Studentische Arbeit SA-26

Modellierung, Simulation, Optimierung und Bau eines Quadrocopters

von
Hannes Pfitzner, Joachim Hecker,
Sebastian Bobrzyk, Jakob Englert

Betreuer: Prof. Dr.-Ing. Prof. E.h. P. Eberhard
M.Sc. W. Luo

Universität Stuttgart
Institut für Technische und Numerische Mechanik
Prof. Dr.-Ing. Prof. E.h. P. Eberhard

Juni 2020

Inhaltsverzeichnis

Kapitel 1

Einleitung

In Zeiten wo manchmal jede Sekunde zählt und der Verkehrsfluss auf den Straßen nicht immer garantiert ist, ist der schnelle Luftweg mit einer Drohne oftmals die bessere Option. Ein Automatisierter Transport über den Luftverkehr könnte die Lebensqualität in vielen Bereichen verbessern und dabei Kosten und Umweltschadstoffe, durch die ersparte Fahrt, verhindern. Die kürzlich rasant wachsende Covid-19 Epidemie zeigt deutlich, dass im medizinischen Bereich enorme Schwachstellen in unserer heutigen Infrastruktur zu finden sind. Während es auf der Nordhalbkugel vor allem die Krankenhäuser sind, welche durch die pure Masse an erkrankten Patienten keine weiteren Kapazitäten mehr haben, sind es in Ländern Afrikas die Wege, welche teilweise stundenlang bis zur nächsten ärztlichen Versorgung sind.

Ein schon laufendes Projekt des Technologieunternehmens „Zipline“ nutzt mit ihren Drohnen den Luftraum, um in Afrika, genauer gesagt in Rwanda und Ghana, Medikamente sowie gespendetes Blut innerhalb von Minuten an Notfallorte zu bringen. [1]

Natürlich stellt sich da die Frage über die geltenden Gesetze im Luftraum. Diese stellen in den USA und Europa immer noch Grenzen den Drohnen gegenüber und schränken dadurch kommende Innovationen in ihrer Realisierbarkeit ein. Afrika setzt jedoch den Fokus auf Innovation und vielversprechende Technik, weshalb sie dieses Projekt fördern.[2]. Doch auch in der EU sind, wegen den immer noch vielen offenen Fragen, Änderungen im Gesetzbuch zu erwarten.[3]

[1] Für einen genaueren Einblick: https://flyzipline.com/ (23.05.2020)

[2] Ein Forbes Artikel darüber: https://www.forbes.com/sites/andrewcharlton5/2020/05/06/when-it-comes-to-sensible-drone-policy-africa-leads-the-way/ (23.05.2020)

[3] https://www.drohnen.de/20336/drohnen-gesetze-eu/ (23.05.2020)

Vor diesem Hintergrund geht es bei dieser Projektarbeit um die Entwicklung einer Paketgreiffunktion für einen bereits funktionierenden Quadrokopter. Die zu entwickelnde Hardware lässt sich dabei in vier Kategorien einteilen:

Ein Greifarm, der in der Lage ist, Pakete mit einer Größe von ca. 10 cm zu greifen und zu halten; eine Kamera, die das zu greifende Paket erkennt; diverse Sensorik und Aktorik für den Greifarm und anderweitige Kontrollfunktionen; ein System-on-a-Chip (SoC) für die Bildanalyse, Regelung und Steuerung des Greifarms und des Quadrokopters, wobei dieser selbst bereits mit einem weiteren SoC ausgestattet ist, mit welchem für die Steuerung nur kommuniziert werden muss.

In Kapitel 2 wird zunächst auf die mechanische Auslegung und Konstruktion des Greifarms eingegangen. Anschließend wird der Greifer in Kapitel 3 mit Sensoren und Aktoren ausgestattet, welche dann in Kapitel 4 mit Software zum Leben erweckt werden. In Kapitel 5 mit einer Einführung in den Bilderkennungsalgorithmus der letzte Baustein geliefert, der dann in Kapitel 6 mit den anderen Komponenten zu einem Gesamtsystem konsolidiert wird.

Kapitel 2

Greifermechanik

2.1 Einleitung zur Mechanik

Ziel der konstruktiven Arbeit am und mit dem Quadrokopter ist ein modular zu montierender Greifarm, welcher mit Kameras und Sensoren sein Ziel selbst finden kann. Dabei ist die Mechanik des Greifarms der Kern der Arbeit und beginnt an der Verbindungsstelle mit dem Motor und endet mit der letztendlichen Kontaktstelle zur Last, welche transportiert werden soll.

Dabei sollte der Fokus der konstruktiven Ausarbeitung eine stabile, funktionssichere, leichte und dauerfeste Konstruktion sein, welche selbst bei nicht idealen Bedingungen ihre Aufgaben erfüllt und dabei insbesondere das Wohlergehen von Passanten nicht gefährdet. Beim Aspekt der Stabilität und Sicherheit ist insbesondere die Greifsicherung im Falle eines Defekts und die Lage des Schwerpunkts zu beachten. Gleichzeitig darf ein maximales Gewicht von 2kg nicht überschritten werden, da sonst ein spezieller Drohnenführerschein als Nachweis zum Bedienen des Quadrokopters nötig ist.

Für die Konstruktion steht als Grundmaterial für den Rahmen Plexiglas zur Verfügung, welches mit Laserschneidemaschinen in Formen geschnitten werden kann. Weitere benötigte Bauteile wie zum Beispiel ein Motor oder Lagerungen gelten als Zukaufteile.

Die tiefere Forschungsfrage hinter dem Projekt ist die Visualisierung des Greifarms im späteren Verlauf, welche Möglichkeiten einem bei der Konstruktion, beim Leichtbau und der Stabilität mit dem vorausgesetzten Material gegeben sind und wie gut das Konzept umsetzbar ist. Bereits heute gibt es schon erste funktionierende Beispiele, welche durch zum Beispiel große Paketlieferanten und ähnlichen Unternehmen ins Leben gerufen worden sind.

