HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

Fabian Brune

# Measurement Framework for Tracking Ride Characteristics of a Bicycle in Outdoor Experiments

Bachelor Thesis in Computer Science

28 February 2020

Please cite as:
Fabian Brune, “Measurement Framework for Tracking Ride Characteristics of a Bicycle in Outdoor Experiments,” Bachelor Thesis (Bachelorarbeit), Heinz Nixdorf Institute, Paderborn University, Germany, February 2020.


Distributed Embedded Systems (CCS Labs)
Heinz Nixdorf Institute, Paderborn University, Germany

Fürstenallee 11 · 33102 Paderborn · Germany

http://www.ccs-labs.org/

# Measurement Framework for Tracking Ride Characteristics of a Bicycle in Outdoor Experiments

Bachelor Thesis in Computer Science

vorgelegt von

**Fabian Brune**

geb. am 28. September 1993
in Paderborn

angefertigt in der Fachgruppe

**Distributed Embedded Systems**
**(CCS Labs)**

**Heinz Nixdorf Institut**
**Universität Paderborn**

Betreuer: **Julian Heinovski**
Gutachter: **Falko Dressler**
**Marco Platzner**

Abgabe der Arbeit: **28. Februar 2020**

## Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde.
Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

## Declaration

I declare that the work is entirely my own and was produced with no assistance from third parties.
I certify that the work has not been submitted in the same or any similar form for assessment to any other examining body and all references, direct and indirect, are indicated as such and have been cited accordingly.

(Fabian Brune)
Paderborn, 28 February 2020

# Abstract

Improving traffic security for Vulnerable Road Users (VRUs), such as cyclists, can avoid accidents and injuries, or even save lives. However, testing Advanced Driver Assistance Systems (ADASs) for bicycles in real traffic is dangerous. Hence, simulators have been developed that enable testing ADASs without putting the driver at risk. For the validation of such simulators a bicycle can be used that is capable of recording real world data in outdoor experiments. Existing instrumented bicycles lack expandability, accuracy or validation of sensors, they use only few sensors or are expensive. Within this thesis I explain my developed system, the Real World Cycling Environment (RCE), an expandable framework capable of tracking several bicycle parameters with its sensors that have been validated and/or calibrated accurately. It is based on a Raspberry Pi and can be built for about 100 Euro.[1] An external GPS receiver and a camera are used to record GPS track and video, respectively. The developed sensors record speed, steering angle, and brake application. The speed sensor measures the rotational speed of the back wheel with nine signals per revolution and considers the tire pressure to calculate the actual circumference of the wheel for speed computation. The steering angle sensor has been validated with a testing station, specifically designed for the bicycle that forms the base of the system. It has a Mean Absolute Error (MAE) of 0.3°. An external high quality GPS device is used to record a GPS track. A comparison of recorded tracks to map data, as well as a comparison to two smartphones that recorded simultaneously, yielded good results. The brake sensors measure the brake handle application that can be related to the actual brake-caused deceleration. Overall, the Raspberry Pi's resources are exploited to a low degree. With all sensors at their highest frequency and precision, the CPU usage (average) was below 39%, memory usage (maximum) and current power consumption (average) were below 26%. The recording software is designed extendable. Implementing additional sensors is straightforward. Furthermore, the developed software offers post processing with the data acquisition unit or with a different computer, as well as a simple review tool for trace analysis.

[1] All hardware excluding the GPS receiver, but including the Raspberry Pi, costs about 100 Euro.

# Kurzfassung

Eine Erhöhung der Verkehrssicherheit von verwundbaren Verkehrsteilnehmern, wie z.B. Radfahrern, kann Unfälle oder Verletzungen verhindern oder sogar Leben retten. Assistenzsysteme für Radfahrer im normalen Straßenverkehr zu testen ist gefährlich, weshalb Simulatoren entwickelt wurden, die die Entwicklung von Assistenzsystemen erlauben, ohne den Radfahrer einer Gefahr auszusetzen. Zur Validierung solcher Simulatoren kann ein mit Sensoren ausgestattetes Fahrrad genutzt werden, das draußen gefahren werden kann. Bereits existierende Fahrräder dieser Art sind schlecht erweiterbar, nutzen nur wenige Sensoren, sind ungenau oder wurden nicht validiert. Das entwickelte System, das Real World Cycling Environment (RCE), ist ein erweiterbares Framework mit verschiedenen Sensoren, die validiert oder sehr genau kalibriert wurden. Kernelement des Frameworks, das für etwa 100 Euro gebaut werden kann[2], bildet ein Raspberry Pi. Ein externer GPS Empfänger sowie eine Kamera nehmen einen GPS Track bzw. ein Video der Fahrt auf. Die Sensorik erfasst Geschwindigkeit, Lenkwinkel und Stellung des Bremshebels. Die Geschwindigkeit wird mit neun Magneten unter Beachtung des Radumfangs und Reifendrucks durch Umrechnung der Rotationsgeschwindigkeit des Hinterrads ermittelt. Der Lenkwinkelsensor wurde mit einer speziell für das verwendete Fahrrad entwickelten Test-Station validiert und hat einen durchschnittlichen Fehler von 0.3°. Das externe GPS Gerät erreichte im Vergleich mit zwei von Smartphones aufgezeichneten GPS Spuren eine deutlich höhere Genauigkeit und wurde zusätzlich mit diversem Kartenmaterial abgeglichen. Für die Bremssensoren wurde eine Verbindung zwischen dem Bremshebelzug und der durch die Bremsen verursachten Verlangsamung hergestellt. Die Evaluierung des Systems hat gezeigt, dass die Resourcen des Raspberry Pi nur zu einem geringen Anteil genutzt wurden. Bei höchsten Einstellungen lag die durchschnittliche CPU-Auslastung unter 39%, Speichernutzung und Stromaufnahme lagen jeweils unter 26%. Zusammen mit der erweiterbar angelegten Aufnahmesoftware kann das System daher gut ergänzt werden. Die Nachbearbeitung der aufgezeichneten Daten kann auf dem Raspberry Pi oder einem anderen Computer durchgeführt werden. Ein einfaches Analysetool ermöglicht das Ansehen der aufgezeichneten Daten.

[2] Gesamtkosten inklusive sämtlicher Hardware außer dem GPS Empfänger.

# Contents

# Chapter 1

# Introduction

Only 8 % of road fatalities happen on motorways, while most accidents happen on rural (55 %) or urban (37 %) roads [1] where Vulnerable Road Users (VRUs) are traveling. Advanced Driver Assistance Systems (ADASs) have been developed for motorized vehicles, primarily for cars, to reduce the number of accidents, but VRUs have not been in the focus of ADASs yet. To investigate the impact of ADASs on the safety of cyclists, Heinovski et al. [2] developed the Virtual Cycling Environment (VCE), a VR-based cycling simulator which integrates a real bicycle into a 3D simulation environment. Without putting the driver at any risk, in this virtual environment it is possible to test ADASs with real test subjects in different traffic situations in an arbitrary setting, e.g., an urban intersection with blocked lines of sight due to building density. Recently, the VCE has been used for conducting experiments to investigate the visual attention of cyclists in a human-in-the-loop setup [3]. During an experiment, the VCE can record movement traces of the cyclist consisting of position, speed, acceleration, and steering angle of the bicycle. These traces can be used for later analysis, and additionally, they can serve as a base for creating models for realistic cycling behavior for large-scale simulation studies. Due to the real bicycle, virtual reality, and realistic simulation of traffic, the VCE is very realistic. However, traces have not yet been compared to real world traces of a bicycle in similar situations. Furthermore, real world bicycle traces could be used to validate existing bicycle driver models as in [4] and serve as a resource for developing driver models. Methodology for recording such real world data in outdoor experiments, especially recording a variety of bicycle parameters, is missing, as there is a lack in functionality to record such traces.

To approach this problem, I developed the RCE, a measurement framework for a real bicycle, that is able to record traces comparable to the traces recorded by the VCE. The framework consists of sensors for cycling speed, steering angle, and GPS data. In addition to that, I built a brake sensor that can be used on almost any

bike, thus also on the VCE. A brake application sensor can be of great use when, e.g., investigating the reaction time of a cyclist with or without a specific ADAS. For later analysis or review, the RCE can record a video of the ride showing the rider's view. The framework is expandable and allows adding more sensors or devices in the future. Furthermore, it is possible to use just a selection of the available sensors, if an experiment requires only a selection of the sensors. The minimum target frequency for each sensor is 10 Hz to allow comparability to the VCE traces that are also recorded at this frequency.

In related work, some bicycle parameter sensors have been validated with poor hardware or expensive hardware that wasn't used to its full potential [5]–[7]. Some developed sensors have not been validated at all, or no information was given on the validation. To ensure realism and accuracy of data as well as usability in experiments, I validated each component of the RCE and evaluated the system. For the speed sensor, I measured the exact circumference of the wheel that is used for speed computation in consideration of the tire pressure. To avoid general errors, I also compared the RCE speed data to two reference systems, the onboard computer of the bicycle itself, and the GPS determined speed. The steering angle sensor is build similar to previously developed steering angle sensors of Kooijman, Schwab, and Meijaard [7] and Moore [8]. It is based on a potentiometer, but avoids the main drawbacks by design. The brake sensors can be used to relate brake handle application to brake-caused deceleration. However, the main advantage of calibration and validation was that the ground truth is not always known, e.g., the actual speed, or requires expensive hardware to be determined accurately. My evaluation includes an estimation of the efficiency and resource usage of the system.

# Chapter 2

# Technical Background

This chapter covers the technical background of the hardware used in the developed system and in related work. I will also give some information about the electronic components and wiring of sensors that can be used for measurements.

## 2.1 Raspberry Pi

A Raspberry Pi is a single-board general purpose computer that can be used in many applications, e.g., for education or in robotics. The version that I deploy in this thesis is the Raspberry Pi 3B+[3]. It runs a 4-core 1.4 GHz CPU with 1 GB RAM and has an onboard wireless network and Bluetooth 4.2 card available. External devices such as sensors or peripheral devices can, among others, be connected via the HDMI port, four USB ports, and 40 General Purpose Input Output (GPIO) pins. A Raspberry Pi Camera can be connected via a special camera port on the Pi and is capable of recording photos or videos (up to 30fps at 1080p). The single-board computer is powered by a 5 V/2.5 A Direct Current (DC) power input that can be implemented with a portable charger, which enables mobile applications. The Raspberry Pi can be run with an Operating System (OS) called Raspbian, which is a Debian-based Linux derivative and also available as a Desktop version. Hence, developing directly on the Raspberry Pi is possible. Most GPIO pins can be addressed programmatically to, e.g., control custom made hardware like input or output devices, or read out values from sensors, some of which are explained in the following sections. In this thesis, whenever GPIO pins are mentioned, I refer to the programmable pins. The non-programmable pins are 5.0 V, 3.3 V and Ground (GND). Programming can, among others, be done with Python[4] and is highly supported by gpiozero[5], a Python

[3] https://www.raspberrypi.org/
[4] https://www.python.org/
[5] https://gpiozero.readthedocs.io/en/stable/

library that simplifies the use of GPIO pins and has many components already built in. The Raspbian OS already has a Secure Shell (SSH) server installed that only needs to be enabled via the Raspberry Pi configuration tool. An SSH connection can be used to access the Raspberry Pi from a different device via, e.g., a Wireless Local Area Network (WLAN). The WLAN card can be setup as a Wireless Access Point (WAP), i.e., the Raspberry Pi can establish its own WLAN. Hence, even in outdoor environments with no other WLAN available, an SSH connection to, e.g., a smartphone, can be established. In consequence, connecting keyboard, mouse, and a display that typically needs a fixed power supply is not required.

## 2.2 Fundamental Electronic Components

Three fundamental electronic components that are frequently used in electronic circuits are resistors, Light Emitting Diodes (LEDs) and capacitors. All are passive electronic components with two terminals. The electronic symbols are shown in Figure 2.1.

A resistor implements electrical resistance and can be used to limit the current flow or drop the running voltage of other electronic components, e.g., LEDs. The electrical resistance is specified by Ohm's law:[6]

$$I = \frac{V}{R} \Leftrightarrow V = IR \Leftrightarrow R = \frac{V}{I}, \quad \text{unit: } \Omega \text{ (Ohm)} \tag{2.1}$$

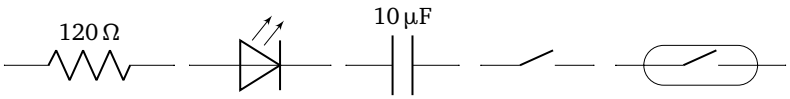
LEDs run with a specific voltage level and have an upper limit for the current flow. Typical specifications are, e.g., 2.1 V at a maximum current flow of 20 mA. Example: If using with a 3.3 V power supply and limiting the current to 5 mA, the resistor drops the voltage by 1.2 V, and hence, the resistance needs to be:

$$R = \frac{V}{I} = \frac{1.2\,\text{V}}{5\,\text{mA}} = 240\,\Omega \tag{2.2}$$

A capacitor can store a specific amount of electrical energy by charging, but does not allow current to flow through. It can function as a buffer for voltage peaks or fluctuating voltage. The capacitance $C$ can be calculated as:[7]

$$C = \frac{Q}{V}, \text{ where } Q = I \cdot t, \quad \text{unit for capacitance: F (Farad)} \tag{2.3}$$

[6] I: current, R: resistance, V: voltage; in Germany the typically used symbol for voltage is U
[7] C: capacitance, Q: charge, V: voltage, I: current, t: time




**(a)** Resistor **(b)** LED **(c)** Capacitor **(d)** Switch **(e)** Reed switch

**Figure 2.1** – Electronic symbols for resistor, LED, capacitor, switch and Reed switch.

## 2.3 Switches

Switches or buttons can be realized with very simple circuits that consist of only a few resistors and the switch itself. Technically, it is also possible to connect the switch directly to one of the GPIO pins and a GND pin, but this results in an undefined logical state of the GPIO pin when the switch is open. Due to this undefined state, it is better to use a pull-up resistor to define the voltage during open state of the switch. Additionally, a second resistor in front of the GPIO pin can be used to avoid short circuits when accidentally assigning the GPIO pin as an output pin while programming. The wiring of pull-up resistor and GPIO protection resistor is shown in Figure 2.2. As this wiring contains no consumer that requires a specific current flow, the resistors can be chosen with high resistance, e.g., 10 kΩ. Reed switches are switches that close when a magnet is nearby. The electric symbol of switch and Reed switch are shown in Figure 2.1d and Figure 2.1e. All switches have a specific bounce time that determines how long it takes until the electrical potential change from low to high or vice versa. By setting the bounce time programmatically, an event is triggered when a switch remained in the new state for at least the given time.

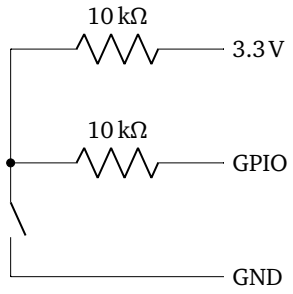



**Figure 2.2** – Pull-up resistor: The voltage level gets pulled up to 3.3 V, when the switch is opened. When closed, the voltage gets pulled down, as GND cancels the connection to 3.3 V. The resistor at the 3.3 V prevents short circuits.

## 2.4 Analog Signals

To input or output any analog signals with a Raspberry Pi, an Analog Digital Converter (ADC) is needed. It converts both digital input to analog output and analog input to digital output. The former can be used, e.g., when controlling a device with a specific voltage, the latter when receiving a specific voltage. An example for an ADC is the MCP3008, an Integrated Circuit (IC) with a digital resolution of 10bit, i.e., it converts an analog voltage level between GND and a reference voltage $V_{REF}$ to a 10-bit number. The electronic symbol is shown in Figure 2.3, more information on the MCP3008 is given in the data sheet.[8] It is also supported by gpiozero. The communication between Raspberry Pi and MCP3008 can be established via Serial Peripheral Interface (SPI), a serial interface which can be used for intra-board communication. Two components that output analog signals in the form of voltage levels are potentiometer and hall sensor, which will be introduced in the next paragraphs.

A potentiometer is a component with three terminals that divides the voltage between input and output and can be seen as an adjustable resistor. Hereafter,

[8] https://www.microchip.com/wwwproducts/en/en010530

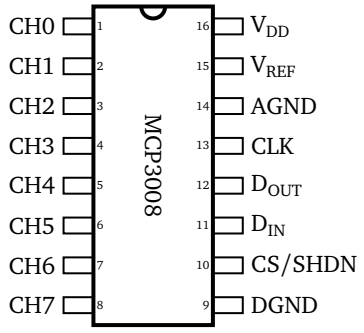
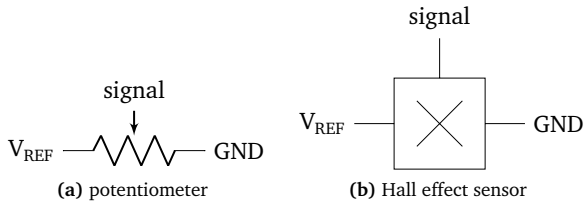



**Figure 2.3** – Pin description for MCP3008:
CH0-CH7: analog input, DGND: digital ground, CS/SHDN: chip select/shutdown input, $D_{IN}/D_{OUT}$: serial digital input/output, CLK: serial clock, AGND: analog ground, $V_{REF}$: reference voltage input, $V_{DD}$: supply voltage.




**(a)** potentiometer **(b)** Hall effect sensor

**Figure 2.4** – Electronic symbols for potentiometer and Hall effect sensor.

I will refer to the pins of a potentiometer as $V_{REF}$, signal, and GND, as shown in Figure 2.4a. Typical potentiometers consist of a shaft that by turning changes the resistance between input, signal, and output, and thus, changes the voltage level of the signal, while the total resistance between input and output remains constant.

Similar to a potentiometer, a Hall effect sensor has three terminals. It divides the voltage depending on the current magnetic field. Hence, the voltage level of the signal changes between $V_{REF}$ (reference voltage) and GND when providing a magnetic field with, e.g., a permanent magnet. Typical hardware pins are ordered in a row: $V_{REF}$, GND and signal. The most common electrical symbol is ordered like a potentiometer, though, as can be seen in Figure 2.4b. When providing no magnetic field, the signal voltage is around the average of $V_{REF}$ and GND, but might be influenced by the environment or earth's magnetic field depending on the location and the sensitivity of the Hall effect sensor.

## 2.5 GPS Receiver

There exists a variety of GPS receivers that can be used for position tracking or speed calculation. Many if not all modern smartphones have a GPS receiver built in. But the accuracy of smartphone GPS is relatively low compared to external GPS receivers [9]. The external GPS receiver that I am using for this thesis is the ublox EVK-7N[9]. It can be connected to a computer via a USB cable to read the current measurement or adjust parameters, such as frequency or whether to enable advanced accuracy by using augmentation systems. When operating on a Windows machine, the above functionality is available in the u-center software[10]. On Linux distributions the u-center software is not available. Hence, reading the current GPS data has to be done manually or programmatically via `/dev/serial`. As GPS has evolved over a long time since its launch in the 1970s [10], several improvements have been developed [11], [12].

According to Oxley [11], Differential GPS (DGPS) improves the default *absolute positioning* by the use of ground stations as reference positions. Each station receives the messages from the satellites and calculates the difference between the actual (known) position and the position determined by the messages. The computed difference is then broadcast from the stationary reference receiver. Any DGPS-enabled receiver can use the difference to correct the position and avoid errors due to, e.g., ionospheric and tropospheric effects, which affect larger regions and hence are almost equal for an individual receiver and the closest reference station [11].

[9] ublox EVK-7N: https://www.u-blox.com/en/product/evk-7
[10] u-center: https://www.u-blox.com/en/product/u-center

Satellite-Based Augmentation Systems (SBASs)[11] add correction messages to the GPS signals that improve the accuracy even further by using additional satellites [12]. The non-GPS satellites send GPS-compatible signals that can be received by the GPS receivers. SBASs are available in the following regions: Europe (EGNOS), India (GAGAN), Japan (MSAS) and North America (WAAS). In my thesis, I will refer to SBASs as WAAS or EGNOS, as they are the most mentioned in the literature.

[11] also known as Wide-Area Differential GPS (WADGPS)

# Chapter 3

# Related Work

In this chapter I will give an overview of related work and methods that have been used to measure ride characteristics of a bicycle and to conduct related experiments. Methods used vary a lot as the purposes of developing measurement devices vary from tracking bicycle lane positioning [9] over measuring the pedaling force and speed of a bicycle to estimate the calorie consumption while cycling [5] to a fully instrumented bicycle [8], [13]. Furthermore the desired accuracy in each case varies a lot and thereby influences the used technology. For example, using GPS to measure the speed or driven distance of a bicycle [14] requires high accuracy while using GPS to measure the choice of specific bicycle routes in a city or region [15] does not. As most researchers focus on a specific value, e.g., only the speed of a bicycle, I separated this chapter by devices and sensors. First of all, I will start with the core component of all sensor technology, the data acquisition unit. The sensors covered in the subsequent sections are for measuring speed, steering angle, GPS, and the brake application. Finally, I will give some examples for smartphone-based measurement systems and their applications.

## 3.1 Data Acquisition Unit

Data acquisition units in bicycle experiments vary almost as much as the applications of bicycle sensor technology. In the next paragraphs I will give some examples of how data acquisition can be realized.

First, the most simple way of recording GPS positioning or GPS velocity data is by using smartphone GPS (for average or low accuracy recordings). In this case the data acquisition unit is already at hand, the smartphone itself. It can record and store the track [15] or directly upload the recorded data to a server, which has been done by Lindsey et al. [9]. They also extended an existing smartphone tracking app to establish a Bluetooth connection between a smartphone and an external

Bluetooth enabled GPS receiver, to store and upload GPS data [9]. Different GPS devices and/or smartphones were also used by Heesch and Langdon [16] and Bíl, Andrášik, and Kubeček [17]. A typical smartphone as well as the chosen external GPS device, have enough internal storage and battery power to allow experiments of several hours.

When it comes to sensor technology, it sometimes is not necessary to store the recorded values, but instead directly process the information. Chen et al. [5] made use of a microprocessor to determine the speed of a bicycle depending on the turning frequency of the rear wheel, and accumulated the total driven distance and total burned caloric energy. The current values were shown on a display, but it wasn't required to store previous values.

In [8] (Delft Bicycle) and [18] a stationary laptop with a receiver for wireless communication was available, as experiments took place within a limited area. The measurement system on the bicycle was equipped with a WLAN transmitter and a radio-telemeter, respectively, and data was immediately sent to and stored on the laptop.

Another option is to have a data acquisition unit on the bicycle itself. In this case, the system needs to be portable, and thus, be battery or bicycle dynamo powered, have a high data writing speed, and it has to be (more or less) lightweight. With a laptop mounted on the rear rack, e.g., a GPS device can directly be connected via USB [14]. An USB ADC or a data acquisition unit can be used to connect sensors to a laptop [19], [7], [20]. Tanaka et al. [6] made use of an STT-MRAM which is a low power, fast writing RAM device. They powered it with a hub dynamo, while at the same time measuring the speed with the dynamo output voltage. Wehrbein [21] used a handheld calculator instead of a laptop, to measure the bicycle speed using a Reed switch. As computers are much smaller and lighter nowadays, it is also possible to use a single-board computer as data acquisition unit. Dozza et al. [13] used a single-board computer from Phidgets Inc. which works on a 400 MHz CPU and has six USB ports and several digital and analog inputs available. They were able to record data at a frequency of 100 Hz and additionally stored GPS position and an image of the current scenery once per second with an external GPS device and a camera, both connected via USB [13]. The developed device was intended to record VRUs, e.g., pedestrians, in road traffic.

## 3.2 Speed

For measuring the speed of a bicycle, three methods are mainly used. First, calculating the speed between measured points of a GPS track. Second, using magnets and a magnetic sensor to determine the rotational speed of the wheel. Knowing the

circumference of the wheel, the forward speed can be computed. And third, using the linearity of turning speed to the output voltage of a generator or DC motor. An alternative method uses the lean angle of a single-track vehicle in a continuous circular motion to compute the speed [22]. Of course, it is also possible to manually take the time a cyclist needs to pass a section of known length and thereby determine the average speed in this section [23].

### 3.2.1 GPS Speed

GPS tracks basically consist of a sequence of points that contain information about latitude, longitude, altitude, and time. Thereby, the speed between two points can be calculated. Arnesen, Malmin, and Dahl [15] used a smartphone app to record GPS tracks of cyclists to build a prediction model of the bicycle speed. They recorded GPS data at a frequency of 1 Hz. A higher frequency or perfectly accurate speed data wasn't required as it was more important to determine the average traveling speed on different routes or road segments. Investigating the accuracy of GPS speed measurements on a bicycle, Witte and Wilson [18] have shown that the GPS speed was within 0.2 m/s of the actual speed in 45 % of the time and within 0.4 m/s in 57 % for non-differential GPS. Under similar conditions, but using WAAS-enabled GPS devices, the calculated speed was within 0.2 m/s of the actual speed in 64 % and within 0.4 m/s in 82 % of the time, as Witte and Wilson [14] stated in continuing research. However, most smartphones do not seem to use WAAS-enabled GPS on system layer, but there exist some apps that use additional data for error correction. Information about whether or not a smartphone uses WAAS-enabled GPS is hard to find, because manufacturers do not give much information about this.

