user to send movement commands to the peripheral in order to control the robot. It can be reached from the connection tab using a swipe finger gesture from the right to the left. Swiping back to the connection tab is not possible, because

the user could accidentally press a button which would negatively influence the reliability aspect - instead, the user has to press a button in the top left corner to come back to the connection tab.
The view is organized into grouped buttons showing direction arrows for moving forward, backward, left, backwards left, right, backwards right, turning back and right and stopping as well as one slicing element determining the desired movement speed on a scale from one to three. On the pressure of one of the buttons, the action combined with the movement speed is sent in a GATT characteristic write message to the CC2540 with the service and characteristic UUIDs belonging to the SimpleProfile service of the TI Simple Peripheral template - when using the dual mode strategy, the UUIDs are theoretically irrelevant as the whole GATT communication should be forwarded by the CC2540 application.
Altogether this results in 19 possible states, which can easily be stored in one data byte with even allowing another 13 combinations for future project extensions or parity checks. In order to reduce the network traffic to a minimum and save power, new control commands should only transmitted at changes of movement - thus, only one control order is sent per button press, even if it is constantly pressed.

Chapter 5

Realization

5.1 Installation and Assembly

5.1.1 Assembling and connecting the hardware

The assembly of the Sumobot parts is very straightforward and can easily done regarding the detailed explanations of the assembly manual supplied by Parallax; the reader may notice that in this project no use of the included QTI and IR sensors was made. The butterfly can be fixed on top of the sumobot. As the UART communication makes it consume a pretty high amount of energy, which would lead to the need to regularly change the coin cell battery, and as it is an annoying task to pull out the coin cell battery each time the device should be turned off in addition to using the switch of the Sumobot, the Butterfly should instead get it's power directly from the four 1,5V batteries of the Sumobot, which significantly reduces the amount of maintenance work needed. To accomplish this, the according voltage/ground pins of the Butterfly, e.g. on port 0, need to be connected to volatage/ground on the Sumobot board.
The Sumobot pins for controlling the servo motors are P12 (right servo) and P13 (left servo) , which should be connected to port 0 P5 respectively P6 of the Butterfly.

The UART connection between the AVR Butterfly and the TI CC2540 requires connecting the TXD/RXD/GND pins of one UART port of each board with each other, whereas the TXD and RXD cables have to be crossed over, which means that the write output of one board is the read input of the other. The Butterfly only has one UART port and therefore does not require to make a choice. For the exact cable connections to be done, see figure 5.1. The CC2540 provides two UART ports with the exact same functionality, the different connection possibilities can be viewed in the following figures.

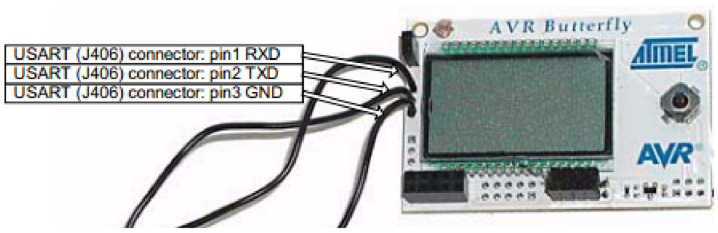
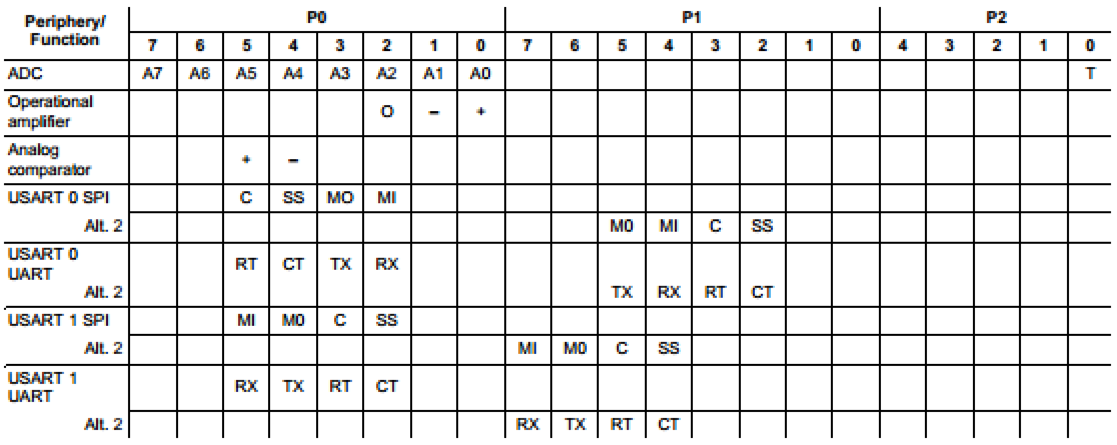

