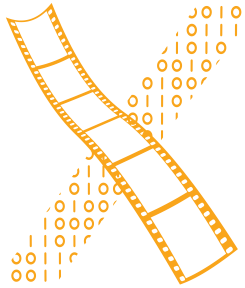

Media
Computing
Group

RWTH AACHEN UNIVERSITY

# *PERCbots: Actuated Tangibles on Capacitive Touch Screens*

Bachelor's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

*by*
*Florian Busch*

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Kjell Ivar Øvergård

Registration date: 25.09.2015
Submission date: 25.01.2016

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

*Aachen, October 2015*
*Florian Busch*

# Contents

# List of Figures

# Abstract

In this bachelor thesis we will engineer and build a PERCbot. A PERCbot is an actuated tangible, so an actuated haptic input device. We designed the PERCbot from the ground up to work on capacitive multitouch surfaces of arbitrary size and shape. By using small motors and a micro controller together with an active tangible [Voelker et al., 2015], we gain an actuated tangible which is not only aware of its position on the screen, but also able to make its way to an desired position on the screen all on its own.

Since actuated tangibles are not entirely new, we will compare the different approaches and see how the proposed design can overcome limitations. Most approaches have issues raised by the underlying technology stack. We will see that if we use a different approach on the touch sensing side we can overcome some limitations already by using PERC tangibles.

In our design the underlying technology connects the PERCbot to a computer wirelessly, which is driving the screen. This wireless connection is then used to transmit the current position and rotation to the PERCbot. If a destination update is send, the tangible is able to calculate distance and rotation. It will then use a loop of continuous destination updates to precisely reach the destination.

# Überblick

In dieser Bachelor Arbeit werden wir einen PERCbot entwickeln und konstruieren. Ein PERCbot is ein sich selbst bewegendes Tangible, also ein haptisch greifbares Eingabeobjekt. Der PERCbot ist von Grund auf für kapazitive Multitouch Tische beliebiger Größe und From ausgelegt. Wir werden kleine Motoren und einen Mikrokontroller in Komposition mit einem aktiven Tangible [Voelker et al., 2015] nutzen um ein sich selbst bewegendes Tangible zu erhalten, welches nicht nur weiß wo es sich befindet, sondern auch berechnen kann wie es zu einem gewünschten Ziel auf dem Tisch gelangt.

Die Idee des beweglichen/motorisierten Tangibles ist nicht neu, deshalb werden wir verschiedene Ansätze vergleichen und herrausarbeiten wie der neue Ansatz dieser Arbeit versucht einige bekannte Probleme zu lösen. Viele bekannte Ansätze hängen stark von der unterliegenden Technologie ab. Wenn wir einen anderen Ansatz bei dem Eingabegerät wählen, fallen dadruch, dass wir PERC Tangibles nutzen, direkt einige Hindernisse weg.

In unserem Ansatz verbindet sich der entworfene PERCbot kabellos mit einem Computer, welcher in der Lage ist den an ihn ageschlossenen Multitouch Tisch zu steuern. Wir nutzen nun die kabellose Verbindung um dem Tangible seine Position und Ausrichtung auf dem Multitouch Tisch mitzuteilen. Wenn ein neues Ziel an den PERCbot gesendet wird, berechnet dieses darauf hin die Distanz und den nötigen Weg. Durch ein periodisches Senden der aktuellen Koordinaten, kann der PERCbot dann präzise sein Ziel erreichen.

# Acknowledgements

I would like to thank Prof. Dr. Borchers for supervising my thesis and Prof. Dr. Øvergård for being my second examiner.

I am very thankful for Simon Völker who helped with constructive feedback during my work which helped me improve this thesis.

I thank my family for supporting and enabling me to write this thesis. I would like to thank Jan Thar for helping me with the electronic design and René Linden for assisting me with the demo.

Last but not least, I am very thankful for everyone at i10, providing help and great ideas which helped me during the writing of this thesis. Thank you, for providing such a pleasant environment.

# Conventions

Throughout this thesis we use the following conventions.

*Text conventions*

Although all the work in this thesis has been done by myself, I will use the first-person plural pronoun when referring to things that have been done.

Definitions of technical terms or short excursus are set off in colored boxes.

**EXCURSUS:**
Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
*Excursus*

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The whole thesis is written in American English.

Download links are set off in coloured boxes.

File: myFile[a]

[a] http://hci.rwth-aachen.de/tiki-download_file.php?fileId=file_number.file

"PERCbot", "Robot" or "Bot" refers to the actuated tangible, which is equipped with a PERC tangible, micro controller and motors.

# Chapter 1

# Introduction

In the last ten years we have seen more and more research in the field of Tangible User Interfaces (TUI).

**TANGIBLE USER INTERFACE:**
The term TUI or Tangible User Interface was first used by Ishii and Ullmer [1997] to describe an interface which makes use of physical objects (tangibles) to provide input and haptic feedback. The physical availability sets this interface apart upon others, since users can interact with it directly or by exploiting the affordances a tangible offers.

Definition: *Tangible User Interface*

Tangibles compensate the lack of multitouch interfaces being not able to provide haptic feedback. Interaction feels much more natural to a user and enables the user to interact without looking at the surface [Weiss et al., 2009]. Tangibles have widely been considered, sitting on top of an multitouch surface which is doing detection by using a optical system. One of the most commonly known early approaches go back until 2005 when Han [2005] presented the technology for optical multitouch sensing. Many approaches followed [Kaltenbrunner and Bencina, 2007, Schöning et al., 2008]. The low-cost and scalability of optical multitouch sensing systems drove many scientist to base their research on such systems.

Adding haptic feedback, by using tangibles

Shortcomings of optical systems

However, these system require a great amount of calibration and their reliability heavily depends on the ambient lighting. To overcome these problems newer approaches are based on capacitive touch sensing system. They are fast to setup and provide thoroughly precise touch detection.

Adding detection of tangibles

To make use of this system for tangibles however, we need to reconsider the patterns expressed towards the system. With PUCs [Voelker et al., 2013] proposed a way to make pattern detection possible on capacitive sensing systems. With PERCs [Voelker et al., 2015] the last shortcomings of the earlier approach were eliminated, yielding a robust and reliable technology for tangibles on capacitive screens.

These tangibles however, are static - meaning the do not express information in other ways than haptically. Furthermore PERC tangibles cannot be displaced on their own or by software.

Introducing actuation to tangibles

To enable this new channel of information and interaction, we introduce PERCbots: Actuated PERC tangibles which are able to move on their own. With PERCbots we are now able to express feedback by moving the tangible on top of the surface to a specific destination.

In addition to that we implemented a few extensions which make the robot less robotic. Furthermore we cautiously considered the hardware and software design, in a way that it is easy to extend and get started with.

Open platform for further extensions

Finally we will see some further improvements which make the PERCbot avoid collision and risky situations.

## 1.1 Thesis Structure

This thesis is structured as follows:

- **1—"Related work"** describes different approaches in the field of actuated tangibles and gives an overview about proposed actuation techniques.

- **2—"PERCbots"** introduces our actuated tangible by explaining the hardware and software in detail. Additionally we take a look at some of the design decisions we made.
- **3—"Summary and contributions"** concludes and wraps up the architecture of a PERCbot and the work we have done on it.
- **4—"Future Work"** gives an overview of future extensions and improvements regarding safety and multi-tangible setups.

# Chapter 2

# Related Work

## 2.1 How Information is Presented

Tangibles enable for haptic feedback

Tangible user interfaces integrate physical real world objects into the virtual, computational sphere. They should be able to receive input by manipulation and communicate changes to the user. Tangibles add the lack of haptic feedback when digital information is presented. Ishii and Ullmer [1997] did present the idea quite early. Tangibles can be seen like a window to the digital world. Nevertheless, interaction has to be as easy as expected and their functions have to be congruent to the interactions a tangible affords. If information gets represented by a tangible, it has to hold this information. Either the tangible itself encapsulates the information or the overlying system has to distinguish different tangibles, their meaning and bound information.

Information resides in the tangible from the user's point of view

It is key to keep the different applications in mind an user might want to use the tangible in. Even if the context changes, the information that a tangible encapsulates has to be accessible in a variety of areas. Ishii and Ullmer [1997] express this variability of a tangible with their example where marbles (tangibles) act as representations of calls on an answering machine. When a marble, a physical object, represent a missed call, a user could expect to call the person by taking and putting the tangible, the marble, on a phone. When designing how information is bound

to an object and presented, the different applications and affordances the objects provides have to be precisely considered.

## 2.2 Optical and Capacitive Sensing

### 2.2.1 Optical Systems

Setup of an optical sensing system

Optical sensing systems are based on a technology stack which is consists of one or more cameras, equipped with infrared filters, one or more projectors, multiple infrared light sources and a specially crafted surface, able to receive the projector's image. This surface consist of a common transparent material (e.g. acrylic glass) and a diffuser layer for the projected image. See figure 2.2 for an overview.

To understand the technology behind it, we will first recall the two major techniques that have been widely used for touch sensing in the past.

Definition: *FTIR*

**FTIR:**
FTIR or Frustrated Total Internal Reflection describes the phenomenon where light is reflected within a transparent material without ever existing it (without manipulation). Derived from the physical term of Total Internal Reflection, FTIR describes the ability of loosing the capability of total reflection at a spot where the material's surface is manipulated (e.g. by touch). Light will then exit at this very spot and illuminate the object (e.g. finger). An infrared camera on the other side of the surface is then able to see the fingers touching the surface. See figure 2.1.

**Figure 2.1:** Images from an infrared camera, demonstrating FTIR by showing a transparent surface with infrared light being fed in from the bottom. Touches on the back side of the surface are visible to the camera. The fingertips get illuminated since the reflection index of the surface is changed at this point.

**DI:**
DI or Diffused Illumination describes, like FTIR, a technique which can be used to provide touch or object sensing for multitouch tables. For DI an array of infrared lights illuminate (either from the front or rear) a transparent, touchable surface (e.g acrylic glass). Objects (with reflecting markers) or fingers will then reflect the infrared light which is emitted. These positions, where the markers or fingers get close to the surface, can then be see by an infrared camera. DI enables sensing objects and fingers before they come in contact with the surface.

Definition: *DI*

When we combine technologies like FTIR and DI and use them in a multitouch tabletop (see figure 2.2), we end up with a setup where we will have to precisely calibrate the light sources, software and one or more cameras and projectors. For simplicity we will now consider only one camera and projector being used.

Combining FTIR and DI

DI, when deployed beneath the surface, next to the camera and projector, will enforce us to cover most parts of the table's construction and cables with black material to minimize internal reflection. Any small change in the setup can force a recalibration of the camera and software. Sunlight

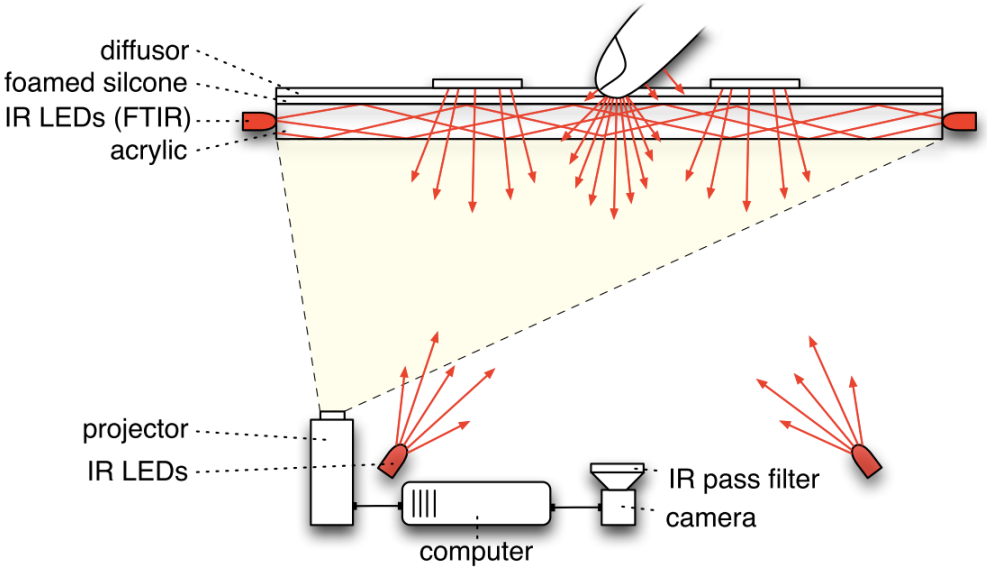


**Figure 2.2:** General setup of an optical multitouch sensing tabletop. Combining FTIR, DI, a projector, camera with infrared filter and infrared floodlights. Image taken from Weiß, Malte [2012]

from above the surface can also highly deteriorate the results for tracking, since sunlight covers all the ranges of visible and invisible, including infrared, light.

Shortcomings of optical systems

To overcome some of these problems we would use infrared filters and make heavy use of filters in software as well. By subtracting the current camera image from a precaptured background still, some internal reflections can already be no longer seen by the tracking algorithm.

Sensing tangibles on optical systems

We can then place objects (e.g. tangibles) on the tabletop. When we equip them with markers (some reflective tape), the tracking of those tangibles become possible. The emitted light is reflected by the markers and bounced back to the camera. If a tangible is moved across the surface it can happen that the intensity of the light, which gets reflected by the markers, varies.