### 3.2.2 Magnetic Speed Sensor

The most common way to measure the speed of a bicycle is to attach a magnet to one of the wheels or spokes and attach a magnetic switch to the frame, so every time the magnet passes the switch an electric signal is induced. This method is also used by standard bicycle speedometers. Using just one magnet leads to a very low signal frequency as there is only one signal per one turn of the wheel. This problem is particularly important at low speeds.

To address this problem, Chen et al. [5] used six magnets attached to the spokes of a bicycle and computed the speed using the circumference of the wheel. In a similar way, they measured the pedaling frequency, but with just one magnet on one of the pedal cranks. Combined with the also measured angle of the pedal, they used the data to compute a calorie consumption. To validate their speed sensor, they compared the measured values to the output of a conventional bicycle speedometer that is working with one magnet. Their developed system was mounted on the back

wheel, while the reference speedometer was mounted on the front wheel. They conducted eleven tests at speeds between 5 km/h and 15 km/h and calculated an accuracy of 96 %, but they didn't describe how the conventional speedometer was read out, and no details on the experiment conditions were given.

Wehrbein [21] started developing a speed sensor with a plastic spoke protector with six magnets on it, but found that it was very difficult to position the magnets accurately. He noticed a specific pattern in the recorded data due to the manual and thus slightly inaccurate positioning and assumed an influence by some magnets being stronger or weaker than the others or some magnets being slightly ahead/behind their should-be position [21]. Time was measured at the entering point of the magnet, thus, a stronger magnet behaves like a magnet that is slightly ahead the should-be position. Wehrbein [21] changed the design to nine magnets attached to the spokes in consideration of the symmetry of a standard bicycle wheel with 36 spokes that are arranged in groups of four spokes creating a nine-fold symmetry. To validate the system he compared the measured speed during non-accelerated idle cycle to a falling curve assuming constant rolling friction. He found that the system worked well, as the curve fits the measured values, but he did not give any statement of accuracy. The storage of his developed system is sufficient for storing 12 000 data points, which corresponds to three hours of recording at 1 Hz [21].

A related design was created by Kooijman, Schwab, and Meijaard [7] and is based on a magnetic ring that is attached to the wheel and has ten magnets in it. Details on the accuracy or validation of the speedometer were not given, since it is a manufactured model and was assumed to be working accurate. The focus of their work was on validating a bicycle model and not the sensors themselves.

### 3.2.3 Generator

Another approach to measure the speed of a bicycle wheel is by using a generator, e.g., a dynamo, to determine the rotational speed of the wheel. The output voltage of a hub dynamo is almost linear to the turning speed at no load [8], [24]. Hence, it can be used to measure the turning speed of the wheel [6], [24], [25]. This also applies for electric motors or generators like side-wall dynamos.

Tanaka et al. [6] used a hub dynamo with 15 Alternating Current (AC) cycles per one revolution of the wheel to calculate the speed. They validated the system with a speed control motor that was placed under the wheel to turn it at the desired speed. For comparison to a standard speedometer, they placed its display in front of a digital camera that recorded the display during an experiment. This showed that the hub dynamo provided very accurate speed data and the standard speedometer did not, especially at low speeds [6]. However, the speed control motor was not used for the validation directly.

Moore [8] mounted a small DC motor with a roller wheel of known size against the tread surface of the wheel. This results in an even higher number of AC cycles per one rotation of the wheel, as the roller wheel is much smaller than the bicycle wheel and thus turns multiple times per one rotation of the wheel. The system was not validated but calibrated in an accurate way. Moore [8] measured both, the output voltage and the actual Rounds Per Minute (RPM), with a digital multimeter and a digital tachometer, respectively. As expected, the relationship of voltage and rotational speed was almost linear. The author used linear regression to fit a function on it [8]. An accuracy value, however, was not stated. Hence, I calculated the accuracy of this function using the voltages and RPM that were given in [8]. With $m = 456.3862$, $b = -1.2846$ [8], the linear function $r(v)$ RPM depending on the voltage, can be written as:

$$r(v) = m \cdot v + b \tag{3.1}$$

With this function I calculated the RPM for each voltage that would be output by the system in an experiment, and compared it to the corresponding RPM value (ground truth). The mean deviation was only 0.45 %, assuming that the DC motor always outputs the same voltage when turned at the same speed again.

## 3.3 Steering Angle

Most mechanisms to measure the steering angle of a bicycle are based on comparing the angle of the steering shaft to that of the main frame. This can mainly be done by either manually reading out the angle from a scale or by using a potentiometer and converting the measured value to the corresponding angle. An alternative approach based on Inertial Measurement Units (IMUs) makes use of several gyroscopes and accelerometers to obtain information about the steering angle of the bicycle [19].

### 3.3.1 Manually Reading

For manually reading the steering angle some sort of display apparatus or graduated disk is needed that shows the current steering angle. An early realization was done by Wilson-Jones [26] in 1951. He firmly connected a steering angle indicator that can hold a movable graduated disk in it, to the mainframe of a motor cycle in front of the handlebar. The disk was then connected to the handlebar to turn contiguously with the handlebar. A pointer on the steering angle indicator shows the current steering angle that is read by the rider of the motor cycle. By this method only single measurements are possible as the rider can not look at the indicator and memorize the values all the time. Fu [27] also used a moving disk that is mounted to the

mainframe in front of the handlebar to measure the steering angle during steady turning, but he did not state exactly *how* it is measured, though.

### 3.3.2 Potentiometer

Moore [8] built several potentiometer-based steering angle sensors that connect sensor axis and steering axis via a gear belt. He mounted the potentiometer to the mainframe and added a gear wheel of suitable size to the potentiometer to transmit a steering angle of ±45° to the full range of the potentiometer of ±168°. In Moore's[8] first design the translation was realized with a cord belt. He then changed to a timing belt due to belt slip, but even after the change he stated that he wasn't "100 % [sure] that belt slip did not occur"[8].

In the design of Kooijman, Schwab, and Meijaard [7] the potentiometer was placed above the steering shaft on a common axis. To avoid damage due to a possible misalignment of the axes, they were connected via a flexible coupling which led to minimal play [7]. The movement of the handlebar was restricted to ±30° by the mount of the potentiometer. But the potentiometer had a working angle of 358°, thus the output range of the signal was far from being exploited. However, both designs yielded good results and were applied to several bikes [8], [7]. For calibration and to ensure accurate data, Moore [8] added a graduated disk to the front rod of the bicycle. The disk turned together with the handlebar and showed different angles on a mark on the mainframe. By measuring the angle and comparing it to the output voltage of the potentiometer, he calibrated the sensor prior to each experiment [8].

## 3.4 Localization/GPS

The most common way to locate an object on the earth is using the Global Positioning System (GPS), because the signal is available for free at any time of the day at any open air location, and GPS devices are affordable. The applications for GPS are widely spread, as it can be used to determine position or speed of a moving object, animal or person. Studies on the accuracy of GPS, however, come to different results that either show a sufficient accuracy for a specific task or point out the limitations of GPS-based measurements.

One alternative to GPS has been investigated by Dardari et al. [28]. They placed reference nodes on an intersection and on surrounding vehicles and thereby estimated the position of several tags on a bicycle. This technology reached a localization error of less than 50 cm in both static and dynamic experiments [28]. The downside of this non-autonomous system is that it is only available, if the intersection and the surrounding vehicles are equipped with anchors.

As stated in Section 3.2.1, GPS can be used to measure the speed with a reasonable accuracy [14]. According to Lindsey et al. [9] it is yet not possible to determine which bicycle lane a cyclist is using. They used a smartphone GPS and a higher-quality external GPS device that is capable of receiving WAAS corrections. 40 experiments on open roads and corridors with high building density were conducted to determine the accuracy. The mean difference between GPS trace and the corridor centerlines in the densest area was more than 11 m for the GPS device (>16 m for smartphone GPS) and on a bridge with no obstructing buildings the deviation was still 1.76 m (5.86 m for smartphone GPS) [9]. Statistical significance was shown with one-sided paired t-tests between GPS device and smartphone GPS [9]. Albeit, it might be possible that the cyclists did not cycle in the exact centerline of the corridor, as, e.g., driving in an *exact* straight line is not very easy. Hence, light swerving could be a reason for these partly high deviations from the centerline of the corridor.

Heesch and Langdon [16] used smartphone GPS to track cyclists' route choices and create a heat map of the traffic density before and after structural changes in the infrastructure. They experienced errors in the measurements that go back to the lack of accuracy in smartphone GPS and led to mismatching some signals to wrong segments of the road network. Best results were reached on segments with a distance of at least 20 m, which made an assignment to this segment easier.

Bíl, Andrášik, and Kubeček [17] and Joo and Oh [29] investigated the comfort of cycling tracks by combining the GPS position with the intensity of vibrations and the subjective feelings of the cyclists. They introduced an objective rating system that can be used for comparison of different surface types or can help to improve bicycle lanes. The measurement frequency was 1 Hz in the setup of Joo and Oh [29].

## 3.5 Brake

There are many applications for brake sensors on a bicycle. They can be used to measure the reaction of the rider, improve the security, or control the engine of an Electrical Bike (e-bike), so it does not accelerate during braking. In this section, two common brake sensor designs are presented.

A bicycle brake sensor can be used to control a rear stop light to indicate the upcoming speed reduction to succeeding vehicles. In this case typically only an on/off function is needed, which has been designed with a contact switch by Wang et al. [30].

If the intensity of the brake interaction is of interest, a floating value is needed though. Tsai [31] introduced a design based on a hall sensor and a magnet that is moving towards or away from the Hall sensor while applying the brake. In this way it is possible to return a percentage value depending on how far the brake handle is

pulled. Several e-bikes and Traditional Bikes (t-bikes) were equipped with brake force sensors (among other sensors) by Dozza, Bianchi Piccinini, and Werneke [32] and Huertas Leyva, Dozza, and Baldanzini [33]. A precise description on how the sensors on the t-bikes work wasn't given, but a recording frequency of 100 Hz was stated. On the e-bikes, they additionally read out the built-in sensors of the e-bike that cause the motor to stop accelerating when either brake is applied.

## 3.6 Smartphone-based Systems

There is a number of smartphone-based approaches that study the choice of routes or record cycling behavior. Applications for this are traffic analysis and improvements of bicycle facilities, as well as warning systems for individual cyclists. Zang et al. [34] investigated the surface roughness of pedestrian and bicycles lanes. Since heavy vehicles can not be used on such roads, they attached a smartphone to a bicycle and implemented an app that is recording acceleration data, as well as GPS position. From the recorded data, a roughness index for different parts of the road network is computed [34].

Similar technology was used by Usami et al. [35]. However, they implemented a bicycle behavior recognition method to identify dangerous riding behavior. The data of acceleration and gyro sensors in a smartphone were processed with a machine learning algorithm. They were able to identify 4 groups of movement, i.e., stopping, running straight, turning left, and turning right [35]. Even statements about the intensity of, e.g., the turning movements, were possible but they did not translate the measured values to an actual steering angle. Difficulties originated from drift and noises that had to be filtered out [35]. A comparison of, e.g., a steering angle to the real steering angle, was not made, though, since the focus was on recognition of the four movement groups instead on measuring actual values. Another warning system that combines acceleration and GPS data measured with a smartphone was implemented by Gu et al. [36]. Their system displays instant warnings to the cyclist in dangerous situations, i.e., stand riding, lane weaving (swerving), and wrong-way riding [36].

Two systems, made for large scale route choice and bicycle traffic analysis, were developed by Hudson et al. [37] and Charlton et al. [38]. The developed smartphone app *CycleTracks* was designed to be power efficient and easy to use, i.e., only one "tap" is needed to start or stop a recording, as these were determined as the key requirements for collecting crowd sourced data Charlton et al. [38]. The app collected GPS information, only. A similar approach was conducted by Heesch and Langdon [16] who investigated changes in route choice and cycling behavior after

infrastructural changes. They also identified the number of app users as a critical component for robust studies.

However, none of the smartphone based solutions is capable of adding various additional sensors. For recording acceleration and GPS, only, a smartphone is a cheap solution. The biggest advantage of a smartphone based solution is that, once implemented, it can be used by many users and doesn't require further (expensive) hardware. Smartphone apps are capable of determining the choice of route, as well as dangerous cycling behavior.

# Chapter 4

# Real World Cycling Environment

In this chapter, I describe the developed architecture, the Real World Cycling Environment (RCE). First, I give a brief thematic classification. The following sections go into more detail, starting in the second section by specifying the requirements for a bicycle measurement framework. In the third section, the data acquisition unit, which is the core element of the framework, will be explained. Following on, the next section explains the function of each sensor individually. The wiring of the entire system is given in the Appendix. Extracts of the wiring are explained in the corresponding sections of data acquisition unit and sensors. Details on the recording software as well as conversion and analysis software are covered in the last sections.

## 4.1 RCE - In A Nutshell

The RCE is a framework of sensors for tracking ride characteristics of a bicycle in outdoor experiments. Central element of this framework is a Raspberry Pi, a single-board computer powered by a mobile charger. Several sensors are connected to the Raspberry Pi to measure the ride characteristics. These are a potentiometer for the steering angle, a speed sensor based on magnets and a Reed switch, a GPS antenna, two brake sensors utilizing magnets and Hall effect sensors, and a Raspberry Pi Camera positioned at the handlebar and directed to the front. All hardware is mounted on an IO Hawk E2, a 26 inch e-bike, that can be driven outdoors to measure the respective values. On the software side during a measurement, nothing much more than storing the raw values is done. After the measurement, the raw data gets converted, which includes error correction, noise reduction and adjusting the frequency of output files, each only if necessary and possible. Noise reduction is, for some sensors, also supported by hardware (if possible). Where it makes sense or is needed, the sensors can also be calibrated, e.g., centering the steering angle sensor. Figure 4.1 shows an image of the bicycle and all hardware.

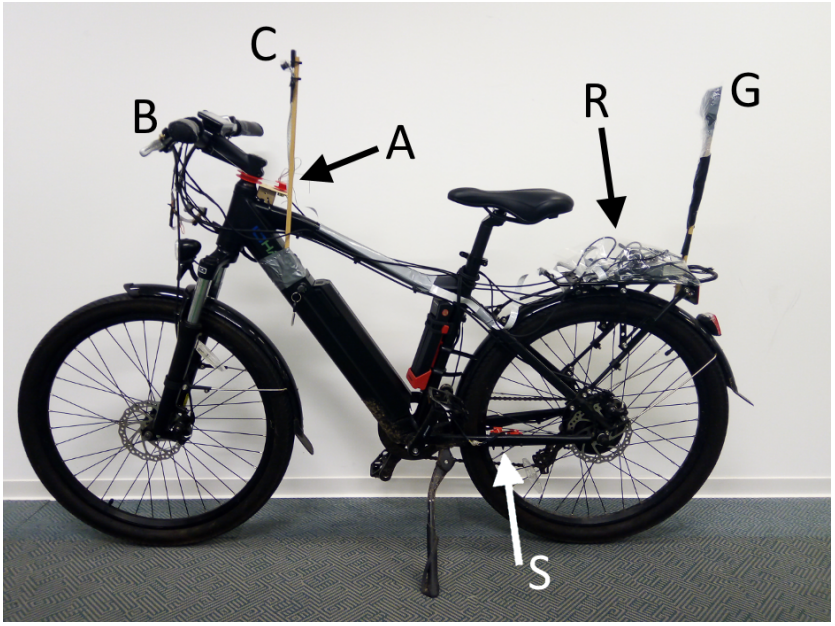



**Figure 4.1** – Photo of the e-bike with all sensors and hardware mounted. The Raspberry Pi, as well as mobile charger and the external GPS device are stored in a see-through plastic bag on the back rack (R). The locations of all sensors and devices are indicated with corresponding letters: Speed (S), steering angle (A), brake (B), camera (C), and GPS antenna (G). Camera and GPS antenna are mounted on poles to give better view and radio reception, respectively.

## 4.2 Requirements/Desired Functionality

A measurement framework should fulfill several requirements which I will explain in this section. First of all, the system should be able to record the main ride characteristics which are speed, steering angle, GPS position, and braking activity. Expandability is also a key feature of a framework, since it allows changes or other sensors to be added in the future. Hence, both hardware and software should be designed in a way that makes them expandable. Besides the sensor data, a video of the ride should be recorded, as it can be of great use for later analysis of the recorded data, e.g., to identify *why* the rider decelerated rapidly. The desired minimum frequency of sensor data is 10 Hz to allow a good comparability with traces of the VCE which are also recorded at 10 Hz. Yet, the frequency should be adjustable individually for each sensor within a reasonable range. For some experiments or studies it is probably not needed to have a high frequency of, e.g., the steering angle, if the focus is on braking reaction and not on the steering behavior of the rider. Additionally, it should be possible to conduct experiments with predefined configurations that specify which sensors are enabled and at which frequency they work. As the framework will be

used outdoors, where neither peripheral devices (screen, keyboard, mouse) nor a fixed power supply is available, it should be controllable with a mobile device, e.g., a smartphone, and allow the power supply via a mobile charger. Using a laptop in conducting experiments is not desired, since a laptop is comparably heavy and difficult to handle safely in outdoor environments or on a bicycle at all. Even if the laptop is fixed on the back rack, it is still much easier to control the system via a smartphone. With the controlling device the rider should be able to start and stop experiments or, during an experiment, be able to check whether everything is running correctly. Furthermore, it should be possible to easily conduct several experiments of predefined configurations in a queue, so the rider does not have to manually specify which configuration to use next. The whole system should be waterproof or at least be water protected by some degree, to withstand outdoor environments, when, e.g., it starts to rain unexpectedly, or the ground is still wet from a previous rain shower. This applies especially for sensors that are close to the ground.

The next sections go into detail and explain, how I realized each part of the framework to achieve the desired functionality.

## 4.3 Data Acquisition Unit

In this section I give information on the implemented data acquisition unit and the reasons for this choice, as well as how I set up the control via a smartphone. Then, I explain the circuits used for user input/output and for reading sensor signals, and how my measurement script works.

### 4.3.1 General Setup

The data acquisition unit that I am using in this thesis is the Raspberry Pi, whose characteristics have been described in Section 2.1. The main reason for this choice is that it already has many needed features onboard. It is small and lightweight, has several options for connecting sensors or additional off-the-shelf hardware, it can be powered with a standard mobile charger, and it can be controlled wirelessly via a smartphone. Additionally to that, it is reasonably priced (currently around 35 Euro) and allows straightforward programming of the GPIO pins using Python. I connected a Raspberry Pi Camera via a 2 m ribbon cable to the Raspberry Pi. To achieve the video recording functionality, I used the Python library PiCamera. All software is written in Python code.[12] Another advantage of the Raspberry Pi in comparison to a larger device, e.g., a laptop, is also that it easily fits into a plastic bag and thus protection against rain can be established very easily. It can be transported in a

[12] Only exception: The RCE Trace Viewer, covered in Section 6.5.

backpack or on the back rack of the bicycle. As most smartphones are at least a little water repellent, the system (Raspberry Pi + smartphone) is much less vulnerable to rain drops than a typical laptop.

### 4.3.2 Control Via Smartphone

To establish the wireless connection to a smartphone, I first enabled the SSH-server on the Raspberry Pi. Then, I set up the built-in WiFi module of the Raspberry Pi as a WAP, so the Raspberry Pi has its own WLAN. Now any smartphone can connect to this WLAN[13]. With an SSH-client app the smartphone can be used to control the Raspberry Pi. There are several free client apps available, one of which is an open source app called *ConnectBot*[14]. During development indoors, I sometimes experienced bad connection to the Raspberry Pi WLAN due to experimental drones (quadrocopters) that were also using wireless communication and flying nearby. Outdoors I did not notice any connection problems.

### 4.3.3 Wiring

To be able to use the RCE in case the connection to its WLAN fails or no smartphone is available, I added a total of three physical buttons to the Raspberry Pi. The buttons can be used for the following interactions:

- start or stop a measurement
- safely shutdown the Raspberry Pi
- restart the WAP daemon or change back to the default wireless mode to connect to other WLANs as a client, e.g., when it is required to have internet access

I also added three LEDs to return some feedback to the user. The represented information is:

- wireless mode: (WAP or default WLAN)
- measurement state: starting, running or stopping
- whether or not the Raspberry Pi is ready to start a new measurement

[13] Actually, *most* smartphones are able to connect. I encountered problems with an Honor 7S (Huawei) running Android 8.1.0. Sometimes the Raspberry Pi wasn't able to assign an IP-address to the Honor 7S, and thus, directly disconnected. Sometimes, however, connection was established and worked as expected. A Samsung Galaxy S3 mini was tested extensively and never caused any problems. I also tested several other phones and none of them caused problems.

[14] ConnectBot for Android can be downloaded on GooglePlay. The source code is available at: https://github.com/connectbot/connectbot/releases

The capabilities of three buttons and three LEDs are by far not exploited yet and can be changed or extended later on, if more functionality is needed. To enable reading analog data, I used an MCP3008, a 10-bit ADC with 8 channels. Communication between Raspberry Pi and ADC is established via SPI. The wiring of a Raspberry Pi, an MCP3008 and one exemplary pair of button and LED, is shown in Figure 4.2. Two more buttons and LEDs are wired analogously, but to different GPIO pins. I soldered the full wiring to a Hardware Attached on Top (HAT),[15] a strip board that fits on the Raspberry Pi. To allow interaction with the LEDs and buttons, I drilled holes and gaps in the case of the Raspberry Pi. The strip board and final case are shown in Figure 4.3.

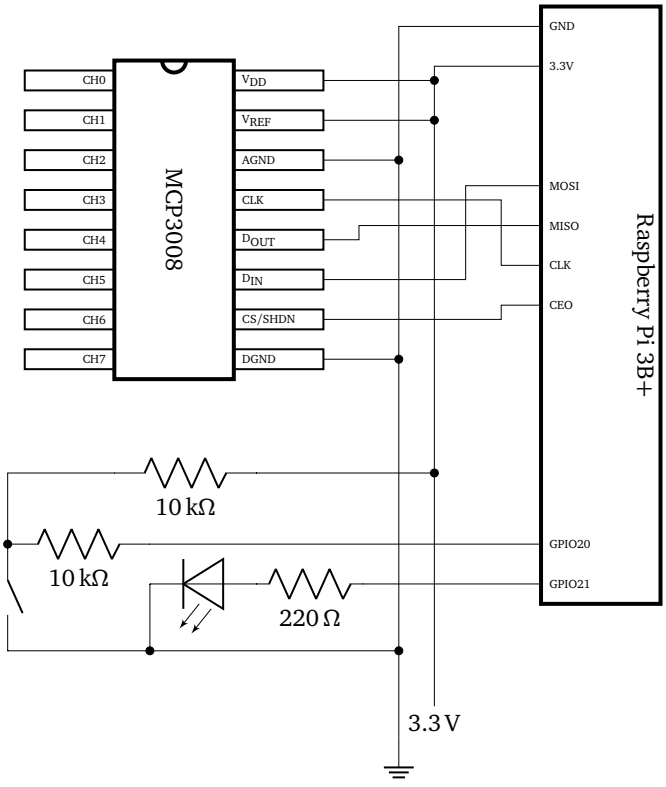
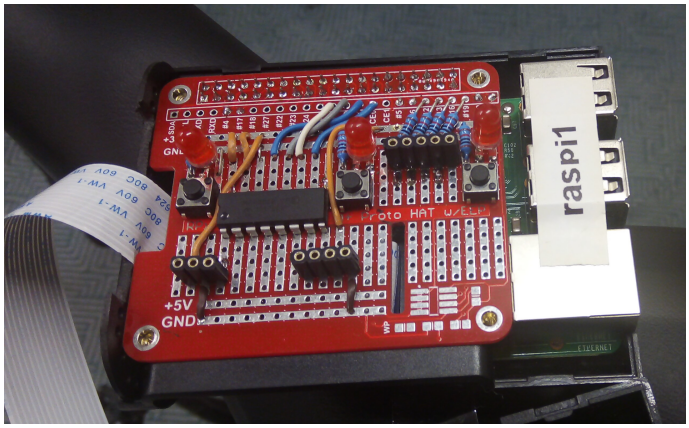
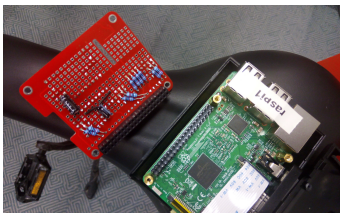



**Figure 4.2** – Wiring of Raspberry Pi, MCP3008, a button and an LED.

[15] e.g., https://www.adafruit.com/product/2310




**(a)** Top side



**(b)** Bottom side

**(c)** Inside the case

**Figure 4.3** – Raspberry Pi and shield with electronic components; for size comparison placed on the bicycle seat.
(a) Raspberry Pi and shield with MCP3008 and electronic components.
(b) Bottom side of the shield with capacitors and resistors.
(c) Closed case with gaps for LEDs, buttons, and sensor cables.