USART (J406) connector: pin1 RXD
USART (J406) connector: pin2 TXD
USART (J406) connector: pin3 GND
AVR Butterfly
AIMEL
AVR

Figure 5.1: *cabling of the UART connection at the AVR Butterfly. [11, 22]*



Periphery/ Function	P0								P1								P2				
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	4	3	2	1	0
ADC	A7	A6	A5	A4	A3	A2	A1	A0													T
Operational amplifier						O	–	+													
Analog comparator			+	–																	
USART 0 SPI			C	SS	MO	MI															
Alt. 2											M0	MI	C	SS							
USART 0 UART			RT	CT	TX	RX															
Alt. 2											TX	RX	RT	CT							
USART 1 SPI			MI	M0	C	SS															
Alt. 2									MI	M0	C	SS									
USART 1 UART			RX	TX	RT	CT															
Alt. 2									RX	TX	RT	CT									

Figure 5.2: *cabling possibilities of the UART connection at the CC2540. [19, 83ff.]*

5.1.2 Deploying to the CC2540

For using the CC2540 USB dongle for testing and measuring connection constraints, download and install the TI BLE software stack and tools[1]. The installation is processed via a wizard which is very intuitive to use - after successful installation, the development tools and various manuals can be found in the selected installation folder (under Windows by default C:/Texas instruments). The dongle then can simply be plugged into a free USB slot of the PC and can normally be instantly used. Under Windows, there might be an error stating that the driver for the dongle was not found - usually, manually selecting the installation directory of the BLE software stack as driver source solves the problem.

For deploying the code onto the CC2540 development board (also called "keyfob"), the tool SmartRF Flash Programmer, also developed by Texas Instruments, is required. It contains the driver for accessing the CC debugger and a simple interface for connecting to the debugger. I used a program called SmartRF Studio[2] which already contains the Flash Programmer and various other tools for evaluating and testing the device which make the development work a lot easier. The single Flash Programmer tool is also available online[3] and easy to install as well.
The supplied debugger then has to be connected to the debug port of the CC2540, as seen in the figure below. Extra care should be taken about the direction of the wire, it should be the same as in the figure, details about the pin connections can also be taken from the official TI quickstart manual. The USB port of the debugger should now be connected to a free USB port at the development PC. After a possible automatic driver recognition, the LED on the debugger should now be green. A red LED usually indicates a hardware error or wrong cable configuration, another reason could be an empty or not properly placed coin-cell battery at the CC2540 keyfob. If the LED is not on at all, the cables are likely not connected properly.

[1]currently version 1.2.1, available for the most common operating systems at http://www.ti.com/blestack (you need to register a free account first)

[2]available for free at http://www.ti.com/smartrfstudio - the used version was 1.7.1 on Windows 7, but there are also download packages available for other popular operating systems. Windows XP upwards should detect the device and associate it with the proper driver automatically - warnings that the driver is not officially certified can be ignored