Kaltenbrunner and Bencina [2007] present with their reacTIVision framework and the Reactable a common setup for an optical multitouch table. A projector, camera and com-

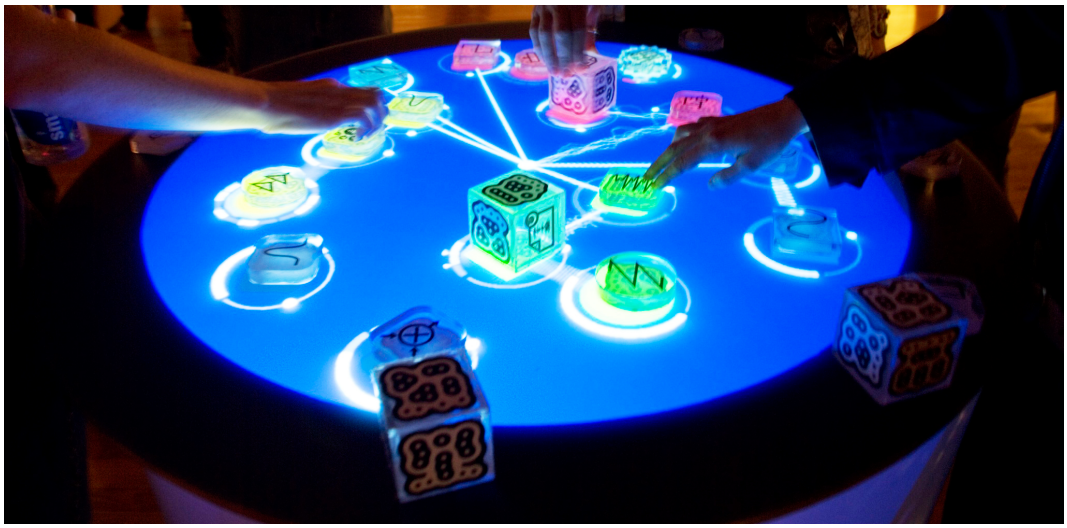

**Figure 2.3:** Reactable running reacTIVision at Altman Center (NYC). Image taken from Williams [2007]

puter form the basis. They then used infrared flood lights (build themselves) to illuminate the marker pattern they put beneath their tangibles. This represents the classical approach of Diffused Illumination (DI) we discussed earlier. In combination with additional infrared filters it was then possible to distinguish between noise, patterns and touch.

reacTIVision enables detection of visual markers

Figure 2.3 shows a Reactable running the reacTIVision framework, while users interact with it, creating music. Tangibles with marker pattern get recognized and allows the users to manipulate the created sound effects. The implemented marker set was re-engineered for better recognition and stability by Kaltenbrunner and Bencina [2007].

Kaltenbrunner and Bencina [2007] specially developed marker set was also used in an approach for an actuated tangible by Krzywinski et al. [2009]. They created a robot (see figure 2.5), which is able to move and sense orientation by a simple radar sensing system. Small motors steered by a micro controller can be actuated via a bluetooth connection. A simple AI (Artificial Intelligence) is able to move the tangible to a specific position on the tabletop by trial-and-error.

Actuated tangibles with visual markers

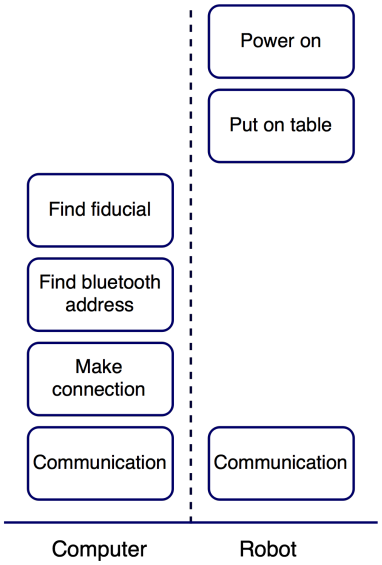


**Figure 2.4:** From top to bottom: Steps before communication between the computer and robot is established.

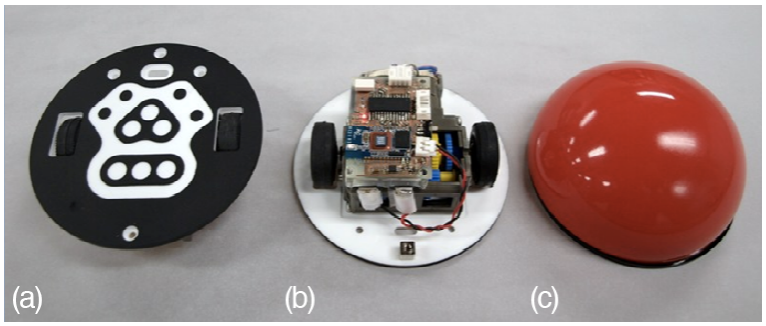


**Figure 2.5:** A robot from RoboTable with markers from Kaltenbrunner and Bencina [2007]. (a) bottom view, (b) top view, (c) top view with a shell. Image taken from Krzywinski et al. [2009]

In Addition to the optical setup Kaltenbrunner and Bencina [2007] proposed Krzywinski et al. [2009] added FTIR to the setup to reliably detect fingers and hands. The connection flow charts (figure 2.4) shows the steps which are necessary before a connection is established.

We can see in both approaches ([Kaltenbrunner and

Bencina, 2007] and [Krzywinski et al., 2009]) that optical system heavily rely on well placed infrared lights and decent ambient lighting, such that it does not interfere with the fiducial detection.

Infrared light as a basis for detection and tracking

A different approach can be seen with TouchBridge by Ladha et al. [2010]. Their approach is based upon Ganser et al. [2006] InfrActables where infrared LEDs, placed on top of the tangible, use different light pulses to transmit their unique id. This approach however is only applicable for a maximum of 8 tangibles. Ladha et al. [2010] replaced the camera by an infrared transceiver which is not only able to receive, but also transmit data via infrared light pulses.

Infrared LEDs at the top enable for data transmission

Due to this change, the proposed system can reliably distinguish between 16 different infrared markers, that are transmitted via light pulses.

All proposed designs heavily rely on infrared light. They have to be precisely calibrated which can later be forced to be redone if the ambient lighting changes. We think that calibrating the tabletop should be non-essential. Tangibles should be recognized when they make contact with the surface, independently of the ambient surrounding. This is why we like to consider an approach which cannot be troubled by highly changing attributes.

### 2.2.2 Capacitive Systems

Capacitive screens are not troubled by ambient light

A good starting point is given with capacitive touchscreens. Large capacitive multi-touch surface gain more and more availability and affordability [Microsoft, 2015]. In comparison to optical sensing techniques, capacitive sensing technology delivers a clearer and more precise image of touches. Additionally the technology is instant available when switched on.

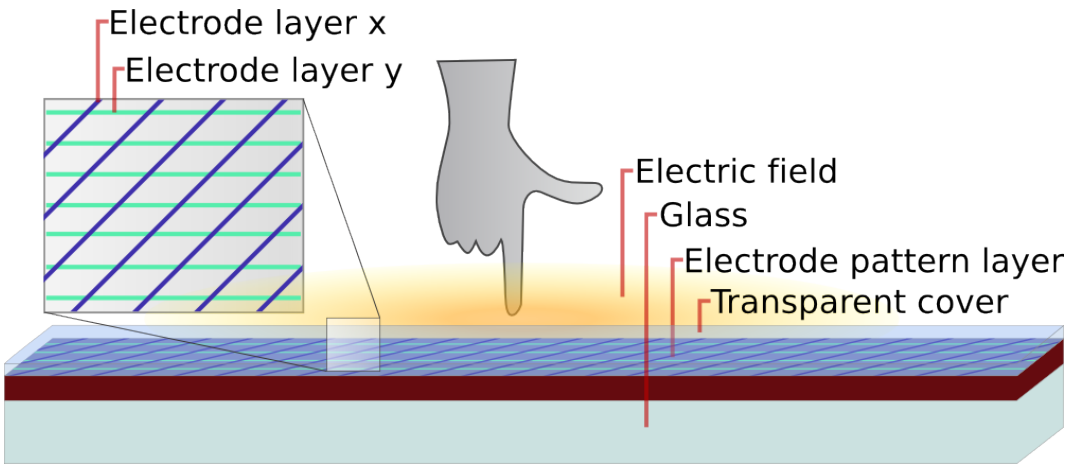


**Figure 2.6:** While touching the surface the user gets close to two of the electrode layer lines. By evaluating the electric capacity of every line (horizontally and vertically), the two lines closest to the touch point can be identified. Further processing enables an even more precise prediction.

Definition:
*Capacitive sensing*

**CAPACITIVE SENSING:**
Capacitive sensing is a technology in electrical engineering, often embedded in a surface, where an objects capacitance is used to determine the distance to it. Capacitive touchscreens use a grid of very thin electrically conductive wires. They run in parallel across the table horizontally and vertically. If an object with bigger capacitance than air is near to these wires, this can be measured. Since the surface consists of multiple lanes of wires, the exact location of the object can be calculated. To do this the sensing electronic measures every lane for its own by applying really small voltages to the wire. At the same time all the other lanes get grounded. If a drop in the applied voltage is detected, the electronic can be sure that there is an object above or in the area of this lane. By sequentially checking every line (horizontally and vertically) the $x$ and $y$ coordinate can be calculated. [Schöning et al., 2008] See figure 2.6

When we looked at optical sensing technologies we made constantly use of markers, which get illuminated by infrared light, to track and detect a tangible. When we used with capacitive touchscreens, those markers will not func-

tion. We will have to find a way to simulate a touch, as if it was generated by a human finger.

PUC tangibles [Voelker et al., 2013] are a proposed design which overcomes exactly this limitation. PUCs use a special electrically conductive bottom layer which enables them to be detected on unmodified capacitive screens.

Tricking the touchscreen into detecting a touch

By providing a conductive bridge between two wires of the touchscreen (see figure 2.6) the touchscreen sensing technology can be tricked into recognizing a touch. There are some orientations where the PUC tangible can be lost due to hardware limitations. If this happens the underlying frameworks is able to quickly recover the state once the tangible is recognized again.

Touches get filtered, creating new problems

Capacitive touchscreens use a lot of filtering to get rid of everything which is not similar to a contact with human skin. This does not only account for the shape of the created touch, but also for the time the touch is present. Touches, which do not move over a longer time, will eventually be filtered out, leaving an unresponsive spot. Only if the capacitive mass is removed, or the mass of the object grows bigger, the spot will become active for sensing again.

When we think of tangibles, which mostly lie on top of the surface, we run into the problem of them vanishing after a while on capacitive screens. To overcome this limitation Voelker et al. [2015] proposed a solution. Based upon PUCs they added electronic into the tangible making it able to sense if it is still on the table or not. The tangible consists of a battery, micro controller with bluetooth, field-sensor and a light sensor. To ensure contact to the surface a lead plate was added (see figure 2.7).

PERC tangibles are not troubled by filtering

With this design we can now overcome the problems described earlier. PERC tangibles can be robustly detected and movement is not a problem. Even if one or two of the three touch points vanish due to hardware limitations, a PERC tangible can stay recognized. If it recovers later, when the touch points are visible again, the tangible detection integrates these changes and the tangible is not lost. [Voelker et al., 2015]

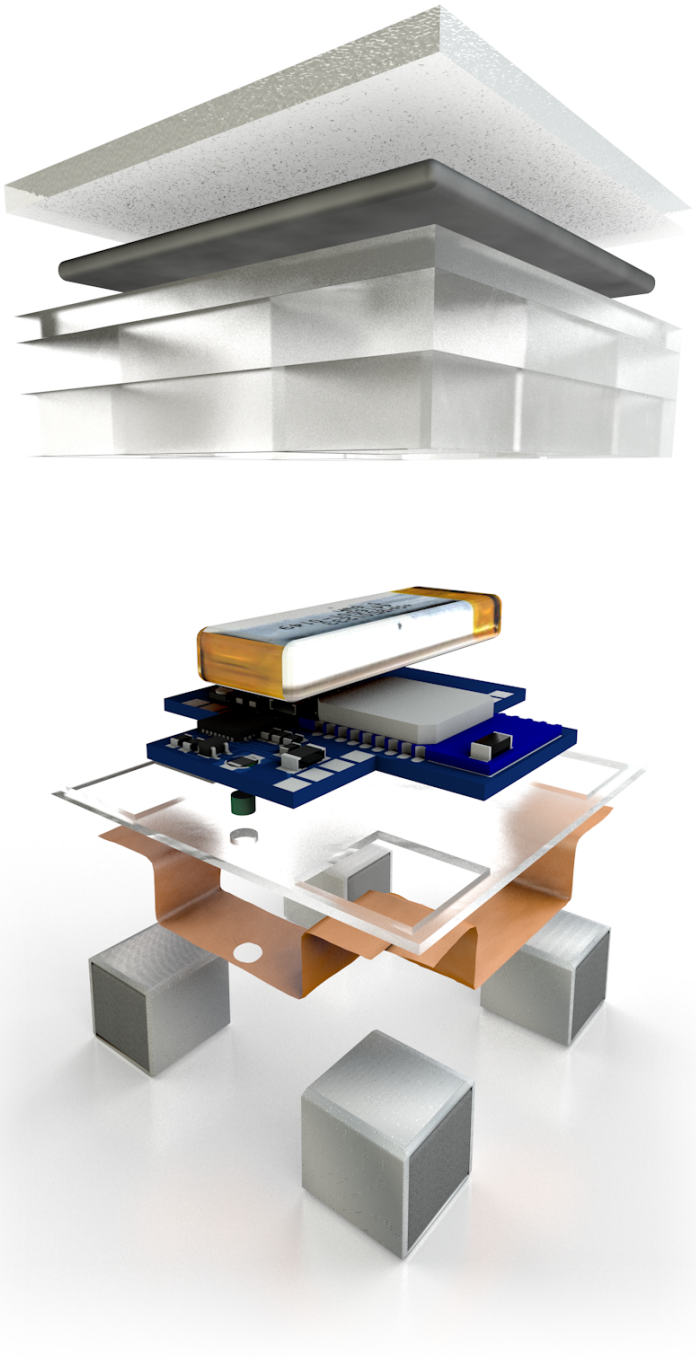

**Figure 2.7:** Explode view of a PERC tangible from bottom to top: Markers, copper tape, acrylic frame, lightsensor (green), microcontroller and bluetooth module, battery, acrylic frames and lead (for weight).