### 4.3.4 Measurement Software

The measurement script for each sensor basically consists of three phases. First, it initializes the hardware and reads settings from a configuration file, e.g., the desired recording frequency of a sensor. The configuration file also determines which sensors will be active. Then, the measurement script continuously reads/measures the values while storing it in a raw format. By using a raw format, different post processing or calibration methods can be used afterwards. For the trigger-based speed sensor, only the timestamps are stored. All other sensors, e.g., the steering angle sensor, read the current value, store it to a file (together with the current timestamp), and sleep for the rest of the time, all of that in an infinite loop. The algorithm for the periodical reading is stated in Algorithm 4.1. The sleep interval $t_{sleep}$ depends on the desired frequency $f$, with $t_{sleep} = \frac{1}{f}$. Though, it is not sufficient to simply sleep $t_{sleep}$ seconds, since reading and writing takes some time. Thus, the total time of one loop would exceed the should-be time, and the actual outcome frequency would be too

**Require:** Sensor is initialized, sleep interval $t_{sleep}$
**Ensure:** Measure and store sensor values and timestamps
1: Open file
2: $t_{next} \leftarrow$ current system time
3: **while** not interrupted **do**
4: $t \leftarrow$ current system time
5: $v \leftarrow$ sensor value
6: Write $t$, $v$ to file
7: $t_{next} \leftarrow t_{next} + t_{sleep}$
8: $t_{remaining} \leftarrow t_{next}$ - current system time
9: **if** $t_{remaining} > 0$ **then**
10: Sleep for $t_{remaining}$ seconds
11: **end if**
12: **end while**
13: Close file

**Algorithm 4.1** – `run` method of any sensor (except speed sensor). $t_{sleep}$ is calculated from the frequency. $t_{next}$ is the time when the next measure is to be taken. $t_{remaining}$ is the actual remaining time in this iteration and can be negative, if this or previous iterations were delayed. The *while*-loop can be interrupted by pressing one of the buttons wired to the GPIO pins, or by a keyboard interrupt from the calling shell.

low. To avoid this problem, my program computes the sleep-until time individually at the end of each iteration and just sleeps for the *remaining* time at the end of the loop, instead of using a static sleep-for time. Each new sleep-until time is computed by adding the sleep interval $t_{sleep}$ to the previous sleep-until time. With this method the program is able to catch up, if any delay occurs, as the *remaining* time will be $\leq 0$, and thus no sleep operation is executed.

I noticed that the timestamps consume a lot more storage than necessary. Instead of storing each full timestamp, I store the start timestamp (in the configuration file) once at the beginning of the recording, and each of the measurement timestamps is computed as the relative time since that start timestamp. This reduced the number of pre-decimals that have to be stored for each timestamp from 10 to, e.g., 3 for timestamps $t \in [100\,\text{s}, 1000\,\text{s})$ since the start time.

Finally, when stopping the measurement, the infinite loop gets interrupted. Afterwards, the raw data gets converted to a useful format, e.g., a value of 0.56 gets converted to a steering angle of 7° to the right. This also includes interpolation and noise reduction, if needed, which is explained in Section 4.6. The automatic conversion after a measurement can be turned off in the configuration file, as it may take a while, depending on frequency and number of active sensors. Since the video is recorded in a `.h264` format, it also has to be converted afterwards, e.g., to a `.mp4` format, to be able to play it with any usual video player. For video conversion I use

the free tool *MP4Box*[16] which is called automatically after a measurement. If the conversion was disabled in the configuration file, or if it is to be repeated later on, this can be done on the Raspberry Pi 3B+ (RP) or a different computer. Generally, I recommend to do the post processing afterwards on a different machine. Hence, continuing with the next experiment can be done without waiting time. Of course, both conversion scripts use the same methods and will produce equal output files.

I implemented several command line arguments which enable quick changes of some settings without editing any configuration file. The commands and explanations are shown in Table 4.1.

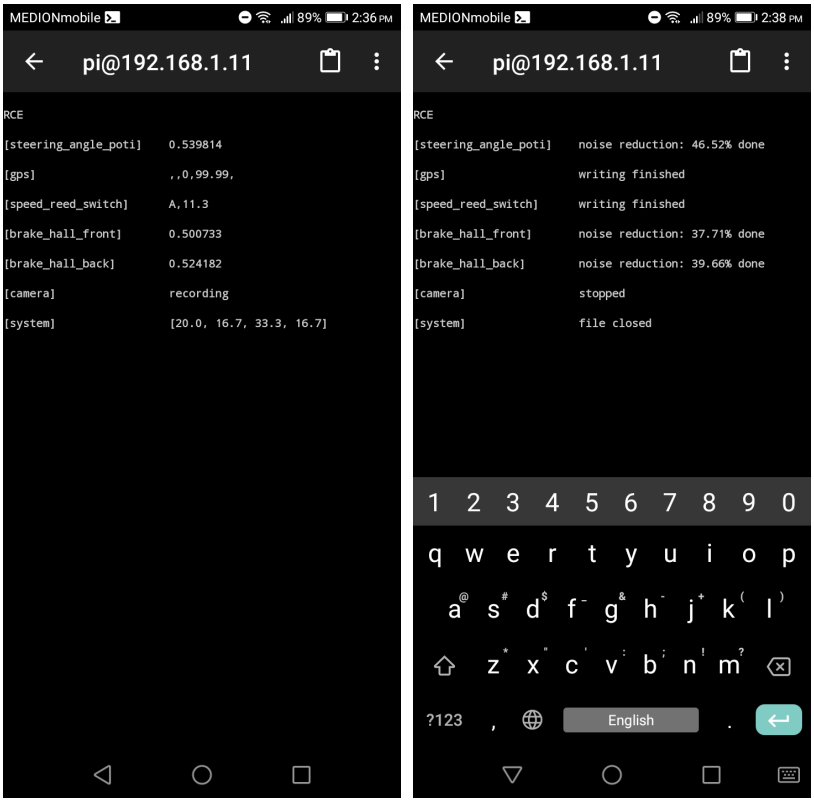
| Argument | Description |
|---|---|
| `-c, --config` | Define which config file to use |
| `-q, --queue` | Define which queue file to use |
| `-pl, --print_live` (0\|1) | Turns the live printer on(1) or off(0) |
| `-nac, --no_auto_convert` | Disable automatic noise reduction and interpolation |
| `-nvc, --no_video_convert` | Disable automatic video conversion |
| `-tf, --target_folder` | Define the folder in which to store the track |
| `-nf, --no_files` | Do not store any files, just run live measurement. This option implies `-pl 1` (true) and `-na` |

**Table 4.1** – Command line arguments of `run_measurement.py`. Except for the `--no_files` argument, all changes can also be done in the configuration file. The command line arguments are only for convenience during experiments.

If no controlling device is available, the default configuration can be used by pressing one of the hardware buttons of the RCE. A lock file prevents the system from trying to start several recordings simultaneously, e.g., if several devices connected via SSH and/or the hardware start button is used. Simultaneous recordings would lead to errors, as several processes try to read from shared hardware at the same time.

It is also possible to run several measurements in a queue. Stopping the current and starting the next measurement can be done with one of the physical buttons of the RCE. Hence, the controlling device, e.g., a smartphone, is only needed for starting the queue. To get instant feedback of all active sensors, the live printer can be activated. It displays the raw values of a sensor that have recently been measured, or an error message, if something went wrong, see Figure 4.4.

[16] https://gpac.wp.imt.fr/mp4box/




**(a)** During measurement **(b)** During conversion

**Figure 4.4** – Screenshots of the live printer on an Android smartphone that has been connected to the RCE via SSH using the app *ConnectBot*.
(a) The current values are displayed in raw format, except for the speed that displays the estimated speed depending on the time difference between the last two signals, as well as the layer which in my case always is `A`, since I ended up using just one layer. The GPS has no valid data, as the screenshot was taken indoors where no signal was available. The system sensor displays the current usage of all four CPU cores.
(b) During post measurement conversion, the current progress of each sensor is shown. The keyboard can be extended to offer additional keys, e.g., control, tab, and arrow keys (feature of the app used).

## 4.4 Sensors

This section, first, gives a rough overview of the sensors that I developed for the RCE. In the subsections, I explain in detail how each of the sensors is constructed and wired. Information on calibration and conversion software is given in the successive sections.

The speed sensor is based on the interaction of magnets and Reed switches that determine the rotary speed of the wheel and thereby the speed of the bicycle. The steering angle is measured with a construction that translates the rotation of the handlebar to a potentiometer. GPS data is obtained by repeatedly reading from a high-quality GPS device. Recording the brake usage is realized with magnets and Hall effect sensors that are capable of determining how far the brake handle is pulled by measuring the distance between Hall effect sensor and a magnet.

### 4.4.1 Speed

The basic idea of my speed sensor is similar to a conventional speedometer: A magnet induces a signal when passing across a sensor, and thus, the speed can be calculated from rotary speed and size of the wheel. The magnet can be detected with a Reed switch that closes when the magnet is nearby. Common speedometers are working with one signal per wheel revolution, which results in a low frequency of data, especially at low speeds. A bicycle with a wheel diameter $d = 26$ inch, e.g., would result in a signal frequency of $f = 1.3\,\text{Hz}$, when traveling at a speed of $v = 10\,\text{km/h}$ (velocity), which is much less than the desired $f_{min} = 10\,\text{Hz}$. For comparison, a velocity $v = 74.7\,\text{km/h}$ would be required to reach a frequency of 10 Hz, when using just one magnet. Let $n$ the number of signals (magnets) per revolution, $\triangle x$ the distance per signal, $\triangle t$ the time difference between two signals. The diameter of the bicycle is $d = 26\text{ inch} = 0.6604\,\text{m}$. A wheel covers a distance of one circumference per full revolution, and thus, an $n$-th circumference per signal:

$$\triangle x = \frac{\pi \cdot d}{n} \tag{4.1}$$

Velocity $v$ and frequency $f$ are defined as:

$$v = \frac{\triangle x}{\triangle t} \tag{4.2}$$

$$f = \frac{1}{\triangle t} \tag{4.3}$$

Hence, it holds:

$$v = \frac{\triangle x}{\triangle t} = \triangle x \cdot f \tag{4.4}$$

$$f = \frac{v}{\triangle x} \tag{4.5}$$

Further, it follows from Equation (4.4):

$$v = \frac{\triangle x}{\triangle t} = \triangle x \cdot f = \frac{\pi \cdot d}{n} \cdot f \tag{4.6}$$

Due to the low frequency when using just one magnet, I increased the number of magnets to nine, as this fits the nine-fold symmetry of a typical bicycle wheel. This increases the frequency by a factor of nine, i.e., 12.0 Hz at a speed of 10 km/h and a required speed of 8.3 km/h to reach a frequency of 10 Hz. An additional idea was, to use nine magnets, placed in three different layers with three correspondent sensors. In this design each sensor processes three signals per revolution, thus, a lower reaction time of each sensor is required. From all three sensors a total of nine signals per revolution are received, though. It turned out that the limitation of the system is not the reaction time of the Reed switches but the processing capabilities of the Raspberry Pi. However, it is very important to put sensors and magnets close to each other and use the right region on the spokes. As the spokes are made of a magnetic material, the signals get very fuzzy at the point where two spokes are crossing each other. Hence, I recommend to use a layer (radius) that is, e.g., 2 cm from the outer end of the spokes and thus far (>10 cm) from spoke crossings.

I used low cost Reed switches of type ORD221 and rod magnets that were stated with a strength of approximately 6.86 N. More details on the Reed switches[17] and magnets[18] are given in the data sheets. Using an oscilloscope, I measured the on and off cycle[19] of this combination of magnets and Reed switches at constant speed. In two test measurements with nine magnets the times were in a ratio of approximately 14 % on and 86 % off. Since all nine magnets cover 360°, each magnet covers an angle of 40°. Hence, the Reed switches are closed for an angle of approx. 5.6° (14 %), and open for 34.4° (86 %). With stronger magnets this could be shifted towards a more balanced ratio.

To firmly attach the Reed switches to the bicycle frame, I created a 3D model of a mount that has a gap in it to hold a Reed switch, see Figure 4.5. It can be fastened to the mainframe with cable ties, e.g., on the bar that is connecting the pedaling hub and back wheel hub. As the spokes are not perfectly parallel to that bar, the gap is tilted by 13°. Due to this tilt, the magnets cross the Reed switch in parallel, which is one of the orientations that is recommended by manufacturers of Reed switches[20]. However, a non tilted wooden prototype also worked well. The spokes of the bicycle are made of a magnetic material, so the magnets already stay in position. But to avoid turning due to rotational and shaking forces while riding, I secured them with

[17] https://standexelectronics.com/products/kofu-reed-switch-ord221/

[18] https://www.supermagnete.de/eng/rod-magnets-neodymium/rod-magnet-4mm-10mm_S-04-10-AN

[19] on = closed; off = open

[20] https://standexelectronics.com/wp-content/uploads/Reed-Switch-And-Magnet-Interaction.pdf

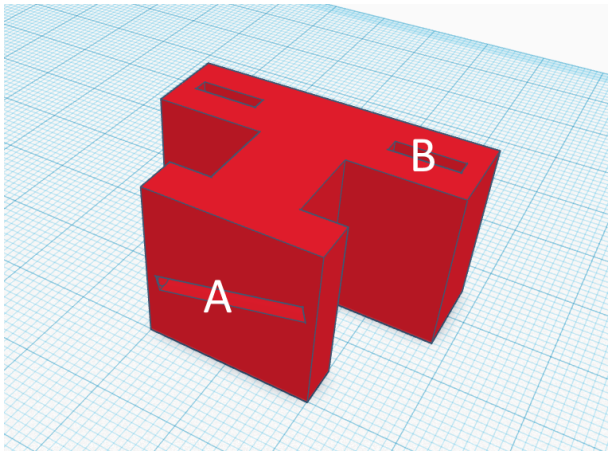
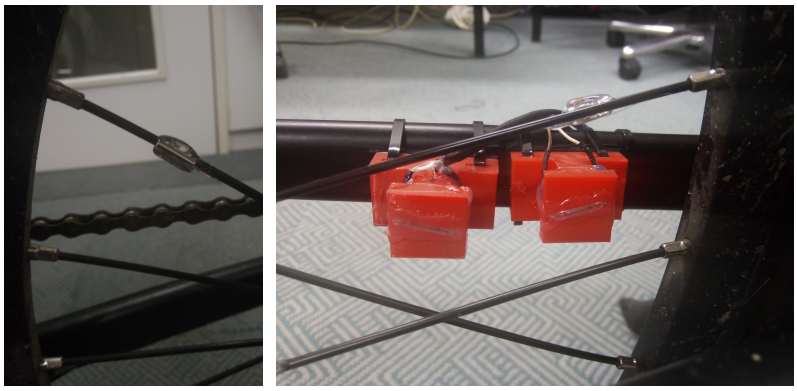



**Figure 4.5** – 3D model of the Reed switch mount. The gap (A) can hold the Reed switch and is tilted by 13°. The two holes (B) enable attaching the mount to the bicycle frame with cable ties.

hot glue. Figure 4.6 shows images of the mount, as well as a magnet on a spoke. Resulting from Equation (4.1) and Equation (4.6) and due to the number of nine magnets, the distance per signal and thereby the speed can be calculated with:

$$\triangle x = \frac{1}{9} \cdot \pi \cdot 0.6604\,\mathrm{m} \tag{4.7}$$

$$v = \frac{\triangle x}{\triangle t} = \frac{\frac{1}{9} \cdot \pi \cdot 0.6604\,\mathrm{m}}{\triangle t} \tag{4.8}$$



**(a)** Magnet on a spoke **(b)** Reed switch mount

**Figure 4.6** – (a) A magnet on a spoke, secured against moving with hot glue. (b) Two mounts with the tilted gap to hold a Reed switch in it. The electronic components are secured against water with hot glue.

The bounce time was set programmatically to 1 ms, since this is the minimum supported bounce time by the gpiozero library and outranges the specification of the Reed switches given in the data sheet[21]. At the same time, this is sufficient for cycling faster than 50 km/h which I will show in the following paragraph.

As stated above, the on-cycle corresponds to an angle of 5.6°. At the possible maximum speed, this angle is passed in twice the bounce time, i.e., 2 ms. With the rule of three this means that 40° (one on-off cycle) is passed in 14.3 ms:

$$v_{max} = \frac{\triangle x}{\triangle t} = \frac{\frac{1}{9} \cdot \pi \cdot 0.6604\,\text{m}}{14.3\,\text{ms}} = 58.09\,\text{km/h} \tag{4.9}$$

$$f_{max} = \frac{1}{\triangle t} = \frac{1}{14.3\,\text{ms}} = 70\,\text{Hz} \tag{4.10}$$

The theoretical maximally measurable speed is 58.09 km/h, which corresponds to a frequency of 70 Hz.

### 4.4.2 Steering Angle

The steering angle sensor is based on a rope connection between the handlebar and a potentiometer. A sketch of the setup is shown in Figure 4.7. The basic idea is to translate the motion of the handlebar to a potentiometer and periodically measure the value of the potentiometer. Its output voltage is a representation of the steering angle and can directly be translated, because angle and value of a potentiometer

[21] https://standexelectronics.com/products/kofu-reed-switch-ord221/

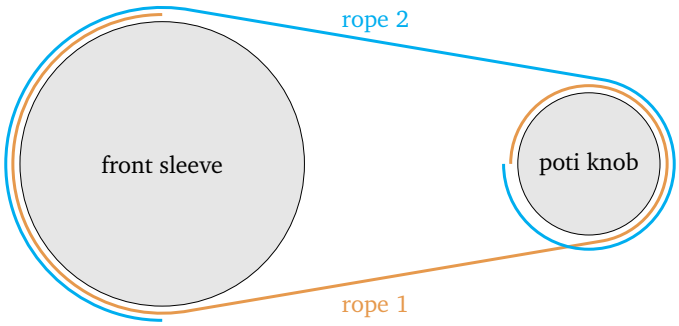



**Figure 4.7** – Sketch of the two ropes connecting front sleeve and poti knob (top view). When rotating the front sleeve clockwise, rope 1 gets wrapped around the front sleeve and pulls and unwraps from the poti knob (also rotating clockwise). It thereby pulls rope 2 towards the poti knob and unwraps it from the front sleeve. When rotating the front sleeve counter-clockwise, this happens in reverse. Both ropes directly lie on the sleeves, the different wrapping diameters are just for this sketch. They do not cross, but lie next to each other.

relate linearly[22]. In my design, the mount is placed on and fixed to the upper bar of the main frame behind the steering shaft, which does not make any difference in the working method compared to Moore [8], but was easier to assemble for this bike. I created 3D models for two sleeves, one around the steering shaft and one that fits on the potentiometer, which I call *front sleeve* and *poti knob*, respectively. By using sleeves of corresponding diameters, it is possible to adjust the range of steering angles to the range of the potentiometer. The low cost potentiometer that I used has a working angle of 300° (±20 %). To get an estimate for the needed steering angle range, I manually measured the angles between main frame and handlebar, while setting the handlebar to typical maximum angles from my personal cycling experience. This resulted in a range of approximately ±60°. I compared these more or less guessed values to the maximal measured values of Moore [8]. He found a range of ±65° [8]. As my design does not contain an end stop and probably some part of the range is consumed by the calibration,[23] I decided to set the range to ±75°. Thus, the 300° of the potentiometer have to be translated to ±75° (150°) of the handlebar. With simple calculations the relation between the diameters of the two sleeves can be determined. The way $w$ a rope or belt walks around a sleeve with a diameter $d$ over an angle $\alpha$ (all angles denoted in degree) can be calculated with:

$$w = \pi \cdot d \cdot \frac{\alpha}{360^\circ} \tag{4.11}$$

The indexes denote handlebar $h$ (front sleeve) and potentiometer $p$ (poti knob). Since the way that is walked on each sleeve needs to be equal, it follows:

$$\pi \cdot d_h \cdot \frac{\alpha_h}{360^\circ} = w_h = w_p = \pi \cdot d_p \cdot \frac{\alpha_p}{360^\circ} \tag{4.12}$$

$$d_h \cdot \alpha_h = d_p \cdot \alpha_p \tag{4.13}$$

$$d_p = \frac{d_h \cdot \alpha_h}{\alpha_p} \tag{4.14}$$

I chose $d_h = 4\,\text{cm}$ which results in $d_p = 2\,\text{cm}$. To avoid possible belt slipping, my design translates the steering motion via two separate ropes, both firmly connected to both front sleeve and poti knob. The ropes are wrapped around each sleeve far enough, to allow full usage of the corresponding angles in a linear way, i.e., they are not crossing each other and always leave a sleeve in tangential direction. Since the ropes are made from Dyneema® SK78 material, they are non-elastic (<1 %), they

[22] Linear relation of angle and value of a potentiometer holds for linear potentiometers. There are also other forms, e.g., logarithmic potentiometers. All potentiometers mentioned in this thesis are linear potentiometers.

[23] Probably the potentiometer is already turned by a few degrees, when steering straight forward.

do not soak water nor change elasticity or strength[24], and thus, there is also no play. An image of the built sensor is shown in Figure 4.8.

For wiring the potentiometer I connected the $V_{REF}$ and GND terminals to 3.3V and GND of the GPIO pins, respectively. The signal line gets connected to one of the channels of the MCP3008 ADC and thus can be read by a Python script. I added a capacitor with a value of 1 µF, as this is recommended in the data sheet of the MCP3008.[25] The circuit is shown in Figure 4.9.

[24] https://www.kanirope.de/shop/dyneema-seile-geflochten
[25] https://www.microchip.com/wwwproducts/en/en010530

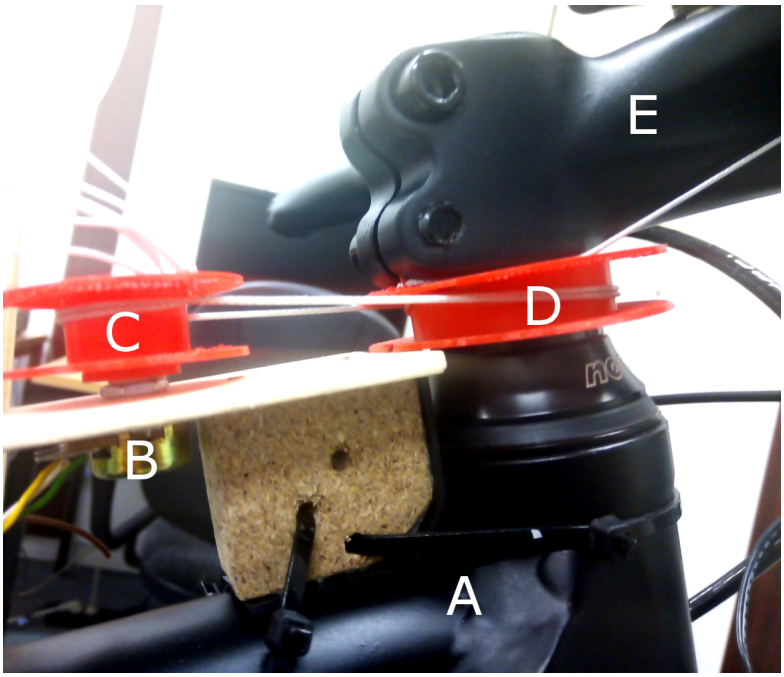



**Figure 4.8** – Steering angle sensor. The wooden mount is attached to the mainframe (A) with cable ties. A black rubber layer between mount and mainframe, secures it against moving around the mainframe. The potentiometer (B) is screwed to the mount and secured against turning. The poti knob (C) and front sleeve (D) are connected with ropes that are firmly attached to both poti knob and handlebar (E).

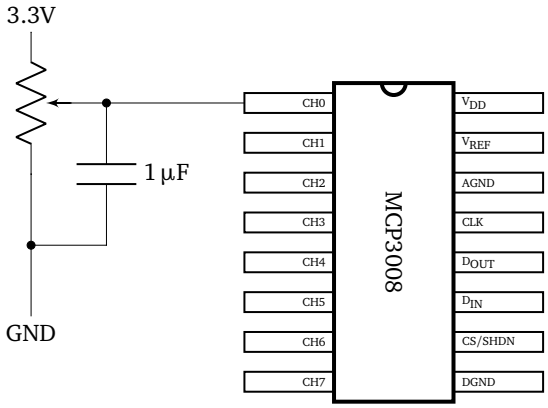



**Figure 4.9** – Wiring of potentiometer and MCP3008. The 1 µF capacitor is recommended in the MCP3008's data sheet.

### 4.4.3 GPS data

The GPS data is recorded with a high quality GPS receiver, the ublox EVK-7N. To achieve high accuracy, I enabled the receiver's WAAS settings. When connecting the GPS receiver via USB to the Raspberry Pi, it automatically starts transmitting the received GPS data, according to the National Marine Electronics Association (NMEA) protocol[26]. NMEA 0183 contains, among other definitions, standardized GPS messages that are called *sentences*. These are sent in groups of at least six sentences, but when many satellites are available, the count increases. As only some of the received NMEA sentences are needed, and directly storing all raw messages would consume much more storage, I added a short filtering and parsing method, to receive and store only the desired information. In this way, I also avoid storing redundant information within one group of sentences.