Für die Auswahl eines Konzepts wird zwischen mehreren Prinzipskizzen eine kategorisch ausgewählt und vollständig dimensioniert. Dann wird nochmals die Realisierung geprüft mit einem CAD Modell bzw. Simulation.

2.2 Materialien und Vorgaben

Wie schon in der Einleitung erwähnt, müssen bestimmte Vorgaben eingehalten werden. Die Greifarmmechanik nimmt dabei, wegen ihres großen Bauraumes und im Verhältnis größten Massenanteil, einen sehr großen Teil an Einschränkungen ein.

2.2.1 Gesetzeslage

So liegt die offizielle Gewichtsgrenze für die Nutzung einer Drohne bei 2kg und das Eigengewicht des zur Verfügung gestellten Quadrokopters liegt bei ca. 1,5 kg womit man eine ungefähre Gewichtsgrenze für die Greifarmmechanik plus Last von 500 g hat. So gilt seit dem 07.04.2017 nach Bundesgesetzblatt in Deutschland „...für Besitzer von Drohnen oder Modellflugzeugen mit einem Gewicht von mehr als 2,0 Kilogramm...Darüber hinaus müssen sie besondere Kenntnisse nachweisen. Der Nachweis wird entweder nach Prüfung durch eine vom Luftfahrt-Bundesamt anerkannte Stelle erteilt oder bei Modellflugzeugen durch einen Luftsportverband nach einer Einweisung ausgestellt.“ [**?**]. Dabei ist zu beachten das jedes eingesparte Gramm an Gewicht, mehr Laufzeit und demnach mehr Effizienz pro Akkuladung mit sich bringt bzw. umgekehrt eine höhere tragbare Last zur Folge hat.

2.2.2 Materialvorgabe

Um in dieser geringen Gewichtsklasse zu bleiben, wird ein stabiler und gleichzeitig leichter Werkstoff benötigt, welcher zugleich auch keine Unmengen an Kosten mit sich bringt, damit auch der wirtschaftliche Aspekt im Laufe verschiedener Experimente und Testdurchläufe nicht komplett ausgereizt wird. Das Institut bietet dafür Plexiglasplatten an, welche in verschiedenen Wandstärken (3 mm und 6 mm) angeboten werden und zugleich mit einer Laserschneidemaschine in verschiedenste Formen geschnitten werden können. Dabei bietet Plexiglas eine brauchbare Steifigkeit für leichte Anwendungsfälle, sowie mit einer Dichte von 1,18 g/cm^3 [**?**] ein geringes Gewicht, wobei es keine enormen Kosten mit sich bringt.

Neben Kunststoffkleber bietet das Institut außerdem, für form- und kraftschlüssige Verbindungen, Schrauben und Muttern in verschiedensten Längen und Durchmessern an. Da es sich hier jedoch um Metall- und keine Kunststoffschrauben handelt, ist während der konstruktiven Auslegung der Greifarmmechanik auf eine geringe Stückzahl zu achten, um ein zu hohes Gewicht zu verhindern.

2.3 Aufgabe und Konzeptideen für die Greifarmmechanik

Die grobe Hauptaufgabe der Greifarmmechanik besteht im Aufnehmen und Ablegen einer Last in Form eines kleinen Paketes, welches mit dem Quadrokopter von einem Ort zum anderen Ort transportiert wird. Dabei sollte eine gewisse Stabilität gegeben sein, sowie eine realistische Sicherheit.

Im Folgenden werden dementsprechend drei Konzeptideen vorgestellt und dessen Vor- und Nachteile gegenübergestellt, wobei dann ein Konzept kategorisch festgelegt wird.

2.3.1 1. Konzept

Die Funktionsweise des ersten Konzepts aus Abbildung 2.1 basiert auf einen Motor, der mittels Schnur entgegen einer Feder eine Art Scherensystem öffnet, an denen sich Greifarme befinden, welche die Last halten sollen. Durch die Federkraft soll die benötigte Haftkraft über die Greifarme an die Last geleitet werden, welche ein Abrutschen verhindern soll.

Das Konzept zeichnet sich mit einem relativ simplen Aufbau aus, welcher einen geringen Bauraum einnimmt. Durch die Federkraft wird ein sicherer Halt selbst bei Defekten an der Elektronik gewährt. Die mögliche, aber nicht notwendige, parallele Bauweise kann weitere Stabilität hinzufügen. Durch die relativ steifen Greifarme, ist eine erhöhte Präzision beim Ansteuern der Last erfordert. Auch die Rutschfestigkeit bei den Greifarmen ist ungewiss und muss experimentell überprüft werden. Genauso ist die Federdimensionierung und, mit dessen Lebensdauer, die gebotene Haftkraft unsicher.