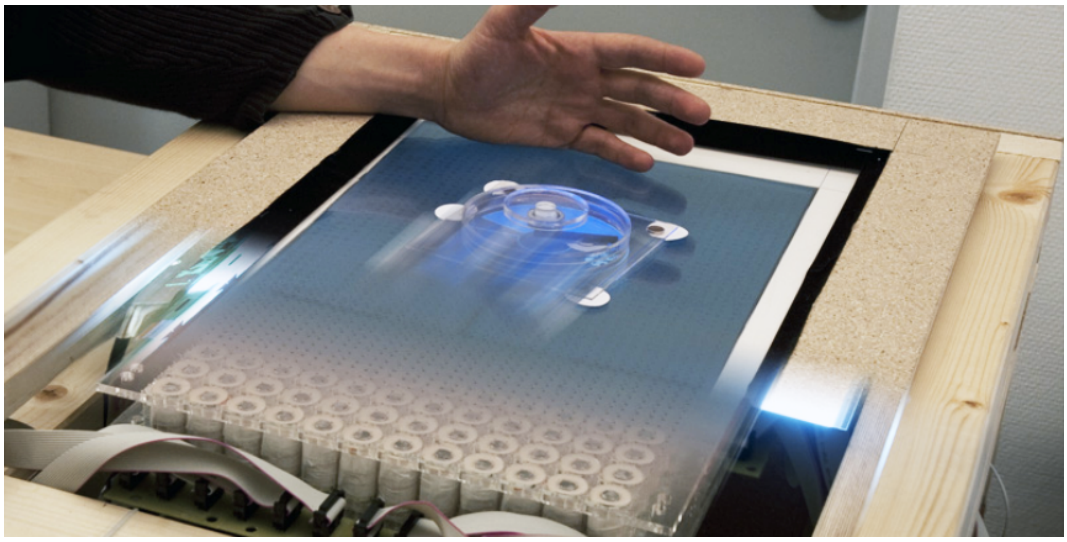

**Figure 2.8:** Madgets: External actuation of a tangible via electromagnetic impulses. Image taken from Weiss et al. [2010]

The robust tracking and detection make PERC tangibles a perfect starting point for our actuated tangible.

## 2.3 Actuation Techniques

Different approaches towards actuating a tangible

To design our actuated tangible we first consider a variety of different approaches. There are different ways to make a robot move or to actuate a tangible. We will compare external and internal actuation. External actuation refers to the tangible being moved by some force applied from the outside (like poking it), where internal actuation describes force being applied from the inside of the tangible (e.g. with motors and wheels).

### 2.3.1 External Actuation

#### Actuation by Magnetic Fields

Applying magnetic force

An approach where the actuation technique has been

moved apart from the tangible is Madgets by Weiss et al. [2010]. Here an array of electromagnets beneath the surface is used to actuate a tangible on top (see figure 2.8). The tangible is equipped with small magnets and white markers. These markers can be seen by cameras sitting beneath the surface, due to a special setup, where small fiber optics deliver the bounced light from these markers. This DSI sensing setup has a lower resolution, due to the limited number of fiber optics.

Size and complexity of the setup

When we consider the scalability, portability of such systems we see that they scale bad [Rosenfeld et al., 2004], and are not easily displaced Weiss et al. [2010]. Larger installations quickly grow in price. Nevertheless, we would create a large magnetic field, which can be dangerous to health for some users (consider people with pacemaker). In addition, we need a lot of power [Weiss et al., 2010] to move a tangible which could be actuated with way less power when internal actuation (e.g the use of motors) is taken into account [Nowacka et al., 2013, D'Ademo et al., 2011].

External actuation is therefore not capable to be used in combination with the tangible we want to built. It has to be instant available in various sizes and places. Internal actuation on the other hand can provide us with a more flexible approach.

### 2.3.2 Internal Actuation

#### Battery-less Hovering

Hovering tangibles

By using small piezoelectric air-blow actuators Miyaki et al. [2011] constructed a hovering batter-less robot. It charges contact-less through inductive charging. Using a 2D power sheet the robot can be charged. It then uses a Plenum Chamber System to efficiently distribute the airflow and lift the robot. Figure 2.9 shows the general design and the Plenum Chamber System.

Miyaki et al. [2011] show that the robot can hover for about 150 to 200 seconds, when charged for 20 seconds. This short

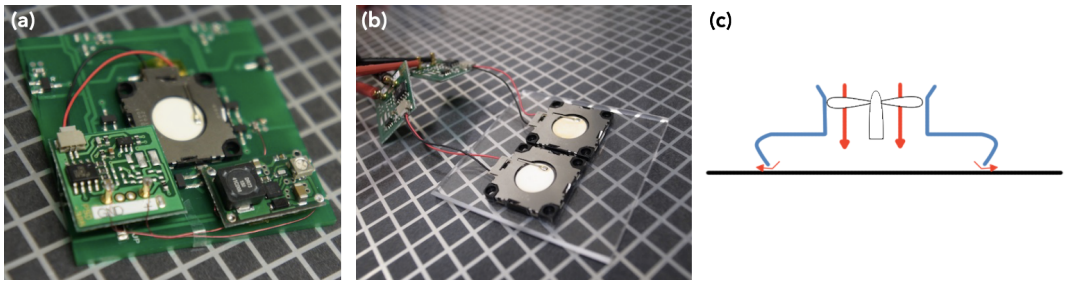


**Figure 2.9:** (a) Complete PCB with microblower and charging circuits. (b) To navigate in a specific direction two microblowers are required. (c) Plenum Chamber system: improves hovering abilities in comparison to Air-bearing and Flexible Skirt Systems (used in hovercrafts). Images taken from Miyaki et al. [2011]

window where an user can operate and interact with the device is far too little for constant interaction. We consider the robot to be permanently able to move and provide feedback to the user. The special setup with a 2D power sheet can also cause problems when applied to the chosen capacitive tabletop.

Issues when applied to capacitive screens

We encountered that additionally applied voltages in close proximity to a PERC tangible can cause issues [Voelker et al., 2015]. In addition to that, we require precise movement and tracking, which is only possible by having a tangible touch the surface.

Although we could think of modifying a PERC tangible to be detected when barely touching the surface, the small friction will nevertheless worsen the short time of possible operation. We will therefore continue looking at other approaches for actuated tangibles for our envisioned tangible.

### Actuation by Vibration Patterns

Actuation by vibration patters

Another way of internal actuation is shown by Nowacka et al. [2013] with TouchBugs, where small vibrator motors are used to move a small tangible across an interactive surface. Communication with the robot is done via Infrared Data Association (IrDA). Steering is done by applying different power levels to the left or right vibration motor. A

**Figure 2.10:** TouchBugs are built upon small brushes on which they move, controlled by vibration patterns. Image taken from Nowacka et al. [2013]

micro controller makes use of bottom mounted light sensors to read a color pattern from the interactive surface. Nowacka et al. [2013] define different color patterns for direction driven actuation.

The robot is similar to the design Miyaki et al. [2011] proposed. Due to the applied actuation technique the robot does only make very little contact with the underlying surface, making it hard to apply our selected PERC design.

Missing contact to the surface

#### Motor-Based Actuation

D'Ademo et al. [2011] showed a motor based actuation technique called eBug. Their robot is built of layers, each of them highly specialized in one function only. The mechanical layer for example will contain all the parts that are indispensable for moving the robot. Other layers may be used for different functions. They are exchangeable and thus the setup and features of the robot highly modifiable [D'Ademo et al., 2011]. Motor based actuation leads to fast

Exchangeable layer design

and precise movements. Due to the modular structure the robot can accomplish many tasks and repair and setup are fast and easy.

The eBug can communicate with a controller via bluetooth. The controller is able to send driving directions making the robot move. Furthermore, the controller can read the sensors which can be equipped to the robot (modular layer design). With this structure it can be controlled on interactive surfaces when the position is known.

Providing similar functionality like a PERCbot

Structure and electrical design gives a good starting point for our tangible. The module layer design makes it easy to evaluate new functions and maintain the robot. We will start by recreating a similar design and attaching markers to make the robot recognizable on a large tabletop.

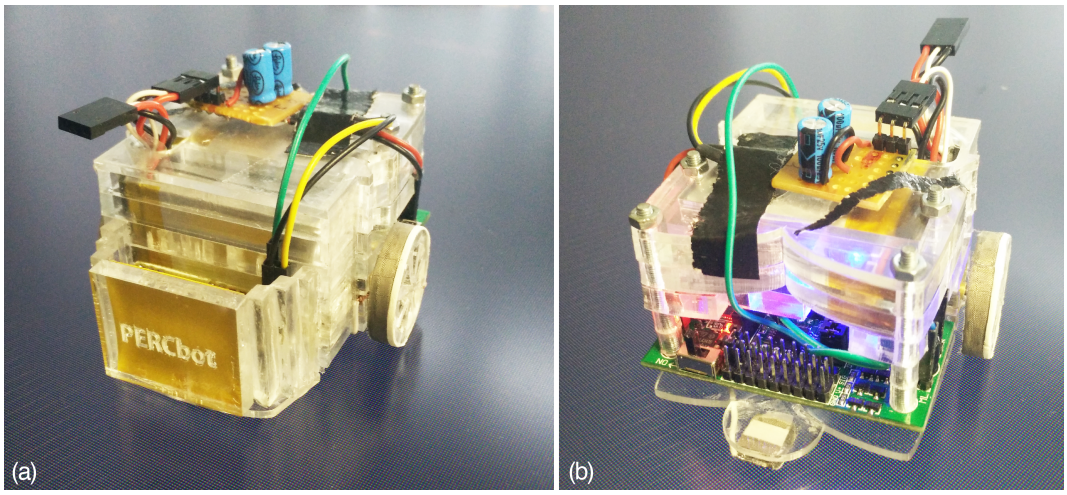


**Figure 3.1:** PERCbot: (a) Front view: At the front of the robot we mounted a PERC tangible. It connects to the voltage regulator and the micro controller. (b) Rear view: At the bottom part of the image we see one of the three markers, the other two are given by the conductive wheels. On top the voltage regulator connects to the power supply, the micro controller and the PERC tangible.

# Chapter 3

# PERCbots

## 3.1 Overview

The actuated tangible, the robot will from now on be referred to as PERCbot.

A video of the PERCbot in action is included in the archive, available for download (see B "Video and materials").

Hardware design of a PERCbot

A PERCbot is build from various components. The housing for the micro controller, batteries and motors was laser cut to match the specific shape of every component. The bottom layer creates the base for every other layer and the motors. At the backside of the robot, we can see the micro controller and the motor controller. They are available on top of one board, the commercially available Dagu Mini Driver Board [Dagu, 2015a]. It is fully compatible to the Arduino IDE and can control two motors, up to eight servos and has various extension headers.

We chose this board because it offers a motor controller, micro controller and valuable headers already assembled on one board. Furthermore it is compatible with the Arduino IDE (3.3.1 "IDE"), which make us able to get a working implementation relatively quickly and lower the barriers for future extension. The wheels are custom created to make use of PERC's [Voelker et al., 2015] technological approach for pattern detection on multitouch tabletop interfaces. See figure 3.5 and section 3.2.6 "Marker Creation".

Communication by using PERC's bluetooth

For the communication between the micro controller and the computer, which is connected to the tabletop, we use PERC's built-in bluetooth. The serial communication uses a special protocol we explain in 3.3.4 "Protocol for Communication". With the then obtained data, the micro controller will calculate the coordinates and the required moves (3.3.3 "Pathfinding").