[26] https://www.gpsinformation.org/dale/nmea.htm

An example with eight NMEA sentences is given in the following. I marked the relevant information that is filtered out: **UTC time**, **status (A=active, V=void)**, **latitude**, **longitude**, **speed in km/h**.

```
$GPRMC,151741.20,A,5143.95189,N,00844.10765,E,0.040,,
    210120,,*1E
$GPVTG,,T,,M,0.040,N,0.074,K*49
$GPGGA,151741.20,5143.95189,N,00844.10765,E,1,07,1.19,
    114.4,M,46.5,M,,*52
$GPGSA,A,3,17,10,15,12,25,24,19,,,,,,2.03,1.19,1.64*00
$GPGSV,3,1,11,06,,,30,10,16,280,32,12,58,235,45,15,25,
    181,33*43
$GPGSV,3,2,11,17,33,052,37,19,45,077,23,20,03,252,,24,
    88,205,51*74
$GPGSV,3,3,11,25,18,242,28,32,,,29,36,,,30*45
$GPGLL,5143.95189,N,00844.10765,E,151741.20,A*03
```

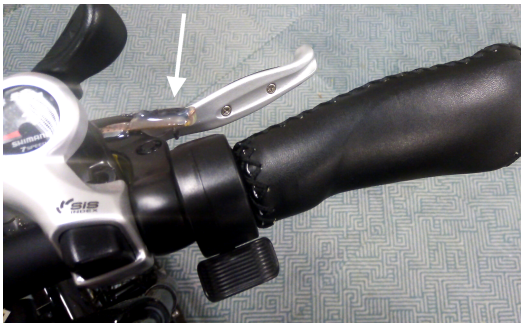
Each sentence begins with its data type and ends with a checksum, indicated by the symbols `$` and `*`, respectively. The data type defines for each sentence what information is contained in which order. Line 2 contains the speed information (in km/h), lines 1, 3, and 8 contain the position information, and UTC time. The status symbol is only given in lines 1 and 8 and indicates whether or not GPS data is available.

### 4.4.4 Brake

The brake sensor is based on a Hall effect sensor which outputs a floating value depending on the direction and strength of the current magnetic field. My construction consists of a tiny magnet that is glued to the brake handle. A Hall effect sensor is soldered and glued to a strip board that itself is glued to the housing of the brake handle, thus, to a non-moving part. When the rider applies the brake, the magnet approaches the Hall effect sensor and the magnetic field changes. Figure 4.10 shows a photo of the sensor from different perspectives.

The $V_{DD}$ and GND terminals of the Hall effect sensor are connected to 3.3V and GND, respectively. Figure 4.11 shows the wiring of MCP3008 and the two Hall effect sensors for both brakes. As the sensor outputs analog signals, the signal pin is connected to one of the ADC channels. Noise reduction with a bypass capacitor is recommended for all sensor devices in the ADC's data sheet.[27] The data sheet of the

[27] https://www.microchip.com/wwwproducts/en/en010530



**(a)** Top view

**(b)** Bottom view, released

**(c)** Bottom view, applied

**Figure 4.10** – (a) Top view of the brake handle and the right part of the handlebar. The brake sensor is marked (white). (b), (c) Bottom view of the brake handle with Hall sensor (blue mark) and a tiny magnet (red mark). When applied, the magnet approaches the Hall sensor.

Hall effect sensor however says that no external noise filtering is needed.[28] To be safe, I tested the recommended bypass capacitor. But as expected, it had no effect. By continuously reading the value of the Hall effect sensor, braking actions of the rider can be detected.

[28] http://www.secsemi.com/products/Linear-Hall-ICs/SS49E.html

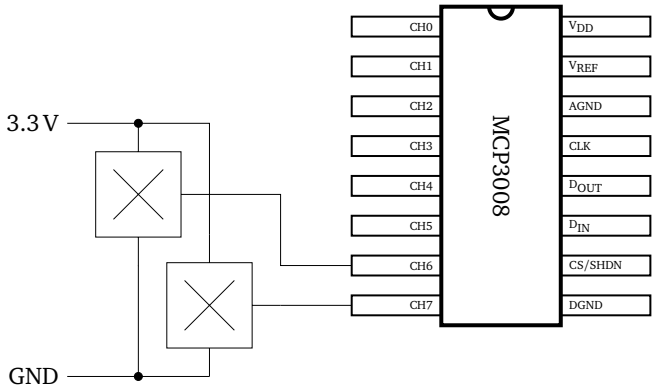



**Figure 4.11** – Wiring of MCP3008 and two Hall effect sensors. No additional electronic components are needed.

## 4.5 Calibration

Some of the sensors need calibration to work correctly, some can be calibrated to achieve higher accuracy.

The steering angle sensor is based on the rope connection of front sleeve and poti knob. Since the ropes were attached with the potentiometer being turned to a central, but not necessarily exact central, position, a value of 0.5 (central) of the potentiometer does not necessarily correspond to an exact steering angle of 0°. To calibrate the sensor, the front wheel must be held in a central, straight forward position, while the value of the potentiometer is measured for several seconds. After measurement, the same noise reduction algorithm as in the measurement script is used. The mean value over the entire calibration measurement is used as the zero degree value and stored in the configuration file.

To calibrate the brake sensors, I use a similar method, but instead of computing the mean value, the minimum and maximum values are needed. During the measured sequence, the brakes should be applied and released fully for several seconds each. Again, after noise reduction, the minimum and maximum is computed and stored in the configuration file.

The calibration of both, steering angle sensor and brake sensor, can be done at any time.[29] By reconverting already taken measurements, these can be reprocessed using the new calibration, even post-recording. Once a calibration has been executed, there is no need to repeat it for each measurement session, except, e.g., the brakes got replaced or adjusted harder or softer.

[29] except during a normal recording

## 4.6 Conversion Software

Conversion of the raw data consists of several steps that are executed consecutively. First, the raw files are read in and converted to a human readable format according to the sensor, e.g., by converting the ADC's output value which is in a range of [0,1] to a steering angle representation in degree. Next, a software noise reduction is run to remove peaks and possible noise that goes back to hardware inaccuracy. To match the output frequency, the values are then interpolated to get a value at the exact moments in time that are given by the frequency. Finally, the values get stored to a `.rce` file without timestamps, as the frequency and thereby the time difference between values determines which value was measured at which time. The start time of the measurement is stored in the configuration file that was copied to the track folder in the beginning of the measurement. If the conversion was skipped after the measurement or is to be repeated, e.g., due to subsequent calibration or any other reason, this can be done on a different computer, as well. Especially for longer recordings, the Raspberry Pi might take a very long time for conversion. Additionally to the post-measurement conversion, it is possible to compute different steps of conversion and various noise reduction versions (e.g., different interval lengths for moving average) to get less noisy sensor data, or to identify sources of errors in the measurement. In this section I will explain the mentioned steps in detail.

### 4.6.1 Sensor Related Format

Each sensor has a method that converts a line of its raw files to a sensor related value representation. In case of the steering angle sensor, the zero degree value was determined by the calibration tool and is stored in the configuration file. It is subtracted from each measured value. All measurement values, as well as the zero degree value $z$ are within [0,1]. Hence, subtracting $z$ from each measurement value will result in values $x \in [0-z, 1-z]$. The range of the measured values of the potentiometer corresponds to the 150° of the handlebar. Hence, the actual turning angle of the potentiometer $\beta$, relative to the straight forward direction, can be computed with:

$$\beta = x \cdot 150° \tag{4.15}$$

For the brake sensors, the calibration gives two values that represent "no break" $b_{min}$ and "full brake" $b_{max}$. So the measurement values are $b \in [b_{min}, b_{max}]$ and the range $r$ of the values is:

$$r = b_{max} - b_{min} \tag{4.16}$$

The percentage $p$ of brake application for a measured value $b$ can be determined with:

$$p = \frac{b - b_{min}}{r} \tag{4.17}$$

The speed sensor just uses the timestamps $t$ and $t_{prev}$, of this and the previous entry in the raw files, and the circumference of the wheel combined with the number of magnets per full revolution to compute the velocity $v$. The time difference is $\triangle t = t - t_{prev}$:

$$v = \frac{\triangle x}{\triangle t} = \frac{\frac{1}{9} \cdot \pi \cdot 0.6604\,\mathrm{m}}{t - t_{prev}} \tag{4.18}$$

When the bicycle stops, a timeout is needed to determine whether it really stopped or is just moving very slow. I chose a timeout of $t_{timeout} = 0.8\,\mathrm{s}$, since this correlates to a velocity of 1.0 km/h with nine magnets on a 26 inch wheel. Two 0 km/h entries are inserted at $t_{prev} + t_{timeout}$ and $t - t_{timeout}$, if two timestamps are $\geq 2 \cdot t_{timeout}$ apart.

For the GPS sensor no real conversion is needed, since the values already consist of position and accuracy information. Additionally, the GPS determined speed is stored and can serve as a reference value. I used the Python library gpxpy[30] to create GPS tracks as `.gpx` files, which can be viewed on map material.

[30] https://github.com/tkrajina/gpxpy

### 4.6.2 Noise Reduction

Sampling values multiple times per second, e.g., at a frequency of 4500 Hz, i.e., approximately $f_{max}$ for reading out the ADC with a Raspberry Pi,[31] outputs slightly different values over a very short period of time, without the actual value changing. The peaks might go back to tiny supply voltage peaks, or are due to the working load and clock of the Raspberry Pi, the specific sensors, or the ADC itself. Converted to, e.g., steering angle values, the peaks made a difference of up to ±2°. Hence, some filtering or noise reduction method is needed. An exemplary sequence is shown in Figure 4.12.

First, I implemented a standard moving average algorithm to reduce the errors. The result is already very good compared to the raw data, as shown in Figure 4.13. However, I detected a specific pattern in the raw data which led me to writing a specific noise reduction algorithm which I will explain in the next paragraph.

Over a period of 16 ms each, the steering angle sensor had some peaks for at most 5 ms (less than a third of the period). The signal was constant at one value for the rest of the time. Thus, an algorithm using a span of 16 ms makes sense.

[31] This frequency was only reached, when reading one sensor, only. It was not maintained constantly, but most of the time.

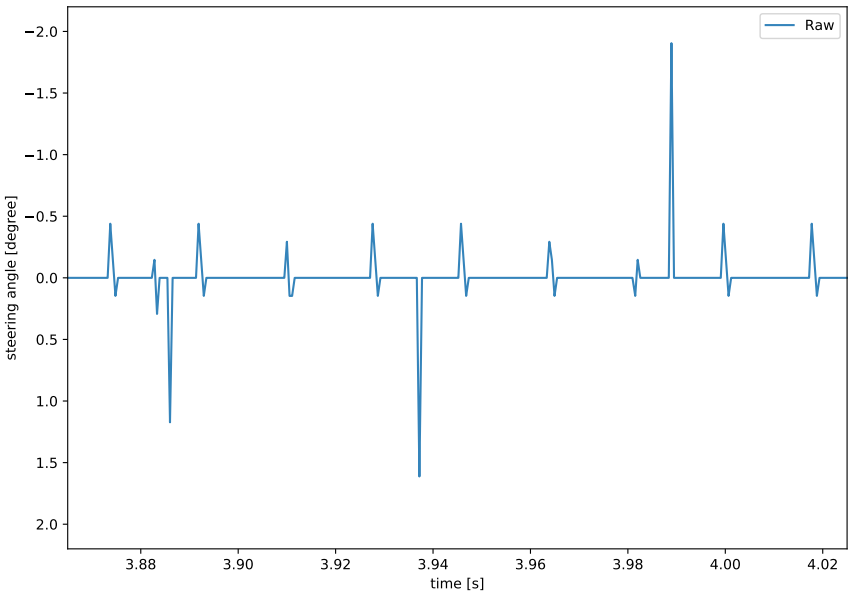
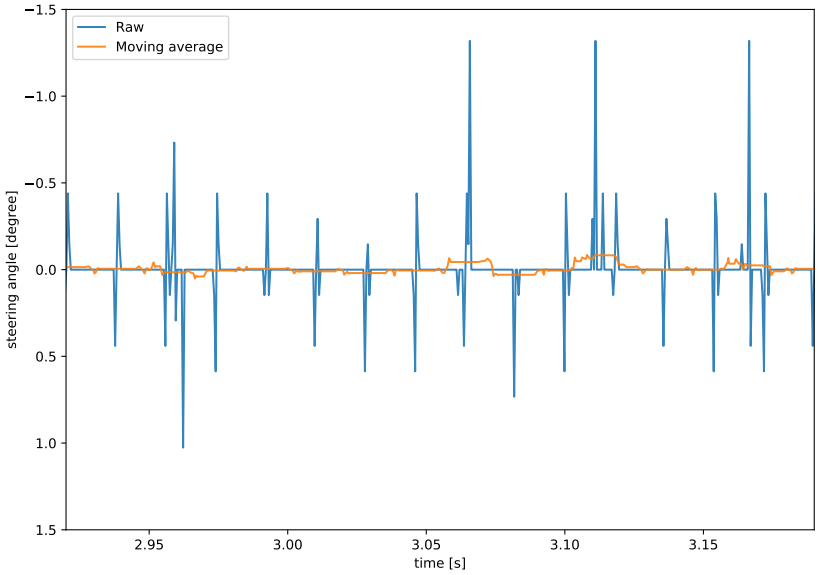
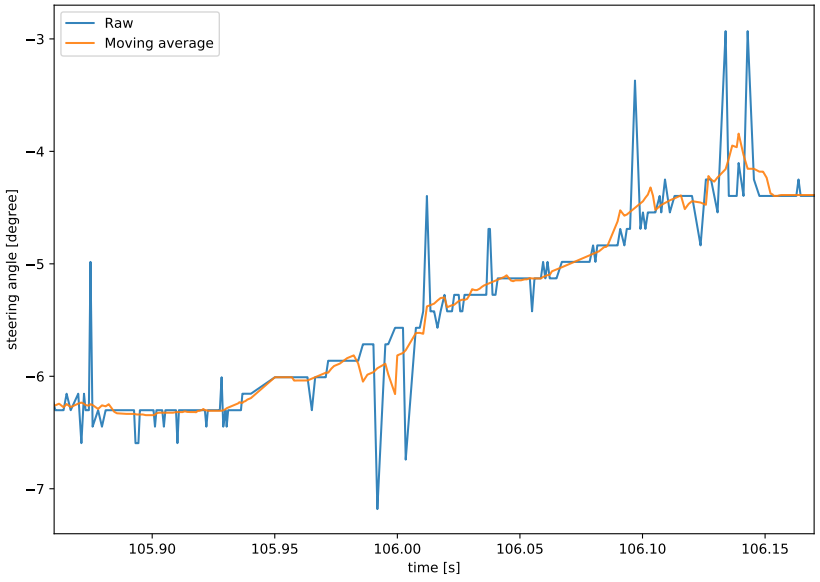



**Figure 4.12** – Raw steering angle values. The periodically minor peaks are mixed with some major peaks that occur irregularly. During each sequence of 16 ms the non-peak values amount to at least $\frac{2}{3}$ of the time. The major peaks reach up to ±2° of the steering angle.




**(a)** Constant value




**(b)** Changing value

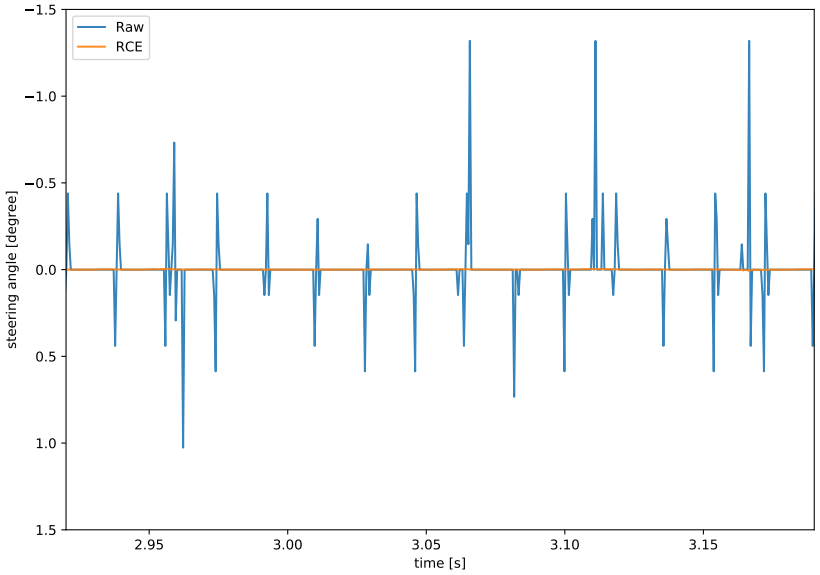
**Figure 4.13** – Comparison of raw steering angle to standard moving average values. It is clearly visible that the moving average is much less noisy. Though, especially at sequences with several major peaks to the same direction, the moving average remains a little noisier.
(a) During standstill of the handlebar in central position the peaks account for overestimation at, e.g., 3.06 s and 3.12 s.
(b) During changes the averaged values do not increase regularly, due to the double peaks at, e.g., 106.0 s and 106.14 s.

The number of high and low peaks were mostly even, while the amplitude differed, and sometimes high or low peaks dominated. In consequence, by simply taking the mean of all values, as done by the moving average algorithm, this would sometimes lead to a higher or lower output value than the correct one, see Figure 4.13. This observation led me to the idea of weighting the values. The algorithm that I came up with is given in Algorithm 4.2. To give fewest weight to the peaks that are far from average, I sort all values by their distance to the average value. The furthest value gets just one weight, and each other value increasingly one weight more than the previous one. The average value can be determined as the actual mean value, but again, this would give some more importance to the highly deviating peaks. I decided to use the median value,[32] since it is one of the non-peak values, and by this method the highly deviating peaks get ignored almost completely. However, I checked the results with both, mean and median, and did not detect any notable differences. Figure 4.14 shows a comparison of raw data to output of the described noise reduction algorithm during the same sequences as in Figure 4.13.

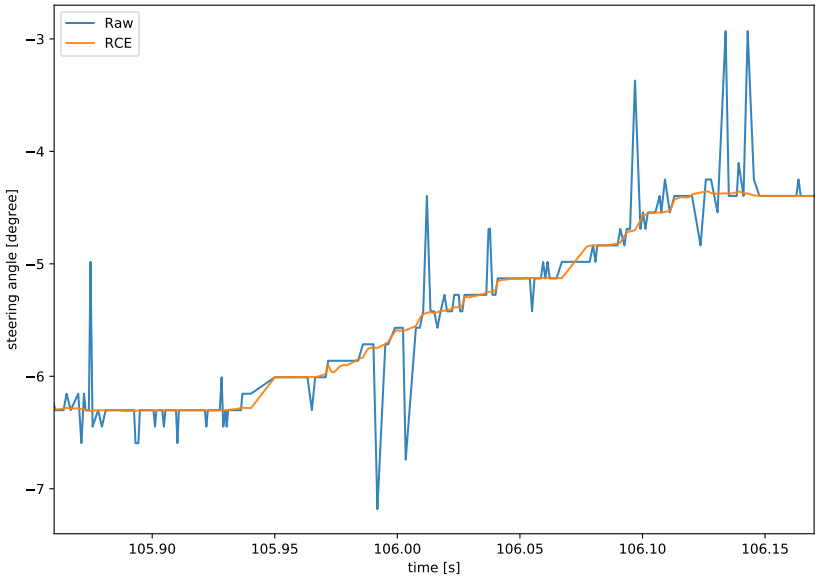
[32] Also known as the *central value* in a sorted sequence of values.

**Require:** Lists of times $t \in T$ and values $v \in V$ of an ADC-based sensor, span $s$ of periodic noise
**Ensure:** Lists of times $t \in T\prime$ and values $v \in V\prime$ with reduced noise
1: $T' \leftarrow \emptyset$
2: $V' \leftarrow \emptyset$
3: **for** $t \in T$ **do**
4: $\overline{T} \leftarrow \{\overline{t} \in T | t - \frac{s}{2} \leq \overline{t} \leq t + \frac{s}{2}\}$
5: $\overline{V} \leftarrow \{\overline{v} \in V |$ corresponding to $\overline{T}\}$
6: $m \leftarrow$ median of $\overline{V}$
7: Sort $\overline{v} \in \overline{V}$ by distance to $m$ (descending)
8: $w \leftarrow 0$, the sum of weights
9: $x \leftarrow 0$, the sum of weighted values
10: **for** $i = 1$ to $|\overline{V}|$ **do**
11: $w \leftarrow w + i$
12: $x \leftarrow x + i \cdot \overline{V}[i]$
13: **end for**
14: $v' \leftarrow \frac{x}{w}$
15: Append $t$ to $T'$
16: Append $v'$ to $V'$
17: **end for**
18: Return $T', V'$

**Algorithm 4.2** – Noise reduction algorithm for ADC-based sensor data. For each time $t \in T$ the values around $t$ get first sorted by their difference to their median. Then, all values get weighted increasingly, starting with the furthest. The output values are less noisy, especially in terms of major peaks. The output times $T'$ are equal to the input times $T$ and are just meant to keep times and values paired. Technically lines 1 and 15 could be removed.




**(a)** Constant value




**(b)** Changing value

**Figure 4.14** – Comparison of raw steering angle to values after the described noise reduction algorithm.
(a) During standstill of the handlebar in central position the peaks are filtered out almost completely.
(b) During changes of the steering angle the noise reduced values increase regularly. The double peaks at, e.g., 106.0 s and 106.14 s, have a negligible influence on the output value.
Note that the standard moving average algorithm and the newly described noise reduction algorithm use the identical span, and thus, are equally reactive to changes.

Inspecting the values measured by the brake sensors, I detected patterns comparable to those of the steering angle sensor, but with a periodic time of approximately 20 ms. Thus, for the brake sensors' noise reduction I use the same algorithm over a period of 20 ms, each.

To achieve having at least three values per sequence of 16 ms, a frequency of 187.5 Hz is needed, for 20 ms it is 150 Hz, respectively. A total of three values results in the majority (two) of these values being non-peak values. Hence, I suggest to use at least these frequencies. For low frequencies, the algorithm doesn't change any values, since the time difference between entries is larger than the span.

The trigger-based speed sensor has a slightly irregular output due to some magnets being turned a little around the spoke, and hence, they are slightly ahead their should-be position. I tried evading this source of errors by using a second sensor that is giving one signal per revolution. This allows measuring the specific angle between each two adjacent magnets of the first sensor. Unfortunately, the sensor missed some signals which is described in more detail in Section 5.1. These missing signals bring up the problem that it needs to be determined which signal(s) was/were missing. I ended up, not using a second layer, as errors increased when using two layers. For the resulting speed data from one layer with nine magnets, I use a normal moving average algorithm to achieve more regular data. To use averaging over a specific period of time and get an equal part of all measured values in this period, I first use linear interpolation[33] to determine a speed value for each millisecond. Afterwards, I compute the average over a period of 300 ms for each of these values (mean value). Using a shorter time left over some small peaks. By looking at the video during some peaks and by comparison to the brake sensors, I found that these peaks do not go back to an actual change in velocity. When increasing the time period to, e.g., 500 ms, this led to undesired behavior in braking situations, as the curve gets smoothed out too much.

[33] Briefly explained hereafter, in Section 4.6.3.

### 4.6.3 Output Files

The output files only contain values of each sensor, but no timestamps or any other information. Start time and frequency are determined by the configuration file of the track, and hence, a timestamp information would be redundant. Since the sensors record at a *desired* frequency that is not necessarily adhered to, or the *real* frequency might fluctuate slightly, there might not be a value for each needed moment in time. In case of a missing value at a specific time, I'm using linear interpolation on the two adjacent data points, to get a representative value. Assuming that $t_a < t_b$, the interpolation of two values $v_a, v_b$ with timestamps $t_a, t_b$ for a desired value $v$ at time $t$, can be done by:

$$v = v_a + (v_b - v_a) \cdot \frac{t - t_a}{t_b - t_a} \tag{4.19}$$

# Chapter 5

# Validation

In this chapter, I will describe my strategies and methods to validate the RCE against reality and ensure reliable data. This will result in accuracy values depending on the frequency that can be used to compare the RCE to related work. I will show up the limitations of my system and point out advantages or drawbacks in comparison to other approaches.

## 5.1 Speed

The RCE's speed sensor is based on a Reed switch and nine magnets that are mounted to the spokes matching the nine-fold symmetry of a typical bicycle wheel.

### Advantages And Drawbacks

Wehrbein [21], who used a similar method to measure the speed, took the time when a magnet was *entering*, thus when the Reed switch was closing. He stated the problem of some magnets probably being a little stronger, and thus, giving an earlier signal [21]. To avoid this source of inaccuracy, I don't only measure the *entering* time of a magnet, but also the *leaving* time. For speed calculation I use the time in between the two, which is the time of highest proximity of magnet and sensor, assuming a constant speed during a magnet passing across the sensor. Compared to speed sensors that use any sort of DC motor or bicycle dynamo, a contact free speed sensor avoids slowing down the bicycle due to friction At the same time, it avoids unreliable data in wet conditions, in which the roller of the DC motor might slip.

