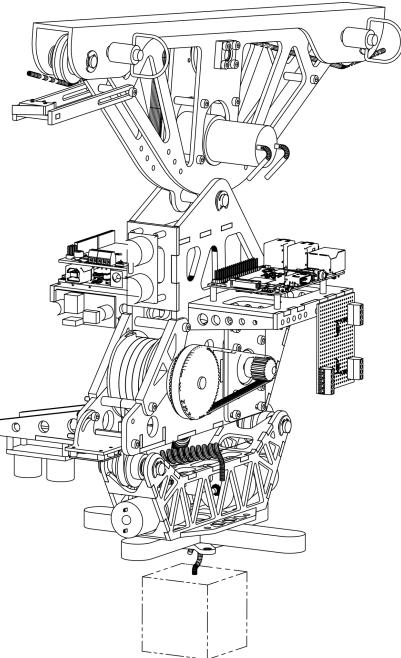


Silisloth

Dokumentation PREN 2

PREN Gruppe 7

2. Juli 2018



Versionierung

Version	Datum	Bemerkung
0.1	Fr, 23.03.2018	Aufsetzen des Dokuments
0.2	Fr, 25.05.2018	Fertigstellung der Vorabversion
1.0	Fr, 10.06.2018	Fertigstellung für Schlussabgabe

Betreuer

Zeno STÖSSEL, Elektrotechnik

Experten

Jörg HOFSTETTER, Informatik

Carsten HAAK, Maschinentechnik

Autoren

Sandro BERTOZZI, Informatik

Christoph BINKERT, Maschinentechnik

Patrick BUCHER, Informatik

Alex DUONG, Elektrotechnik

Quentin FREI, Maschinentechnik

Jan GREBER, Elektrotechnik

Marko LOVRINOVIC, Maschinentechnik

Johannes TOGAN, Maschinentechnik

Teamvorstellung Gruppe 7

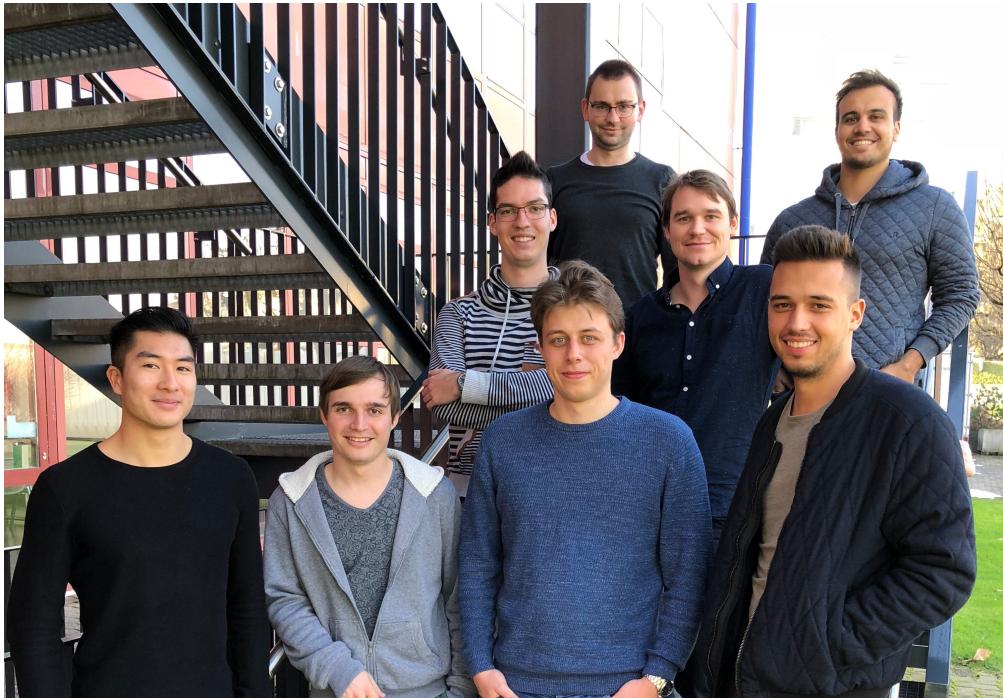


Abbildung 1: Gruppenfoto (von links nach rechts): Alex Duong, Christoph Binkert, Sandro Bertozzi, Quentin Frei (vorne), Patrick Bucher (hinten), Jan Greber, Marko Lovrinovic (vorne), Johannes Togan (hinten)

- Alex DUONG (Elektrotechnik)
alex.duong@stud.hslu.ch
- Christoph BINKERT (Maschinentechnik)
christoph.binkert@stud.hslu.ch
- Sandro BERTOZZI (Informatik)
sandro.bertozzi@stud.hslu.ch
- Quentin FREI (Maschinentechnik)
quentin.frei@stud.hslu.ch
- Patrick BUCHER (Informatik)
patrick.bucher@stud.hslu.ch
- Jan GREBER (Elektrotechnik)
jan.greber@stud.hslu.ch
- Marko LOVRINOVIC (Maschinentechnik)
marko.lovrinovic@stud.hslu.ch
- Johannes TOGAN (Maschinentechnik)
johannes.togan@stud.hslu.ch

Management Summary

Dieses Dokument wurde im Rahmen des Moduls PREN 2 im Frühlingssemester 2018 von der Gruppe 7 erstellt und dokumentiert die Umsetzung der autonomen Laufkatze *Silisloth*, wie sie im Herbstsemester 2017 im Rahmen des Moduls PREN 1 konzipiert worden ist. *Silisloth* ist ein Gerät, das an einem Seil hängend nach erteiltem Startsignal eine Last lokalisieren, aufnehmen, über Hindernisse hinweg befördern und in einem durch konzentrische Quadrate markierten Zielbereich absetzen kann. Die hier vorliegende Dokumentation beschreibt die Umsetzung der Bereiche Mechanik (Aufbau, Montage), Elektronik (Schaltungen, Verdrahtung, Motorensteuerung), Informatik (Bildverarbeitung, Steuerung, Smartphone-App) wie auch interdisziplinäre Fragestellungen (Kommunikation zwischen den Geräten, vorgenommene Tests) sowie den geplanten Ablauf beim Absolvieren der gestellten Wettbewerbsaufgabe. Wie schon das zugrundeliegende Konzept (Bertozzi et al., 2017), auf das in diesem Dokument bei Bedarf verwiesen wird, folgt die Umsetzung der Maxime «eine sichere, zuverlässige und für alle verständliche Lösung zu erarbeiten, die dennoch originell und für alle Beteiligten lehrreich ist und zur Zusammenarbeit über die Fachgrenzen hinweg anregt», wie sie in PREN 1 als Gruppenziel formuliert worden ist.

Inhaltsverzeichnis

1. Einleitung	6
1.1. Verwendete Werkzeuge	7
2. Konstruktion	8
2.1. Kontrollbox	10
2.2. Greifmechanismus	10
2.3. Weitere Aspekte der Konstruktion	II
3. Elektronik	13
3.1. Steuerstromkreis	13
3.2. Laststromkreis	13
3.3. Elektronikkomponenten	14
4. Ablauf	17
4.1. Berechnung der Enddistanz	19
5. Kommunikation	20
5.1. Laptop – Raspi	20
5.2. Raspi – Arduino	21
5.3. Smartphone-App – Raspi	22
6. Bildverarbeitung	24
6.1. Lösungsansatz	24
6.2. Umsetzung	24
6.2.1. Die Polygonhierarchie	26
6.2.2. Abwägungen	26
6.3. Auflösung und Verarbeitungsgeschwindigkeit	27
7. Software	30
7.1. Smartphone-App: Java	30
7.2. Raspi: Python	31
7.2.1. Projektstruktur	31
7.2.2. Eingesetzte Python-Bibliotheken	33
7.3. Arduino: C	33
8. Zielerreichung	36
8.1. Testphase	36

Inhaltsverzeichnis

8.2. Beurteilung gemäss Anforderungen	37
9. Rückblick	40
9.1. Lessons Learned	41
9.2. Fazit	42
9.2.1. Gruppenziele	42
A. Anhang	44
A.1. Kostenübersicht	44
A.2. Thresholding-Benchmarks	44
A.3. Weitere	47
Quellen	48
Abbildungsverzeichnis	49
Tabellenverzeichnis	50

1. Einleitung

In PREN 1 wurde das Konzept für *Silisloth* – eine autonome Laufkatze – entwickelt. In PREN 2 wurde dieses umgesetzt. Die verwendeten und entwickelten Komponenten und die Konstruktion folgen dabei grösstenteils diesem Konzept. Das vorliegende Dokument gibt einen Überblick über den dabei erstellten Prototypen und handelt von dessen Umsetzung. Änderungen gegenüber dem ursprünglichen Konzept werden dabei gesondert hervorgehoben. Um Wiederholungen und das Kopieren von Passagen zu vermeiden, wird bei Bedarf auf das Konzept (Bertozzi et al., 2017) verwiesen.

Die Konstruktion bildet die Grundlage von *Silisloth*. Die schlanke Lösung, die den Prinzipien der Leichtbauweise folgt, wird in Abschnitt 2: *Konstruktion, Seite 8* beschrieben. Eine Explosionsansicht des CAD-Modells bietet einen Überblick über die einzelnen Komponenten und deren Anordnung.

In Abschnitt 3: *Elektronik, Seite 13* geht es einerseits um verschiedene Elektrotechnikkomponenten, die im Konzept von PREN 1 noch nicht erwähnt wurden, andererseits um die Verdrahtung der elektronischen Komponenten miteinander. Last- und Steuerstromkreis werden in einem Blockschema dargestellt.

In Abschnitt 4: *Ablauf, Seite 17* wird das Absolvieren der Wettbewerbsaufgabe aus einer technischen Perspektive erläutert. Der Leser erhält dadurch einen Einblick in die Funktionsweise von *Silisloth*, und erfährt, welche Überlegungen gemacht wurden, um das gestellte Problem erfolgreich zu lösen. Für Implementierungsdetails wird auf später folgende Abschnitte verwiesen.

Silisloth ist mehr als ein elektromechanisches Gerät. Es ist mit einem Raspberry Pi (fortan «Raspi» genannt) und einem Arduino ausgestattet, was schon genügt um es als fahrendes Rechennetzwerk zu bezeichnen. Das Startsignal wird von einer Smartphone-App aus erteilt, welche auch die Lastkoordinaten anzeigt. Ein externer Laptop dient dazu den Ablauf des Raspi-Programms zu überwachen – und um *Silisloth* in Bereitschaft zu versetzen. Die verschiedenen Kommunikationsprotokolle, die im Zusammenspiel dieser vier Computer zum Einsatz kommen, werden in Abschnitt 5: *Kommunikation, Seite 20* beschrieben. Zur Veranschaulichung wird ein UML-Sequenzdiagramm verwendet.

Neben der Kommunikation stellt das Erkennen des Zielfeldes und die Ermittlung des Abstandes zu diesem eine grosse Herausforderungen dar. Der Abschnitt 6: *Bildverarbeitung, Seite 24* geht ausführlich auf die erarbeitete Lösung ein. Dabei wird zwar auch die dabei verwendete OpenCV-Library und die Programmiersprache Python angesprochen, die Erläuterungen sollen aber dennoch für einen Leser verständlich sein, der in diesen Technologien nicht bewandert ist. Eine Bildserie des Zielfeldes vermittelt dem Leser die Perspektive der Kamera, was die erstellte Lösung besser nachvollziehbar macht.

Silisloth erfordert zum Betrieb verschiedene Softwarekomponenten. Die Smartphone-App zur Erteilung des Startsignals wird in Abschnitt 7.1: *Smartphone-App: Java, Seite 30* erläutert. Die ganze Programmlogik, die *Silisloth* zu einem autonomen Gerät machen, ist auf einen Raspi und einen Arduino verteilt. Abschnitt 7.2: *Raspi: Python, Seite 31* geht auf die Python-Lösung ein, welche einerseits die Aktionen von *Silisloth* koordiniert und andererseits verschiedene Hardware- und Kommunikati-

1. Einleitung

onsschnittstellen bedient. Ein UML-Klassendiagramm verschafft Übersicht über das erstellte Python-Projekt. Der Abschnitt 7.3: *Arduino: C, Seite 33* handelt von der in C und mit der RTOS-Library geschriebenen Softwarelösung, welche in erster Linie für die Motorensteuerung zuständig ist, jedoch auch Kommunikations- und Koordinationsprobleme löst. Besonders wichtig sind hier interne Stati, welche mit einer State-Machine veranschaulicht werden.

Der in PREN 2 erarbeitete Prototyp soll, wie schon das in PREN 1 erstelle Konzept, den gestellten Anforderungen genügen. Inwiefern diese Anforderungen erfüllt sind, und wie der Prototyp mittels Tests kontinuierlich verbessert wurde, ist Thema von Abschnitt 8: *Zielerreichung, Seite 36*.

Im Abschnitt 9: *Rückblick, Seite 40* wird die Umsetzung des *Silislotb*-Projektes reflektiert. Dabei wird einerseits auf die einzelnen Projektphasen in der Umsetzung, andererseits auf bestimmte dabei gemachte Erfahrungen eingegangen. Am Schluss wird ein Fazit gezogen, welches – soviel vorweg – aufgrund der gelungenen Lösung positiv ausfällt.

1.1. Verwendete Werkzeuge

Die Arbeit wurde mit \LaTeX^1 und Xe\LaTeX^2 in den Schriftarten EB Garamond, Open Sans und Fira Mono gesetzt. Die Grafiken wurden mit *Graphviz*³, *plantuml*⁴ und *LibreOffice Draw*⁵ erstellt. Weitere Hilfsmittel waren *git*⁶ zur Versionsverwaltung der Quelldateien und *make*⁷ zum Erstellen der Grafiken und des Dokuments aus den Quelldateien.

¹<https://www.latex-project.org/>

²<https://www.sharelatex.com/learn/XeLaTeX>

³<https://www.graphviz.org/>

⁴<http://plantuml.com/>

⁵<https://www.libreoffice.org/discover/draw/>

⁶<https://git-scm.com/>

⁷<https://www.gnu.org/software/make/>

2. Konstruktion

Die Konstruktion von *Silisloth* (siehe *Abbildung 2, Seite 9*) folgt weitgehend den CAD-Plänen von PREN 1 (Bertozzi et al., 2017, S. 12-13). Einige Teile sind dazugekommen (Endschalter, PCB), manche Teile wurden anders positioniert (Ultraschallsensor in x-Richtung, Luftpumpen und Magnetventil) und in manchen Bereichen wurde die Konstruktion verfeinert und optimiert (Greifeinheit, Kontrollbox). Die Konstruktion lässt sich in drei Bereiche unterteilen und besteht aus den folgenden Komponenten:

- | | |
|---|--|
| A: AUFHÄNGUNG | 13. Raspberry Pi |
| 1. Halterungsschiene für Aufhängung | 14. Laufrolle für Hubmechanismus |
| 2. Laufrollen mit Bolzen | 15. PCB für Ultraschallsensoren |
| 3. Antriebsrad | 16. Ultraschallsensor in x-Richtung |
| 4. Quick-Pins zur Befestigung | 17. Ultraschallsensor in z-Richtung |
| 5. Riemengetriebe für den Antrieb (Übersetzung) | 18. Raspi-Cam (neigbar) |
| 6. Getriebemotor für den Antrieb | 19. Seiltrommel |
| 7. Endschalter | 20. Schrittmotor mit Kühlkörper |
| 8. Bolzen zur Befestigung der Kontrollbox | 21. Riemengetriebe für den Hubmechanismus (Untersetzung) |
| B: KONTROLLBOX | 22. Laufrollen für Seilwinde |
| 9. LiPo-Akku (Laststromkreis) | C: GREIFMECHANISMUS |
| 10. Motor Shield | 23. Luftpumpen |
| 11. Arduino | 24. Spiralkabel |
| 12. LiPo-Akku (Steuerstromkreis) mit USV
(verborgen) | 25. Magnetventil |
| | 26. Silikongreifer |

In den folgenden Abschnitten wird erläutert, auf welche Eigenschaften bei der Umsetzung besonderer Wert gelegt wurde, und wie dabei die Designentscheide gefällt wurden.