## 3.2 Hardware

### 3.2.1 First Approaches

Choosing the correct motors

In our early approaches we concentrated first on the actuation part, namely the motors. We used standard DC motors, which spin fast but only provide a really small force at the motors shaft (see figure 3.2). The problem of these

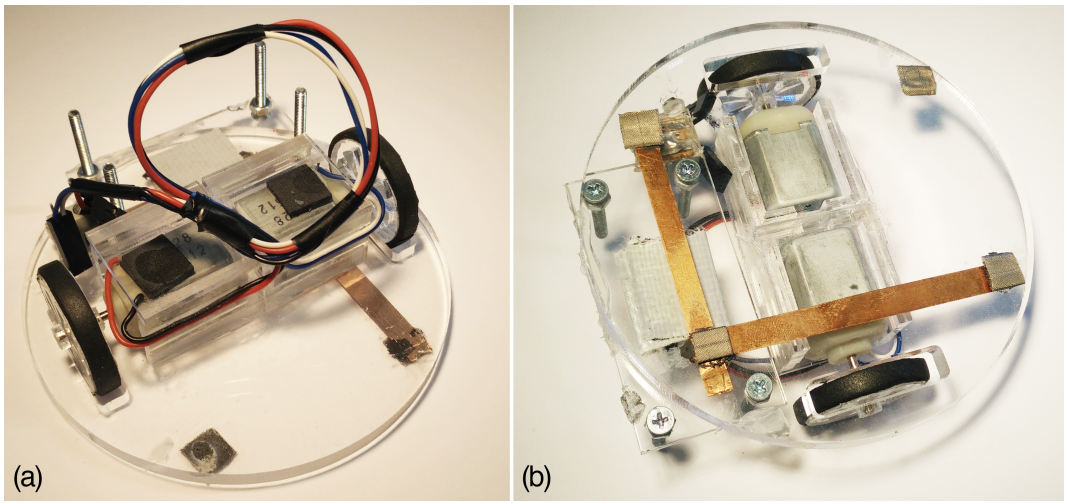


**Figure 3.2:** First hardware design: (a) Top view of motors with PUC pattern beneath. In the back the micro controller is about to be placed, which then drives the motors (b) Bottom view of markers used for detection on a capacitive surface

motors were, albeit cheap, their lack of a gearbox. Without it we had to put really small voltages on the motor, such that the robot does not overshoot the surface. Furthermore the force at the motor's shaft was too little, such that when we equipped the robot with only one battery, it was barely able to move. This lead us to use some different motor, that includes a gearbox (see 3.2.4 "Motors").

Early tests on functional, conductive markers

Our first approach towards a functional pattern design was closely based on the proposed design by Voelker et al. [2015] (see figure 3.2 (b)). When the robot moved quickly we ran into problems that some markers were not touching the surface anymore. Precise leveling of the wheel and marker height cause us to rethink this entire approach of pattern creation. We came up with a more advanced design towards pattern creating by integrating the markers into the wheels, making them conductive. This forced us to create some custom wheels for driving, we explain in 3.2.6 "Marker Creation".

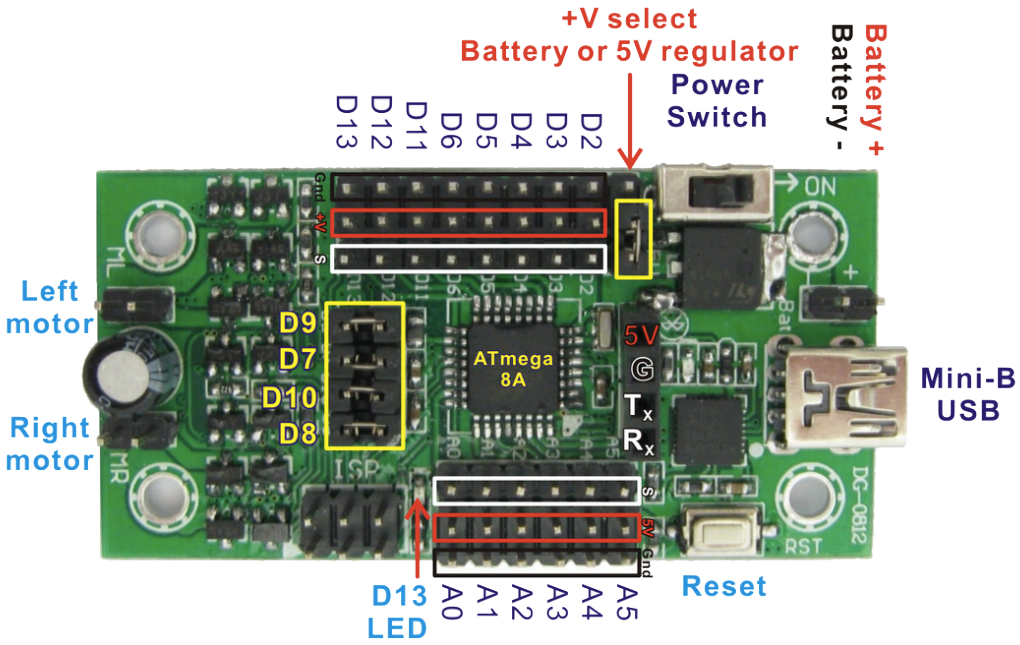


**Figure 3.3:** Dagu Mini Driver Board brings together all components we need to control the actuated tangible and gives addition space for future expansion (Image taken from Dagu [2015b])

### 3.2.2 Micro Controller

Features of the micro controller we used

We use the Dagu Arduino Mini Driver Board [Dagu, 2015b] (see figure 3.3) to do our computational pathfinding and actuation control. The board consists of mainly an ATmega8A with 16Mhz, Dual FET "H" Bridges which are capable of 2.5A peak current each, a serial header (for e.g. a bluetooth module), GPIO pins and headers for up to 8 servos. The speed of the attached DC motors can later be adjusted via Pulse-width modulation (PWM). The board is capable to function within the range of five to nine volt.

Limited flash might force alternative micro controller

When we wrote the program to do the pathfinding and navigation, we noticed how easily we could max out the available memory of the ATmega8A, especially when doing float operations. To overcome these limitations Dagu offers a more advanced version of the Dagu Mini Driver Board called Dagu Mini Driver MkII. It makes use of an ATmega328, offering 32KB instead of the ATmega8A's 8KB

flash. In our case this was not necessary but for future work worth a consideration.

Providing different voltage levels

The voltage regulator (see 3.2.3 “Power Supply”) provides three and five volt. The three volt rail is exclusively for the PERC tangible electronic, the five volt rail only for the micro controller. The micro controller will then provide five volt at the serial header. Since we are not interested in powering a module over the serial header, we only connect one wire between the micro controllers RX (receiving pin) and the PERC's TX (transmission pin) (see figure 3.1 (b) green cable).

### 3.2.3 Power Supply

Using Lipo rechargeable batteries to power the PERCbot

We use one Lipo, Lithium Polymer, rechargeable battery which provides 7.4 volt [1]. The current is fed into a voltage regulator (see figure 3.4 for a circuit diagram) which delivers constant five volt to ensure constant and battery voltage independent actuation speeds. The robots housing is able to carry up to two batteries of the same type. Only one battery is connected to the voltage regulator at the same time. This allows for fast battery changing.

### 3.2.4 Motors

Chosing a motor which enables further extension

To ensure maximum mobility and different movement velocities we wanted to equip the robot with motors, which reach the desired speed when only half of the maximum capable voltage is supplied. Therefore we chose to use the G50 motors from SOL-EXPERT-group [2] which reach about 350 rotations per minute when supplied with five volt.

Since all different models of this motor type provided by the manufacturer have the same size they can easily be

[1] Polymer Lithium Ion Battery 1000mAh 7.4V

[2] Datasheet: http://www.sol-expert-group.de/downloads/G50-100-150-298.pdf

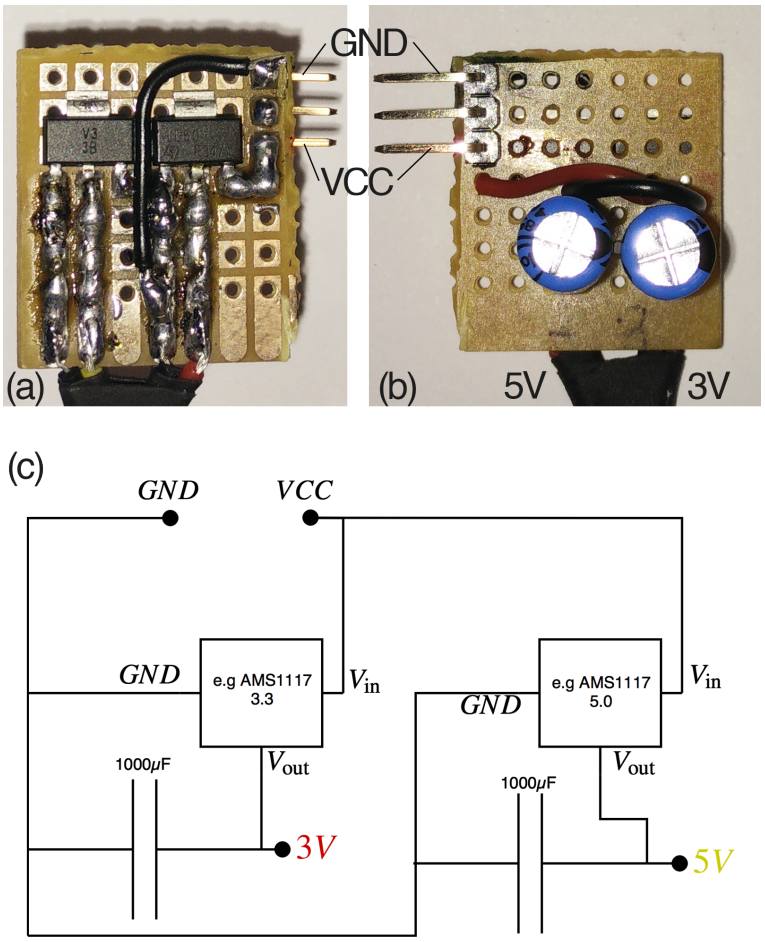


**Figure 3.4:** Voltage regulator: Provides five and three volt when supplied with 7.4 volt

swapped when the task of the robot changes where the currently incorporated motors are either turning too fast or too slow.

### 3.2.5 Physical Housing

Figure 3.1 shows the outer appearance of the robot PERCbot's housing is custom created to fit all components on at least space as possible, making the robot more mobile and agile. The vector graphics used to create the housing

are included in the archive, available for download (see B "Video and materials").

Housing design decisions

We placed the PERC tangible electronic in front of the robot to ensure equilibrium. At the backside of the robot we but the micro controller and the third marker. The other two markers are given by the wheels being fully conductive. They are then wired to the third marker to create a triangle of markers which can then be used to detect the PERCbot on a capacitive screen.

When assembling the PERCbot, we firstly put the acrylic glass layers on top of one another. Four screws are put in place to hold all layers in place. The screws can be removed if changes are to be made.

Layers that can easily be replaced

We used an approach similar to to the layer principle from D'Ademo et al. [2011]. Based on this approach we put functions in a discrete place, such that they can be easily swapped or repaired. Instead of putting exchangeable layers on top of another we put the things that accomplish one function next to each other. By doing so we can build a robot much smaller, since we do not waste space at a layer just to fit the design. Consider for example an layer whose function it is to only provide visual feedback. Equipping one LED would provide the function. Nevertheless is has to be as large as any other layer to be exchangeable and provide all the headers and connector the others layers do. This waists space, space we do not want to loose.

### 3.2.6 Marker Creation

Overcoming common sensing issues

Markers are simulating human interaction with a multitouch surface. Normally multitouch surfaces and screens are engineered to reliably detect human fingers or hands. Most of them try to filter out anythings apart from that. By applying PERC's technology [Voelker et al., 2015] we can overcome these limitations and create a set of markers that form a pattern that can reliably be detected by a multitouch screen. Markers have to be conductive to each another and towards the sensing surface.

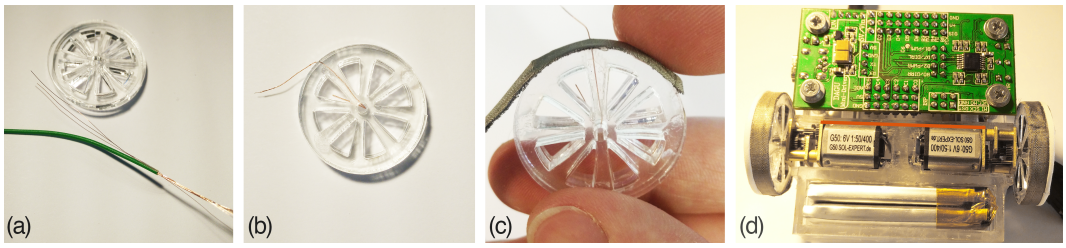


**Figure 3.5:** Creation of an electrically conductive wheel: (a) Splicing copper wire to obtain one thin wire with two laser cut wheels from acrylic glass. (b) Placing the wire in the motor's shaft and between the two acrylic layers while gluing each to the other. (c) Applying conductive fabric all around the wheels, making contact with the wire. (d) mounting each wheel to its motor shaft. Since both motors are soldered to a copper wire, the two wheels are now electrically connected.

Considering positions for markers

When we designed the PERCbot we had to consider where to put the markers. Either way, since we wanted to use wheels to actuate the robot we saw the risk that if the robot moved to sudden the markers could loose contact to the sensing surface. This would unnecessarily worsen the tracking and detection phase. We came up with the idea of integrating the markers into the wheels, making them conductive to another. But we could not simply solder a wire to them since their used to rotate. A sliding contact seemed wrong by design when considering deterioration.

Making the wheels act as markers

We came up with a design that uses the motor as if it were the wire. First we sliced one wire from an ordinary copper cable (figure 3.5 (a)) We made a part of the wheels surface conductive to the wheels hub (figure 3.5 (b)). Next, we wrapped around a conductive material [Voelker et al., 2015] such that it covers the entire surface of the wheel (figure 3.5 (c)). The hub is then again in contact with the motor's shaft and therefor with the gearbox. Since a small wire connects both motor's gearboxes such that as a consequence the wheels are electrically conductive to each another (figure 3.5 (d)).