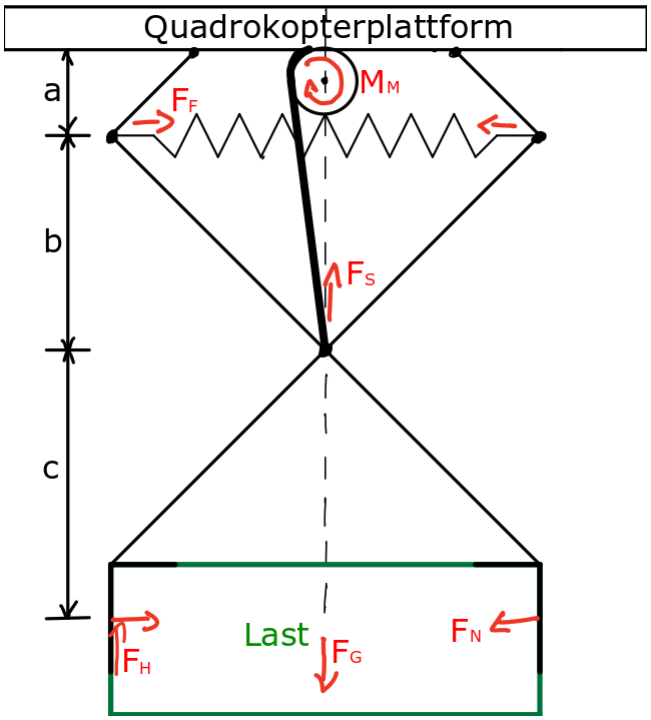

Abbildung 2.1: Erste Prinzipskizze

2.3.2 2. Konzept

Bei der Funktionsweise des zweiten Konzepts aus Abbildung 2.2 übernimmt ein elektrischer Hubzylinder die Arbeit. Dieser öffnet und schließt Greifarme und hält diese in Position mit seiner eigenen Steifigkeit.

Auch hinter dem zweiten Konzept steckt ein simples Grundprinzip. Vorteil des Hubzylinders wäre die Steifigkeit, die ohne elektrisches Signal nicht schwindet. Auch ist hier eine parallele Bauweise für mehr Stabilität möglich. Durch den einfach geregelten Hubzylinder, lässt sich die Greifarmbreite beliebig einstellen, wodurch verschieden große Pakete aufgenommen werden können.

Jedoch wird trotzdem eine gewisse Präzision beim Ansteuern des Paketes benötigt. Auch ist die Rutschfestigkeit wie beim ersten Konzept nicht gewiss. Zusätzlich wird durch die horizontale Bauweise des Hubzylinders ein großer Bauraum eingenommen.

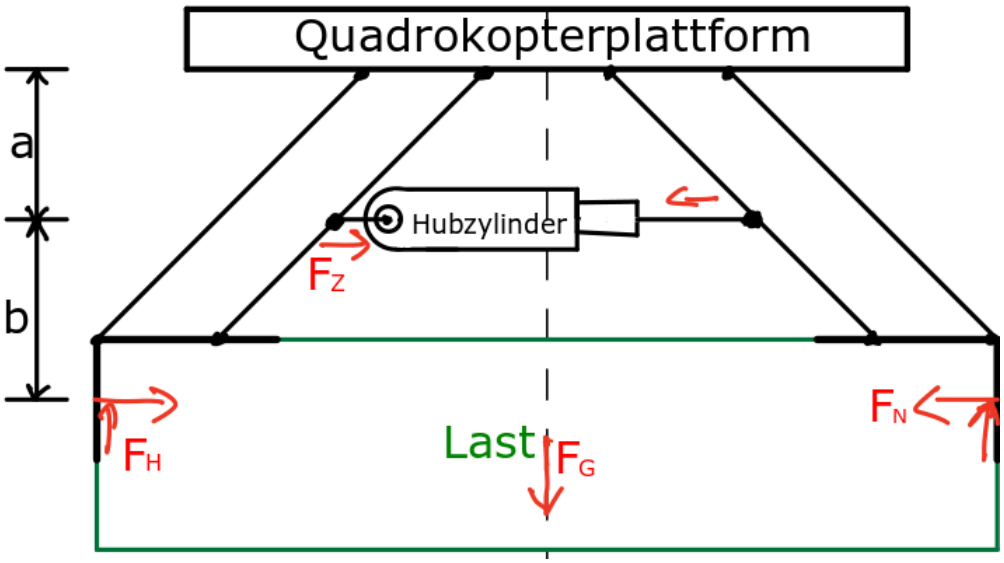

Abbildung 2.2: Zweite Prinzipskizze

2.3.3 3. Konzept

Hinter der Funktionsweise des dritten und letzten Konzepts aus Abbildung 2.3 verbirgt sich ein Knickgelenk an dessen Ende ein Magnet ist. Dieses Knickgelenk wird mittels zwei Motoren ausgefahren.

Das letzte Konzept ist zwar, durch den zweiten integrierten Motor, etwas komplexer, jedoch entfällt durch den Magneten die Reibungskomponente, was eine erhöhte Zuverlässigkeit bietet. Auch wird dadurch das Ansteuern des Paketes vereinfacht. Durch das Knickgelenk, kann der Schwerpunkt des gesamten Systems sehr gut mittig gehalten werden. Jedoch wird die Gesamtmasse durch den zweiten Motor deutlich erhöht, was auch die maximale Paketmasse niedrig hält. Auch ist die letztendliche Kamera- und Sensorpostion nicht wirklich zentrierbar durch die Lage des Knickgelenks

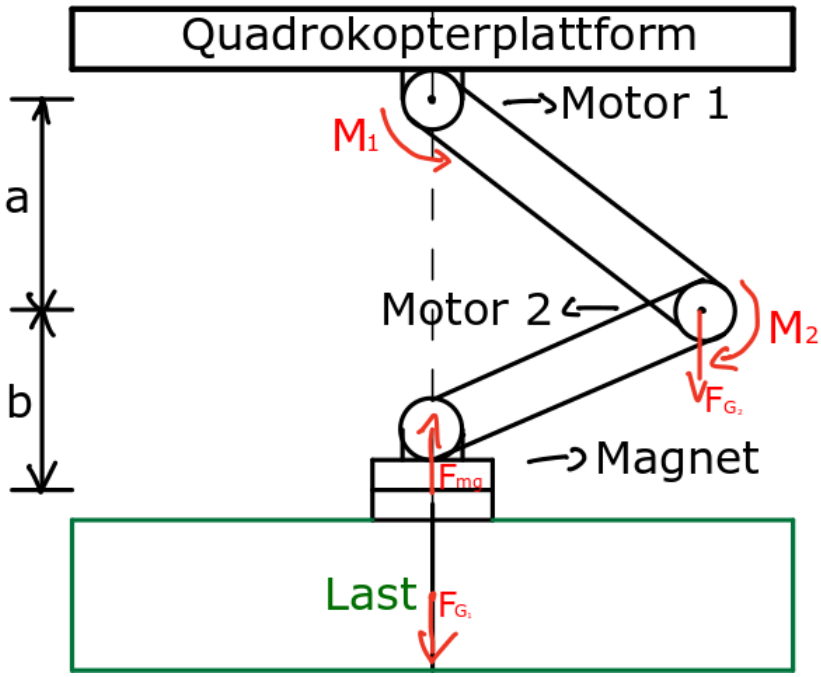

Abbildung 2.3: Dritte Prinzipskizze

2.3.4 Konzeptauswahl

Für ein letztendliches Fazit werden Vor- und Nachteile der vorgestellten Konzepte verglichen und abgewogen, um ein finales Konzept auszuwählen.

