UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Steve Mägi

Tracking framework for object localization using Arduino

Bachelor Thesis (6 EAP)

Supervisor: Satish Srirama, PhD
Co-supervisor: Carlos Paniagua

Author: .... "...." May 2012
Supervisor: .... "...." May 2012
Co-supervisor: .... "...." May 2012
Professor: .... "...." May 2012

TARTU, 2012

Contents

1

Introduction

Mobile devices such as smartphones, digital cameras, personal navigation devices and handheld game consoles are quite popular these days. New technologies and improvements in hardware have made mobile devices more widespread. They offer various ways of communication. For instance, Bluetooth [1] is used for wireless data transfer between Bluetooth enabled devices such as headsets, personal computers, mobile phones, remote controls - anything that needs to exchange digital audio or other data over short distances. Furthermore, Bluetooth module is embedded in almost every smartphone today.

Android [2] is one of the popular smartphone platforms. It is an open source operating system for mobile phones. Developers can create software, which runs on Android mobile phones and makes use of its features, such as multi-touch screen, camera, accelerometer, GPS, gyroscope and Bluetooth. By using Bluetooth, Android phone can communicate with other devices, for example, Arduino [3] microcontroller. Arduino is an open source physical computing platform which is based on a microcontroller board. Arduino can be used for prototyping a hardware system which is easy to integrate with Android or other mobile technologies.

In this work we attempt to bring mentioned technologies together to develop a tracking framework for object localization. In addition, we analyse the prototype.

1.1 Motivation

Finding a lost mobile phone can be a real challenge, even if the phone was misplaced at home. Of course, if the phone was lost at home, it is possible to call the missing phone from another one and find the misplaced phone. But what if there are no other phones nearby? Then we would need another solution.

Besides mobile phone, the object that is important does not have to be an electronic device. A wallet is an example of an object that we want to keep close to us whenever we go. Especially, if the wallet was left in the pocket of a jacket in coatroom, then it may cause us discomfort. In worst case scenario, someone may steal it.

The object we care about can sometimes move away from us, but in some scenarios it is critical to keep it close to us. As an example, when a parent goes to a shopping mall with children, it is important for the parent to keep an eye on the children. Children may move away from the parent, but not too far.

There are always some objects that we want to keep close to us. It is important to know the location of these objects and to get notified whenever any of these objects moves away from us. Therefore we propose a tracking framework for object localization based on Arduino.

1.2 Contributions

For the problems mentioned in previous section we propose a solution - an object localization and tracking system. The system consists of one or multiple *Objects* which are connected to the central *Tracker* device. The *Tracker* is always attached to the user. For example, user carries it on his belt. The *Object* is a personal item, its location is important for the user. *Object* can be a mobile phone, keys with attached microcontroller as a keychain, a wallet with embedded microcontroller or microcontroller in a child pocket, among others. See an example of different *Objects* in the Figure 1.1. The system must notify user when an object of interest is left behind or help him find the object.

The *Tracker* has two major tasks. Firstly, it is responsible for managing connection between the *Tracker* and the *Object*. Secondly, it has to notify the user about events such as alarm sent from the *Object*, connection was lost to the *Object*, and new *Object*

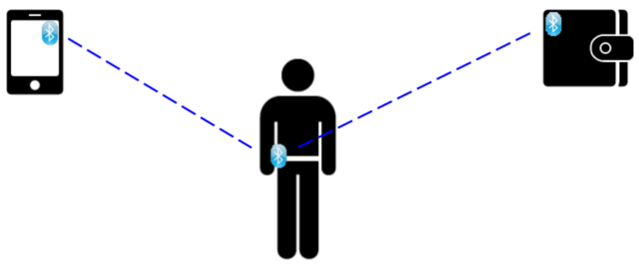

Figure 1.1: Tracker device attached to user's belt and two Objects connected to the Tracker.

was connected. In contrast, the *Object* is in a more passive role. Depending on the available hardware features, *Object* can send and/or receive alarms.

As for the *Tracker* we will be using an Arduino microcontroller with an integrated Bluetooth module. For *Object*, we will be using an Android smartphone which also has Bluetooth module.

1.3 Outline

Chapter 2: Introduces the state of the art of the localization framework and technology choices. Moreover, it gives an overview of the related work.

Chapter 3: The problem definition is discussed in this chapter.

Chapter 4: This section describes the framework for object localization.

Chapter 5: Explains performance and scalability of the framework.

Chapter 6: Provides conclusions and future work.

The Appendix A section contains the source code and binaries which were developed or used in this work.

2

State of the Art

This thesis combines different technologies that can be used for creating a tracking system for object localization. In the following sections each of the used technologies is described and explanation is provided about why this particular technology was chosen.

2.1 Bluetooth

Bluetooth [1] is a radio frequency based wireless communication technology which is based on the IEEE 802.15.1 standards [4]. Bluetooth was designed to be short-range and cheap replacement for cables for creating Wireless Personal Area Network (WPAN). WPAN is a network for interconnecting personal devices in close range. For example, Bluetooth can be used for connecting keyboard, printer, audio headset to a computer or mobile phone. Moreover, it is possible to create a bridge between mobile phone and a computer to share the Internet connection by using Bluetooth.

Bluetooth is often compared to Wireless Local Area Network (WLAN) [5] technology, which is also known as Wi-Fi [6]. Wi-Fi is similar to Bluetooth in means of data transmission, they both use the same range of radio frequency to transmit data. However, Wi-Fi transmission output power is higher. This makes the working radius larger than the Bluetooth has, but on the other hand, this also has bigger impact on the battery life. Depending on the class of the Bluetooth device, the connectivity range can be up to few hundred meters. A major disadvantage of using WLAN is that Wi-Fi devices need an access point such as Wi-Fi router to connect to other devices, whereas Bluetooth device can be connected directly to multiple devices simultaneously.

All things considered, the Bluetooth technology was chosen in this thesis because a wide range of devices already support Bluetooth compared to other WPAN technologies like IrDA [7] (Infrared Data Association) or ZigBee [8]. Furthermore, the connectivity radius of a Bluetooth device is large enough for short distance object localization.

2.2 Arduino

Arduino [3] is an open source platform for developing electronics prototypes, it is based on a simple microcontroller board and software. The software includes integrated development environment (IDE), compiler and a bootloader for the microcontroller. Programs can be written in Arduino programming language, which is very similar to C++ programming language.

Arduino board has an Atmel AVR microprocessor and input/output support via on-board pins. There are many official Arduino boards available which come in different shapes and sizes. As an example, two Arduino boards are shown in the Figure 2.1. Some

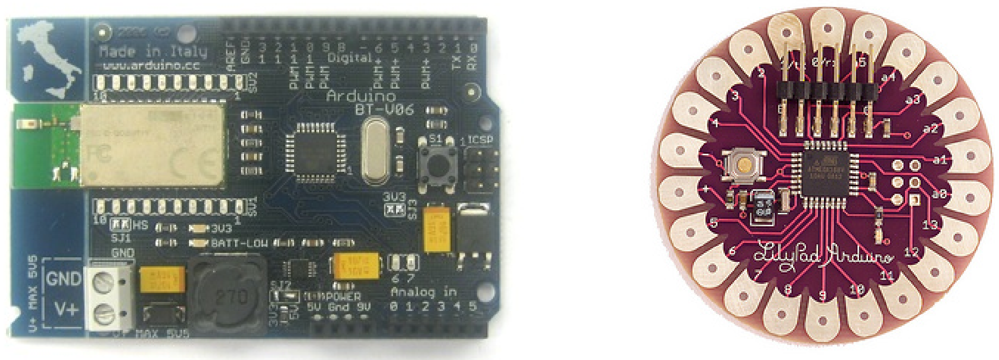

Figure 2.1: Arduino BT [9] (left) and Arduino LiliPad [10] (right).

of them may be coupled with additional modules, for instance, Arduino BT board has embedded Bluetooth module as shown in the Figure 2.1.

What makes Arduino boards very useful is that they have input and output support, meaning that the board can interact with the physical world. For example, by connecting light dependent resistor sensor into the board input pins it is possible to measure the light intensity falling on the sensor. Electric voltage values from analog input pins are converted into digital levels by using the 10-bit analog-to-digital converter (ADC) unit that is part of the microprocessor [11]. Similarly, light emitting diode (LED) can be connected to the output pins of the board. When board applies electrical power on the output pins, LED will lit up.

Compared to other microcontroller boards, Arduino is relatively inexpensive with open source platform and extensible hardware. Arduino has also a board which has already a built-in Bluetooth module which makes it easier to use with other wireless devices.

2.3 Android

Android is an open source operating system for mobile devices. According to Andy Rubin [12], who is the Senior Vice President of Mobile at Google Inc, over 300 million Android mobile phones have been sold and activated, which makes it one of the most popular operating system for smartphones.

Android architecture can be divided into 5 parts [13]: Linux kernel, Android runtime, Libraries, Application framework and Applications. Figure 2.2 shows the major components of the Android operating system.