Establishing the pattern

Another wire is soldered to the third marker (see figure 3.1 (b): silver marker). Now every marker is electrically conductive to each another and because of PERCbot's equilibrium and weight we can ensure that all three markers are in constant contact with the underlying surface.

Further adjustments to ensure wheels' grip

To improve the wheels' grip we later added a thin white stripe of cellular rubber at the outer area of each wheel. We also added a thin wire across the conductive surface which makes contact to the wheels hub even better and has no notable effect on the wheels grip.

## 3.3 Software

### 3.3.1 IDE

Setting up the IDE

To implement our path finding algorithm we used the Arduino IDE in version 1.6.3. We set the board type to "Arduino NG or older" and after installing CP210x USB to UART drivers, which are required for the Dagu Mini Driver Board to access the USB to Serial Converter. After that point the Dagu Mini Driver Board can be programmed like every other Arduino board. Nevertheless AVR Studio can also be used to develop the code which can later be pushed to the board.

Fast code development

We chose to stick to the Arduino IDE since it is cross-platform compatible and gives access to many convenient functions. To do simple string operations one could simply use the provided String library. To parse and decode the received input string, which follows the format of the defined protocol (see 3.3.4 "Protocol for Communication"), we made use of this library at first. Later when we implemented in-place rotation, we turned away from the String Library in favor of a character array. By this we used way less of the ATmega8A's valuable flash.

### 3.3.2 Program Flow

Refer to 3.6 for a visual representation. After power up, which is switching the micro controller on (see 3.3 "Power Switch") the micro controller configures each motor pin and the serial port. After successful boot it turns a blue

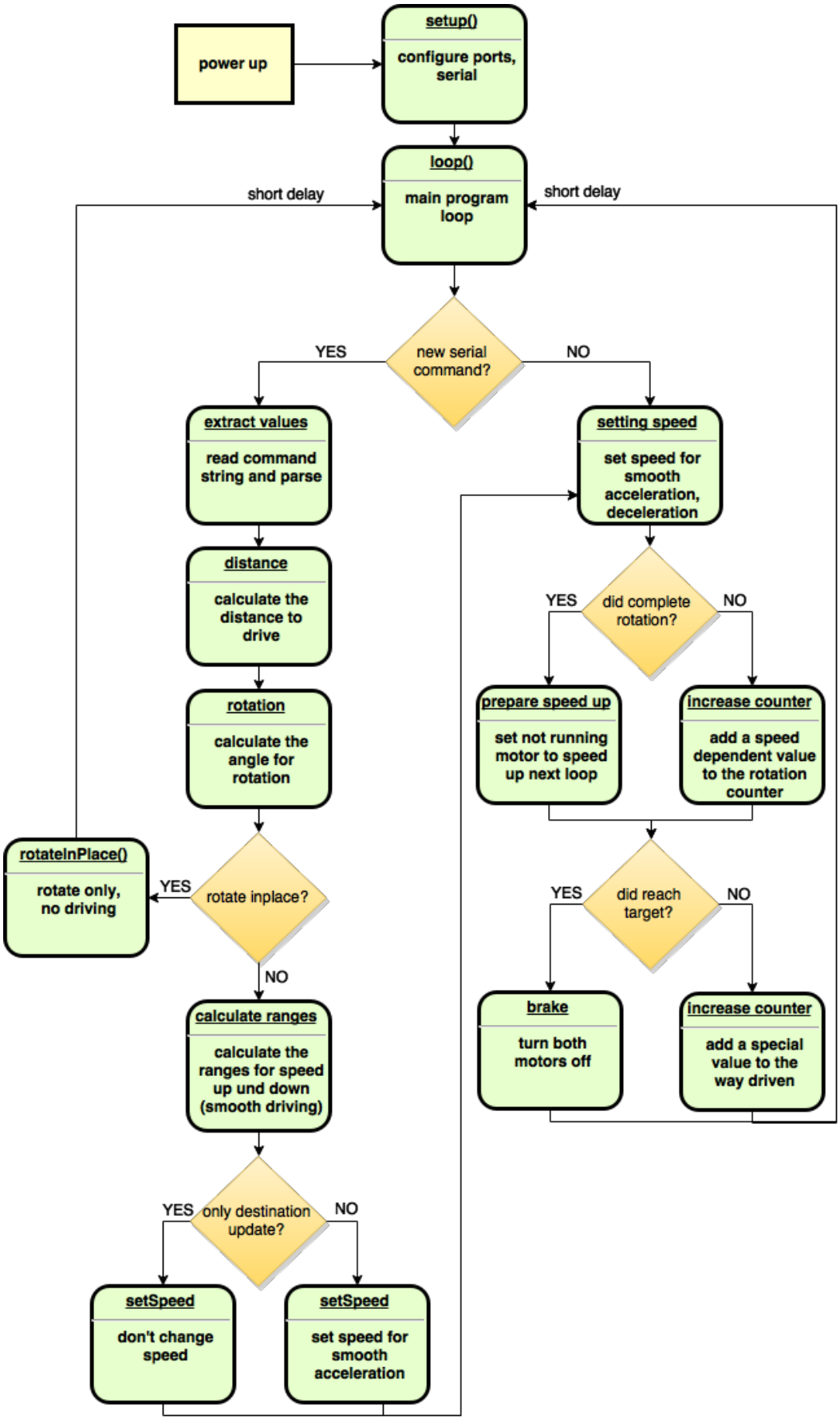


**Figure 3.6:** Diagram showing the update cycle and the general program flow.

led on. The LED comes pre-soldered with the Dagu Mini Driver Board.

Main loop is divided into two flows

After this initial setup the controller enters the main loop where all calculation and controlling takes place. The following flow can be divided into two partial flows. One, and we will refer to this as flow A (left tree in 3.6), which updates the variables the robot's moves depend on, the other one, flow B (right tree in 3.6), handles the main driving and speed controlling part.

Once a new string reaches the serial interface it gets checked for the defined format (see 3.3.4 "Protocol for Communication"). If it is valid, the micro controller will drop into flow A.

Flow A handles computational aspects

In Flow A the micro controller continues by extracting the necessary values from the received string. The distance and rotation angle to the destination get calculated and stored for further use.

If the calculated distance is too small for the robot to reach the desired end rotation (see sixth chunk 3.3.4 "Protocol for Communication") by driving, it will pause all current ongoing movements and perform a rotation in-place. This means the robot will not move forward, but rotate at the coordinate it currently is at. It will turn one wheel forward, the other one backwards to reach the end rotation. After this it will start again at the beginning of the main loop.

Deciding between in-place rotation and destination focused driving

If, in flow A, the distance is not too small for driving, the robot will not perform an in-place rotation. The micro controller will continue by calculate the sections for smooth actuation (see 3.3.3 "Smooth Actuation"). If flow A was reached while the robot was already performing a movement to a destination, the current motor speeds will remain untouched. If the robot has yet not been driving, the speeds will be set to the starting values for smooth actuation. The micro controller will then continue with flow B.

Flow B handles actuation specific tasks

Flow B handles driving specific tasks like smooth actuation and control of rotation dependent values. The first thing the micro controller does when it executes the commands

for flow B, is to check currently set actuation speeds. It will then decrease or increase the speed for each motor depending on in which section of smooth actuation the robot is currently in.

Next it will check if the desired rotation towards the destination has already be completed and increase a counter if not. Finally the way driven by the robot gets checked. If the destination got reached both motors will be turned off and the micro controller will return to the beginning of the main loop and wait for further commands via the serial port. If the robot has yet not reached the destination, a counter, expressing the way driven, will be increased.

### 3.3.3 Pathfinding

Code allows for rapid destination updates

The micro controller is able to calculate necessary movements on its own (see 3.3.2 "Program Flow"). We designed the software to be able to receive updates regarding the current position and the destination seamless and integrate the newly obtained information into the driving moves. The micro controller goes through the functions to calculate distance and orientation every time and then updates current values accordingly.

#### Distance

To calculate the robots distance to the destination position we use Pythagoras' theorem. After parsing the data string we receive from the PERC tangible, we end up with six values. We are interested in the current and destination position. The current position is given by the vector $\vec{c} = \begin{pmatrix} x_c \\ y_c \end{pmatrix}$, the destination position by the vector $\vec{d} = \begin{pmatrix} x_d \\ y_d \end{pmatrix}$ (figure 3.7 (a)) Next we subtract $\vec{c}$ from $\vec{d}$, getting $\vec{t}$ (figure 3.7 (b)):

$$\vec{t} = \begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} x_d \\ y_d \end{pmatrix} - \begin{pmatrix} x_c \\ y_c \end{pmatrix}.$$

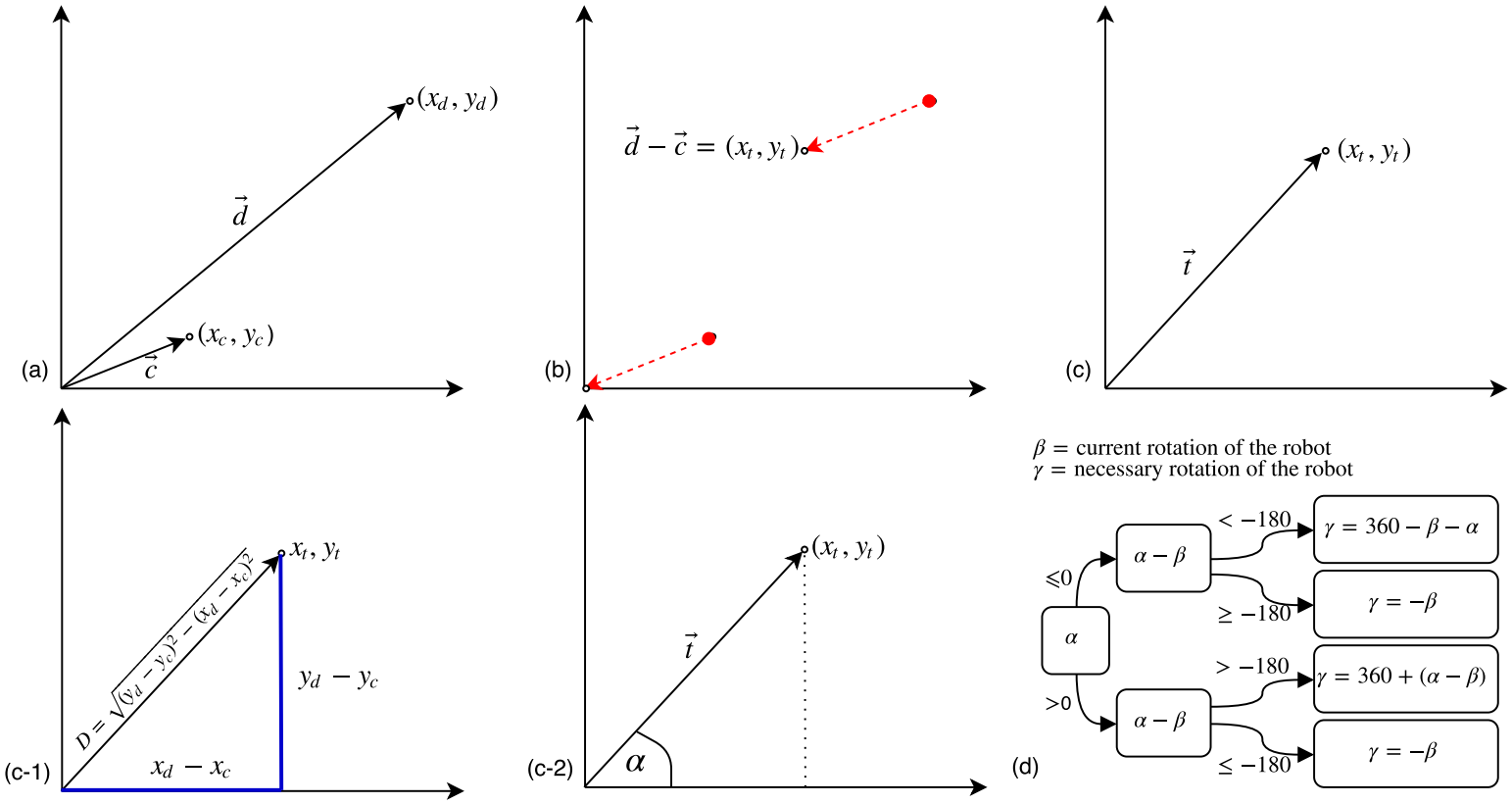


**Figure 3.7:** Steps to calculate distance and orientation: (a) $\vec{c}$ and $\vec{d}$ are given as current and destination position. (b) Moves $\vec{d}$ in negative $\vec{c}$ direction, $\vec{c}$ into the point of origin. (c) $\vec{t}$ now points to the new coordinate of $\vec{d}$. (c-1) $D$ represents the distance to the destination point by applying Pythagoras' theorem. (c-2) $\alpha$ represents the degree of change we need to apply to the robot to aim at the destination if the robot is currently facing 0 degree. (d) The diagram shows how we take the current orientation of the robot into account. (see A.2 "Utility Functions" file `utils.ino` line 28-39)

Distance calculation by applying Pythagoras' theorem

To get the distance we apply Pythagoras' theorem and get $D = \sqrt{x_t^2 + y_t^2}$ (figure 3.7 (c-1)). $D$ denotes the distance (in cm) to the target. This value controls the duration the robot moves in a specific direction. In the code we used similar variable names. This functionality is implemented in `core.ino` (A.1 "Core Program") at line 123-133.