The theoretical maximally measurable velocity of my sensor is 58.09 km/h. This number is limited by the bounce time of the Reed switch, but not with respect to the reaction time of the Raspberry Pi. I yielded good results when using the speed sensor only, and especially at low speeds that do not require the Raspberry Pi to react fast,

e.g., at 10 km/h. Though, I was faced with a problem when cycling faster and/or using all sensors at once. The sensor, or rather the Raspberry Pi, started missing out signals of the Reed switch, since the on-time was too short for the Raspberry Pi to react to it and recognize it as closing and opening of a switch. This problem also occurred, when using three Reed switches in three different layers. Possible reasons for missing signals are widely spread. The Reed switch might be malfunctioning or broken. If the Reed switch is working properly, the Raspberry Pi could be too slow or too busy to recognize the interrupts, or the magnets could be too weak, resulting in a too short on-time of the switch. From the four Reed switches that I originally built to function in several layers, only one was working properly. Three Reed switches cracked during hot gluing them to the 3D printed mount, since the mount starts melting and contracting. Due to the contraction, it puts too much pressure on the tiny glass tube of the Reed switch and causes the breaking. Hence, I do not recommend to use a 3D printed mount in combination with hot glue. This problem did not occur, when gluing Reed switches to a wooden mount (which I used as a first prototype). To eliminate the problem of too weak magnets and a too short on-time of the switches, I measured the on-off cycle of the Reed switches with an oscilloscope. I found that the switches work very fast and do not seem to have a significant bounce time (it was $< 0.1$ ms), even though it is stated with 0.5 ms in the data sheet[34]. Thus, the problem has to be the reaction time of the Raspberry Pi, which also goes in hand with getting better results when using only the speed sensor at low speed.

To a degree the missing signals can be inserted automatically after a recording, by simply looking at a value $v$ and the predecessor $p$. Figure 5.1 shows a comparison of raw and repaired[35] speed values in an exemplary section. For each tuple, I computed the factor $f = \frac{v}{p}$ and if $f > 1.5$, it is likely to have missed a signal. As this reduction in speed could also originate from an abrupt brake application, looking at the following values can help to assess, whether the speed actually reduced or it was just one signal missing. Of course, this method comes with limitations, if the number of errors increases and it becomes more likely to have several missing signals in a row or repeatedly, e.g., for signals `s` and missing signals `m`, a sequence of `sssmsmsmsss` would result in three speed values being halved. An additional problem in missing signals is that they might occur at any point during regular acceleration, constant speed, or deceleration, and thus are harder to identify. If "repairing" the data too much, actual speed reductions might get identified as an error, and thus, be repaired consecutively by comparing to an already repaired predecessor.

[34] https://standexelectronics.com/products/kofu-reed-switch-ord221/
[35] after repair algorithm and moving average noise reduction

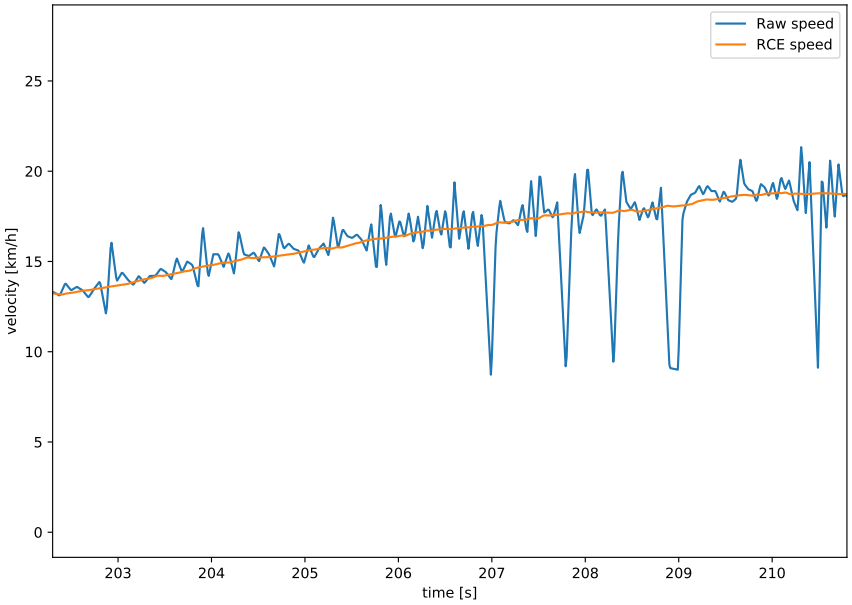



**Figure 5.1** – Comparison of the raw recorded speed values (blue) and the output after repair and noise reduction algorithms. The double error around second 209 s was repaired consecutively. The peaks on the left half are caused by single magnets being turned forward or backwards around the spoke, and thus, giving a slower/higher speed value for themselves and affect the next value the opposite way (higher/slower).

## Validation/Accuracy

| Tire pressure [bar] | Revolutions # | Distances [m] | | | | Circumference [m] |
|---|---|---|---|---|---|---|
| 1.1 | 24 | 49.14 | 49.10 | 49.05 | 49.06 | 2.045 |
| 2.0 | 23 | 48.29 | 48.28 | 48.30 | 48.30 | 2.100 |

**Table 5.1** – Measured distances during the wheel circumference test. The bicycle was pushed forward along a straight line while a person of 91 kg sitting on it. Both, low (1.1 bar) and high (2.0 bar) tire pressure were tested. By dividing the average wheel distance by the number of revolutions, the circumference was computed.

The first step in validation of the speed sensor was to measure the circumference of one wheel revolution as exactly as possible, since the speed computation highly depends on that value. I pushed the bicycle indoors and on flat ground along a straight line, while a person of 91 kg was sitting on it. To count the number of full wheel revolutions, I marked the bottom most spot on the wheel and stopped after a specific number of revolutions with the mark at the bottom again. The distance

along the straight line was measured with a measurement tape. To get an even more representative value and avoid errors, I repeated the experiment eight times, four times with each, low and high tire pressure. At a low tire pressure of 1.1 bar (almost flat tire), the circumference was 2.045 m, at a high tire pressure of 2.0 bar (almost absolutely filled) it was 2.1 m. This deviation is equivalent to a difference of 2.6 %, or a difference in velocity of, e.g., 14.72 km/h (low pressure) to 15.12 km/h (high pressure) at an equal rotary speed. The measured distances are listed in Table 5.1.

For a comparison between the measured speed of the developed sensor, GPS computed speed, and the displayed speed of the e-bike's onboard computer, I used a sequence of 290 s. The data of the speed sensor and the GPS speed were already available. To get the speed values of the onboard computer, I first investigated reading out the Universal Asynchronous Receiver Transmitter (UART) connection between engine and display of the e-bike. While reading the communication with a logic analyzer during acceleration and deceleration, only one value changed, corresponding to the current velocity. When stopping the wheel abruptly, I noticed that the recent value was sent repeatedly for two further seconds. One communication cycle of engine and display took approximately 500 ms. Hence, speed information can be read at a maximum of 2 Hz. Reading the UART communication directly with the Raspberry Pi, however, caused the communication to fail completely, thus it would require forwarding or an amplification of the signal with additional hardware, which might slow down the communication even further. I decided not to read out the communication programmatically, as the information gain (2 Hz) is quite low compared to the effort. Instead, I used the recorded video of the ride to get reference speed values during single rides. First, I split up the video file into single frames, to match the refresh rate of the display, 2 Hz. An exemplary frame is shown


**Figure 5.2** – View of the Raspberry Pi Camera during a ride. In the bottom, the display of the e-bike can be seen. It shows the current speed, 21.8 km/h.

in Figure 5.2. Then, I manually read the displayed velocity and noted it to a file, together with the time computed by the frame count.

Figure 5.3 shows all three speed data sets over the full sequence of the ride. It is clearly visible that the recorded values of all three speed sensors are similar, but differ in amplitude and noise. In comparison to the RCE's speed sensor, most of the time, the GPS underestimated the speed, while the onboard computer overestimated. A close up view of a short sequence is given in Figure 5.4 and shows that the RCE's speed sensor is between GPS and onboard computer in terms of noise, as well. The onboard computer's values were shifted by 0.8 s to the beginning of the ride to compensate for the communication and processing delay.

Additionally, I computed the mean deviation between the three data sets, as shown in Table 5.2. This required interpolation of each data set, to get values at the exact same times. Let $T$ the time interval of the recording and $n = |T|$. The Mean Absolute Error (MAE) of two data sets $A$ and $B$, or the *Mean Absolute Deviation*,[36] can be computed with:

$$MAE = \frac{1}{n} \sum_{t \in T} | A(t) - B(t) | \tag{5.1}$$

[36] Computing the absolute *error* would require to have a should-be value, but here I am just computing the deviation. The formula, however, is the same.

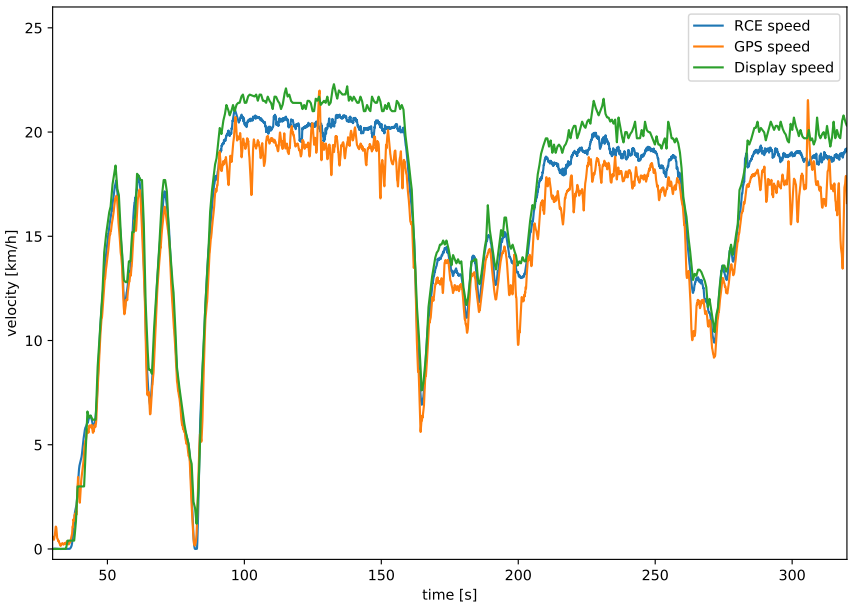



**Figure 5.3** – Full sequence of 290 s. It's clearly visible that the GPS (orange) underestimates the speed and has many peaks. The onboard computer (green) overestimates and has very few peaks due to the slow refresh rate (2 Hz).

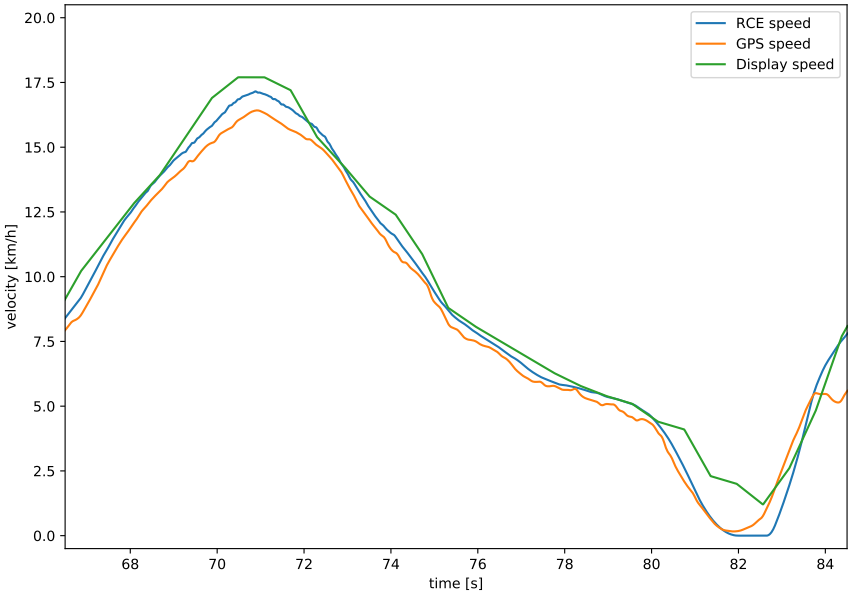



**Figure 5.4** – Comparison of RCE, GPS and onboard computer speed. In terms of noise, the RCE's sensor is in between both other data sets. Only the RCE detected the full brake (due to traffic; confirmed with video) between 82 s and 83 s.

I also computed the sums of the mean deviations to each of the other data sets, which is given in Table 5.3. The sums represent the average deviation to both other data sets. In this comparison the RCE gets the smallest value, as it deviates the fewest.

| Compared Data Sets | | | Mean Absolute Deviation |
|---|---|---|---|
| RCE | vs. | onboard | 0.89 |
| RCE | vs. | GPS | 1.08 |
| GPS | vs. | onboard | 1.85 |

**Table 5.2** – Mean Absolute Deviation of all three speed data sets compared to each other. Each value represents the average deviation in km/h.

| Sums of Mean Absolute Deviation | |
|---|---|
| RCE | 1.97 |
| GPS | 2.94 |
| onboard | 2.74 |

**Table 5.3** – Sums of the Mean Absolute Deviations. A low value represents high proximity to both other data sets.

## 5.2 Steering Angle

My steering angle sensor uses a potentiometer that is connected to the handlebar with two separate ropes, and translates the turning angles with sleeves of corresponding diameters. It works in a similar way to the designs of Moore [8] and Kooijman, Schwab, and Meijaard [7], but avoids the main drawbacks of those designs. Moore [8] placed the mount of the potentiometer in front of the steering shaft and fixed it to the non-moving part (mainframe). He translated the turning movement via a gear belt, but wasn't able to rule out slipping with absolute certainty [8]. To avoid possible belt slip by design, I use two separate ropes, both firmly connected to both, front sleeve and poti knob. Kooijman, Schwab, and Meijaard [7] limited the steering angle with end stops, which resulted in just using a small part of the full working angle of the potentiometer. The two sleeves of my sensor translate the angles, so the full range of the potentiometer is used. Additionally, the non-elastic ropes avoid any play. The calibration tool defines the straight forward angle.

### Validation/Accuracy

To measure the accuracy of the steering angle sensor, I needed to compare the measured angles to the actual angles. Since the bicycle parts are all curved and have many connectors in between the main bars, it is very difficult to measure any angles directly, e.g., the angle between the top bar of the mainframe and the handlebar. In addition to the validation, the actual angle of the front wheel in relation to the mainframe might be of interest. Hence, I built the Steering Angle Test Station (SATS). It consists of a wooden plank with some poles on it, to hold the bicycle in a straight and non-leaning position, while the front wheel is hovering a few millimeters above the ground. Figure 5.5 shows imagery of the SATS with the bicycle on it. To measure the angle of the front wheel, I built a frame that fits over the front wheel and gives a straight line on the ground plank, which is parallel to it and thus gives an equal angle. The frame is shown in Figure 5.6.

I plotted several angles on the ground plank, in detail 0°, 1°, 2°, 3°, 4°, 5°, 7°, 10°, 15°, 20°, 30°, 45°, 60°. The large angles are more widely spread, as they are less common during cycling. Some of the angles and the frame on the front wheel, can be seen in Figure 5.6. By adjusting the steering angle to each of the angles on the plank, and recording the measured angle using my sensor, I was able to compare actual to measured angles. Due to the backwards lean of the steering axis, the plank is not orthogonal to it. And hence, the angles have to be projected from one plane to the other, which I will explain in the next paragraphs.

Let a Cartesian coordinate system anchored in the ground plank, with the x, y, and z-axes, directing in straight forward driving direction, straight upright direction

**(a)** Steering Angle Test Station (SATS), full view

**(b)** Back wheel bars

**(c)** Center pole

**Figure 5.5** – (a) Full view of the Steering Angle Test Station (SATS) with the bicycle on it. (b) Two bars are securing the back wheel against sideways movements. (c) The center pole is carrying most of the weight of the bicycle. Only a small amount is carried by the back wheel. The front wheel is hovering a few millimeters above the ground plank.

**Figure 5.6** – The wooden frame that fits over the front wheel gives a sharp line indicating the angle of the front wheel on the ground plank. Both wheel and frame hover a few millimeters above the plank to avoid friction. In this image, the angle is adjusted to 20°.

(orthogonal to the ground plank), and towards the side. Since the setup is symmetric, the computation holds for both sides, left and right. Let $\alpha$ the backwards lean of the steering shaft, and $\beta$ the current steering angle (around the steering shaft), all angles in degree. Let further, two unit vectors $\vec{a}$ within the steering axis pointing towards the center of the front wheel, and $\vec{b}$ pointing straight forward with $\vec{a} \perp \vec{b}$. A sketch of the setup is given in Figure 5.7. These two vectors span a plane that is describing the plane of the front wheel $P_w$. When splitting up the vectors into rectangular components, this plane is:

$$P_w : \vec{x} = s \cdot \vec{a} + t \cdot \vec{b}, \qquad s, t \in \mathbb{R} \tag{5.2}$$

$$= s \cdot \begin{pmatrix} sin(\alpha) \\ -cos(\alpha) \\ 0 \end{pmatrix} + t \cdot \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \\ 0 \end{pmatrix} \tag{5.3}$$

Figure 5.8 demonstrates the turning of $\vec{b}$ by $\beta$ around the steering axis. The resulting vector $\vec{b^*}$ represents the current steering direction, while turning the handlebar. $P_w^*$ is the corresponding turned plane, spanned by $\vec{a}$ and $\vec{b^*}$. $\vec{a}$ remains constant, as it lies within the steering axis. Using the marked triangle in Figure 5.8 and $|\vec{b}| = 1$, it

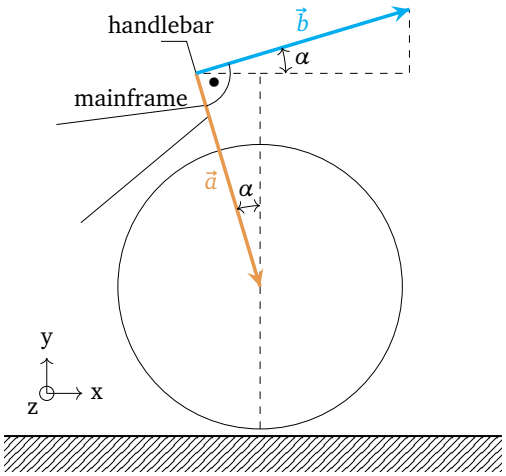



**Figure 5.7** – Sketch of the vectors $\vec{a}$ within the steering shaft, and $\vec{b}$ pointing forward, while the steering is at $\beta = 0°$. The circle depicts the front wheel, hovering a few millimeters above the ground plank. A construction on the plank holds the back of the bicycle in position.

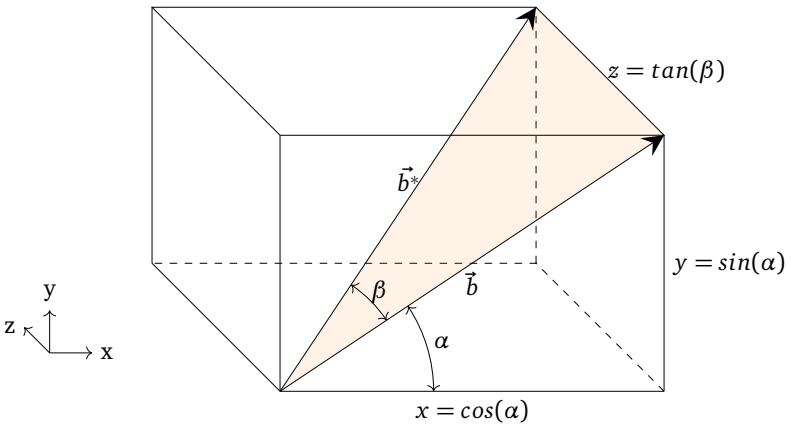



**Figure 5.8** – The vector $\vec{b}$ is turned by $\beta$ to the left side and results in $\vec{b}^*$. Since $\vec{b}$ is a unit vector, its x- and y-components are cos($\alpha$) and sin($\alpha$), and the z-component that was 0 in $\vec{b}$, is now tan($\beta$).

holds for $z$ the z-component of $\vec{b}^*$:

$$tan(\beta) = \frac{z}{|\vec{b}|} = z \tag{5.4}$$

$$\vec{b}^* = \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \\ tan(\beta) \end{pmatrix} \tag{5.5}$$

$$P_w^* : \vec{x} = s \cdot \begin{pmatrix} sin(\alpha) \\ -cos(\alpha) \\ 0 \end{pmatrix} + t \cdot \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \\ tan(\beta) \end{pmatrix}, \qquad s,t \in \mathbb{R} \tag{5.6}$$

To compute the angle between two planes, it is sufficient to compute the angle between the normal vectors of both planes. A normal vector $\vec{n}$ of a plane, can be computed with the cross product of the two vectors spanning the plane. When swapping the vectors of the cross product, the normal vector will be inverted. In the following, I took care of selecting the *right* normal vectors, so there is no need to take the adjacent angles (remaining angle to 180°). When turned by $\beta = 0°$, the front wheel plane $P_0$, can also be written as:

$$P_0 : \vec{x} = u \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + v \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \qquad u,v \in \mathbb{R} \tag{5.7}$$

The corresponding normal vector $\vec{n_0}$ is:

$$\vec{n_0} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \tag{5.8}$$

For the plane of the (turned) front wheel, it holds:

$$\vec{n_w} = \vec{a} \times \vec{b}^* = \begin{pmatrix} sin(\alpha) \\ -cos(\alpha) \\ 0 \end{pmatrix} \times \begin{pmatrix} cos(\alpha) \\ sin(\alpha) \\ tan(\beta) \end{pmatrix} = \begin{pmatrix} -cos(\alpha) \cdot \tan(\beta) \\ -sin(\alpha) \cdot \tan(\beta) \\ sin^2(\alpha) + cos^2(\alpha) \end{pmatrix} \tag{5.9}$$

By setting the y-component of $\vec{n_w}$ to 0, the vector is projected into the xz-plane (here the ground plane):

$$\vec{n_w}' = \begin{pmatrix} -cos(\alpha) \cdot \tan(\beta) \\ 0 \\ sin^2(\alpha) + cos^2(\alpha) \end{pmatrix} \tag{5.10}$$

$\vec{n_0}$ is already within the xz-plane. The angle $\gamma_{uv}$ between two vectors $\vec{u}, \vec{v}$ can be computed with:

$$cos(\gamma_{uv}) = \frac{\vec{u} * \vec{v}}{|\vec{u}| \cdot |\vec{v}|} \tag{5.11}$$

Let $\gamma$ the angle between the straight forward direction (0°) and the projection line of the front wheel plane on the ground plane, thus, the measured angle on the ground, when turning the handlebar by $\beta$. In the following, I use the Pythagorean trigonometric identity, i.e.:

$$sin^2(\alpha) + cos^2(\alpha) = 1 \tag{5.12}$$

Then, the should-be value of $\beta$ can be computed as:

$$cos(\gamma) = \frac{\vec{n_w}' * \vec{n_0}}{|\vec{n_w}'| \cdot |\vec{n_0}|} \tag{5.13}$$

$$= \frac{-sin^2(\alpha) - cos^2(\alpha)}{\sqrt{(-cos(\alpha) \cdot \tan(\beta))^2 + (sin^2(\alpha) + cos^2(\alpha))^2}} \tag{5.14}$$

$$= \frac{-(sin^2(\alpha) + cos^2(\alpha))}{\sqrt{cos^2(\alpha) \cdot \tan^2(\beta) + (sin^2(\alpha) + cos^2(\alpha))^2}} \tag{5.15}$$

$$= \frac{-1}{\sqrt{cos^2(\alpha) \cdot \tan^2(\beta) + 1}} \tag{5.16}$$

$$\sqrt{cos^2(\alpha) \cdot \tan^2(\beta) + 1} = \frac{-1}{cos(\gamma)} \tag{5.17}$$

$$cos^2(\alpha) \cdot \tan^2(\beta) + 1 = \frac{1}{cos^2(\gamma)} \tag{5.18}$$

$$cos^2(\alpha) \cdot \tan^2(\beta) = \frac{1}{cos^2(\gamma)} - 1 \tag{5.19}$$

$$\tan^2(\beta) = \frac{1}{cos^2(\alpha)} \cdot \left( \frac{1}{cos^2(\gamma)} - 1 \right) \tag{5.20}$$

$$\tan(\beta) = \frac{1}{cos(\alpha)} \cdot \sqrt{\frac{1}{cos^2(\gamma)} - 1} \tag{5.21}$$

$$\beta = \tan^{-1}\left( \frac{1}{cos(\alpha)} \cdot \sqrt{\frac{1}{cos^2(\gamma)} - 1} \right) \tag{5.22}$$

$$\tag{5.23}$$

In three iterations, I adjusted the steering angle to several angles $\gamma$ on the ground plank and noted the output angles $\beta$ that have been measured by the RCE's sensor. I also measured several angles after shaking the bicycle while it was placed on the SATS, and after removing and repositioning the bicycle. The measured values did not change which ensures that the validation can be repeated. Prior to this validation, the

steering angle sensor has been calibrated. To determine the accuracy of the sensor compared to the calculated angles, I first calculated the average of all three iterations. The comparison of should-be values $\beta_{should-be}$ to measured values $\beta$, shows a similar distribution, but much less intense than in $\beta_{should-be}$. A reason for this could be the inaccuracy of the used potentiometer, in terms of a smaller working range. By reducing the working range in the steering angle calculation, the values can be adjusted to the actual working range. I computed the MAE and Mean Squared Error (MSE) for all smaller working ranges in steps of 0.1 %. Both, MAE and MSE are small, if the deviation between should-be and measured value is small. A reduction of the working range by 11.7 % resulted in minimum values, $MAE = 0.3°$ ($MSE = 0.14$). Since the potentiometer has a working range of $300° \pm 20\,\%$, a deviation of 11.7 % is plausible. Using the adjusted working range, my sensor achieves a MAE of 0.3°. Table 5.4 shows the average of the measured values, the values after adjusting to the smaller working range, and the should-be values from above calculations.

| Angles [deg] | | | |
|---|---|---|---|
| Measured | Adjusted | Calculated | Ground |
| 0.0719 | 0.0644 | 0.0000 | 0.0 |
| 1.3173 | 1.1793 | 1.0642 | 1.0 |
| 2.2951 | 2.0547 | 2.1282 | 2.0 |
| 3.2706 | 2.9280 | 3.1921 | 3.0 |
| 4.7639 | 4.2649 | 4.2558 | 4.0 |
| 5.8910 | 5.2739 | 5.3191 | 5.0 |
| 8.0100 | 7.1710 | 7.4444 | 7.0 |
| 11.1372 | 9.9706 | 10.6276 | 10.0 |
| 17.1889 | 15.3885 | 15.9153 | 15.0 |
| 22.9386 | 20.5359 | 21.1728 | 20.0 |
| 34.5631 | 30.9428 | 31.5667 | 30.0 |
| 52.2872 | 46.8104 | 46.7808 | 45.0 |
| 69.1746 | 61.9289 | 61.5188 | 60.0 |

**Table 5.4** – Comparison of the steering angles in the validation of the steering angle sensor. The *measured* values are the output of the sensor assuming the full working range. The *adjusted* values assume a 11.7 % smaller working range. The *calculated* values are the should-be values depending on the *ground* angle that has been set on the ground plank.

