Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Communication Theory Group
Prof. Dr. H. Bölcskei

Group Project

Spring Semester 2016

Raphaela Eberli
Nicolas Früh
Felix Graule

Induc able

A Table for Position-Independent Wireless Charging

Supervisor: Michael Lerjen

June 2016

Abstract

Today, there is a very popular wireless charging standard called Qi, suitable especially for smartphones. Unfortunately, the user is required to align his device with the charging coil. We want to remove this limitation and came up with the idea of dynamically moving the charging coils to the devices. For this project, we built a table which detects when a smartphone was put onto its surface and moves a charging coil to that position. Although our prototype is working in principle, there is still room for improvement. In our testing, the system as a whole only managed to charge a device a few times.

Acknowledgements

This project would not have been possible without the kind support and help we received of many people. We would like to extend our sincere thanks to all of them.
We would like to express our special gratitude towards *Prof. Dr. H. Bölcskei* for supporting our idea and giving us the opportunity to carry out this project at his institute.
We are highly indepted to *Mr. M. Lerjen* for his guidance and constant supervision. We could discuss our progress with him on a weekly basis and he always gave us very helpful feedback. Furthermore, he generously granted us access to many ressources of the institute.
Furthermore, we would also like to thank the workshop at D-ITET for printing our 3D models and helping us with some mechanical questions.
Moreover, we want to thank *Mr. K. Keller* for giving us access to his private 3D printer if we needed a 3D model very quickly.
Finally, we want to thank *Mr. A. Bódis-Szomorú*, working at the Computer Vision Lab at D-ITET, who assessed our initial thoughts on the image recognition algorithm and gave us valuable input on that topic.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1
- 1.1 Context . . . 1
- 1.2 Aim . . . 1
- 1.3 Existing Technology . . . 2
  - 1.3.1 Wireless Charging Standard Qi . . . 2
  - 1.3.2 Eddystone Bluetooth Low Energy Beacon Standard . 2
- 1.4 Glossary . . . 3

2 Our Implementation 4
- 2.1 The Setup . . . 5
- 2.2 Key Requirements for our Implementation . . . 6
- 2.3 Step-by-Step Walk-Through of the App . . . 7
  - 2.3.1 System Requirements . . . 7
  - 2.3.2 Wait until Device Ready for Charging . . . 8
  - 2.3.3 Take the Photo . . . 10
  - 2.3.4 Transmit the Photo to the Server . . . 10
- 2.4 Server and Communication . . . 12
  - 2.4.1 Access from the World Wide Web . . . 12
  - 2.4.2 Local WiFi Network . . . 13
  - 2.4.3 Control the System . . . 14
  - 2.4.4 The Communication Library . . . 15
  - 2.4.5 Communication with the Cable Rollers . . . 19
- 2.5 Localisation of the Smartphone . . . 19
  - 2.5.1 Problem and Main Idea . . . 19
  - 2.5.2 Overview of Image Recognition Process . . . 20
  - 2.5.3 Software Implementation of the Image Recognition . 22
  - 2.5.4 Preparation . . . 23
  - 2.5.5 Seedpoint Search . . . 25

2.5.6 Identification of QR Codes 26
2.5.7 Rotation Analysis 27
2.5.8 Scaling Analysis 28
2.5.9 Extraction of the QR Code 29
2.5.10 QR Code Analysis with zBar 31
2.5.11 Transform Coordinates to the Table 31
2.6 Drones 33
2.6.1 Main Hardware Components of the Drone 34
2.6.2 Drone Start-Up and Control Software 37
2.6.3 Drone Functionality 37
2.7 Cable Management and Power Supply 42
2.7.1 Cable Roller Hardware 42
2.7.2 Cable Roller Software 43
2.7.3 Power Supply 46

3 Results 48
3.1 Measurements of Qi Charging Kit 48
3.1.1 Vertical Offset 49
3.1.2 Horizontal Offset 49
3.2 Measurements of System Performance 51
3.2.1 Precision of Drone 51
3.2.2 Reliability of System 51

4 Reflection and Outlook 54
4.1 Conclusion 54
4.2 Known Issues and Bugs 54
4.2.1 App 54
4.2.2 Image Recognition 55
4.2.3 Drone and Power Supply 56
4.3 Team Work and Reflection 56
4.3.1 Difficulties Encountered 57
4.3.2 What was Handled Well 58
4.4 Possible Future Enhancements 58
4.4.1 Improvements Based on our Implementation 58
4.4.2 Different Approaches 60

A Measurements of Qi Charging Kit 62

Chapter 1

Introduction

1.1 Context

Sitting in the foyer of the ITET-building you may notice the following: getting access to a power outlet not too far away from your table is not an easy task. What might come to your mind in the next second is, that it is indeed quiet strange that students in electrical engineering have to fight for a place to charge their phones and laptops and have to be constantly cautious not to fall over a plugged-in power cable. We wanted to change this and thus decided to start this group project.

We thought about different possibilities to ease access to a power supply and came to the conclusion, that wireless power transmission would be both very convenient and modern. A brief internet research revealed that there does not yet exist a hassle free wireless charging solution on the market. All available technology requires the user to position his device more or less precisely over a charging spot. This is inconvenient. We imagine a charging solution, especially for smartphones, that automatically charges a device wherever it is positioned on a table. The user should not have to think about it, his device should be charged without him even noticing it.

1.2 Aim

We thought more deeply about the problem and identified the following requirements that such a *charging table* solution needs to satisfy.

Localisation of the Device The table should be aware of chargeable devices that are laid onto it and be able to determine their precise posi-

tion. This is actually not a trivial problem since a smartphone needs to be distinguished from any other object, e.g. a book, that might be laid onto the table as well. There does not exist a applicable solution to this problem on the market yet.

Charging the Device The actual charging process must be wireless, i.e. using inductive coupling, and as efficient as possible. The system should be able to provide enough power to charge even bigger smartphones.

Simplicity The whole system should be very simple, require as little service as possible and the user should not have to worry about it at all. It should work in the background whenever possible.

Such a charging table could be used in many public places such as restaurants, libraries or student learning rooms and would mitigate the problem of running out of battery. It could for example replace an existing table.

1.3 Existing Technology

1.3.1 Wireless Charging Standard Qi

The most common wireless charging standard today is the Qi standard by the Wireless Power Consortium [1]. This standard uses inductive coupling and resonance between two coils and is able to transmit up to 15 watts over a distance of 45 mm. It operates at frequencies between 110 and 205 kHz. Almost every smartphone we found on the market that features wireless charging supports the Qi standard. Furthermore, there are projects like the one by Starbucks in the UK and US where tables are equipped with Qi charging cradles [2]. That is why we decided to use this standard for our project. Since Qi does not allow us to charge high power devices such as laptops, we only considered smartphones for our project.

1.3.2 Eddystone Bluetooth Low Energy Beacon Standard

Bluetooth Low Energy is a feature of the current Bluetooth 4.0 standard. It allows devices to operate with minimal energy consumption. A Bluetooth device can serve as a beacon by periodically broadcasting a signal. This signal can be received by e.g. a smartphone. Using received signal strength indication (RSSI), the smartphone can determine the distance to the beacon, as well as an ID number. There are two major standards, *iBeacon* by Apple and *Eddystone* by Google. We use the latter one in our project.

1.4 Glossary

Throughout this report, we are going to use the following terms.

Server This is our central computation and coordination unit that connects to both the internet and the drones. By *server* we mean both the hardware and the software.

Drones These are the little cars that move the Qi charging coils to the smartphones.

Cable Roller This is the system responsible to avoid cable tangle. It consists of an Arduino, a stepper motor and a cable roller.

Chapter 2

Our Implementation

Figure 2.1: Diagram Showing the Whole System

We had to solve two main problems during our problems: First, we needed to find a way to charge many devices at arbitrary positions. We decided to use drones on wheels to do this since they can move freely, contrary to e.g. Qi chargers mounted on rails. Second, the position of the smartphone on the table has to be determined. We use the smartphone camera together with image analysis to do this. In the next chapters, we will describe each component of our solution in detail. As an overview, refer to Figure 2.1 and Figure 2.2.

Figure 2.2: This diagram shows an overview of the software components we use. Since we use three pieces of hardware (the server, the smartphone and the drone), we grouped everything accordingly.

The whole process starts with the user laying his device (i.e. smartphone) onto our table. To take a photo and send it to our server for analysis, we need an app running on the device that should be charged. It recognises when it is laid down in the proximity of our table and automatically takes a photo. This photo is then transmitted to the server (which is built into the table) where our image recognition algorithms analyse the image and determine the position of the camera from it. This position information is afterwards sent to a drone that is not busy yet. This drone then moves to the target. Simultaneously, the cable roller rolls the correct amount of cable to avoid cable tangle. Once the target position is reached, fine search starts until the Qi wireless charging kit mounted on top of the drone can start charging the clients device. Our system may be equipped with arbitrarily many drones if needed.

2.1 The Setup

The core part of our system is the table. Our table, as depicted in Figure 2.3, consists of a glass table top. Under the glass top, there is a second wooden plate mounted in a distance of seven centimetres. In-between the glass

and the wood, small drones on two wheels will drive to and charge the devices. On the wooden plate, a matrix of QR-codes is printed, as described in section 2.5.2. To locate a device on the table, we use the integrated smartphone camera. Thanks to the glass surface, it can take a picture of the QR map.

Figure 2.3: The top part of the table. In a final product, it would be closed on the edges to avoid interference by the user. The cable rollers and the server will be mounted on the edges of the table, cables can be routed along the bezel.

2.2 Key Requirements for our Implementation

App The app may not distract the user at all. It must be installed once and afterwards run in the background. Energy and processing power consumption should be at a minimum. The app must reliably detect when it is ready to be charged, then take a photo with sufficient resolution and send it to the server via internet. Additional information may be sent as well, such as a device identifier, charging coil offset with respect to the camera or the current battery level. This will allow the server to calculate the exact position of the coil centre and prioritise devices.

Server The server is our core control unit, implemented using a Raspberry Pi 3. It must be able to receive and analyse images from the client devices to send drones to the appropriate positions. It must be aware of all drones available and what they are currently doing. Furthermore, it controls the cable rollers. No direct interaction between the user and

the server should be required, it must start working automatically if the power cable is plugged in. Security is an important aspect as well since the server is accessible from the world wide web. Appropriate measures should be taken to make the system secure.

Drones The main task of the drones is to position the attached Qi charger under the users phone and start charging it. Inductive charging calls for highly accurate alignment (see Appendix A, about 8mm is the maximum deviation from optimal position that is possible) of the emitter and the receiver coil. For this reason, the most fundamental requirement for the drones is high driving precision of a few millimetres. Also, every drone should be positioned independently according to the orders of the server. Finally, power consumption should be minimised to increase the efficiency of the whole charging system.

Table The table must be robust. The whole table top (including the wooden and the glass plate) should be as thin as possible, otherwise it will be uncomfortable to sit since the legs might not fit between the chair and the table anymore.

Localisation In order to charge the users phone, it is vital to know its position as precisely as possible. As well, the drone's position has to be determined to recalibrate its position. The chosen localisation method should therefore focus on precision. Furthermore it should not require any additional hardware; especially no externally mounted devices, since this would violate the idea of a movable and integral table. Eventhough, we are free to add e.g. a camera to the drones. The precision of our approach using a QR map in combination with image recognition should be at ± 2 mm. The same code shoud be usable to locate the smartphone and the drone respectively.

2.3 Step-by-Step Walk-Through of the App

2.3.1 System Requirements

The user has to install our app on the smartphone he wants to use with our table. The smartphone must have at least a back-facing camera and a Qi compatible inductive charging kit installed or a built-in Qi coil. Additionally, our software requires Android 4.4 or higher. As of April 2016, over 70% of the devices active on the Google Play Store support Android 4.4 or higher (Source: Android Studio).

Figure 2.4: Process diagram of the Android app

Once the device is ready to be charged, the app is required to take a photo of the QR map and send it to our server. From this photo, we can then determine the exact position and rotation angle of the device as described in section 2.5. Additionally, we encode information such as a device ID, battery level or Qi coil offset with respect to the camera into the file name.

We laid big focus on hiding the app from the user. The user should only worry about it once—at the installation—and not be disturbed by our app afterwards. In this section we explain what happens on the smartphone behind the scenes. We have thus split up the process in the following sub processes.

2.3.2 Wait until Device Ready for Charging

We now assume that the user has already installed our app on his device. As shown in Figure 2.4, the process starts with the user launching the app for the first time. He will be presented with the `Splash_Activity`. An activity on Android is "*an application component that provides a screen with which users can interact in order to do something*" [3].

This activity immediately starts our `BackgroundService` which will handle all the subsequent tasks. On Android, a service is "*an application component that can perform long-running operations in the background and does not provide a user interface*" [4]. Using a service for the actual tasks our app has to carry out allows the user to close the user-interface and use other apps, e.g. a web browser, while our program is still running. Via the `Splash_Activity`, the user is able to kill or restart the service at any time. Additionally, he can change settings such as the MAC-address of the

Bluetooth Low Energy beacon, the host name of our server, the device ID, the Qi coil offset with respect to the camera or activate a debugging mode.

As soon as the `BackgroundService` is started (actually within the `onHandleIntent`-method), it registers a listener which *listens for Eddystone beacons*. To do this, the Bluetooth Low Energy library by Estimote [5] is used. Since we are deploying an Eddystone beacon with our server and its MAC-address is stored in the app settings, we can check if a found beacon corresponds to our table. Once our beacon is detected, the app assumes the smartphone to be in the proximity of the table.

