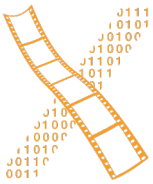

Chair for Computer Science 10 (Media Computing and Human-Computer Interaction)

RWTH AACHEN UNIVERSITY

Tracking Manufacturing Workflows in Makerspace Environments

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

by
Frederik Menke

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Ulrik Schroeder

Registration date: 12.11.2019
Submission date: 16.03.2020

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matrikelnummer

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel

_____

_____

_____

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift

Contents

List of Figures

Abstract

Makerspaces provide tools for creating physical artifacts. Makers and their community benefit from Documentation of such projects as it makes them reproducible, but documenting is a time-consuming, difficult task and interrupts the makers workflow. Therefore, documenting should be assisted, for example in the form of automatic generation of documentation.

In this Bachelor's thesis, a power dill was equipped with sensors to track its movements during a maker project. The sensor data is evaluated live on a desktop application and from the calculated 3D path of the drill, a 3D animation is generated that can serve as documentation.

Überblick

Makerspaces bieten Werkzeuge um physische Objekte zu erzeugen. Projekte in Makerspaces werden besser reproduzierbar wenn sie Dokumentiert werden. Allerdings ist dokumentieren als Tätigkeit zeitintensiv, schwierig und unterbricht den Arbeitsprozess. Daher sollte dokumentieren unterstützt werden, beispielsweise in Form von automatischer Generierung von Dokumentation.

In diese Bachelorarbeit wurde ein elektrischer Schraubendreher mit Sensoren ausgestattet um dessen Bewegungen aufzunehmen. Die Sensordaten werden unmittelbar von einer Desktopapplikation ausgewertet. Aus dem resultierenden 3D Pfad des Schraubendrehers wird daraufhin eine 3D Animation generiert die zur Dokumentation dienen kann.

Acknowledgements

I'd like to thank Marcel Lahaye for advice on scientific writing and help with numerous questions. I also thank Sofie van Anrooij for proofreading and emotional support.

Finally, I'd like to thank the numerous open source projects that made this thesis possible though their work.
Thank you!

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured 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 Canadian English.

Download links are set off in coloured boxes.

File: myFile[a]

[a] http://hci.rwth-aachen.de/public/MenkeTracking/file_number.file

Chapter 1

Introduction

1.1 Motivation

Documenting projects in makerspaces is a difficult and time-consuming task. At the same time, documentation is highly beneficial for both the maker and her/his community.

Documenting makerspace projects is beneficial for Researchers and Makers.

Makerspaces are open public workshops where people can semi-professionally craft artifacts. During a project, a sequence of actions that involve tools and material are taken. The type of Documentation this thesis focuses on is a record of such an action sequence. For completion, other types of documentation are covered in Section 2.1. Documentation can serve multiple purposes. For example, proper documentation of a project enables readers that can have access to the proper tools and materials to copy the design. This is especially true for FabLabs that are a brand of makerspaces that share a recommended standardized inventory of machines and tools[2002]. This means that a project that has been created in a FabLab should be reproducible in any other FabLab given proper documentation. Other uses of documentation include educating readers, giving inspiration for related projects, sharing the makers experience with others, finding flaws in the process that could be improved on and calculating the cost that went into material, tools and workforce[2014][2015]. Also documentation can be used as part of the makers portfolio

[2015]. Additionally, Zimmerman et al. [2010] have shown that missing documentation in a design project can limit the value of a research contribution, because it makes presented artifacts difficult to reproduce. This suggests that the presence of documentation for projects is beneficial for makers and researchers alike.

Creating documentation documentation is time consuming and difficult

Writing by hand is a task that adds overhead to the makers project [2019]. In addition it requires many skills like image manipulation, text formatting, etc. and tools like a webserver to publish the documentation[2019].
Makers might have to document their process during its execution to keep the documentation accurate and to record the current state in the process (e.g. by taking pictures) [2014]. In a survey by Tseng and Resnick [2014] participants stated that they tend to forget documenting certain steps which leads them to having to recreate the state of the project at that step in order to take a picture. When makers have to document a process during its execution, both task become more time consuming. Breaking out of a process in order to create documentation and then going back to making introduces switching costs for each context switch. These costs come in the form of slower and more error prone performance of the subject for a period of time after switching between working and documenting a project [2009][2001][2003].

Documenting could be automated

To avoid a workflow where a maker has to constantly switch between making and documenting, the capturing of the process state may be automated. This way, data can be gathered that would stop makers from having to log the process themselves, and it even might be used for automatically generating (or supporting generation of) documentation.

1.2 Gathering Process Data

This thesis provides an approach of collecting ”process data“ of projects in a makerspace. The gathered data is then supposed to aid authoring documentation for said process.

PROCESS DATA:
Process Data is any data that contains information about the physical execution of a workshop process.

Definition:
Process Data

Cooperative work that involves more than a single person is neglected to keep the scope of the thesis manageable. Other than process data that can be measured physically, authoring of documentation also relies on reasoning behind manual actions during a process. Such reasoning can, for example, explain the function of a groove that has been filed into a bottle. The process information could potentially allow to locate and describe the groove, but it might seem difficult to automatically deduct that the groove marks the maximum fill level of the bottle without reasoning from the maker. Since the presented approach aims to automate documentation, the focus in this thesis lies on process-oriented documentation, that by default mainly consist of process data.
Creating artifacts in a makerspace requires the maker to move tools and materials in relation to each other. If these movements are tracked, the resulting process data can be used to generate documentation.

1.3 Prototype

For this thesis, a cordless drill was equipped with inertial sensors to track its motion. In addition, an application was created that can calculate the position and orientation of the drill from the raw sensor data. A 3D animation can then be generated from the collected process data. Attaching the sensors to the users body directly (e.g. his/her hand), multiple tools could be tracked by using only a single set of sensors. But if the sensors are attached to the tools/material, the movement of the tools/material in relation could be tracked even if the user is not holding any at that moment. This way the data can, for example, tell when a screw clamp slips of the material and has to be reapplied. The object that the user is holding can always be identified, since it each tool and piece of equipment can provide its own stream of data in a separate channel. When attached to

Process Data is gathered by motion tracking tools.

Figure 1.1: A photo of the Prototype. The lid of the hardware case was removed to show the wiring.

the user, the sensor could induce discomfort from wearing them and overhead for putting them on. In contrast, sensors can be integrated into tools making them completely transparent to the user.
To keep the hardware of the prototype simple, the sensors were only embedded into a single tool. The position of the material can be fixed to a certain point and orientation in relation to the starting point of the sensor. Thus, the position of the material relative to the tool is still always known. Since a cordless drill is a common tool, already includes a battery for powering the hardware and is usually big enough for embedding a microcontroller and sensors into it, it was chosen as an ideal candidate for tracking.

Therefore, a microcontroller and an IMU were embedded into a cordless driver in the aim to track a workshop task. The sensor data is sent via WLAN to a PC that calculates the position and orientation of the drill for each sample of the IMU output. A python script for the 3D creation suite blender[1] was created that uses the samples to generate an animation of the process.

1.4 Overview

The thesis is split into three Chapters. In Chapter 3 the hardware of the prototype will be presented and explained. Chapter 4 covers the structure of the prototype software. Finally, chapter 5 will explain how the sensor data are used and will showcase an example program that shows the usage of presented libraries.

[1] www.blender.org

Chapter 2

Related work

2.1 Documentation

Documentation can take multiple forms. A few examples are:

- **Bill of Materials** (BOMs) are just tables that list the materials (and sometimes tools) that are needed to fulfill a project. Each row usually lists kind, amount and shape of the needed material.
- **Manuals** describe the features of a creation and how to use it.
- **Process-Oriented Documentation** consists of a list of consecutive actions that have been performed in the makerspace during a project. This includes iterations of modifying the design and decisions that were later reverted and didn't make it into the final artifact [2015].
- **Step-By-Step Tutorials** are the subset of the contents of story-like documentation that only contains the actions that directly lead to the creation of the artifact. As such, they contain a list of tasks that have to be fulfilled in order to create an artifact. They often contain BOMs and provide thus just enough information to recreate a project.

Process-Oriented documentation will be targeted in this thesis.

The creation of documentation can only be automated if the documentation mainly consists of process data. Documentation types like manuals need reasoning by the author that can explain the features of a creation. This makes manuals less suitable for automated documentation than e.g. tutorials or process-oriented documentation. Tseng and Resnick [2014] conducted a survey on users of Instructables[1], which is an online platform for sharing documentation. In the survey, participants ranked their reasons for looking at Instructables from most to least important. Between the choices "To get Ideas for a project", "To learn a particular technique" and "To look for projects I want to recreate", the latter one was the least favorite choice.
Tseng and Resnick [2014] therefore recommend to use process-oriented documentation that not only states how to create a specific artifact, but also shows what other things were tried by the author and possibly what to avoid when designing for such a product. A step-by-step tutorial for a project is a subset of the actions that could be found in process-oriented documentation. This means that an author could always create a step-by-step tutorial of his project from the process documentation. Similarly, BOMs could be generated from process documentation as well, since each action where the maker takes out new material from storage could be tracked.
For the reasons above, the rest of this thesis will focus on the generation of process-oriented documentation.