## 5.3 GPS

This section covers the validation of the recorded GPS tracks using various map materials.[37] The external GPS antenna uses DGPS and is WAAS-enabled. This means,

[37] All plots shown in this section are screenshots of the corresponding website, if it allowed uploading GPS tracks. Additionally, I used the following website that enables uploading several GPS tracks and viewing it on various map materials: https://www.j-berkemeier.de/ShowGPX.html

it uses corrected GPS data, if available, and the values should be accurate, already. To validate the system in a test ride, I first compared the recorded GPS track to several map data. In a second step, I compared the RCE track to two smartphone GPS tracks that have been recorded simultaneously. Additionally, the speed that has been derived from the GPS data, has already been compared to the other speed recordings. The GPS speed was underestimating due to the bicycle leaning into curves. Thus, the resulting distance traveled by the GPS antenna is shorter than the trail of the wheels. In related work this behavior was observed as well [14], [18]. Figure 5.9 shows the entire route using map material of Open Street Map (OSM).[38]

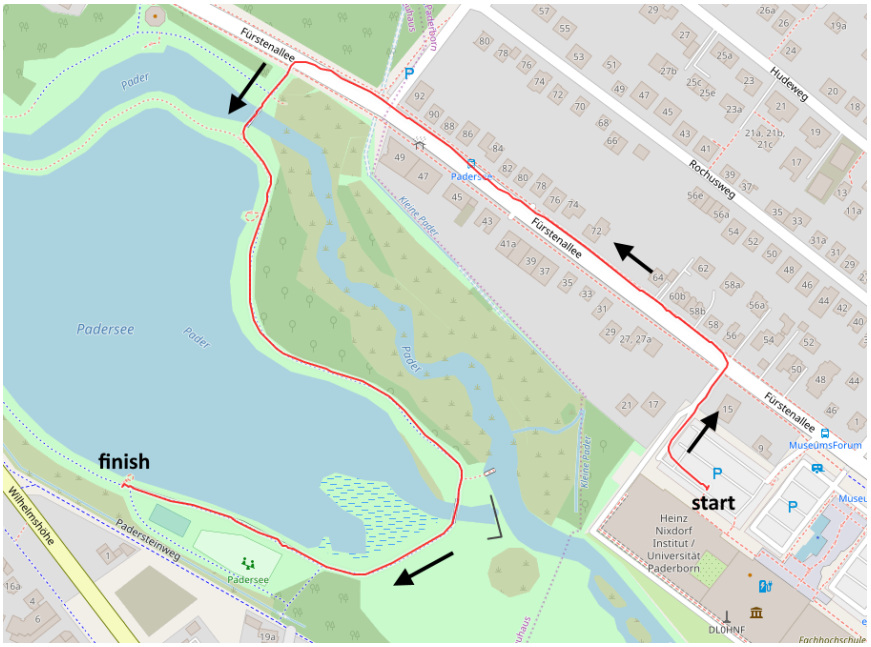



**Figure 5.9** – The driven route on an OSM map. The driving direction is indicated by black arrows. Start and finish point are marked accordingly.

### 5.3.1 Comparison to Map Data

I inspected the track on several maps, to avoid misinterpretations due to errors in map data of one map. The actual route started at the parking area of the university. Along a road with houses and large trees, the route led to a lake. Thus, both, urban and open areas were covered. The cyclist rode on the right side of the road, but didn't (mis)use the sidewalk. Then, the route led on a recently renewed cycle path around the lake.

[38] https://www.openstreetmap.org/

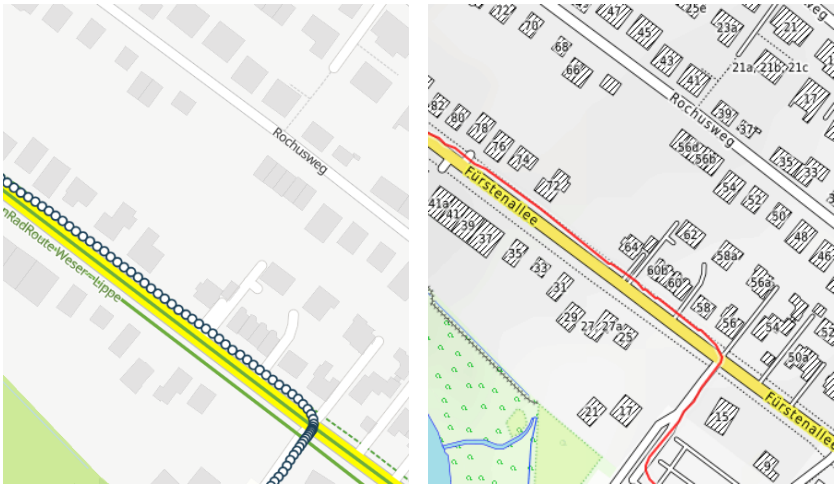
When uploading the track to Google Maps,[39] it seems like the track would lead through the lake, since the renewed path is not yet integrated, see Figure 5.10. On all other tested map data, this was already updated (construction was finished about one year ago). This example shows the need of comparison on several map data. Using bikemap.net, which is based on OSM, but draws the road (correctly) with two lanes, it looks like the track is located on the right side of the road, as shown in Figure 5.11a. With opentopo data,[40] the road is drawn thinner, and hence, the track seems to be besides the road, see Figure 5.11b. A clear statement on what part of the road the cyclist was riding can not be made. These comparisons show that it is not sufficient to use map data of a single source, but necessary to check the GPS track on several map data.

Deviations along the road can be seen at the red mark in Figure 5.12. I checked the video, to ensure that no swerving to the center of the road happened during the ride. On turning left into the path around the lake, however, it is possible to detect a lane change that actually was performed (blue mark in Figure 5.12).

[39] https://www.google.de/maps/
[40] https://opentopomap.org/


**(a)** Map data **(b)** Satellite images

**Figure 5.10** – GPS track (blue line) on Google Maps. (a) The map data is outdated and does not yet contain the new path, so it looks like the track goes through the lake at the spot, marked with a red arrow. On the satellite images (b) the construction site can be seen.




**(a)** bikemap.net (OSM based) **(b)** opentopo map

**Figure 5.11** – Comparison of two maps with different road width. The track is drawn as dots and as a red line, respectively. Driving direction is starting at the bottom, going to the top left. In (a) the yellow road is drawn with two lanes, one in each direction, so it looks like the track is positioned on the right side of the right lane. In (b) the road is drawn thinner and the track is positioned besides the road.

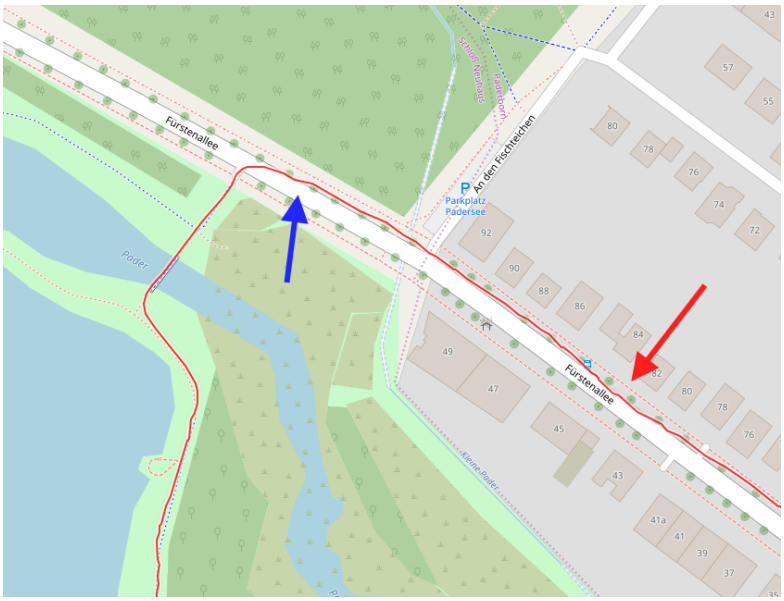


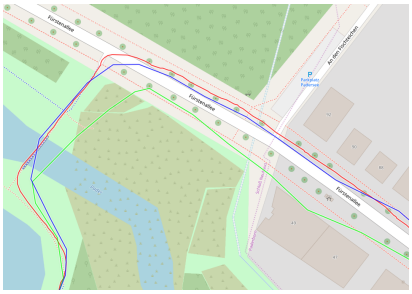

**Figure 5.12** – OSM data, the track is drawn as a red line. The red mark shows a swerving movement to the center of the road that was not performed but recorded. The blue mark indicates the lane switching prior to the left turn.

All in all, the result of this comparison is that it is possible to identify the choice of route on any map that is up-to-date. Though, predictions of the choice of lane, as well as, the position of the cyclist on the road, are hard to make. Deviations in the GPS signal, combined with map data that might not contain or display the exact width of the road or other details, complicate assumptions about the current situation of a recording. A video for detailed analysis or double checking parts of a recording, can be very useful.

### 5.3.2 Comparison of GPS Devices

Additionally to the GPS receiver of the RCE, two smartphones recorded a ride at a frequency of 1 Hz. The devices are a Samsung Galaxy S3 mini and a Honor 7S. Both smartphones were positioned at the same position as the external GPS receiver, to avoid deviations due to different positions on the bicycle. The full tracks can be seen in Figure 5.13a. Clearly visible, the GPS receiver (red) matches the map data the best. This was also the case with all other map material. In Figure 5.13b the high inaccuracy of smartphone GPS, especially the Honor (green), can be seen. Generally speaking, the smartphone GPS seems to cut off corners, which might be intensified by the lower recording frequency. An example for this, is given in Figure 5.13c. The result of this corner cutting are a too low total distance estimation and related to that a lower GPS speed value.


**(a)** full comparison



**(b)**

**(c)**

**Figure 5.13** – Comparison of GPS tracks recorded by external device ublox EVK-7N (red) and smartphone GPS of a Samsung Galaxy S3 mini (green) and a Honor 7S(blue). (b) The Honor 7S (green) deviates for large episodes for more than 10 m. Both smartphone tracks (green, blue) deviate from the actual driven route. (c) The smartphone tracks (green, blue) cut off corners. This is not only due to the lower frequency, but especially due to not using error corrections.

## 5.4 Brake

Each of the two brake sensors is based on a Hall effect sensor in combination with a tiny magnet that is attached to the movable part of the brake handle. Thus, the proximity of both which directly relates to the proximity of the brake disk and brake shoes can be measured. To validate the output values of one brake sensor, I compared its curve to the speed values in several rides with different brake application levels. I decided to use the back brake for this experiment, since the speed is measured at the back wheel, as well. During the tests, only that one brake was applied, and the second brake was left unused. There are many effects that cause a bicycle to slow down, some of which are surface and slope of the road, tire pressure, weight and cycling behavior of the rider, weather conditions, e.g., wind speed, and friction of the wheel hub. To cancel out all of these effects, a cyclist first rode without using brakes at all. In this way, I was able to determine the total deceleration caused by all non-brake effects during a non-accelerated straight forward ride on level ground. I didn't investigate velocities below 5.0 km/h, since the cyclist used the feet to further slow down and not fall over below that speed. A second ride with a specific brake application resulted in a deceleration curve different to that of the reference ride. The described experiment was repeated several times at different levels of brake application or without using the brakes. All iterations were executed by the same cyclist under equal conditions on the identical track. The total weight of rider and bicycle was 117 kg.

After deriving the speed curve to obtain the deceleration curve, I applied a simple moving average flattening algorithm (100 ms span, 20 times applied). This resulted in a more monotonous curve and can be done without problems, since the brake application was at a constant level during each ride, and fluctuations go back to the slight inaccuracies of the speed sensor or irregularities of the surface or slope. I noticed that the speed reduction was induced on application of the brake handle. Though, a constant deceleration level was only reached after 500 ms, see Figure 5.14. A reason for this delay might be that the brake shoes only fully engage after that time. Hence, the investigated sequence for each experiment begins 500 ms after brake application and ends when the speed is at 5 km/h. Especially for hard braking, this results in few data points, e.g., at 62 % brake application, the bicycle decelerates from 20 km/h to 0 km/h in just 2 s. For some recordings, I had to cut at a higher speed than 5 km/h, because the brake application increased or released before reaching 5 km/h. In Figure 5.15, the deceleration values in relation to the velocity are shown. The data of the un-braked and 19 %-braked experiments is still quite inconstant due to slight unevenness of the ground that caused the bicycle to decelerate irregularly. I grouped the experiments by their brake application level into six groups. The brake shoes start touching the brake disk at 19 %. From a subjective

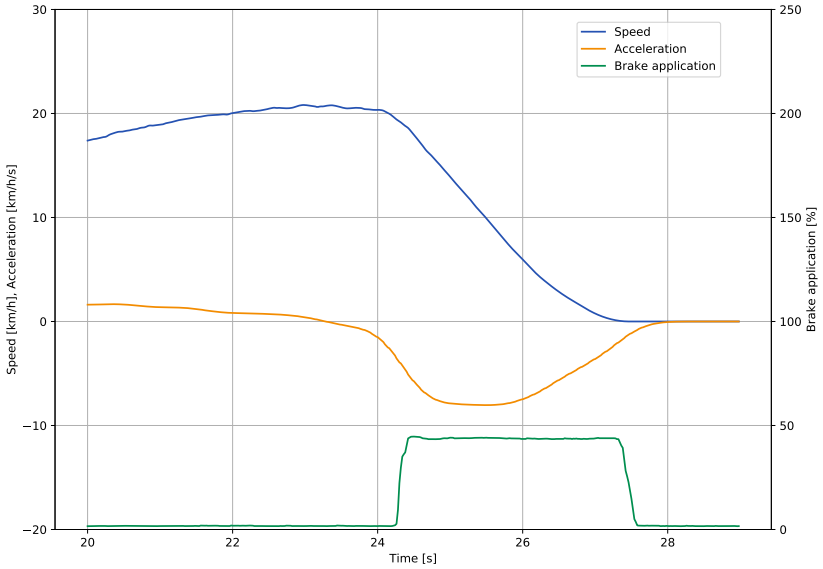



**Figure 5.14** – Sequence of one of the brake validation rides. The speed (blue) decreases from 20 km/h to 0 km/h. The resulting acceleration (orange) is given in km/h per second. A value of, e.g., −4 km/(h s) represents a deceleration by 4 km/h within one second. The brake application (green) initializes the brake maneuver (at 24.2 s), but the full deceleration level is only reached approx. 500 ms later. The negative acceleration prior to the brake application is caused by non-brake effects.

point of view, 26 % and 33 % are normal brake applications, 45 % is a harsh brake maneuver. At 62 % the wheel is almost blocking and the maximal deceleration is achieved. Higher brake application causes the wheel to block. Since, e.g., the friction resistance or aerodynamic resistance, increase at higher velocities, and this increase in deceleration was also visible in the experiments (see 0 % data points in Figure 5.15), I used polynomial regression to fit a quadratic function $f_0$ on the un-braked group. A regression algorithm fits a function with minimal error on a given set of data points. In this case, this function represents the *non-brake related* deceleration depending on the velocity.

Due to the forces on the cyclist's body during braking, it is difficult to hold the brake handle at a constant level trough the entire brake maneuver. To get a representative for each group, anyhow, I computed the mean brake application, as well as, minimum and maximum. I also removed recordings with changing brake application. As can be seen in Figure 5.15, the deceleration levels per group are almost constant over all velocity levels. Thus, the velocity seems to have no or only a small impact on the deceleration during braking. Further, it is clearly visible that

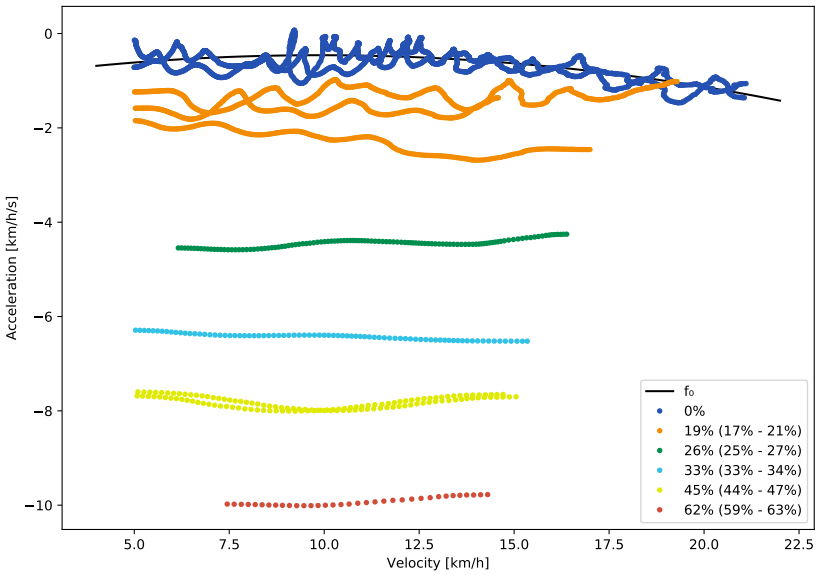



**Figure 5.15** – Acceleration depending on the velocity during brake maneuvers of different brake application levels. The fluctuations of the 0 % and 19 % curves are due to slight unevenness of the ground that caused the bicycle to slow down nonuniform. At higher brake application levels fewer data points are available, as the velocity decreases faster. $f_0$ is a quadratic curve that was fitted on the 0 % group using polynomial regression. It can be seen that a higher brake application corresponds with a higher deceleration.

higher brake application levels result in a higher deceleration what would also be expected.

By subtracting $f_0$ from each of the braked rides (depending on the velocity), I was able to compute the brake-caused deceleration. Using linear regression, I fitted a linear function on each group after subtracting $f_0$, see Figure 5.16.

I related the average brake application levels of each experiment group (similar to the labels in Figure 5.16) to the average deceleration of each group. The relation of brake application level and brake-caused deceleration is depicted in Figure 5.17. Additionally, Table 5.5 gives the absolute values. Of course, in this graph the speed information is lost. This is without problem, since the main speed related effects are already canceled out by the previously explained subtraction. For brake applications below 20 % no or almost no deceleration is detected, because the brake shoes are not in contact with the brake disk. At higher levels, one might expect that not much more brake application is needed, since pushing the brake shoes just a little more against the brake disk results in a much higher brake force. Though, for this tighter contact much more brake force is needed. Due to this increase the brake cable and cable housings get stretched and pressed together, respectively. Brake applications

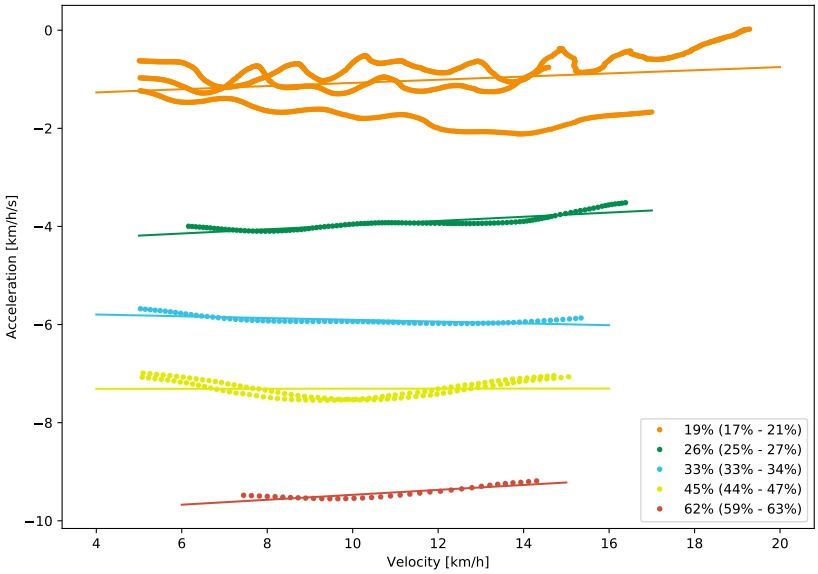



**Figure 5.16** – Brake-caused deceleration depending on the velocity during brake maneuvers of different brake application levels. The data points were computed by subtracting the non-brake deceleration (depending on the velocity) from each total deceleration value. I used linear regression to fit linear functions on each group. All functions are almost constant.

of more than 65 % caused the wheel to block. In this case, no further speed values can be measured.

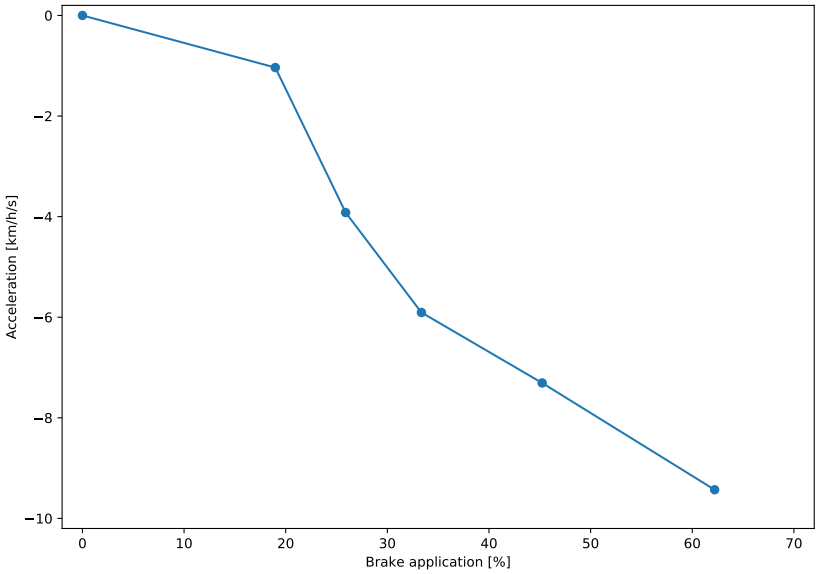



**Figure 5.17** – Deceleration depending on the brake handle application. Up to a brake application of 19 %, almost no deceleration is caused, as the brake shoes are not in contact with the brake disk. At higher brake applications (from on 33 %), much more brake application is needed, because brake cable and cable housing get stretched and pressed together, respectively. Again, the acceleration is given in km/h per second.

| Brake application [%] | Acceleration [km/h/s] |
|---:|---:|
| 0.0 | 0.000 |
| 19.0 | −1.036 |
| 25.9 | −3.919 |
| 33.4 | −5.905 |
| 45.2 | −7.307 |
| 62.2 | −9.430 |

**Table 5.5** – Deceleration depending on the brake handle application. The acceleration is given in km/h per second.

# Chapter 6

# Evaluation

This chapter focuses on the evaluation and usability of the RCE. First, the system's capability to record bicycle characteristics is evaluated. A look on the efficiency regarding resource usage, storage usage, and power consumption, gives an estimate for how long the RCE can be used, and if or how many further sensors could be added. For this estimate, the used recording frequency is a decisive factor. Finally, I will introduce a reviewing software that is capable of replaying and analyzing traces that have been recorded with the RCE.