The `BackgroundService` class implements `SensorEventListener`. This allows us to get current sensor data via the `onSensorChanged` method, which is depicted in 2.5. In the first part, we analyse whether the *device lay flat* (i.e. $|x|, |y| <$ `ISFLAT_THRESHOLD` and $z > 0$, where x, y, z denote the acceleration values in the three spatial directions) for a long enough time period (`flat_age` is being counted down from `INITIAL_flat_age=5` to zero). If so, we *take the photo*.

In the second part of the code, we detect if the device was *moved* quickly (using `SHOCK_THRESHOLD`) or is *tilted* slightly. In our scenario, this would be the case if the user moved his device on the table or walked away. In both cases we need to restart the whole `BackgroundService`. The tilting detection is achieved by counting down the `layingFlatSince` variable. It is set to an initial value > 0 whenever the device lies flat. If it reaches zero (i.e. the device was aslope for too much time) or a shock was detected, the `BackgroundService` is *restarted*.

Figure 2.5: This flowchart represents the `onSensorChanged` function.

2.3.3 Take the Photo

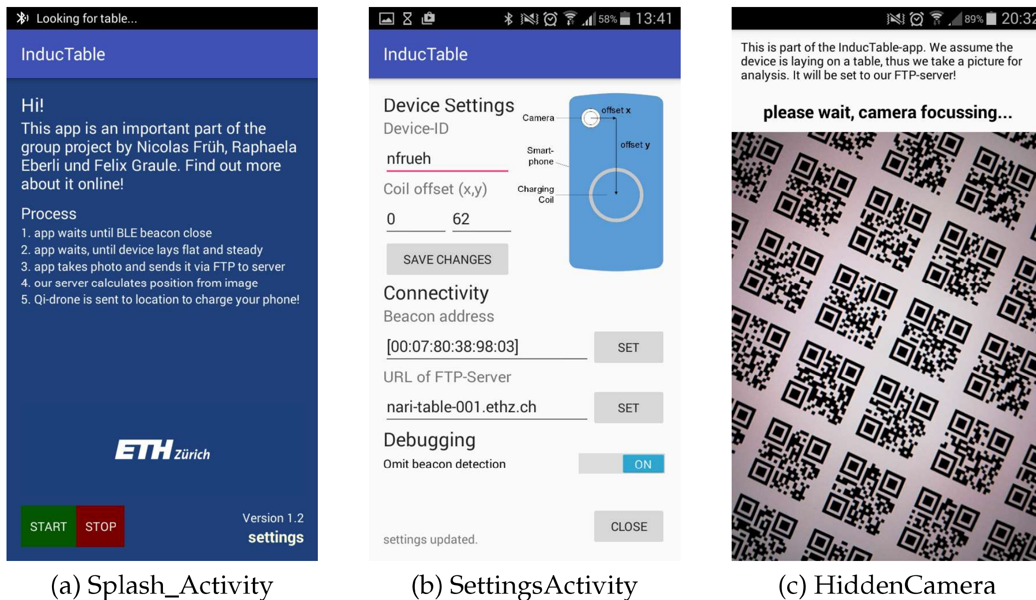
We saw that a photo must be taken as soon as the device lies flat. Using an intent, which is "*a messaging object you can use to request an action from another app component*" [6], we start an activity (meaning that we show a new user interface component on the screen) called `HiddenCamera` and register a receiver which will wake up the service once the photo has been taken successfully. We need this extra activity because the Android operating system does only allow you to call the `takePicture` on a `Camera` object if a preview display has previously been attached to it. This preview display (e.g. an instance of the `SurfaceHolder` class) can only be created within an activity context, thus we need one to take a picture. Probably, this is to make it harder for developers to take pictures without the user noticing it. This constraint forces us to show the camera preview while focusing and taking the picture.

For taking the photo, we first need to focus the camera using the `autofocus` member of the `Camera` class. Since this is done asynchronously by the operating system, we need the callback `focusAndShotCallback`, that is called once the focus is acquired. Within this callback-function, we immediately call `takePicture` to get a picture. The JPEG image data is returned and can be stored to a file on the internal storage of the device (see `onPictureTaken` method of `HiddenCamera`). We use a static file name and location which simplifies the process of finding the image again later.

In order to increase usability, we enabled this `HiddenCamera` activity to run in front of a lock screen using the `WindowManager.LayoutParams.FLAG_SHOW_WHEN_LOCKED`-flag on the window-instance. Hence, our app is able to turn the screen on and draw on-top of the lock screen without disabling it. This does not require any special Android permission and allows us to take photos even if the device is locked. However, the user will see the camera image while focusing. This would be rather hard to bypass.

2.3.4 Transmit the Photo to the Server

Using a broadcast-receiver (an Android class that allows to receive messages sent from other app components, in this case the `HiddenCamera` class), our `BackgroundService` gets notified as soon as the photo was taken successfully. Before sending the photo to our FTP-server (see section 2.4.1), we rename the file as described below.



(a) Splash_Activity (b) SettingsActivity (c) HiddenCamera

Figure 2.6: The three views of the Android App. a) shows a push notification in action telling the user that the device is looking for the BLE beacon. In view b) the user can adjust the coil offset, his ID and other settings. View c) will be displayed while the device is focusing.

```
DEVICE_ID:offsetX:offsetY:BATTERY_LEVEL:RANDOM_INT.jpg
```

`DEVICE_ID` should be unique to every device and can be a string of arbitrary length (within the limits of the file system). Can be chosen by the user via app settings.

`offsetX` and `offsetY` indicate how much the centre of the Qi coil is shifted with respect to the camera centre, see Figure 2.7.

`BATTERY_LEVEL` represents the current battery charge condition (0..100%). The size may vary between one and three digits.

`RANDOM_INT` is a random two-digit integer. We need it to avoid file name collisions on the server.

Once the file is renamed, the app transmits it over FTP using a so called `AsyncTask` which is implemented by our `NetCom`-class. Using an `AsyncTask`, which basically just creates a new thread, is required by Android for all network operations. To implement the FTP file transfer, we use the "SimpleFTP" library by jibble.org [7]. To make our task simpler,

Figure 2.7: Illustration of the coil offset respectively to the smartphone camera (back view of smartphone)

we set security aside and use unencrypted FTP. Once the image has been transmitted, a second—empty—file is sent with the same file name but `.ready` added as suffix. This file is useful to determine whether the file transfer has ended yet or not. Especially if the smartphone has a weak cellular signal, this step is important. Our server can be sure the file has completely arrived as soon as he sees the corresponding `.ready` file. Next, the server will run our image recognition algorithm.

2.4 Server and Communication

This section covers the implementation of our central server. We use a Raspberry Pi 3 with the Raspian operating system which is a version of Debian Linux optimised for our hardware.

The server's job is to coordinate the whole process. It receives the images from the smartphones, analyses them, determines the target coordinates and sends them to a drone and controls the cable rollers.

2.4.1 Access from the World Wide Web

The server is connected to the ETH-network via LAN-cable. From the internet, it is accessible at 129.132.69.253 or *nari-table-001.ethz.ch*.

Administrator Access via SSH

Thanks to the static IP we receive from the ETH-network via DHCP, we can easily connect to the server via SSH. Using putty on Windows (or any other client), a remote login to the command line is possible. Thanks to the operating systems X11-forwarding support, the user interface can be displayed on a remote computer. On windows, it is required to install an X-server such as Xming [8].

FTP-Server Implementation

To allow our app to upload images to the server, we installed the ProFTPD FTP-server software [9] on the Raspberry. The configuration can be set through the file `/etc/proftpd/proftpd.conf`. It uses the standard port 21 without an encryption layer. Currently, the user we use to upload files is the `pi` user which has admin-privileges. This could be a security issue as mentioned in subsection 4.2.1.

The root directory of the FTP-server is `/home/pi/ftp`. This directory is being monitored by a python script using the pyinotify-library [10]. Once the file upload from the app is complete (e.g. the `.ready` file has been received), a python function is invoked. It will determine the newest file in the directory and hand this path over to our image recognition algorithm. If a new file is uploaded while

2.4.2 Local WiFi Network

We set our Raspberry Pi 3 which features a state of the art WiFi chip up to act as a WiFi router. To achieve this, we installed the packages hostapd [11] and dnsmasq [12] on the system. hostapd allows us to use the built-in WiFi as an access point in the first place. dnsmasq is a combined DHCP and DNS server and can be used to assign IP-addresses to our drones. We set our server's WiFi interface up to always have the IP-address 192.168.1.1, as subnet mask we chose 255.255.255.0.

All these settings can be adjusted in the `/etc/hostapd/hostapd.conf` file. dnsmasq is configured to act as a DHCP server. It distributes IP-addresses in the range .10 to .50, valid for 12 hours each. We do not use the DNS functionalities of dnsmasq since we did not install a network address translation (NAT) software to route packages from the WiFi to the Ethernet interface. Hence, the internet cannot be reached from within our WiFi network. Not using a NAT makes sure we do not infringe the ETH Zurich Acceptable Use Policy for Telematics Resources (BOT) [13].

2.4.3 Control the System

To control the whole system, we wrote the `Server` class. After system boot, a Python script is started and an instance of `Server` is created. The constructor will immediately create an instance of `ComLib` (see subsection 2.4.4). Communication is completely handled by `ComLib`. Next, a parser (see below) is registered as callback to the `ComLib` object and two arrays are created to hold `Device` objects. `Device` is a simple class to represent either a drone or a smartphone and store information such as the current state and position. Finally, `generalWorker` is called in a seperate thread. This function will call other workers, such as the `DroneWorker` repeatedly. After this, the script starts monitoring the FTP-servers folder to detect if an image was uploaded and waits for anything to happen.

Given a drone connects to the server. The drone is assigned an ID by the `ComLib` library. `Server` receives a notification via the parser method. This method splits the message up using the built-in `split` function, as displayed in Listing 2.1.

Listing 2.1: parser method, gets `message` as argument, parts have been removed

```
# Break up message into arguments
args = message.split(':')

# Check if EVENT
if args[0] == 'EVENT':
    if args[1] == 'NEW_CLIENT':
        # Add new drone to list with appropriate state
        newDrone = Device(args[3])
        newDrone.state = "TODO:REQUEST_LOCATION"
        self.connectedDrones.append(newDrone)

# Check if TRANSMISSION via TCP
elif args[0] == 'TCP':
    if args[2] == 'TEXT':
        message = args[3]
        # Split the message even further
        message = message.split('/')
                # Get the affected drone
        drone = Device.getDrone(self, args[1])
        if message[0] == 'POSITION':
            # Update position and make drone 'READY'
        elif message[0] == 'ACK':
            # Received an acknowledgement-message
```

```
            if message[1] == '00':   # General failure
            elif message[1] == '01': # Position reached
            elif message[1] == '02': # Charging
            elif message[1] == '03': # Driving
                drone.state = 'MOVING'
                # Send command to cable roller as well
                cableControl.rollCable(curX,curY,toX,toY)
            elif message[1] == '04': # Recalibration success
    elif args[2] == 'IMAGE':
        # Process image for recalibration
```

In the case of a newly connected drone, we create a new `Device` object containing the ID. This object is then added to the array of connected drones with the state `TODO:REQUEST_LOCATION`. Later on, the `droneWorker` will walk through this array and eventually find this drone. It will request the drones position by sending `POSITION` to the drone. Once the answer is received, it will be passed once again to our parser method which will then update the position data on the server.

Now, consider the case that a users phone uploaded an image to our FTP server, meaning that it wants to be charged. As described in subsection 2.4.1, our image analysis algorithm will automatically be invoked. As soon as it gets the target coordinates, a drone will be sent to the target using `sendDrone`. `sendDrone` walks through the array of connected drones and looks for one that has state `READY`. The first drone it finds will be sent to that position.

Once a drone has been sent to a target, it will answer with the appropriate acknowledgement messages. First, it will rotate itself to head into the correct direction. As soon as this is finished, `ACK/03` is sent to the server to indicate that it will now start moving. This allows the server to send a command to the cable roller. Once the drone has arrived, `ACK/01` is sent to signal that everything went as expected. The state of the drone is then set to `TODO:REQUEST_LOCATION` again and everything starts over. This is depicted in Figure 2.8.

2.4.4 The ComLib Library to Communicate between the Server and the Drones

To establish a communication link between a drone and the server, we wrote a Python library called `ComLib`. Python scripts on both the server and the client use this library to communicate. We use our WiFi-network described in the section above to establish a connection in the first place.

Figure 2.8: Communication between server and drone after a new file has arrived until drone reached position

Our library contains only one class called `Communicator`. The constructor of this class takes three arguments: the IP address and the port as strings as well as a boolean that indicates if it should run in server or client mode. After creating an instance of `Communicator`, the programmer should register a callback function using `registerCallback`. The function he registers here will be called whenever an event happens. It must take one argument of type string. Refer to Listing 2.1 to see our implementation of that function.

Server Mode

Let's talk about the server mode first: Once an instance of `Communicator` is created, it will automatically try to open a TCP-socket using the built-in socket-library. If this succeeds, it will listen to incoming connections in a separate thread. Thanks to this, we can wait for a connection and communicate with other devices simultaneously. Once a connection with a client is established, this connection will run using a different port number than the one of the server (this is the normal behaviour of TCP sockets). Thus, our server will always be reachable under the same port for new connections.

If a connection was established, the callback function is called with the appropriate message. Below we list all the messages that `Communicator` may pass to the callback function which should obviously know what to do with those messages.

`EVENT:NEW_CLIENT:DRONE:ID` A new client has connected successfully with `ID`. `DRONE` signifies that a new drone was connected rather than a smartphone. However, we did not implement the latter.

`TCP:ID:IMAGE:Client_ID.png` A image was received from client `ID` and stored to `Client_ID.jpg` in the working directory.

`TCP:ID:TEXT:DATA` A string message was received from client `ID`. The `DATA` field contains an integer of arbitrary length. The content of data can be parsed as well, see Listing 2.1.