2.2 Simplifying Documentation

Related work has tried minimizing the overhead of writing documentation

Milara et al. [2019] embraced "documenting-while-doing" by creating a mobile phone application and a corresponding webapp. These aim at minimizing the overhead that stems from having to use various software programs and hardware to compile the documentation. Instead, the user records the process through a single application during execution. The collected pictures, notes and voice messages are uploaded to a web server where they are compiled into a webpage. The webapp allows the user to later access

[1] www.instructables.com

this page and modify it further. The technique that is presented in this thesis fits right into the approach by Milara et al. [2019]. By recording the process data without interrupting the process, additional overhead is removed from creating documentation. Also Milara et al. [2019] designed their system with process-oriented documentation in mind, which can be further supported by the here presented prototype, since it outputs a continuous data stream instead of a series of photos. This way it is ensured that not only the successful parts of the users process are documented. Tseng and Resnick [2016] conducted multiple case studies on their approach on supporting the creation of documentation for young people. They designed a turntable with an integrated camera that automatically creates spinning animations of any project that was laid onto it. This way, the users were able to document their progress on a project by simply laying it on the turntable every now and then. Learning from their case studies, Tseng and Resnick [2016] proposed guidelines for supporting the creation of documentation. These include that such a device should minimize disruption of a task, and that it benefits from being integrated into the workshop. Since the artifact that is presented in this thesis integrates into a power drill without changing its shape significantly, both of these rules can be fulfilled by it.

2.3 Generating Documentation

Techniques for automatic generation of tutorials have been implemented for image manipulation tasks on desktop PCs.

Automatic creation of step-by-step tutorials has already been implemented by Grabler et al. [2009] for image processing software on desktop computers. They created a system that records user input and screen contents during a photo manipulation task. For each step in the users process, screen captures are outfitted with labels, text descriptions and annotations to highlight certain areas. In the end, these pictures are assembled into a whole step-by-step tutorial. Chi et al. [2012] took a similar approach, but mixed video and image footage to show the effect of actions in the image processing software. A major difference to the topic of this thesis is the way the process is recorded. Grabler et al. [2009] and Chi et al. [2012] modified the image processing

software to record any actions within the interface and also recorded the screen. This way they were able to use the input, that the user made during the task, directly for generating the tutorials from a sequence out of a fixed set of possible actions in the software. For this thesis, a physical motion in 3D space will be tracked. Therefore neither is the set of possible actions finite, nor is the process data as accurate as when recording mouse and keyboard input. Instead, in this thesis, a stream of continuous defective 6-axis data will be measured. Additionally, all of the aforementioned techniques aimed at creating tutorials instead of process-oriented documentation. This means that a different approach to the generation of documentation will have to be designed.

2.4 Tracking Power Tools

There have been several approaches for detecting user movements in a workshop setting. Ward et al. [2006] used accelerometers and microphones to detect the current activity of a user. In this approach the data was only interpreted by learning algorithms in order to classify data as a corresponding activity. The way in which the activity was performed, or what influence it had on the process was not examined. Schoop et al. [2016] augmented power tools in order to guide the user through woodworking tasks. They mounted three distance sensors on a power drill to measure the distance and angle to the workpiece. Doing so, the system could tell if the drill was held an angle or straight onto a sheet of material. This thesis presents a technique for gathering 3D data that would contain information about angle towards the material. Antifakos et al. [2002] used serveral different sensors to infer the current state in a furniture assembly task. While these techniques were able to detect tool usage, the researchers focused in guiding the user through a workshop task, starting from already set-up instructions. The technique presented in this thesis is not limited to a single predefined process, but aims to detect tool usage within arbitrary drilling tasks. It therefore could potentially generate data for the techniques by Antifakos, Schoop, Ward et al.

2.5 3D Motion Tracking

3D position and orientation of tools will be tracked as main process data for generating documentation.

To find out how process information could be gathered, one first has to restrict the information to the significant bits. In the abstract, process-oriented documentation describes how a human moved material and tools in relation to each other in order to produce or modify an artifact. The movements can be grouped within the timeline into sections that make up a single task within the process. Therefore, the 3D movements of tools and material have to be tracked, and this data has to be dissected at reasonable points in time. As seen in a technical report by Campbell et al. [2015], 3D data can be used to recognize activities. Using such an approach, the detection of a step could be by looking for pauses and changes in activities. Campbells approach is limited by the raw nature of the used sensor that could only gather information on acceleration of the users hand. If 3D movement data of a specific tool was gathered, the recognition of activities could possibly be more accurate, since the type of tool and its movement relative to other tools would be known at all time.

3D-tracking is possible through a variety of techniques.

Welch and Foxlin [2002] list common methods to perform 3D motion tracking.

- Mechanical sensing can be used by attaching a limb-like contraption to the tracked object. The contraption is fixated at a well known position in space and thus the position of the object can be calculated by measuring the angles at the joints of the limb (This is usually done by incorporating sensors into the joints). While this technique can provide highly precise positional data, using such mechanical sensing has a variety of drawbacks that make it not suitable for our purpose. The limbs length restricts the positional range of the assembly, making it impossible to track the tool further than a few meters. Additionally, the assembly may block the view on the tool, and when tracking multiple tools at the same time, the limbs might physically interfere with each other, thus restricting the users mobility.

- Acoustic sensing that uses microphones to detect the distance to a signal doesn't restrict the user in movement or sight, but (depending on configuration) lacks either precision or temporal resolution if used in an indoor environment.

- Optical sensing uses light sensors to detect the distance to a light source, or cameras to locate objects or patterns within an image. These techniques can provide precise data with a high rate. To calculate the position of an object, the light sensors always require at least one light source or pattern as reference point that has to be placed in the scene. Furthermore, the sensor has to be in constant direct line of sight for measurement. In a similar fashion as for the mechanical sensing, this would restrict the users movement when holding the tool to avoid blocking the view of the sensor to a reference point.

Inertial and magnetic sensing via accelerometer, gyroscope and magnetometer was chosen as the 3D-tracking method.

- For this thesis, inertial and magnetic sensing was chosen. An inertial measurement unit (IMU) is a combination of a 3-axis gyroscope that measures angular velocity and a 3-axis accelerometer that measures acceleration. A magnetometer is a 3-axis sensor that provides the direction and strength of a local magnetic field. These sensors can provide 3D vectors in often high temporal resolution. By integrating the output of the gyroscope once and the accelerometer twice, orientation and position (relative to a starting point) can be calculated. In combination with the magnetometer data, the global orientation can be determined from earths gravity and magnetic field.
  IMUs are available in IC-Packages with footprints of less than $5\,\mathrm{mm}^2$ which allows for embedding them into small objects. Inertial sensing requires a single external reference for giving a starting point in global space and the sensor becomes independent from it afterwards. This means that IMUs can be integrated into many tools without obstructing the user and without the need for a reference point in constant line-of-sight.
  Still, for 3D-tracking, IMUs suffer from drift that introduces error into the calculated position and orientation data. Since the accelerometer output has to be

integrated twice, the positional error will be at least proportional to the cubed sensor error. The resulting limitations will be discussed later on.

Chapter 3

Prototype Hardware

This chapter will cover the hardware that comprises the prototype. It will explain why certain components were chosen and how the custom hardware was designed and manufactured. The cad files of the custom hardware can be found at

File: cad.zip[a]

[a] http://hci.rwth-aachen.de/public/MenkeTracking/cad.zip

The prototypes hardware is discussed in this chapter.

In the abstract, the protoype is composed of a sensor unit and a receiver. The sensor unit is attached to the cordless drill via a plastic harness (see Figure 1.1). It collects data from the onboard accelerometer, gyroscope, magnetometer and current sensor and serves it via a TCP server and its WLAN interface. The receiver runs as a python application on a desktop computer. It connects as a client to the TCP server on the sensor unit, receives the sensor data and processes it.

3.1 Sensor Unit Hardware

The sensor unit hardware is comprised of modules on a mainboard.

The electronics of the Sensor Unit consist of 4 main components:

- An ESP-WROOM32 Development Board (ESP32 DEVKITV1) for polling sensors and wifi communication.
- A "GY-87" inertial measurement unit (IMU) module containing a gyroscope, an accelerometer and a magnetometer.
- A current sensor module based on the ACS712.
- A custom mainboard that connects and powers the other components.

To keep reproducibility simple, the components come in breakout boards that can be connected through standard $2.54\,\mathrm{mm}$ pin headers. This way the components can be connected with bread-/perfboards and wire for easy prototyping.
The custom mainboard was also kept simple, as it is single sided and uses three types of standard, low-cost, through-hole components. Since single-sided boards with only through-hole components can be easily manufactured using perfboards, even a makerspace that lacks the equipment for etching or milling printed circuit boards can recreate the prototype.

An ESP32 microcontroller was chosen for IOT capabilities.