[3]http://www.ti.com/tool/flash-programmer

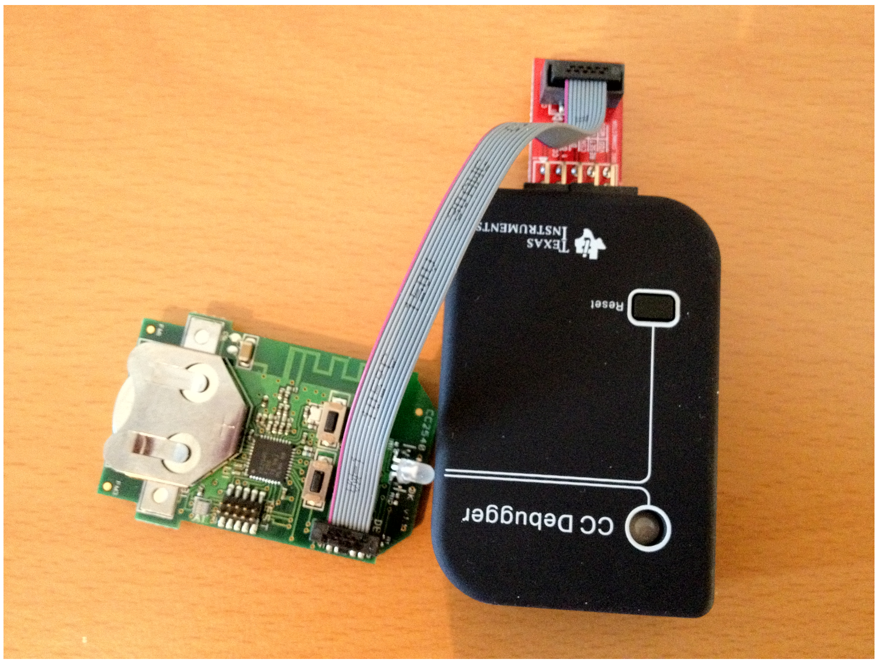


Figure 5.3: *proper connection of the CC2540 board with the CC debugger.*

When starting SmartRF studio for the first time, eventually a firmware update for the debugger is recommended to be installed. This can be done via the program interface without further required knowledge. A double click on the connected device then opens the specific device control panel, where all further testing and debugging configuration can be done.

As the Texas Instruments BLE stack is delivered in binary format only and containing compiler-specific optimizations, the only possible compiler choice is the IDE suggested by TI, which is IAR Embedded Workbench, at the current moment available in version 8.11.1. The software itself is commercial, but there is a free 30-day trial available online[4]. The installation procedure and the use of the program is similar to other IDEs.

5.1.3 Deploying to the AVR Butterfly

AVR studio is a powerful IDE for AVR-based microcontrollers provided by Atmel and the easiest way to program and debug the AVR Butterfly. It can be obtained for free via the official homepage [5] - to this date, the most recent version is 6.0, for the development of the thesis application version 4.19 was used. The installation process and the application interface is again very intuitive and similar to other standard products. An alternative development environment is the package compilation AVR-tools, which can be obtained for free as well. Likewise the CC2540, the code can be deployed to the Butterfly using the supplied hardware debugger.

5.1.4 Deploying to the iPhone 4S

Due to the restrictions of Apple, programming the iPhone is only possible via a Mac PC using MacOS X. But, given that constraint, programming and deploying to the iPhone is rather simple. MacOS X already provides the IDE XCode, which, beginning at version 3 and including the iPhone developer tools (which can be simply installed using the Add-Ons function of XCode), is able to deploy code on the iPhone 4S instantly.

[4]http://supp.iar.com/Download/SW/?item=EW8051-EVAL
[5]in several versions at http://www.atmel.com/tools/STUDIOARCHIVE.aspx

Another requirement though is the active membership in the iPhone developer program and an iPhone associated with that membership account. If the developer is studying or working at a university or company which is part of the developer program, this can be achieved easily by contacting the responsible administrator along with the specific iPhone device ID, which can be extracted of the iOS general settings. After having registered the device, the provided registration certificate just needs to be inserted into XCode. Afterwards, the code can be deployed out of the program using the USB cable which is provided with the iPhone.

5.2 Software Implementation

5.2.1 CC2540

The CC2540 application is based on a template project by Texas Instruments, which can be found in the BLE stack installation folder under /Projects/ble/SimpleBLEPeripheral. It is designed for the use of the chip in single and peripheral mode and already including the SimpleGATTProfile for testing purposes, containing five characteristics. The main function is initializing the hardware, OSAL and BLE stack and then calling the non-returning osal_start_system() function. GATT profile functions are registered in the GATT service as callback functions - e.g., a GATT characteristic write on a specific service UUID is triggering the registered callback GATT write function of the service with the according UUID.
The reader may notice that no special attention to UART receive functionality is paid in the CC2540 application, as well as to UART send in the Butterfly application, because the needed UART communication is only one-directional.