## 6.1 Capabilities of the RCE

The requirements for and desired functionality of a measurement framework have been presented in Section 4.2. The RCE has sensors to measure speed, steering angle, GPS position and brake application. It, further, can record a video of the ride for later analysis. The measurement frequency of all sensors exceeds the minimum requirement of 10 Hz. Only for very low speeds below 8.3 km/h, the speed sensor has a lower frequency. Below 1.0 km/h it assumes that the bicycle stopped. The maximum frequencies for all sensors are given in Table 6.1. This holds for a measurement with all sensors active at the given maximum frequency. Configuration files can be used to define the recording frequency and to enable or disable sensors. The stated maximum frequencies can be exceeded a little, if only few sensors are active.

However, the recordings turned out to, sometimes, have gaps of up to 2 s in it. During that time, no values are measured or stored. I investigated the erroneous behavior, but wasn't able to identify the source of missing values. The gaps occur on any sensor independently, thus, not at the same moments in time, but apparent randomly for a few milliseconds up to 2 s, approximately once over a time of 3 min. Single sensors, the communication between Raspberry Pi and ADC, and the ADC

| Sensor | Maximum frequency |
|---|---|
| Speed | 70 Hz |
| Steering angle | 1875 Hz |
| GPS | 18 Hz |
| Brakes | 1500 Hz |
| Camera | 30 fps |

**Table 6.1** – Maximum recording frequencies when recording with all sensors at once. The speed sensors frequency depends on the current velocity and cannot be adjusted. GPS and camera frequency are limited by the external device. Steering angle and brake frequency are limited by the Raspberry Pi.

itself, can be ruled out as source of the gaps, because they occurred on all sensors and even the camera. I checked the CPU usage during the gaps, but found that all cores were below 80 % working load all the time. Additionally, the entire recording script runs on one process in concurrent threads. Thus, process changing times can not be the reason. Process displacement by, e.g., system processes, are eliminated, as well, because this would result in simultaneous gaps of all sensors. Too many writing operations during the measurement could cause delays. By using a flash drive instead of the SD card as storage device, and by recording at lower frequencies or with just one sensor at low frequencies, the problem couldn't be eliminated, either. Hence, writing speed can be ruled out, as well. However, using less sensors and a lower frequency, resulted in fewer gaps.

After a recording the raw data gets converted to VCE-like trace that contains sensor data at a desired, constant frequency. On the Raspberry Pi this conversion might take a while, especially for long recordings with many sensors. To solve this problem, the conversion can be done afterwards on any computer, as well.

The RCE is designed expandable. Hardware can easily be added, e.g., only three of the eight channels of the MCP3008 are in use, and many GPIO pins are still available, as well as three of the four USB ports. The recording software is designed expandable. It allows adding new sensors that can, e.g., reuse writing and conversion methods, and can be shown by the live printer. In Section 6.3 and Section 6.4, I analyzed CPU and power consumption which are both far from being fully exploited.

Controlling the RCE is possible with, e.g., a smartphone. During recordings for validation and evaluation, the control via smartphone turned out to be very convenient, as there is no need for the cyclist to get off the bike. The command line arguments made it easy to use different configurations or repeat the previous experiment with very few clicks. Additionally, the live printer was helpful during development, to quickly check whether a newly developed sensor works correctly.

As part of the framework, the live printer enables straightforward development of sensors.

The entire system is to a degree water repellent, as, e.g., Raspberry Pi, camera and GPS antenna are packed in plastic bags, and the sensors are secured with hot glue. Thus, unexpected slight rain or wet roads do not destruct the measurement system.

A listing of the approximate prices is given in the Appendix Table A.1. When excluding the GPS device, the entire system can be build about 100 Euro.

## 6.2 Accuracy vs. Storage Usage

For the sensors that are periodically reading and storing the current value, i.e., steering angle and brake sensors, the storage usage during a recording depends on measuring frequency and precision of the stored value. Defined by the ADC, the highest precision is 10 bit. I took a recording that measured steering angle and brake application at a very high frequency and reduced the frequency and precision afterwards to compare the reduced values to the original values and determine the accuracy. Algorithm 6.1 gives an overview of the workflow for getting lower frequency and precision recordings of an identical ride.

**Require:** Ground truth $G$ data set, list of desired parts of frequency $F$, list of desired precision degrees $P$
**Ensure:** $|F| \cdot |P|$ different output data sets with lower frequency and precision according to $F$ and $P$
1: Output data sets $O \leftarrow \emptyset$
2: **for** $f \in F$ **do**
3: $R_f \leftarrow$ set of each $f$-th entry in $G$
4: **for** $p \in P$ **do**
5: $R \leftarrow$ set of each entry of $R_f$ scaled to a range of $2^p$ values
6: $O \leftarrow O \cup \{R\}$
7: **end for**
8: **end for**
9: Return $O$

**Algorithm 6.1** – Reduce a ground truth data set's frequency and digit precision, to obtain reduced data sets of an identical recording. $G$ is the original data set. $F$ is a list of frequency parts that should be taken. $f \in F$ means that each $f$-th entry will be used. $P$ is a list of precision values that should be taken. $p \in P$ means that each entry's last $(10 - p)$ bits are set to 0. Thus, $p$ bits of information remain.

The steering angle was recorded at 1875 Hz, which is ten times faster than the recommended 187.5 Hz.[41] By reducing both, the frequency and the precision, to several levels, I got various combinations to test against the ground truth which is the original recording. I reduced frequency and precision in several steps down to 15 Hz and just one bit of information. On each reduced data set, I applied the identical noise reduction as after a usual recording. Then, I compared the reduced version to the original recording, by computing the MAE. A comparison depending on frequency for different precision levels (number of bits) is shown in Figure 6.1a. I left away the lower precision levels in the plot, as they just increase in error and are almost constant. At a time $t \in T$, the MAE of the reduced data set $R$ compared to the corresponding ground truth $G$, can be calculated as follows. Let $r(t) \in R$ the values of the reduced data set, $g(t) \in G$ the values of the ground truth.

$$MAE = \frac{1}{|T|} \cdot \sum_{t \in T} (|g(t) - r(t)|) \tag{6.1}$$

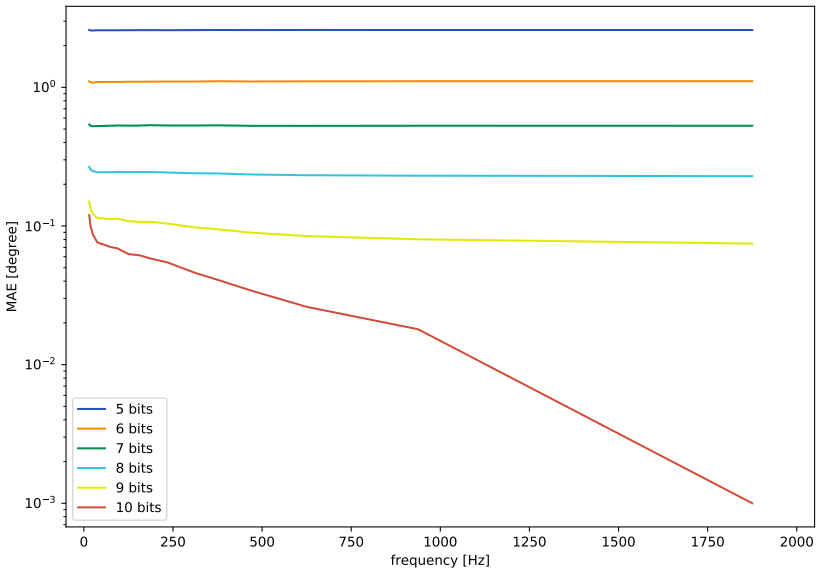
The brake application was recorded in a similar way to the steering angle (periodically reading the value of the ADC), but at a frequency of 1500 Hz. Repeating the evaluation for the brake sensor might seem redundant, as it is comparable to the steering angle, especially the noise is very similar. Though, I expected possibly different results, as the behavior in terms of brake application is much different to that of the steering angle. The brake value is rarely, but rapidly changing with a large amplitude, while typical steering angles swerve around a value and rarely use the full amplitude. Figure 6.1b shows the comparison of frequencies and precision levels for the brake sensor.

From both figures, it can be seen that the number of bits is most important for the accuracy. Especially at frequencies above 200 Hz no large gain in the accuracy is made by further increasing the frequency.

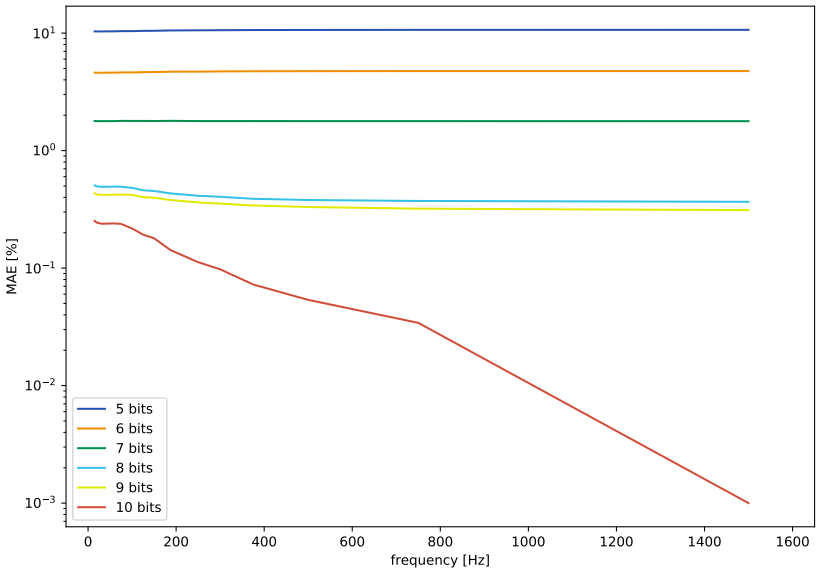
Different levels of detail (frequency and precision) result in different storage usage. To give an orientation of what level of detail is worth the storage I also related the MAE to the storage usage per second instead of the frequency. Since the fewer bit values are again, much higher than the others, Figure 6.2 just shows the 8+ bit curves. It can be seen that for all storage consumption levels the lowest MAE is achieved with 10 bits. Due to the large improvements prior to the 2.5 kB/s, I recommend to use at least 2.5 kB/s.

Albeit, it has to be said that a recording of 4 min 26 s required a total of only 17.2 MB for all sensor data at highest frequency and precision. Thus, 1 GB of storage would suffice for recording more than 4 h 25 min, or 32 GB for more than five days.

[41] As described in Section 4.6.2, by using 187.5 Hz and 150 Hz for steering angle and brake sensors, respectively, one peak and two correct values are measured per interval of 16 ms and 20 ms, respectively. This is required for the noise reduction algorithm to work correctly.




**(a)** Steering angle




**(b)** Brakes

**Figure 6.1** – MAE in relation to used frequency. The 10 bit curves for both, steering angle and brakes, are lowest for all frequencies. At the highest frequency, the 10 bit curve has an MAE of 0, as it is the ground truth. In log scale, 0 can not be represented and would be infinitely low. Hence, I set that value to $0.001 = 10^{-3}$ as the minimum value which represents "no error".

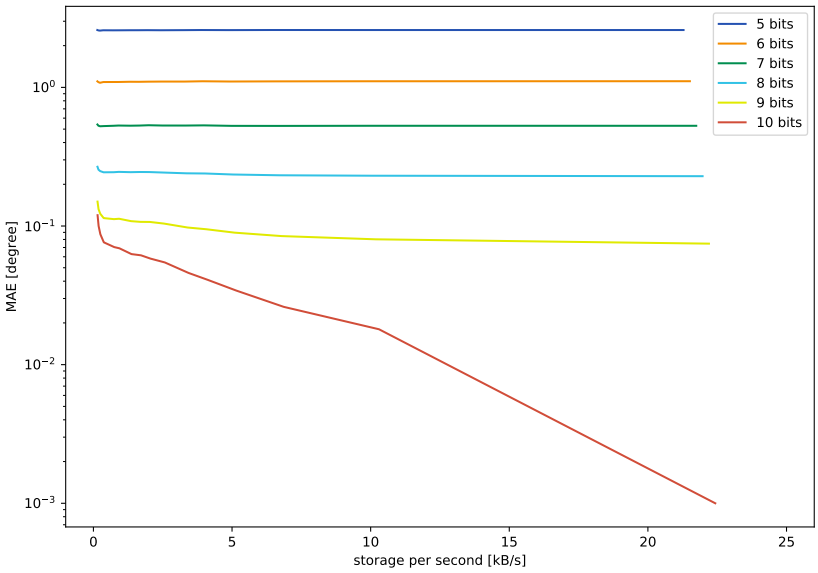



**Figure 6.2** – Comparison of MAE and storage usage per second. At all storage usage levels, the 10 bit curve is the lowest which means that using lower precision does not bring any advantage. Hence, I recommend to always use highest precision. Again, I set the minimal possible value to $0.001 = 10^{-3}$ which represents "no error".

Further, a flash drive can be used as storage device, so no internet connection of the Raspberry Pi is needed for transferring the recordings to a computer, and swapping the storage can be done easily. However, when recording a video of the ride, of course, much more storage is used. For comparison, the sequence of 4 min 26 s took 134 MB (frame rate: 30 fps; resolution: 1280x720). Hence, reducing frame rate or resolution might be a better option to achieve a reduction in storage usage than reducing sensor accuracy.

## 6.3 CPU and Memory Usage

The CPU usage of the Raspberry Pi depends on the current tasks, e.g., recording script, WAP activity, and operating system tasks. To determine the dependency of CPU usage on the selection of sensors and the used configuration of those, I implemented a *system sensor* that is recording the current CPU and memory usage. In this section, I will refer to all sensors and recording related devices as *sensors*, including the camera or GPS device. I tested several configurations with the frequencies listed in Table 6.2. The speed sensor has no adjustable frequency, as it is trigger-based. No changes appeared on different frequencies of the GPS device. A reason for this

| Configuration | Sensors | | |
|---|---|---|---|
| | Steering angle frequency [Hz] | Brakes frequency [Hz] | Camera frame rate, resolution |
| max | 1875 | 1500 | 30 fps, 1280x720 |
| med | 625 | 500 | 24 fps, 960x540 |
| min | 187.5 | 150 | 2 fps, 640x360 |

**Table 6.2** – Configurations for steering angle and brake sensors, and camera. All other sensor don't have an adjustable frequency or changes didn't have any impact. The configuration categories are: max (maximum), med (intermediate), and min (minimum).

might be that even the highest available frequency of approx. 18 Hz is still quite low compared to, e.g., the brake sensor which works at a frequency $\geq 150$ Hz. The results show that the CPU usage increases from 15.3 % with no active sensors (idle) to a maximum of 38.6 % with all sensors enabled and at maximum frequencies.

All values are given in Table 6.3. For minimum and average frequencies, the CPU usage accumulates from the single-sensor configurations to the all-sensors configuration (only a slightly smaller value for the all-sensor configuration). Figure 6.3 shows the comparison. The maximum frequency, however, did not further increase the CPU usage in an accumulative way but just by 0.6 %. The corresponding accumulated increase would have been 20.0 %. Over all frequencies, a combination of sensors resulted in a slightly lower total CPU usage compared to the accumulated values. A reason for this could be that reading from two channels of the ADC doesn't take up twice as much processing power as reading from one channel, or that the sensor threads get scheduled in a more efficient way.

The overall result of my CPU usage analysis is that the recording frequency does have an impact, but even at the highest frequency only less than 40 % of the CPU capacity is used. Almost no impact on the CPU usage is caused by the speed sensor, and high frequency ADC reading causes the highest increase in CPU usage. At no

| Config | CPU usage [%] | | | | | | |
|---|---|---|---|---|---|---|---|
| | Idle | Camera | Speed | Steering angle | GPS | Brakes | All |
| max | 15.3 | 17.3 | 16.7 | 31.6 | 19.2 | 36.0 | 38.6 |
| med | - | 16.6 | - | 22.5 | - | 25.8 | 38.0 |
| min | - | 15.4 | - | 18.3 | - | 18.0 | 27.0 |

**Table 6.3** – Average CPU usage during measurements using various configurations. It can be seen that higher frequency configurations reach higher CPU usages. The highest value (38.6 %) is reached when using all sensors at maximum frequencies. This value, is not much higher than that of the intermediate (med) configuration, though.

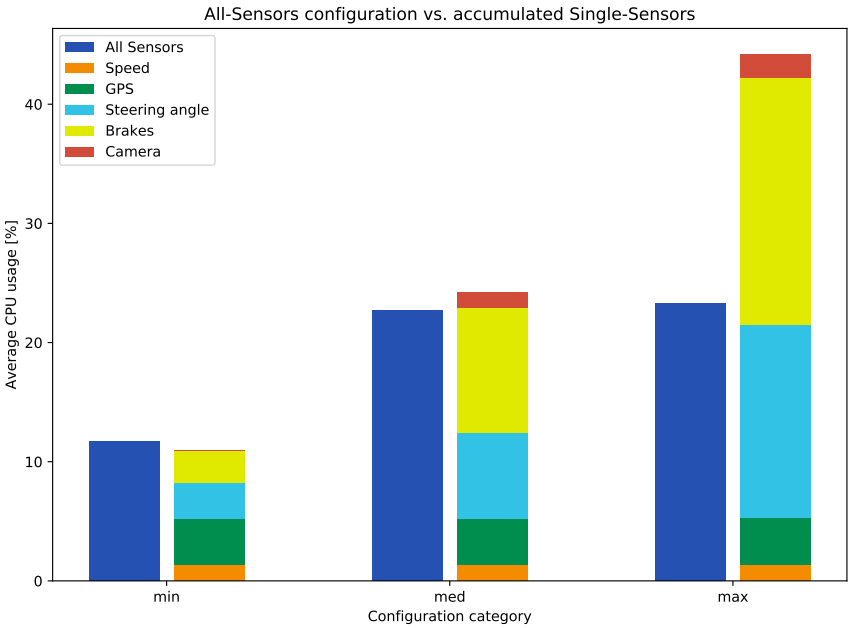



**Figure 6.3** – Comparison of CPU usage (above idle) of different configurations on three frequency levels, see Table 6.2. The *all sensors* configuration records all sensors at once. The other configurations record only the given sensor. It can be seen that the accumulated single-sensor CPU usages are similar to the all-sensors configuration for minimum (min) and intermediate (med) frequencies. The maximum (max) frequency increased the CPU usage only slightly (0.6 %) compared to the intermediate frequency when recording all sensors at once. A reason for this might be a better interleaving of the sensor threads.

time the usage of a single core was higher than 85.7 %. Hence, no delays can be caused by an overcharge or insufficient processing speed.

I compared the memory usage on the identical set of tests. Though, I wasn't able to identify any changes due to the measurement script, or any changes that could be explained by the active sensors. The highest amount of memory was in use when recording with the minimum camera configuration, the second highest with the idle configuration. All in all, the memory usage fluctuated between 25 % and 26 %. My result regarding the memory usage is that the Raspberry Pi's memory is more than sufficient for the developed measurement framework. This result is as expected, since my measurement methods do not store large amounts of data in arrays or variables, but continuously write the measured values to the data storage device, e.g., a flash drive.

## 6.4 Power Consumption

Similar to the CPU usage, the power consumption of the Raspberry Pi depends on the current tasks. To measure the current flow and determine the power consumption, I split up a USB cable that can be used to power the Raspberry Pi with a mobile charger. By cutting the +5V wire and connecting an ampere meter in between, I was able to measure the current flow. The flow fluctuated a lot which made manual reading difficult and error prone. I made use of the averaging function of the ampere meter that displays the average of all measured values after enabling the function. Each configuration was measured for at least 30 s. I used the same configurations as in the CPU usage analysis, see Table 6.2. The results of the power consumption measurements are shown in Table 6.4. Additionally, I measured the power consumption in both wireless modes (client in a WLAN, and server/WAP) and found a value of 264 mA for both (no active measurement). The value didn't change after assigning a device to the WAP network and connecting via SSH, either. Thus, no additional power is consumed when establishing a WAP. I started a measurement with the idle configuration. It doesn't measure anything (except CPU and memory usage) but has the live printer enabled, and hence, gives a realistic baseline for a comparison to configurations with active sensors. The current flow using the idle configuration was 324 mA. With all sensors enabled and at the maximum frequency, the power consumption was 614 mA. The used mobile charger has a maximum output current of 2400 mA, and thus, the capabilities are utilized to about a fourth, only.

Similar to the CPU usage analysis, I compared the values of each configuration to the idle value, see Figure 6.4. The accumulated power consumption (above idle) matches to the value of the all-sensors configuration. Again, the all-sensors configuration yielded slightly lower values which also fits to the results of the CPU usage analysis. Regarding the speed sensor, I noticed that no additional power is used

| Config | Average current flow [mA] | | | | | | |
|---|---|---|---|---|---|---|---|
| | Idle | Camera | Speed | Steering angle | GPS | Brakes | All |
| max | 324 | 471 | 334 | 360 | 396 | 379 | 614 |
| med | - | 450 | - | 346 | - | 354 | 584 |
| min | - | 394 | - | 329 | - | 342 | 486 |

**Table 6.4** – Average current flow (mA) during measurements using various configurations. Each column shows the current flow when using the sensor(s) stated above. The values increase when using higher frequencies or more sensors. The highest value (614 mA) is reached when using all sensors at maximum frequencies. The idle value was measured with all sensors disabled, but using the live printer. This results in a realistic baseline for estimation of power consumption per sensor.

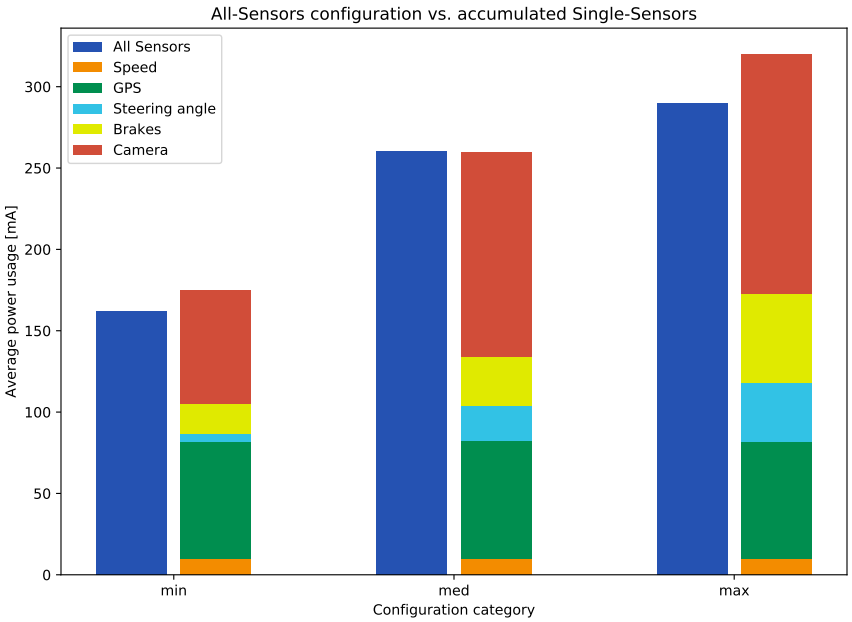



**Figure 6.4** – Comparison of power consumption (above idle) of different configurations on three frequency levels, see Table 6.2. The *all sensors* configuration records all sensors at once. The other configurations record only the given sensor. It can be seen that the accumulated single-sensor power consumption is similar to the all-sensors configuration at all frequencies. The biggest consumption is caused by the Raspberry Pi Camera and the external GPS device.

during the standstill of the wheel. This is explained by the speed sensor being trigger-based, thus, at standstill nothing is done. At a speed of approximately 25 km/h, the current flow was at 10 mA (above idle) which is the value I used for comparison to the idle (during all measurements the wheel was at a speed of 20 km/h - 30 km/h).

The power consumption can limit the usage time of a recording system. With all sensors active at highest configurations, the current flow was $I = 614\,\text{mA}$. Typically, the capacity of mobile chargers is given in mAh. I used a mobile charger with a capacity of 10 400 mAh. Hence, the maximal recording time $t$ is:

$$t = \frac{10\,400\,\text{mAh}}{614\,\text{mA}} \approx 16\,\text{h}\ 56\,\text{min} \tag{6.2}$$

The more conventional way of stating the power consumption is as follows. For voltage $V$ and current flow $I$, the power $P$ (unit: W) can be computed by:

$$P = V \cdot I \tag{6.3}$$

At the USB voltage level of $V = 5\,\text{V}$ this results in a power consumption of $P = 3.07\,\text{W}$ at highest load.