2. Konstruktion

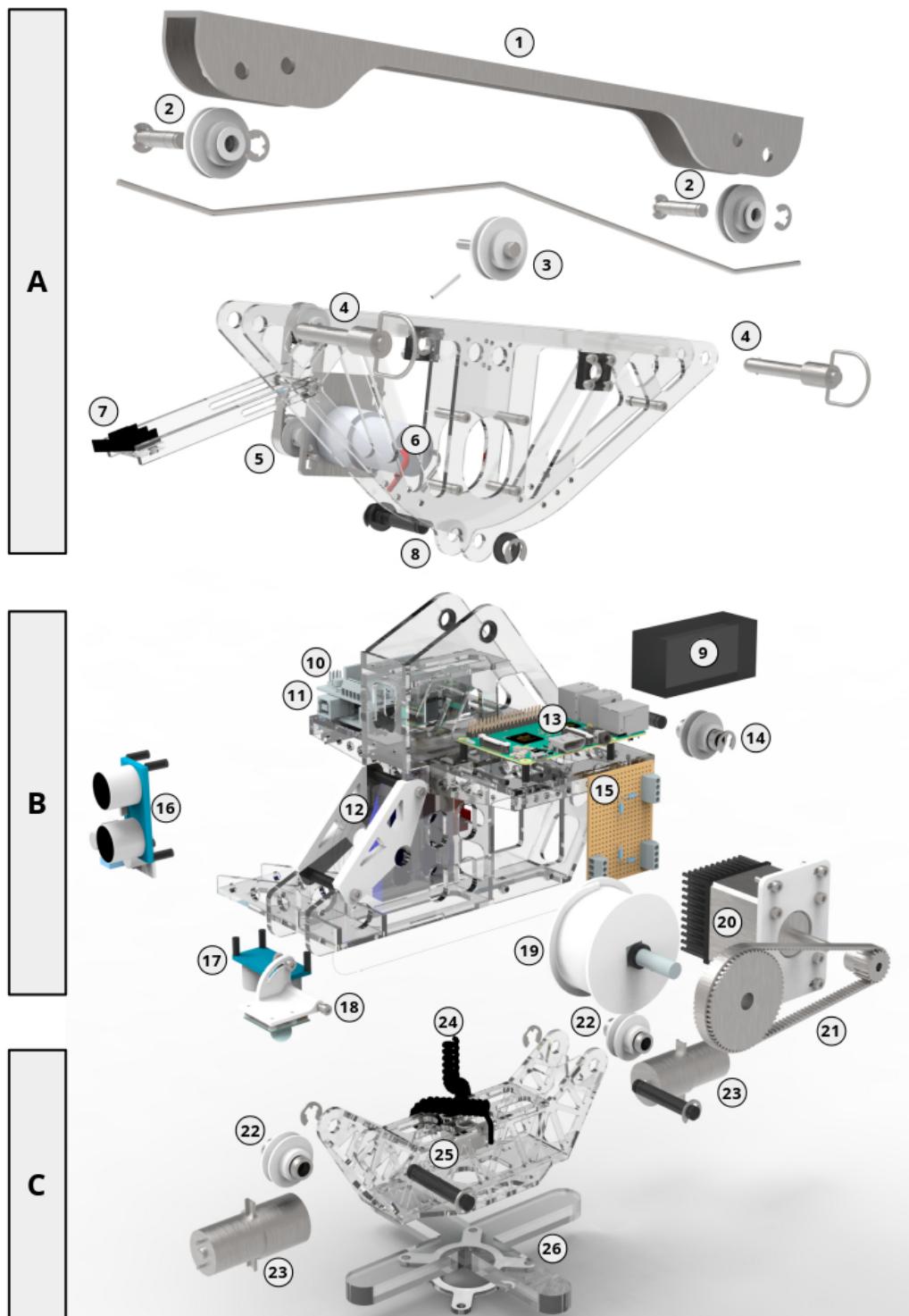


Abbildung 2: Die Explosionsansicht des CAD-Modells von *Silisloth* mit nummerierten Komponenten

2.1. Kontrollbox

Während der konstruktiven Phase wurde besonderen Wert auf die Designeigenschaften Zugänglichkeit und Kompaktheit gelegt. Es sollte gewährleistet sein jederzeit auf alle Komponenten zugreifen zu können, damit an diesen, wenn nötig, Eingriffe vorgenommen werden können, ohne dass eine Demontage nötig oder Betriebsunterbrüche daraus resultieren würden. Es wurde speziell darauf geachtet Stapelung von Komponenten zu vermeiden.

In Bezug auf die Kompaktheit wurde besonders auf platzsparendes Design, einfache Transportierbarkeit und ein geringes Gewicht geachtet.

Beim Seilwindenmotor war während der Designphase nicht sicher, ob dieser den Anforderungen genügt. Deshalb wurde hier eine Adapterplatte eingesetzt, sodass nur diese Platte und nicht eine ganze Komponente ausgetauscht werden müsste, sollte ein anderer Seilwindenmotor zum Einsatz kommen. Zusätzlich wurde die Adapterplatte mit Langlöchern versehen, womit sich der Achsenabstand leicht variieren lässt. Dies dient einerseits dem Toleranzausgleich und andererseits zum Spannen des Riemens.

Grosser Wert wurde bei der Konstruktion darauf gelegt, dass der Schwerpunkt an der richtigen Stelle zu liegen kommt, sodass der Verbindungsbolzen und der Schwerpunkt der Kontrollbox zusammen mit dem Greifmechanismus im Lot sind. Dies setzte voraus, dass das Einzelgewicht aller Komponenten bekannt bzw. die Dichte aller Materialien hinterlegt war. Deshalb mussten zunächst alle Kaufteile beschafft werden, bevor die Kontrollbox gelasert werden konnte.

Für die finale Justierung wurde die Konstruktion so gewählt, dass der Akku in x-Richtung leicht variabel verbaubar ist, was einen gewissen Korrekturspielraum zulässt.

Dieser Ansatz wurde gewählt, damit für die Sensorikelemente (Ultraschallsensoren) mit einer geringen Toleranzbreite gearbeitet werden konnte. So konnte zusätzlich die Anzahl möglicher Fehl- oder Störquellen bei den Sensoren minimiert werden. Bei der Kamera war der optimale Blickwinkel noch nicht bekannt. Deshalb fiel die Wahl bewusst auf eine einstellbare Halterung, die sich um 75° an der y-Achse rotieren lässt.

2.2. Greifmechanismus

Der Greifmechanismus ist symmetrisch um die x-z-Ebene und die y-z-Ebene aufgebaut. Damit liegt der Schwerpunkt in seinem Zentrum, wodurch gewährleistet ist, dass der Mechanismus jederzeit horizontal, und der Silikongreifer optimal ausgerichtet ist.

Bei der Konstruktion des Greifmechanismus wurde auf einen kompakten Aufbau geachtet, damit die Perspektive der Kamera möglichst wenig eingeschränkt wird. Auch diese Konstruktion folgt der Leichtbauweise, sodass der Seilwindenmotor mit dem Hubmechanismus nicht am Limit betrieben werden muss.

2. Konstruktion

Um die Stabilität des Greifers zu gewährleisten, wurde eine Halterung konstruiert. Zwei Platten halten den Greifer zusammen. Greifer, Platten, Luftpumpen, Magnetventil, Umlenkrollen und unteres Gerüst bilden den Greifmechanismus.

Der Silikongreifer wird mit zwei 12V-Luftpumpen aufgeblasen. Bei den ersten Versuchen wurde festgestellt, dass eine Pumpe alleine den Greifer nicht genug schnell mit Luft füllt. Bei zwei Pumpen wurde dann die gewünschte Griff-Geschwindigkeit erreicht.

Damit der Luftstrom gesteuert werden kann, ist ein Magnetventil zwischen Luftpumpen und Silikongreifer geschaltet. Somit kann die Luft zu einem gewünschten Zeitpunkt aus dem Greifer weggeführt werden.

Um die Dichtheit zu testen, wurde der Greifer mit Luft gefüllt und ein Würfel eingeklemmt. Der Greifer hielt den Würfel über vier Minuten in der Luft, somit erfüllte er die Anforderungen. Die Herstellung des Silikongreifers wurde bereits in der Dokumentation von PREN 1 (Bertozzi et al., 2017, S. 44-46) beschrieben.

2.3. Weitere Aspekte der Konstruktion

Fortbewegung Das treibende, sich in der Mitte der Aufhängung befindende Kunststoffrad wird über einen Zahnriemen von einem Getriebemotor angetrieben. Das Antriebsrad befindet sich unterhalb des Stahlseils, damit mehr Normalkraft für die Kraftübertragung (Reibung) gewonnen werden kann. Zwei zusätzliche, nicht angetriebene Kunststoffräder, welche sich oberhalb des Seils befinden, stützen die Laufkatze. Die Räder sind auf Stahlwellen befestigt, welche wiederum in Kugellagern laufen, um einen möglichst geringen Reibungsverlust zu erzielen.

Berechnung für die Wellenabstände Die Berechnung für die Wellenabstände, welche für die Aufhängung und Kontrollbox verwendet wird, erfolgt über die *Formel 1, Seite 12* (Wittel, Jannasch & Voßiek, 2016, Kapitel 16, Formel 22).

Polyvalenz In der Testphase hat sich gezeigt, dass das Konzept mit dem Silikongreifer multiple Einsatzmöglichkeiten bietet. Diese polyvalenten Greifer, die einer menschlichen Hand ähnlich sind, können für verschiedenste Objekte mit unterschiedlichen Geometrien und aus verschiedenen Materialien eingesetzt werden. Diese Polyvalenz zeigt sich auch beim Flaschenzug, da sich der Einsatz im Längsbereich fast beliebig erweitern lässt.

Gewicht Mit der Leichtbauweise wurde das Ziel der Kompaktheit erreicht – unter anderem mit Erleichterungslöchern und Fachwerkstrukturen.

Mehrfachverwendung Der Herstellungs- und Designaufwand wurde durch gezielte Mehrfachverwendung von Komponenten reduziert. Zum Beispiel wurde für die drei Seilumlenkrollen

2. Konstruktion

des Hubmechanismus dreimal das gleiche Konzept mit den gleichen Bauteilen verwendet, d.h. identische Rollen, Lagerungs- und Sicherungselemente.

Symmetrie Bei der Konstruktion der tragenden Strukturelemente wurde auf Symmetrie geachtet, sodass Teile gespiegelt oder mehrfach verwendet werden konnten. Dadurch liess sich der Konstruktionsaufwand optimieren. Symmetrie hat auch einen positiven Effekt auf den Schwerpunkt.

Verwendete Materialien Die verwendeten Materialien sind Acrylglass der Stärken drei und vier Millimeter. Die etwas komplexeren Teile wurden aus Acrylnitril-Butadien-Styrol (ABS) 3D gedruckt. Für die Wellen und Achsen fiel die Wahl auf S235-Stahl.

Montage Im CAD-Prozess wurde bewusst ein sehr hoher Detaillierungsgrad der einzelnen Teile angestrebt, sodass bei der Montage der Kaufteile keine Nachbearbeitung nötig war. Dies geschah auch zum Schutz der Komponenten, die durch solche Eingriffe beeinträchtigt worden wären.

$$e \approx \frac{L_d}{4} - \frac{\pi}{8} \cdot (d_{dg} + d_{dk}) + \sqrt{\left[\frac{L_d}{4} - \frac{\pi}{8} \cdot (d_{dg} + d_{dk}) \right]^2 - \frac{(d_{dg} - d_{dk})^2}{8}} \quad (1)$$

e = Wellenabstand

L_d = Zahnriemenlänge (Zähnezahl Riemen multipliziert mit dessen Teilung)

d_{dg}, d_{dk} = Scheibendurchmesser kleines bzw. grosses Rad

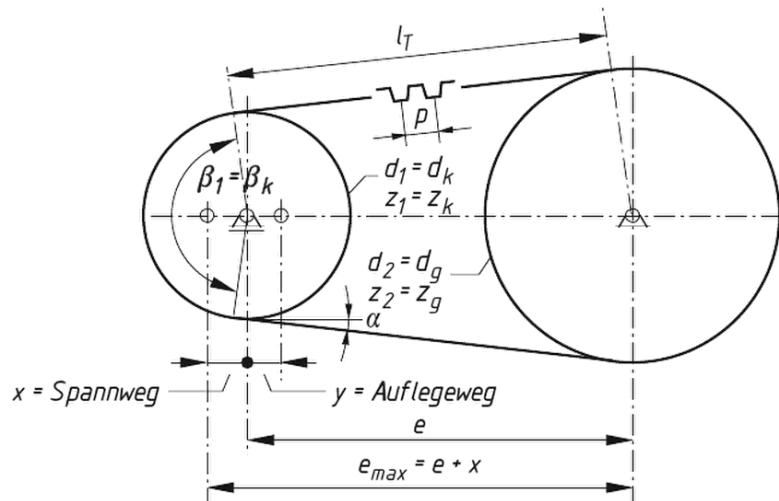


Abbildung 3: Die Berechnung des Wellenabstands e (Wittel et al., 2016, S. 206)

3. Elektronik

Silisloth ist aufgeteilt in einen Last- und Steuerstromkreis. Jeder der Stromkreise wird von einem eigenen Akku gespeist. Damit wird sichergestellt, dass beim Anlaufen von Komponenten mit hohem Stromverbrauch (DC-Motor, Schrittmotor) die Spannung am Raspi oder Arduino nicht zusammenbricht, was einen Neustart zur Folge hätte. Jedoch ist es wichtig die beiden Akkus auf dieselbe gemeinsame Masse (Potential null) zu bringen um ungewollte Ausgleichsströme zu verhindern. Realisiert wird dies mit einer einfachen $0.25mm^2$ Kupferverbindung zwischen den beiden Minuspolen der Akkus. Abbildung 5, Seite 15 zeigt das Blockschaltbild mit den beiden Stromkreisen.

3.1. Steuerstromkreis

Der Steuerstromkreis beinhaltet die Intelligenz von *Silisloth*. Spannungsquelle ist ein Lithium-Polymer-Akku (3.7V/3000mAh). Die 3.7V werden über einen Step-Up-Converter von Adafruit auf 5.2V gebracht, welcher auch das Laden des Akkus über das Netzgerät ermöglicht. Über den Converter wird der Raspi mit Strom versorgt, welcher dann über eine USB-Schnittstelle gleichzeitig den Arduino speist. Der Raspi ist sozusagen das Gehirn von *Silisloth*: er empfängt Signale von aussen, verarbeitet diese und leitet dann die nötigen Befehle oder Informationen an den Arduino oder an die App weiter. Direkt am Raspi angeschlossen über ein Flachbandkabel ist die Raspi-Kamera, welche für die Zielfelderkenntung verwendet wird. Des Weiteren führt ein einfacher Taster auch direkt auf die Eingangspins des Raspis. Dieser ist zuvorderst an der Laufkatze angebracht und erzeugt ein Signal sobald der Endpfosten berührt wird, was zum sofortigen Stoppen des Antriebsmotors führt. Über ein selbst hergestelltes PCB (Printed Circuit Board, Abbildung 4, Seite 14) werden die beiden Ultraschallsensoren vom Typ HC-SR04 am Raspi angeschlossen. Diese messen zu jeder Zeit den Abstand vom Boden und zum Endpfosten. Das PCB beinhaltet lediglich vier Widerstände, welche benötigt werden um die Sensoren korrekt anzusteuern. Es erleichtert aber auch das Anschließen der Sensoren und spart Pins am Raspi. Wie bereits erwähnt wurde das Board in einfachster Art und Weise selbst hergestellt. Auf einer vorgefertigten Loch-Platte wurden die Widerstände und Anschlussklemmen aufgelötet und mit isolierten Trafo-Drähten verbunden.