This thesis is directed towards makers. Therefore, one could suggest that an Arduino microcontroller should have been used, since they are well established in the maker community. Nonetheless, the ESP32 microcontroller has been chosen in favor of an Arduino, because of multiple reasons: first of all, the platform is performant for a microcontroller, running at a clock speed of up to 240Mhz (almost three times the speed of an Arduino Due) in a dual-core setup with an internal SRAM of 520kB (5.5 times the size of an Arduino Due). Secondly, the ESP32 comes mounted on a way smaller development board than any of the more performant Arduinos, having a footprint of only 28x52mm in

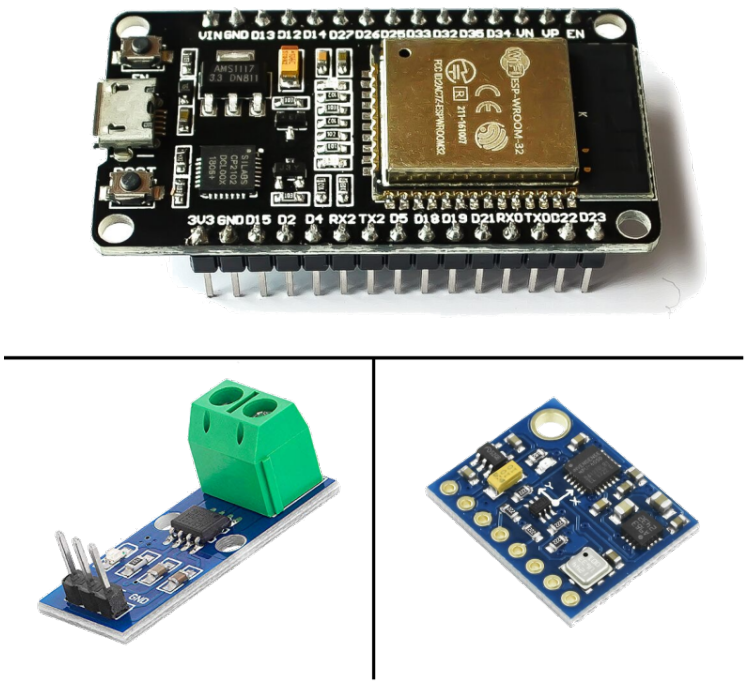

Figure 3.1: The modules that plug onto the mainboard. Visible are the ESP32 microcontroller (top), the current sensor (bottom left) and the GY-87 inertial measurement sensor (bottom right).

comparison to the 100x50mm of an Arduino Due. In addition, the ESP32 development board provides hardware peripherals as an "Inter-Integrated Circuit"-bus (I^2C), analog-digital converters (ADCs), and WLAN out of the box. This allows the ESP32 to CPU-efficiently communicate with the sensors and the receiver.

The GY-87 was chosen as Inertial Measurement Unit.

The inertial measurement unit comes with two integrated circuits: An MPU-6050 6-axis sensor that consists of a gyroscope and an accelerometer, and a "HMC5883L" 3-axis digital compass. The ICs are connected to each other via an I^2C bus that allows the MPU-6050 to serve as an I^2C master over the HMC5883L and automatically read out its data. Thus, when configured correctly, the 6-axis sensor can collect the gyroscope, accelerometer and compass data within its internal buffer, saving CPU-time on the ESP32. The data

can then be burst-read via another I^2C bus that connects to the ESP32 and the 6-axis sensor.

The current sensor unit measures the current flow via two screw clamps that can be connected between a power source and a power sink. In this particular setup, the sensor is connected to one of the cables coming from the power drill motor. It outputs the current readings via an analogue signal on one of its pins. The signals voltage is proportional to the current that is flowing through the screw clamps.

3.2 Receiver Hardware

A PC runs the receiver application as a python program.

The receiver application runs on any unixoid system with a python3 interpreter and a network interface card. If a screen is connected to the system, the sensor movements can be displayed on it. The software is completely written in python and thus portable to other operating systems like Microsoft Windows with minor modifications. For the purpose of testing the application and debugging, a 2018, low-end lenovo laptop was used. This means that the receiver application can run even on affordable hardware. As the application (being written in python) is confined to a single process, it could be run on a multi-core system several times in parallel to track multiple tools at the same time.

3.3 Custom Hardware Design

Custom parts were created.

For the creation of the prototype, two custom pieces of hardware were manufactured within a Makerspace using a mid-level 3D printer and a CNC PCB milling machine. The following subsections document the design and manufacturing process of the custom PCB and the harness that fixes the sensor unit on the cordless drill.

3.3.1 Drill Modification

The drill motor cables were exposed.

To measure the motor current via the current sensor, the motor cables have to lead through the screw clamps of the sensor. Therefore the drill was opened by unscrewing the screws in its casing. Then the motor cable was cut in half and extended by soldering an additional length of cable to each end. To lead these cables out of the case, a hole was created on the end of the motor housing by filing out a notch on each half of the casing. The casing was then re-assembled while taking care that the cable extensions were left hanging out of the back of the motor housing.

3.3.2 Custom Printed Circuit Board

Mainboard Purpose

The mainboard serves three purposes: First of all, it holds the hardware modules in place with female headers and can be bolted onto the drill harness with screws (see section 3.3.3). This makes sure that the sensor unit follows the movements of the drill.
Secondly, the sensors and the microcontroller are powered by the mainboard. It contains a voltage regulator that takes a 9 Volt input and outputs 5 Volt to the components. The regulator itself can be powered through a set of screw clamps. To dissipate excess heat from the voltage regulator, its heatsink is soldered onto a copper plane on the mainboard.
Finally, the mainboard interconnects the components. The I^2C bus of the ESP32 and the IMU are connected through traces. Additionally, two resistors serve as voltage divider to map the maximum 3.3 Volt output from the current sensor to the ESP32s 1 Volt analogue reference.

Mainboard Design and Manufacturing

The mainboard PCB was designed in KiCAD.

The mainboard was designed using the KiCAD[1] electronic

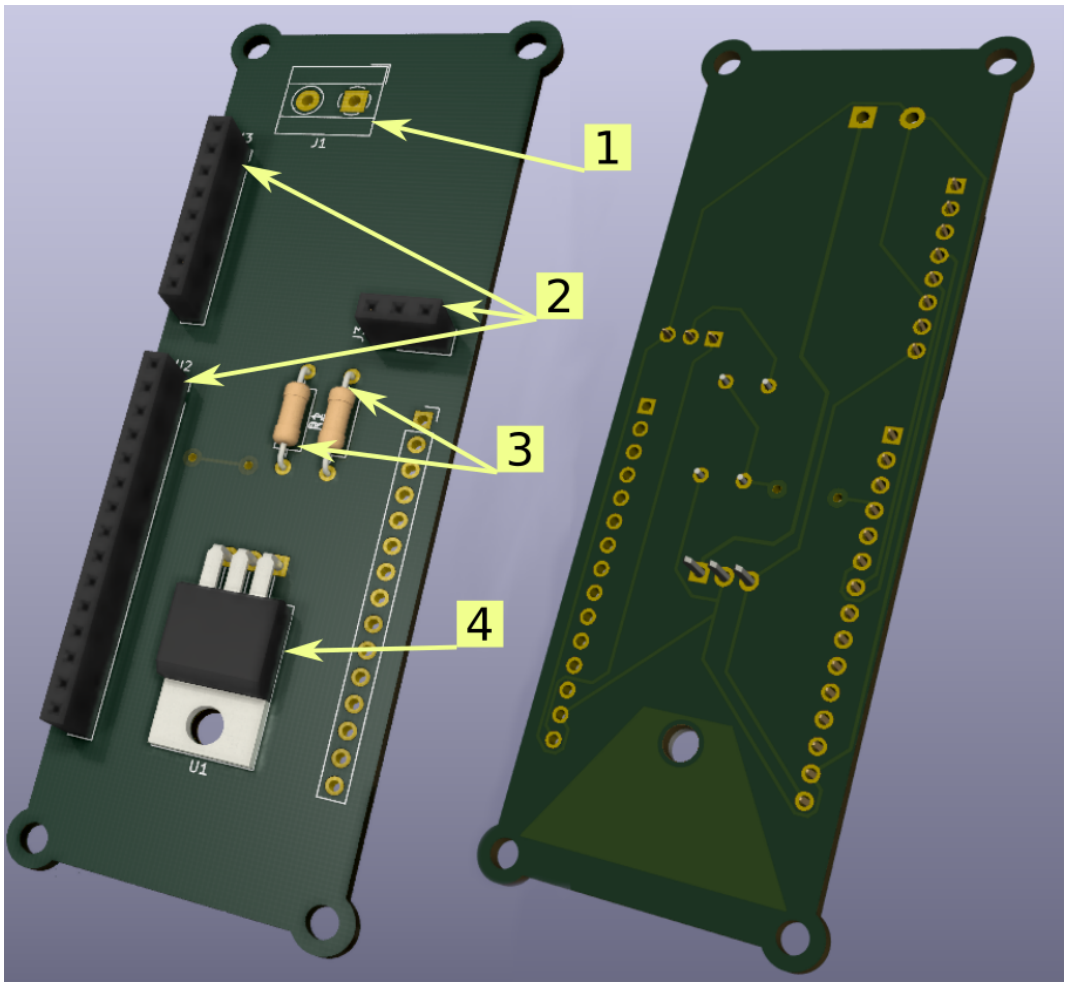

Figure 3.2: The figure shows rendered pictures of the custom PCB that was designed to hold the sensor modules and the microcontroller. The screw clamp and one of the female headers have been removed to show the pads and silkscreen below. Visible are the pads for the screw clamp (1), female headers (2), two resistors as voltage dividers (3) and the voltage regulator (4).

design automation suite. This software was chosen as it is well supported by the PCB mill that was made accessible to me by the FabLab Aachen. Also, KiCAD is free and open source, making it available to the maker community.