The core of Android architecture is a Linux kernel which is slightly modified to match the special needs for mobile devices such as power and memory management. The kernel is a layer between hardware and rest of the software. Android runtime sits on top of the kernel, it consists of core libraries and Dalvik Virtual Machine (DVM). Every Android application process has an instance of DVM. The DVM executes files that are compiled from the Java language to the Dalvik Executable format. Applications can use a set of available general purpose libraries for managing databases, media, graphics and other resources. The application framework makes use of general purpose libraries and combines them into reusable components. Components help developers to create rich and innovative applications. Android applications are written in the Java language. Android operating system includes some core applications such as Phone, Contacts, Web browser, Email client and others.

A typical Android application contains one or more Activities. The Activity is a screen with a user interface. In addition, an application may have a Service. Service

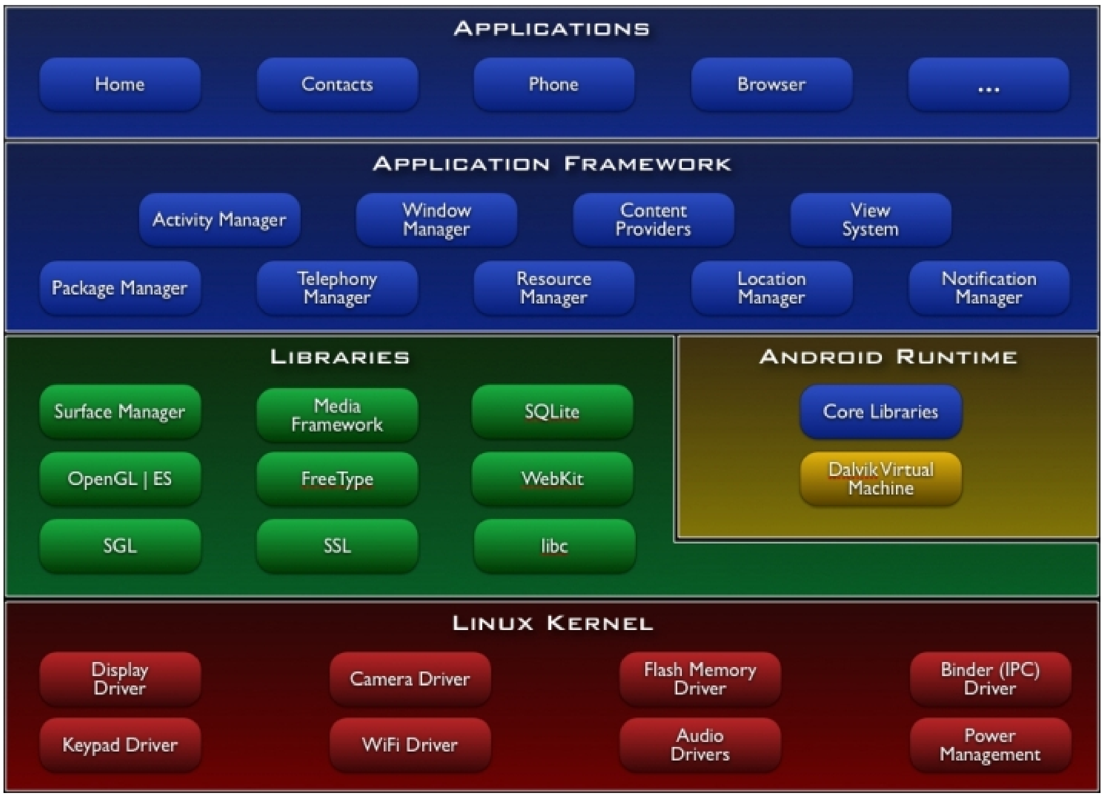

Figure 2.2: Android system architecture [13].

runs in the background and executes long running tasks, like music player. The difference between Activity and Service is that the Activity does not perform long operations and may be destroyed when another Activity is opened, but the Service has to complete its task before it is destroyed.

Services and Activities can be in different applications, yet it is possible for them to communicate with each other. Android has an interprocess communication system, the communication uses messages that are called Intents. Intent contains an information about the operation to be performed and it can also contain some extra custom information. Activity and Service are subclasses of Context. To send a message to another Context, one must create an Intent and broadcast it. To receive a message, Context must register broadcast receiver and set up an IntentFilter to filter Intents of interest.

Android smartphone was chosen in this thesis for the prototype object that needs to be localized because most of the Android phones have a Bluetooth module, also the development tools for Android are cross-platform and there are lot of resources about the development available on the Internet.

2.4 Amarino

Amarino [14] is a toolkit to connect Android mobile devices with Arduino microcontrollers via Bluetooth. The toolkit consists of two parts, ”Amarino” and ”MeetAndroid”. ”Amarino” includes an application for Android with a graphical user interface (GUI) and a library with an application programming interface (API) to communicate with Arduino. ”MeetAndroid” is a library for Arduino, it provides an API to exchange data with Android.

Furthermore, ”Amarino” offers a GUI to monitor Bluetooth connections. It also includes a background service that is called AmarinoService. The service manages Bluetooth connections, that includes connecting, disconnecting, sending and receiving messages from Arduino Bluetooth device. The GUI interacts with the AmarinoService by using Android Intents. Decoupling between the GUI and the AmarinoService allows other Android applications to use the same service as an interface to connect to Arduino devices. The library which comes with the ”Amarino” does not include the AmarinoService, it only has a wrapper to pass Intents to the service. In order to use the Amarino toolkit in a custom Android application the ”Amarino” application has to be installed.

On the Arduino side, ”MeetAndroid” library is used for sending and receiving data from Android device. Unlike the Amarino toolkit for Android, the ”MeetAndroid” library does not manage Bluetooth connections. Arduino does not provide standard API for Bluetooth, instead, each Bluetooth module may have its own API, for example IWrap [15] which Arduino BT uses.

Moreover, Amarino API offers a communication protocol which simplifies sending and receiving messages. A message has identifier and content. The identifier is a character such as ”A”, ”B”, ”C”, etc. Content can be a wide variety of data types, including string of characters, integers, floating point numbers and booleans.

On the whole, Amarino simplifies the setup and communication between Android and Arduino devices, it also provides tools for creating a protocol for the communication. There are no other similar libraries available, these were the main reasons for using Amarino toolkit in this thesis.

2.5 Object Tracking and Localization Frameworks

There are different approaches which can be used to create an object localization system. A way of categorize object localization systems is to use location sensing techniques [16] such as triangulation/trilateration, scene analysis and proximity. Triangulation and trilateration techniques use trigonometry between multiple objects to calculate their location. The triangulation technique uses location of two known objects, distance between them and angles to the object which needs to be localized to calculate its location. Trilateration uses location of two known objects and distances between all objects to calculate location of third object. Likewise, scene analysis observes the scene and makes conclusions about the location of objects. Further, proximity technique uses the physical contact, line of sight or wireless technologies to determine whether the object is near or away from the measuring object. These techniques can be subdivided into absolute and relative location systems.

In absolute location system, if one object measures or calculates other object's location, the result will be the same as if another object would have measured the same object's location. For example, when geographic coordinates of the same place are calculated by Global Positioning System (GPS) device or measured by using world map, then it doesn't matter which method was used, the longitude and the latitude of the result would be the same. In relative location system, the object's location is given relative to the measuring object's location. For example, if two objects are measuring the location of another object, then results can be different such as "next building" or "building across the street" which actually refer to the same building.

In outdoor environment, GPS [17] can be used to get accurate location information. GPS is a network of multiple satellites in the orbit of Earth, originally 24 satellites. Satellites are placed in the orbit so that at least 5 are in the view at any point on the Earth. Each of the satellites transmit navigational data which is used by receivers on Earth to calculate the location of the receiver device. GPS technology is an example of trilateration where the calculated location is absolute. Advantage of GPS technology is wide area coverage. In contrast, the disadvantages are: limited accuracy, not available for indoor environments, devices using GPS consume lot of battery power.

RADAR [18] is a Radio-frequency (RF) based indoor user location and tracking system which uses both, scene analysis and proximity techniques. The idea is that there are some wireless hosts with fixed locations in a building. Wireless hosts broadcast and listen data packets. The building is divided into 2-dimensional grid. In the setup phase, user visits all the sectors of the grid and broadcasts its absolute location and facing direction in the building. Wireless hosts listen this broadcast and store the data along with timestamp and the signal strength of the broadcast. Later, in the real time phase, this information is combined and can be used to calculate user's location based on the signal strength. The advantages of this system are that it doesn't require ad hoc wireless hosts, it can use existing RF wireless LAN (Local Area Network) devices. Disadvantages are that the RADAR can be used only indoor. If the room plan in the building changes or some bigger objects change place, then it may need recalibration. In addition, RADAR cannot track users on multiple floors of the building.

Reminder systems keep track of objects of interest and raise alarm when any of these objects are lost. The solution presented in this thesis is probably most similar to reminder systems. An example of such system is presented in Ubicomp 2004: Ubiquitous computing [19] where passive Radio-frequency identification [20] RFID tags on objects are used to generate reminders about objects left behind. This design is using proximity technique with relative location system. The advantage of this system is that it is not dependent on the environment like RADAR. The main disadvantage of this design is that it doesn't allow two-way communication between tag (transponder) and reader. This becomes a problem when the tag is lost and reader needs to send alarm to the tag.