`TCP:S:TEXT:DATA` A string message was received from the server. This applies only to client mode. Like above, `DATA` contains a string message.

As soon as a connection has been established, we can start communicating. `ComLib` creates a separate thread for each connected socket and listens to all connections simultaneously using the `waitForData` function. This method may receive either an image or just plain text. If some plain text is received, it will be sent to the callback-function using

```
self.callback('TCP:' + str(ID) + ':TEXT:' + data)
```

However, if a message starts with `StartOfImage`, the library will treat the following data as image and write it to a file using the following code (Listing 2.2). The three bytes `EOF` mark the end of an image. Hence, if a transmission ends with `EOF`, `ComLib` assumes the file transfer is complete.

Listing 2.2: Code to Receive an Image

```
while True: # We are now receiving an image...
    if PartOfImage[-3:] == 'EOF':
        break # End of file reached
    # Read next 1028 byte from buffer
    PartOfImage = connection.recv(1028)
    imageListOfStrings.append(PartOfImage)
finalString = ''.join(imageListOfStrings)
finalImage = finalString[:-3] # Remove 'EOF'
try: # Try/Except important with file system operations
    fh = open('Client_' + str(ID) + '.png', 'wb')
    # Remove Base64 encoding
    fh.write(finalImage.decode('Base64'))
    fh.close()
    self.callback('TCP:' + str(ID) + ':IMAGE:'
        + 'Client_' + str(ID) + '.png')
except: print 'COMLIB:␣Error␣while␣writing␣to␣image'
```

We can read portions of 1028 bytes (more would be possible as well) from the socket buffer. As long as the transmission has not ended yet, i.e. `EOF` was not reached, we just append these chunks to the array `imageListOfStrings`. As soon as `EOF` was reached, we concatenate the array-elements to one single string using the `join` method. After removing the last three characters (`EOF`), we remove the Base64 encoding from the image—see *client mode* paragraphs—and write it to a file. If everything succeeded, we send an appropriate message to our callback function.

The library allows us not only to receive, but also to send plain text to the clients. This can be done using the `serverWrite` function. It takes the ID of the receiver and the message as arguments. The implementation is straight forward using a single Python socket command. Images cannot be sent since this is not required in our scenario.

Client Mode

As mentioned above, a `Communicator` object can be put into client mode as well. Its behaviour is different in that case. First, it will connect to the server (using the same socket-techniques as in server mode). If this is successful, a data listener is started in a separate thread. Receiving plain text data works exactly the same as in server mode. Images cannot be received in client mode since this is not implemented in server mode either.

To send plain text messages to the server, the `write` function is used which takes only the message as argument. To send images to the server however, there is the function `writeImage`.

Listing 2.3: Code to Send an Image

```
# to announce an image to the server...
self.sock.sendall('StartOfImage')
try:
    with open(image, "rb") as imageFile:
        # encode image with base65 encoding
        imageAsString = base64.b64encode(imageFile.read())
        #binaryData = imageFile.read()
    self.sock.sendall(imageAsString)
    # send end of file message
    self.sock.sendall('EOF')
    print 'COMLIB:␣Sent␣image␣to␣server:',image
except:
    print 'COMLIB:␣Failed␣sending␣image␣to␣server:',image
```

The implementation is rather straight forward. Since we open the socket in a non-binary way (which makes it very easy to send plain text messages), we cannot send the image as a bitstream directly. Thus, we need to encode it into a proper string. This can be done using the Base64 code [14]. In short, this converts the binary image data into a string of ASCII characters. Thus, upon receiving the image, the server has to decode the image data once again. Luckily, Base64 encoding is supported by Python using the standard library `Base64`. Client to client communication is not implemented, since we do not need it.

2.4.5 Communication with the Cable Rollers

The cable roller is described in detail in subsection 2.7.1. Since the motor is controlled using an Arduino, we use serial communication via the USB interface of the Arduino. The server can send commands to the cable roller by writing a properly formatted string to the serial connection. Any output can be read from the serial connection by the server if desired—nevertheless, our system does currently not require such a feedback.

On the server side, the connected Arduino appears as a standard USB device, i.e. as `/dev/ttyACM1`. The script will try different devices to find the Arduino which is connected via USB. Once a simple serial connection is established, reading and writing is straight forward. We created a function `rollCable` which takes four integers as arguments (x and y coordinates of start and end point of the drones journey), formats them into one string which is then sent to the cable roller.

2.5 Localisation of the Smartphone

2.5.1 Problem and Main Idea

The main task of the localisation is to determine the position of the smartphone with high accuracy. This can be achieved by using a variety of different approaches (some of these are described in subsection 4.4.2). After we elaborated several of these approaches, we decided to focus on the smartphone localising itself using built-in sensors. We came up with the idea of putting a glass plate on top of the table and let the phone take a picture of the surface beneath the glass. The phone sends the image to the server, where its position is calculated using a computer vision algorithm. Finally, the calculated position is sent to the drones defining a new target.

The biggest challenge of such an approach is to achieve robustness against all sorts of disturbances and variations in the picture taken by the phone. Luckily all optical distortions can be ignored, since the smartphone is always lying perfectly flat on the table. However, all disturbances due to lighting cannot be ignored as the desired area of application might feature very deviating lighting. Furthermore, variations in scaling must be considered as well, as different smartphone cameras might feature varying resolutions and zooming ratios.

2.5.2 Overview of Image Recognition Process

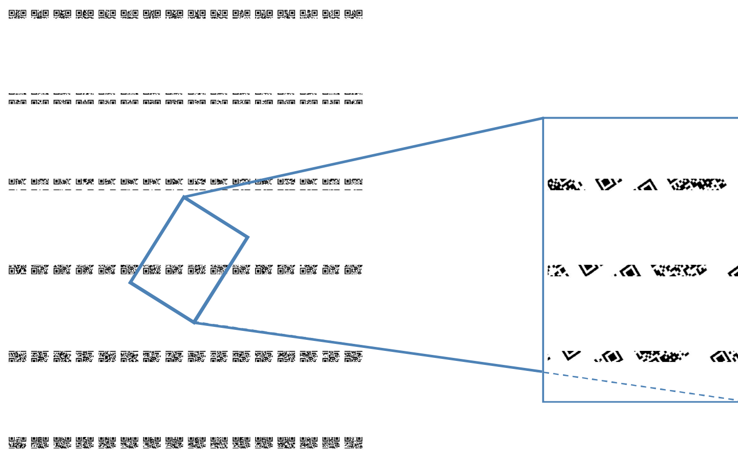


Figure 2.9: Main Idea of Phone Localisation. The left part shows the QR map, the right side shows the photo taken by the phone.

The task of the computer vision algorithm described above is to determine the phone's position given only the photo of the surface beneath the phone. This can only be achieved by relating at least two points in the photo to the global table pattern. In our project the table pattern is a map of QR codes where the values stored in the codes describe their positions on the table. Therefore, the QR codes provide the mentioned relation between the photo and the QR map.

The main idea of the image recognition is to analyse both the rotation and the scaling of the photo taken by the phone. This is achieved by separating the image into QR codes and the white spacing between the QR codes, further referred to as lanes. We can calculate the rotation of the photo as the angle of the lanes. The scaling of the photo is determined by the

Figure 2.10: Overall Image Recognition Process

distance between two lanes. We analyse these parameters using the lines described in section 2.5.3. With this information, the corners of two distinct QR codes are calculated and the codes are cropped from the photo and analysed. Since the QR map and the image taken by the smartphone are neither the same size nor rotated equally, two distinct coordinate systems are used to describe points on the planes. Therefore, we need to transform the calculated coordinates from the photo coordinate system to the table coordinate system. The values of the QR codes combined with their positions inside the photo offer enough information to do this. Thus, the centre of the photo, i.e. the position of the camera, can be transformed by applying an affine transformation combined with a displacement vector.

QR Map

Assuring that the approach described above works properly requires the surface beneath the glass plate to have a unique pattern for every image section in order to avoid ambiguity. We achieved this by building a QR map consisting of a big matrix of distinct QR codes.

The matrix consists of 60 rows and 92 columns and accordingly contains 5520 distinct QR codes as shown in Figure 2.11. The values stored in the codes describe the position the code is situated inside the matrix. The distance between two centres of two neighbouring QR codes is 12.663 mm. The distance from the table edge to the centre of the first QR code is referred to as margin. This margin in x-direction is 376 mm and the one in y-direction is 21 mm.

Figure 2.11: Arrangement of QR map on table with margin on both sides

2.5.3 Software Implementation of the Image Recognition

Third Party Libraries

The algorithm requires tools for analysing images which are already implemented in many open source libraries. We used the open source code scanning framework zBar [15] to analyse the QR codes. For image operations connected to zBar, the Python Image Library (PIL) [16] is used. For more advanced operations on images such as thresholding used in section 2.5.4, the open source computer vision library OpenCV [17] is used. Even tough OpenCV was originally designed for C++, it features bindings for usage with Python and even supports the use of the NumPy (Numerical Python) [18] library for a more efficient operation.

Class Structure

The software implementing our algorithm is divided into the five classes shown in Figure 2.12. The most important class called `QRMapAnalyser` implements the phone localisation itself and uses the other four classes. Three of those classes implement geometric objects, namely vectors, matrices and lines. They implement basic operations and are therefore not described at all. The fifth class is a helper class to use the zBar framework.

Figure 2.12: Classes implementing phone localisation

Important Geometric Objects

The phone localisation is based on finding structures inside the QR map. The most important objects used during the phone localisation are summarised in the following description and visualised in Figure 2.13.

`seedPoint`
: Starting point for further analysis situated inside a lane, see subsection 2.5.5.

`rotLine`
: Line fitted inside a lane describing the rotation of the photo, see subsection 2.5.7.

`nextLineAlongside`
: Line fitted into a lane along the QR code. This line is orthogonal to `rotLine`, see subsection 2.5.8.

`nextLineOpposite`
: Line fitted into a lane opposite the QR code. This line is parallel to `rotLine`), see subsection 2.5.8.

2.5.4 Preparation

Start of the Analysis

To start the phone localisation, one must create an instance of the `QRMapAnalyser` class. All the steps of the image recognition are called inside the constructor.

Figure 2.13: Important geometry objects

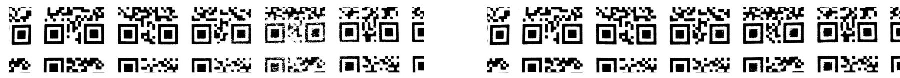
The parameter `searchOnSide` is used to pass the side in the image the algorithm should search for a QR code (either left or right). This is used to assure that by running the analysis twice on the same image, two distinct QR codes are found. The results of the analysis are stored as members of the created instance.

Read the Image

Before the recognition process can start, the image must be read from memory. Its dimensions are stored for later use and its orientation is tested. Since the algorithm is designed for landscape orientation, images in portrait mode get rotated to landscape orientation. The implementation of these steps is straight forward and not explained here.

Binary Image

Finding certain structures inside an image can only be achieved by dividing pixels or groups of pixels into different categories. For our problem, it is necessary to distinguish between *QR codes* and *lanes* (refer to section 2.5.3). To do this, the image should be as clear as possible, thus black and white



(a) Non-adaptive Thresholding (b) Gaussian Thresholding

Figure 2.14: Comparision of thresholding methods. One can clearly see the shadow of the phone in image a). This issue is solved using adaptive thresholding as seen in b).

(also referred to as binary) in order to simplify the segmentation to a certain group. The image is transformed to a binary image using greyscaling and thresholding [19]. First, the image is converted to greyscale using OpenCV. Then, all the pixels with a colour value above a certain threshold (thus with a higher grey value) are converted to white pixels and all the pixels below this threshold are converted to black pixels.

The picture taken by the smartphone might include shadows and other disturbances. Thus, applying a naive threshold leads to bad results, since the optimal threshold value changes throughout the image. The OpenCV library offers a more advanced thresholding function which uses Gaussian thresholding called `adaptiveThreshold()`. Using Gaussian thresholding eliminates the shadowing problem, since it uses a weighted sum of the pixels neighbouring the considered pixel [20]. Therefore, the thresholding value changes throughout the image according to the environment the considered pixel is located in. The following comparison displays the effect of using the function `adaptiveThreshold()`.

The code used to convert the image to binary is rather simple and not explained. The `adaptiveThreshold()` function receives the number of pixels considered for the Gaussian thresholding and an additional colour value each pixel is shifted by during the thresholding. This leads to better results in case the overall brightness is not ideal. These values are set to 111 and 45 respectively, as this lead to the best results during testing.

2.5.5 Seedpoint Search

Before the rotation analysis can begin it is necessary to find a good starting point inside the image. Hence, the first step of the image recognition is to find the so called seedpoint. The seedpoint has to be situated inside a

Figure 2.15: Seedpoint Search

Yellow:	Rejected seedpoints
Blue:	Good seedpoints (replaced by better one)
Green:	Final seedpoint

lane described in subsection 2.5.7. For stability reasons it is also desirable that the seedpoint is in the inner part of the image (since the algorithm might find a partly cropped QR code which cannot be analysed). Finding a seedpoint is implemented in the following steps:

Generate Random Point Define the inner part of the image and generate a random point inside it.

Analyse its Environment Define a rectangle around the random point and calculate the percentage of white pixels inside this rectangle. If this percentage is new a maximum, the point is stored as new best `seedpoint`.

These steps are repeated 200 times to assure that a good seedpoint is found. The following figure shows an exemplary seedpoint-search on the left side and the analysed rectangle of the final seedpoint on the right side.

2.5.6 Identification of QR Codes

The steps described in the next two sections rely heavily on finding the next obstacle i.e. the next QR code in the image. Finding such an obstacle is achieved by analysing the colour around the endpoints of the considered

line. In fact, the endpoint itself and a ten pixels ahead of it are analysed. The criterion for finding an obstacle is that more than thirty percent of the analysed pixels are black. This margin is implemented in order to make the method robust against pixel errors in the image.