Dabei überzeugt das zweite Konzept aus Abbildung 2.2 am meisten, da der Hubzylinder einen stabilisierenden Faktor ins System mit einbringt und sich nur in der Länge verändert, wenn das dafür geeignete Regelgerät das elektrische Signal dafür bekommt. Bei dem ersten Konzept aus Abbildung 2.1 wäre die konstant benötigte Federkraft zu anfällig auf die anhäufenden Öffnungszyklen. Außerdem sind die simple Bauweise und gleichzeitig stabile Konstruktion des zweiten Konzepts weitere Faktoren, welche für das Konzept sprechen. Gleichzeitig ist ein größerer Spielraum bei der Paketmasse auch ein überzeugender Aspekt, welcher gegen das dritte Konzept aus Abbildung 2.3 spricht.

Der Hauptfokus liegt demnach im Bauraum und der Länge des elektrischen Hubzylinders, wobei zweiteres als Zukaufteil nicht beliebig variabel ist. Ebenfalls ist die Leistung des Hubzylinders zu beachten, da er bei einem Material wie Plexiglas durchaus in der Lage ist, dieses, bei einem Fehler des Systems, zu zerstören.

2.4 Berechnung und Vordimensionierung

Bei der Berechnung und Vordimensionierung der Greifarmmechanik liegt der Hauptfokus auf dem Bauraum, dem Gewicht des Systems und der letztendlichen benötigten Kraft, welche von dem elektrischen Hubzylinder geleistet werden soll, um das Gewicht fest im Griff zu haben. Ersteres wird durch die Höhe und Lage der Standbeine des Quadrokopters begrenzt. Grobe Messungen ergaben einen Bauraum von $80 \times 80 \times 130mm^3$. Dieser lässt sich jedoch durch genaue Formanpassung der Plexiglasteile in die Länge erweitern.

2.4.1 Benötigte Normalkraft

Die letztendlich wichtige Kraft ist die wirkende Hubzylinderkraft, welche in Form der Normalkraft an der Greiffläche wirkt. Letztere lässt sich mittels Reibgesetze berechnen. Da die Greifebene zum tragenden Objekt eben ist, ist die Greifebene zur Horizontalen senkrecht. Die Abbildung 2.4 verdeutlicht die Verteilung der Kräfte. Aus der Abbildung 2.4 ergibt sich, dass die letztendlich benötigte Nor-

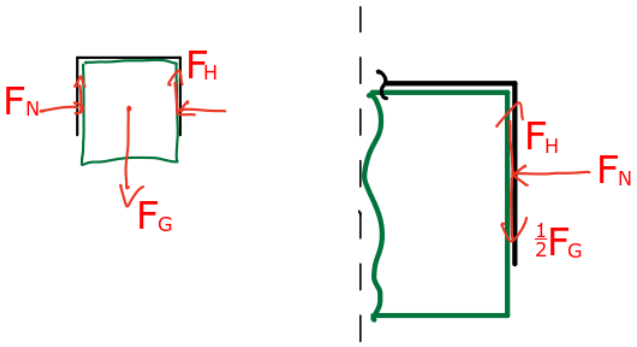

Abbildung 2.4: Kraftverteilung an der Greiferfläche

malkraft abhängig von der Gewichtskraft und dem Haftreibwert ist. Da sich die Reibwerte zu jeweiligen Gleitebenen verschiedener Materialien nur experimentell erörtern lassen, muss ein Mittelwert genommen werden, welcher nach den ersten Experimenten eventuell zu korrigieren ist. Als Oberflächen werden Gummi und Plexiglas genommen. Nach längerer Recherche setzt sich ein ungefährer Mittelwert von $\mu_H = 0,3$ durch.[1]

[1] Quelle für Haftreibungszahlen: https://www.chemie.de/lexikon/Haftreibung.html (23.05.2020)

Für die zu ermittelnde Gewichtskraft ist von einem Gewicht von $m = 50g$ bis $m = 100g$ für die Last zu schätzen. Um eine gute Fehlertoleranz und damit Sicherheit zu bieten, wird der errechnete Wert mit dem Faktor $S = 1,8$ angepasst. Mit dem gegebenen Haftreibwert und der Erdbeschleunigung $g = 9,81\frac{m}{s^2}$ auf die Last, lässt sich nun die nötige Haftkraft in Form der Normalkraft berechnen:

$$\begin{aligned}
\frac{1}{2}F_G &= F_H \\
F_G &= mg \\
F_H &= F_N \times \mu_H \\
mg &= 2F_N \times \mu_H \\
F_N &= \frac{mg}{2\mu_H} \\
&= \frac{0,1kg \times 9,81 \times \frac{m}{s^2}}{2 \times 0,3} \\
&= 1,635N \\
F_{Nmin} &= F_N \times S \\
&= 1,635N \times 1,8 \\
&= 2,943N
\end{aligned} \tag{2.1}$$

Aus der Berechnung (2.1) ergibt sich also eine ungefähr benötigte Normalkraft von $F_{Nmin} = 3N$.