In object search systems, every trackable object's location is stored in the central database when the tracker passes by. Later on, when an object needs to be located, the central database can be queried. In Hourglass [21], there can be several trackers which collect the data. Each tracker may use different sensors to identify objects. The database can be queried by another service which doesn't need to be the tracker itself. Depending on the trackers and their methods of data collection, this system can be combination of triangulation, scene analysis and proximity techniques. Location system can be both, absolute and relative. Advantage of this object search system is that it can track many objects simultaneously in a very large area. Disadvantage is that it collects and stores large amount of data which is unnecessary most of the time, the data is shared with all the participants who wish to use the localization system which may raise concerns about privacy.

3

Problem Definition

In everyday life, it is often important to know the location of objects which we are interested in. The location of immovable objects is easier to remember. The problem arises when we are dealing with smaller objects such as TV remote control, wallet or mobile phone, which can be moved or misplaced. In this case it may be sometimes hard to remember the location where the object was placed last time we used it. Even if we had a good memory, then someone else may move the object, so this is still a problem.

By specifying the problem a bit without simplifying it, we restrict the objects of interest to be personal objects (*Objects*). Personal objects like wallet, mobile phone, purse or back bag need to be close to us when we travel from one place to another. The owner of these objects carries a device called *Tracker*. In this scenario, we do not want to lose these objects. In other words, we need to know the relative position of the *Object* from the perspective of the *Tracker*. This problem can be divided into sub problems.

The first sub problem: **Is the *Object* in the vicinity of the *Tracker*?** To define the "vicinity" we say that the *Object* and the *Tracker* are both electronic devices and they can communicate over the wireless network, then the "vicinity" will be the connectivity range of the *Tracker* device. In essence, if an *Object* can connect to the *Tracker*, then the *Object* is in close range.

Knowing that the *Object* is in close range may not be good enough. For example, we do not want to move away from our mobile phone more than a couple of meters, but we can leave our back bag into another room. So the second sub problem is similar to the first one: **If the *Object* is in the vicinity of the *Tracker*, is it possible to evaluate the proximity of the *Object*?**

Usually, there are more than one *Object* that we want to track. For instance, if a person leaves from home then he wants to have a key chain and a mobile phone with him. Therefore the *Tracker* should be able to track multiple *Objects* simultaneously. This is the third sub problem: **Is it possible to track multiple *Objects*?**

Even if we know that the *Object* is close to the *Tracker*, then we still may not be able to find the *Object* because it may be hidden. For example, if the mobile phone was placed on the sofa and it fell down under the sofa, then it is still in the vicinity of the *Tracker*, but we cannot find it. If the *Object* is able make a noise or other action which helps the owner of the *Object* to identify its location, then the owner could use the *Tracker* to send a signal to the *Object* which will make the noise. The final sub problem: **Is it possible for the *Tracker* to send an alarm to the *Object*, which, in return, makes a sound so that a person can follow the sound and find the *Object*?**

To counter these problems this thesis proposes a tracking framework for object localization. The next chapter will discuss about the architecture of the proposed solution.

4

Object Localization Framework using Arduino and Android

In this chapter we propose the tracking framework for object localization based on the Arduino and the Bluetooth technology. The framework will be used for offering a solution for the problems described in the previous chapter.

To begin with, the system consists of multiple devices. One of the devices is called *Tracker* which tracks other devices called *Objects*. The *Tracker* is based on the Arduino microcontroller board with an embedded Bluetooth module. The *Tracker* has to be close to a person who wants to track multiple *Objects*. Considering that, it could be attached to a belt or put into the pocket. The *Object* is a device with a Bluetooth module, it can be embedded in items which do not have Bluetooth available such as back bag, thus making these objects trackable. Also, the *Object* itself can qualify as the object of interest, for example, a smartphone with a Bluetooth module. Different types of *Objects* are illustrated in the Figure 4.1. The bare minimum requirement for an object to qualify as the *Object* is that it must be able to establish a Bluetooth connection to the *Tracker* device.

4.1 Use Cases

In this section we give an overview of the available features of the proposed object localization system by describing common usage scenarios. As an *Object* we use an Android smartphone with the PhoneWatch application installed that was developed in

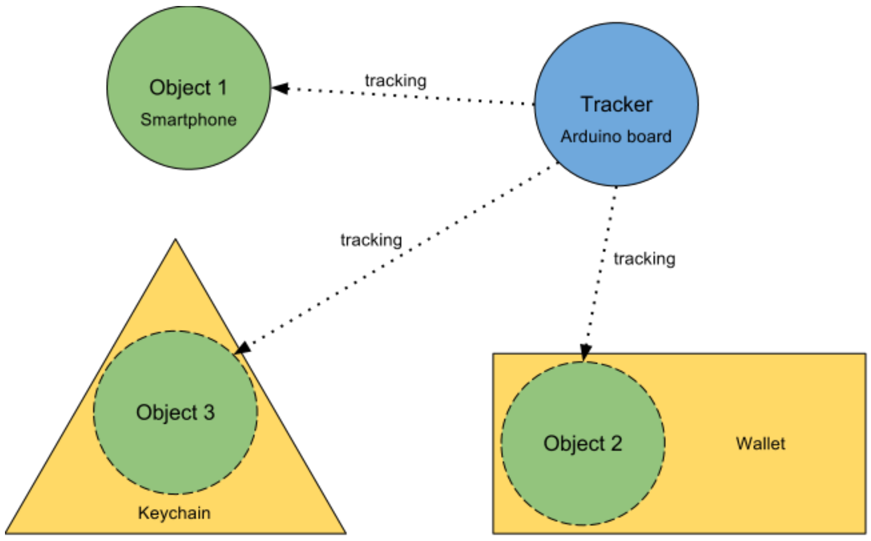

Figure 4.1: Tracker tracking multiple Objects.

this thesis. As a *Tracker* we use the Arduino BT microcontroller with the ObjectTracker software installed which was also developed in this work. To summarize, the following list contains the common use cases, more detailed information about each use case can be found in the referenced use case tables:

- Connect an *Object* to the *Tracker* (Table 4.1)
- Disconnect an *Object* from the *Tracker* (Table 4.2)
- Send an alarm to the *Tracker* (Table 4.3)
- Send an alarm to the *Object* (Table 4.4)
- The user sets the *Object* as "short range" device (Table 4.5)
- The user gets notified about an *Object* which was left behind (Table 4.6)

Use case number	1
Use name	Connect an *Object* to the *Tracker*
Description	The user connects the *Object* to the *Tracker* to start tracking it.
Actors	User
Preconditions	The *Tracker* is turned on and is ready to receive new connections. The *Object* is in the connectivity range of the *Tracker*. The user has started the PhoneWatch application.
Trigger	The user taps on the "Connect" button in the PhoneWatch application.
Expected result	The *Object* is successfully connected to the *Tracker*.
Main scenario	1. The user taps on the "Connect" button in the PhoneWatch application. 2. The PhoneWatch shows a message that the connection was successful.

Table 4.1: Connect an Object to the Tracker.

Use case number	2
Use name	Disconnect an *Object* from the *Tracker*
Description	The user disconnects the *Object* from the *Tracker* in order to stop tracking it.
Actors	User
Preconditions	The *Object* is connected to the *Tracker.*
Trigger	The user taps on the ”Disconnect” button in the PhoneWatch application.
Expected result	The *Object* is successfully disconnected from the *Tracker.*
Main scenario	1. The user taps on the ”Disconnect” button in the PhoneWatch application. 2. The PhoneWatch sends a Disconnect command to the *Tracker.* 3. The ObjectTracker receives the command and marks the connection as disabled. 4. The PhoneWatch closes the Bluetooth connection. 5. The PhoneWatch shows a message that the connection was closed.

Table 4.2: Disconnect an Object from the Tracker.

Use case number	3
Use name	Send an alarm to the *Tracker*
Description	The user sends an alarm from the *Object* to the *Tracker*. Useful when the user cannot find the misplaced *Tracker*.
Actors	User
Preconditions	The *Object* is connected to the *Tracker*.
Trigger	The user taps on the ”Alarm” button in the PhoneWatch application.
Expected result	The *Tracker* raises alarm.
Main scenario	1. The user taps on the ”Alarm” button in the PhoneWatch application. 2. The PhoneWatch sends an Alarm command to the *Tracker*. 3. The ObjectTracker receives the command and activates the alarm.

Table 4.3: Send an alarm to the Tracker.

Use case number	4
Use name	Send an alarm to the *Object*
Description	The user sends alarm from the *Tracker* to the *Object*. Useful when the user cannot find the misplaced *Object*.
Actors	User
Preconditions	The *Object* is connected to the *Tracker*.
Trigger	The user pushes the alarm button on the *Tracker*.
Expected result	The *Object* raises the alarm by making a noise.
Main scenario	1. The user pushes the alarm button on the *Tracker*. 2. The ObjectTracker sends an Alarm command to the *Object*. 3. The PhoneWatch receives the command and activates the alarm by showing notification and playing the alarm sound.