3.2. Laststromkreis

Ein Lithium-Polymer Akku (14.8V/1300mAh) bildet die Grundlage für den Laststromkreis. Dieser besteht aus vier Zellen. Die Spannung der einzelnen Zellen darf nicht unter ca. 3.3V fallen, ansonsten kann der Akku nicht mehr geladen werden und wird somit unbrauchbar. Um dies zu verhindern kommt ein Spannungsüberwacher zum Einsatz, welcher am Ladeanschluss des Akkus eingesteckt wird und permanent die Zellspannungen überprüft. Gerät die Spannung einer Zelle in den kritischen Bereich, wechselt die jeweilige LED von grün auf rot. Die 14.8V vom Akku sind noch ein wenig zu hoch

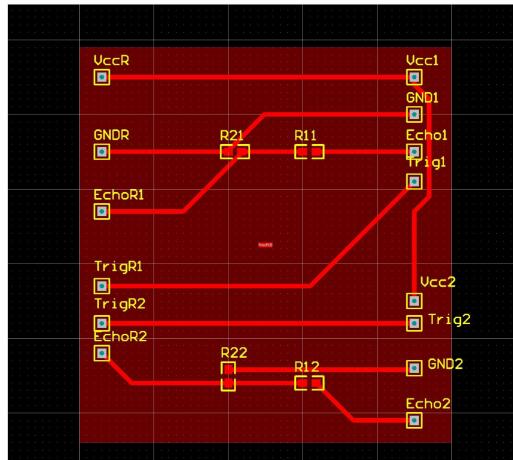


Abbildung 4: Printed Circuit Board für die Ultraschallsensoren, die am Raspi angeschlossen sind

für die übrigen Komponenten und werden deshalb über einen Step-Down-Converter auf exakt 12V gebracht. Mit diesen 12V wird das Adafruit Motor Shield v2 gespeist, welches einfach auf der Arduino aufgesteckt wird. Direkt am Motor Shield angeschlossen ist der 12V DC Getriebemotor Igarashi TYP 33G-50, welcher die Laufkatze antreibt, und der 12V bipolar Schrittmotor von SparkFun Electronics, welcher die Greifeinheit rauf und runter lässt. In der Greifeinheit befinden sich zwei 12V-Luftpumpen und das Magnetventil. Diese Komponenten werden über ein Spiralkabel mit Strom versorgt. Damit der Silikongreifer aufgepumpt werden kann, muss der Luftrauslass des Magnetventils geschlossen sein. Dies geschieht, sobald eine Spannung von 12V am Ventil anliegt. Wird die Spannung unterbrochen, öffnet sich der Luftrauslass und der Greifer löst sich. Um eine hohe Spannung von 12V mit dem Arduino zu schalten, kommt ein 5V-Relais zum Einsatz. Dieses wird als Öffner angeschlossen, d.h. im Ruhezustand ist der Stromkreislauf zum Magnetventil geschlossen, was bedeutet, dass dieses dicht macht. Wird nun vom Arduino über den 5V-Ausgang das Relais angesteuert, zieht dieses und unterbricht somit den Stromkreis zum Magnetventil, was bedeutet, dass der Greifer die Last loslässt. Die beiden Luftpumpen werden direkt am Motor Shield angeschlossen.

3.3. Elektronikkomponenten

Im Folgenden werden sämtliche verbauten Elektronikkomponenten aufgelistet. Die Nummerierung der Liste stimmt mit derjenigen auf dem Blockschaltbild (*Abbildung 5, Seite 15*) überein.

Gegenüber PREN 1 sind einige neue Komponenten dazugekommen. Da für diese in der Dokumentation zu PREN 1 (Bertozzi et al., 2017, S. 50) eine Quellenangabe fehlt, ist diese bei den betreffenden Komponenten – Arduino, Motor Shield, LiPo-Akku für Laststromkreis, Spannungsüberwacher, Step-Down-Converter, Relaismodul und Magnetventil – hier entsprechend angefügt.

3. Elektronik

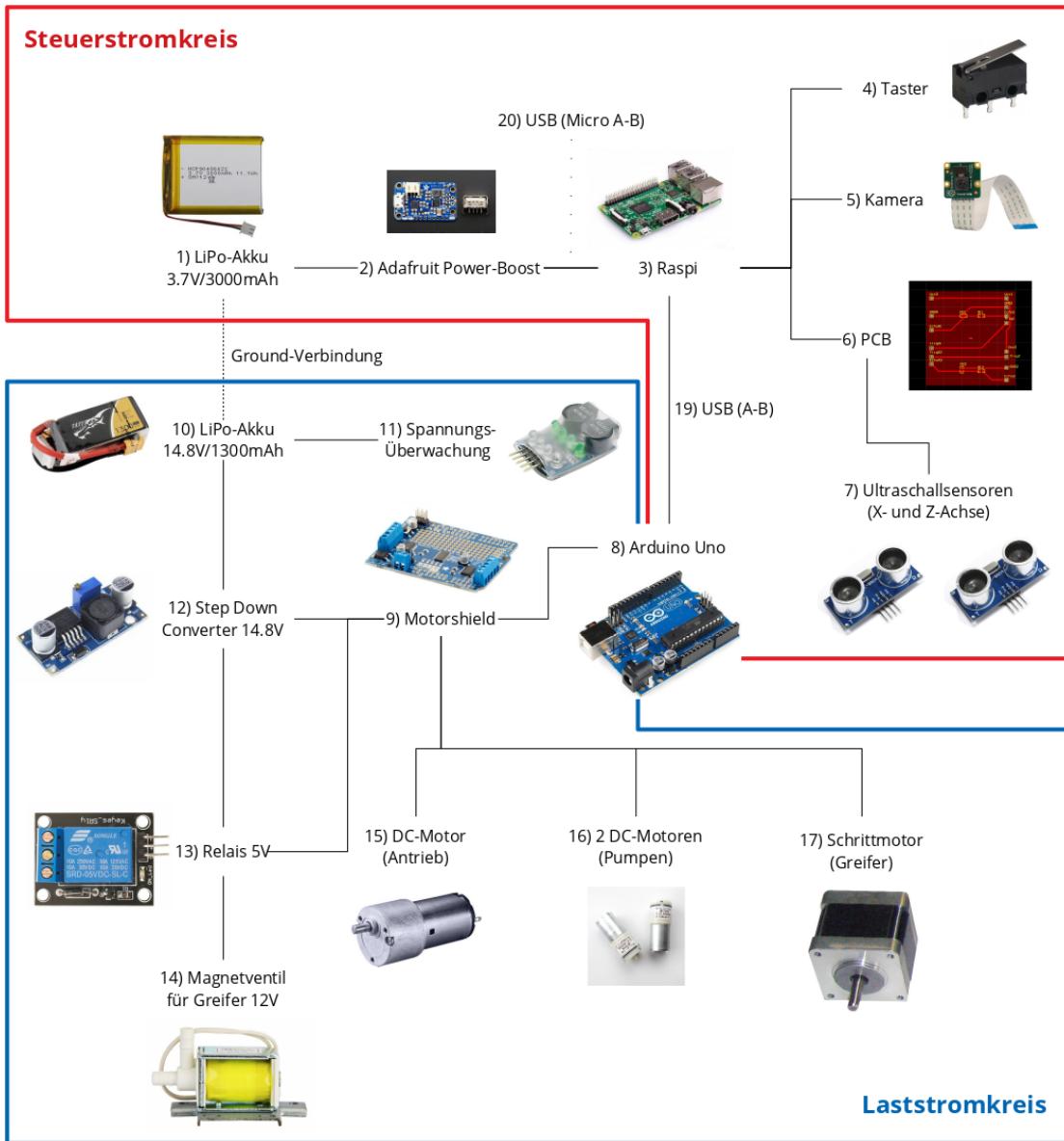


Abbildung 5: Das Blockschaltbild der Elektronik-Komponenten, aufgeteilt in einen Steuerstromkreis (rot) und einen Laststromkreis (blau). Der Arduino ist das Verbindungsglied zwischen diesen beiden Stromkreisen.

3. Elektronik

1. Lithium-Polymer Akku 3.7V/3000mAh HCP90486ZC
2. Adafruit USB PowerBoost/Charger 1A-1000C
3. Entwicklerboard Raspberry Pi 3 Model B
4. Taster 30V DC/2A DC
5. Raspberry Pi Camera Module v2
6. PCB (Printed Circuit Board)
7. Ultraschallsensor HC-SR04 (2×)
8. Mikrocontroller Arduino Uno (*Arduino Uno Rev3*, o.J.)
9. Adafruit Motor Shield v2 (*Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit*, o.J.)
10. RC-Akku Tattu Lithium-Polymer 1300mAh, 14.8V 45C 4SiP (*RC-Akku LiPo 1300 mAh*, o.J.)
11. Spannungsüberwacher LiPo-Summer Pichler (*LiPo-Summer Pichler*, o.J.)
12. LM2596 S DC-DC Adjustable Step Down Buck Converter (*LM2596 S DC-DC Adjustable Step Down Buck Converter*, o.J.)
13. Relaismodul HP-ARL-5V 5V (*Single Relay Board*, o.J.)
14. 12V DC-2-Position 3-Wege elektrisches Magnetventil (*U.S. Solid Mini 12V DC pneumatische Magnetventil 2 Weg Luft Ventil*, o.J.)
15. Getriebemotor 12V Igarashi TYP 33G-50
16. DC5V-12V RF-370 Mini Motor Luftpumpe (2×)
17. Bipolar Stepper Motor 200 Step 0.33A 12V DC
18. Spiralkabel 188mm/500mm 4 × 0.12mm² zwischen Kontrollbox und Greifeinheit
19. USB 2.0 (A-B) Verbindungskabel zwischen Raspberry Pi und Arduino
20. USB A auf Micro USB B Ladekabel für das Raspberry Pi

4. Ablauf

Im Folgenden wird der Ablauf beschrieben, den *Silisloth* beim Absolvieren der gestellten Wettbewerbsaufgabe befolgt:

Vorbereitung *Silisloth* wird hinten am Seil montiert und in Ausgangsposition gebracht. Der Raspi ist mit dem eigens über einen Access-Point zur Verfügung gestellten WiFi-Netzwerk pren7 verbunden. Der Arduino ist per USB-Kabel an den Raspi angeschlossen. Der Raspi wird über eine USV von einem LiPo-Akku gespeist. Die Motoren, Pumpen und das Ventil werden über einen weiteren LiPo-Akku mit Strom versorgt. Die Smartphone-App (siehe *Abschnitt 7.1: Smartphone-App: Java, Seite 30*) ist auf einem Smartphone geöffnet. Auf einem Laptop, der per ssh über das WiFi-Netzwerk mit dem Raspi verbunden ist, kann die Raspi-Anwendung (siehe *Abschnitt 7.2: Raspi: Python, Seite 31*) gestartet werden, sodass der darin enthaltene TCP-Server auf das vom Smartphone aus erteilte Startsignal wartet. Beim Start der Raspi-Anwendung werden weiter die Ultraschallsensoren initialisiert und die serielle Verbindung zum Arduino erstellt. Auf dem Arduino läuft ebenfalls ein Programm (siehe *Abschnitt 7.3: Arduino: C, Seite 33*), das auf Befehle vom Raspi hört (siehe *Abschnitt 5: Kommunikation, Seite 20*).

Erteilung Startsignal Nachdem das Startsignal mündlich erteilt worden ist («drei, zwei, eins, Start»), wird die *Start*-Schaltfläche auf der Smartphone-App betätigt. Das Smartphone nimmt die Verbindung zum TCP-Server auf dem Raspi auf und erteilt diesem das Startsignal.

Zur Last fahren Der Raspi erteilt dem Arduino den Befehl zum Losfahren (G wie «go»). Da die Last zu Beginn an einer genau definierten Position vor dem Anfangspfosten liegt, und da der Abstand vom Anfangs- zum Endpfosten bekannt und konstant ist, kann die Position über der Last mit dem nach vorne zeigenden Ultraschallsensor (X-Ultraschallsensor) ermittelt werden. Der Getriebemotor läuft so lange, bis die zuvor berechnete Distanz zwischen X-Ultraschallsensor und Endpfosten erreicht ist. (Der Abstand zwischen X-Ultraschallsensor und Mittelpunkt des Silikongreifers ist miteinberechnet.)

Die Last greifen Ist nun der Silikongreifer über der Last platziert, muss *Silisloth* anhalten und den Greifer herunterlassen. Dazu wird mit dem Ultraschallsensor, der nach unten zeigt (Z-Ultraschallsensor), die Distanz zum Boden gemessen. Der Raspi erteilt dem Arduino nun den Befehl Sxxx;, wobei S für «stop» steht und xxx für die Anzahl Millimeter (terminiert mit Semikolon), die der Z-Ultraschallsensor als Abstand über dem Boden ermittelt hat. Der Abstand zwischen Z-Ultraschallsensor und Greifer wird herausgerechnet, und die von der Greifereinheit zurückzulegende Distanz wird in Schritte für den Stepper-Motor umgerechnet. Ist die Greifereinheit auf der Last positioniert, wird der Silikongreifer mit den beiden Luftpumpen aufgepumpt, so-

4. Ablauf

dass er sich ausdehnt und die Last möglichst eng umschliesst. Die Ventile werden anschliessend geschlossen.

Weiterfahren Die Greifeinheit wird um die gleiche Distanz angehoben, wie sie zuvor heruntergelassen wurde. Ist dies abgeschlossen, schickt der Arduino dem Raspi ein entsprechendes Signal (`l` für «Last bereit»). Von nun an wird der Abstand der Last zum Boden mittels Ultraschallmessungen (Z-Ultraschallsensor) ermittelt, wovon der Abstand zwischen vertikalem Lastmittelpunkt und Z-Ultraschallsensor jeweils von der ermittelten Höhe subtrahiert wird. Der Abstand zum Endpfosten wird über den X-Ultraschallsensor ermittelt; X- und Z-Koordinaten werden auf das Smartphone übertragen. *Silisloth* fährt mit erhöhtem Tempo weiter.

Zielfeld entdecken Nachdem der Raspi erkannt hat, dass die Fahrt wieder aufgenommen wurde, wird die Bildverarbeitung aktiviert. Sobald das Zielfeld zum ersten Mal «erkannt» wurde, wird die Fahrtgeschwindigkeit reduziert, wozu der Raspi dem Arduino den Befehl zum Verlangsamen (`d` für «decelerate») schickt. Die Fahrtgeschwindigkeit wird reduziert, damit die Zielfelderkennung mehr Bilder pro Zeiteinheit verarbeiten und so die Endposition genauer bestimmen kann.

Zielposition bestimmen In verlangsamter Fahrt werden nun weitere Bilder vom Zielfeld aufgenommen und ausgewertet. Sobald die Zielfelderkennung eine horizontale Distanz von unter 15cm zum Mittelpunkt des Zielfeldes ausgemacht hat, wird die weitere Positionierung mit dem X-Ultraschallsensor vorgenommen. (Da die Kamera vorne angebracht ist, kann sie das Zielfeld nicht mehr «sehen», sobald die Last darüber positioniert ist, da die Greifeinheit den Blick darauf versperrt.) Die beiden zuletzt ermittelten Distanzen – optisch und per Ultraschallsensor – werden zwischengespeichert. Aus diesen Angaben kann nun die Distanz berechnet werden, die der X-Ultraschallsensor zum Endpfosten haben muss, um die Last in der Mitte des Zielfeldes abzuwerfen (siehe dazu *Formel 2, Seite 19* und *Abbildung 6, Seite 19*).

Last abwerfen Sobald die vom X-Ultraschallsensor gemessene Distanz kleiner oder gleich der zuvor ermittelten Enddistanz ist, kann die Last abgeworfen werden. Hierzu verwendet der Raspi wiederum den Befehl der Form `sxxx;`, um die Greifeinheit um die vom Z-Ultraschallsensor ermittelte Anzahl Millimeter zum Boden herunterzulassen. Der Greifer wird über die Ventile gelöst – was wesentlich schneller geht, als das Aufpumpen – und die Last wird möglichst genau in der Mitte des Zielfeldes liegen gelassen. Die Übertragung der X- und Z-Koordinaten wird nun eingestellt, wobei die Z-Koordinaten beim Absetzen der Last berechnet und nicht gemessen werden.

Zum Endpfosten fahren Der Arduino teilt dem Raspi über das `l`-Signal («Last bereit») mit, dass die Fahrt nun wieder aufgenommen wird. Dies geschieht zunächst mit voller Geschwin-

4. Ablauf

digkeit, bis der X-Ultraschallsensor eine Distanz von weniger als 25cm meldet. Die Geschwindigkeit wird reduziert, indem der Raspi das D -Signal («*decelerate*») an den Arduino schickt. Der Endtaster ist ca. 15cm vor dem X-Ultraschallsensor angebracht, sodass die letzten 10cm in langsamer Fahrt zurückgelegt werden. Bei der Berührung mit dem Endpfosten wird der Mikroendschalter betätigt, was vom Raspi registriert wird. Er erteilt nun das H -Signal (für «*halt*») an den Arduino, damit dieser den Motor stoppen kann.

Zurückfahren Nach wenigen Sekunden am Endpfosten fährt *Silisloth* automatisch einige Sekunden zurück, sodass ein weiterer Durchlauf in Angriff genommen werden kann.