2.5.7 Rotation Analysis

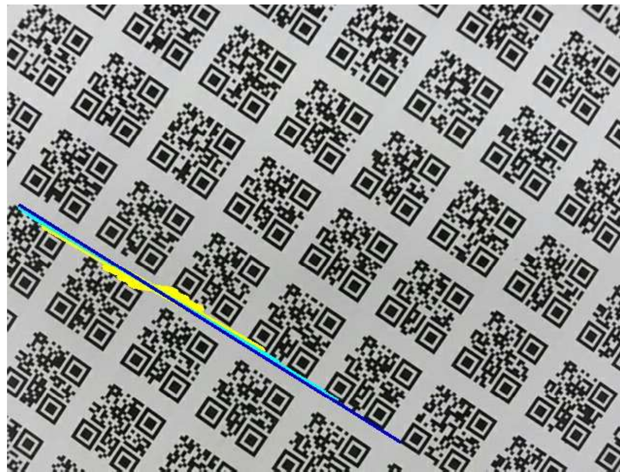


Figure 2.16: Rotation Analysis

Rough analysis:	Yellow:	Rejected lines
	Orange:	Final line
Precise Analysis:	Light Blue:	Rejected lines
	Dark Blue:	Final line

In order to crop a QR code from the image and pass it to the `zBar` class, the rotation of the codes must be analysed. The rotation can be determined by finding a lane between the QR codes and calculating its angle with the x-axis. This is implemented as follows:

New Line Define a new line pointing from the seedpoint into the y-direction.

Rough Analysis Extend the line until it hits a QR code. If the extended line is shorter than a threshold, the line is rotated by 6 degrees and extended again. These steps are continuously repeated.

Precise Analysis Extend the line found after the above steps until it hits a QR code. If this extended line is shorter than a more restrictive threshold, the line is rotated by 1 degree and extended again. These steps are continuously repeated as well. The final line is then stored as `rotLine`.

Obtain Angle Calculate the angle of `rotLine` using basic trigonometry and return it.

2.5.8 Scaling Analysis

The scaling of the image and thereby the size of one QR code might vary between different phone cameras and cannot be chosen as a fixed constant. The scaling analysis can be divided into two subtasks: finding a lane alongside the QR code (orthogonal to the `rotLine`) and finding the lane opposite of the QR code (parallel to `rotLine`).

Lane Alongside the QR Code

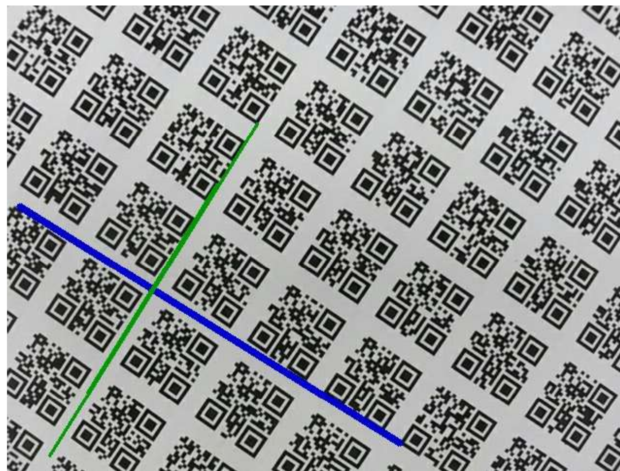


Figure 2.17: Lane alongside the QR Code
Light Green: Rejected lines
Dark Green: Final line (the longer one)

Orthogonal Line The `rotLine` is rotated by 90 degrees.

Fit into next Lane The orthogonal line is copied, resized to its initial length of 1 and shifted to both sides. Both of the shifted lines are then extended until they hit a QR code. If one of the extended lines is longer than a threshold, the orthogonal line is stored as `nextLineAlongside`. If this does not hold, both the orthogonal lines are shifted further. Then the process of extending until hitting a QR code and testing the length of the extended line is repeated.

Lane Opposite the QR Code

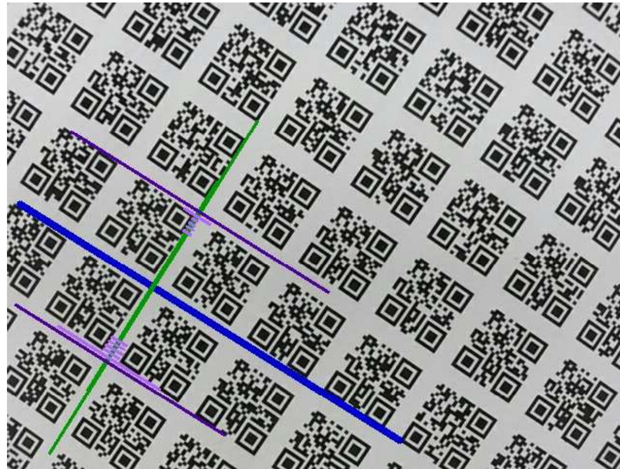


Figure 2.18: Lane opposite of the QR code
Light Violet: Rejected lines
Dark Violet: Final line (the longer one)

Parallel Line The `rotLine` is copied and duplicated.

Fit into Next Lane The parallel line is resized and shifted to both sides by an initial shift. Both of the shifted lines are extended until they hit an QR code. If one of the extended lines is longer than a threshold, the parallel line is stored as `nextLineOpposite`. If this does not hold, both the parallel lines are shifted further. Then the process of extending until hitting a QR code and testing the length of the extended line is repeated.

2.5.9 Extraction of the QR Code

Now we want to use the obtained lines to actually extract a QR code from the image. To achieve this, the corners of a QR code must be found.

Corners of the QR Code

The corners are chosen in a way such that choosing a incomplete QR code can be avoided.

First Corner Calculate intersection of `rotLine` and `nextLineAlongside` and store is as `firstCorner`.

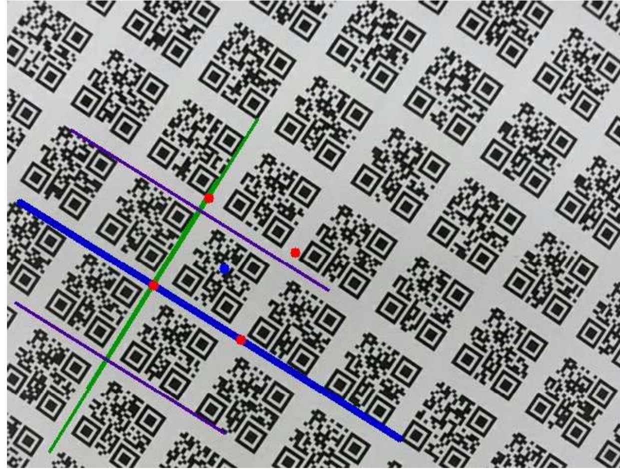


Figure 2.19: Corners of QR code
Red: Corners of the QR code

Second Corner Create four points by adding the length of one QR code into each cardinal direction (referred to the direction of `rotLine`). Find point with smallest distance to the middle of the image and store is as `secondCorner`.

Third Corner Create two corners by adding the length of one QR code vertically to the connection of the existing corners (positively and negatively). Find corner with smallest distance to the middle of the image and store is as `thirdCorner`.

Forth Corner Calculate position of forth corner (already defined by previous corners)

Cutting-out

Unfortunately, only non-rotated rectangles can be extracted using OpenCV for Python. This requires the image to be rotated by the angle found in subsection 2.5.7 in order to align the QR code with the image borders. This requires the corners calculated in section 2.5.9 to be rotated as well, which is solved using a linear transformation. The rotated corners now define the borders of the rotated and therefore aligned QR code. Finally, the QR code can be cropped from the image.

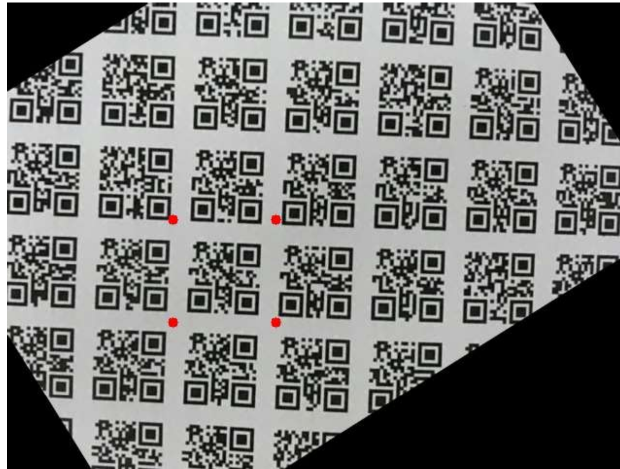


Figure 2.20: The rotated QR code is now ready to be cut out

2.5.10 QR Code Analysis with zBar

The value stored inside the cropped QR code is analysed with zBar. Using this library makes this task very simple: the image is passed to the scanner object. If a code is found, its value can be accessed by calling `zBarImage.data`. The variable storing the QR code value is initially set to -1. To find out whether the analysis was successful, this value is checked to be not equal to -1 after the image recognition process.

2.5.11 Transform Coordinates to the Table

As mentioned in subsection 2.5.2 we analyse two distinct QR codes. We now transform their position to the table coordinates to find the location of the receiver coil.

Coordinate Systems on Picture and Table

After the above described algorithm of image recognition, the position and value of two QR-codes in the picture are known. There are two distinct Cartesian coordinate systems on the QR map and the photo taken by the phone. Hence, a simple coordinate transform must be applied. The orientation of the coordinate systems are shown in Figure 2.21.

Figure 2.21: The left part is an illustration of the picture taken by the smartphone, the right one represents the QR map on the table.

QR Position on Table

The QR-codes on the table are arranged as shown in Figure 2.11. The coordinates on the picture are measured in pixels and those on the table are measured in milimeters. Therefore the coordinates of the centres of a QR-code are calculated as follows.

$$\begin{pmatrix} x_{\text{table}} \\ y_{\text{table}} \end{pmatrix} = \begin{pmatrix} (QR_{\text{value}}/100) \cdot l + x_{\text{margin}} - l \\ (QR_{\text{value}} \mod 100) \cdot l + y_{\text{margin}} \end{pmatrix}$$

Transformation of Coordinate System

To be able to do a simple affine coordinate transform, the point of origin in both coordinate systems is set at the centre of the first QR code. This means the centres of the QR codes in Figure 2.21 and table now are

$$\vec{P_1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \vec{P_2} = \begin{pmatrix} u_2 - u_1 \\ v_2 - v_1 \end{pmatrix}, \quad \vec{Q_1} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \vec{Q_2} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix}$$

Then the affine transformation matrix that satisfies the following equation is calculated as described below.

$$A \cdot \overrightarrow{P_1 P_2} = \overrightarrow{Q_1 Q_2} \qquad \text{with } A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Listing 2.4: Calculation of matrix A

```
# Calculate transformation matrix
    a11 = (dx*du+dy*dv)/float(pow(du,2)+pow(dv,2))
    a12 = (dx*dv-dy*du)/float(pow(du,2)+pow(dv,2))
    a21 = (dy*du-dx*dv)/float(pow(du,2)+pow(dv,2))
    a22 = (dy*du+dy*dv)/float(pow(du,2)+pow(dv,2))
```

In this code `dx`, `dy`, `du` and `dv` are the distances where

$$dx = x_2 - x_1 \quad dy = y_2 - y_1 \quad du = u_2 - u_1 \quad dv = v_2 - v_1$$

The calculated matrix A will now transform any vector on the picture to its corresponding vector in the coordinates of the table (with the coordinate system's point of origin at the first QR code).

Calculation of Coil Position

The vector from the first QR code to the centre of the picture $\overrightarrow{P_1 M_{\text{pic}}}$ and the unity vectors in u and v directions are transformed to the coordinates of the table. Then the offset of the first QR code is added.

$$\vec{Q}_{\text{camera}} = A \cdot \overrightarrow{P_1 M_{\text{pic}}} + \vec{Q}_{\text{QRoffset}} \quad \vec{u}_{\text{table}} = A \cdot \vec{u} + \vec{Q}_{\text{QRoffset}} \quad \vec{v}_{\text{table}} = A \cdot \vec{v} + \vec{Q}_{\text{QRoffset}}$$

The transformed centre of the picture corresponds to the location of the camera on the table. The offset (in u and v direction) from camera to coil is measured in millimetres[1] and added to the camera position.

$$\vec{Q}_{\text{coil}} = \vec{Q}_{\text{camera}} + u_{\text{offset}} \cdot \vec{u}_{\text{table}} + v_{\text{offset}} \cdot \vec{v}_{\text{table}}$$

The coil position is then sent to the drone.

2.6 Drones

The main task of the drones is to position the emitter coil as precisely as possible at a given target position. We analysed different approaches to achieve this, some of which are described in subsection 4.4.2. Finally, we decided to position the coils with independently moving drones. Contrary to the majority of vehicles called *drone*, the robots positioning the emitter coils drive on wheels.

[1]Set in a feature in the app, see subsection 2.3.2

Figure 2.22: Photo of the final prototype

2.6.1 Main Hardware Components of the Drone

The chassis and the wheels of the drone are 3D printed, we designed them using the 3D modelling software SketchUp Make [21]. Figure 2.22 shows the overall composition of the drone. The most important components mounted on the drone are labelled in the photo. Since the final drone is still a prototype, the design was not of highest priority.

Qi Charger

The main task of the drone is to charge the user's smartphone wirelessly. This is achieved by transferring energy between two coils using electromagnetic induction [22]. Since implementing such a charging system is a fairly hard task itself, we used a Qi charger kit including the emitter coil and its controllers. The charger is powered via USB and can charge a smartphone with a current of up to 500 mA at a fixed voltage of 5 V [23]. The charger implements the Qi standard and is therefore compatible with all Qi certified hardware. The charger features a small LED that indicates its state.