Table 4.4: Send an alarm to the Object.

Use case number	5
Use name	The user sets the *Object* as ”short range” device
Description	The user sets the *Object* as ”short range” device which suggests the *Tracker* to raise alarm when the *Object* leaves from the short range.
Actors	User
Preconditions	The *Object* is connected to the *Tracker*.
Trigger	The user taps the ”Short range” button in the PhoneWatch.
Expected result	The *Tracker* marks the *Object* as a ”short range” trackable object.
Main scenario	1. The user taps the ”Short range” button in the PhoneWatch. 2. The PhoneWatch sends an Alarm range command with the Short range parameter to the *Tracker*. 3. The ObjectTracker receives the command and marks the *Object* as a ”short range” trackable object.

Table 4.5: The user sets the Object as ”short range” device.

Use case number	6
Use name	The user gets notified about an *Object* which was left behind
Description	When the user leaves an *Object* behind and the *Object* moves out of the vicinity of the *Tracker*, the *Tracker* notifies the user with an alarm.
Actors	User
Preconditions	The *Object* is connected to the *Tracker*.
Trigger	The *Object* goes out of the vicinity of the *Tracker* (connection breaks) or the *Object* moves out of the ”short/long range” tracking distance, which was requested by the *Object*.
Expected result	The *Tracker* raises alarm.
Main scenario	1. The *Object* goes out of the vicinity of the *Tracker* (connection breaks). 2. The ObjectTracker receives NO CARRIER event from the Bluetooth module. 3. The ObjectTracker marks the connection as disabled and raises alarm.
Alternative scenario	1. The *Object* moves out of the ”short/long range” tracking distance, which was requested by the *Object*. 2. The ObjectTracker receives RSSI event from the Bluetooth module with a Bluetooth signal strength value that is lower than the minimum required for the ”short/long range”. 3. The ObjectTracker marks the connection as disabled and raises an alarm.

Table 4.6: The user gets notified about an Object which was left behind.

4.2 Architecture

4.2.1 Communication Protocol

In general, when two persons meet and want to exchange the information they use a language to accomplish this. Similarly, the communication protocol specifies the message format and rules for exchanging the information. It is needed for understanding the content and intention of the messages sent between the *Tracker* and the *Object*. By understanding the communication protocol it is possible to develop a new *Object* without changing the existing software on the *Tracker* device.

At this level, to pair and to connect two Bluetooth devices is out of the scope of this communication protocol, it is the responsibility of the Bluetooth module itself. The Bluetooth module automatically accepts all connections coming from the paired devices. The communication is initiated when the *Object* connects to the *Tracker*. The *Tracker* will be notified of this event, however, it does not send any response back. Even more, the *Tracker* does not send any messages at all if the *Object* has not requested to be notified about special events such as alarm. This is to maximize the flexibility of the possible *Objects* which could be used with the *Tracker*. If an *Object* is able to connect to the *Tracker*, then it is possible to track this *Object*. Further, the communication ends when the connection breaks. Although, it is possible that the *Object* can send a disconnect command before it closes the connection. In case if the *Object* supports more than just opening the connection, the *Tracker* supports various commands such as raising alarm or setting the distance at which the *Tracker* activates alarm when the *Object* moves away from the *Tracker*. These and other commands are described in the following paragraph.

The message format used in communication protocol has two parts: *Command* identifier and *Parameter*. Both, *Command* and *Parameter* are represented with a character. The character is actually one byte (256 different possible values), but only values of ASCII [22] characters from "A" to "Z" are being used. The message content is encapsulated into Amarino message format which wraps it between two bytes, respectively ASCII codes 18 and 19. The following Table 4.7 contains all the available commands.

Command	**ASCII character**	**Parameter values**	**Description**
Alarm	A	none	Tells to activate the alarm feature.
Enable feature	E	A	An *Object* can send this command. By default, the *Tracker* expects no features enabled. It means that the *Object* supports the feature defined by the parameter value: • A - When an Alarm command is received, the *Object* will activate its alarm.
Disable feature	D	A	The *Object* can send this command. The *Object* tells that it does not support the feature defined by the parameter value: • A - The *Object* does not want to receive the Alarm command.
Alarm range	R	S, L, C	The *Object* suggests the *Tracker* from which distance it should raise an alarm when it leaves from the vicinity of the *Tracker*. The distance is given by the parameter value: • S - Short range. • L - Long range. • C - Connectivity range, the connection is broken. This is the default setting if the *Tracker* does not receive this command.
Disconnect	T	none	The *Object* sends this before it disconnects, the *Tracker* will not raise the alarm then.
Status	S	none	The *Tracker* sends this in every minute. It was needed for the battery experiment.

Table 4.7: Communication protocol commands.

4.2.2 Tracker

4.2.2.1 Hardware Realisation

The physical architecture of the *Tracker* contains 5 major parts: Arduino microcontroller, Bluetooth module, power supply, sensor and actuator. All of these components can be seen in the Figure 4.2.

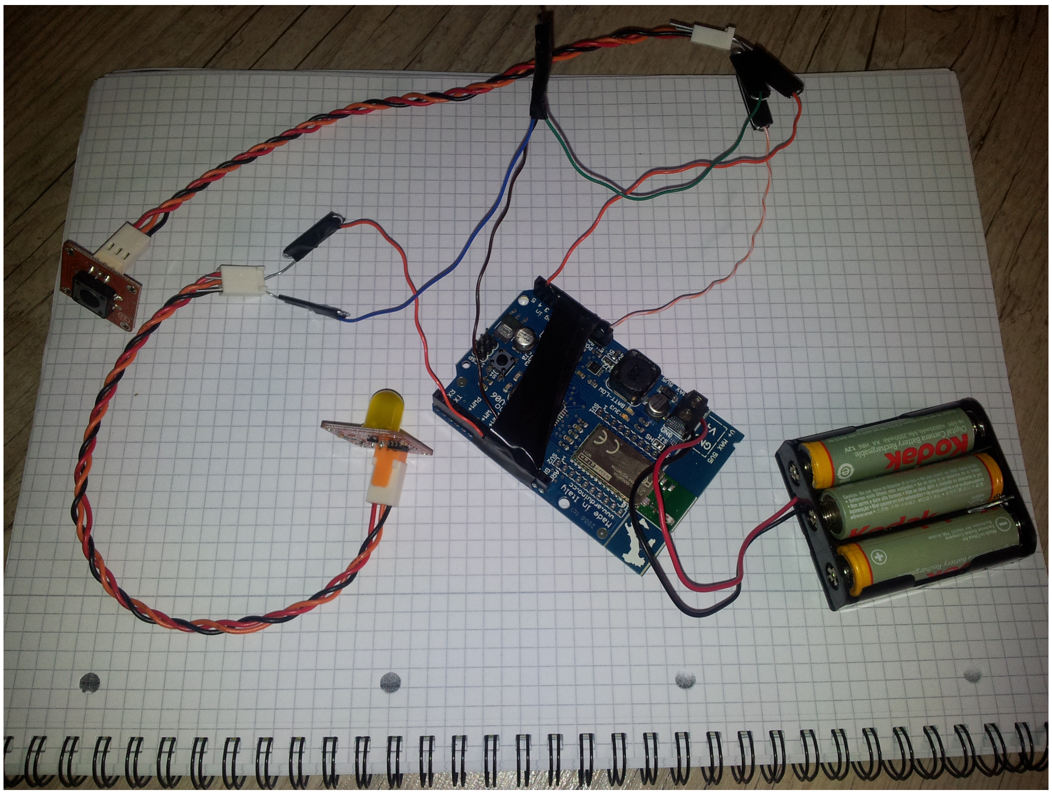

Figure 4.2: Tracker.

The core of the *Tracker* is an Arduino BT-V06 [9] microcontroller board. The Arduino BT is based on the ATmega328 microcontroller and it has a Bluegiga WT11-A Bluetooh module embedded on the board. Here are the characteristics of the microcontroller:

- **Microcontroller** ATmega328
- **Operating Voltage** 5V
- **Input Voltage** 1.2-5.5 V (uses DC-DC converter to convert to 5V)

- **Digital I/O Pins** 14 (of which 6 provide PWM output)
- **Analog Input Pins** 6
- **DC Current per I/O Pin** 40 mA
- **DC Current for 3.3V Pin** 50 mA
- **Flash Memory** 32 KB (of which 2 KB used by bootloader)
- **SRAM** 2 KB
- **EEPROM** 1 KB
- **Clock Speed** 16 MHz

The microcontroller and the Bluetooth module are integrated in the same board, however, the only bridge between the Bluegiga module and the microcontroller is the serial connection via RX (receive pin 0) and TX (transmit pin 1) pins. The Bluetooth module has two important purposes. Firstly, it is used for communication between the Arduino BT and other Bluetooth devices. Secondly, while some Arduino boards use the USB port for uploading new program to the board, then on the Arduino BT the Bluegiga module is the only interface which can be used for this task.