2.4.2 Benötigte Hubzylinderkraft

Die schlussendlich benötigte Hubzylinderkraft lässt sich über einfache Hebelgesetze mit der Normalkraft schlussfolgern. Anhand der Abbildung 2.5 und den darauf rotierenden angedeuteten Hebelarmen, erkennt man, dass die maximale Hebellänge in senkrechter Stellung erreicht ist. Die Wahl des Hebelverhältnisses zwischen a und b ist hier entscheidend. Denn Bauraum und Maße der in Frage kommenden Hubzylinder setzen hier starke Grenzen. Für die grobe Vordimensionierung wird das Verhältnis von a zu b auf $1 : 1$ gesetzt.

$$\begin{aligned}
F_N \times (a + b) &= F_Z \times a \\
F_Z &= F_N \times (1 + \frac{b}{a}) \\
2F_N &= 6N
\end{aligned} \tag{2.2}$$

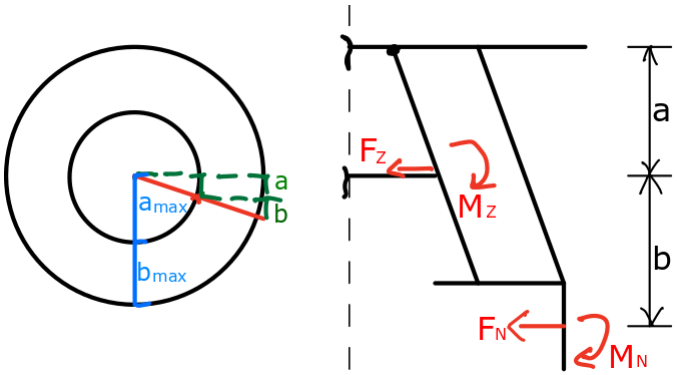

Abbildung 2.5: Kraftverteilung an den Hebelarmen

2.4.3 Auswahl des Hubzylinders

Mit der in (2.2) berechneten Hubzylinderkraft von $F_Z = 6N$ fällt die Wahl des elektrischen Hubzylinders auf den „Titan 30“ von CTI-Modellbau [1], welcher mit dem „Thor4HF Titan 1 Regler“, ebenfalls von CTI-Modellbau [2], betrieben wird. Durch den begrenzten Bauraum bieten sich nicht viele Alternativen an, da die Firma CTI-Modellbau sich explizit auf den Modellbau konzentriert. Mit einer erwünschten Öffnungsbreite von jeweils 30 mm pro Seite und einem Faktor von 2, durch die mittige Position des Hubzylinders, lässt sich eine erwünschte Hublänge von 30 mm errechnen.

2.5 Vom Modell zu den ersten Bauteilen

2.5.1 Creo Modell

Mit den ersten vordimensionierten Bauteilen und den ersten festen Maßen von den Zukaufteilen, lässt sich nun ein erstes 3D CAD-Modell bauen. Dabei werden je nach Absprache und Änderungen von Eigenschaften Anpassungen unternommen, um dem finalen Produkt so nah wie möglich zu ähneln. Aus dem CAD Modell aus Abbildung 2.6, modelliert mit "Creo Parametrics5", lässt sich schon die grobe Gestalt der Greifarmmechanik erkennen.

[1] Titanzylinder: https://www.cti-modellbau.de/-74-102-110-170-185-328-532.html (23.05.2020)
[2] Titanregler: https://www.cti-modellbau.de/-74-102-110-181-182-191-329-350-543.html (23.05.2020)

Ausgehend von der Nummerierung in Abbildung 2.6 erkennt man die Bauteile Bodenplatte(1), Hebelarm(2), Führungsschiene(3) und Greifplatte (4). Sie bilden die Grundbausteine des Endprodukts der Greifarmmechanik. Im Zwischenraum von Bodenplatte und Hebelarmen soll später der Hubzylinder(5) Platz finden. Bauteile wie zum Beispiel Verbindungsschrauben oder Abstandshülsen sind außen vor und werden erstmal nicht in der Auflistung berücksichtigt, da diese sich in der Anzahl und der Position noch stark ändern können.