### Orientation

To calculate the orientation we start similar to when we calculated the distance. Vector $\vec{c}$ denotes the current position, $\vec{d}$ the destination position. We calculate $\vec{t}$ like in 3.3.3 "Distance".

Calculating rotation towards the destination

We now calculate $\alpha$ (figure 3.7 (c-2)) via the arc tangent function. It returns the principal value of the arc tangent of $y_t/x_t$ in radiant. We convert that into degree by multiplying it with $(180/\pi)$

The obtained value does only represent the necessary rotation if the robot is currently facing 0 degree. If not, and this case is most likely, we need to adjust $\alpha$ by $\beta$, the current orientation. We will thereby obtain $\gamma$, the necessary rotation change. Nevertheless we need to ensure that we stick to the protocol defined 3.3.4 "Protocol for Communication". Figure 3.7 (d) shows the different cases we have to check when we take the current rotation $\beta$ into account to calculate the necessary rotation $\gamma$.

### Smooth Actuation

Smooth actuation ensures less robotic behavior

We implemented something we call smooth actuation, which refers to the acceleration the robot has while making its way to the destination position. When the micro controller has calculated the distance and required rotation, it does not simply put the motors on the maximum defined speed, but starts accelerating slowly until it has reached the maximum allowed velocity. When the robot comes closer and closer to the destination it does also decrease its speed.

Every time the robot gets a position update (3.3.4 “Protocol for Communication”) it divides the computed distance into three sections. The first marking the time to full velocity, the second the time the robot drives at full speed and the third the time the robot will decelerate. First and third take each 20% of the total distance, leaving 80% of the way where the robot moves at maximum defined speed (calculation see A.1 “Core Program” at line 176-181).

How smooth actuation works in code

To reach the full velocity the robot adds a rotation specific value (has to be less, if the robot rotates) to a counter if it is smaller than the maximum defined speed. The counter expressing the updated speed is then assigned to the corresponding motor. Each motor has its own independent speed. The code taking care of de/acceleration lives on line 204-234 A.1 “Core Program”.

We implemented the smooth actuation techniques not only to make the robot less robotic but also to ensure that it does not overshoot the destination and then needs to turn around for a new attempt.

### 3.3.4 Protocol for Communication

Communication with the robot uses a special protocol

If we transmit data to the bluetooth module, the micro controller expects the string it receives to be in a special format. The string has to consist of exactly 24 characters. These will be split up in six chunks of four characters each. The first three chunks express the current position and orientation (see 3.8 (a)). The last three chunks hold the destination and the angle the robot has do be in (end rotation) when it reached the destination.

Structure of a chunk

All chunks have the same format: The first character indicates if the number is negative. The following three characters express the number itself. Since coordinates must not be negative, all chunks, except for the third and sixth (which hold degree values), have a null set as the first character. We chose this design, because is simplifies the processing of the chunks in the micro controller and is faster as if the controller had to distinguish between different chunk

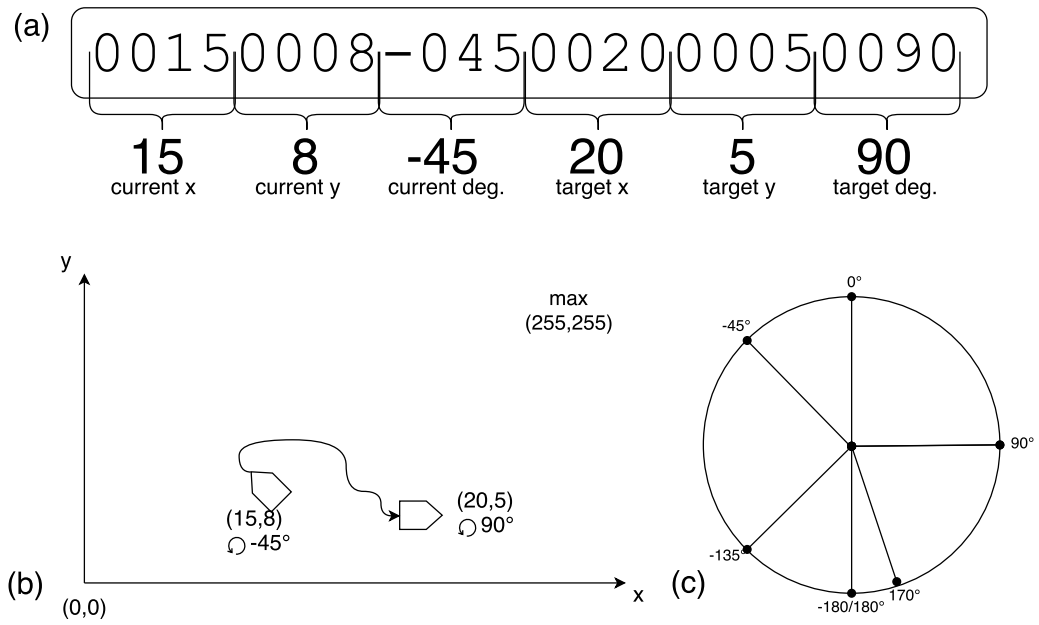


**Figure 3.8:** Protocol for communication with the robot: (a) Input string which resolves into six values. (b) Corresponding physical representation. (c) Angle definition

sizes.

Coordinate convention

Coordinates are always absolute to the lower left corner of the surface, which is at $x = 0$ and $y = 0$ (see 3.8 (c) for a visual representation). Coordinates must be not greater than 255, which is a limitation of the micro controller and degree values not greater than 180, which is defined by the protocol.

The first and second chunk mark the current $x$ and $y$ coordinate on the surface. Chunk four and five, using the same format, stand for the destination coordinates.

Expressing degree

The third chunk encodes the current orientation (see 3.8 (c)) in degrees. Following the convention the first character expresses with a "-" that the number is negative, "0" that the number is positive. The number formed by the following three characters needs to be less than 180. This encoding rules are the same for the sixth chunk, which expresses the orientation the robot has to be in when it completed the

driving task (end rotation). Degree values are limited to 180 because with this convention we can create much memory efficient code.

If the string passed to the micro controller does match the desired length of 24 characters, the micro controller will wait until enough characters are available. This is due to the serial transmission being slower than one main loop cycle of the controller. If the micro controller would not wait until enough characters are available, partially read strings, buffer under and overflows would the consequence.

## 3.4 Parameters and Conventions

The coordinate system we chose has its point of origin set in the lower left corner, stretching to the right with positive $x$, up with positive $y$ values. Coordinates translate into centimeter one to one. By this design and the limitations of the micro controller we can actuate a PERCbot within a rectangle of height and width of 255 cm.

Limitations in screen size

Since we control the tangible on top of a capacitive table top, this chosen micro controller and coordinate system limits us to a table top with 141 inch screen diagonal. This limitation is given by the micro controller and a controller with slightly more flash would overcome this issue (see 3.2.2 “Micro Controller” for an alternative micro controller).

Degree convention

The degree values have to be always smaller than 180. They range between 0 and 180. An additional flag is set if the value has to be interpreted as a negative number. This enables us to develop more memory efficient code. Thus the degree system of the PERCbot has been adjusted to make use of values between negative and positive 180 degree which can express any given angle.

Deployed motor model

We chose motors which are not fully maxed out when moving the PERCbot with desired speed. By this approach we can implement different driving speeds, depending for example on the surface or the users’ interaction. Smooth actuation, like explained earlier was mainly implemented to

ensure that the PERCbot does not overshoot a destination. Nevertheless it makes the robot act less robotic and thus convey a more human behavior.

Designing a faster movement towards the destination

In the beginning we thought about different ways of how the PERCbot could reach the destination. We mainly came up with two ways. The PERCbot could either correct its way when a destination update comes in by stopping both motors, performing an in-place rotation and continue driving in the corrected direction. It could also include the updated destination into its current movement. We decided to implement the second approach since this makes the robot reach its destination faster. This is because it does not stop every time an update is received. We call this technique curved driving. Test showed that this technique does worsen the precision unrecognizable, but minimizes the time the robot needs to reach its destination. The precision is nearly unchanged, compared to the driving with path correction by in-place rotation, because the robot gets constant destination upgrades.

Less robotic behavior by design

In addition to that, the chosen method of integrating updated coordinates into the current movement makes the robot seam less robotic toward human interacting with it. The video, included in the archive available for download (see B "Video and materials"), shows the chosen method in action. When we think of the different usage scenarios this can develop to a huge and important feature.

Finding the right update interval

Destination updates are sent by the computer attached to the table top and connected to the PERC's bluetooth interface. We found that sending an update every 1500 ms results in precise and smooth movements of the PERCbot. Updates more often (than every 500 ms) are to frequent for the incorporated micro controller, since this matches the minimum time the controller takes to calculate distance and orientation. Since it processes the serial commands first (see flow A 3.3.2 "Program Flow") it will than less often reach flow B (see 3.3.2 "Program Flow"), where the main driving and speed controlling takes place. The robot will seem unresponsive to commands. Updates less often (than every 1500 ms) will worsen precision such that the robot will take way longer to reach the desired destination.

Updates are sent frequently, because the distance, the robot moves, depends on many factors, like current battery voltage, size of wheels, wheel's friction and surface's attrition. They sometimes vary, even with a consistent setup.

Why frequent destination updates are indispensable

# Chapter 4

# Summary and Contributions

Based on the provided design of a PERC tangible [Voelker et al., 2015] we started creating our actuated tangible, a PERCbot, from the ground up.

Micro controller

We started thinking about a micro controller which suits our needs but still provides a platform which is well known and documented. We came up with the Dagu Mini Driver Board [Dagu, 2015a] which not only incorporates a well documented and widely used micro controller but also provides the necessary connectors for motors and communication right out of the box. This board provides us with two individually controllable motor connectors, a serial header for communication with the PERC tangible, eight servo pins for future use and an integrated LED.

Furthermore the board is compatible with the commonly known Arduino IDE and its framework, making development fast and easy.

Power supply

Next we considered the power supply, providing us with five and three volt. We attached a battery and got every circuit powered. The motors were chosen to offer some extension for future development. We created a custom housing which consist of layers each being swappable for repair or future developments.

Marker design

This left us with the markers for the PERCbot to be created like the markers for a PERC tangible. After some pitfalls we decided to make the wheels act as markers, each providing one touch point. They, and an additional third marker, form the triangular marker pattern of the PERCbot which makes recognition by the underlying surface possible.

After completing the hardware setup we brought our focus to the software, controlling the robot.

Arduino IDE

First we noticed that since we are using the Arduino IDE we get a lot of nice things for free, like the String class. To understand the algorithms implemented for pathfinding we first had to understand the program flow. We saw that the program consists of a main loop in which either new commands get processed or the tangible's movement is controlled.

Conventions and Parameters

We understood how distance and orientation calculation are performed and how the robot tries to ensure precision regarding the accurate destination finding. After this the general protocol for communicating with the PERCbot showed us how we can tell the PERCbot to move to a chosen position, while considering some conventions and ranges for parameters.

After this we had created a PERCbot, understood its hardware and software architecture, empowering us sending command to the PERCbot, making it actuate into the desired direction.

With this work future approaches in the field of actuated tangible can refer back to a functioning actuated tangible which is able to find a path on its own to a desired position. Even if we implemented this concept on top of a capacitive touchscreen, the technology could also be ported to an optical system, but then also facing all the earlier described problems.

When we compare our approach to a design that is based on an optical sensing system, we can say that our approach is more robust regarding tracking and detection - especially when in motion. Nevertheless our approach is based on

internal actuation, presenting us with the problem of power supply and consumption. Right now a PERCbot provides around three hours of interaction time, slightly varying due to different distances and orientations.

Power consumption

Nevertheless, the battery will be low at some point, a situation that would never have occurred, if external actuation had been used. Yet, regarding the relatively long interaction time, in comparison to other approaches [Miyaki et al., 2011] and much more precise movement ([Weiss et al., 2010], [Nowacka et al., 2013]), we consider internal actuation still a better choice for actuating a PERCbot.

# Chapter 5

# Future Work

## 5.1 Collision Avoidance

We can think of many extension to the here presented system for future work. One could equip the PERCbot at the top with a ring of infrared LEDs and receivers. Making them really dense to each another enables the PERCbot, if there are multiple on a surface, to detect each other. This can make the robot be aware of other robots and prevent colliding. Since we use a ring of infrared LEDs and receivers we can offer the robot a 360 degree view to the scene and other robots.

Sensing other PERCbots

When we think of collision avoidance, which is something we think has to be done in the physical world, the robot could also be enabled to sense obstacles of any kind. If we equip robots with infrared rings we could now tell if the obstacle is static or another robot. We can then tell the connected computer via bluetooth about this obstacle.

## 5.2 Dynamic Rerouting

Whenever a robot is predictably about to pass the position of the obstacle which was discovered earlier, the computer

Avoiding known obstacles

sending the commands to the PERC tangible can dynamically reroute the PERCbot to not collide with the obstacle. Such an obstacle would then consist of a position, size and lease time. The obstacle will get removed again if the lease time is up. The does not impair the path finding, because the obstacle, which's lease time is up, will be rediscovered by a PERCbot if it still there.