The Bluegiga WT11-A [23] is a Class 1, Bluetooth 2.0 module. It is an autonomous unit with its own processor and memory. The following list contains more technical information about the module:

- Fully Qualified Bluetooth system v2.0 + EDR, CE and FCC
- Class 1, range up to 200 meters
- Integrated chip antenna
- Enhanced Data Rate (EDR) compliant with v2.0.E.2 of specification for both 2 Mbps and 3 Mbps modulation modes
- Support for 802.11 Coexistence
- 8 Mbits of Flash Memory
- 16-bit reduced instruction set computer (RISC) microcontroller

Furthermore, the IWrap [15] firmware runs on the Bluegiga module. The IWrap can be configured over the serial interface by using simple ASCII commands. The module supports up to 16 Bluetooth device pairings. By default, the firmware operates in the Data/Command mode which means that it is possible to configure the module or exchange data with the connected Bluetooth device one at the time by selecting between the Command or Data mode respectively. However, this method has some pitfalls, switching between the modes takes some time (over 2 seconds according to IWrap documentation). Moreover, if there are multiple connections then receiving and sending the data is difficult. Each time the data needs to be sent or received from another connection, first, the Command mode should be selected and then it is possible to select the next connection. In contrast, there is the Multiplexing (MUX) mode available, it allows the data and commands to be sent at the same time, without the need for switching between the Data/Command mode. Furthermore, it supports up to 4 multiple simultaneous connections. However, the disadvantage of the MUX mode is that all the data/commands have to be wrapped into frames using the IWrap MUX protocol so that the module would know where to send these frames.

As all electronic devices the Arduino BT board needs a power supply to operate. Because of the DC-DC converter on the board, it is possible to power the board with the minimum of 1.2V and the maximum of 5.5V power supply. Therefore, we are using the battery holder which has place for three AA type batteries. Rechargeable batteries of 1.2V each are being used, so the board is powered with 3.6V. Actually it can be up to 3.6V because the voltage drops when batteries drain. The Arduino BT has a special on-board LED that will lit up before batteries run out. The battery holder (B) is connected via two wires to the V+ and GND screw terminals on the board as can be seen in the Figure 4.3.

The *Tracker* has one alarm module attached to the microcontroller board. The purpose of the alarm module is to notify the user about events like when the object being tracked goes out of the vicinity of the *Tracker*. The alarm module is an actuator. The actuator performs an action when it receives power. For the alarm module we are using a Tinkerkit's LED [24]. The LED (A) is connected to the digital output pin 12 of the board, see Figure 4.3. When the microcontroller applies 5V electrical voltage on the output pin, the LED will lit up.

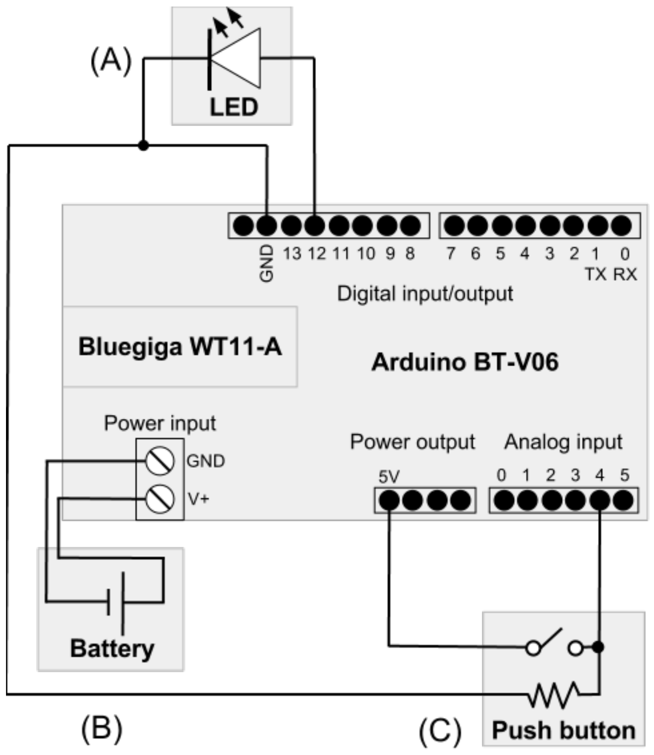

Figure 4.3: Tracker's wiring schematic.

Besides the alarm module, the *Tracker* has also a push button module. This module is a sensor. The sensor module outputs electrical voltage when it is activated by the environment. The push button (C) is connected to the analog input pin 4 of the board as shown in the Figure 4.3. We are using a Tinkerkit's Push button module [25], it outputs 5V when it is pressed and 0V when it is released. The purpose of this push button is to send an alarm command to the object being tracked.

4.2.2.2 Software Realisation

To make use of the components described in the previous section the software called ObjectTracker was developed. The software controls the interaction between the microcontroller and other modules. It consists of a program written in the Arduino programming language. The program is stored in the flash memory on the Arduino board. When the microcontroller is powered up, it starts the bootloader. It is a small program which has two important tasks. First, by using the bootloader it is possible to upload a new program onto the board. When the new program is uploaded to the board, the old one is overwritten. Second, the bootloader starts the program which was uploaded onto the board when the board is powered on.

Next, we describe the ObjectTracker software. The program architecture can be divided into multiple components as seen in the Figure 4.4. The following list gives a short overview for each of the components:

- **Communication manager** - Responsible for communicating with the Bluetooth module (follows the IWrap protocol by packing/unpacking data frames).
- **Event manager** - Responds to events coming from the Bluetooth module, checks and responds to the events, for instance, a button press.
- **Scheduler** - Creates events at regular intervals such as measuring signal strength of a Bluetooth device.
- **Button controller** - Checks the state of a button and creates an event when the button is pressed.
- **Alarm controller** - Responsible for turning on/off the alarm.

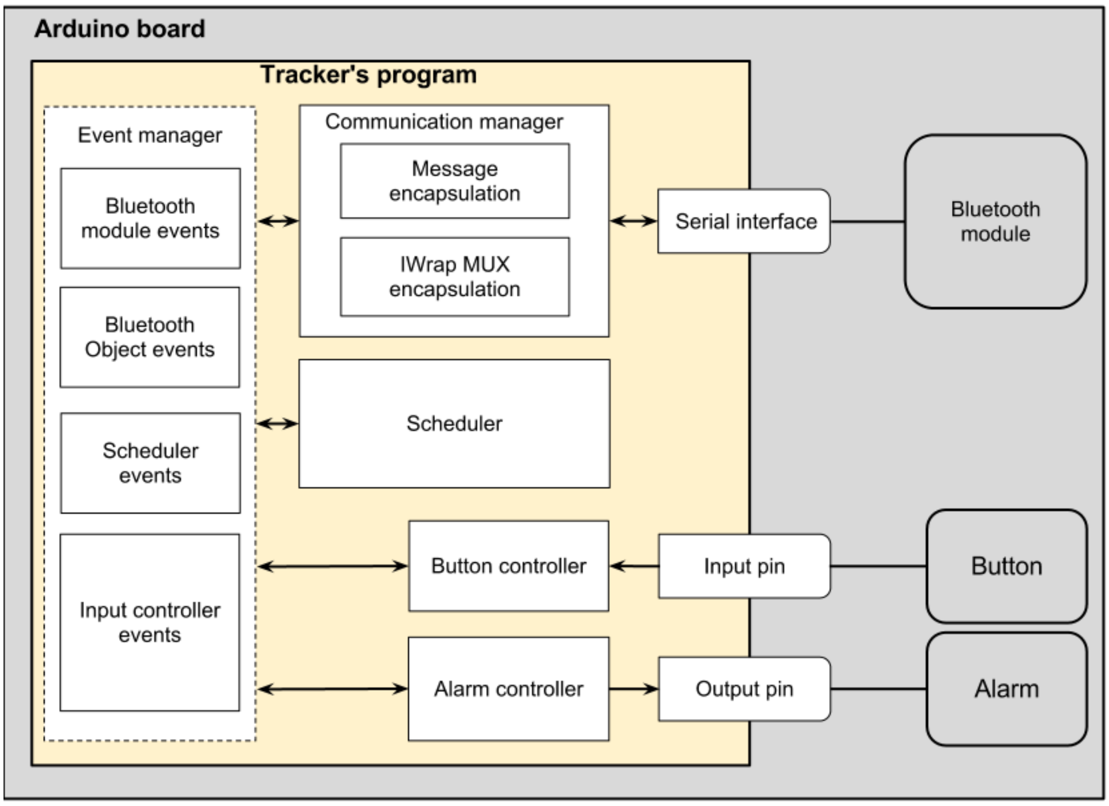

Figure 4.4: Tracker's architecture.