First, using the Eeschema editor in KiCAD, a schematic was created (see Figure 3.3). This included creating symbols for

[1] www.kicad-PCB.org

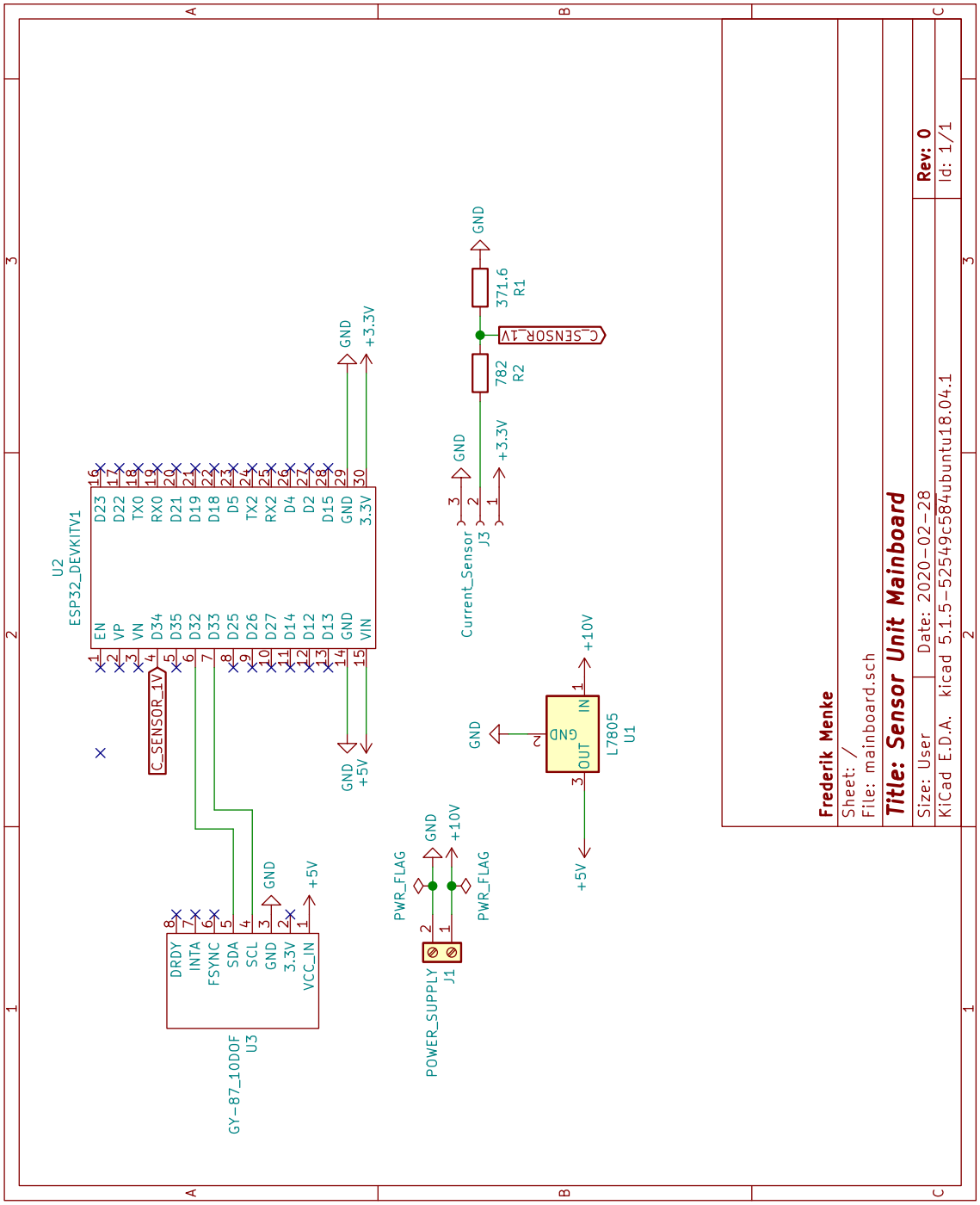

Figure 3.3: The schematic of the mainboard

the GY-87 and the ESP32, since those are not included in KiCAD.
From this schematic, a PCB layout was designed. To disturb the user as little as possible when handling the drill, focus was layed on minimizing the size of the PCB. For that reason, the resistors and the voltage regulator were placed between the headers of the ESP32 (see Figure 3.2). This way, these components sit in the gap below the ESP32. The resulting PCB has an area of 28.21cm^2 while the footprints of the used components total at 24.88cm^2. To keep the design single sided, a bridge connects two pads that would otherwise require a second layer for routing. The gap can be connected with a $0\,\Omega$ resistor or a wire. For the voltage divider, $2.2\,\mathrm{k}\Omega$ and $1\,\mathrm{k}\Omega$ resistors were chosen.

The mainboard was milled from FR4.

The mainboard was milled from a single sided copper clad (FR-4) on an LPKF Protomat S104 circuit board plotter. After milling, the components were soldered onto the PCB. The sensors and the ESP32 were then inserted into the female headers.

3.3.3 Case and Mounting Harness

3D prints encase and attach the mainboard.

The sensor unit hardware is encased by a plastic box that protects the electronics from shorting out and from mechanical strain. With a plastic harness that fits tightly around the power drill, the hardware and the case are fixed to the drill.
All of the 3D models were designed in OpenSCAD[2]. OpenSCAD is an open source, scripted, computer aided design language. The designs were then 3D-printed in PLA on an Ultimaker 3 extended, which is a mid-level fused filament modeling (FFM) 3D-printer.

Case Design

The 3D model of the case was created.

To design the hardware case, the bounding box of the hardware assembly was measured. From that, a cuboid was

[2] www.openscad.org

created with a hollow interior that fits the bounding box. At the bottom of the cuboid, 4 cylindrical pedestrals were added at the position of each screw hole on the mainboard. In the center of those pedestrals, $3\,\mathrm{mm}$ through holes were created to fit M3 screws. At one side, an additional hole serves as outlet for the current sensor and battery cables. To make handling of the case more comfortable, all exterior edges of the cuboid were champfered. Then, the top surface of the cuboid was removed, making the bottom part of the case complete (see Figure 3.4 (a)).
The lid of the case is created from a plate that has pins added to each corner, to align the lid with the case and keep it there by friction (see Figure 3.4 (c)).

Battery Plate Design

A plate to hold the battery was designed.

A rectangular plate was created as base of the battery plate (see Figure 3.4 (b)). On top of that plate, a box with an opening on one side hosts the battery. The box is slightly undersized, thus holding the battery in place by friction. Similar to the hardware case, the battery plate has holes for M3 screws in each corner.

Harness Design

A harness to attach the case to the drill was designed.

The harness sits around the motor housing of the cordless drill. Rings around both ends of the housing clamp down on it. They are held together by the hardware case on one side and the battery case on the other side of the drill (see Figure 3.4 (d) and (e)). The rings were designed by taking two measurements on each side of the motor housing and then creating a short tube that has a slight slope towards the center of the drill. The rings also have additional features that accommodate for notches in the motor housing.
Since the exact parameters were difficult to measure, the harness parts were printed multiple times and the according variables of the OpenSCAD script were tweaked each time until a proper fit was achieved. Then, cuboid pedestrals with slots for square nuts were added to the rings, so that the case and the battery plate can be mounted.

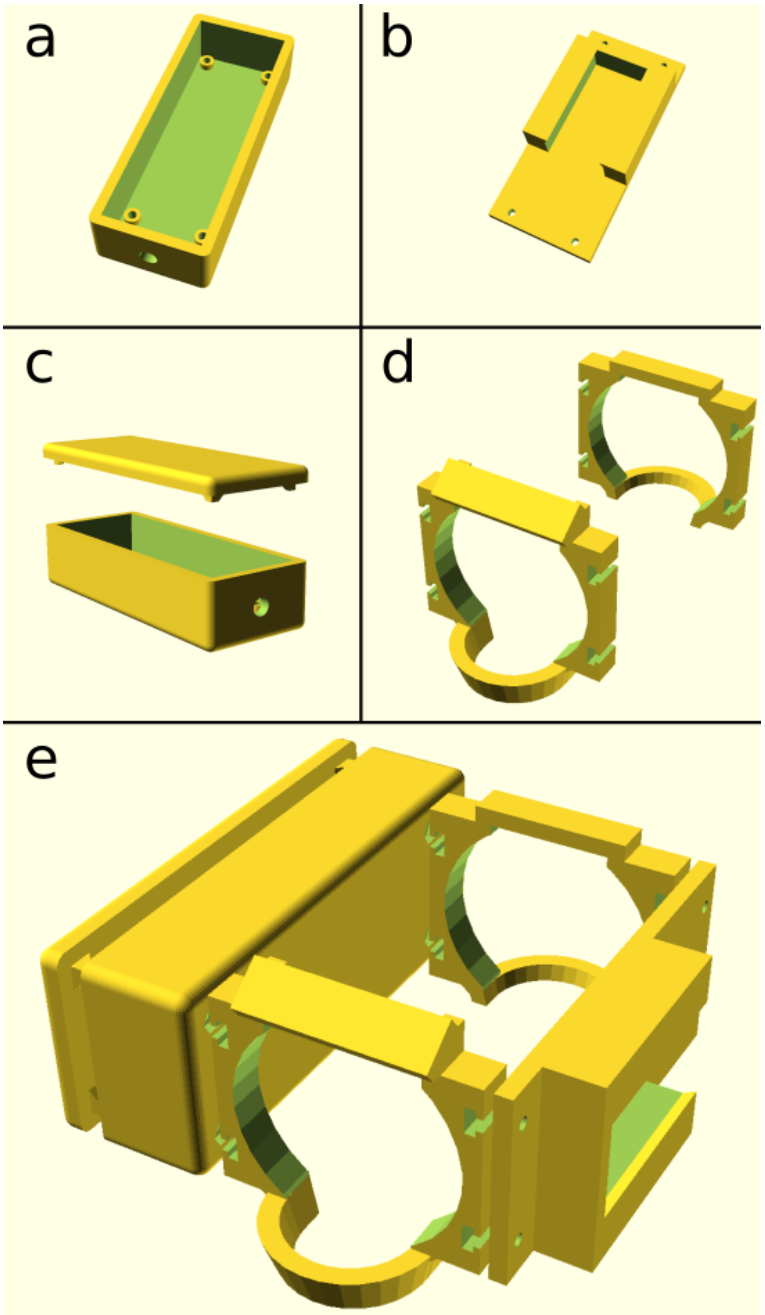

Figure 3.4: Renderings of the 3D models for the case (a), battery plate (b), lid with the case (c) and harness (d) of the prototype. At the bottom, the parts are seen arranged as they would be on the drill (e).