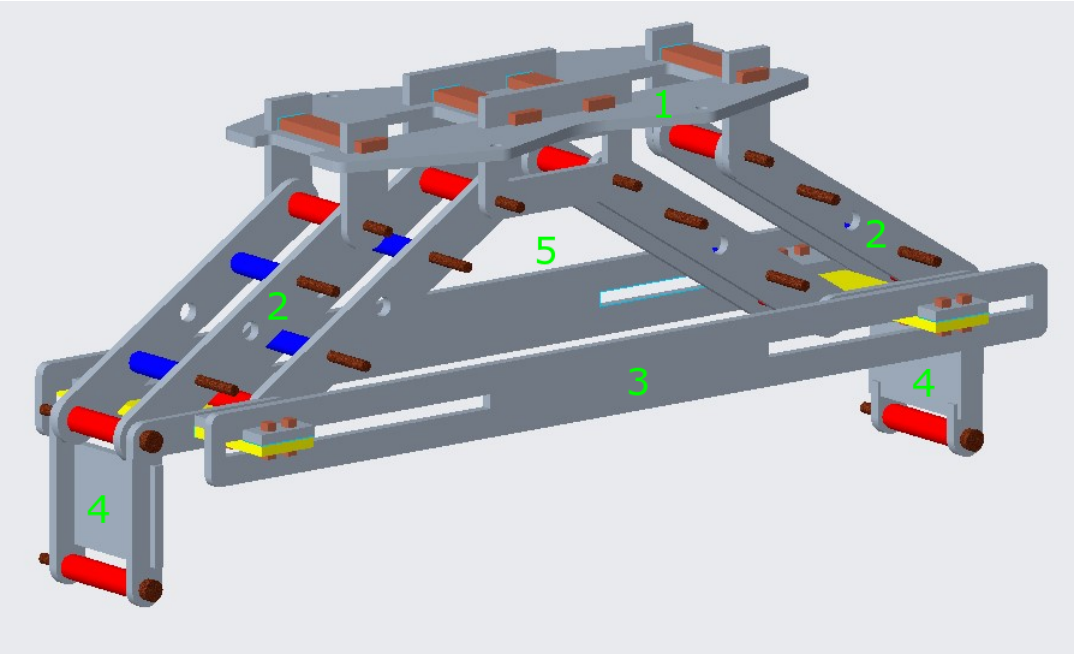

Abbildung 2.6: Erstes CAD-Modelle

2.5.2 Inkscape

Die letztendlich fertigen dreidimensionalen CAD-Bauteile können nun mit „Inkscape“, eine Software zur Bearbeitung und Erstellung zweidimensionaler Vektorgrafiken, zu Schnittmustern erzeugt werden. Mithilfe dieser Schnittmuster werden die fertigen Bauteile zweidimensional aus den Plexiglasplatten mit konstanter Wandstärke von der Laserschneidemaschine ausgeschnitten. Dementsprechend ist bei der Konstruktion der Bauteile zu beachten, dass diese zweidimensional als Schnittmuster abgebildet werden können.

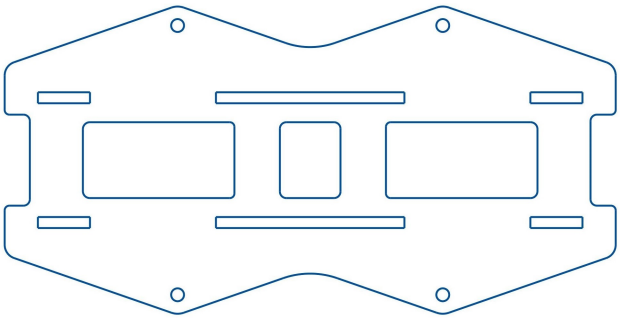

Abbildung 2.7: Inkscapeschnittmuster der neuen Bodenplatte

2.5.3 Erste Ergebnisse

Die ausgeschnittenen Bauteile werden zur Probe vormontiert, um mögliche Problemstellen ausfindig zu machen. Die Abbildungen 2.8 und 2.9 zeigen erste Montageversuche.

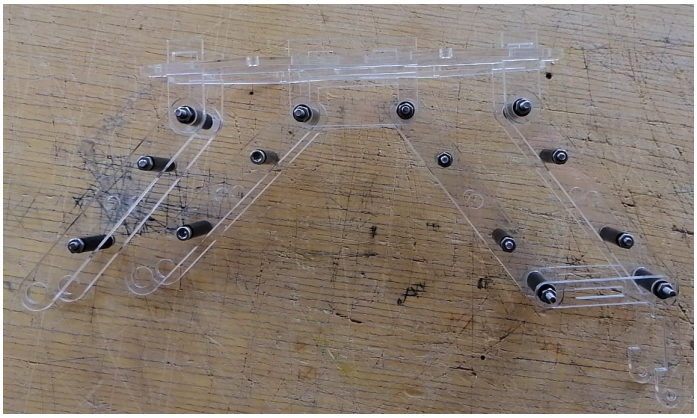

Abbildung 2.8: Zusammengebaute Hebelarme

Um die aus den Berechnungen gewünschte Reibung zu erlangen wird auf den Flächen der Greiferplatten (Nummer 4 in Abbildung 2.6) noch eine Gummischicht draufgeklebt. Nach der letztendlich fertigen Konstruktion muss für die Sensoren noch eine Befestigung hinzugefügt werden. Diese ist aber letztendlich stark vom eingenommenen Volumen des Greifsystems im Bauraum abhängig, da sie bestenfalls zentriert liegt und wird erst nach den ersten Testläufen beigefügt

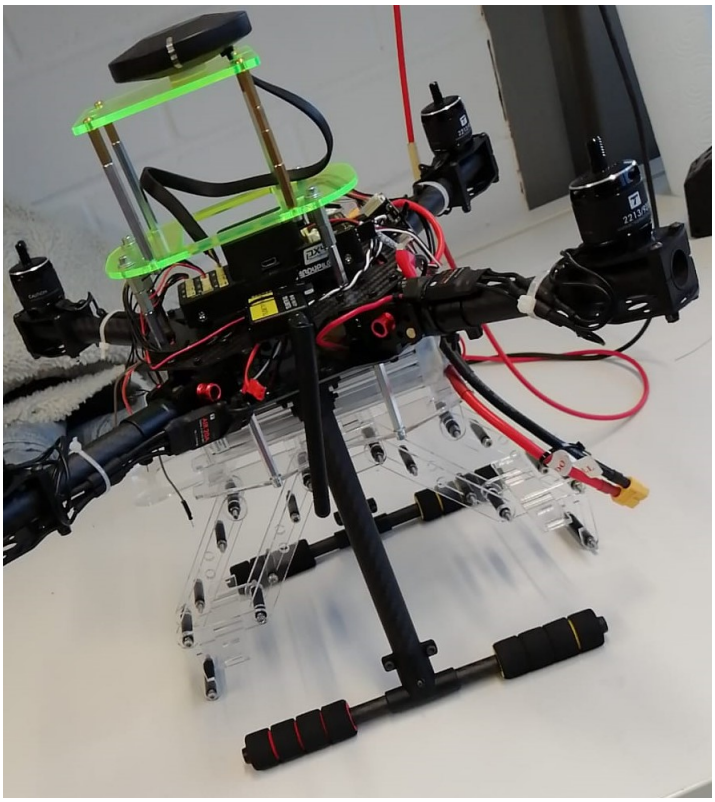

Abbildung 2.9: Vormontage der Greifarmmechanik am Quadrokopter

2.6 Problematiken und Lösungen

Bei den ersten Testläufen ist vermehrt aufgefallen, dass die Wandstärken mit dem CAD Modell schlecht einschätzbar sind und demnach lange und dünne Bauteile wie zum Beispiel die Führungsschienen (Nummer 3 in Abbildung 2.6) so umkonstruiert werden müssen, damit die gewollte Steifigkeit von dem benutzten Plexiglas trotzdem noch erhalten bleibt.