Firstly, the communication manager is a layer between the serial interface and rest of the ObjectTracker program. The serial interface is a bridge between the Bluetooth module and the Arduino microcontroller. The communication manager's task is to pack the data into frames before it is sent to the serial interface. There can be two levels of encapsulation. The data is always encapsulated into the first level frames, it is required by the IWrap MUX mode. The first level frame contains the destination's identification number (ID) and the data to be sent. The ID can be from 0 to 8 for connected *Objects* or it can be 255, which is the ID of the Bluetooth module. The second level of encapsulation is used when the data needs to be sent to the *Object*. This encapsulation is needed to be compatible with the Amarino communication protocol. The Amarino MeetAndroid library for the Arduino already has the Amarino protocol implemented, although, this library was not used, because it supports only one device. The MeetAndroid library was only used as a base for implementing the communication manager. However, to meet the requirements it was extended by adding support for multiple Bluetooth devices. The communication manager is also responsible for unpacking incoming frames and forwarding the content to the event manager. The incoming data is encapsulated in the same fashion as the outgoing data except there is no second level encapsulation. Based on the ID, incoming messages will go to the event manager's Bluetooth module event handler or Bluetooth Object event handler.

Secondly, the event manager handles Bluetooth module events, Bluetooth Object events, Scheduler events and Input controller events. The event manager processes incoming events and makes decisions, for example, if Alarm command comes, it raises alarm in the alarm controller. Furthermore, there are three events which can be received from the Bluetooth module: RING, NO CARRIER and RSSI. The RING event is sent by the Bluetooth module when a new Bluetooth device connects. The *Tracker* stores this information, so it does not have to query the Bluetooth module for all the connected devices. The NO CARRIER event is sent when the connection to one of the *Objects* closes. This event includes the end device ID. If the *Tracker* receives NO CARRIER event, it marks the corresponding connection as inactive which means that global outgoing events such as Alarm will not be forwarded to this connection any more. The RSSI event is a response to the Bluetooth signal strength RSSI query. The event contains device ID and its Bluetooth signal strength. This information is used for raising an alarm when the *Object*'s Bluetooth signal strength drops below -11 or -32.

More information about the Bluetooth signal strength and how these values were picked can be found in the Chapter 5. Another responsibility of the event manager is to handle events (commands) coming from the *Objects*. All these commands are described in the Table 4.7. For example, if the *Tracker* receives Disconnect command from an *Object*, it marks the corresponding connection as disabled and ignores future NO CARRIER events on that connection which would normally raise an alarm. Another task of the event manager is to forward scheduler commands to the communication manager. The last purpose of the event manager is to respond to the button controller events by sending Alarm to all connected devices that support the Alarm command.

Thirdly, the scheduler has two tasks. First, it needs to send RSSI queries to the Bluetooth module in every 5 seconds. Second, the Status command is sent to each *Object* in every minute, it was needed for the battery testing, which was one of the experiments discussed in Chapter 5.

Fourthly, the button controller is responsible for checking the alarm button state and sending the button pressed event to the event manager.

Finally, the alarm controller manages the on/off state of the alarm, also it controls how long the alarm is active.

4.2.3 Object

As a prototype *Object* we will be using an Android smartphone, although any Bluetooth device that can open a connection could be used as an *Object*. The reason we are using the Android phone is that it allows us creating a program which fully supports the communication protocol that the *Tracker* is using. A prototype application called PhoneWatch was developed in this thesis for the Android Phone 2 (Table 5.1) and was verified to be working on all three test phones (Table 5.1).

The PhoneWatch application needs the Amarino Android application [26] to be installed in order to work correctly. The Amarino application is required because it contains an Android Service called AmarinoService that is a layer between the Bluetooth module and other Android applications. The AmarinoService is responsible for following the Amarino communication protocol and managing Bluetooth connections. When the AmarinoService receives the data from one of the connected Bluetooth devices, it broadcasts the message by using Android Intents. Any Android application can receive Intents by registering a BroadcastReceiver. Therefore, the PhoneWatch application has the BroadcastReceiver registered for Amarino Intents.

The incoming data is received by using Intents in the PhoneWatch application. The outgoing data moves a bit differently. Instead of Intents, the PhoneWatch uses the Amarino class from the Amarino library for Android. The Amarino class can connect, disconnect and send data to a Bluetooth connected device. Internally, the Amarino class still uses Intents to forward the data to AmarinoService.

In addition to the interaction with the AmarinoService, the PhoneWatch has a GUI, shown on Figure 4.5. The GUI has several buttons which are binded to the responding commands of the *Tracker*'s communication protocol:

- **Connect** - Connects to the *Tracker*.
- **Disconnect** - Disconnects from the *Tracker*.
- **Alarm** - Sends an Alarm command to the *Tracker*.
- **Alarm short range** - Sends an Alarm range command with the corresponding parameter to the *Tracker*.
- **Alarm long range** - Sends an Alarm range command with the corresponding parameter to the *Tracker*.
- **Alarm range off** - Sends an Alarm range command with the corresponding parameter to the *Tracker*.
- **Enable alarm feature** - Sends an Enable feature command to the *Tracker* with parameter A (*Tracker* is allowed to send alarm commands).
- **Disable alarm feature** - Sends a Disable feature command to the *Tracker* with parameter A (*Tracker* will not send alarm commands any more).

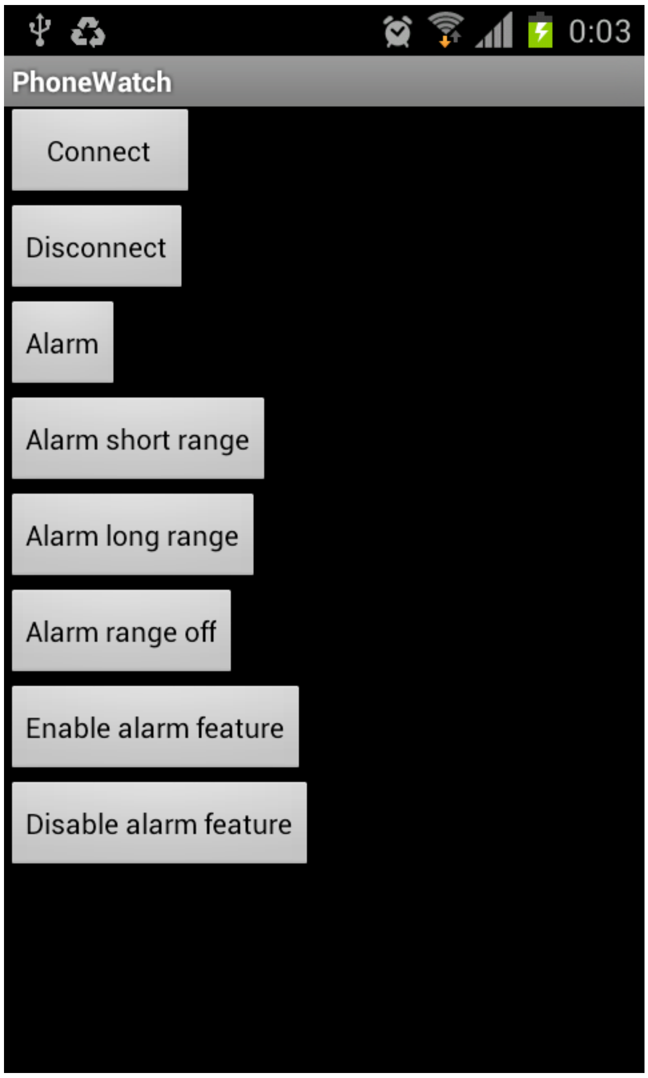

Figure 4.5: Android PhoneWatch application GUI.

5

Object Localization Framework Performance

Several experiments were conducted in this thesis to study the performance, usability and scalability of the framework. The first three experiments analysed the connectivity of the *Tracker* device. The usability was measured by the power consumption experiment. Finally, multiple devices were connected to test the scalability.

Experiments were carried out between one *Tracker* and one *Object*, except the last experiment where more than one *Object* was involved. As an *Object*, Android phone 1 (Table 5.1) was used during the tests. The *Tracker*'s technical parameters were provided in the previous chapter. To monitor events and send commands to the IWrap, a laptop computer with Bluetooth module was connected to the *Tracker* as a second *Object*. Bluetooth signal strength was measured with the IWrap RSSI command which returns the Bluetooth signal strength of the connected device. According to the IWrap documentation [15], the signal strength can be within the range of 20 (Good) to -120 (Poor).

5.1 Outdoor Connectivity

The aim of the first experiment was to test the limits of the connectivity range. The experiment was conducted in outdoor environment on an open field. The *Object* was moved away from the *Tracker* and when the connection link between the *Tracker* and the *Object* broke, the distance to the *Tracker* was measured. Even though the IWrap

	Android phone 1	**Android phone 2**	**Android phone 3**
Model	LG GT540 Optimus [27]	Samsung I9100 Galaxy S II [28]	Samung Galaxy Mini S5570 [29]
OS	Android 2.1	Android 4.0.3	Android 2.3
CPU	600 MHz	Dual-core 1.2 GHz Cortex-A9	600 MHz ARMv6
RAM	156 MB	1 GB	384 MB
Bluetooth	2.1 with A2DP	3.0+HS	2.1 with A2DP

Table 5.1: Technical information about Android phones which were used in the experiments.

sends NO CARRIER event when the connection breaks, the alarm command was sent to the *Tracker* every few seconds while the *Object* was moved away to ensure that the communication was working correctly. Five tests were performed with the following results in the Table 5.2.

Test 1	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
73 m	98 m	81 m	80 m	99 m	**86 m**

Table 5.2: Outdoor connectivity range test results, shows at which distance the connection breaks.