3.3.4 Assembling

The mainboard, case and harness were fixed on the dill.

To assemble the prototype, first the mainboard has to be equipped with the hardware modules. Then, it can be inserted into the case and 4 M3 screws can be pushed through the mounting holes in the mainboard and case. Then the rings of the harness are pushed onto the motor housing of the drill and M3 square nuts are inserted into the slots. Next, the screws in the case are screwed into the nuts in the harness. Similarly, the battery plate is mounted on the opposite side. Then, a 9 Volt battery with a battery clip is inserted into the battery plate and the cables of the battery and the motor are screwed into their according screw clamps on the mainboard. Finally, the lid can be pushed onto the case, completing the assembly.

Chapter 4

Prototype Software

The chapter presents the software for the receiver and the sensor unit.

This chapter will present the structure of the software for the sensor unit and the receiver. Afterwards, details about their interaction will be explained within Chapter 5.

The software of the prototype is split into a C program that runs on the ESP32 and coordinates sensor and WLAN communication, and a python3 library that receives and interprets the sensor data. The interface between these components is realized through a TCP connection over WLAN. The source code of the software can be found at

File: software.zip[a]

[a] http://hci.rwth-aachen.de/public/MenkeTracking/software.zip

4.1 Sensor Unit Software

Custom C libraries and a main program were written.

The sensor unit software consists of a C program with two processes that call three custom libraries. While one process constantly polls the sensors for new data, the other controls the TCP interface. The libraries build upon the Espressif IoT Development Framework (ESP-IDF). This section first describes the functionality of the libraries and then their use within the main program.

4.1.1 Current Sensor Library

A library to interface the current sensor was created.

This library provides a function to initialize the analog digital converter (ADC) of the ESP32 and a second function that reads a single sample from the ADC and thus from the current sensor. In addition, it has a private variable that can be set to the current value of the ADC by calling `current_sensor_update()`. The content of this variable is returned by the `current_sensor_getSample()` function.

4.1.2 I²C Abstraction Library

I²C register manipulation was abstracted by a library.

This Library provides abstraction functionality for register access on the IMU. It provides a `i2c_abs_init()` function that initializes and configures the ESP-IDF I²C interface. The sensor-ICs of the IMU are accessible through I²C register reads and writes. This requires to first send a register address to the ICs and then access it through a successive read or write. These accesses have been abstracted by the functions `i2c_abs_read_byte()` and `i2c_abs_write_byte()`. During the initialisation of the IMU, often single bits have to be set. Since only whole bytes can be written, this requires to read the register content before writing so no other bits are modified. This read/write process is abstracted through the `i2c_abs_write_bit()` function. The MPU-6050 has an internal buffer that needs to be burst-read when operating with high sample rates. Therefore an `i2c_abs_read()` function was implemented that reads a specified amount of bytes.

4.1.3 IMU Library

The IMU is accessible through a library.

The force_sensor library contains functions to initialize and access the MPU-6050 gyroscope/accelerometer and the HMC5883L digital compass. The sensors are configured by writing to their configuration registers via I²C .
The initialisation of the IMU has to be done in three stages: First, the MPU-6050 has to be configured to "I²C

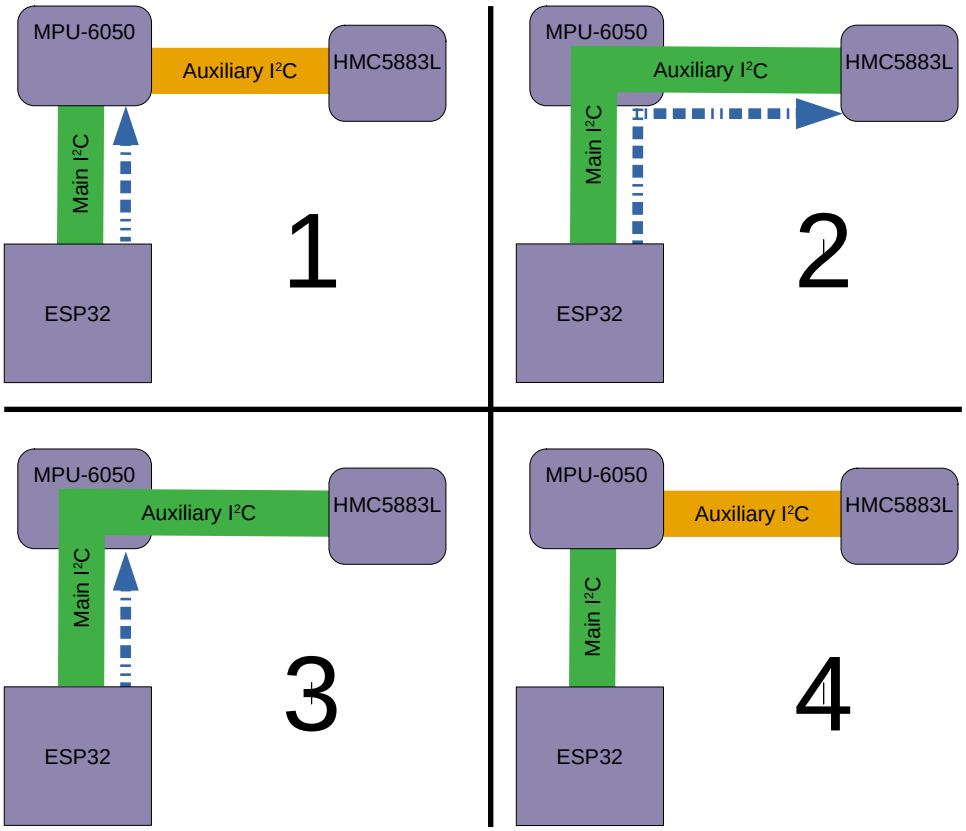

Figure 4.1: The figure shows the stages of configuring the IMU: First, the MPU-6050 has to be configured to go into "I^2C Passthrough Mode" (1). In this mode, the Auxiliary I^2C is connected with the main I^2C and the HMC5883L can be configured (2). Then, the MPU-6050 is configured and the "I^2C Passthrough Mode" is disabled (3). Afterwards, the IMU is ready to start sampling (4).

Passthrough Mode". This mode exposes the auxiliary I^2C interface of the MPU-6050 to its main interface. The HMC5883L that is connected to the auxiliary I^2C , can now be configured to run in continuous measurement mode at the maximum sample-rate (see figure 4.1 (2)). Finally, the MPU-6050 is configured, turning the "I^2C Passthrough Mode" off, configuring the sample rate and full scale range, configuring the master mode over the HMC5883L and setting up the FIFO buffer where samples are gathered for burst read. After the configuration, the IMU waits in sleep

mode. While each stage has been implemented in a separate function, the whole initialisation process can be performed by calling `force_sensor_init()`.

After initialisation, library functions can be called to interact with the IMU:

- `force_sensor_set_sleep()` starts or stops sampling of the IMU.
- `force_sensor_ready_samples_num()` polls and returns the number of samples in the FIFO
- `force_sensor_read_samples()` reads samples from the FIFO into a given buffer.

4.1.4 Control Flow

The main program has to use both CPU cores.

Receiving sensor data from the IMU and sending it to the receiver via TCP has to be done in separate processes, as the TCP driver of the ESP32 uses the slow-start algorithm for congestion control. This algorithm causes the TCP connection to be slow at first, speeding up exponentially. Therefore, if the sensor unit was to send small packages in a high frequency, the throughput would be lower than the output of the IMU. On the other hand, if a single process was to burst write a large buffer of samples to the TCP connection, the IMUs internal FIFO would overflow before the needed TCP speed was reached. To keep up with the incoming data, both CPU cores of the ESP32 are used as follows.

One CPU core reads from the sensor while the other one sends data to the receiver.