Gleichzeitig muss auf die Anziehkraft der Schrauben geachtet werden, da bei zu hohen Kräften das Plexiglas durchbrechen könnte oder auch zu hohen Reibungskräften unterliegen würde, welche das System zu sehr unkontrolliert beeinflussen würde. Demnach sind alle Schrauben nicht komplett versteift eingeschraubt, was gleichzeitig zur Folge hat, dass Steifigkeit und Stabilität des Systems in gewissen Maßen drunter leiden.

Auch funktioniert die Führungsschiene nicht so wie angedacht, da zu hohe Reibungskräfte zwischen den einzelnen Plexiglasflächen entstehen. Ein kurzer Umbau lässt nun die Führungsschiene das System, über die schon angebrachten

Bolzen, an den Greifern führen. Auch diese Lösung ist nicht perfekt, denn wenn die Querkräfte zu hoch werden, verbiegt sich das Plexiglas und erfüllt demnach nicht mehr seine Funktion einwandfrei. Auch das ist mit einer erhöhten Wandstärke auf Kosten höheren Gewichts teilweise besser geworden. Da aber auch hier die Reibung zwischen Führungsbolzen und Plexiglas nicht komplett zu verhindern ist, lässt sich das Problem nicht ganz lösen.

Ein weiteres Problem ist der Öffnungswinkel des Systems. Denn um 60 mm gewollten Öffnungsspalt zu erreichen, müssten die Hebel weiter voneinander entfernt liegen, was das Volumen des Greifarms noch größer machen würde. Denn die Länge des Hubzylinders ist nicht stark variabel, weshalb der Bauraum zur Mitte hin begrenzt ist. Der momentane Öffnungsspalt von 45 mm sollte demnach erstmal beibehalten werden, da Verbesserungen größere Maßnahmen erfordern würden.

2.7 Fazit zur Mechanik

Bei der letztendlichen Fertigstellung der Mechanik zeigt sich die Komplexität darin die Balance zwischen Leichtbau, Stabilität und Steifigkeit beizubehalten, ohne dabei die Funktion des Greifarms einzuschränken und das mit einfachsten Mitteln. Der Verzicht auf einen Elektromagneten für die Greiffunktion führt zu einem großen eingenommenen Bauraum und viel Zusatzgewicht, da die Konstruktion der Greifarme für die anliegenden Kräfte eine gewisse Stabilität brauchen. In realitätsnahen Szenarien werden Faktoren wie zuverlässige Stabilität und möglichst lange Akkulaufzeiten jedoch benötigt. Dies stellt sich als eine zukünftige Hürde für andere Projekte dieser Art heraus. Die Grundfunktion an sich wird jedoch erfüllt.

Das Bauen ohne zuverlässige Lagerung erwies sich auch nicht als einfach und kostete dem System weitere Stabilität. Auch der enorme Bauraum, generiert durch den Hubzylinder, setzt dem System starke Grenzen und führt zu einem erhöhten Gewicht.

Kapitel 3

Sensorik und Aktorik

Was eine autonome Paketdrohne anspruchsvoller als eine herkömmliche Drohne mit Fernbedienung macht, ist vor allem die Fähigkeit, selbstständig sich zurechtzufinden und ihre Aufgabe ohne die Hilfe von Menschen zu erledigen. Um mit der Umgebung zu interagieren braucht man sowohl Sensoren, welche zur Wahrnehmung sprich zur Messung und Kontrolle von Umgebungsvariablen dienen, als auch Aktoren, welche letztendlich die Umgebung, gesteuert durch elektrische Signale, beeinflussen können. Die Sensoren sind also die Sinne, wie zum Beispiel: fühlen, sehen oder auch riechen, welche der Drohne zur Verfügung stehen, wobei die Aktorik die Arme und Beine der Drohne sind. Um also eine Drohne zu entwickeln, welche filigrane Bewegungen durchführen kann und sich überall zurechtfinden kann braucht man dementsprechend viele Sensoren.

3.1 Sensorik

Mit das Wichtigste bei einem automatisierten Projekt ist die Sensorik, denn das ist die Grundlage auf der das ganze System basiert. Ohne die richtig gewählte Sensorik ist das Projekt, in unserem Fall die Drohne nicht in der Lage sich zu recht zu Finden und ist quasi blind und somit unbrauchbar. Die Software kann noch so gut sein, jedoch hat auch sie keine Chance, wenn sie zum Beispiel die Drohne richtig positionieren soll, ohne die aktuelle Position zu kennen.

In diesem Kapitel geht es folglich um die Sensorik, welche wir explizit ausgewählt und verwendet haben und nicht um Beispielsweise das Gyroskop, welches in dem Flight Controller integriert ist, mit welchem wir aber nicht wirklich etwas zu tun hatten, da die Stabilitätsregelung bereits in dem Flight Controller vorprogrammiert war. Es wird einen Überblick geben, was für Sensoren sich in unserem

Projekt befinden und warum wir uns für diese entschieden haben und was unsere Auswahl beeinflusst hat.

3.1.1 Positionssensorik

Die Position eines Objektes kann man entweder relativ, also in Relation zu einem anderen Objekt angeben oder man gibt die Position absolut an, wie zum Beispiel bei einem GPS in Längen- und Breitengrad. Wir haben uns dazu entschieden eine Mischung aus beidem zu benutzen, da man so eine besonders hohe Genauigkeit erreichen kann.