Although, the Bluegiga WT11-A documentation states that the maximum connectivity range is 200 meters, the average distance measured in this experiment was 86 meters. This is still good enough for the tracking framework which aims for tracking personal items.

Another part of this experiment was to check at which distance the *Object* is able to open a connection to the *Tracker*. The *Object* was moved to the farthest point (according to the results in previous tests) where the connection link broke and then it was moved back towards the *Tracker* while in every two meters the connection attempt was made. If the connection was successful, distance to the *Tracker* was measured. Again, five tests were performed with the following results in the Table 5.3. The results of this experiment show that the *Object* cannot be farther than approximately 30 meters from the *Tracker* in order to open the connection. It is a noticeable difference compared to the maximum connectivity range where the *Object* is able to communicate with the

Test 1	Test 2	Test 3	Test 4	Test 5	Average
28 m	33 m	31 m	31 m	29 m	**30 m**

Table 5.3: Outdoor connectivity range test results, shows at which distance the connection can be opened.

Tracker. Still, this is good enough because the user is expected to wear the *Tracker* device. When the user turns on the *Object* and connects to the *Tracker*, then the *Tracker* and the *Object* should be very close to each other.

5.2 Outdoor Bluetooth Signal Strength

The purpose of the next two experiments was to check whether it is possible to use the Bluetooth signal strength for proximity calculations in different conditions. If the signal strength could be used to measure the distance between the *Tracker* and the *Object* device, then it would allow setting custom range for different *Objects* at which distance the alarm would be raised.

Five tests were performed at distances up to 30 meters (20 tests in total). The results are available in the Table 5.4 and illustrated in the Figure 5.1. As we can

Distance	Test 1	Test 2	Test 3	Test 4	Test 5	Average
0 m	0	0	0	0	0	**0**
10 m	-17	-11	-16	-11	-20	**-15**
20 m	-26	-26	-31	-32	-31	**-29**
30 m	-31	-32	-32	-32	-32	**-32**

Table 5.4: Outdoor Bluetooth signal strength test results. Object was in direct line of sight. Results are in IWrap RSSI units.

see, the Bluetooth signal strength at different distances was not a constant value, for example it varied from -11 to -20 when the *Object* was 10 meters away from the *Tracker*. However, if we have a signal strength value then we can say that this value was not measured farther than X meters. For instance, if we measured signal strength -25 then we can say that the *Object* is not likely farther than 20 meters, because the average signal strength at 20 meters is -29.

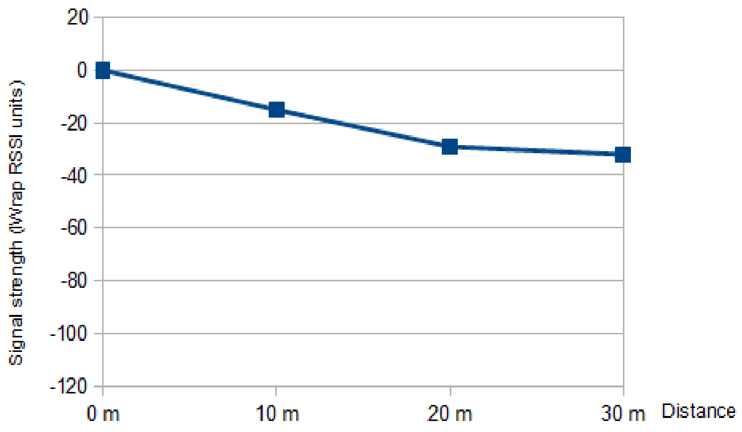

Figure 5.1: Outdoor average Bluetooth signal strength.

5.3 Indoor Bluetooth Signal Strength

The second Bluetooth signal strength experiment was carried out in the indoor environment. The purpose of this experiment was to check if the indoor environment affects the signal strength and also use the results for calibrating the *Tracker*. Tests were performed at distances up to 10 meters, also with and without the 10 cm thick concrete wall in between the *Tracker* and the *Object*. In total, 20 tests were performed in the experiment. The results of these tests are in the following Tables 5.5, 5.6 and illustrated in the Figure 5.2: At 10 meters the average indoor signal strength was

Distance	**Test 1**	**Test 2**	**Test 3**	**Test 4**	**Test 5**	**Average**
0 m	0	0	0	0	0	**0**
2.5 m	-2	-2	-3	-3	-2	**-2**
5 m	-11	-10	-13	-11	-11	**-11**
10 m	-22	-22	-20	-20	-22	**-21**

Table 5.5: Indoor Bluetooth signal strength test results. Object was in direct line of sight. Results are in IWrap RSSI units.

poorer compared to the average signal strength measured in the outdoor environment at the same distance. However, the signal strength was more stable in the indoor environment. If we compare the indoor direct line of sight results with the results where there was a wall in between the *Object* and the *Tracker*, then there is even greater

Distance	Test 1	Test 2	Test 3	Test 4	Test 5	Average
2.5 m	-19	-18	-19	-20	-19	**-19**
5 m	-23	-23	-24	-22	-23	**-23**
10 m	-23	-25	-24	-24	-24	**-24**

Table 5.6: Indoor Bluetooth signal strength test results. Object was behind the wall. Results are in IWrap RSSI units.

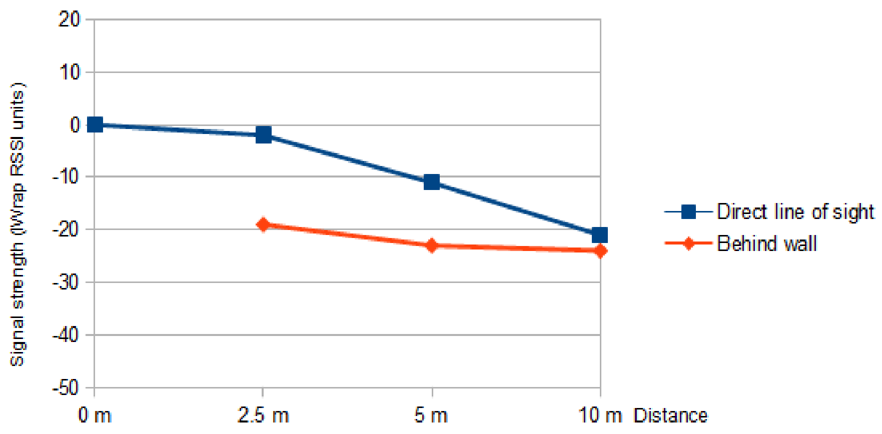

Figure 5.2: Indoor average Bluetooth signal strength.

difference in the signal strength.

In conclusion, it seems reasonable to use the indoor signal strength results for calibrating the *Tracker*'s proximity measurement. For the "short range" alarm distance we are using the average signal strength of the indoor tests at 2.5 meters. This gives us the average signal strength -11 (calculated by combining the data in the Table 5.5 and Table 5.6). If the signal strength of a "short range" tracked *Object* is worse than -11, then the alarm will be raised. However, for the "long range" alarm distance we use the worst signal strength measured in the experiments. This is -32, an average signal strength measured at 30 meters in the outdoor environment (Table 5.4).

5.4 Power Consumption

Because the *Tracker* is a mobile object, it needs a power supply which has to store the energy. This sets some limits for the usage of such a system. To check the usability of the *Tracker* device we did a power consumption experiment. The purpose of this experiment was to check whether it is feasible to use regular alkaline batteries that can be found in a commodity store.

We used three AA type 1.5 V (LR6) batteries. The *Tracker* was scheduled to continuously send out a Status command once in a minute. When the phone received this command it stored the current time stamp. At first, when the experiment started, the start time was noted. After the *Tracker*'s batteries were drained out, the last Status time stamp was retrieved from the phone. The result was 39 hours and 20 minutes.

From the experiment, it can be observed that the system has reasonable performance levels when using three AA type alkaline batteries. However, rechargeable batteries can be also used. Moreover, since this is a prototype of the *Tracker* device, it is not optimized for the power consumption. According to EngBlaze [30] it is possible to increase the battery life by using the Arduino sleep mode. Even more, the IWrap offers several ways to reduce the power consumption by configuring the Bluetooth module such as using the SET BT PAGEMODE to specify how often the module checks for incoming connections.

5.5 Multiple Objects

To see if the tracking framework is scalable, we made an experiment where multiple *Objects* were connected to the *Tracker*. The IWrap manual states that it supports up to 4 simultaneous Bluetooth connections [31], but when we tried to connect all three mobile phones (Table 5.1), the Bluegiga WT11-A module did not accept more than 2 simultaneous Bluetooth connections. Although, each phone was able to establish the connection when we tried to connect them one at the time.

Despite this issue, we connected Android phone 1 and Android phone 2 to the *Tracker*. The *Tracker* was able to send and receive the Alarm commands without any problems. Also, the tracking by signal strength was tested by setting both phones as ”short range” *Objects*. The *Tracker* was able to track both devices and when one of them left from the short range, the alarm was raised.