To enable the device, which is communicating the destination updates to the robots, being able to dynamically reroute robots, we would implement a spline object expressing the expected path the robot is about to take to arrive at the desired destination. This spline can then be computationally modified to prevent known objects on top of the surface. This spline object has to consist of as less points a possible, because the robot is able to make its way to a way point on its own. To many points would restrict the robots ability to chose the best appropriated way and thereby avoid collisions with other PERCbots.

Leaving the PERCbot still doing pathfinding on its own

If an obstacle is known and predictably sitting in the PERCbot's way (see figure 5.1 (a): red path), a new way around the obstacle is calculated by the computer (sending the destination updates) (see figure 5.1 blue way points). The closest way point will be send as the new destination to the robot. Once it reaches the destination, the next way point will be send. This will guide the PERCbot around the obstacle predictably creating the green path (b) of figure 5.1.

Rerouting with guidance

The next time the computer is about to send a destination update it can check beforehand if the predicted path (see figure 5.1 (a): red path) is interfering with known obstacles. It will then reroute the PERCbot to ensure it reaches the desired destination.

## 5.3 Damage Prevention

Setup

To avoid the robot from leaving the surface (and maybe dropping from high altitude) we could equip the PERCbot with a light sensor on top, which is able to receive light from 360 degree around the robot. We then put one laser

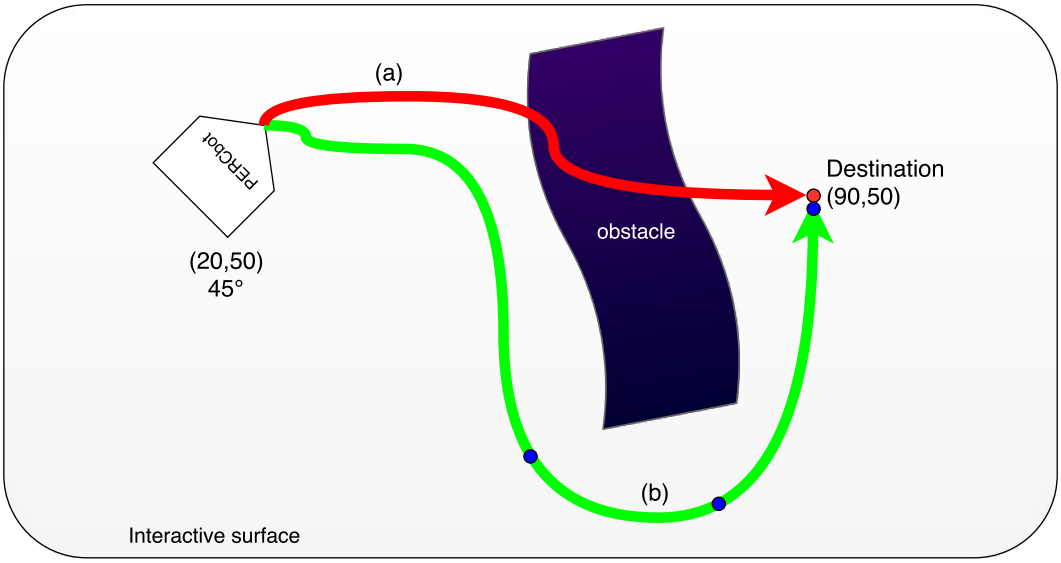


**Figure 5.1:** Dynamic rerouting if an obstacle has been discovered by a PERCbot and communicated to the computer sending the destination updates. Red (a): The expected path the PERCbot will follow, hitting the obstacle in the meantime will cause alerting the computer. Blue: calculated way points which are sent to the robot when approaching. Green (b): Resulting rerouted path, avoiding the obstacle.

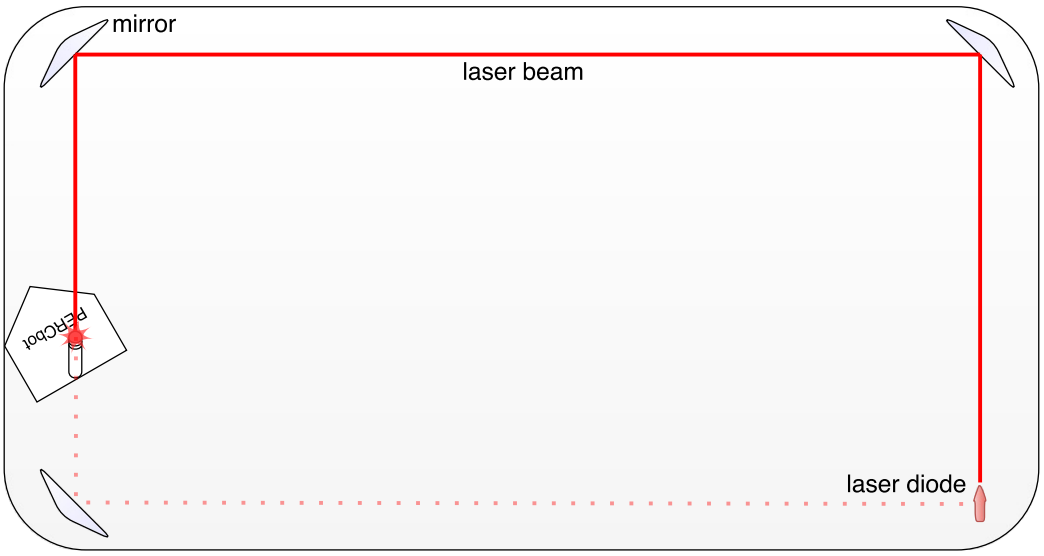


**Figure 5.2:** Invisible borders made with light

emitting diode (e.g. a common hand held laser pointer) at one edge of the surface pointing at some other edge. By placing small mirrors at the remaining three edges of the surface, the laser beam will be reflected such that it surrounds the surface (see figure 5.2 for a visual representation).

Stop when the laser beam hits the robot

If the light sensor on top of the PERCbot exposed to a high amount (pre-calibrated) of light (see figure 5.2 PERCbot crossing the laser beam), it instantly turns of the motors and backs up.

This approach, where we use an invisible border of light not only prevents the robot from damage but also benefits the freedom of interaction with the robot in comparison to ordinary physical boundaries.

Future work could also include working with a micro controller which offers more flash than the currently used one. This step will nevertheless be inevitable if one decides to implement the earlier described extensions.

# Appendix A

# Source Code

## A.1 Core Program

core.ino

```
////////////////////////////////////////
/// Hardware specific values go here ///
////////////////////////////////////////
#define LEFT_MOTOR_DIR_PIN 8
#define LEFT_MOTOR_PWM_PIN 10
#define RIGHT_MOTOR_DIR_PIN 7
#define RIGHT_MOTOR_PWM_PIN 9
#define maxSpeed 55 // (0-255)
// Define when to start driving and when to
// rotate inplace (in cm)
#define minDistanceToStartDriving 10
// Controls how much the robot rotates for
// a given angle (smaller value -> smaller rotation)
#define rotationIncrDivider 16
// Controls the minimum rotation angle at
// which to start inplace rotation
#define minAngleToStartInPlaceRotation 8
// Controls the offset the robot rotates
// when rotating inplace (size of wheels,
// grip, etc.) (more is more)
byte hardwareSpecificRotMult = 2;
// Controls the time while updating all values
// in the loop (more waittime -> longer moves)
#define waitInEveryLoop 15
////////////////////////////////////////


#define FWD HIGH
```

```
29 #define REV LOW
30
31 #define INVROTATION 0
32 #define DRIVING 1
33 #define TURNING 2
34 #define FLIPFLOP 3
35 // We got max of 8 bit here (so max 7 items!)
36
37 #define is(bit) bitRead(boolValues, bit)
38 #define set(bit) bitSet(boolValues, bit)
39 #define unset(bit) bitClear(boolValues, bit)
40
41 //In /Applications/Arduino.app/Contents/Java/
       hardware/arduino/avr/cores/arduino/Arduino.h:103
42 //e.g.#define bitRead(value, bit) (((value) >> (bit)
       ) & 0x01)
43
44 unsigned char speedStep;
45 unsigned int  speedUpUntil;
46 unsigned int  slowDownFrom;
47
48 byte boolValues;
49
50 // rotationToPerform will be between 0 and 180
51 // it uses invRotation to determine if e.g. 130 or
       -130
52 byte rotationDriven;
53 byte totalRotation;
54
55 unsigned char leftCurrentSpeed;
56 unsigned char rightCurrentSpeed;
57
58 byte cPosX, cPosY;
59 byte dPosX, dPosY;
60 int  cRot, dRot;
61 unsigned int totalDistance;
62 unsigned int distanceDriven;
63
64 //////////////
65 /// SETUP ///
66 //////////////
67 void setup() {
68   configurePorts();
69   Serial.begin(9600);     // opens serial port, sets
       data rate to 9600 bps
70
71   // Signal boot complete
72   digitalWrite( 13, HIGH );
73 }
74
75 ////////////
76 /// LOOP ///
77 ////////////
```

```
78 void loop() {
79
80   // Do we have new serial commands?
81   /// NEVER SEND MORE THAN 24 chars! - It will wrap
       the buffer ///
82   /// and then mess up the Positions - No Bug
                ///
83   /// Watch CR/Newline not appearing! they count.
                ///
84   // We can optimize here -> 4chars for every is not
       nesse.
85   digitalWrite( 13, LOW );
86   if (Serial.available() > 0) {
87     // Read until we have enough data
88     byte maxWait = 0;
89     while (Serial.available() < 24 && maxWait < 100)
       {
90       delay(5);
91       maxWait++;
92     }
93     char data [25]; // One more for null terminator
94     Serial.readBytes(data, 24);
95
96     // Emtpy the buffer, we only wanted the first 24
       chars
97     while (Serial.available()) {
98       Serial.read();
99     }
100    data[24] = 0; // Adding null terminator at the
       end
101    for (byte i = 0; i <= 5; i++) {
102      char subbuff [5];
103      memcpy( subbuff, &data[i*4], 4 );
104      subbuff[4] = 0;
105      switch (i) {
106        case 0: cPosX = atoi(subbuff);
107          break;
108        case 1: cPosY =  atoi(subbuff);;
109          break;
110        case 2: cRot =  atoi(subbuff);;
111          break;
112        case 3: dPosX =  atoi(subbuff);;
113          break;
114        case 4: dPosY =  atoi(subbuff);;
115          break;
116        case 5: dRot =  atoi(subbuff);;
117          break;
118        default: break;
119      }
120    }
121    free(data);
122
123    /////////////////
```

```
124     /// Distance ///
125     ////////////////
126     // calculate the distance from current to
        destination
127
128     unsigned int xSq = (unsigned int)sq(dPosX -
        cPosX);
129     unsigned int ySq = (unsigned int)sq(dPosY -
        cPosY);
130
131     totalDistance  = (unsigned int)sqrt( (double)xSq
         + (double)ySq );
132     totalDistance = totalDistance * 2;
133     distanceDriven = 0;
134
135     ///////////////////////
136     /// Rotation/angle ///
137     ///////////////////////
138     // calculate the offset in degrees from current
        ratation (cRot) to the destination point
139     // clockwise is +
140     int angle = calcRot( cRot, cPosX, cPosY, dPosX,
        dPosY );
141     // match into smaller variable
142     // angle will be always <= 180
143     totalRotation = abs(angle);
144     rotationDriven = 0;
145     (angle < 0) ? set(INVROTATION) : unset(
        INVROTATION);
146
147     /// Checking distance threshold for rotation in
        place ///
148     if (totalDistance < minDistanceToStartDriving) {
149       // So the distance we would drive is to small
150       // We will rotate in place, setting
        totalDistance to 0
151       totalDistance = 0;
152
153       // This is the last call before we drop back
        in the driving loop
154       // (which will be doing nothing because we set
         totalDistance to 0)
155       // This snippet is taken from the utils file
        where the totalRotation
156       // is calculated
157       if ( dRot <= 0 ) {
158         if ( dRot - cRot < -180 )
159           dRot = 360 - cRot - dRot;
160         else
161           dRot -= cRot;
162       } else {
163         //degVector > 0
164         if ( dRot - cRot > 180 )
```

```
165             dRot = -360 + (dRot - cRot);
166           else
167             dRot -= cRot;
168         }
169         // We misuse dRot here, but that is okey, since if we use it
170         // again it will have gotten an update already
171         if ( abs(dRot) >= minAngleToStartInPlaceRotation ) {
172           rotateInPlace(dRot, hardwareSpecificRotMult);
173         }
174
175       } else {
176         // We get here, we need to drive some way
177         /// Calc the 3 speedUp/maxSpeed/slowDown areas respective to distance ///
178         //[ a:20% |    b:80%     | c:c==a ] % of distance
179         speedUpUntil = totalDistance * 0.2;
180         // Only works for a==c
181         slowDownFrom = totalDistance - speedUpUntil;
182
183         /// Checking if driving and then using current speed instead starting at 1 ///
184         speedStep = 1;
185
186         if ( is(INVROTATION) ) {
187           (is(DRIVING)) ? goLeft(rightCurrentSpeed) : goLeft(1);
188         } else {
189           (is(DRIVING)) ? goRight(leftCurrentSpeed) : goRight(1);
190         }
191
192         set(TURNING);
193         set(DRIVING);
194
195       } // else totalDistance < minDistanceToStartDriving
196
197     } // if serial.available()
198
199     digitalWrite( 13, HIGH );
200     // There was an target update some time ago
201     // We are going fwd straight or still in a curve
202
203
204     ////////////////////////////
205     ///     Speed control    ////
206     ////////////////////////////
207     // Check where we are
```

```
208   if (distanceDriven <= speedUpUntil) {
209     //Serial.println("#|-|-");
210     // We are starting and speeding up
211     // We are here: [######|------|------]
212     if (leftCurrentSpeed + speedStep < maxSpeed &&
      leftCurrentSpeed > 0) {
213       leftCurrentSpeed += speedStep;
214     }
215     if (rightCurrentSpeed + speedStep < maxSpeed &&
      rightCurrentSpeed > 0) {
216       rightCurrentSpeed += speedStep;
217     }
218   } else if (distanceDriven >= slowDownFrom) {
219     //Serial.println("-|-|#");
220     // We are reaching the target, slow down slowly
221     // We are here: [------|------|######]
222     if (leftCurrentSpeed - speedStep >= 0) {
223       leftCurrentSpeed -= speedStep;
224     }
225     if (rightCurrentSpeed - speedStep >= 0) {
226       rightCurrentSpeed -= speedStep;
227     }
228   } else {
229     //Serial.println("-|#|-");
230     // We should be going straight with max speed
      defined
231     // We are here: [------|######|------]
232     if ( leftCurrentSpeed > 0 ) setLeftSpeed(
      maxSpeed );
233     if ( rightCurrentSpeed > 0 ) setRightSpeed(
      maxSpeed);
234   }
235
236
237   // Make the calculated speeds live
238   setGlobalSpeed( leftCurrentSpeed,
      rightCurrentSpeed );
239   ////////////////////////////////////////
240   /// Check if go straight or in a curve ///
241   ////////////////////////////////////////
242   if ( rotationDriven >= totalRotation ) {
243     if (is(TURNING)) {
244       // We get here, we are now in the correct
      direction, lets go fwd
245       // +1 is important, because we will auto-brake
       if speed is == 0
246       if (leftCurrentSpeed  == 0) {
247         setLeftSpeed( 1 );
248       }
249       if (rightCurrentSpeed == 0) {
250         setRightSpeed( 1 );
251       }
252
```

```
253         goFwd( leftCurrentSpeed, rightCurrentSpeed );
254         unset(TURNING);
255       }
256     } else {
257       // Still need rotation to look at the target
258       // Only one, left XOR right can be != 0 since we
      're rotating
259       char incr = (maxSpeed /2  + leftCurrentSpeed +
      rightCurrentSpeed) / rotationIncrDivider;
260       (rotationDriven + incr >= totalRotation) ?
      rotationDriven = totalRotation : rotationDriven
      += incr;
261     }
262
263
264     ////////////////////////////////
265     /// Did we reach the target ? ///
266     ////////////////////////////////
267     if ( distanceDriven >= totalDistance ) {
268         // We reached the target, stop and check
      rotation
269         doBrake();
270         // Finally we need to set our driving finished
       flag
271         unset(DRIVING);
272
273     } else {
274         // Update the way we already drove
275         if ( is(TURNING) ) {
276           // If we are rotating we only take every
      second loop and add
277           // to the distanceDriven, because we are
      going way slower fwd
278           if ( is(FLIPFLOP) ) {
279             distanceDriven++;
280             unset(FLIPFLOP);
281           } else {
282             set(FLIPFLOP);
283           }
284         } else {
285           distanceDriven++;
286         }
287
288     }
289     delay(waitInEveryLoop);
290
291 }
```