4.1. Berechnung der Enddistanz

$$E_{\overline{XP}} = D_{\overline{XP}} - D_{\overline{KZ}} - D_{\overline{KX}} - D_{\overline{KL}} \quad (2)$$

$E_{\overline{XP}}$ = Enddistanz von X-Ultraschallsensor zum Endpfosten

$D_{\overline{XP}}$ = aktuelle Distanz von X-Ultraschallsensor zu Endpfosten

$D_{\overline{KZ}}$ = aktuelle Distanz von Kamera zu Zielfeldmittelpunkt

$D_{\overline{KX}}$ = Abstand von Kamera zu X-Ultraschallsensor und

$D_{\overline{KL}}$ = Abstand von Kamera zu Lastmittelpunkt

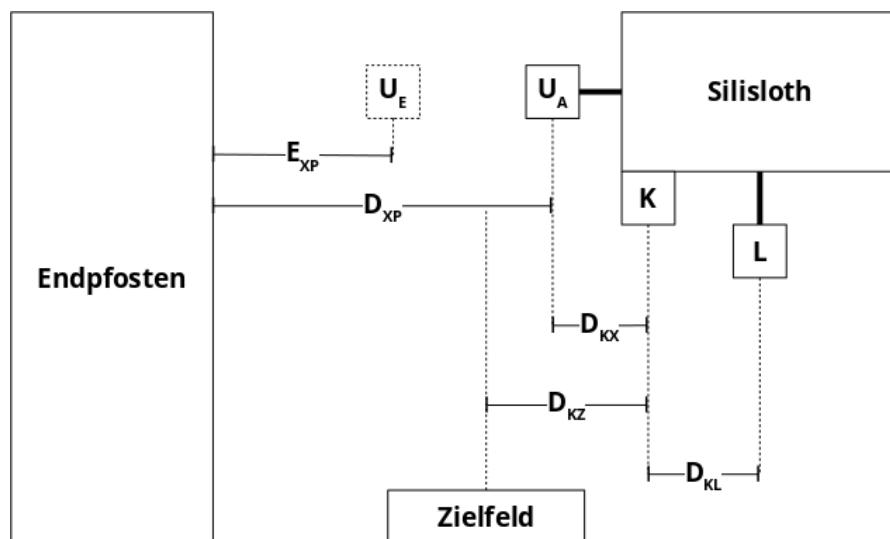


Abbildung 6: Die Berechnung der Enddistanz (Position des Lastabwurfs): Der Ultraschallsensor muss von Position U_A nach U_E gebracht werden, sodass er eine Distanz von $E_{\overline{XP}}$ bis zum Endpfosten misst, und sich die Last über der Mitte des Zielfelds befindet.

5. Kommunikation

Zum Betreiben von *Silisloth* sind mehrere Computer involviert:

- In *Silisloth* integrierte Computer
 - Raspberry Pi: Steuerung, Bildverarbeitung, Ultraschallsensoren, Mikroendschalter
 - Arduino: Steuerung von Schrittmotor, Getriebemotor, Luftpumpen und Ventil
- Externe Computer
 - Laptop: Aufstarten der Steuerungssoftware, Betrachten und Auswerten der Logdateien
 - Smartphone: Erteilung des Startsignals, Überwachung der Lastkoordinaten

Diese vier Computer müssen miteinander kommunizieren. *Abbildung 7, Seite 20* zeigt, wie sie miteinander verbunden sind und welche Protokolle dabei zum Einsatz kommen.

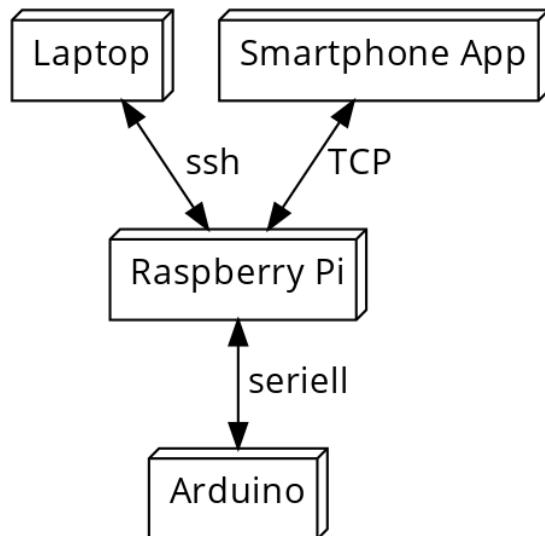


Abbildung 7: Die Verbindungen zwischen den vier involvierten Computern

Das Sequenzdiagramm am Ende des Abschnitts (siehe *Abbildung 8, Seite 23*) bietet einen abschließenden Überblick bezüglich der Kommunikation zwischen den vier involvierten Computern beim Absolvieren der Wettbewerbsaufgabe.

5.1. Laptop – Raspi

Der Laptop verbindet sich per `ssh` mit dem Raspi. Da die IP-Adresse des Raspi nicht konstant ist, erfolgt die Verbindungsaufnahme über ein Skript, das in der Dokumentation zu PREN 1 (Bertozzi et al.,

SYNTAX	SEMANTIK	SENDER → EMPFÄNGER
G	«Go» für losfahren bzw. weiterfahren	Raspi → Arduino
Sxxx;	«Stop» mit xxx als vertikale Distanz in mm zur Last	Raspi → Arduino
D	«Decelerate» für verlangsamen	Raspi → Arduino
L	«Load ready» für Last bereit (aufgenommen/abgesetzt)	Arduino → Raspi
H	«Halt» für anhalten (und beenden)	Raspi → Arduino

Tabelle 1: Die Befehle für das Kommunikationsprotokoll zwischen Raspi und Arduino

2017, S. 34) beschrieben ist. Ein Raspi ist seit Mitte Semester fix auf dem Prototypen verbaut. Damit auch zu Hause Tests mit einzelnen Komponenten durchgeführt werden konnten, kam ein weiterer Raspi zum Einsatz. Dieser hat eine andere MAC-Adresse als derjenige, der auf dem Prototypen verbaut ist. Das *raspi-ssh*-Skript wurde für das komfortable Auffinden des richtigen Raspis im Netzwerk deshalb parametrisiert.

5.2. Raspi – Arduino

Die Kommunikation zwischen Raspi und Arduino erfolgt per serieller Verbindung über ein USB-Kabel. Die Kommunikation geht in beide Richtungen. Die Befehle wurden so kurz wie möglich gehalten und bestehen nur aus einem möglichst selbstsprechenden Zeichen (siehe *Tabelle 1: Die Befehle für das Kommunikationsprotokoll zwischen Raspi und Arduino, Seite 21*). Eine Ausnahme bildet der «Stop»-Befehl, der zusätzlich einen ganzzahligen Parameter erwartet. Das Ende dieses Parameters wird mit einem Semikolon markiert.

Auf dem Arduino läuft ein Kommunikationstask, der Befehle parallel zur eigentlichen Steuerungslogik empfangen kann. Der Raspi wartet hingegen blockierend auf die Kommunikation vom Arduino, da es in den betreffenden Phasen auf dem Raspi nichts zu tun gibt. Threads würden an dieser Stelle nichts bringen und nur unnötig die Komplexität erhöhen.

Zu Beginn des Projekts wurden die Befehle nicht in einzelne Zeichen sondern in ganzen Wörtern kodiert. Das hat den Vorteil, dass ganze Wörter selbstsprechend sind und das Protokoll so selbsterklärend wird. Die Übertragung dieser Strings wurde beidseitig mit gepuffertem I/O umgesetzt. Dies führte jedoch zu starken Verzögerungen bei der Kommunikation. Deshalb wurde auf ein primitiveres Protokoll gewechselt, bei dem nur einzelne Bytes geschrieben und gelesen werden müssen. Das Parsen eines ganzzahligen Parameters erforderte jedoch etwas zusätzliche Programmlogik.⁸

⁸Die Höhe wird bewusst in Millimetern und nicht in Zentimetern angegeben, da das Auslesen einer Fliesskommazahl wesentlich schwieriger ist als das Auslesen einer Ganzzahl.

5.3. Smartphone-App – Raspi

Die Kommunikation zwischen Smartphone-App und Raspi erfolgt über eine TCP-Verbindung. Der Raspi übernimmt dabei die Rolle des Servers. Das Steuerungsprogramm auf dem Raspi wird aufgestartet und wartet auf eine Verbindung. Wird auf der Smartphone-App die Schaltfläche *Start* betätigt, wird die Verbindung mit dem Server (Raspi) aufgenommen. Dies gilt sogleich als Startsignal.

Ist die Last aufgenommen, werden die x- und z-Koordinaten an die Smartphone-App übertragen. Die beiden Parameter werden in eine Zeichenkette der Form `x=xxx;z=zzz\r\n` übertragen, wobei xxx und zzz für Distanzangaben in Millimetern stehen. Die Nachricht wird mit der Zeichenfolge *carriage return* (`\r`), *new line* (`\n`) terminiert. Auf der Smartphone-App wird diese Zeichenkette geparsst, und die Koordinaten werden in den entsprechenden Textfeldern angezeigt.

5. Kommunikation

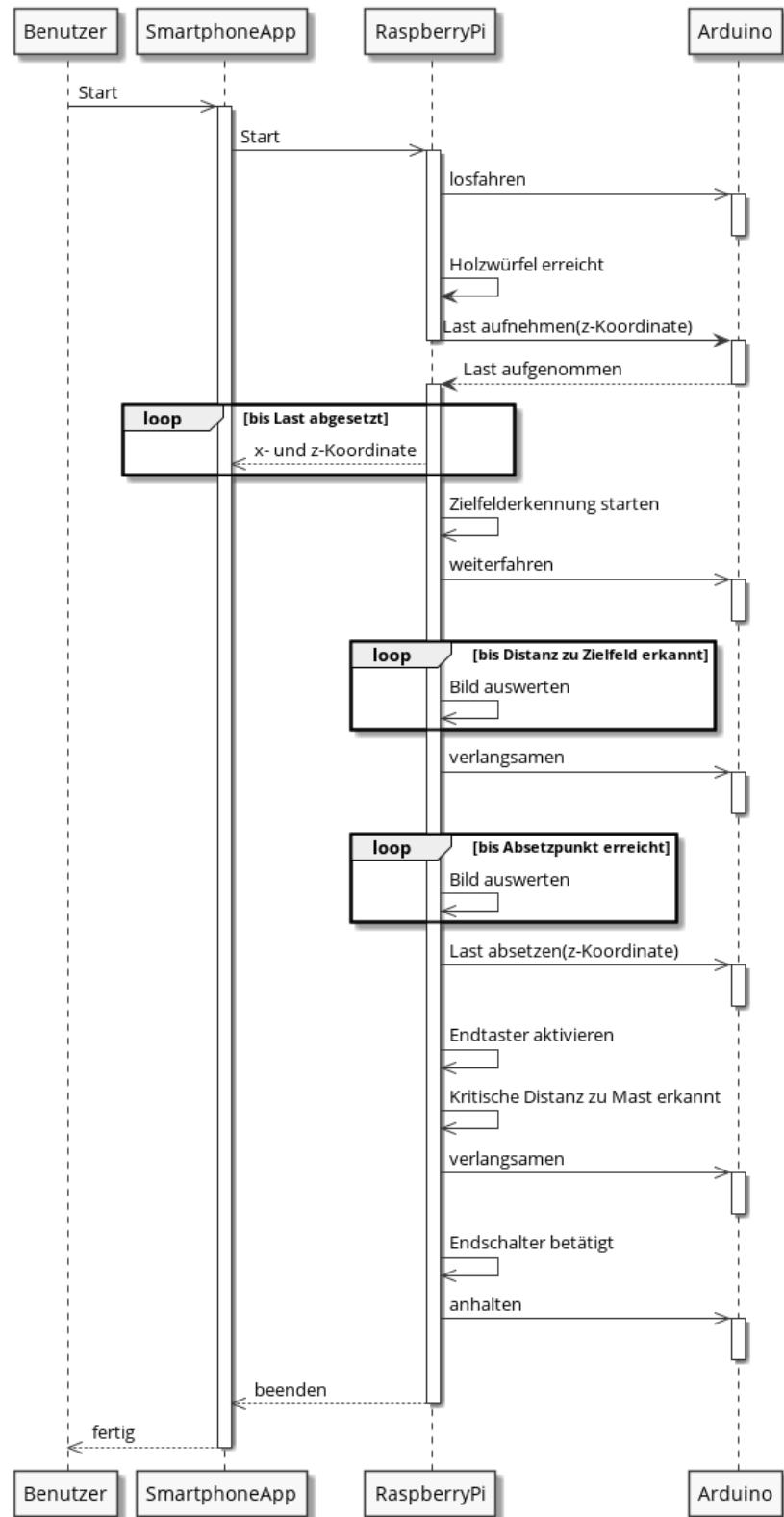


Abbildung 8: Die Kommunikation zwischen Benutzer, Smartphone, Raspi und Arduino

6. Bildverarbeitung

Weil das Zielfeld im Gegensatz zum Holzwürfel nicht an einer vorher definierten Stelle auf der x-Achse zu liegen kommt, muss dieses optisch erkannt werden. Hierzu wird, wie in PREN 1 (Bertozzi et al., 2017, S. 25-26) beschrieben, die Raspi-Cam zusammen mit der Bildverarbeitungsbibliothek OpenCV und der Programmiersprache Python 3 verwendet. Die Herausforderung besteht nicht nur in der Erkennung des Zielfeldes, sondern auch in der Ermittlung der jeweils gegenwärtigen Distanz der Last zum Zielfeld.

6.1. Lösungsansatz

Wird die Kamera nach unten zeigend parallel zur Unterlage ausgerichtet, ist der Bildmittelpunkt direkt unterhalb der Kameralinse. Die Distanz von der Kameralinse zum Lastmittelpunkt kann anhand der CAD-Pläne auf ca. 11cm festgelegt werden. Dies ist die Distanz, welche die Kamera über das Ziel hinweg positioniert werden muss, um die Last möglichst in der Mitte des Zielfeldes abzulegen.

Die Seitenlänge des innersten Zielfeldquadrates ist in der Aufgabenstellung mit 6cm definiert. Dies ist ein wichtiger Referenzwert, anhand dessen der horizontale Abstand zwischen Kameralinse und Zielfeldmittelpunkt ermittelt werden kann.

Die Kamera kann die Seitenlänge des innersten Zielfeldes sowie den Abstand von Bildmitte zum Mittelpunkt des innersten Zielfeldes in Pixeln ermitteln. Mit dem Bruch $\frac{\text{ZielfeldhöhePixel}}{6\text{cm}}$ kann ermittelt werden, wie vielen Pixeln ein Zentimeter auf dem Bild entspricht. Der horizontale Abstand zwischen Kameralinse und Zielfeldmittelpunkt kann so von Pixeln in eine Distanz in Zentimetern umgerechnet werden.

Dieser Lösungsansatz stellt besondere Anforderungen an die Konstruktion:

1. Die Sicht der Kamera nach unten und vorne darf durch kein Bauteil versperrt werden.
2. Die Kamera muss möglichst parallel zum Untergrund ausgerichtet sein. Seitliche Schwingungen sind unproblematisch, solange das Zielfeld im Sichtfeld bleibt. Starke Schwingungen nach vorne und hinten hingegen lassen das Zielfeld optisch über die gedachte horizontale Bildmittellinie springen, wodurch der hier ausgearbeitete Lösungsansatz nicht mehr funktionieren würde.

Die erarbeitete mechanische Konstruktion erfüllt diese beiden Anforderungen. (Einzig der Blick nach hinten ist von der Greifeinheit versperrt, wodurch zur genauen Positionierung über dem Zielfeld auf den Ultraschallsensor zurückgegriffen werden muss.)

6.2. Umsetzung

Die Python-Klasse `ImageAnalyzer` verfügt über eine Methode namens `calculate_distance`, welche von aussen aufzurufen ist. Diese erwartet als Parameter ein Bild als NumPy-Matrix im BGR-Farbraum.

6. Bildverarbeitung

Das Bild wird zunächst in Graustufen umgewandelt, indem alle Bildpunkte abhängig von ihrem Helligkeitswert schwarz oder weiss eingefärbt werden.⁹ Das Verfahren wird in *Anhang A.2: Thresholding-Benchmarks* beschrieben. So wird verhindert, dass feine Linien und Verunreinigungen sowie Klebebander nicht als Kanten erkannt werden, sondern gleich verschwinden.