6 Conclusions and Future Work

In this thesis the tracking framework for object localization was developed and studied. The aim of this work was to see if it is possible to create a system which help us track or find misplaced personal belongings. The existing object localization systems are rather expensive, immobile or not usable in both indoor and outdoor environments. The proposed solution is based on a relatively low cost Arduino BT microcontroller and widely used Bluetooth technology. The microcontroller tracks other Bluetooth devices. As a prototype object that needs to be tracked Android smartphone was used. For both prototypes, Arduino BT microcontroller and Android phone, a software was developed which supports various features, for instance, alarm is raised when object is left behind or misplaced phone can be located by sending alarm message to it.

A series of experiments were conducted with the prototypes such as connectivity range and battery consumption tests to study and verify the feasibility of the implemented system. The experiments confirmed that the Bluetooth signal strength could be used as proximity indication. The power consumption test verified that the system can run on batteries.

Moreover, experiment results were used for calibrating the software for the tracking device (Arduino BT microcontroller). The Bluetooth signal strength was used as proximity indicator, although as a future research it could be improved by calibrating each object individually when it is connected to the system. Moreover, the power consumption of the tracking device could be improved by using special power saving modes for Arduino BT microcontroller and Bluetooth module.

Furthermore, the prototype application for Android phone has very primitive graphical user interface which could be improved, it was designed to be functional rather than putting emphasis on the user interface. In addition, the application depends on the Amarino application, it is possible to extract the required parts and put them into a library which would significantly improve the usability of the application.

Arduinol põhinev jälgimise raamistik objekti asukoha määramiseks

Bakalaureusetöö (6 EAP)
Steve Mägi
Resümee

Paljudel inimestel on kodust lahkudes kaasas isiklikud asjad, nagu näiteks rahakott, koduukse võtmed või mobiiltelefon. Kui vahel mõni neist koju ununeb, siis see tekitab ebameeldivusi. Selle vältimiseks oleks vaja esemetel paremini silma peal hoida.

Käesolev bakalaureusetöö pakub välja lahenduse, mis aitab objekte jälgida ja vajadusel nende asukohta kindlaks määrata. Süsteem põhineb Arduino mikrokontrolleril (*Jälgija*) ja Bluetooth'i tehnoloogial. Jälgitavaks objektiks (*Objekt*) võib olla näiteks võtmehoidja, milles on Bluetooth'i moodul. *Objektiks* sobib ka Bluetooth'iga varustatud mobiiltelefon. *Jälgija* kinnitatakse kasutajale vööle, seejärel ühendab kasutaja jälgitavad objektid üle Bluetooth'i ühenduse *Jälgija* külge. Kui mõni *Objektidest Jälgijast* liiga kaugele liigub, siis käivitub alarm. Samas on võimalik *Jälgijast* saata signaal *Objektile* ja helina järgi üles leida näiteks diivani padja alla ununenud mobiiltelefon.

Lahendusena valmisid prototüübid *Jälgijast* ja *Objektist.* Jälgijana kasutati Arduino BT (Bluetooth) mikrokontrollerit ning *Objektina* Android operatsioonisüsteemil põhinevat nutitelefoni. Töös antakse ülevaade kasutatud tehnoloogiatest. Seejärel kirjeldatakse erinevaid objekti asukoha määramise ja jälgimise tehnoloogiaid ning tuuakse välja nende eelised ja puudused. Töös vaadeldakse ka valminud lahenduse nii riistvaralist kui ka tarkvaralist arhitektuuri.

Valminud prototüüpidega tehtud eksperimendid näitasid, et Bluetooth'i signaalitugevust on võimalik kasutada seadme kauguse hindamiseks. Elektrienergia tarbimise katses veenduti, et *Jälgija* prototüüp on võimeline ka tavaliste Alkaline patareidega edukalt töötama.

Bibliography

[1] Bluetooth SIG, Bluetooth, [Online; accessed 06.05.2012].
URL http://www.bluetooth.com/Pages/Bluetooth-Home.aspx 5, 9

[2] Google Inc, Android, [Online; accessed 06.05.2012].
URL http://www.android.com/ 5

[3] Arduino, Arduino, [Online; accessed 06.05.2012].
URL http://www.arduino.cc/ 5, 10

[4] IEEE, IEEE Standard 802.15.1-2005, IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements. - Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs) (08 2005). 9

[5] IEEE, IEEE Standards 802.11x, IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. 9

[6] Wi-Fi Alliance, Wi-fi, [Online; accessed 06.05.2012].
URL http://www.wi-fi.org/ 9

[7] Infrared Data Association, Infrared Data Association, [Online; accessed 06.05.2012].
URL http://www.irda.org/ 10

[8] ZigBee Alliance, Zigbee technology, [Online; accessed 06.05.2012].
URL http://www.zigbee.org/ 10

[9] Arduino, Arduino BT (Bluetooth), [Online; accessed 06.05.2012].
URL http://arduino.cc/en/Main/ArduinoBoardBluetooth 10, 28

[10] Arduino, Arduino LiliPad, [Online; accessed 06.05.2012].
URL http://arduino.cc/en/Main/ArduinoBoardLilyPad 10

[11] Arduino wiki, Arduino analog input, [Online; accessed 06.05.2012].
URL http://arduino.cc/playground/CourseWare/AnalogInput 11

[12] A. Rubin, Number of android phones activated, [Online; accessed 06.05.2012].
URL https://plus.google.com/u/0/112599748506977857728/posts/Btey7rJBaLF 11

[13] Google Inc, Android architecture, [Online; accessed 06.05.2012].
URL http://developer.android.com/guide/basics/what-is-android.html 11, 12

[14] B. Kaufmann, Amarino toolkit, [Online; accessed 06.05.2012].
URL http://www.amarino-toolkit.net/ 13

[15] Bluegiga Technologies, IWrap 2.2.0 User Guide, Version 3.3, Bluegiga Technologies (09 2007). 13, 30, 39

[16] J. Hightower, G. Borriello, Location systems for ubiquitous computing, Computer 34 (8) (2001) 57–66. 14

[17] R. Bajaj, S. Ranaweera, D. Agrawal, Gps: Location-tracking technology, Computer 35 (4) (2002) 92–94. 14

[18] P. Bahl, V. Padmanabhan, Radar: An in-building rf-based user location and tracking system, in: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, Vol. 2, Ieee, 2000, pp. 775–784. 14

[19] G. Borriello, W. Brunette, M. Hall, C. Hartung, C. Tangney, Reminding about tagged objects using passive rfids, in: N. Davies, E. Mynatt, I. Siio (Eds.), UbiComp 2004: Ubiquitous Computing, Vol. 3205 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2004, pp. 36–53, 10.1007/978-3-540-30119-6-3.
URL http://dx.doi.org/10.1007/978-3-540-30119-6_3 15

[20] C. Roberts, Radio frequency identification (rfid), Computers &amp; Security 25 (1) (2006) 18 – 26. doi:10.1016/j.cose.2005.12.003.
URL http://www.sciencedirect.com/science/article/pii/S016740480500204X

[21] J. Shneidman, P. Pietzuch, J. Ledlie, M. Roussopoulos, M. Seltzer, M. Welsh, Hourglass: An infrastructure for connecting sensor networks and applications, Tech. rep., Harvard Technical Report TR-21-04 (2004). 15

[22] Wikipedia, Ascii, [Online; accessed 06.05.2012].
URL http://en.wikipedia.org/w/index.php?title=ASCII&oldid=489175974 26

[23] Bluegiga Technologies, WT11 Data Sheet, Version 3.3, Bluegiga Technologies (10 2011). 29

[24] Tinkerkit, Tinkerkit's LED module, [Online; accessed 06.05.2012].
URL http://tinkerkit.com/en/Modules/T010117 30

[25] Tinkerkit, Tinkerkit's Push Button module, [Online; accessed 06.05.2012].
URL http://tinkerkit.com/en/Modules/T000180 32

[26] B. Kaufmann, Amarino Android application, [Online; accessed 06.05.2012].
URL http://www.amarino-toolkit.net/index.php/download.html 35

[27] GSM Arena, LG GT540 Optimus, [Online; accessed 06.05.2012].
URL http://www.gsmarena.com/lg_gt540_optimus-3081.php 40

[28] GSM Arena, Samsung I9100 Galaxy S II, [Online; accessed 06.05.2012].
URL http://www.gsmarena.com/samsung_i9100_galaxy_s_ii-3621.php 40

[29] GSM Arena, Samsung Galaxy Mini S5570, [Online; accessed 06.05.2012].
URL http://www.gsmarena.com/samsung_galaxy_mini_s5570-3725.php 40

[30] EngBlaze, Arduino sleep mode, [Online; accessed 06.05.2012].
URL http://www.engblaze.com/hush-little-microprocessor-avr-and-arduino-sleep-mode-basics/ 44

[31] Bluegiga Technologies, IWrap 2.2.0 User Guide, Version 3.3, page 73, Bluegiga Technologies (09 2007). 44

Appendix A

CD Content

The accompanied CD-ROM has the following content:

- Source code for the ObjectTracker program for Arduino BT microcontroller
- Source code of the PhoneWatch application for Android smartphone
- PhoneWatch application for Android smartphone
- Amarino application for Android smartphone