By default, the simpleBLEPeripheral project is not using UART communication. Hence, one UART port has to be initialized and opened. Both UART 0 and 1 would be able to communicate with the Butterfly by same functionality, in this implementation, UART 0 has been chosen without further intention. As the AVR butterfly UART does not support RTS/CTS flow control, this feature has to be switched off at initialization. The UART port initialization code, executed after HAL initialization, thus looks like the following fragment:

```
HalUARTInit();
halUARTCfg_t uartConfig;
uartConfig.configured          = TRUE;
uartConfig.baudRate            = HAL_UART_BR_57600;
uartConfig.flowControl         = FALSE;
uartConfig.tx.maxBufSize       = 128;
uartConfig.intEnable           = FALSE;
(void)HalUARTOpen( HAL_UART_PORT_0, &uartConfig );
```

In addition, for enabling UART communication the HAL_UART symbol has to be defined and set in the C compiler / preprocessor settings of the project, and the data direction of the TX pin has to be set to output.

The transmission of the received movement commands over UART is done in the simpleProfile_WriteAttrCB() function of SimpleGATTProfile, which is triggered on a received GATT write characteristic message on the according profile UUID. As the service-internal characteristics are not needed, no further check on the received characteristic UUID has to be done - all that needs to be done is transmitting the received value over UART:

```
if(len == 1)
        HalUARTWrite ( HAL_UART_PORT_0, pValue, len );
```

In comparison to the dual mode approach, control of the CC2540 from the Butterfly is not possible or would require the establishment of additional commands, leading the single mode approach ad absurdum as it would only result into a light imitation of the HCI protocol. Thus, advertising has to be enabled initially, as well as set to be enabled for an infinite amount of time - in simpleBLEPeripheral.c, where the advertising parameters are defined, setting *initial_advertising_enable = TRUE* and commenting out *uint16 gapRole_-AdvertOffTime = 0* is leading to the desired result. The slave latency flag on the same place is already set to 0 by default, as desired. The flag for minimum connection interval should be set to 44 - the maximum connection interval, determining the connection time-out at no activity, should be at least 80000 = 100 seconds and the supervision timeout should be 10000 to ensure a satisfactory user experience. In order for those settings to take effect, the enable update request flag has to be set in addition.

After execution of the steps stated above, the code then can be compiled in IAR and deployed to the device using the CC debugger and either the debug function in IAR or SmartRF Flash Programmer.

5.2.2 AVR butterfly

In order to achieve code which is reusable and easily adjustable to possible future feature implementations, the code is split up as far as possible into the program logic and abstraction layers for UART and servo movement.

servo control

The speed of the two controlled servo motors is manipulated using the internal 16-bit timer 1 using Pulse-Width modulation in phase and frequency correct mode with the option to clear on compare match and an 8-bit prescaler, by changing the values of its two *output compare registers* (OCR) according to the desired forward or backward speed, whereas an offset to a predefined NO_SPEED value is added. As the OCRs are internally mapped to port B 5 and 6, these pins should also be used to connect the servos to the Butterfly. Like already said, this thesis primarily focuses on the Bluetooth connection aspect, hence the reader is advised to look up third-party material when dealing with servo control [6].

UART

Analog to the focus on sending UART data on the CC2540, on the Butterfly only the receive functionality is needed. Hereby it is important that the same UART configuration settings as in the CC2540 application settings are chosen, meaning a baudrate of 57600, 8 bit data package size, no parity bit and one stop bit. There is no need to worry about turning off hardware flow control, as it is not supported on the Butterfly. After initialization, the UART data register should be flushed to ensure that there is no values left which were received in a previous run. UARTRx() is waiting for incoming data by continuously checking the Receive Complete flag - as discussed before, this approach of actively polling on the data register does not result in a performance loss as the only state changes of the system are servo speed adjustments triggered by received movement commands. Thus, the UARTRx() function looks like the following fragment:

[6] e.g. the already mentioned bachelor thesis, https://publications.theseus.fi/handle/10024/15164

```
char UARTRx(void)
{
    while(!(UCSRA & (1<<RXC)));
        return UDR;
}
```

program logic

The received byte is split into two bits determining the movement speed and three bytes determining the direction - the remaining three bytes are not used. The directions are predefined at the Butterfly and the iPhone side.
The main function is first calling UARTInit() with a UBRR parameter of (F_CPU) / (BAUD * 16) - 1, whereas F_CPU is the system frequency and BAUD is the desired baud rate (57600), and then calling motorInit(). The whole program logic after hardware initialization looks like this:

```
char mvCmd=0, move=0, speed=0;
while(1){
        mvCmd = UARTRx();
        move = mvCmd >> 2;
        speed = mvCmd & 3;
        if(speed == STOP){
                setSpeed(0,0);
                continue;
        }
        switch(move){
                case FORWARD:
                        setSpeed(33*speed, 33*speed);
                        break;
                case BACKWARD:
                        setSpeed(-33*speed, -33*speed);
                        break;
                (...)
        }
}
```

5.2.3 iPhone 4S

The application template provided by Texas Instruments is already providing basic functionality for communicating with the CC2540, where only few changes have to be made - the majority of the adjustments concentrates on the user interface.
All functional operations are working on a TIBLECBKeyfob object (t) using the delegate pattern for notification and MVC for button control. This object is now a singleton in order to make several view controllers able to make use of it. A TIBLECBKeyfob is working on an array of found peripherals, a Core Bluetooth Central Manager object and a CBPeripheral holding the currently connected peripheral. There are plenty of functions defined which are controlling the behavior of the peripheral - the important ones in this scope are now introduced.

The function *writeValue* sets the value of a GATT service characteristic of the specified connected device. The characteristic has to be in the found characteristics list. The newly implemented function *sendMovement:(Byte)moveVal p:(CBPeripheral *)p* is triggering writeValue() with the given movement command and the connected CC2540 device,

specifically to the first characteristic of SimpleGATTProfile, as parameters.
findBLEPeripherals:(int) timeout is executing a device discovery request for the specified time interval. *connectPeripheral:(CBPeripheral *)peripheral* is executing a connection request on one device, which has to be existing in the devices array beforehand. The responses to those requests are caught by two delegate functions, *didDiscoverPeripheral* and *didConnectPeripheral*, which are updating the peripherals array and the current peripheral, as well as the GUI elements accordingly, the latter of them then triggering services and characteristics discovery.
The functions *getAllServicesFromKeyfob* and *getAllCharacteristicsOnKeyfob* are basically just executing the underlying library functions for triggering services and characteristics discovery on a specified, connected device.

The user interface has been modified and extended using the storyboard UI editing system of the iPhone developer tools and extending the controller functions linked to the buttons of the interface. The behavior of the connection establishment has been improved, not used elements thrown out and new ones introduced. The GUI is now divided into two tabs, which are internally NavigationControllers: The connect tab, which is also the initial tab, and the control tab. On the connect tab, a label showing the connection status and possible error messages has been added. The tabs are linked with predefined swipe gesture and back button elements.
The *TIBLEUIScanForPeripheralsButton* function, linked to the connect/disconnect button of the connect tab, either executes a disconnect request and resets the connection handle if an old one is existing, or, if no device is currently connected, executes findBLEPeripherals with a two seconds timeout and afterwards executes *connectPeripheral* on the first device in the peripherals array, hence the first one found.
The movement direction buttons on the control tab are each linked to a separate function executing *sendMovement* with their specific command and the current value of the speed selector added. The command values are predefined and the same as on the Butterfly side:

```
- (IBAction)stop:(id)sender{
        [t sendMovement:0x00 p:[t activePeripheral]];
}
- (IBAction)moveForward:(id)sender{
        [t sendMovement:0x00+[[movementSpeed text] intValue]
                p:[t activePeripheral]];
}
 (...)
```

Chapter 6

Evaluation

After the implementation of the specified system, in this chapter several validation tests with regards to the functional and nonfunctional requirements are defined. Eventually, a summary of the achieved work and the current state in comparison to the initial concept, as well as an outlook on possible improvements and adjustments is given.