Die Polygon-Konturen auf einem Bild können mit der OpenCV-Funktion `findContours` ermittelt werden. Diese gibt neben einer Liste von Konturmatrizen – bestehend aus den Koordinaten der Polygonknoten – die dazugehörigen Polygonhierarchien zurück. Letztere gibt Auskunft über die Verschachtelungen der ermittelten Polygone.¹⁰

Die so ermittelten Polygone müssen genauer untersucht werden, denn es werden bei weitem mehr gefunden als bloss die konzentrischen Quadrate des Zielfeldes: etwa die gezackten Hindernisse in Form von Tannenattrappen oder die Holzplatten der Unterlage. Diese sollen nach dem Ausschlussverfahren eliminiert werden, indem folgende Heuristiken zum Einsatz kommen:

Benachbarte Polygone Liegen zwei Polygone nebeneinander, bezeichnet man diese als benachbart. Das Zielfeld ist eine Hierarchie von Polygonen, und nur das äusserste Quadrat kann so plausiblerweise einen (oder mehrere) Nachbarn haben. Ein Polygon, das einen Nachbarn hat, ist kein inneres Quadrat des Zielfeldes und kann somit ausgeschlossen werden.

Quadratische Form Die zu ermittelnden Polygone sind auf dem Bild nahezu quadratisch, wobei durch perspektivische Verzerrung und Verschwimmen des Bildes (wegen Bewegungsunschärfe) niemals perfekt quadratische Polygone gefunden werden können. Es genügt jedoch Polygone, die nicht einmal näherungsweise ein Quadrat darstellen können, aufgrund zweier Kriterien auszuschliessen:

1. **ANZAHL ECKEN**: Ein Quadrat hat vier Ecken. Ein Hindernis, wie die grüne Tannenattrappe ungefähr zwanzig. Polygone mit einer von vier abweichenden Anzahl Knoten können ausgeschlossen werden.
2. **VERHÄLTNIS HÖHE/BREITE**: Bei einem Quadrat ist die Höhe gleich der Breite. Bei einem nahezu quadratischen Polygon sollte das Verhältnis $\frac{\text{Höhe}}{\text{Breite}}$ im Bereich $[0.75..1.25]$ ¹¹ liegen.

Absolute Grösse Das innerste Quadrat des Zielfeldes nahm bei Tests immer mehr als 0,5% der gesamten Bildfläche ein. Das äusserste Quadrat war niemals grösser als 95% der Bildfläche. Polygone, deren Grösse nicht in dieses Fenster fällt, können ausgeschlossen werden.¹²

⁹Das Zielfeld ist schwarz-grau und nicht schwarz-weiss. In der Praxis trifft man kaum auf ein reines Schwarz oder Weiss.

¹⁰Ein Eltern-Polygon beinhaltet ein Kind-Polygon vollständig. Benachbarte Polygone schneiden einander nicht. Das äusserste Polygon einer Hierarchie hat kein Elternpolygon, das innerste Polygon hat kein Kindpolygon.

¹¹Diese Werte wurden bei Versuchen ermittelt und funktionierten zuverlässig.

¹²Je nach konfigurierter Vorder- und Hintergrundfarbe kann der Bildrand als Polygon erkannt werden. Die Fläche dieses Polygons beträgt dann 100% der Bildgrösse.

Nachdem die Anzahl der gefundenen Polygone nach dem beschriebenen Heuristik-Ausschlussverfahren reduziert worden ist, bleibt eine Liste von *Kandidaten* übrig, die möglicherweise ein Zielfeld darstellen. Nun gilt es, diese Kandidaten genauer zu untersuchen. Hierzu wird wiederum die Polygonhierarchie verwendet, die schon bei der Ermittlung benachbarter Polygone hilfreich war.

6.2.1. Die Polygonhierarchie

Die bereits erwähnte OpenCV-Funktion `findContours` gibt neben den Polygonkoordinaten auch für jedes Polygon eine Liste bestehend aus vier Zahlen zurück. Die ersten beiden Zahlen handeln von der Nachbarschaftsbeziehung und sind Referenzen auf den Index des vorhergehenden bzw. des nachfolgenden Nachbarpolygons. Da sich in der gefilterten Kandidatenliste keine Polygone mit Nachbarn mehr befinden, können diese Angaben ignoriert werden.¹³

Die dritte Zahl gibt den Index des ersten Kindpolygons an. (Da Polygone mit Nachbarn nicht in der Liste vorkommen, gibt es immer nur ein Kindpolygon.) Die vierte Zahl gibt den Index des Elternpolygons an.¹⁴

Gesucht ist das *innerste Zielfeld*. Dieses ist definiert als ein Quadrat, das von einem anderen Quadrat umschlossen wird, aber selber keine weiteren Quadrate enthält. Übersetzt auf die Hierarchie-Liste bedeutet das also, dass das dritte Element (erstes Kindpolygon) auf kein anderes Polygon verweisen darf (Wert: -1) und das vierte Element (Elternpolygon) einen gültigen Wert haben muss (Konturindex des jeweiligen Elternpolygons). Wird ein solches Quadrat gefunden, das zuvor schon durch verschiedene Heuristikprüfungen gekommen ist, wird dieses als das innerste Feld des Zielfeldes angenommen. Der Mittelpunkt dieses Polygons wird über die OpenCV-Funktion `moments` ermittelt.

Wie bereits weiter vorne beschrieben (siehe *Abschnitt 6.1: Lösungsansatz, Seite 24*) kann nun der Abstand zu diesem Zielfeld errechnet werden. Die genaue Positionierung der Last über dem Zielfeld ist im Abschnitt 4: *Ablauf, Seite 17* beschrieben.

6.2.2. Abwägungen

Die Ermittlung des innersten Zielfeldes könnte noch etwas solider umgesetzt werden, indem nicht nur geprüft wird, ob das innerste Zielfeld *ein* Elternpolygon hat, sondern indem eine *rekursive* Prüfung vorgenommen würde. Das Elternpolygon des innersten Zielfeldes hat wiederum ein Elternpolygon, welches wiederum ein Elternpolygon hat usw. Das vorliegende Zielfeld besteht aus sechs konzentrischen Quadraten, das innerste Zielfeld müsste also fünf übergeordnete Hierarchiestufen besitzen – und nicht bloss eine. Eine rekursive Prüfung wäre somit solider.

Andererseits bringt eine rekursive Prüfung auch Probleme mit sich:

¹³Sie haben allesamt den Wert -1. Das bedeutet: Kein Nachbar vorhanden.

¹⁴Die Hierarchie-Liste mit vier Elementen ist also als Knoten von gleich zwei verketteten Listen zu verstehen: Einerseits als Element der Nachbarschaftskette und andererseits als Element der Eltern-Kind-Beziehungskette.

Implementierung Eine rekursive Prüfung ist aufwändiger umzusetzen und somit fehleranfällig. Es besteht die Gefahr für Endlosschleifen und somit für einen Stack Overflow, der das Programm zum Abstürzen bringen würde. Das Problem ist zugegebenermaßen nicht schwierig zu lösen, und die Endlosschleife könnte mittels Heuristiken vermieden werden. Die Codemenge und -komplexität würde dennoch zunehmen – und damit auch die Fehlerwahrscheinlichkeit.

Anzahl Hierarchiestufen *Abbildung 9e, Seite 28* und *Abbildung 9f, Seite 28* zeigen das Zielfeld während zweier kurz aufeinanderfolgenden Phasen der Überfahrt. Einmal werden die äusseren beiden Quadrate erkannt, einmal nicht. Dies hat wohl damit zu tun, dass die feine weisse Linie zur Ausrichtung des Zielfeldes auf der x-Achse auf dem etwas verschwommenen Bild den Eindruck macht durchgezogen zu sein. Optisch können keine Quadrate sondern nur halbierte Quadrate (eine andere Art von Polygon mit acht statt vier Ecken) erkannt werden. Die Anzahl der zu prüfenden Hierarchiestufen kann also nicht mit absoluter Sicherheit angegeben werden.

Früherkennung Diese beiden genannten Probleme sind eher Erschwernisse und Einschränkungen und sollten kein Hindernis für die Umsetzung der rekursiven Hierarchieprüfung darstellen. Aus Gründen der Performance ist die Hierarchieprüfung aber ein grosses Problem. Hierbei geht es nicht etwa um die Programmlaufzeit, sondern um den Zeitpunkt, ab dem das Zielfeld zum ersten mal erkannt werden kann. Ein Blick auf die Bildserie (Seite 28) führt dies vor Augen: Auf dem ersten Bild wird noch kein Zielfeld erkannt, da noch kein Quadrat vollständig auf dem Bild sichtbar ist. Auf dem zweiten Bild wird ein Quadrat erkannt.¹⁵ Auf dem dritten Bild werden bereits drei Quadrate erkannt, auf dem vierte Bild sind es bereits fünf Quadrate. Sechs Quadrate werden zum ersten mal auf dem fünften Bild erkannt. Würde die Hierarchie rekursiv geprüft, würde das Zielfeld nicht zum ersten mal mit einem Abstand von 23.4cm (*Abbildung 9c, Seite 28*) sondern erst mit einem Abstand von 16.3cm (*Abbildung 9e, Seite 28*) erkannt. Je tiefer die zu prüfende Hierarchie gewählt wird, desto später wird das Zielfeld erkannt.

Aus diesen genannten Gründen wurde auf eine *rekursive* Hierarchieprüfung verzichtet. Bei Tests ist bisher nie ein falsches Zielfeld erkannt worden. Die Probleme bestanden eher darin, dass zu Beginn bestimmte Parameter zu streng definiert waren, was später etwas gelockert wurde.

6.3. Auflösung und Verarbeitungsgeschwindigkeit

Das verwendete Kameramodul (*Camera Module, 2016*) unterstützt eine Bildauflösung von bis zu 3280×2464 Bildpunkten, was acht Megapixeln entspricht. Bei den Tests wurden verschiedene Auflösungen

¹⁵Die Bildserie stammt von einem frühen Prototyp der Bildverarbeitung, die alle erkannten Quadrate markiert hat. Mit der Hierarchieprüfung dürfte auf dem zweiten Bild noch kein Zielfeld erkannt werden.

6. Bildverarbeitung

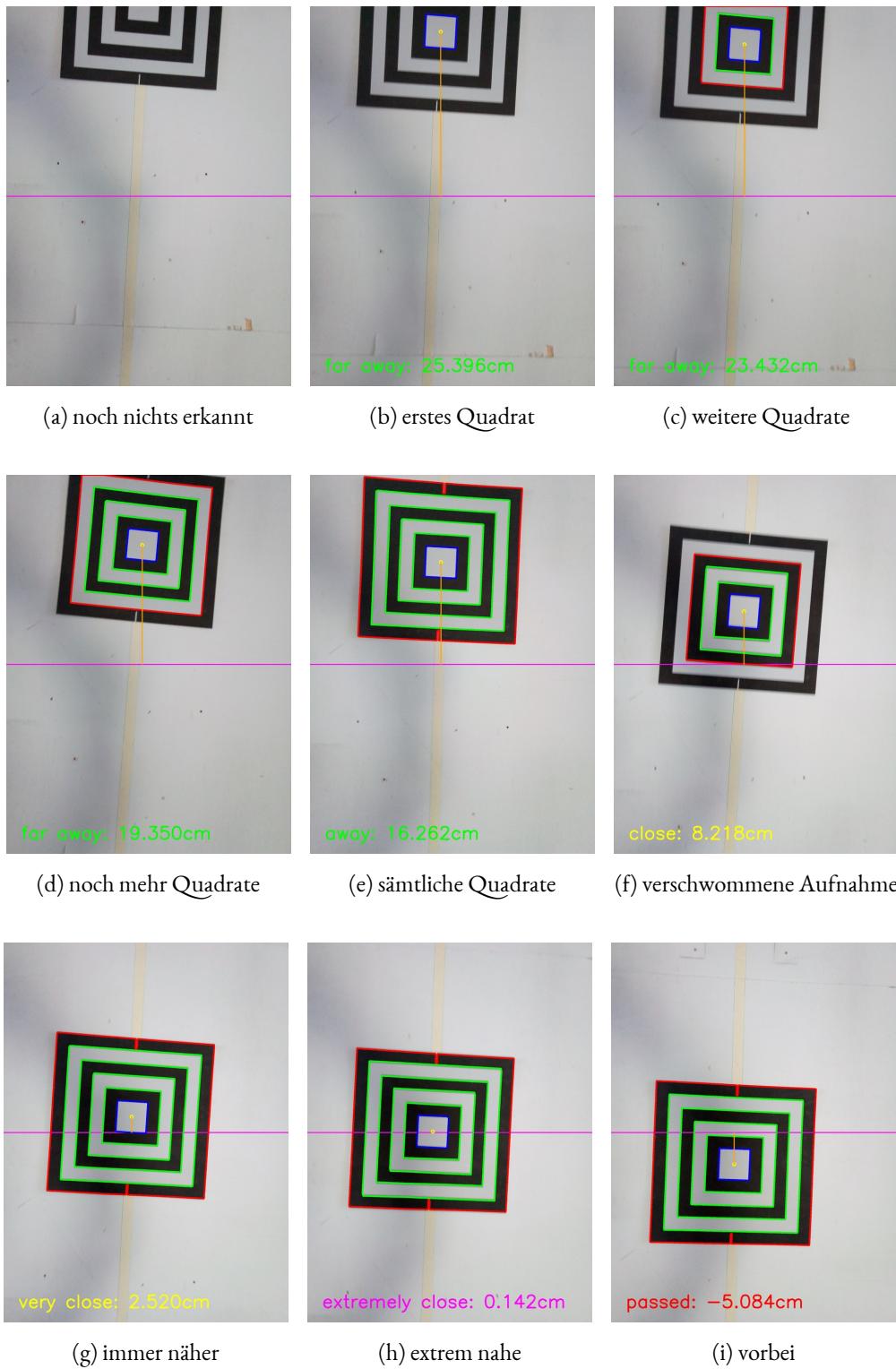


Abbildung 9: Die Bildserie zeigt das Überfahren des Zielfeldes aus Kameraperspektive.

6. Bildverarbeitung

ausprobiert, etwa HD-Auflösung (1080×1920 Pixel).¹⁶ Die besten Ergebnisse wurden mit einer Auflösung von 480×640 (VGA-Auflösung) erreicht. Diese geringe Auflösung hat folgende Vorteile:

Performance Das Auswerten von Bildern in HD-Auflösung dauerte ca. 550 Millisekunden. Mit VGA-Auflösung konnten Bilder in durchschnittlich 45 Millisekunden ausgewertet werden. Das entspricht einem Performancegewinn von ca. Faktor 12.

Unschärfe Bei der vorliegenden Aufgabe hat sich eine gewisse Unschärfe bei den aufgenommenen Bildern als vorteilhaft erwiesen, da kleinere Konturen, wie etwa Verschmutzungen auf dem Zielfeld und Klebestreifenrückstände auf der Unterlage, bereits bei der Bildaufnahme verschwimmen und so nicht als Konturen erkannt werden. Die scharfen Konturen des Zielfeldes waren bei den Aufnahmen aber dennoch genug scharf.

Für die Testdurchläufe wurden die aufgenommenen Bilder zur späteren Auswertung jeweils abgespeichert. Das Abspeichern der Bilder nahm jeweils ungefähr die Hälfte der Bildverarbeitungszeit in Anspruch: Dauerte die Verarbeitung eines Bildes 45 Millisekunden, benötigte die Abspeicherung weitere 20 Millisekunden. Für den Wettbewerb wird das Abspeichern der Bilder aus Performancegründen deaktiviert.

¹⁶Die Orientierung der Kamera wird über die Auflösung der Form Breite \times Höhe angegeben. Die Auflösung 1920×1080 entspricht der HD-Auflösung in *Landscape*-Orientierung (Breitbild), die Auflösung 1080×1920 entspricht der HD-Auflösung in *Portrait*-Orientierung (Hochformat). Da das Blickfeld der Kamera möglichst weit nach vorne und nicht in die Breite reichen soll, wird das Hochformat verwendet.

7. Software

7.1. Smartphone-App: Java

Um die autonome Laufkatze zu starten wird eine Smartphone-App verwendet. Dazu muss sich das Smartphone im selben Netzwerk wie der Raspi befinden. Der Raspi übernimmt dabei die Rolle des Servers, und das Smartphone verbindet sich als Client mit ihm. Betätigt der Anwender die Starttaste der App, verbindet sich das Smartphone mit dem Raspi, und die Laufkatze wird gestartet. Sobald die Last aufgenommen wurde, sendet der Raspi fortlaufend die aktuellen Koordinaten der Last an die Smartphone-App. Auf diese Weise sieht der Anwender ununterbrochen die aktuellen Lastkoordinaten auf seinem Smartphone. Dabei zeigt das Smartphone lediglich die empfangenen Koordinaten an und übernimmt keinerlei Logik.