Wheels and Motors

The drone is driving on two 3D printed, plastic wheels. Because the friction between plastic and paper is very low, an elastic band is spanned around each wheel. Both wheels are directly mounted onto the shaft of the corresponding motor, so the gear transmission ratio is equal to one i.e. no transmission is needed. The wheels are powered by two independently controlled, bipolar stepper motors manufactured by Trinamic [24]. As the drone should be as compact as possible to fit inside the gap between the glass plate and the QR map, the motors should not consume too much space. This is why we decided to use motors with cases measuring only 28mm by 28mm (NEMA 11). The motors are driven at a constant voltage of 8 volts and consume up to 1.5 A each. The overall power consumption does not change between driving or holding the current position. Thus, the motors must be turned off when the drone stands at a fixed position for a longer period of time.

Using a stepper motor to drive a drone might seem unusual but enables us to achieve the required accuracy. Stepper motors do not move continuously but move step-by-step (as suggested by the name). The functional principle is similar to a normal electric motor since a magnetic field is applied to the shaft to move it. However, in stepper motors the shaft is connected to a gear which discretises the positions of the shaft [25]. The overall accuracy of the motor is defined by the fineness of this gear. Nevertheless, the motor can achieve higher accuracy with the same gear by using different stepping styles. Most stepper motors support the following stepping styles: full, half and micro stepping. The difference between these is the timing and location at which the magnetic field inside the motor is applied. Driving at full stepping is faster but features lower spacial resolution as half or micro stepping.

The high accuracy of a stepper motor can also be achieved with a normal DC motor by combining it with a rotary encoder [26]. A rotary encoder is a sensor measuring the rotation of a motor. This feedback can be used to add a closed loop controller to the system. This approach would have been significantly more expensive and did not match our financial limits. Furthermore, the stepper motors allow us to control the drone with an open loop controller which is easier to implement.

Raspberry Pi and WiFi

Since the drone itself is controlled by the server, it needs some sort of computer to process new target positions, recalibrate itself by taking pictures

or report error messages. This is all handled by a Raspberry Pi 2 Model B mounted on the drone. This Raspberry Pi obviously needs to be connected to the server. In order to reach maximum flexibility and minimize the number of cables to the drone, we decided to use WiFi for this connection. Since the Raspberry Pi 2 does not feature onboard WiFi, we extended it with a WiFi dongle manufactured by Edimax [27].

Motor Shield

Both motors are controlled by an extension board stacked on top of the Raspberry Pi [28]. The board is manufactured by Adafruit and gives the Raspberry Pi the ability to control up to two stepper motors or four DC motors independently. The shield is a so called HAT (Hardware Attached on Top), the communication with the Raspberry Pi runs via I^2C [29]. The shield can be programmed using Python and a library offered by Adafruit. One can easily control the motors with the methods `step(numberOfSteps, direction, stepStyle)` and `oneStep(direction, stepStyle)`. The difference between these functions is that the first one is blocking, while the second one is not.

Camera for Position Calibration

As perfectly accurate driving cannot be achieved, the drone has to be able to localise itself on the table. This is accomplished using the same approach as for the users phone, thus by taking a picture of the QR map beneath the drone. Naturally, this requires the drone to have a camera. The Raspberry Pi Camera fits our needs perfectly: it is small, easy to use and offers a sufficient resolution (5MP) as well as an adjustable focus length. This is important, because the distance between the camera and the QR map is less than seven centimetres.

Measuring Equipment for Fine Search

Again we make the assumption that perfect positioning of the coil is not possible on the first try. Even though, getting close to the coil seems possible. In order to find the user's smartphone inside this target region, the drone requires a routine to search for the receiver coil. The Qi charger kit used by the drone provides a status LED, which is lighted blue by the Qi charger whenever a stable connection between the charger and a receiver coil is established. The drone uses this status LED by analysing the voltage between the LED pins. In order to do this, the LED pins are connected to

two GPIO pins of the Raspberry Pi. Luckily, no amplifier is needed as the pins threshold voltage is below the blue light LED voltage.

2.6.2 Drone Start-Up and Control Software

The main task of the software running on the drone is to translate commands sent by the server into actual commands for the shield controlling the steppers. Most of this translation process is implemented in two classes, the `Drone` and `Computation` class. Figure 2.2 gives a brief overview of the software running on the drone.

After the Raspberry Pi is booted, the script `startup` checks the WiFi connection and waits for the connection to be established if necessary. It then goes on and tests whether all connection settings are valid and if so, starts the `main` script.

The `main` script is both the creator of the `drone` object described in subsection 2.6.3 and the parser between the drone and the server described in subsection 2.4.3. The drone object represents the drone itself and is controlled by the parser. The parser is continuously waiting for commands from the server and parses them upon receiving. According to the received command, the parser invokes the corresponding methods of the drone object, e.g. moving the drone to a new position.

2.6.3 Drone Functionality

The `Drone` class is one of the fundamental classes of our project. It contains a few central methods, which are explained in more detail based on the actual code.

Send Drone to new Target Position

The method `goToTarget` is used to send the drone from point A to point B. The main idea is to rotate the drone first and then drive it to the target position on a straight line. As the drone stores its position and angle, they need to be updated at the end of every movement. The implementation is shown in Listing 2.5.

Listing 2.5: goToTarget, used to send drones to new target (parts removed)

```
def goToTarget(self, tarX, tarY):
    #Rotate drone
    newAngle = self.rotateDrone(tarX, tarY)
    #Move drone
```

```
    self.moveDrone(tarX, tarY)
    #Update Location
    self.updateLocation(tarX, tarY, newAngle)
```

As we can see, this is a rather high level function, so we dive deeper into the used functions `moveDrone` and `rotateDrone`.

Drive Straight Line

The `moveDrone` function is a good example for the general process of moving the drone: first, we calculate the distance to get to the target. This distance is then transformed into a command for the motors i.e. into a numbers of steps that each motor should make. Finally, the method `startDoSteps` is called in order to move the drone. The code implementing the function `moveDrone` is shown in Listing 2.6.

Listing 2.6: moveDrone, used to drive straight lines (parts removed)

```
def moveDrone(self, tarX, tarY):
    #Calculate distance to target position
    dist = Comp.computeDist(self.curX, self.curY, tarX, tarY)
    #Calculate number of steps it takes to get there
    numberOfSteps = Comp.getStepsToTarget(self, dist)
    #Start moving
    self.startDoSteps(numberOfSteps, numberOfSteps)
```

Rotate Drone

The basic structure of the moving process is used to rotate the drone as well. We start by calculating the angle and transform it into the number of steps the motor has to drive. Contrary to the `moveDrone` function, the value of the angle needs to be considered since the motors need to be controlled differently for right and left turns and must not drive over his own cable. The implementation of the `rotateDrone` function is not shown here as it is very similar to Listing 2.6.

Calculate the Number of Steps

The functions used to calculate the distance and angle to get to a target are included in the `Computation` class. Their implementation is straight forward and not explained here. However, the derivation of the two formulas used to transform the distance and the angle to a number of steps is not

Figure 2.23: On the left side of the figure we can see a wheels profile. In the right part the drone is shown from top view.

trivial and needs a fairly good spatial sense. It is best explained using the two illustrations shown in Figure 2.23.

The formula used in `getStepsToTarget` calculates the number of steps N_s it takes to drive along a straight line. The length of this straight line is further denoted as l. The stepper used in our drone needs to step two hundred times to do one complete turn. Thus, the drone will drive exactly two hundredth of the wheels circumference when the motor is doing only one step. The wheels of the final prototype have a diameter of $d = 45$mm. Therefore, the distance covered in one step is equal to

$$s = \frac{\pi \cdot d}{200} = 0.706\text{mm} \tag{2.1}$$

The number of steps required to drive a distance l can be derived directly and is equal to

$$N_s = \frac{l}{s} = \frac{200 \cdot l}{\pi \cdot d} \tag{2.2}$$

The formula used in `getRotationToTarget` calculates the number of steps N_r it takes to rotate the drone by a certain angle β. The drone is rotated by turning both wheels into opposite directions. Therefore, the centre of the drone does not move at all. By looking at Figure 2.23 one can easily see that N_r depends on the distance between the wheels and the rotation centre i.e. the wheelbase b. In the final prototype, the wheelbase is equal to $b = 45$mm.

Next, we will derive a relation between the turning of the wheels and the rotation of the drone itself. By doing one step the wheel covers the distance $s = 0.706$mm as described above. Knowing that the wheel will move on a circular curve, one can see that s is equal to the arc length a, by

which the drone itself is rotated in one step, thus $a = s$ holds. The angle of centre θ (measured in degrees) of the sector created by the arc length a can be calculated as

$$\theta = \frac{180 \cdot a}{\pi \cdot b} \tag{2.3}$$

Now, as θ describes the angle the drone is rotated after one step, the total number of required steps N_r is equal to

$$N_r = \frac{\beta}{\theta} = \frac{\beta \cdot b}{180 \cdot a} \tag{2.4}$$

Substituting $a = s$ into (2.4) leads to

$$N_r = \frac{200 \cdot \beta \cdot b}{180 \cdot d} \tag{2.5}$$

Start the Steppers

The function `startDoSteps` starts the movement of the drone. The function invokes the `doSteps` script twice, each of them running in a new sub-process. The actual code is straight forward and not displayed here. In the `doSteps` script the `stepperWorkers` are started and the drone starts moving. The `stepperWorkers` are the most low-level component of the control process and described in subsection "Stepper Control" of section 2.6.3.

We outsourced the actual controlling into sub-processes since the library from Adafruit uses blocking methods to control the motors. Hence, controlling the motors within the same thread would make the drone unresponsive. Using two sub-threads made the steppers started moving back and forth randomly. We assume that the two threads interfered with each other which caused the motors to receive mixed up PWM signals[2] from the motor shield. This was solved by using two processes [3], but complicated the communication between the different routines. We decided not to implement this but rather let the main process wait for the sub-processes to finish and then continue it. This is achieved by using `Popen` to start the new process and `process.communicate()` to wait for the sub-processes to finish.

[2]PWM stands for Pulse-width modulation, see [30]

[3]Processes do not share their virtual memory and are more independent than threads

Stepper Control

The stepperWorkers mentioned in subsection "Start the Steppers" in section 2.6.3 are explained in full detail here. First, the input arguments are checked for validity in order to prevent false movement. After this, the stepper object is created using the Adafruit library and the speed of the stepper is set. The function `myStepper.step(stepCount,dir,stepStyle)` is called according to the moving direction. This is the blocking method described in subsection "Drive straight Line" in section 2.6.3. We decided to use double stepping, which combines high torque with decent speed[4]. Finally, the motors are turned off which is very important as the motors consume the same amount of power while being on hold as while driving (see section 2.6.1). Not turning them off does not only waste a lot of energy but might eventually damage the drones because of the generated heat.

Listing 2.7: stepperWorker, used to control steppers (parts removed, indentation changed)

```
#Detemine direction
if stepCount > 0: forward = True
else: forward = False
#Create stepper object
mh = Adafruit_MotorHAT()
myStepper = mh.getStepper(200, stepperID)
#Start moving
if forward: myStepper.step(stepCount, FORWARD, DOUBLE)
else: myStepper.step(stepCount, BACKWARD, DOUBLE)
#Turn off motors
mh.getMotor(1).run(RELEASE)
```

Fine Search

Once the drone has reached its target position, the coils of the Qi charger and the receiver are most probably not aligned precisely enough yet. Thus, it is required to perform some kind of fine search. Our Qi charging kit features an LED that lights up in blue colour if it is transmitting power to a receiver. We managed, by reverse engineering, to find two contacts on the board where the voltage changes from zero to about 1.5 volts when the light turns blue. By soldering cables to these contacts and connecting them to the GPIO pin 4 and ground of our Raspberry Pi[5], we are able to detect

[4]For further information about stepping styles refer to [31]

[5]Although the HIGH level of the Raspberry is at 3.3 volts, this has worked flawlessly so far. It seems that the threshold voltage is below 1.5 volts.

whether the Qi kit is charging the smartphone or not.

We programmed the drone to slowly move first forward for about 5 cm[6], then backward twice the distance, then forward again to return to the initial position. Next, the drone will turn a small angle and restart the same procedure, until it is again pointing into the initial direction. If during this process the drone detects that the Qi kit started charging, it stops and sends an acknowledgement back to the server. If the fine search process ended without success, a negative acknowledgement is sent to the server to indicate failure.

Recalibration of the Drones Location

Using an open loop system to control the motors, the drone requires some sort of recalibration system to gauge its position as we cannot react to disturbances such as friction losses or inaccurate motor movement. To tackle this problem we decided to use the same approach as for the smartphone localisation described in section 2.5 and mounted a camera onto the drone as described in subsection 2.6.1. As the Raspberry Pi Camera is very easy to use, the code implementing this is not displayed here. The routine starts by taking a photo using the command `raspistill`, which also allows very flexible adjusting of the camera settings. The photo is stored locally and sent to the server using the method described in Listing 2.3. In analogy to the smartphone localisation process, the image is processed on the server and the actual position of the drone is returned to the drone. Since the distance between the camera and the QR map is slightly smaller, a few parameters of the image recognition need to be adjusted. Finally, the drone updates its own position and is ready to receive new targets.