On start up of the ESP32, the main task initializes the aforementioned libraries and ESP32 core libraries that are needed for TCP functionality. The task finishes by starting the `tcp_server_task` on CPU core 0. This task sets up a TCP server and waits for a client to connect. When a connection is established, the `i2c_gather_task` is started on core 0 and wakes up the IMU. It proceeds to poll the IMU for samples and save the gathered data into one of two send buffers. As soon as the buffer is filled, `i2c_gather_task` sets a flag and the `tcp_server_task`

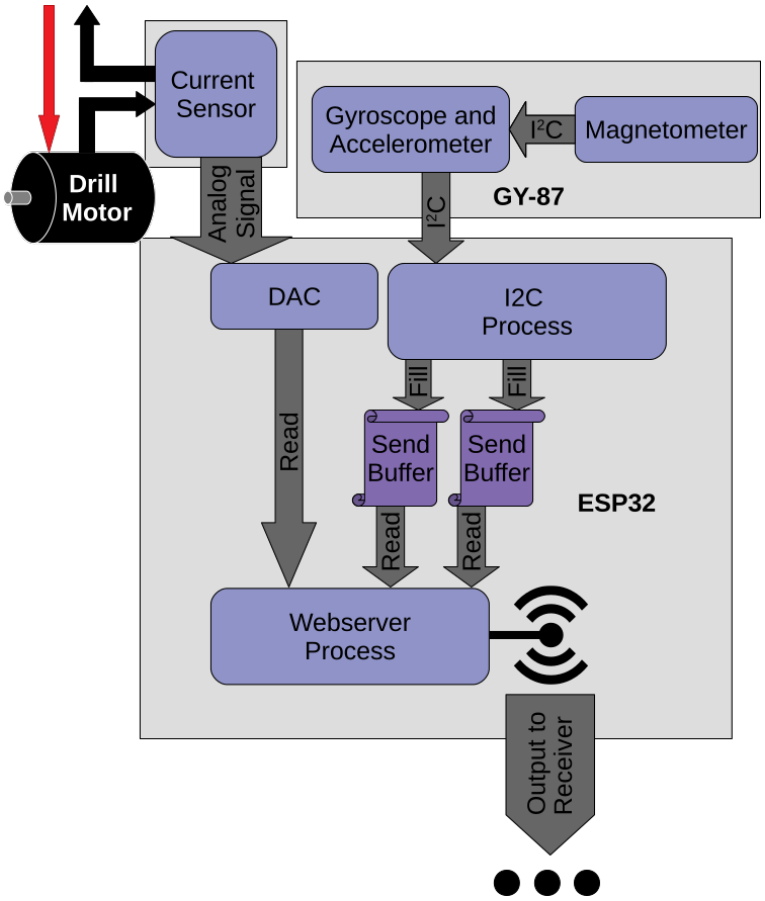

Figure 4.2: The figure shows the data flow within the sensor unit. Originating at the sensors, the measurements are read by the processes on the ESP32 and then sent to the receiver via WLAN.

starts writing the data to its TCP client. While sending, the `i2c_gather_task` continues collecting data into the second send buffer to avoid an overflow within the IMUs internal FIFO. When that buffer is full, another flag is set, the `tcp_server_task` writes it to its client and the cycle continues. The current sensor samples are measured by the `tcp_server_task` while it waits for the `i2c_gather_task` to fill a buffer. The `i2c_gather_task` then saves the current sensor samples to the send buffers when writing the IMU data. A visualisation of how data flows within the sensor unit can be seen at figure 4.2.

The TCP connection has sufficient throughput.

The TCP capabilities of the ESP32 allow for a throughput of multiple $\mathrm{Mbit/s}$ if sending large chunks of data, while the sensor has an output of only $144\,\mathrm{kbit/s}$. Therefore, the `tcp_server_task` will send a complete buffer before the `i2c_gather_task` starts filling that buffer again. Should the IMU FIFO still overflow due to e.g. poor WLAN conditions, the buffer size can be increased to further improve TCP performance.

4.2 Receiver Software

A desktop application receives and interprets the sensor data.

The data from the sensor unit is read and evaluated by python code on a unix system as described in section 3.2. Therefor, the receiver acts as a TCP-client of the sensor. A model describing the sensor position and orientation is created from the thus retrieved data. From this model, a live plot and a rendered animation can be created. Other than the sensor unit that is run by a fixed firmware, the receiver software is given in the form of libraries that can be plugged together in order to create an application that generates documentation.
This subsection will describe each part of the receiver software. The code is organized within two python modules: `backend` and `generating_documentation`. Aside of the python core libraries, the NumPy[1] and the SciPy[2] packages and their dependencies must be installed on the system in order to use these libraries.

4.2.1 TCP Receiver

The TCP connection is handled by a `Receiver` object.

The module `backend.io.tcp_receive` exports the `Receiver`-object. This object provides functions to connect to the sensor unit and receive data from it. The `read`-method returns a list of `DataPoints` as exported from `backend.model.data`.

[1] numpy.org
[2] www.scipy.org

4.2.2 Data Model

The received data is organized and evaluated.

Three classes that are exported by the files in `backend.model` handle the data model:

- `DataPoint`-objects contain a single sample with values in SI-units and a timestamp. The conversion from the raw sensor values is done in the constructor.
- `Model`-objects contain a state of the sensor unit including 3D position, an orientation matrix, velocity and motor current. They also provide a method to interpolate from the current state towards a given `DataPoint`.
- A `DataSet`-object contains a set of `DataPoint`s and `Model`s and has methods to calibrate the orientation from samples that were measured in a stationary position, to generate an initial model. It also removes sensor biases if it is configured that way and proper measurements are available (see section 4.2.3).

Section 5.2 explains how these classes interact to model the movements of the sensor unit.

4.2.3 Debiasing

Sensor bias is measured and removed.

Biases are constant errors within the sensor data. To measure such biases, the `backend.model.bias` module provides applications with graphical user interfaces. These guide the user through taking measurements and save the measured biases to a json file.

4.2.4 Plotting

Live plotting of sensor data was implemented.

For debugging and showcase, the python script `plot.py` shows a plot of live sensor data via the SciPy library. The application reads json-encoded `Model` samples from a named pipe. It then plots the orientation of the model as 3

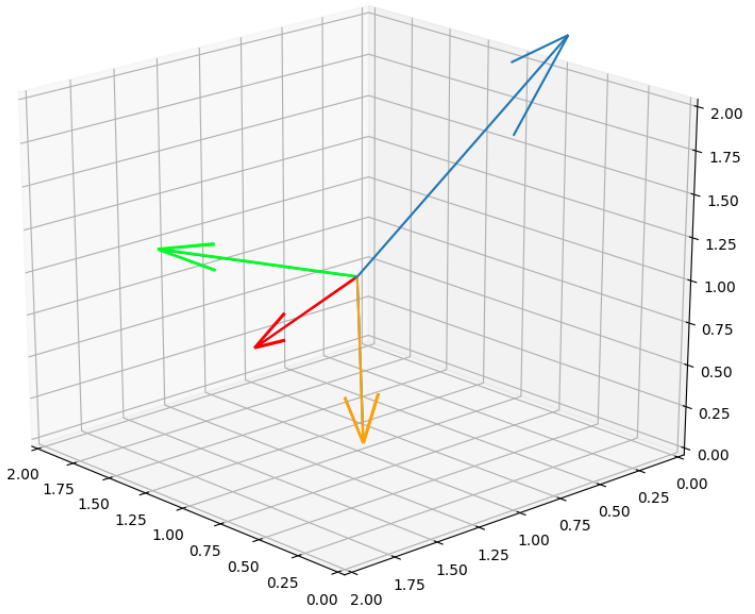

Figure 4.3: The figure shows a live plot of the sensor model. The 3 axes of the gyroscope can be seen as colored arrows in red, orange and green. The current position of the model relative to the origin is indicated by the blue arrow.

vectors within a 3-dimensional cartesian coordinate system (see figure 4.3). An additional vector displays the position of the model. This way the rotation and movement of the real drill can be directly compared to the movement of the vectors in the graph.

4.3 Generating Animations

3D animations are created from tracking data.

As an example of generating process documentation, a video animation can be created from `Models`. This is done by utilizing the scripting capabilities of the open source 3D creation suite blender[3]. The provided .blend file includes a scene with a 3D-model of a drill. The script creates a keyframe for the model drill from each given `Model`. These

[3] www.blender.org

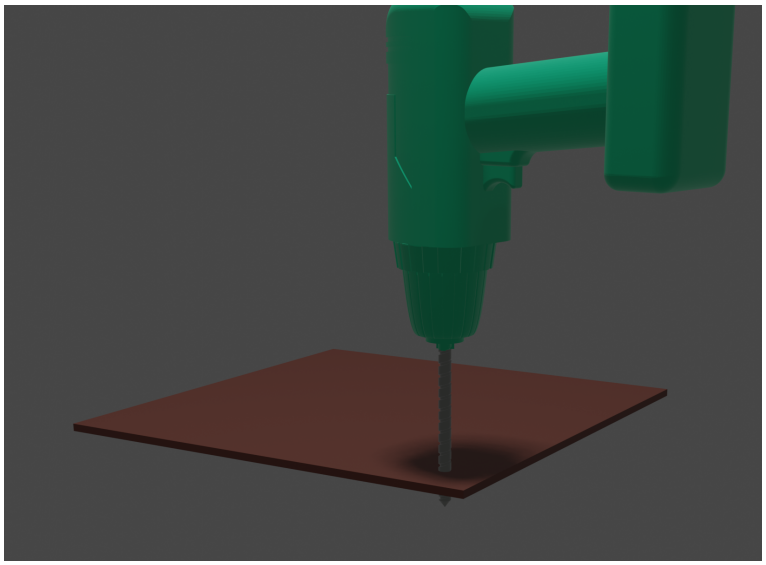

Figure 4.4: This figure shows a frame of the animation generated from dummy sensor data.

keyframes serve as reference for animating the movement of the drill. The animation can then be rendered and saved as a video file (see figure 4.4). To show how the script works, another python library was created that can generate dummy sensor data to test it.

Chapter 5

Theoretical Background

This chapter explains the calculations that the software performs during operation. Additionally, an example program for the receiver is presented.

5.1 Debiasing

5.1.1 Measuring Gyroscope Bias

Gyroscope bias is measured.