Damit die Kommunikation zwischen Smartphone und Raspi funktioniert, müssen sich beide Geräte im selben Netzwerk befinden. Dazu wird mittels eines *maplite*-Routers ein eigenes Netzwerk mit der SSID `pren7` aufgebaut. Mit diesem Netzwerk verbinden sich der Raspi und das Smartphone. Damit sich das Smartphone mit dem Raspi verbinden kann, wurde für den Raspi eine statische IP-Adresse konfiguriert.

Für die Entwicklung der Smartphone-App wurde das Android Studio verwendet. Das Android Studio bietet viele Vorteile für die Entwicklung, zum Beispiel einen integrierten Emulator, oder das Debuggen direkt auf dem Smartphone.

Abbildung 10, Seite 30 zeigt die Benutzeroberfläche der Smartphone-App.

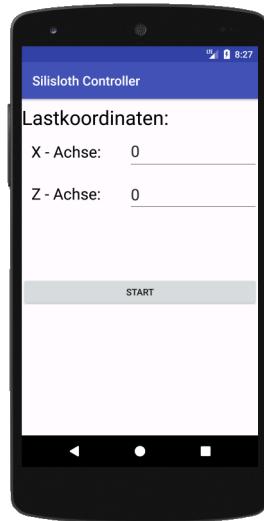


Abbildung 10: Die Smartphone-App zur Steuerung von *Silisloth* und zum Ablesen der Koordinaten

7.2. Raspi: Python

Die übergeordnete Steuerung und Koordination von *Silisloth* wurde auf dem Raspi mit Python umgesetzt. Der Quellcode zum Projekt findet sich im Anhang (*silisloth-raspi.zip*)

7.2.1. Projektstruktur

Das Projekt besteht aus verschiedenen Quellcodedateien:

Arduino.py Kommunikation mit dem Arduino per serieller Schnittstelle (USB): Die angebotenen Funktionen bieten eine selbsterklärende Schnittstelle für die zu erteilenden und empfangenden Befehle.

Brain.py Die übergeordnete Programmlogik: Der ganze Ablauf (siehe *Abschnitt 4: Ablauf, Seite 17*) wird von diesem Skript gesteuert.

EndSwitch.py Ansteuerung des Mikroendschalters: Es wird eine Funktion zur Verfügung gestellt, die blockierend auf das Betätigen des Mikroendschalters wartet.

ImageAnalyzer.py Die Bildverarbeitungslogik: Verarbeitet ein einzelnes Bild und gibt eine Millimeterangabe zurück, die den optischen Abstand in x-Richtung von Kameralinse zum Zielfeldmittelpunkt in Millimetern zurückgibt.

Log.py Logger-Konfiguration und -Format für sämtliche im Projekt geschriebenen Logmeldungen: Für die Ausgabe auf den Bildschirm wird aus Gründen der Übersichtlichkeit ein tieferes Log-Level verwendet als für die Ausgabe in die Logdatei.

TargetDetection.py Fassade für den ImageAnalyzer: Kümmert sich um die Aufnahme von Bildern und stellt eine vereinfachte Schnittstelle für die Bildverarbeitungslogik und das Abspeichern der aufgenommenen Bildern zu Debug-Zwecken zur Verfügung.

TcpServer.py Kommunikation zum Smartphone: Wartet blockierend auf eine eingehende Verbindung und schickt die formatierte Zeichenkette mit den Koordinaten an die Smartphone-App.

UltrasonicSensor.py Ansteuerung der Ultraschallsensoren: Bietet Funktionen zur Erstellung von Messreihen. Es kann eine bestimmte Anzahl von Messungen erstellt oder eine bestimmte Zeit lang gemessen werden. Von der Messreihe wird der Median-Wert zurückgegeben, was die Messung robust für Fehlmessungen macht (Bertozzi et al., 2017, S. 46-48).

Zum Testen einzelner Komponenten wurden folgende Demo-Programme entwickelt:

7. Software

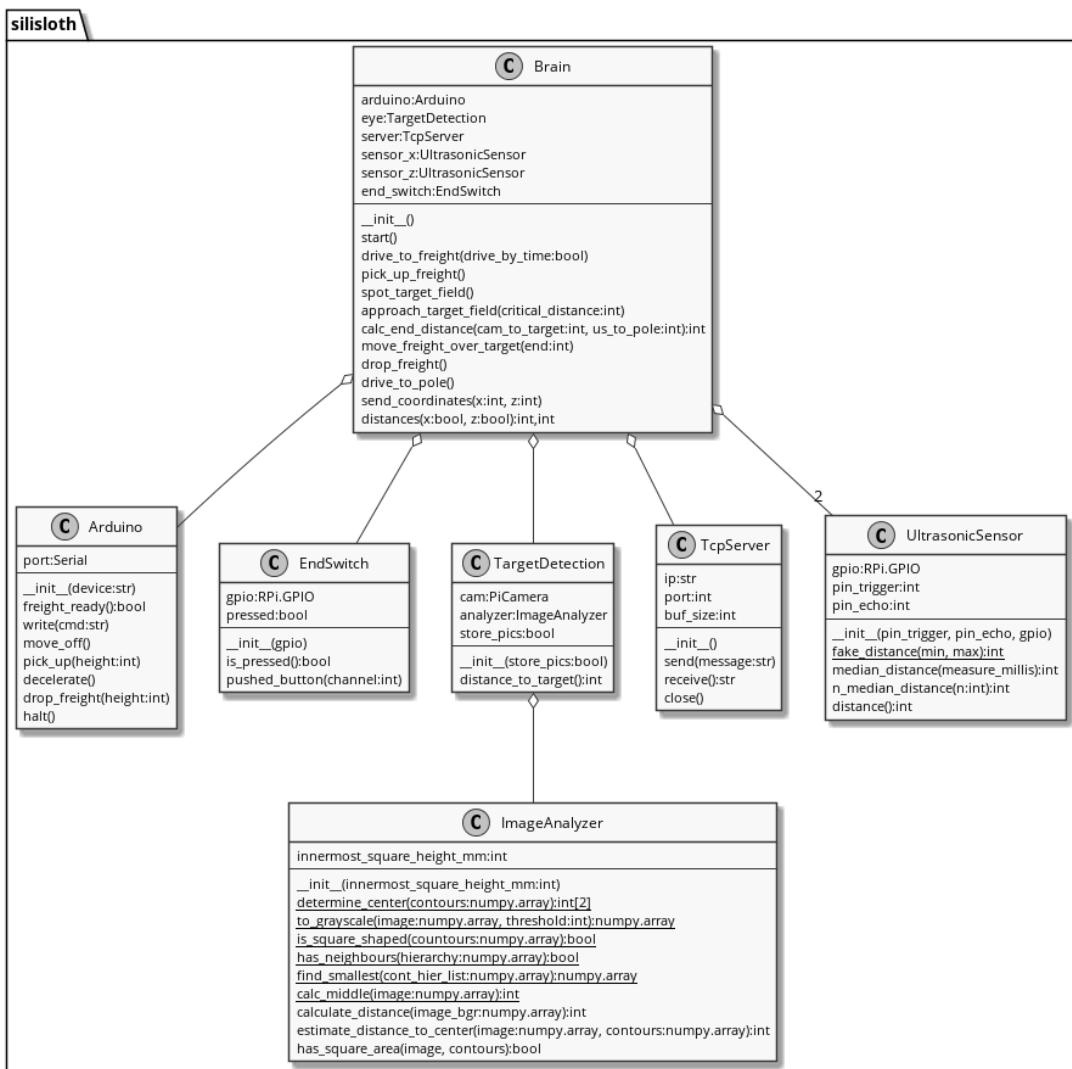


Abbildung II: Das Klassendiagramm des Python-Projektes

7. Software

ArduinoMock.py Testprogramm für die serielle Kommunikation mit dem Arduino: Befehle können direkt über die Kommandozeile an den Arduino gesendet werden, was zum Testen und zur Fehlersuche hilfreich ist.

ImageAnalyzerDemo.py Testprogramm für die Zielfelderkennung: Verarbeitet eine im Dateisystem abgespeicherte Bildserie und gibt die dabei errechneten Abstände vom Zielfeldmittelpunkt auf der Kommandozeile aus.

TargetDetectionDemo.py Weiteres Testprogramm für die Zielfelderkennung: Im Gegensatz zur ImageAnalyzerDemo werden die zu verarbeiteten Bilder direkt über die Raspi-Kamera aufgenommen.

UltrasonicDemo.py Testprogramm für die Ultraschallsensoren: Es wird eine bestimmte Anzahl von Messungen durchgeführt. Der Median des Messergebnis wird auf die Kommandozeile ausgegeben. Der Vorgang wird endlos wiederholt, womit die Messungen beim manuellen Fahren am Seil überprüft werden können.

7.2.2. Eingesetzte Python-Bibliotheken

OpenCV Bildverarbeitungsbibliothek mit Python-Interface (Beyeler, M., 2015)

NumPy Matrizenberechnungen: OpenCV verwendet Matrizen als Datenstruktur für die Bilder (Idris, I., 2015)

RPi.GPIO Ansteuerung von GPIO-Ports: Interaktion mit Ultraschallsensoren und dem Mikroendschalter (*RPi.GPIO 0.6.3*, 2016)

Picamera Ansteuerung der Raspi-Kamera zur Aufnahme von Bildern (*picamera 1.13*, 2017)

PySerial Ansteuerung der seriellen Schnittstelle: Kommunikation per USB-Verbindung zum Arduino (*pyserial 3.4*, 2017)

7.3. Arduino: C

Die Motorensteuerung wurde auf dem Arduino umgesetzt. Der Quellcode zum Projekt findet sich im Anhang (*silisloth-arduino.zip*). Das Projekt ist in folgende Quellcodedateien aufgeteilt:

Sloth_Main.ino Definition und Initialisierung von globalen Variablen und Tasks.

Sloth_Communication.ino Kommunikation mit dem Raspi per serieller Schnittstelle (siehe *Abschnitt 5.2: Raspi – Arduino, Seite 21*)

7. Software

Sloth_SStateMachine.ino Die Programmlogik, gemäss einer State-Machine (siehe *Abbildung 12, Seite 35*)

Sloth_Functions.ino Implementierung diverser Funktionen

Es wurden folgende Arduino-Bibliotheken eingesetzt:

Wire Kommunikation mit I₂C

FreeRTOS Multitasking (Kommunikation und State-Machine)

Adafruit_Motorshield Ansteuerung von Motoren via Motor Shield

Utility PWM

Die State-Machine (siehe *Abbildung 12, Seite 35*) gibt Auskunft über die internen Stati und die Befehle, die intern für die Transitionen verwendet wurden.

7. Software

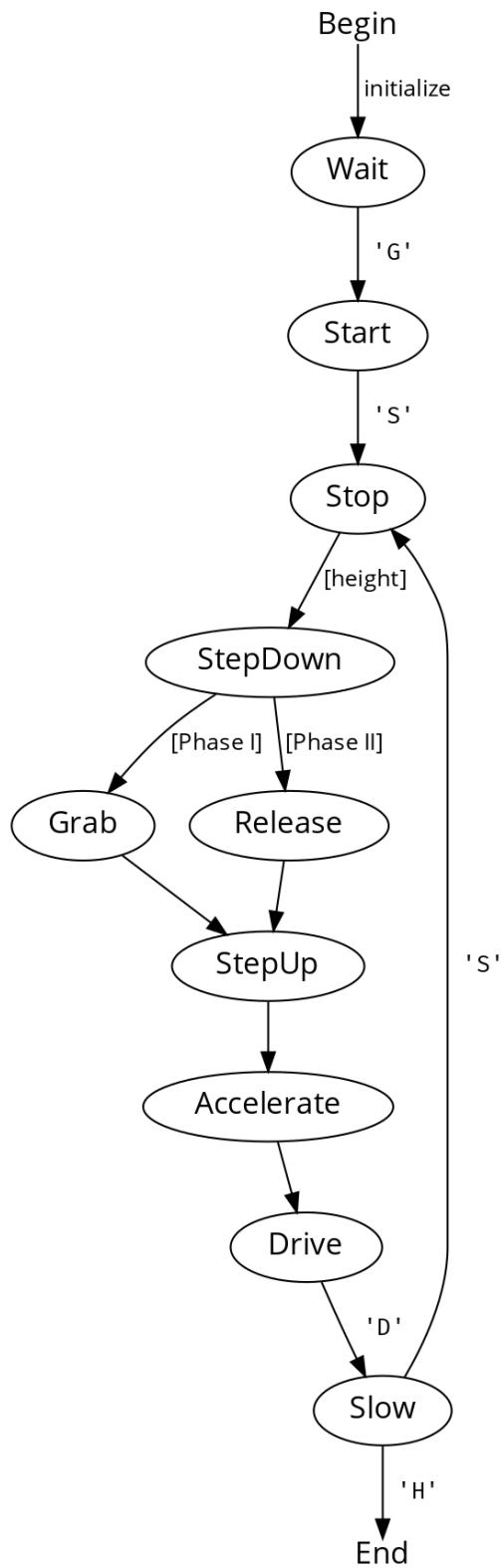


Abbildung 12: Die State-Machine der Arduino-Lösung

8. Zielerreichung

Bei der Arbeit an *Silisloth* mussten – wie die vorliegende Dokumentation zeigt – viele kleine Details beachtet werden. Es bestand die Gefahr sich in diesen Details und den Blick fürs Ganze zu verlieren. Um dies zu verhindern wurden die einzelnen Komponenten möglichst bald am Prototypen montiert, sodass Tests früh an der «Produktivumgebung» stattfinden mussten, bzw. das Aufbauen eines isolierten Testsettings aufwändiger als ein Test am Prototypen gewesen wäre. Dies förderte zudem die Zusammenarbeit innerhalb der Gruppe, da man sich für Tests immer absprechen musste.

8.1. Testphase

Im letzten Drittel des Semesters wurden Tests «am Seil» durchgeführt, wobei das Absolvieren der gestellten Wettbewerbsaufgabe geübt wurde. Diese Tests wurden nicht im Einzelnen protokolliert. Stattdessen ist hier eine Liste mit Erkenntnissen und Verbesserungen aufgeführt, die einen Überblick über den Test- und Lernprozess geben soll:

Antriebsmotor Es stellte sich bald heraus, dass *Silisloth* im Vergleich zu den Prototypen anderer Gruppen sehr langsam fährt. Der zu Beginn verbaute Getriebemotor war eine sehr konservative Wahl: Mit der verbauten Untersetzung hätte er einen weit schwereren Prototypen antreiben können, jedoch nur sehr langsam. Deshalb wurde ein stärkerer Motor verbaut.

Zielfelderkennung Die Zielfeldererkennung schien bei den ersten Tests bereits perfekt zu funktionieren. Bei geänderten (d.h. dunkleren) Lichtverhältnissen und mit auf dem Zielfeld entstandenen Spuren traten aber plötzlich Probleme auf, die mit kleineren Korrekturen nur scheinbar gelöst werden konnten, indem der Schwarz-Weiss-Schwellwert auf die jeweiligen Lichtverhältnisse optimiert wurde. Da Lichtverhältnisse plötzlich ändern können, wurde in der zweitletzten Woche eine zuverlässigere Lösung mit adaptivem Thresholding erarbeitet, was bei allen Lichtverhältnissen zuverlässig funktioniert.

Silikongreifer Der Silikongreifer erforderte einiges an Abstimmungsarbeit. Es wurden verschiedene Versionen mit jeweils kleinen Detailverbesserungen gefertigt, welche sich beim Aufpumpen mehr oder weniger gleichmäßig ausdehnten. Beim Greifen der Last mussten die Höhe der Greifeinheit und die Pumpzeit genau eingestellt werden. Auch die Verbindungen zwischen den Luftschnäbeln mussten mehrmals überprüft und abgedichtet werden.