2.7 Cable Management and Power Supply

2.7.1 Cable Roller Hardware

All the power used by the drone is delivered with a two-core cable. To avoid a mess with the cable on the table and loose cable hindering the drone, the cable is wound up on a cable reel that is produced with a 3D printer. Inside the reel is a cable untangler to avoid unwanted contortion of the cable to the power supply. To control the cable length on the table,

[6]This distance seems reasonable considering our measurements in Table 3.1 that show a maximum driving error of about 4 cm if the drone drives across the whole table

Figure 2.24: Setup of the cable roll with all its components.

a bipolar stepper motor[7] is directly mounted on the cable reel. The reel is held horizontal by the untangler and the motor as shown in Figure 2.24.

To control the motor, it is connected to an Arduino Motorshield V5 which is attached to an Arduino UNO. The Arduino UNO communicates with the server on the table via USB cable to get the coordinates to which the drone is currently driving. The communication is described in subsection 2.4.5. The Arduino UNO also runs the program which is described below.

2.7.2 Cable Roller Software

Approximations for Motor Program

To make the program compact and fast we made some approximations for values used in the calculation. Described here are the values used inside the Arduino program.

[7] The same as used in the drone, see section 2.6.1

Constant Reel Radius The cable we used to deliver power to the drone is 1.3×2.6 mm thick. The cable reel is 18 mm wide. There is never more than 2.5 meters of cable on the reel. We made the approximation of a constant reel radius with cable added of 24 mm.

Velocity of Drone The movement of the drone is composed of two parts. First it turns itself and then moves straight forward, see subsection 2.6.2. We measured the average velocity of the forward motion of 50mm/s which is needed for the computation.

Motor Speed We measured the maximum rotational speed speed of the stepper motor as 1.23 turns/s.

Spare Length An extra cable segment of 5 cm is added initially so the above mentioned approximation and the imprecision of the calculation will not cause problems.

Setup for Motor

To control our stepper motor with the Arduino Motorshield, we need a `setup` and a `loop` function. In the `setup` function all the pins for the motor we use are set to make sure the motor is accessed correctly.

The `loop` function is a process that repeats infinitely. The program for the cable roller waits until it gets an input[8]. All characters except decimal digits are ignored. The seperate numbers are then interpreted as the coordinates of the position where the drone starts and ends driving. Finally the algorithm which calculates the needed cable length and steers the motor is started.

Algorithm

Whenever the drone moves from one point on the table to another, the cable needs to be made either longer, shorter or both. To figure out at which point of the driving route the turning point between rolling the cable out and rolling the cable in is, the placement shown in Figure 2.25 is considered.

The turning point is exactly at the place where the line from A to B intersects the altitude (to vertex R) of the triangle. The intersecting point I is calculated:

$$t = \frac{(x_R - x_A) \cdot (x_B - x_A) + (y_R - y_A) \cdot (y_B - y_A)}{(x_B - x_A) \cdot (x_B - x_A) + (y_B - y_A) \cdot (y_B - y_A)}$$

[8]The input is received via the serial interface, compare subsection 2.4.5

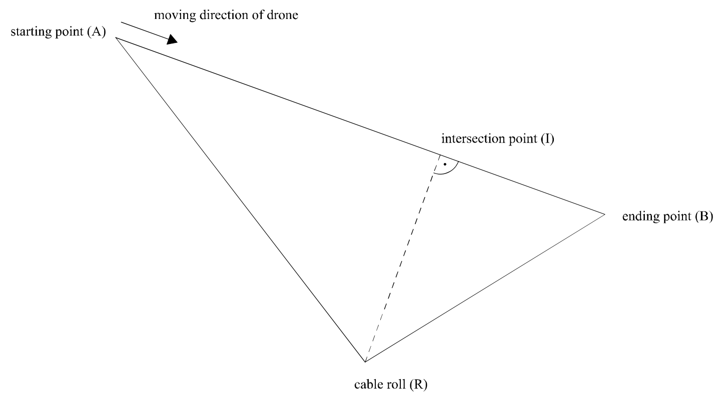


Figure 2.25: Placement of the drone and cable roll

$$x_I = x_A + t \cdot (x_B - x_A)$$

$$y_I = y_A + t \cdot (y_B - y_A)$$

Next we figure out if the cable is rolled in or out. For this the following criteria are used.

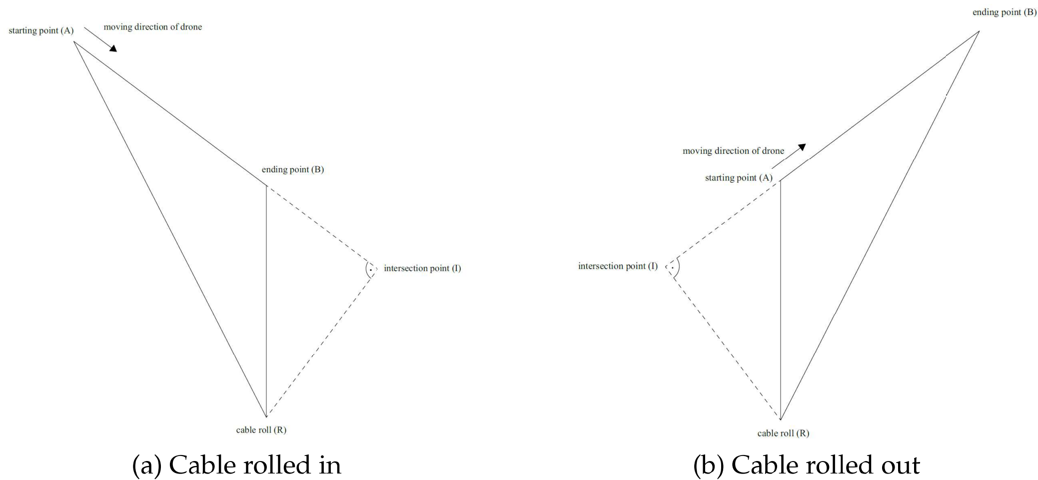
- If $|\overrightarrow{\mathrm{AI}}| \geq |\overrightarrow{\mathrm{AB}}|$
  Function `roll_in(A, B)` is called, i.e. the drone is approaching the cable roller, see 2.26a.

- If $|\overrightarrow{\mathrm{BI}}| \geq |\overrightarrow{\mathrm{AB}}|$
  Function `roll_out(A, B)` is called, i.e. the drone is driving away from the cable roller, see 2.26b.

- If $|\overrightarrow{\mathrm{BI}}| < |\overrightarrow{\mathrm{AB}}|$ or $|\overrightarrow{\mathrm{AI}}| < |\overrightarrow{\mathrm{AB}}|$
  First the cable is rolled in if the drone approaches the cable roller or rolled out if the drone moves away from the cable roller[9]. When the drone arrives at the turning point I the cable is rolled in the opposite direction.

Inside the `roll_in` and `roll_out` functions several important parameters are calculated first and then the functions that actually control the motor (in the according direction) are called.

[9] In this case the points A and B in the Figure 2.25 are swapped and the drone moves from right to left.



(a) Cable rolled in (b) Cable rolled out

Figure 2.26: Cases where cable is only rolled in or out but not both

Motor Control

`drive1` and `drive2` control the bipolar stepper motor. The algorithm used to make one single step is the one suggested on the official webpage of "Arduino Motor Shield Tutorial" [32]. The number of turns describe how many steps are made, since one turn is equal to 50 steps.

The speed of the motor is regulated with a delay time. We use delay correction for compensating any resulting speed differences between the drone and the cable roller. For the process of rolling the cable out, we subtract the delay correction so that the cable is rather a bit too fast rolled out and thus too long for a short while. For rolling the cable in, we added the correction to the ordinary delay to achieve the opposite effect. A bigger delay correction has a bigger impact on the speed.

2.7.3 Power Supply

Supplying our drone with power is not as trivial as we first expected since the long cable of about two meters has a significant resistance of about 0.5 Ω. When turned on, the stepper motors draw about 2 A, resulting in a voltage drop of about one volt over the cable. Thus, only four volts remain on the drone which is not sufficient to power the Raspberry Pi. To tackle this problem, we decided to increase the supply voltage to eight volts and use two low drop-out regulators (LDO) [33], each able to provide one ampere at five volts. One powers the Raspberry Pi, the other the Qi charging kit. The motors can deal with up to 36 volts, so we do not need to regulate their

voltage. The network is depicted in Figure 2.27. The capacitors are required to operate the LDOs. We use two stages of capacitors (smaller and bigger ones) to ideally decouple the power supply.

Figure 2.27: Power supply circuit including all the units on the drone that are powered via the cable reel.

Chapter 3

Results

3.1 Measurements of Qi Charging Kit

We use a Qi charging kit in our project. This kit consists basically of a coil (emitter coil) that is connected to a power source and a coil (receiver coil) that can be connected to any smartphone with Micro USB connector. The following measurements were made to identify the distance between the smartphone and the used Qi charging kit where the equipment is still transmitting power to the device.

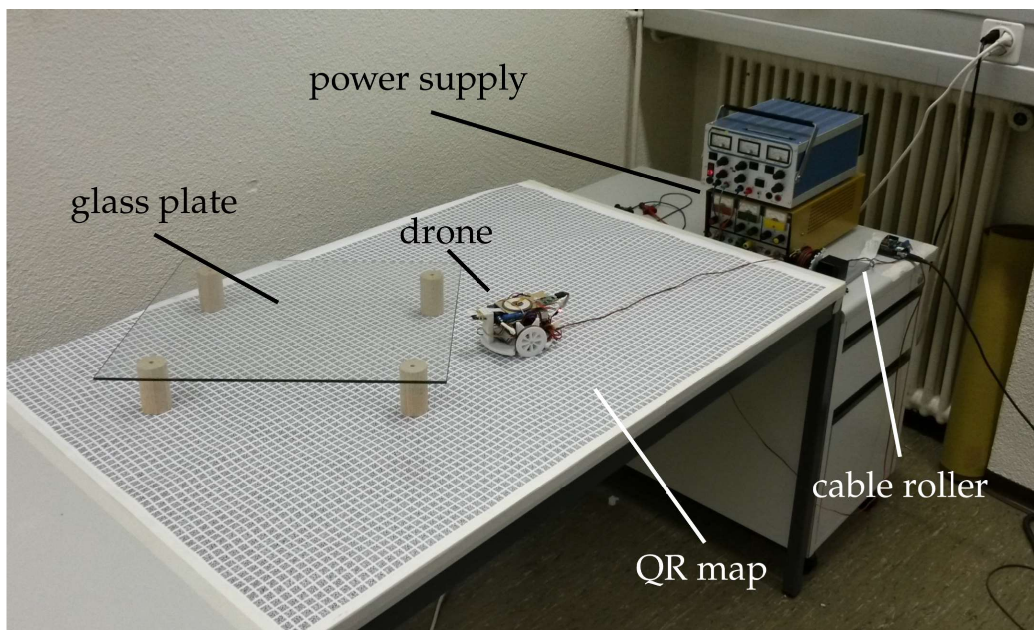


Figure 3.1: Testing environment. The server is not visible on this image.

3.1.1 Vertical Offset

The following measurements were taken on the 17.01.2016 with a Samsung Galaxy S4 mini, charged to 89%, equipped with the receiver coil. The smartphone with the receiver coil was positioned on top of the table top consisting of 4.9 mm thick glass. Under the glass was the emitter coil. The distance between emitter and receiver was gradually increased by putting sheets of paper in between the glass and the smartphone.

The results are presented in Appendix A and depicted in Figure 3.2. They show that the charging stops when the smartphone is more than one centimetre apart from the coil in a vertical direction (including the glass).

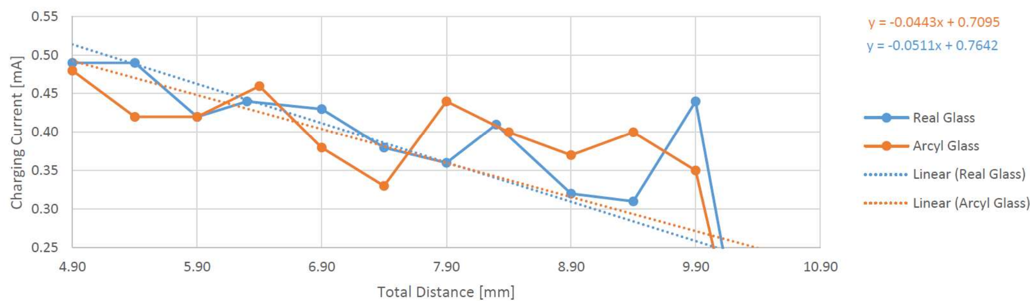


Figure 3.2: Measurements with varying vertical distance

Alignment Issues The maximal distance is very sensitive towards horizontal displacement of the phone. If the device was moved about half of the coil diameter to either side, coupling (i.e. charging current) already decreased to nearly zero.

NFC-compatibility NFC tags can still be read even if the phone is charging simultaneously. This is notable since the NFC receiver coil and the Qi receiver coil were positioned ontop of each other.

Approximately Linear Dependency We observed that the charging current decreases approximately linearly with the distance.

3.1.2 Horizontal Offset

To measure the offset in horizontal direction we first did measurements where the smartphone with the receiver coil was moved. We then also made measurements where the emitter coil is moved and the smartphone lies still. This allows better simulation of the actual use case where the drone moves with the emitter coil.

Receiver Coil Moved

The following measurements were taken on the 19.01.2016 with a Nokia Lumia 550, charged to 30%, equipped with the receiver coil. The smartphone was positioned on top of a glass plate. Under the glass plate was the emitter coil. The horizontal offset from the coil was gradually increased by moving the smartphone.