A gyroscope measures the angular velocity around its axes. If we ignore sensor errors other than bias, the output $p(t)$ at the time t of a biased gyroscope is therefore given by

$$p(t) = \phi(t) + b_\phi \tag{5.1}$$

Where $\phi(t)$ is the non-biased angular velocity and b_ϕ is the gyroscope bias. The gyroscope bias is determined by resting the sensor unit at a stationary position while taking measurements. When the gyroscope is not in movement, it holds that

$$\phi(t) = 0 \implies p(t) = b_\phi \tag{5.2}$$

To reduce sensor noise, the bias is calculated as the average of the measured angular velocity. The graphical user interface consists of a progress bar that shows the amount of samples left to measure.

5.1.2 Measuring Accelerometer Bias

Accelerometer bias is measured.

An accelerometer measures the acceleration that the sensor experiences along its axes. If we ignore sensor errors other than bias, the output $a(t)$ at the time t of a biased gyroscope is therefore given by

$$a(t) = \alpha(t) + b_\alpha \tag{5.3}$$

Where $\alpha(t)$ is the non biased acceleration and b_α is the accelerometer bias. Due to earths gravity that inflicts a force $g(t)$ on the sensor at some angle, in a resting position the accelerometer still measures a force $a_{resting}(t)$ as follows

$$a_{resting}(t) = g(t) + b_\alpha \tag{5.4}$$

To get the bias from this, for each axis $k \in \{x, y, z\}$ the application asks the user to hold the sensor unit stationary at two orientations o_1 and o_2 at times t_1 and t_2. Between t_1 and t_2 the sensor unit is rotated by $180°$ around an axis other than k. It then holds that

$$g(t_1) = -g(t_2) \tag{5.5}$$

$$\implies a_{resting}(t_1) + a_{resting}(t_2) = 2b_\alpha \tag{5.6}$$

The bias can then be calculated as

$$b_\alpha = \frac{a_{resting}(t_1) + a_{resting}(t_2)}{2} \tag{5.7}$$

To reduce sensor noise, the bias is again calculated from an average of multiple samples.

5.2 Sensor Fusion

The 3 sensor types are combined for 3D tracking.

To make sense of gyroscope, accelerometer and magnetometer data, two algorithms were implemented. One calculates the initial orientation and another one updates the orientation and position from new sensor data.

5.2.1 Calibrating Orientation

The drill model is calibrated at startup.

The `DataSet.create_acceleration_axes()`-method calculates how the coordinate system of the IMU is oriented in respect to the global coordinate system of the workshop. This information can then be used to map the sensor data to the global coordinate system. The orientation is kept in the form of a rotation matrix $M = (x, y, z) \in \mathbb{R}^{3\times3}$ where x, y, z are the global coordinate axes within the IMU coordinate system.

`DataSet.create_acceleration_axes()` takes a `DataPoint`-object during the measurement of which the drill was held in a fixed position. The magnetometer value points towards north and since the drill wasn't moved, the accelerometer value points in the opposite direction of gravity. The orientation is then derrived from the averaged accelerometer and magnetometer readings α, ω as follows:

[1] **Input:** accelerometer_value: α,
magnetometer_value: ω
[2] **Output:** axes_matrix: M
[3] $Z = \omega$; `// use -gravity as Z`
[4] $X = \alpha \times \omega$; `// use east as X`
[5] $Y = X \times Z$; `// use north as Y`
[6] **return** $[\frac{X}{|X|}, \frac{Y}{|Y|}, \frac{Z}{|Z|}]$; `// normalize vectors`

In line 3, the Z-axis is initialized to the accelerometer value. This will later help to remove gravity from the accelerometer readings, since the direction of the gravity will always be $-Z$ as long as M is rotated in unison with the sensor unit. X is set to the cross product of Z and ω in line 4. Thus it is orthogonal to the Z-axis and points east. Since X and Z are already established, we can calculate Y from their cross product. Finally, the axes are normalized to make the resulting matrix orthogonal, which is a requirement for rotation matrices.

5.2.2 Interpolating from Data

The sensor model is updated with each incoming data point.

After initializing the first model, the incoming `DataPoints` are used to continuously alter the model, keeping it as accurate as possible to the real sensor movements. The `Model.interpolate_towards()`-method of a model M takes a `DataPoint` D and calculates an updated Model M' in four consecutive steps:

- **Rotating** the IMU coordinate system within the global coordinate system.
- **Accelerating** the model by the value given by the accelerometer.
- **Translating** the model by the velocity that has been acquired through acceleration over time.
- **Corrections** to reduce drift from the real world.

In the following paragraphs, these will be discussed in more detail. Let therefore

- t_Δ be the time difference between the last update of M and the timestamp of D
- $A_{\tilde{M}} \in \mathbb{R}^{3\times3}$ be the orientation Matrix of a `Model` $\tilde{M}$
- $V_{\tilde{M}} \in \mathbb{R}^3$ be the velocity of a `Model` $\tilde{M}$
- $P_{\tilde{M}} \in \mathbb{R}^3$ be the position of a `Model` $\tilde{M}$

Rotating

The rotation of the model is updated.

The gyroscope data is used to rotate the models orientation. The gyroscope measures the angular velocity around the axes of the IMU in $^\circ/\mathrm{s}$. It outputs a rotation vector $\omega = (x, y, z)^T \in \mathbb{R}^3$ where $|\omega|$ is the angle of rotation that the accelerometer turns around the axis $\frac{1}{|\omega|}\omega$ per second[2018]. During the rotation step, the orientation matrix $A_{M'}$ of M'

is obtained by rotating the columns of A_M by $-t_\Delta\omega$. The rotation is negated, since the orientation matrix contains the position of the global coordinate axes. These rotate in the opposite direction that the IMU rotates within them.

Accelerating

The velocity of the model is updated.

The accelerometer data is used to change the models velocity. The accelerometer measures the positional acceleration within the IMU coordinate system and outputs a vector $\alpha = (x, y, z) \in \mathbb{R}^3$ in $\mathrm{m/s^2}$. The acceleration v in the global coodinate system is calculated as

$$v = Sol(A_{M'}|\alpha - g * (A_{M'})_{\_,3}) \tag{5.8}$$

Where $g \approx 9.81\,\mathrm{m/s^2}$ is the magnitude of earths gravity. As the Z axis in $A_{M'}$ is the direction of the gravity-induced force it used to substract gravity from the acceleration in 5.8. Then the velocity of M' can be calculated as

$$V_{M'} = V_M + t_\Delta * v \tag{5.9}$$

Translating

The position of the model is updated.

The Position of M′ is then obtained as

$$P_{M'} = P_M + t_\Delta * V_{M'} \tag{5.10}$$

Corrections

Sensor error is reduced.

Sensor error can introduce drift into the IMU model[2005]. To reduce the drift, the interpolation is modified as follows.

If $A_{M'}$ is not true to the orientation of the sensor unit, the gravity correction during the "Accelerating" step will introduce significant error into the velocity. Assuming that earths gravity is a major factor on the accelerometer readings, the orientation matrix $A_{M'}$ is therefore slightly rotated

in a way that the angle between the Z-axis and α decreases. The angle ϕ between the Z-axis and α is

$$\phi = arcos\left(\frac{\alpha * (A_{M'})_{\_,3}}{|\alpha| * |(A_{M'})_{\_,3}|}\right) \tag{5.11}$$

The rotation axis is the cross product of α and the Z-axis. The columns of $A_{M'}$ is rotated around this axis by an angle of up to ϕ but at maximum by a predefined constant ϕ_{max}. This constant can be chosen in a way that the Z-axis slowly converges towards gravity during operation.

If $V_{M'}$ starts drifting into some direction due to accelerometer error, $P_{M'}$ experiences the same error squared. Therefore, in addition to the orientation correction, $V_{M'}$ is reduced by a predefined constant during each interpolation.

5.3 Example Program Execution

An example program shows how to use the libraries.

The python code in Appendix A comprises an example receiver program that reads from the sensor unit, performs sensor fusion and pipes the result into a unix FIFO. This section explains the application in detail to show how the libraries can be used together. The program assumes that the debiasing applications have been executed beforehand and that the sensor unit is running and connected to the same network as the receiver system.

At the beginning, sensor unit is connected and calibrated.

The lines 13 and 14 initialize a `DataPoint`- and a `Receiver`-object. These objects will handle most of the interaction with the presented libraries. The `Receiver` is initialized with the `load_gyro_bias` and `load_accl_bias` flags to tell the constructor to load the sensor biases from disk and remove them from future `DataPoint`s when they are appended to it.
The function `receive_routine()` on lines 17-79 is the main method that will be executed when calling this script directly.
First, on line 19 it opens the unix FIFO that will receive the output of this program. Then it calls the `.connect` method of the `Receiver` on the next line. This method

sets up a TCP socket and connects it to the TCP server on the sensor unit. On the lines 30-34, the `.read()`-method of the `Receiver` is called periodically for a total of 5 seconds. This method reads samples from the TCP connection and returns a list of `DataPoint`s that are then appended to the `DataSet`. Within these 5 seconds, the real drill has to be held in a stationary position. By calling the `.calibrate()`-method of the `DataSet`, the `DataPoint`s that were received so far, are used to generate a `Model` of the sensor unit. Since the drill wasn't moved and the resulting `Model` is supposed to be used as initial model, positional velocity and position can be set to 0. The orientation is calculated as stated in section 5.2.1.

After initialisation, the sensor data is received and written to a unix FIFO.