## 6.5 Trace Viewer

To enable reviewing a recording and all sensor data, I created a Graphical User Interface (GUI), the RCE TV. It is similar to a traditional media player, but split into four screen parts that display (1) the video, (2) a map containing the GPS track, (3) the current steering angle, and (4) the current speed and brake application. Since


**Figure 6.5** – Screenshot of the RCE TV during playback of a recording. The screen is split into four parts, showing the video, the GPS track on a map, the current steering angle, and current speed and brake values. The bad video quality is caused by slight rain during the recording. On the bottom of the video frame, the current speed (e-bike engine) is displayed. In the shown scene a harsh brake maneuver was performed which also can be seen by the high brake values (displayed on the right). Using the video it is clearly visible that the speed reduced radically. The e-bike display, however continues to show the previous speed, i.e., 17.1 km/h. The RCE's speed sensor detected the speed reduction and shows a velocity of 9.6 km/h. Next to the handlebar symbol (bottom left) the steering angle during the entire recording is shown in a line chart. Speed (green) and brake application (orange and red) of the entire recording are shown in the second line chart (bottom right). The control bar can be used to control recording playback, skip single frames back or forth, or change the playback speed.

drawing a map from raw map material, e.g., OSM, requires a lot of work and was not in the focus of my thesis, I decided to use a web view to show a website that the GPS track has been uploaded to manually. Simple graphs give an overview of the steering angle (degree), speed (km/h) and brake application (100 % correspond to 50 km/h), for the entire recording. Additionally, a drawing of a handlebar and a typical speedometer move corresponding to the current value. The control buttons in the bottom can be used to open or close a recording by selecting a folder, to play/pause the video, to skip through it frame by frame, go to first or last frame, and to change the playback speed of the video.[42] The RCE TV reads the output files of the RCE after conversion, and uses the configuration file to match the output values and frequency to the video file. If no video was recorded, a virtual player is used to manage the playback. I added this feature, because a video is likely to be not available, as it consumes a lot of storage, but is not always needed. A screenshot of the RCE TV is shown in Figure 6.5. It gives a good example of the deviation between RCE speed sensor and e-bike display, due to inaccuracy and slowness of the latter, and delay between both. After applying the brakes, the bicycle slows down, which can also be seen in the video and the RCE's speed values decrease accordingly. The display however, continues to show a higher speed.

[42] The audio button has currently no function, as the RCE is not recording any audio. If, however, the video is recorded with an external camera that includes audio, e.g., a smartphone, it can be inserted into the folder of the recording. In this case, the audio button can be used to regulate the volume.

# Chapter 7

# Conclusion

The Real World Cycling Environment (RCE) is a measurement framework that is capable of recording bicycle characteristics. Recorded traces are comparable to those of the Virtual Cycling Environment (VCE). In the present work, I described the developed architecture, i.e., the data acquisition unit and sensors for speed, steering angle, Global Positioning System (GPS) information, and brake application, as well as the recording software chain. To ensure reliable data, I calibrated and validated each sensor. In this chapter, I will briefly explain my results and give examples for future work.

## 7.1 Results

In related work, the calibration of speed sensors was often done with inaccurate and/or low frequency reference values [5], [6]. I didn't want to rely on low frequency reference values, e.g., from a standard bicycle speedometer with one signal per revolution. Hence, I calibrated the sensor very precisely by determining the exact distance per revolution. After calibration, errors could only come from missing or double signals. To avoid these general errors in the measurement, I compared the RCE's speed sensor to two low accuracy signals, i.e., the onboard speedometer of the Electrical Bike (e-bike), and the speed determined by GPS data. All three sources produced similar values.

I built the Steering Angle Test Station (SATS), a special construction that can hold the bicycle on it, to validate the steering angle sensor. It allows comparing the measured angle at the handlebar to the real angle on the ground. Its accuracy is given with a Mean Absolute Error (MAE) of 0.3°.

A comparison of the GPS track to numerous map material and two smartphone GPS traces that have been recorded simultaneously, showed that the RCE's GPS track

was most accurate. However, I also identified some deviations due to the general lack in perfect accuracy of GPS that has been investigated in related work.

The brake sensor was validated by comparison to the speed data and the acceleration derived from the speed. This resulted in a relation of brake handle application to brake-caused deceleration that can, e.g., be used in bicycle simulators.

Videos of test rides turned out to be very useful in the trace analysis, e.g., to check whether the bicycle actually stopped in a specific traffic situation which was detected by the RCE's speed sensor, but not by onboard speedometer and GPS. The RCE Trace Viewer (RCE TV) was useful in reviewing traces, and thus, showing that the system can be compared to VCE traces.[43]

A drawback of the RCE are gaps in the recordings whose source could not be identified. Though, some probable sources were cut out. Neither resource usage of the Raspberry Pi, nor erroneous behavior of a single sensor or the Analog Digital Converter (ADC), cause these gaps that occur for up to 2 s per approximately 3 min of recording sequence.

Expandability is a main feature of a framework. The evaluation showed that the RCE's resources are far from being fully exploited. The CPU usage didn't exceed 38.6 % during recordings with all sensors and highest accuracy. No significant differences in memory usage were detected in comparison of no-sensors-active to all-sensors-active configurations. The power consumption during highest load was 614 mA which is around a 4th of the maximum output of the mobile charger. Both, hardware and software, can be extended easily. Thus, adding more sensors and functionality to the framework can be done without problems.

## 7.2 Future Work

The possibilities for future work are widely spread. Some of the sensors can be improved for higher accuracy or reliability. In the recordings, I noticed that the steering angle during normal rides is in the range of ±20°. By using oval gear wheels, instead of round ones, for the steering angle sensor, more of the potentiometer's range could be translated to this area while maintaining a possible full range of 75°. The speed sensor is limited by the interrupt handling speed of the Raspberry Pi. A microchip that is processing the signals and is connected to the Raspberry Pi could solve this problem.

I recommend positioning the data acquisition or control unit of a framework at the handlebar to improve the usability during experiments. This has two advantages

[43] I did not actually compare RCE and VCE traces on a specific route, since no VCE traces of the region were available. The RCE TV, however, showed that it is possible to use traces in an entirely different software environment. Hence, reading and comparing the RCE traces will be possible for future comparisons.

against a placement on the back rack: Operating the buttons of the RCE is much easier when the device is already in reach of the riders hands. And second, shorter cables to the steering angle and brake sensors might reduce noise in the signal.

To extend battery life during experiments, or to build an autonomous recording system, a solar panel could be used to charge the mobile charger. Thus, recharging the data acquisition unit's battery manually, would no longer be necessary, and recording traces during everyday life would be possible.

Of course, there are several sensors that could be added to the system. Similar to the speed sensor, a magnet in the pedal crank could be used to measure the pedaling frequency. An Inertial Measurement Unit (IMU) on the mainframe can be used to record acceleration data, as well as the leaning angle of the bicycle. Alternatively, a smartphone and a corresponding app that records values of the builtin IMU could be used to deliver such data. Using the Secure Shell (SSH) connection and a command line on a smartphone, is sometimes not very convenient. Hence, developing an app can improve usability, as well.

The recording software stores all measured values as plain text, even though mainly numbers and very few other symbols are stored. A reduction in storage usage can be achieved by compressing the raw data.

# Appendix A

# Price Listing RCE

| Hardware | Price [Euro] |
|---|---|
| Raspberry Pi 3B+ | 39.50 |
| Raspberry Pi HAT board | 4.20 |
| MCP3008 | 4.59 |
| Capacitors. resistors | 2.00 |
| Buttons. Light Emitting Diodes (LEDs) | 4.00 |
| Reed switch | 3.83 |
| Reed switch mounts | 5.40 |
| Magnets | 4.49 |
| Potentiometer | 3.49 |
| Sleeves | 2.40 |
| Rope | 0.60 |
| ublox EVK-7N | 220.10 |
| Hall effect sensors | 4.35 |
| Magnets | 4.60 |
| Raspberry Pi Camera | 9.95 |
| Ribbon cable 2m | 4.08 |
| Cables | 5.00 |
| Hot glue. cable ties | 2.00 |
| Total | 324.58 |

**Table A.1** – Prices for the required hardware of the RCE, based on current internet prices.

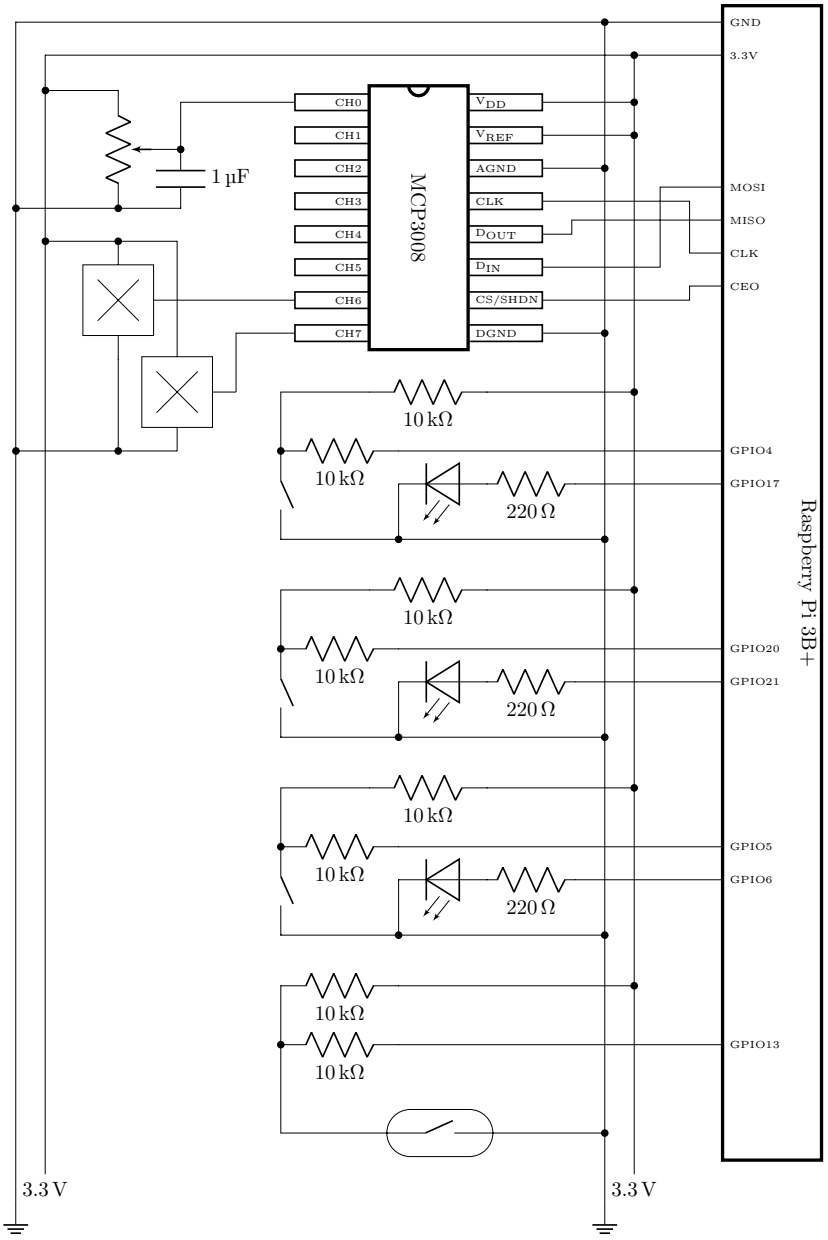



**Figure A.1** – Full wiring of the Real World Cycling Environment (RCE).
Top left: Potentiometer (steering angle)
Top left below: Hall effect sensors (brake)
Top center: MCP3008 connecting sensors to the Raspberry Pi
Center: Three buttons and LEDs for user interaction/feedback
Center bottom: Speed sensor with Reed switch

# List of Abbreviations

| | |
|---|---|
| **AC** | Alternating Current |
| **ADAS** | Advanced Driver Assistance System |
| **ADC** | Analog Digital Converter |
| **DC** | Direct Current |
| **DGPS** | Differential GPS |
| **e-bike** | Electrical Bike |
| **EGNOS** | European Geostationary Navigation Overlay Service |
| **GAGAN** | GPS Aided Geo Augmented Navigation |
| **GND** | Ground |
| **GPIO** | General Purpose Input Output |
| **GPS** | Global Positioning System |
| **GUI** | Graphical User Interface |
| **HAT** | Hardware Attached on Top |
| **IC** | Integrated Circuit |
| **IMU** | Inertial Measurement Unit |
| **LED** | Light Emitting Diode |
| **MAE** | Mean Absolute Error |
| **MSAS** | Multi-functional Satellite Augmentation System |
| **MSE** | Mean Squared Error |
| **NMEA** | National Marine Electronics Association |
| **OS** | Operating System |
| **OSM** | Open Street Map |
| **RCE** | Real World Cycling Environment |
| **RCE TV** | RCE Trace Viewer |
| **RP** | Raspberry Pi 3B+ |
| **RPM** | Rounds Per Minute |
| **SATS** | Steering Angle Test Station |
| **SBAS** | Satellite-Based Augmentation System |
| **SPI** | Serial Peripheral Interface |
| **SSH** | Secure Shell |

| | |
|---|---|
| **t-bike** | Traditional Bike |
| **UART** | Universal Asynchronous Receiver Transmitter |
| **VCE** | Virtual Cycling Environment |
| **VRU** | Vulnerable Road User |
| **WAAS** | Wide Area Augmentation System |
| **WADGPS** | Wide-Area Differential GPS |
| **WAP** | Wireless Access Point |
| **WLAN** | Wireless Local Area Network |

# List of Figures

# List of Tables

# Bibliography


[1] EC, "Road safety in the European Union," European Commission, Luxembourg City, Luxembourg, Report MI-AC-18-001-EN-N, Apr. 2018. DOI: 10.2832/060333.

[2] J. Heinovski, L. Stratmann, D. S. Buse, F. Klingler, M. Franke, M.-C. H. Oczko, C. Sommer, I. Scharlau, and F. Dressler, "Modeling Cycling Behavior to Improve Bicyclists' Safety at Intersections – A Networking Perspective," in *20th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2019)*, Washington, D.C.: IEEE, Jun. 2019. DOI: 10.1109/WoWMoM.2019.8793008.

[3] L. Stratmann, D. S. Buse, J. Heinovski, F. Klingler, C. Sommer, J. Tünnermann, I. Scharlau, and F. Dressler, "Psychological Feasibility of a Virtual Cycling Environment for Human-in-the-Loop Experiments," in *Jahrestagung der Gesellschaft für Informatik (INFORMATIK 2019), 1st Workshop on ICT based Collision Avoidance for VRUs (ICT4VRU 2019)*, C. Draude, M. Lange, and B. Sick, Eds., vol. LNI P-295, Kassel, Germany: Gesellschaft für Informatik e.V., Sep. 2019, pp. 185–194. DOI: 10.18420/inf2019_ws21.

[4] M. Hubbard, R. A. Hess, J. K. Moore, and D. L. Peterson, "Human Control of Bicycle Dynamics with Experimental Validation and Implications for Bike Handling and Design," in *2011 NSF Engineering Research and Innovation Conference, Poster Session*, Atlanta, GA, Jan. 2011.

[5] J. Y. Chen, F. Wen, C. H. Lin, and D.-F. Lin, A new automatic measurement system for a bicycle. Taipei, Taiwan: Piscataway, 2005. DOI: 10.1109/ICMECH.2005.1529311.

[6] A. Tanaka, T. Douseki, Y. Umeki, H. Kawaguchi, M. Yoshimoto, K. Tsunoda, and T. Sugii, "Batteryless sensorless bicycle speed recorder with hub dynamo and STT-MRAM," in *IEEE Sensors Journal*, vol. 14, IEEE, 2015, pp. 1–4. DOI: 10.1109/ICSENS.2015.7370539.

[7] J. D. G. Kooijman, A. L. Schwab, and J. P. Meijaard, "Experimental validation of a model of an uncontrolled bicycle," in *Multibody System Dynamics*, vol. 19, Springer, 2008, pp. 115–132. DOI: 10.1007/s11044-007-9050-x.

[8] J. K. Moore, "Human control of a bicycle," PhD Thesis, Department of Mechanical and Aerospace Engineering, University of California, Davis, Davis, CA, Aug. 2012.

[9] G. Lindsey, S. Hankey, X. Wang, A. Gorjestani, and J. Chen, "Feasibility of Using GPS to Track Bicycle Lane Positioning," Department of Mechanical Engineering, University of Minnesota, Minneapolis, MN, Technical Report, Mar. 2013, p. 22.

[10] A. Oxley, "Chapter 2 - Introduction to GPS," in *Uncertainties in GPS Positioning*, A. Oxley, Ed., Elsevier Academic Press, 2017, pp. 19–38. DOI: 10.1016/B978-0-12-809594-2.00002-2.

[11] ——, "Chapter 3 - Basic GPS Principles," in *Uncertainties in GPS Positioning*, A. Oxley, Ed., Elsevier Academic Press, 2017, pp. 39–60. DOI: 10.1016/B978-0-12-809594-2.00003-4.

[12] ——, "Chapter 11 - Improving Accuracy With GPS Augmentation," in *Uncertainties in GPS Positioning*, A. Oxley, Ed., Elsevier Academic Press, 2017, pp. 129–134. DOI: 10.1016/B978-0-12-809594-2.00011-3.

[13] M. Dozza, M. Idegren, T. Andersson, and A. Fernandez, "Platform enabling intelligent safety applications for vulnerable road users," in *IET Intelligent Transport Systems*, IET, Jun. 2014, pp. 368–376. DOI: 10.1049/iet-its.2012.0201.

[14] T. H. Witte and A. M. Wilson, "Accuracy of WAAS-enabled GPS for the determination of position and speed over ground," in *Journal of Biomechanics, 38*, Elsevier, Aug. 2005, pp. 1717–1722. DOI: 10.1016/j.jbiomech.2004.07.028.

[15] P. Arnesen, O. K. Malmin, and E. Dahl, "A forward Markov model for predicting bicycle speed," in *Springer Nature 2019 Transportation (Transportation)*, Springer, Jun. 2019, pp. 1–23. DOI: 10.1007/s11116-019-10021-x.

[16] K. C. Heesch and M. Langdon, "The usefulness of GPS bicycle tracking data for evaluating the impact of infrastructure change on cycling behaviour," in *Health Promotion Journal of Australia, 3*, 3, vol. 27, Brisbane, Australia: Australian Health Promotion Assosiation, Sep. 2016, pp. 222–229. DOI: 10.1071/HE16032.

[17] M. Bíl, R. Andrášik, and J. Kubeček, "How comfortable are your cycling tracks? A new method for objective bicycle vibration measurement," in *Elsevier Transportation Research Part C: Emerging Technologies*, vol. 56, Elsevier, Jul. 2015, pp. 415–425. DOI: 10.1016/j.trc.2015.05.007.

[18] T. H. Witte and A. M. Wilson, "Accuracy of non-differential GPS for the determination of speed over ground," in *Journal of Biomechanics, 37*, Elsevier, Feb. 2004, pp. 1891–1898. DOI: 10.1016/j.jbiomech.2004.02.031.

[19] S. Miah, I. Kaparias, and P. Liatsis, "Evaluation of MEMS sensors accuracy for bicycle tracking and positioning," in *International Conference on Systems, Signals and Image Processing (IWSSIP)*, vol. 22, IEEE, 2015, pp. 299–303. DOI: 10.1109/IWSSIP.2015.7314235.

[20] A. Koellner, C. J. Cameron, and M. A. Battley, "Measurement and analysis system for bicycle field test studies," in *Elsevier Procedia Engineering*, vol. 72, Elsevier, Dec. 2014, pp. 350–355. DOI: 10.1016/j.proeng.2014.06.061.

[21] W. M. Wehrbein, "Collecting and Recording Bicycle Speed Data by CBL," in *The Physics Teacher*, vol. 41, American Association of Physics Teachers with American Institute of Physics, 2003, pp. 243–244. DOI: 10.1119/1.1564508.

[22] Y. Ben-Abu, I. Wolfson, and H. Yizhaq, "Finding the speed of a bicycle in circular motion by measuring the lean angle of the bicycle," in *Physics Education (Phys Educ)*, 3, vol. 53, IOP Publishing, 2018. DOI: 10.1088/1361-6552/aaa0f3.

[23] S. Bernardi and F. Rupi, "An Analysis of Bicycle Travel Speed and Disturbances on Off-street and On-street Facilities," in *Elsevier Transportation Research Procedia, 5*, vol. 5, Elsevier, Dec. 2015, pp. 82–94. DOI: 10.1016/j.trpro.2015.01.004.

[24] C. Valerius, J. Krupar, and W. Schwarz, "Electronic Power Management for Bicycles," in *European Conference on Power Electronics and Applications*, vol. 11, IEEE, 2005, pp. 1–10. DOI: 10.1109/EPE.2005.219481.

[25] N. Kovácsová, J. de Winter, A. L. Schwab, M. Christoph, D. Twisk, and M. P. Hagenzieker, "Riding performance on a conventional bicycle and a pedelec in low speed exercises: Objective and subjective evaluation of middle-aged and older persons," in *Elsevier Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 42, Elsevier, Oct. 2016, pp. 28–43. DOI: 10.1016/j.trf.2016.06.018.

[26] R. A. Wilson-Jones, "Steering and stability of single-track vehicles," in *Institution of Mechanical Engineers Automobile Division 1947-1970*, Sage journals, Jun. 1951, pp. 191–213. DOI: 10.1243/pime_auto_1951_000_023_02.

[27] H. Fu, "Fundamental Characteristics of Single-Track Vehicles in Steady Turning," in *Bulletin of JSME*, 34, vol. 9, The Japan Society of Mechanical Engineers, 1966, pp. 284–293. DOI: 10.1299/jsme1958.9.284.

[28] D. Dardari, N. Decarli, A. Guerra, A. Al-Rimawi, V. Marín Puchades, G. Prati, M. De Angelis, F. Fraboni, and L. Pietrantoni, "High-accuracy tracking using ultrawideband signals for enhanced safety of cyclists," in *Hindawi Mobile Information Systems*, vol. 2, Hindawi, 2017, pp. 1–13. DOI: 10.1155/2017/8149348.

[29] S. Joo and C. Oh, "A novel method to monitor bicycling environments," in *Elsevier Transportation Research Part A: Policy and Practice*, Elsevier, Aug. 2013, pp. 1–13. DOI: 10.1016/j.tra.2013.07.001.

[30] C.-Y. Wang, C.-N. Street, C.-S. Cheng, and T. Hsien, "Brake Signal Sensor Device," US Patent Office, Patent US 6,320,499, Oct. 2001.

[31] S.-F. Tsai, "Brake Operating Device of Electric Bicycle With Hall Effect Sensor," US Patent Office, Application Publication US 2013/02284.05, Sep. 2013.

[32] M. Dozza, G. F. Bianchi Piccinini, and J. Werneke, "Using naturalistic data to assess e-cyclist behavior," in *Elsevier Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 55, Elsevier, May 2015. DOI: 10.1016/j.trf.2015.04.003.

[33] P. Huertas Leyva, M. Dozza, and N. Baldanzini, "Investigating cycling kinematics and braking maneuvers in the real world: e-bikes make cyclists move faster, brake harder, and experience new conflicts," in *Elsevier Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 54, Elsevier, Apr. 2018, pp. 211–222. DOI: 10.1016/j.trf.2018.02.008.

[34] K. Zang, J. Shen, H. Huang, M. Wan, and J. Shi, "Road surface roughness from GPS and accelerometer in smartphone," in *MDPI Sensors*, 1, vol. 18, Multidisciplinary Digital Publishing Institute, Jan. 2018. DOI: 10.3390/s18030914.

[35] Y. Usami, K. Ishikawa, T. Takayama, M. Yanagisawa, and N. Togawa, "Bicycle Behavior Recognition using Sensors Equipped with Smartphone," in *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, IEEE, Sep. 2018. DOI: 10.1109/ICCE-Berlin.2018.8576254.

[36] W. Gu, Y. Liu, Y. Zhou, Z. Zhou, C. J. Spanos, and L. Zhang, "BikeSafe: Bicycle Behavior Monitoring via Smartphones," in *Proceedings of the 2017 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers*, Association for Computing Machinery, Sep. 2017, pp. 45–48. DOI: 10.1145/3123024.3123158.

[37] J. G. Hudson, J. C. Duthie, Y. K. Rathod, K. A. Larsen, J. L. Meyer, et al., "Using smartphones to collect bicycle travel data in Texas," Texas Transportation Institute. University Transportation Center for Mobility, Tech. Rep., Aug. 2012.

[38] B. Charlton, E. Sall, M. Schwartz, and J. Hood, "Bicycle route choice data collection using GPS-enabled smartphones," in *Transportation Research Board 90th Annual Meeting*, San Francisco, CA, Jan. 2011.

# Acknowledgements

Thanks to everyone who read my thesis. Seriously! Thanks for all the feedback, positive and negative, on my work in general, on my plots, the colors in plots, the photos, tables, Ti*k*Z sketches, LaTeX in general, missing commas, erroneously set commas, wording, sentence structure, as well as sentence overlengths... Thanks to my friends and flatmates for cheering me up during difficult times of development and writing. Thanks also, to Snickers® for saving me from starvation, or, in less extreme cases, for giving me energy to make my way home after an average working day, e.g., Sunday, of up to 19 hours. Thanks to the FÜ for being open almost always, and thanks to the gate keeper for knowing my name.

As it is hard to decide whom to thank most, first, last, or not at all, I decided to approach this problem differently. For each person, I took all letters contained in his*her name. I put these letters in a statistically ordered list based on the number of occurrences of each letter. Hopefully, I didn't forget anyone :) But if so, maybe you find the letters of your name anyways:
`eilnaOrAkovFdSgumHMchtfJsb`

And finally:
Special thanks to you, for taking the time to read my thesis :)

Ich bremste die bremsende Bremse und die bremsende Bremse bremste mich.
Fährst du noch oder ***schreibst du schon?***