## A.2 Utility Functions

utils.ino

```
void configurePorts()
{
  pinMode( LEFT_MOTOR_DIR_PIN, OUTPUT );
  pinMode( LEFT_MOTOR_PWM_PIN, OUTPUT );
  pinMode( RIGHT_MOTOR_DIR_PIN, OUTPUT );
  pinMode( RIGHT_MOTOR_PWM_PIN, OUTPUT );
  pinMode( 13, OUTPUT );
  digitalWrite( LEFT_MOTOR_DIR_PIN, FWD );
  digitalWrite( RIGHT_MOTOR_DIR_PIN, FWD );
}


int radiantToDegree(float radians)
{
  return radians * (180 / PI);
}

////////////////////////////////
/// returns -180 <= x <= 180
////////////////////////////////
int calcRot(int cRot, int ax, int ay, int bx, int by
    )
{
 int x = bx-ax;
 int y = by-ay;

 // * (180 / PI) will get us from radians to degrees
 int degVector = vectorAngle( x, y ) * (180 / PI);
 if ( degVector <= 0 ) {
   if ( degVector - cRot < -180 )
     degVector = 360 - cRot - degVector;
   else
    degVector -= cRot;
 } else {
   //degVector > 0
   if ( degVector - cRot > 180 )
     degVector = -360 + (degVector - cRot);
   else
    degVector -= cRot;
 }
 return degVector;
}

float vectorAngle(int x, int y)
{
  return atan2f(x, y);
}
```

## A.3 Motor Driver

low_motor.ino

```
void setGlobalSpeed( byte leftPwmSpeed, byte
    rightPwmSpeed )
{
    setLeftSpeed( leftPwmSpeed );
    setRightSpeed( rightPwmSpeed );
}

void setLeftSpeed( byte leftPwmSpeed )
{
  analogWrite( LEFT_MOTOR_PWM_PIN, leftPwmSpeed );
  leftCurrentSpeed = leftPwmSpeed;
}

void setRightSpeed( byte rightPwmSpeed )
{
  analogWrite( RIGHT_MOTOR_PWM_PIN, rightPwmSpeed );
  rightCurrentSpeed = rightPwmSpeed;
}

void doBrake()
{
  setLeftSpeed( 0 );
  setRightSpeed( 0 );
}

void goFwd( byte leftPwmSpeed, byte rightPwmSpeed )
{
  digitalWrite( LEFT_MOTOR_DIR_PIN, FWD );
  digitalWrite( RIGHT_MOTOR_DIR_PIN, FWD );
  setGlobalSpeed( leftPwmSpeed, rightPwmSpeed );
}

void goLeft( byte pwmSpeed)
{
  digitalWrite( LEFT_MOTOR_DIR_PIN, FWD );
  setGlobalSpeed( 0, pwmSpeed );
}

void goRight( byte pwmSpeed)
{
  digitalWrite( RIGHT_MOTOR_DIR_PIN, FWD );
  setGlobalSpeed( pwmSpeed, 0 );
}

void rotateInPlace( int rotationOffset, byte
    hardwareSpecificRotMult) {
  if (rotationOffset < 0 ) {
    digitalWrite( LEFT_MOTOR_DIR_PIN, REV );
    digitalWrite( RIGHT_MOTOR_DIR_PIN, FWD );
```

```
48    } else {
49      digitalWrite( LEFT_MOTOR_DIR_PIN, FWD );
50      digitalWrite( RIGHT_MOTOR_DIR_PIN, REV );
51    }
52
53    setRightSpeed( maxSpeed );
54    setLeftSpeed( maxSpeed );
55
56    // Basically "wait" for some time, needs fine
        adjustment
57    delay( abs(rotationOffset) *
        hardwareSpecificRotMult);
58    doBrake();
59
60    digitalWrite( LEFT_MOTOR_DIR_PIN, FWD );
61    digitalWrite( RIGHT_MOTOR_DIR_PIN, FWD );
62  }
```

# Appendix B

# Video and materials

File: PERCbot-thesis.zip[a]

[a] http://hci.rwth-aachen.de/tiki-download_file.php?fileId=1793

# Bibliography

Dagu. Arduino Mini Driver Board. http://www.dawnrobotics.co.uk/dagu-arduino-mini-driver-board/, 2015a. [Online; accessed 18-Sept-2015].

Dagu. Arduino Mini Driver Board Datasheet. https://drive.google.com/file/d/0B__O096vyVYqczRjX1k5LTRkcjA/edit, 2015b. [Online; accessed 22-Sept-2015].

Nicholas D'Ademo, Wen Lik Dennis Lui, Wai Ho Li, Y. Ahmet Sekercioglu, and Tom Drummond. eBug - An Open Robotics Platform for Teaching and Research. *Robotics and Automation (ACRA), 2011 Australasian Conference on*, pages 1–9, 2011.

Christoph Ganser, Adrian Steinemann, and Andreas Kunz. InfrActables: Multi-user tracking system for interactive surfaces. *Proceedings - IEEE Virtual Reality*, 2006:36, 2006. ISSN 1087-8270. doi: 10.1109/VR.2006.86.

Jefferson Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. *Proceedings of the 18th annual ACM symposium on User interface software and technology - UIST '05*, pages 115–118, 2005. ISSN 00030147. doi: 10.1145/1095034.1095054. URL http://portal.acm.org/citation.cfm?doid=1095034.1095054.

H. Ishii and B. Ullmer. Tangible bits: towards seamless interfaces between people, bits and atoms. *Proceedings of the SIGCHI conference on Human factors in computing systems*, (March):241, 1997. ISSN 1473558X. doi: http://doi.acm.org/10.1145/604045.604048. URL http://portal.acm.org/citation.cfm?id=258549.258715.

Martin Kaltenbrunner and Ross Bencina. reacTIVision : A Computer-Vision Framework for Table- Based Tangible Interaction. *Group*, pages 15–17, 2007. doi: 10.1145/1226969.1226983.

Aleksander Krzywinski, H Mi, Weiqin Chen, and M Sugimoto. RoboTable: a tabletop framework for tangible interaction with robots in a mixed reality. *Proceedings of the ...*, pages 107–114, 2009. doi: 10.1145/1690388.1690407. URL http://dl.acm.org/citation.cfm?id=1690407.

C. Ladha, Karim Ladha, Jonathan Hook, Daniel Jackson, Gavin Wood, and Patrick Olivier. TouchBridge: augmenting active tangibles for camera-based multitouch surfaces. *ACM International Conference on Interactive Tabletops and Surfaces*, pages 271–272, 2010. doi: 10.1145/1936652.1936711. URL http://portal.acm.org/citation.cfm?id=1936652.1936711.

Microsoft. Microsoft Surface Hub. https://www.microsoft.com/microsoft-surface-hub/, 2015. [Online; accessed 08-Sept-2015].

Takashi Miyaki, Yong Ding, Behnam Banitalebi, and Michael Beigl. Things that Hover: interaction with tiny battery-less robots on desktop. *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems - CHI EA '11*, pages 531–540, 2011. doi: 10.1145/1979742.1979624. URL http://portal.acm.org/citation.cfm?doid=1979742.1979624.

D Nowacka, K Ladha, N Hammerla, D Jackson, C Ladha, E Rukzio, and P Olivier. Touchbugs: actuated tangibles on multi-touch tables. *Proceedings of CHI 2013*, pages 759–762, 2013. doi: 10.1145/2470654.2470761. URL http://dl.acm.org/citation.cfm?id=2470761.

Dan Rosenfeld, Michael Zawadzki, Jeremi Sudol, and Ken Perlin. Physical Objects as Bidirectional User Interface Elements. *IEEE Computer Graphics and Applications*, 24(1): 44–49, 2004. ISSN 02721716. doi: 10.1109/MCG.2004.1255808.

Johannes Schöning, Peter Brandl, Florian Daiber, Florian Echtler, Otmar Hilliges, Jonathan Hook, Markus

Löchtefeld, Nima Motamedi, Laurence Muller, Patrick Olivier, Tim Roth, and Ulrich Von Zadow. Multi-Touch Surfaces : A Technical Guide Technical Report TUM-I0833 Categories and Subject Descriptors. *Most*, page 19, 2008.

Simon Voelker, Kosuke Nakajima, Christian Thoresen, Yuichi Itoh, Kjell Ivar Ø vergård, and Jan Borchers. PUCs. In *Proceedings of the adjunct publication of the 26th annual ACM symposium on User interface software and technology - UIST '13 Adjunct*, pages 1–2, New York, New York, USA, 2013. ACM Press. ISBN 9781450324069. doi: 10.1145/2508468.2514926. URL http://dl.acm.org/citation.cfm?doid=2508468.2514926.

Simon Voelker, Christian Cherek, Jan Thar, Thorsten Karrer, Christian Thoresen, Kjell Ivar Øvergård, and Jan Borchers. Percs: Persistently trackable tangibles on capacitive multi-touch displays. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology (to appear)*, UIST '15, New York, NY, USA, November 2015. ACM. doi: 10.1145/2807442.2807466. URL http://dx.doi.org/10.1145/2807442.2807466.

Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan, and Jan Borchers. Slap widgets: Bridging the gap between virtual and physical controls on tabletops. In *CHI '09: Proceedings of the 27th international conference on Human factors in computing systems*, pages 481–490, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-246-7. doi: http://dx.doi.org/10.1145/1518701.1518779.

Malte Weiss, Florian Schwarz, Simon Jakubowski, and Jan Borchers. Madgets: Actuating Widgets on Interactive Tabletops. *Proceedings of the 23nd annual ACM symposium on User interface software and technology - UIST '10*, pages 293–302, 2010. doi: 10.1145/1866029.1866075. URL http://portal.acm.org/citation.cfm?doid=1866029.1866075.

Weiß, Malte. Bringing Haptic General-Purpose Controls to Interactive Tabletops. August 2012.

Daniel Williams. Reactable. https://upload.wikimedia.org/wikipedia/commons/e/e3/Reactable_Multitouch.jpg, 2007. [Online; accessed 28-Sept-2015].

# Index

Typeset October 5, 2015