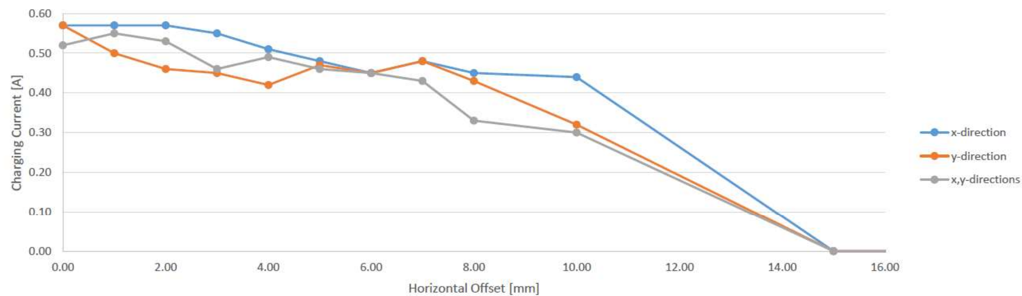


Figure 3.3: Measurements with varying horizontal distance

The results are presented in Table A and depicted in Figure 3.3. They show that the smartphone stops charging as soon there is more than one centimetre horizontal offset from the coil.

Emitter Coil Moved

In these measurements we moved the emitter coil instead of the smartphone. The measurements were taken on the 26.01.2016 with a BlackBerry Q10, charged to 52%, equipped with the receiver coil. The smartphone was positioned beneath the glass plate.

The results of this measurement are listed in Table A.6 and Table A.7. The measured values are similar to the tests where the receiver was moved. We can see that the device needs to be positioned with a maximum offset of about 8.9 mm in x and 10.2 mm in y direction. This requires our drone to position with about 9 mm accuracy, more accurate is always better since efficiency increases significantly if the alignment is better.

Additional Observations

- **Approximately Linear Dependency** We observed that the charging current decreases approximately linearly with the distance (same as with the vertical offset).
- **Cutoff Offset** After about 1 cm offset, the charging current vanishes.

3.2 Measurements of System Performance

3.2.1 Precision of Drone

With the prototype of the drone, we tested how precise the drone really drives. The results are shown in Table 3.1. When driving straight forward there was a deviation in the direction of driving between 3 and 11 millimetres. In the direction orthogonal to the direction of driving the deviation was between 9 and 105 millimetres. If the drone had to turn before moving forwards, discrepancy on both sides lay between 0 and 80 millimetres.

These values show that there is some deviation even without any rotation of the drone. This could result of the number of steps set for one turn of the wheel and imprecisions in the motor itself. The deviation orthogonal to the direction of driving can be explained by wheels that are not perfectly mounted straight. The deviation is however clearly bigger when the drone first rotates. This behaviour can be explained by imprecision of the calculated number of steps for the drone to turn around its own centre and by imprecision of the motors.

For the measurements it was difficult to place the drone in the direction it was supposed to point. An angle that is a little off the set value can imply big differences in a long drive. This means that not all the deviation measured was necessarily produced by imprecision on part of the drone but on behalf of the tester.

3.2.2 Reliability of System

To test if our implemented solution works, we run through the process consisting of multiple steps several times. We did thirteen run-throughs on the system as shown in Table 3.2. Only once the process actually ran successfully and the phone began to charge.

The most failures happened during the image recognition that timed out and at the fine search. The problem with the fine search was that the drone was either too far away to be able to detect the smartphone or it missed the smartphone by a few millimetres. There were also some unexpected things like the drone moving forwards without turning although it should and we were not able to figure out why. The time for the picture analysis fluctuates between 11.3 and 137 seconds which is quite a wide range. These are the time spans with debugging mode running. Because of saving many pictures while running in debugging mode, it was slower than normally.

Although the process is supposed to work it fails more often than not. Sometimes it even fails without obvious reason.

Table 3.1: This table shows measurements on the driving precision of the drone. It was positioned at position *from* and sent to position *to*. The column *was* shows where the drone actually ended up and *off* is the difference between the target and the actual position. Date: 29.05.2016

from x	from y	to x	to y	was x	was y	off x	off y
straight							
1500	146	500	146	504	251	4	105
1500	146	500	146	507	160	7	14
1500	146	500	146	506	179	6	33
1500	146	500	146	503	205	3	59
1500	146	500	146	511	155	11	9
1500	146	500	146	503	164	3	18
with slight turn to the right							
1500	146	500	600	520	613	20	13
1500	146	500	600	514	623	14	23
1500	146	500	600	496	595	4	5
1500	146	500	600	520	637	20	37
1500	146	500	600	499	591	1	9
1500	146	500	600	526	640	26	40
1500	146	500	600	502	605	2	5
1500	146	500	600	544	680	44	80
1500	146	500	600	502	600	2	0
1500	146	500	600	543	668	43	68
1500	146	500	600	510	612	10	12
with turn to the left							
1474	596	1100	100	1073	126	27	26
1474	596	1100	100	1064	127	36	27
1474	596	1100	100	1078	122	22	22

Table 3.2: Measurement of system performance: Every row corresponds to a run-through. The columns show whether the step was successful (*yes*) or not (*no*). Empty cells for steps that were not reached in the run-through.

app sends image	server receives image	analysis successful	image analysis time [s]	sent coordinates to drone	rotate drone	move drone	cable roller starts	cable roller length and direction correct	finesearch ended	charging
yes	yes	yes	11.3	yes	yes	yes	yes	no		
yes	yes	yes	79.6	yes	yes	yes	yes	no	no	no
yes	yes									
yes	yes	no								
yes	yes	no								
yes	yes	yes		yes	yes	yes	no		yes	no
no										
yes	yes	no								
yes	yes	yes	137.0	yes	no	yes	yes	no	yes	yes
yes	yes	yes	28.0	yes	yes	yes	yes	yes	yes	yes
yes	yes	yes		yes	yes	yes	yes	yes	yes	no
yes	yes	yes	84.0	yes	no	yes	yes	yes	yes	yes
yes	yes	yes	30.0	yes	yes	yes	yes	yes	no	no

Chapter 4

Reflection and Outlook

4.1 Conclusion

Looking at the outcome of our project, we can say that we achieved most of the basic goals we set ourselves. We built a working prototype of the system we imagined, although it only operates successfully in about 10% of the run-throughs. Nevertheless, there is still room for improvement. Currently, it only works under optimal conditions and requires a lot of maintenance. Still, we have solved a lot of problems and learned a lot, not only about engineering but also about working on such a project.
The biggest improvement for the current prototype would be to make the movement of the drone more precise. Possible future improvement would be to make the system available to more than one smartphone at once by having multiple drones in one table. Another desirable enhancement of the project would be to make wireless charging available for laptops. We encountered the interesting problem of locating a smartphone accurate to a few millimetres, for which we could not find an existing general solution.

4.2 Known Issues and Bugs

4.2.1 App

Security As mentioned earlier, we use unencrypted FTP. This may allow an attacker to steal our server credentials e.g. using packet sniffing. Moreover, an experienced hacker could probably extract the credentials directly from our binaries because they are hard-coded. This is an issue since our server does run as the root user. The server could therefore be completely hijacked once the password is stolen.

Hijacking the server would allow to overcome the network separation (WWW $\leftrightarrow$ WiFi) by installing a NAT service or using SSH to bridge a connection to the drone or the cable roller. To avoid this, the use of encrypted SFTP should be implemented. Furthermore, the FTP-server should run as a normal user without administrative privileges.

Internet Connection Required The app only works if the smartphone is connected to the internet. If the system is being deployed in a place where there is no cellular network available, a Wifi network is required for the process of localising the smartphone to operate correctly. A slow internet connection is an issue as well since the image may take up to a minute to arrive.

Lock Screen In subsection 2.3.3 we described that `HiddenCamera` can bypass the lock screen. This actually does not work as expected. Before the implementation we tested this and it worked, but in the final app it does not work anymore. We could not figure out the reason for this and we strongly believe that it should be possible without manipulating the operating system (rooting the device).

Empty Battery If the device battery is completely discharged, our app can obviously not run and thus the whole system fails. On a final product one should account for this by adding a fixed charging spot or a USB outlet that can be used in this case.

4.2.2 Image Recognition

Lighting Conditions The approach for the image recognition does not work without sufficient lighting on and inside the table. Enough contrast is needed for the algorithm to work properly.

Duration of Image Recognition Process As discussed in chapter 3, the duration of the image processing algorithm takes more than half a minute most of the times. The current algorithm is based on low level analysis of the image. A much faster algorithm could be achieved by using more advanced computer vision techniques.

Reliability The algorithm we implemented does not work every time. Minor errors that did not show up until now or could not be debugged still occur every now and then. Taking a second picture might resolve the problem because of different lighting conditions and new seed points but there are still errors that can not be resolved so easily.

Inaccuracy The position of the seed point within a lane is not analysed in the image recognition algorithm. Thus a small inaccuracy occurs. Implementing this localisation would solve this problem.

Restriction on QR Codes The localisation of the smartphone on the table is now solved by using detection and decoding of QR codes. Although it is a method that works well enough, it is not the most aesthetic.

4.2.3 Drone and Power Supply

Noise and Vibration The stepper motors used in the drone and cable roller produce a sound while turning and generate a faint vibration. The noise produced is loud enough to be heard even if ... by a plate (e.g. the table top).

Imprecision of the Drone The approach for the drone to drive to a certain point is dependent on values like the number of steps for one turn of the wheel. Due to an imprecision in these, there is the risk of some deviation between the actual arrival position and the calculated position of the drone[1]. We tried to counterfight this using the calibration of the drone but there is still a small deviation.

Power Inefficiency Several parts inside the drone, the cable roller and the Qi charging kit itself are not the most efficient. Since we only built a prototype, the cost of the components was usually more important than their efficiency. Therefore the power efficiency of the table leaves scope for improvement[2].

Delay at Starting Cable Roller The calculation for the motor of the cable roller sometimes takes longer than the calculation for the motor of the drone. This can cause the drone to move before the cable is rolled out.

4.3 Team Work and Reflection

In this section is a description of the mistakes and what was done well during the project. It is a reflection on the team work as welle as on the approaches made to accomplish the goal persued during this semester project.

[1]Compare to measurements in chapter 3

[2]See "Effectiveness regarding power consumption" in 4.4 for details

4.3.1 Difficulties Encountered

Search for Best Solution

Several times during this semester project we were looking for a solution to a certain problem and tried to realise the first idea that came to our minds. An example for this is the data connection between the drone and the server. First there was the idea of connecting the two via LAN cable. We thought without further investigation that it would probably be an easy solution. After having difficulties with the amount of cable this caused, we came up with the the idea of providing the connection via WiFi which is much easier to implement.

We learned that it is wise to first research as much as possible about the different approaches before trying something that might not work in the end. Searching for the best solution before actually implementing it is what should be done in a case like this.

Spent Time with Cumbersome Approaches

There were times when we encountered difficulties implementing particular approaches. We did not try a different approach but rather tried to find a way around the problem. Looking for a simpler approach in the first place would have been a better and less time-consuming way to handle the problem. We learned to look at the details of a problem before considering a solution to be easy without properly thinking it through.

Asking team members for help is very important. We lost a great amount of time figuring out problems in the image recognition because we did not ask each other for help sooner.

Unrealistic Goals

At the beginning of the project a rough estimation for the time management was made. Optimistic judgements of the time used for each step were made and in many cases there was no time considered for handling encountered difficulties. This led to the fact that some features (e.g. more than one drone on the table[3]) could not be realised. Especially for the image recognition algorithm, the power supply and for all the 3D printings we planned too optimistically.

[3] Compare to "More drones on one table" in 4.4

4.3.2 What was Handled Well

The communication inside the team was really good. Motivation was held up inside the group by comforting each other and there were no misunderstandings to make working difficult.

Frequent meetings and many brainstormings made the work efficient. Concerning time management, there was never a period when someone could not continue his or her work because of delay of another group member. What made the team members work efficiently together on this project is also that the strengths and weaknesses of everyone were considered before assigning the tasks.

4.4 Possible Future Enhancements

4.4.1 Improvements Based on our Implementation

After working one semester on the project there are still lots of things that can be improved. Most of them were not realized due to a lack of time. Our thoughts of how the project could be enhanced are listed here.

As displayed in Figure 3.1, we only managed to build a working prototype. The design and overall construction may be significantly improved.

Efficiency Regarding Power Consumption

To make the charging more efficient, there are several enhancements to reduce unnecessary power consumption.

- The motors that drive the wheels and the cable roll are not efficient. We chose them because of their accuracy and mainly because hey are cheap compared to others. Brushless DC motors with encoders would improve efficiency but are more expensive.
- The Qi standard itself consumes about 60% more power than charging with wire [34]. Wireless charging is also not the fastest way to charge a battery. Using an improved wireless charging standard would be a good way to raise the efficiency.
- The voltage regulator described in subsection 2.7.3 simply generates heat with the voltage that is too much for the desired output voltage. Clearly there is some potential here to reduce some unnecessary power consumption by using a switching regulator instead.

Bluetooth Equipment

In our prototype there is a bluetooth beacon for the smartphones to recognise the table. There would be a possibility to integrate the bluetooth on the Raspberry Pi that is the server. This would help to make the system more consistent.

Time Efficiency

Each time a phone is placed on the table it takes some time to send the picture to the computing unit, do the image recognition, calculate the location of the smartphone on the table, move the drone to the right location and do the fine search. This should be sped up. The possibility to do so lies in every step of the process.

Android App Improvements

Right now the application does only work correctly when the screen of the smartphone is not locked and the phone is not in standby mode. For a totally smooth process this little issue should be removed. Not only the operation of the application can be improved, but also the performance of the user interface.

Another extra feature that could be added to the app is the response from the charging table. The idea is that the user gets information about what happens while he is waiting for the phone to start charging. It would also be a good opportunity to be able to have some debugging information about what part of the process failed.

The reliability of the app is improvable by offering a WiFi access point next to the table. Like this the problem of missing internet connection to send the image could be solved permanently.

Improvements for the Drone

Probably the greatest weakness of the drone is its imprecision. Although it drives quite accurately to where it should, there is no feedback on how far it actually drove. This could be improved by using different motors. The precision is also limited by the fact that the drone turns on two wheels. Using a different kind of mechanism to turn the drone could change that.