Netzwerk Zu Beginn der Tests wurden Laptop, Smartphone und Raspi mit dem HSLU-WiFi verbunden. Das Auffinden des Raspi im Netz mittels ARP-Scan dauerte dabei sehr lange. Gegen Ende des Semesters wurde eine eigene Netzwerkumgebung mit dedizierten Geräten (WiFi-Access-Point, Laptop ohne GUI, unpersonalisiertes Smartphone) aufgebaut. Damit konnte sehr

8. Zielerreichung

schnell getestet werden. Der Austausch mit der Arbeitsumgebung wurde dadurch hingegen etwas erschwert (Übertragen aufgenommener Bilder zur Auswertung, Austausch des Python-Quellcodes). Aus diesem Grund kam die «produktive» Netzwerkumgebung erst in den letzten beiden Wochen zum Einsatz.

Vorbereitung An den offiziellen PREN-Terminen (Donnerstag- und Freitagvormittag) musste man oft länger beim Testgelände warten, bis ein Seil frei war. Eine gute Vorbereitung sparte wertvolle Testzeit – und reduzierte die Wartezeit für andere Gruppen. Dazu gehört, dass die Akkus geladen, die Verbindungen zwischen den involvierten Geräten erstellt, die Steuerprogramme auf Raspi und Arduino aktualisiert und genügend helfende Hände vorhanden sind. Dies funktionierte schon bald so gut, dass bereits nach fünf Minuten Testzeit mehrere Durchläufe absolviert werden konnten.

Notausschalter Wurde in einem Testlauf das Zielfeld nicht richtig erkannt, fuhr *Silisloth* unbremst in den Endpfosten, was zu Beschädigungen an einzelnen Komponenten und an der ganzen Konstruktion hätte führen können. Zu Beginn der Tests wurde in einem solchen Notfall der Akku vom Laststromkreis getrennt, was einige Sekunden dauerte und erheblichen Kraftaufwand benötigte. Darum wurden später zwei Schalter – je einen für Last- und Steuerstromkreis – eingebaut, welche bei Bedarf umgelegt werden konnten. So mussten auch vor und nach den Tests keine Kabel mehr ein- bzw. ausgesteckt werden.

8.2. Beurteilung gemäss Anforderungen

Die Anforderungsliste (siehe Ordner Anhang) wurde zu Beginn des Projekts festgelegt. Der im Rahmen von PREN 2 erstellte Prototyp soll kritisch darauf geprüft werden, ob er diesen Anforderungen gerecht wird. Dabei werden die Anforderungen in den ersten drei Spalten aufgeführt; in der vierten Spalte wird die Erfüllung der jeweiligen Anforderung gemäss den Tests eingeschätzt. Auf Wunsch- und Unteranforderungen wird dabei nicht eingegangen.

NR.	BEZEICHNUNG	DETAILS	EINSCHÄTZUNG GEMÄSS KONZEPT
I RAHMENBEDINGUNGEN			
I.1	Aufbau	<i>Silisloth</i> lässt sich in max. zwei Minuten aufbauen.	Zu zweit in ca. 20 Sekunden möglich
I.2	Autonomie	<i>Silisloth</i> und das I/O-Gerät arbeiten nach dem Startsignal autonom.	Nach Startsignal (per Smartphone-App) kein weiteres Eingreifen nötig
I.3	Temperaturbereich	Die Geräte sind in einem Temperaturfenster von 0°C bis 70°C einsatzfähig.	Keine Temperaturprobleme festgestellt, Schrittmotor und Raspi-CPU mit Kühlkörper ausgestattet
I.4	Lichtverhältnisse	<i>Silisloth</i> funktioniert bei 1'000 – 100'000 lux.	Probleme bei unterschiedlichen Lichtverhältnissen behoben

8. Zielerreichung

1.5	Zeitrahmen	<i>Silisloth</i> erledigt ihre Aufgaben innerhalb von vier Minuten.	Gemessene Zeitdauer: 1 Minute, 7 Sekunden
2	DIMENSIONEN		
2.1	Länge	max. 480mm	431mm (CAD-Modell)
2.2	Breite	max. 480mm	204mm (CAD-Modell)
2.3	Höhe	max. 580mm	379mm (CAD-Modell)
2.4	Gewicht	max. 7'000g Leergewicht	4'000g (CAD-Modell, Herstellerangaben)
3	ANTRIEB		
3.1	Höhenüberwindung	<i>Silisloth</i> ist in der Lage eine Steigung von max. 40° zu überwinden.	Höhenüberwindung der Testanlage problemlos (8.1°)
3.2	Einsatzbereich	<i>Silisloth</i> kann auf einem Seil mit Durchmesser 2 – 4mm montiert werden.	Funktioniert mit Seil der Testanlage
3.3	Ziel	<i>Silisloth</i> stoppt nach Berührung des Endpfostens.	Drucktaster meldet Berührung
3.4	Geschwindigkeit	<i>Silisloth</i> bewegt sich durchschnittlich mit mindestens 15 $\frac{mm}{s}$.	20 $\frac{cm}{s}$ möglich
3.5	Fahrtrichtung	<i>Silisloth</i> ist in der Lage sich vorwärts und rückwärts am Seil zu bewegen.	Vorwärts- und Rückwärtstfahren möglich
4	LASTBEFÖRDERUNG		
4.1	Greifen	<i>Silisloth</i> kann eine quaderförmige Last mit den Kantenlängen von mindestens 45mm und höchstens 55mm und einem Gewicht von bis zu 200g greifen.	Tests mit 50mm Kantenlänge und ca. 200g Last erfolgreich
4.2	Heben/Senken	<i>Silisloth</i> kann eine Last von bis zu 200g heben und senken.	Bei Tests unproblematisch
4.3	Abladen	<i>Silisloth</i> kann die Last auf dem Zielfeld absetzen.	Luftventile arbeiten schnell und zuverlässig, Greifer löst sich
5	SENSORIK		
5.1	x-Koordinate	Die x-Koordinate muss mit einer Toleranz von ±20mm bestimmt werden können.	Messgenauigkeit der Ultraschallsensoren: ±20mm(> 100cm), ±2.5%(< 100cm)
5.2	z-Koordinate	Die z-Koordinate muss mit einer Toleranz von ±20mm bestimmt werden können.	Messgenauigkeit der Ultraschallsensoren: ±20mm(> 100cm), ±2.5%(< 100cm)
5.3	Zielerkennung	Das spezifizierte Zielfeld muss mit einer Toleranz von ±20mm erkannt werden können.	optische Erkennung in diesem Rahmen
5.4	Lasterkennung	Die Last muss mit einer Toleranz von ±15mm erkannt werden können.	Funktioniert nach Feinabstimmung
6	KOMMUNIKATION		

8. Zielerreichung

6.1	Startsignal	<i>Silisloth</i> empfängt das Startsignal.	Funktioniert per WiFi
6.2	Koordinaten	<i>Silisloth</i> sendet die Koordinaten an das Ausgabegerät.	Funktioniert per WiFi
7	I/O-GERÄT		
7.1	Startsignal	Das Gerät sendet beim Start das Signal an <i>Silisloth</i> .	Funktioniert per Smartphone-App
7.2	Koordinaten	Das Gerät gibt die x- und z-Koordinaten der Last an.	Die Koordinaten werden in Abständen von 500ms angezeigt
8	AUSNAHMEBEHANDLUNG		
8.1	Lasterkennung	Falls <i>Silisloth</i> den Endpfosten nicht erkennt, soll es durch einen Nothalt beim Berühren des Mastes anhalten.	Drucktaster beendet Ablauf zuverlässig
8.2	Lasterkennung	Falls <i>Silisloth</i> die Last nicht erkennt, fährt es automatisch zum Ziel.	Last wird nicht erkannt sondern abgeschätzt, manuelles Not-Aus nötig
8.3	Zielerkennung	<i>Silisloth</i> lässt die Last beim Endpfosten fallen, falls es das Zielfeld nicht erkennt.	Drucktaster beendet Ablauf endgültig
8.4	Schwingen	<i>Silisloth</i> hält an, falls die Schwingung in y-Richtung 20° überschreitet	Nicht umgesetzt, da Schwingungen in y-Richtung unproblematisch
8.5	Lastverlust	<i>Silisloth</i> erkennt, wenn es seine Last verliert.	Kann nicht erkannt werden

Tabelle 2: Einschätzung der Anforderungserfüllung aufgrund Prototyps

Nach Betrachtung von *Tabelle 2: Beurteilung gemäss Anforderungen, Seite 39* kann unter dem Strich festgestellt werden, dass sich der Prototyp weitgehend im Rahmen der Anforderungen bewegt. Kleinere Abweichungen, wie etwa die Messungenauigkeit der Ultraschallsensoren bei grossen Distanzen, verursachen keine grösseren Probleme. *Silisloth* kann nicht erkennen, ob die Last nicht aufgenommen oder verloren wurde. Wie auf ein solches Ereignis reagiert werden müsste, wurde nicht konzipiert.

Der Wettbewerb wird zeigen, wie gut die erarbeitete Lösung tatsächlich ist.

9. Rückblick

Der Wiedereinstieg nach den kurzen Semesterferien verlief zu Beginn etwas schleppend. Im Bereich Projektmanagement wurde zunächst ein Versuch unternommen mit Jira zu arbeiten, welches den Informatikern aus ihrer Berufspraxis bereits bekannt war. Es stellte sich bald heraus, dass Jira zwar für die Abhandlung von Scrum-Sprints sehr geeignet ist, aber nicht die erwünschten Funktionen für das Projektmanagement bietet. (Ausserdem sind Studentenlizenzen nur für die selbst gehostete Variante von Jira gültig, und die Jira-Lizenzen hätten das Budget bei weitem gesprengt.) So wurde Jira fallengelassen. Das Projekt wurde seither über einfache Aufgabenlisten und mit einem zentralen Projektplan abgewickelt.

Zu Beginn wurde das Projekt in drei Phasen aufgeteilt, wobei zum Ende jeder Phase verschiedene Arten von Tests erfolgreich absolviert werden sollten:

1. Phase: Komponenten Einzelne Komponenten wie Ultraschallensoren, Motoren, Silikongreifer, Luftpumpen etc. wurden isoliert getestet und in ihrer Handhabung verstanden.

2. Phase: Komponentengruppen Die erfolgreich getesteten Komponenten wurden zu verschiedenen Komponentengruppen, wie Greifmechanismus, Bildverarbeitung, Distanzmessung etc., kombiniert und zusammen getestet. Hierbei konnte eine Komponente in mehreren Komponentengruppen vorkommen.

3. Phase: Autonome Laufkatze Sobald alle Komponentengruppen funktionierten, wurde *Silisloth* komplett montiert und getestet, um so die Wettbewerbssituation zu simulieren.

Zwar gab es mit einzelnen Komponenten im späteren Projektverlauf immer wieder vereinzelte Probleme (z.B. mit ungenauen Ultraschallsensoren, defekten Akkus und gerissenen Silikongreifern), ansonsten konnte dieser Plan aber gut eingehalten werden.

Bei der Maschinentechnik gab es gleich von Beginn an grosse Fortschritte zu verzeichnen. Die CAD-Pläne wurden gegenüber PREN 1 noch etwas verbessert und ergänzt. Die mechanischen Teile waren schnell gefertigt und schon nach wenigen Wochen montiert. In der Semestermitte war *Silisloth* bereits komplett montiert und – was die Hardware betrifft – voll einsatzfähig.

Im Bereich Elektrotechnik wurden zu Beginn Versuche mit dem Freedom Board unternommen. Da hier Fortschritte ausblieben, wurde die Entscheidung gefällt auf Arduino umzusteigen. Esstellten sich nun bald erste Erfolge ein, sodass die zu Beginn verlorene Zeit bald aufgeholt werden konnte. Die Schnittstelle zwischen Raspi und Arduino wurde mit der Informatik zusammen definiert und später vereinfacht.

In der Informatik ging es in den ersten Wochen darum ein besseres Verständnis für die Programmiersprache Python 3, die Raspberry-Plattform (v.a. GPIO-Programmierung) und die Bildverarbeitungsbibliothek OpenCV zu erlangen. Mit der bald funktionierenden Zielfelderkennung konnte eines

der Hauptprobleme früh aus dem Weg geräumt werden. Weiter wurde auf Basis von alter Hardware eine komplette Testumgebung bestehend aus Smartphone, Laptop und Wireless-Access-Point eingerichtet, um so an der Wettbewerbssituation nicht auf das HSLU-Netzwerk angewiesen zu sein. In der zweiten Semesterhälfte ging es darum die einzelnen Softwarekomponenten zu kombinieren, wobei ein Grossteil von Programmlogik und Parametern während der Tests verbessert wurde.

9.1. Lessons Learned

Arduino/Freedom Board Sind auch nach mehreren Wochen kaum Fortschritte zu verzeichnen, befindet man sich wohl auf dem falschen Pfad. Hier hilft es mehr umzudenken als weiter dem Holzweg entlang zu schreiten. Der Wechsel vom Freedom Board auf Arduino führte zu schnellen Fortschritten im Bereich Elektrotechnik.

Bildverarbeitung Die Bildverarbeitung war die grösste Hürde im Bereich Informatik. Nach ersten grundlegenden Funktionstests in PREN 1 ist diese Komponente aber recht lange liegen geblieben. Doch schon nach kurzer (wenn auch intensiver) Beschäftigung mit der Bildverarbeitung im ersten Drittel von PREN 2 konnte eine solide Lösung ausgearbeitet werden. Desto länger man eine schwierige Aufgabe herausschiebt, desto schwieriger und bedrohlicher wirkt sie. Darum ist es wichtig, heikle Aufgaben sofort anzugehen, auch wenn ein schneller Erfolg nicht in Aussicht steht. Durch kleine, inkrementelle Verbesserungen verliert die ungelöste Aufgabe ihr Bedrohliches.

Multithreading Python unterstützt die wirklich parallele Ausführung von Programmlogik nur über separate Prozesse, nicht über Threads (Slatkin, 2015, S. 122). Es dauert aber länger einen Prozess aufzustarten als einen Thread. Ausserdem ist die Kommunikation zwischen verschiedenen Prozessen verglichen mit Threads nur mit grossen Einschränkungen möglich. Die Idee, die vier Prozessorkerne des Raspi mit paralleler Ausführung von Programmlogik auszunützen, musste bald verworfen werden. Die Kommunikation vom Raspi zur Smartphone-App bzw. zum Arduino war anfänglich auch mit Threads gelöst. Dies führte zunächst bei den Messungen mit den Ultraschallsensoren zu Problemen, welche später mit fehleranfälligem Synchronisationscode teilweise gelöst wurden. Im weiteren Verlauf wurden Programmlogik und Kommunikationsprotokoll soweit vereinfacht, dass schliesslich ganz auf Threads verzichtet werden konnte. Ein Programmierproblem sollte zunächst so einfach wie möglich gelöst werden. Über die Ausführungsgeschwindigkeit sollte man sich erst dann Gedanken machen, wenn sie sich als ungenügend herausstellt.

Race Conditions Im Gegensatz zum Raspi wurden auf dem Arduino mehrere Threads ausgeführt. Einer kümmerte sich um die Kommunikation mit dem Raspi (Kommunikationsthread), der andere regelte die Motoren (Koordinationsthread). Beim Erteilen des `sxxx;`-Signals zum

9. Rückblick

Herablassen der Greifeinheit traten dabei Probleme auf, da der Arduino den Befehl ignorierte. Es stellte sich heraus, dass der Arduino das Zeichen `s` über den Kommunikationsthread erhielt, der Koordinationsthread aber vor dem Einlesen der Anzahl Millimeter nicht in den entsprechenden Status wechselte, da in der Zwischenzeit kein Kontextwechsel stattfand. Es lag also eine klassische *Race Condition* vor! Das Problem wurde gelöst, indem der Kommunikationsthread nach Erhalt des Zeichens `s` selber den Statuswechsel vornahm, sodass die darauf übertragene Zahl korrekt eingelesen und ausgewertet werden konnte. Bei Threads darf man keine Annahmen über Kontextwechsel treffen!

Entladen von LiPo-Akkus Werden LiPo-Akkus (oder einzelne Batteriezellen) zu stark entladen, sodass die Spannung unter 3V fällt, gilt der Akku als beschädigt und kann nicht mehr aufgeladen werden. Dies ist während Tests einmal geschehen, wonach die Testserie unterbrochen, sowie ein neuer Akku angeschafft werden musste. Dies hat nicht nur Geld sondern auch wertvolle Testzeit gekostet. Deshalb wurde mit dem neuen LiPo-Akku sogleich ein *LiPo-Checker* angeschafft: ein kleines elektronisches Bauteil, das an den LiPo-Akku angeschlossen wird und mit einer grünen und roten LED den Ladestatus jeder Zelle darstellt. Fällt die Spannung bis zu einem kritischen Bereich ab, erzeugt der LiPo-Checker einen ohrenbetäubenden Lärm, sodass man den Akku sofort aussteckt und nicht länger betreibt. Für den kleineren LiPo-Akku, der den Raspi speist, ist das Problem weniger gravierend, da ein Spannungsabfall vorher durch das Herunterfahren des Betriebssystems bemerkt wird. Der Akku sollte dennoch nach den Testläufen von der USV getrennt werden, da eine LED auch Strom verbraucht und so den Akku entlädt.