After these initialization steps, the rest of the code runs in an infinite loop at the lines 47-77. In each iteration, first `Receiver.read()` is called and its result is again appended to the `DataSet`, but this time the `congest` flag ist set. This makes the `DataSet` apply the `Model.interpolate_towards()` method for each received `DataPoint` to update its current `Model`. The previous `Model`s are automatically saved to a list by the `DataSet`. On the lines 53-70, a second list is created that is a sublist of the first one, such that it contains just enough samples to have a sample rate as defined by the variable `PLOT_RATE`. This list is then serialized by mapping the `Model.to_json_serializable()` method over it and then using the json core library to dump the resulting dictionaries as a json-`str`. The iteration ends by writing the string to the FIFO on lines 74-76.

Chapter 6

Summary and Future Work

In this Chapter, the contributions of the thesis are summarized, and possible improvements on the approach are discussed.

6.1 Summary

In this thesis, a system for tracking movements of a power tool was created. The presented system is capable of measuring movement of a sensor unit that is mounted on a cordless drill, sending the resulting data to a computer, evaluating it and creating a 3D animation from that.

The sensor unit is firmly attached to the cordless drill and doesn't move in relation to the drill when applying manual force. This makes sure that the sensor data reflects the movements of the drill. The sensor unit uses the full $1\,\mathrm{kHz}$ data rate of the IMU sensors and its TCP connection allows to send the whole data stream to the receiver. This means that rapidly changing movement can be tracked.

The receiver can perform sensor fusion to combine the three kinds of IMU data and obtain a model of the drill move-

ment. The live plot shows that the orientation of the model closely follows the orientation of the real drill. Also, the positional movements of the model correlate with the movements of the drill, but after moving the drill for the first time since the model initialisation, the model position drifts away from the real world due to sensor error.

The provided script for generating 3D animations works as intended when using dummy data. This can be seen in the following video that was rendered from the dummy data.

File: dummy_data.avi[a]

[a] http://hci.rwth-aachen.de/public/MenkeTracking/dummy_data.avi

6.2 Future work

Future work should focus on improving the systems accuracy. Currently, the sensor error renders the positional data that the system provides useless after a short time. Approaches as in Batista et al. [2010] that aim to filter non-constant sensor error, greatly improve sensor accuracy and could improve stability of positional data. The IMU could also be combined with other sensors such as cameras[2010] to reduce the positional drift.
The sensor unit is currently fixed to the side of the drill, potentially obstructing the user. In the future, it could be integrated seamlessly into the tools. This could be done by using smaller components on a single PCB that hosts both sensors and the microcontroller. The drill battery could then be used for powering the sensor unit, making it even smaller without the need for an additional battery.

When the system delivers reliable tracking data, research should use it to automatically generate documentation. The presented technique of creating 3D animations could be extended by extracting high level information as e.g. distinction of steps within a workflow. This information could then be used to cut out less interesting parts of the process, or to move camera angles depending on the task performed.

Also, multiple tools in a single workshop could be equipped with sensors to extend the set of actions that can be tracked.

Appendix A

Example Program

```
1  import time
2  from backend.model.data import DataPoint, Model
3  from backend.model.data_set import DataSet
4  from backend.io.tcp_receive import Receiver, SAMPLE_RATE
5  import os
6  import json
7  import numpy as np
8
9  # the amount of samples to write to the FIFO per second
10 PLOT_RATE = 8
11 FIFO_PATH = "/tmp/bachelor_prototype"
12
13 data_set = DataSet(load_gyro_bias=True, load_accl_bias=True)
14 rec = Receiver()
15
16
17 def receive_routine():
18
19     fifo = os.open(FIFO_PATH, os.O_RDWR)
20
21     rec.connect()
22
23     calibration_duration = 5  # seconds
24     start_time = time.time()
25     last_time = 0
26
27     fifo = os.open(FIFO_PATH, os.O_RDWR)
```

```
28
29      # calibrate orientation
30      while time.time() - start_time < calibration_duration
31          received_data = rec.read()
32          data_set.append(received_data)
33          last_time = received_data[-1].time
34      data_set.calibrate((0, last_time))
35
36      # generator that selects a subset of the received sa
37      def selection_indices_generator():
38          res = 0
39          while True:
40              yield int(res)
41              res += SAMPLE_RATE / PLOT_RATE
42
43      selection_indices = selection_indices_generator()
44      selection_index = next(selection_indices)
45      num_samples_received = 0
46
47      while True:
48          received_data = rec.read()
49          num_samples_received += len(received_data)
50          data_set.append(received_data, congest=True)
51
52          # returns a list of samples within the last read
53          def select_all_samples():
54              nonlocal selection_index
55
56              result = []
57              while selection_index < num_samples_received
58                  result.append((data_set.story[selection_i
59                  selection_index = next(selection_indices)
60
61              return result
62
63          def serialize_sample(data):
64              model, data_point = data
65              print(data_point.current_value)
66              return json.dumps({
67                      "model": model.to_json_encodable(),
68                      "data_point": data_point.to_json_enco
69              }) + "\n"
70
```

```
71          selected_samples = select_all_samples()
72          to_pipe = list(map(serialize_sample, selected_samples))
73          to_pipe = "".join(to_pipe)
74
75          num_written = 0
76          while num_written < len(to_pipe):
77              num_written += os.write(fifo, bytes(to_pipe, encoding="UTF8"))
78
79      # will never be called but should be done in an actual application
80      os.close(fifo)
81
82
83  if __name__ == "__main__":
84      if not os.path.exists(FIFO_PATH):
85          os.mkfifo(FIFO_PATH)
86      receive_routine()
```

Bibliography

Stavros Antifakos, Florian Michahelles, and Bernt Schiele. Proactive instructions for furniture assembly. In *International Conference on Ubiquitous Computing*, pages 351–360. Springer, 2002.

Pedro Batista, Carlos Silvestre, Paulo Oliveira, and Bruno Cardeira. Accelerometer calibration and dynamic bias and gravity estimation: Analysis, design, and experimental evaluation. *IEEE transactions on control systems technology*, 19(5):1128–1137, 2010.

Timothy Campbell, Jonathan Harper, Björn Hartmann, and Eric Paulos. Towards digital apprenticeship: Wearable activity recognition in the workshop setting. Technical report, Tech. Rep. UCB/EECS-2015-172, EECS Department, University of California, Berkeley, 2015.

Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. Mixt: automatic generation of step-by-step mixed media tutorials. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 93–102, 2012.

W Flenniken, J Wall, and D Bevly. Characterization of various imu error sources and the effect on navigation performance. In *Ion Gnss*, pages 967–978, 2005.

Floraine Grabler, Maneesh Agrawala, Wilmot Li, Mira Dontcheva, and Takeo Igarashi. Generating photo manipulation tutorials by demonstration. In *ACM SIGGRAPH 2009 papers*, pages 1–9. 2009.

Chris Hide, Tom Botterill, and Marcus Andreotti. Low cost vision-aided imu for pedestrian navigation. In *2010 Ubiq-*

uitous Positioning Indoor Navigation and Location Based Service, pages 1–7. IEEE, 2010.

Sophie Leroy. Why is it so hard to do my work? the challenge of attention residue when switching between work tasks. *Organizational Behavior and Human Decision Processes*, 109(2):168–181, 2009.

Bakhtiar Mikhak, Christopher Lyon, Tim Gorton, Neil Gershenfeld, Caroline McEnnis, and Jason Taylor. Fab lab: an alternate model of ict for development. In *2nd international conference on open collaborative design for sustainable innovation*, volume 17, 2002.

Iván Sánchez Milara, Georgi V Georgiev, Jani Ylioja, Onnur Özüduru, and Jukka Riekki. " document-while-doing": a documentation tool for fab lab environments. *The Design Journal*, 22(sup1):2019–2030, 2019.

Stephen Monsell. Task switching. *Trends in cognitive sciences*, 7(3):134–140, 2003.

Kylie Peppler, Adam Maltese, Anna Keune, Stephanie Chang, Lisa Regalla, and Maker Education Initiative. Survey of makerspaces, part ii. *Open Portfolios Maker Education Initiative*, pages 1–6, 2015.

Joshua S Rubinstein, David E Meyer, and Jeffrey E Evans. Executive control of cognitive processes in task switching. *Journal of experimental psychology: human perception and performance*, 27(4):763, 2001.

Eldon Schoop, Michelle Nguyen, Daniel Lim, Valkyrie Savage, Sean Follmer, and Björn Hartmann. Drill sergeant: Supporting physical construction projects through an ecosystem of augmented tools. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 1607–1614. ACM, 2016.

Sara Stančin and Sašo Tomažič. On the interpretation of 3d gyroscope measurements. *Journal of Sensors*, 2018, 2018.

Tiffany Tseng. Making make-throughs: Supporting young makers sharing design process. *Proceedings of Fablearn*, 2015.

Tiffany Tseng and Mitchel Resnick. Product versus process: representing and appropriating diy projects online. In *Proceedings of the 2014 conference on Designing interactive systems*, pages 425–428, 2014.

Tiffany Tseng and Mitchel Resnick. Spin: Examining the role of engagement, integration, and modularity in supporting youth creating documentation. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 996–1007, 2016.

Jamie A Ward, Paul Lukowicz, Gerhard Troster, and Thad E Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE transactions on pattern analysis and machine intelligence*, 28(10): 1553–1567, 2006.

Greg Welch and Eric Foxlin. Motion tracking: No silver bullet, but a respectable arsenal. *IEEE Computer graphics and Applications*, 22(6):24–38, 2002.

John Zimmerman, Erik Stolterman, and Jodi Forlizzi. An analysis and critique of research through design: towards a formalization of a research approach. In *proceedings of the 8th ACM conference on designing interactive systems*, pages 310–319, 2010.

Typeset March 15, 2020