Making the drone more agile, faster and reducing the noise could be accomplished by using different kinds of motors, preferably one with an

encoder[4] to easily control the movement of the drone. Understandably a motor like that would cost more than the ones used in this project.

More drones on one table

In the beginning of the project, the goal was to be able to serve more than one customer at once. The code for driving with more than one drone already exists, so the next logical step is to increase the amount of drones inside the table. The different cables running to the different drones and the power supply are an issue to be solved but certainly possible since the drones are able to drive over the cable with some difficulty and wobbling around.

4.4.2 Different Approaches

In the following are some approaches we did not pursue.

Linear Displacement Instead of drones that move on wheels there would be linear movable charging kits. The charging coil would need to be able to be moved up and down unless it crashes with other moving charging kits.
The problem here is that two coils would not be able to cross each other. Therefore charging a new smartphone would mean that all other phones stop charging in order to make one coil able to move. This would be rather irritating and annoying for the user.

Table Packed with Coils A simple way to make wireless phone charging available on an entire table is to integrate wireless charging stations lying next to each other inside the whole surface. They would be operated all at the same time.

Image Recognition without QR Codes The image recognition and localisation would work on any embedded picture. This picture could be something nice to look at to add some aesthetics to the table. To accomplish this, plenty of knowledge about picture processing is required. We did not persue this approach because there was not enough time to develop an image recognition algorithm of this kind.

No App Needed If there is a reasonable way to locate a smartphone very precise in range of millimetres without interaction with the phone in which the user is involved. For example, one could localise the

[4] Compare to the described motors in section 2.6.1

smartphone by analysing its electromagnetic radiation. This would make the table easier to use, as no application would be needed. Another approach without an application is to ask the user to knock on the table before he places his smartphone and locate the knocking by using microphones. Finally, a camera mounted above or beneath the table could be used to localise the smartphone.

Appendix A

Measurements of Qi Charging Kit

Table A.1: Increase distance between **glass** plate ($4.9\,\mathrm{mm}$). d_{table}: distance table to smartphone [mm], d_{coil}: distance coil to smartphone [mm], I_{phone}: current going into the phone [mA]

d_{table}	d_{coil}	I_{phone}	% of 500mA
0.0	4.9	0.49	0.98
0.5	5.4	0.49	0.98
1.0	5.9	0.42	0.84
1.4	6.3	0.44	0.88
2.0	6.9	0.43	0.86
2.5	7.4	0.38	0.76
3.0	7.9	0.36	0.72
3.4	8.3	0.41	0.82
4.0	8.9	0.32	0.64
4.5	9.4	0.31	0.62
5.0	9.9	0.44	0.88
5.5	10.4	0.00	0.00

Table A.2: Increase distance between **acryl glass** plate ($4.9\,\mathrm{mm}$). d_{table}: distance table to smartphone [mm], d_{coil}: distance coil to smartphone [mm], I_{phone}: current going into the phone [mA]

d_{table}	d_{coil}	I_{phone}	% of 500mA
0.0	4.9	0.48	0.96
0.5	5.4	0.42	0.84
1.0	5.9	0.42	0.84
1.5	6.4	0.46	0.92
2.0	6.9	0.38	0.76
2.5	7.4	0.33	0.66
3.0	7.9	0.44	0.88
3.5	8.4	0.40	0.80
4.0	8.9	0.37	0.74
4.5	9.4	0.40	0.80
5.0	9.9	0.35	0.70
5.5	10.4	0.00	0.00

Table A.3: Increase horizontal distance in x-direction (shorter side of receiver coil) on **acryl glass** plane ($4.9\,\mathrm{mm}$). x_{off}: offset [mm], I_{phone}: current going into the phone [mA]

x_{off}	I_{phone}	% of 500mA
0.00	0.57	0.97
1.00	0.57	0.97
2.00	0.57	0.97
3.00	0.55	0.93
4.00	0.51	0.86
5.00	0.48	0.81
6.00	0.45	0.76
7.00	0.48	0.81
8.00	0.45	0.76
10.00	0.44	0.75
15.00	0.00	0.00
20.00	0.00	0.00

Table A.4: Increase horizontal distance in y-direction (longer side of receiver coil) on **acryl glass** plane ($4.9\,\mathrm{mm}$). y_{off}: offset [mm], I_{phone}: current going into the phone [mA]

y_{off}	I_{phone}	% of 500mA
0.00	0.57	0.97
1.00	0.50	0.85
2.00	0.46	0.78
3.00	0.45	0.76
4.00	0.42	0.71
5.00	0.47	0.80
6.00	0.45	0.76
7.00	0.48	0.81
8.00	0.43	0.73
10.00	0.32	0.54
15.00	0.00	0.00
20.00	0.00	0.00

Table A.5: Increase horizontal distance in radial direction on **acryl glass** plane ($4.9\,\mathrm{mm}$). $(x, y)_{\mathrm{off}}$: offset [mm] (radius), I_{phone}: current going into the phone [mA]

$(x, y)_{\mathrm{off}}$	I_{phone}	% of 500mA
0.00	0.52	0.88
1.00	0.55	0.93
2.00	0.53	0.90
3.00	0.46	0.78
4.00	0.49	0.83
5.00	0.46	0.78
6.00	0.45	0.76
7.00	0.43	0.73
8.00	0.33	0.56
10.00	0.30	0.51
15.00	0.00	0.00
20.00	0.00	0.00

Table A.6: Increase horizontal distance in x-direction(shorter side of receiver coil) on **acryl glass** plane ($4.9\,\mathrm{mm}$). x_{off}: offset [mm], I_{phone}: current going into the phone [mA]

x_{off}	I_{phone}	% of 1A
-1.00	0.00	0.00
0.00	1.75	0.83
1.00	1.76	0.84
2.00	1.72	0.82
3.00	1.67	0.80
4.00	1.66	0.79
5.00	1.64	0.78
6.00	1.64	0.78
7.00	1.65	0.79
8.00	1.69	0.80
10.00	1.77	0.84
15.00	0.00	0.00
20.00	0.00	0.00

Table A.7: Increase horizontal distance in y-direction(shorter side of receiver coil) on **acryl glass** plane ($4.9\,\mathrm{mm}$). y_{off}: offset [mm], I_{phone}: current going into the phone [mA]

y_{off}	I_{phone}	% of 1A
-1.00	0.00	0.00
0.00	1.69	0.80
1.00	1.51	0.72
2.00	1.49	0.71
3.00	1.47	0.70
4.00	1.47	0.70
5.00	1.49	0.71
6.00	1.50	0.71
7.00	1.53	0.73
8.00	1.59	0.76
9.00	1.67	0.80
10.00	1.75	0.83
11.00	0.00	0.00

List of Figures

2.1 Diagram Showing the Whole System 4
2.2 Process Diagram Including all Software Components 5
2.3 Schematic of our Table 6
2.4 Process Diagram of the Android App 8
2.5 onSensorChanged Flowchart 9
2.6 Views of the App 11
2.7 Coil Offset 12
2.8 Communication between Server and Drone 16
2.9 Main Idea of Phone Localisation 20
2.10 Overall Image Recognition Process 21
2.11 QR Map Arranged on Table 22
2.12 Classes Implementing Phone Localisation 23
2.13 Important Geometry Objects 24
2.14 Comparision of Thresholding Methods 25
2.15 Seedpoint Search 26
2.16 Rotation Analysis 27
2.17 Lane Alongside the QR Code 28
2.18 Lane Opposite of the QR Code 29
2.19 Corners of QR Code 30
2.20 Cutting-out of the QR Code 31
2.21 Vector Conventions for Coordinate Transformation 32
2.22 Photo of the Drone 34
2.23 Illustration for Formula Derivation 39
2.24 Hardware Setup of Cable Roll 43
2.25 Placement of the Drone and Cable Roll 45
2.26 Special Cases of Rolling Algorithm 46
2.27 Power Supply Circuit 47

3.1 Testing Environment 48
3.2 Qi Measurements Vertical 49
3.3 Qi Measurements Horizontal 50

Listings

2.1 Parser Method 14
2.2 Code to Receive an Image 17
2.3 Code to Send an Image 18
2.4 Calculation of matrix A 33
2.5 goToTarget 37
2.6 moveDrone 38
2.7 stepperWorker 41

List of Tables

3.1 Precision Measurements of Drone 52
3.2 Overall Performance Test 53

A.1 Vertical Offset, Glass 62
A.2 Vertical Offset 63
A.3 Horizontal Offset in x-dir 63
A.4 Horizontal Offset in y-dir 64
A.5 Horizontal Offset in x, y-dir 64
A.6 Horizontal Offset in x-dir (Emmiter Moved) 65
A.7 Horizontal Offset in y-dir (Emmiter Moved) 65

Bibliography

[1] "Wireless Power Consortium," https://www.wirelesspowerconsortium.com, (Accessed: 30.05.2016).

[2] "Starbucks, Powermat Wireless Charging," http://www.starbucks.co.uk/coffeehouse/powermat, (Accessed: 02.06.2016).

[3] "Android Developers: Activities," https://developer.android.com/guide/components/activities.html, (Accessed: 18.04.2016).

[4] "Android Developers: Services," https://developer.android.com/guide/components/services.html, (Accessed: 18.04.2016).

[5] "GitHub: Estimote SDK for Android," https://github.com/Estimote/Android-SDK, (Accessed: 18.04.2016).

[6] "Android Developers: Intents and Intent Filters," https://developer.android.com/guide/components/intents-filters.html, (Accessed: 18.04.2016).

[7] "jible: SimpleFTP," http://www.jibble.org/simpleftp/, (Accessed: 03.05.16).

[8] "Sourceforge: Xming X Server for Windows," https://sourceforge.net/projects/xming/, (Accessed: 08.05.16).

[9] "ProFTPD," http://www.proftpd.org/, (Accessed: 08.05.16).

[10] "GitHub: pyiontify," https://github.com/seb-m/pyinotify, (Accessed: 03.05.16).

[11] "Wikipedia: hostapd," https://en.wikipedia.org/wiki/Hostapd, (Accessed: 08.05.15).

[12] "Wikipedia: dnsmasq," https://en.wikipedia.org/wiki/Dnsmasq, (Accessed: 08.05.16).

[13] "ETH Zurich Acceptable Use Policy for Telematics Resources (BOT)," https://www1.ethz.ch/id/documentation/rechtliches/BOTfinal-2005_EN.pdf, (Accessed: 31.05.2016).

[14] "Wikipedia: Base64," https://en.wikipedia.org/wiki/Base64, (Accessed: 13.05.2016).

[15] "Sourceforge: ZBar," http://zbar.sourceforge.net/, (Accessed: 24.05.2016).

[16] "Python Imaging Library (PIL)," http://www.pythonware.com/products/pil/, (Accessed: 25.05.2016).

[17] "OpenCV," http://docs.opencv.org/2.4.11/, (Accessed: 25.05.2016).

[18] "NumPy v1.10 Manual," http://docs.scipy.org/doc/numpy-1.10.1/index.html, (Accessed: 25.05.2016).

[19] S. Brahmbhatt, *Practical OpenCV*. Apress, 2013.

[20] "OpenCV: Miscellaneous Image Transformations," http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html, (Accessed: 25.05.2016).

[21] "SketchUp Make," https://www.sketchup.com/de/products/sketchup-make, (Accessed: 30.05.2016).

[22] "Wikipedia: Inductive Charging," https://en.wikipedia.org/wiki/Inductive_charging, (Accessed: 25.05.2016).

[23] "Adafruit Shop: Universal Qi Wireless Charging Transmitter," https://www.adafruit.com/products/2162, (Accessed: 31.05.2016), Qi charger kit used for project differs slightly.

[24] "Reichelt: Datenblatt C300," http://cdn-reichelt.de/documents/datenblatt/C300/DS_QSH2818.pdf, (Accessed: 17.05.16).

[25] “Wikipedia: Stepper motor,” https://en.wikipedia.org/wiki/Stepper_motor, (Accessed: 30.05.2016).

[26] “Wikipedia: Rotary encoder,” https://en.wikipedia.org/wiki/Rotary_encoder, (Accessed: 30.05.2016).

[27] “Edimax: EW-7811Un,” http://www.edimax-de.eu/edimax/merchandise/merchandise_detail/data/edimax/de/wireless_adapters_n150/ew-7811un/, (Accessed: 30.05.2016).

[28] “Adafruit DC and Stepper Motor HAT for Raspberry Pi,” https://cdn-learn.adafruit.com/downloads/pdf/adafruit-dc-and-stepper-motor-hat-for-raspberry-pi.pdf, (Accessed: 17.05.16).

[29] “Wikipedia: I^2C,” https://en.wikipedia.org/wiki/I%C2%B2C, (Accessed: 31.05.16).

[30] “Wikipedia: Pulse-width modulation,” https://en.wikipedia.org/wiki/Pulse-width_modulation, (Accessed: 25.05.2016).

[31] “Adafruit: All About Stepper Motors,” https://cdn-learn.adafruit.com/downloads/pdf/all-about-stepper-motors.pdf, (Accessed: 25.05.2016).

[32] “Instructables: Arduino Motor Shield Tutorial,” http://www.instructables.com/id/Arduino-Motor-Shield-Tutorial/step6/Stepper-Motor/, (Accessed: 25.05.2016).

[33] “Wikipedia: Low-dropout regulator,” https://en.wikipedia.org/wiki/Low-dropout_regulator, (Accessed: 18.05.16).

[34] “Stiftung Warentest: Induktives Ladegerät für Smartphones,” https://www.test.de/Induktives-Ladegeraet-fuer-Smartphones-Kissen-statt-Kabel-4535216-0/, (Accessed: 31.05.2016).