9.2. Fazit

Die Arbeit an *Silisloth* begann Mitte September 2017 und dauerte bis Juni 2018 an, wobei das Projekt im Januar und Februar 2018 ruhte. Gab es vor neun Monaten nur einen Projektauftrag und eine (anfangs unvollständige) Liste von Gruppenmitgliedern, gibt es jetzt eine voll funktionsfähige autonome Laufkatze namens *Silisloth* und ein Team, das gut zusammengewachsen ist und alle Herausforderungen erfolgreich meistern konnte. Um ein abschliessendes Fazit zu ziehen lohnt sich der Vergleich mit den Zielen, die sich das Team selber im letzten September gegeben hat (Bertozzi et al., 2017, S. 49).

9.2.1. Gruppenziele

Ziel 1 *Wir wollen gut zusammen arbeiten – auch über die Fachbereiche hinweg.*

Die Zusammenarbeit über das ganze Team hinweg hat während des ganzen Projekts gut funktioniert. Es gab viele Diskussionen, aber keinen Streit. Probleme und Meinungsverschiedenheiten konnten unkompliziert gelöst werden.

Ziel 2 *Wir wollen Spass an der Arbeit haben.*

9. Rückblick

Das Projekt war sehr motivierend und hat den Gruppenmitgliedern nicht nur viel Arbeit beschert, sondern auch Freude bereitet. Für Tests wurde auch Freizeit geopfert.

Ziel 3 *Wir wollen etwas lernen – in unserem Fachbereich und fachbereichübergreifend.*

Die Gruppe war in den drei Fachbereichen Maschinentechnik, Elektrotechnik und Informatik gut besetzt, sodass niemand im Fachbereich der anderen eingreifen musste. Wo gegenseitige Hilfe über die Fachgrenzen hinweg möglich war, wurde diese geleistet. Es ist nicht so wichtig, in den für einen fremden Fachbereichen etwas zu leisten, sondern die Perspektive der anderen Fachbereiche einzunehmen, um eine gute Zusammenarbeit zu ermöglichen. Spezialisten dürfen Spezialisten sein, solange sie zu einem Perspektivwechsel fähig sind.

Ziel 4 *Wir wollen, dass jedes Gruppenmitglied seinen Beitrag leistet.*

Bei den sehr vielfältigen Tätigkeiten – Konzeption, Planung, Recherche, Berechnung, Fertigung, Montage, Messung, Beschaffung, Programmierung, Testen, Projektplanung, Dokumentation, Netzwerkverwaltung, Konfiguration usw. usf. – konnte sich jedes Teammitglied einbringen und einen Beitrag leisten.

Ziel 5 *Wir wollen eine sichere, zuverlässige und für alle verständliche Lösung erarbeiten, nicht den Wettbewerb um jeden Preis gewinnen.*

Zwar versteht nicht jedes Gruppenmitglied die Details der gesamten Lösung. Dennoch kann jeder Auskunft darüber geben, welches Problem mit welchen Komponenten gelöst wurde, wie diese grob funktionieren, und die Entscheidungen dahinter nachvollziehen.

Die gefundene Lösung arbeitet zwar langsamer als andere Prototypen, dafür sehr zuverlässig und genau. Was beim Wettbewerb drin liegt, wird sich zeigen.

A. Anhang

A.1. Kostenübersicht

Tabelle 3: Das Kostenbudget für PREN 1 und PREN 2, Seite 45 gibt Auskunft darüber, wie das Budget von CHF 500.00 in PREN 1 und PREN 2 verwendet wurde.

A.2. Thresholding-Benchmarks

Um die verschiedenen Thresholding-Verfahren bei unterschiedlichen Lichtverhältnissen zu erproben, wurden zwei Bildserien verwendet.¹⁷

1. Bildserie

- Kamera: Raspi-Cam
- Auflösung: 480×640
- zu erkennende Zielfelder: 13
- Lichtverhältnisse: eher dunkel, Kunstlicht

2. Bildserie

- Kamera: Smartphone-Kamera
- Auflösung: 1920×2560
- zu erkennende Zielfelder: 18
- Lichtverhältnisse: eher hell, Sonnenlicht

Zu Beginn wurde nur das statische Thresholding-Verfahren implementiert. Beim Simulieren der Wettbewerbssituation kam es dabei immer wieder zu Unregelmässigkeiten: einmal funktionierte die Zielfeldererkennung, einmal nicht. Dieser Umstand konnte auf Verschmutzungen und Beschädigungen am verwendeten Zielfeld zurückgeführt werden. Mit der Anpassung des absoluten Threshold-Wertes konnte das Zielfeld wieder erkannt werden. Diese Korrektur führte jedoch dazu, dass Tests an anderen Tagen und – wie es sich herausstellte – bei anderen Lichtverhältnissen erneut fehlschlugen.

Aus diesem Grund wurde ein adaptives Thresholding-Verfahren erprobt. Dabei wird das Bild nicht anhand eines fixen Wertes in Graustufen umgewandelt – ist ein Pixel heller als x wird er weiss, sonst schwarz. Stattdessen wird die Umgebung eines Pixels auf dessen Helligkeitsverteilung analysiert, und die dabei ermittelte Verteilung als Entscheidungsgrundlage für das Einfärben verwendet. Die Grösse der Umgebung hat sich dabei als kritischer Parameter herausgestellt und wurde mit 1/4 der Bildgrösse

¹⁷Die Helligkeit ist nicht von der Kamera abhängig. Die Bildserien wurden an verschiedenen Tagen aufgenommen, wobei der Raum einmal mit den Storen verdunkelt und das andere mal von Sonnenlicht durchflutet war.

	ARTIKEL	BEREICH	STK.	PREIS	SUMME
PREN ₁	Silikon	Greifeinheit	0.50	71.90	35.95
	Luftpumpe	Greifeinheit	2.00	1.95	3.90
	Raspberry Pi	Steuerung	1.00	39.90	39.90
	Raspi-Cam	Zielfelderkennung	1.00	29.90	29.90
	SD-Karte 16GB	Steuerung	1.00	19.90	19.90
	Arduino Uno	Steuerung	1.00	25.00	25.00
	Motor Shield	Steuerung	1.00	6.95	6.95
	LiPo-Akku 3000mAh/3.7V	Stromversorgung	1.00	19.90	19.90
	Adafruit Power-Boost (USV)	Stromversorgung	1.00	27.90	27.90
	Netzteil	Stromversorgung	1.00	10.00	10.00
ZWISCHENTOTAL					219.30
PREN ₂	Zahnriemen	Hubmechanismus	1.00	6.95	6.95
	Riemenrad klein	Hubmechanismus	1.00	10.95	10.95
	Riemenrad gross	Hubmechanismus	1.00	12.95	12.95
	Angelschnur	Hubmechanismus	1.00	9.90	9.90
	Schrittmotor	Hubmechanismus	1.00	14.05	14.05
	Kugellager	Antrieb	2.00	4.15	8.30
	Getriebemotor 1:50	Antrieb	1.00	29.45	29.45
	Getriebemotor	Antrieb	1.00	24.95	24.95
	Riemenrad klein	Antrieb	2.00	12.95	25.90
	Magnetventil	Greifeinheit	1.00	4.80	4.80
	Spiralkabel	Greifeinheit	1.00	5.80	5.80
	Silikon	Greifeinheit	0.50	71.90	35.95
	Quick-Pins	Aufhängung	2.00	2.50	5.00
	Motor Shield	Steuerung	1.00	18.75	18.75
	DC-Konverter	Schaltung	1.00	5.95	5.95
	USB-Kabel A-Micro-A	Schaltung	1.00	7.45	7.45
	USB-Kabel A-B	Schaltung	1.00	9.45	9.45
	Prüfsummer	Schaltung	1.00	10.95	10.95
	Relais	Schaltung	1.00	3.45	3.45
	Ultraschallsensor	Sensorik	2.00	3.71	7.42
	LiPo-Akku 1300mAh/14.8V	Stromversorgung	1.00	27.45	27.45
	Kleinteile, Diverses	Fertigung	1.00	30.00	30.00
	Holz, Haken, Seil etc.	Gestell	1.00	24.10	24.10
ZWISCHENTOTAL					339.92
GESAMTTOTAL					559.22

Tabelle 3: Das Kostenbudget für PREN₁ und PREN₂

VERFAHREN	1. BILDSERIE			2. BILDSERIE		
	Erkannt	Zeit (Mean)	Zeit (Median)	Erkannt	Zeit (Mean)	Zeit (Median)
Statisch (100)	11/13	35ms	36ms	11/18	354ms	355ms
Statisch (110)	7/13	36ms	36ms	18/18	348ms	346ms
Statisch (120)	1/13	34ms	34ms	18/18	355ms	355ms
Adaptiv (Gauss)	13/13	333ms	334ms	18/18	77499ms	77520ms
Adaptiv (Mean)	13/13	46ms	47ms	18/18	552ms	553ms

Tabelle 4: Die Benchmarking-Ergebnisse der verschiedenen Thresholding-Verfahren

sehr gross gewählt, was sich positiv auf das Erkennungsergebnis, jedoch negativ auf die Performance auswirkt (siehe *Tabelle 4: Die Benchmarking-Ergebnisse der verschiedenen Thresholding-Verfahren, Seite 46* – die Werte 100, 110, 120 bei den statischen Verfahren beziehen sich auf den gewählten absoluten Threshold-Wert im Bereich von 0: schwarz bis 255: weiss).

Aus den Thresholding-Benchmarks¹⁸ wurden folgende Erkenntnisse gewonnen:

- Das statische Verfahren ist performanter als das adaptive.
- Bei den statischen Verfahren konnte kein Threshold gefunden werden, mit dem alle Zielfelder erkannt werden können. Der Threshold müsste an die Lichtverhältnisse angepasst werden.¹⁹ Das statische Verfahren ist somit unsicher.
- Bei den adaptiven Verfahren konnten alle Zielfelder erkannt werden.
- Das Gauss-Verfahren ist um ca. Faktor 7 (geringe Auflösung) bzw. 140 (hohe Auflösung) langsamer als das Mean-Verfahren.²⁰
- Für kleine Auflösungen beträgt die Performance-Einbusse mit dem Adaptiv-Mean-Verfahren gegenüber den statischen Verfahren ca. 30%. Diese Performance ist immer noch wettbewerbsfähig.

FAZIT: Das Adaptiv-Mean-Verfahren (in Kombination mit einer Auflösung von 480×640 Pixeln) kommt zum Einsatz. Dieses ist zwar etwas langsamer als das statische Verfahren, arbeitet aber bei allen erprobten Lichtverhältnissen fehlerfrei.²¹

¹⁸Die Benchmarks wurden auf einem Raspberry Pi 3 Modell B durchgeführt.

¹⁹Bei Sonneneinstrahlung ändern sich die Lichtverhältnisse in Sekunden, wenn eine Wolke die Sonne verdeckt.

²⁰Die Laufzeit wächst etwas schneller als linear zur Auflösung.

²¹Ein fehlerhafter Algorithmus kann mit beliebig hoher Performance implementiert werden, sogar mit $O(0)$.

A.3. Weitere

Im Verzeichnis Anhang befinden sich folgende Anhänge:

Dokumentation_PREN-1.pdf Das *Silisloth*-Konzept aus PREN 1, auf welches in der vorliegenden Arbeit oft verwiesen wurde.

Anforderungsliste_PREN-1.pdf Die Anforderungen aus PREN 1, welchen *Silisloth* genügen sollte.

Assembly-Zeichnungen.pdf Das CAD-Modell für *Silisloth* aus verschiedenen Perspektiven.

silisloth-app.zip Der Quellcode zur Smartphone-App (Java).

silisloth-raspi.zip Der Quellcode zum Raspi-Projekt (Python).

silisloth-arduino.zip Der Quellcode zum Arduino-Projekt (C).

Quellen

- Adafruit Motor/Stepper/Servo Shield for Arduino v2 Kit.* (o.J.). <https://www.adafruit.com/product/1438>.
- Arduino Uno Rev3.* (o.J.). <https://store.arduino.cc/arduino-uno-rev3>.
- Bertozzi, S., Binkert, C., Bucher, P., Duong, A., Frei, Q., Greber, J., ... Togan, J. (2017). *Silisloth*.
- Beyeler, M. (2015). *OpenCV with Python Blueprints*. Packt Publishing.
- Camera Module.* (2016). <https://www.raspberrypi.org/documentation/hardware/camera/README.md>. (Camera Module v2)
- Idris, I. (2015). *NumPy*. Packt Publishing.
- LiPo-Summer Pichler.* (o.J.). <https://www.conrad.ch/de/lipo-summer-pichler-l-x-b-x-h-35-x-25-x-10-mm-1435735.html>.
- LM2596 S DC-DC Adjustable Step Down Buck Converter.* (o.J.). <https://hobby-elektronik.ch/Module-Sensoren/Stromversorgung/Step-Down-Abwaertswandler/LM2596-S-DC-DC-Adjustable-Step-Down-Buck-Converter::24.html>.
- picamera 1.13.* (2017). <https://pypi.org/project/picamera/>.
- pyserial 3.4.* (2017). <https://pypi.org/project/pyserial/>.
- RC-Akku LiPo 1300 mAh.* (o.J.). <https://www.brack.ch/tattu-rc-akku-lipo-1300-mah-366107>.
- RPi.GPIO 0.6.3.* (2016). <https://pypi.org/project/RPi.GPIO/>.
- Single Relay Board.* (o.J.). <https://www.parallax.com/product/27115>.
- Slatkin, B. (2015). *Effective Python*. Addison-Wesley.
- U.S. Solid Mini 12V DC pneumatische Magnetventil 2 Weg Luft Ventil.* (o.J.). <https://www.amazon.de/dp/B01MR60WQK>.
- Wittel, H., Jannasch, D. & Voßiek, J. (2016). *Roloff/Matek Maschinenelemente Formelsammlung* (13. Aufl.). Springer.

Abbildungsverzeichnis

I.	Gruppenfoto (von links nach rechts): Alex Duong, Christoph Binkert, Sandro Ber tozzi, Quentin Frei (vorne), Patrick Bucher (hinten), Jan Greber, Marko Lovrinovic (vorne), Johannes Togan (hinten)	2
2.	Die Explosionsansicht des CAD-Modells von <i>Silisloth</i> mit nummerierten Kompo nenten	9
3.	Die Berechnung des Wellenabstands e (Wittel et al., 2016, S. 206)	12
4.	Printed Circuit Board für die Ultraschallsensoren, die am Raspi angeschlossen sind . .	14
5.	Das Blockschaltbild der Elektronik-Komponenten, aufgeteilt in einen Steuerstrom kreis (rot) und einen Laststromkreis (blau). Der Arduino ist das Verbindungsglied zwischen diesen beiden Stromkreisen.	15
6.	Die Berechnung der Enddistanz (Position des Lastabwurfs): Der Ultraschallsensor muss von Position U_A nach U_E gebracht werden, sodass er eine Distanz von $E_{\overline{XP}}$ bis zum Endpfosten misst, und sich die Last über der Mitte des Zielfelds befindet. . .	19
7.	Die Verbindungen zwischen den vier involvierten Computern	20
8.	Die Kommunikation zwischen Benutzer, Smartphone, Raspi und Arduino	23
9.	Die Bildserie zeigt das Überfahren des Zielfeldes aus Kameraperspektive.	28
10.	Die Smartphone-App zur Steuerung von <i>Silisloth</i> und zum Ablesen der Koordinaten	30
II.	Das Klassendiagramm des Python-Projektes	32
12.	Die State-Machine der Arduino-Lösung	35

Tabellenverzeichnis

1.	Die Befehle für das Kommunikationsprotokoll zwischen Raspi und Arduino	21
2.	Einschätzung der Anforderungserfüllung aufgrund Prototyps	39
3.	Das Kostenbudget für PREN 1 und PREN 2	45
4.	Die Benchmarking-Ergebnisse der verschiedenen Thresholding-Verfahren	46