Testing

The evaluation on the fulfillment of the requirements can again be separated into testing of functional and nonfunctional behavior. Because there is an infinite number of possible input variations, real-world system safety cannot be guaranteed, but an approximation can be made by executing and validating practice tests covering possible wanted and unwanted scenarios.

The functional tests which are required to prove the successful implementation of the system concept are as follows:

T0 Connect to the device, verify the status label text, disconnect, verify the status label text again

T1 Connect, disconnect, reconnect, disconnect with verification of the status label text after each step

T2 Connect, switch to movement tab, iteratively execute all possible movements with all possible speed steps and verify on the behavior of the Sumobot, disconnect

T3 Connect, execute and verify a single movement, disconnect, reconnect, execute and verify another movement, disconnect

T4 Connect, try to execute several movements at a time, verify that only one of the executed movements is processed by the Sumobot, disconnect

T5 Try to connect without (activated) device in range, verify status label text

T6 Connect, execute single forward movement, disconnect, verify that the Sumobot is stopped

T7 Connect, execute single forward movement, disconnect, try to execute a movement, verify that the Sumobot is still not moving

F0 and F5 are given by the system architecture design and the implementation approach. F1, F2 and F7 are proven by T2 and T3, F3 is proven by T2 and T4, F4 is proven by T6 and T7, F6 is proven by T0, T1, T3, T5, T6 and by the implementation approach.
F8 is ensured through iOS design principle that allows an application to be opened just

once at a time and by the implementation of the connect button controller.

The nonfunctional requirements can be proved via user testing - a certain amount of participants given, which should be stakeholders chosen according to the target environment and have mixed abilities, age, project background etc.. The requirement of a maximum latency between user input and reacting motor movement can also be validated by analyzing the communication process and measuring the data round-trip-time from the iPhone to the Butterfly, which is rather complicated due to the necessary clock synchronization of those two devices and because of the time constraints of this thesis project not described in detail.

Summary

In this thesis, the development of an remotely controllable embedded system utilizing Bluetooth Low Energy has been described from the concept to realization and testing. In practice, the Bluetooth communication, has shown to be fully implemented and working - unfortunately, due to timing constraints of the project, it has not been possible to fully implement the UART communication between the devices on the peripheral side. After several debugging sessions, there can be said that a configuration option on the CC2540 might be set incorrectly which is preventing the device to transmit UART data, but the UART settings and register values seem to be set correctly and it is estimated that the issue is rather small and might be resolved with further debugging and more time given. As the focus of the thesis lies on the Bluetooth Low Energy communication, the main goal can be seen as achieved.

The development of this project was a very enriching experience. On the one hand, the desired system was a complex embedded application with three interacting subsystems utilizing several communication protocols and been written in several programming languages, which turned out to be challenging but very instructive task. On the other hand, the difficulties coming up during design and implementation phase concerning dual-mode approach and UART transmission were drawbacks, but turned out to be very rewarding lessons on situations that come up often during work life and that need to be dealt with.

Future improvements

The resulting system described and implemented is just a basic example on how energy-efficient short-range mobile device control can look like. Beyond that, there are many further enhancements possible, improving the behavior of the system, the user interface, or the possible operations of the peripheral application.

The implemented iPhone control interface, especially the UI of the movement control tab, is intuitive to understand, but rather simple and not allowing precise movements as desired in non-entertainment application areas. There are several other possibilities which come in mind, such as using the three-axis gravity sensor of the iPhone for control by motion - e.g., a forward tilting movement of the iPhone could result in a forward movement of the peripheral with the speed according to the tilting angle. Another possibility could be speech recognition, perhaps supported by the Sumobot-supplied line and IR sensors to detect borders or objects in the way. The iPhone 4S has a speech recognition system called *Siri*, but unfortunately it is not possible yet to address it out of custom-made applications.

By finishing this thesis project, there are also other mobile devices coming on the market supporting Bluetooth Low Energy, such as the new HTC One S [4] with the open source Android OS, which can be exploited for this kind of applications in future. Another possible change at the interface could be the offering of a list of devices in range and the possibility to connect to one specific or even multiple devices at a time.

Many discussed application areas require certain security and safety standards. The Bluetooth 4.0 standard is providing AES encryption out of the box, implementing a solution which is using encrypted traffic would be no big deal - additionally, the possibility of giving one or more peripherals one controlling iPhone as "owner" could be interesting. Analyzing energy supply and radio signal intermission resulting into the development of an environmental-friendly and cost-effective peripheral and according application would be another basis of an industrial use of the presented concepts, as well as insuring safety by reacting accordingly to unwanted situations such as connection loss, free fall or collision with objects.

The range of possible application areas for remotely-controlled energy-friendly robots is wide, such is the amount of extension options and new developments, and even standards such as Bluetooth might get replaced by new standards in near future.

Glossary and list of abbreviations

AES Advanced Encryption Standard, symmetric encryption system, also called Rijndael algorithm

ATT Attribute Protocol

BLE Bluetooth Low Energy

GAP Generic Access Profile

GATT Generic Attribute Profile

HAL Hardware Abstraction Layer

HCI Host-Controller Interface

IDE Integrated Development Environment

IEEE Institute of Electrical and Electronics Engineers (often called ”I triple-E”) - association responsible for developing and maintaining technological standards

iOS a proprietary operating system developed by Apple Inc. for mobile devices such as the iPhone 4S

IR Infrared light

OS Operating System

OSAL Operating System Abstraction Layer

RTOS Real-Time Operating System

SIG Bluetooth Special Interest Group, an association maintaining and developing the Bluetooth standard. A collaboration by various companies involved in Internet and network technology.

Sumobot a sport where robots have to push each other out of an arena - in this thesis related to the Parallax mini-sumo robot development kit ”SumoBot”

TI Texas Instruments Inc.

UART Universal Asynchronous Receiver/Transmitter, industrial standard for asynchronous communication between devices

USART Universal Synchronous/Asynchronous Receiver/Transmitter, like UART, but additionally providing functionality for synchronous communication

USB Universal Serial Bus, widely supported industrial serial communication standard

WiFi Wireless Local Area Network, commonly used IEEE standard for wireless network communication

Bibliography

[1] Atmel. *ATmega169(V) Datasheet*, 2006.

[2] Atmel. Avr butterfly, 2012.

[3] Bluegiga Technologies Bluegiga. Bluegiga enables the development of bluetooth® 4.0 accessories for iphone 4s, 2012.

[4] Sam Churchill. Htc one s: Android 4 and bluetooth low energy, 2012.

[5] John Donovan. Bluetooth low-energy: An introduction, 2010.

[6] Albert S. Huang and Larry Rudolph. *Bluetooth essentials for programmers.* Cambridge University Press, 2007.

[7] Nicole Lee. Bluetooth 4.0: What is it, and does it matter?, 2011.

[8] PaloWireless. *HCI Layer Tutorial*, 2012.

[9] Parallax. Basic stamp 2 microcontroller module, 2012.

[10] Parallax. Sumobot robot, 2012.

[11] Joe Pardue. *C Programming for Microcontrollers.* Smiley Micros, 2005.

[12] Gianluca Perotto. Bluetooth low energy: the technology behind the standard, 2012.

[13] Paul Read and Mark-Paul Meyer. *Restoration of Motion Picture Film.* Butterworth Heinemann, 2000.

[14] Sawyve. What is bluetooth 4.0?, 2012.

[15] Bluetooth Special Interest Group SIG. Sig introduces bluetooth low energy wireless technology, the next generation of bluetooth wireless technology, 2009.

[16] Bluetooth Special Interest Group SIG. Data transport architecture, 2012.

[17] Texas Instruments TI. *TI BLE Vendor Specific HCI Reference Guide v1.1*, 2011.

[18] Texas Instruments TI. Cc2540 (active) 2.4ghz bluetooth low energy system-on-chip solution, 2012.

[19] Texas Instruments TI. *CC2540/41 System-on-Chip Solution for 2.4-GHz Bluetooth® low energy Applications - User's Guide*, 2012.

[20] Texas Instruments TI. *Texas Instruments CC2540 Bluetooth Low Energy Software Developer's Guide v1.2*, 2012.

[21] Wikipedia. Bluetooth, 2012.

[22] Wikipedia. Bluetooth low energy, 2012.

[23] Wikipedia. Bluetooth protocols, 2012.