Da die Drohne Pakete ausliefern können soll, brauchen wir dementsprechend einen Sensor welcher eine absolute Position wiedergibt. Um die absolute globale Position zu bestimmen, gibt es nur eine sinnvolle Möglichkeit, und zwar GPS, was für Global Positioning System steht und weltweit funktioniert. Da GPS Sensoren allerdings nur auf etwa 10 Meter genau sind und wir Pakete mit einer Genauigkeit von etwa +-5cm anfliegen wollen ist die Positionsbestimmung über GPS nicht ausreichend. Somit müssen wir weitere Sensoren zur Unterstützung verwenden. Zur genauen Lokalisierung des Pakets verwenden wir eine Kamera, die, wie in dem Kapitel Bilderkennung beschrieben, die genaue Position des Pakets ermittelt. Durch die Kamera lässt sich die Drohne sehr gut mittig über dem Paket platzieren, jedoch kann man sehr schwer die Höhe mit einer einzelnen Kamera bestimmen. Um die Höhe genau zu bestimmen bräuchte man entweder eine Stereokamera oder einen Abstandssensor, für welchen wir uns aus Kostengründen entschieden haben.

3.1.2 Ultraschallsensor

Zur Bestimmung von Abständen gibt es mehrere Ansätze, mit unterschiedlichen Vor- und Nachteilen. Man kann den Abstand über die Signallaufzeit, Triangulation oder auch über die Phasendifferenz bestimmen. Am Häufigsten sind hierbei die Signallaufzeitsensoren, welche ein Sendeimpuls abgeben und die Zeit bis zu dem Echo messen. Diese Art der Sensoren lässt sich nochmal in drei Unterkategorien einordnen mit jeweils unterschiedlichen Signalarten. Es gibt elektromagnetische (wie zum Beispiel Radar), optische (hierzu zählt unter anderem Infrarot und Lidar) und akustische (Beispielsweise Ultraschall) Abstandssensoren, die alle ihre Vor- und Nachteile haben. Wir haben uns für einen Ultraschallsensor entschieden, da dieser in dem von uns gewünschtem Bereich (ca. 2 Meter) auf etwa 10cm genau ist und vor allem weil es ein besonders kostengünstiger Sensor ist. Ein Lidarsensor wäre auch eine gute Alternative, jedoch sind diese Sensoren noch deutlich teurer als herkömmliche Ultraschallsensoren.

Den Ultraschallsensor haben wir direkt mit dem Raspi verbunden, da wir so mehr Kontrolle über den Sensor haben und dies in unseren Augen mehr Sinn macht als ihn mit dem Flight Controller zu verbinden.

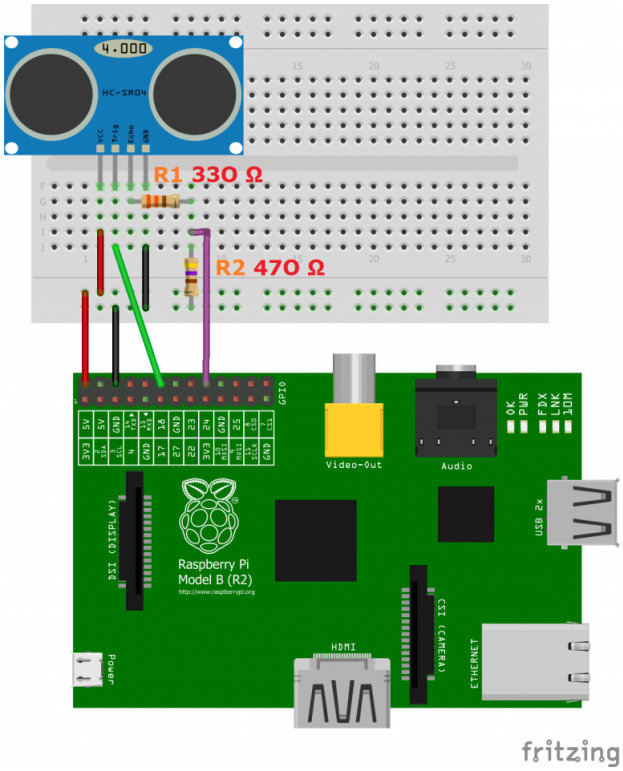

Abbildung 3.1: Ultraschallsensor Schaubild[1]

Unser Aufbau ist wie in der Abbildung 3.1 dargestellt, der Eingang des Sensors ist mit einem GPIO OUT Pin verbunden und der Ausgang über einen Spannungsteiler mit einem GPIO OUT verbunden um hier eine Spannung von 3,3V zu erreichen. Außerdem wird noch jeweils Masse mit Masse vom Raspi verbunden und 5V mit den 5V des Raspi verbunden. Es wird nun einmal die Sekunde die Entfernung nach Unten gemessen und an eine ROS Node gesendet, hierzu aber später mehr.

[1] Quelle: https://tutorials-raspberrypi.de/entfernung-messen-mit-ultraschallsensor-hc-sr04/

3.1.3 Global Positioning System (GPS)

3.1.3.1 Funktionsweise

Wie der Name Gloabl Positioning System schon impliziert handelt es sich bei GPS um ein System, welches einem die globale Position angibt. Es gibt mehrere solcher Systeme, das jedoch weitverbreitetste ist das NAVSTAR GPS, welches von den Vereinigten Staaten von Amerika in den 1970er-Jahren entwickelt wurde und bis heute ständig renoviert und instand gehalten wird. Dieses System besteht momentan aus 31 aktiven Satelliten im Orbit, es werden jedoch nur 24 Satelliten benötigt damit die aktuelle Konstellation funktioniert. Die Überschüssigen Satelliten werden lediglich zur Verbesserung der Signalstärke und steigerung der Fehlertoleranz verwendet. Das System ist so ausgelegt, dass man an jedem Punkt auf der Erde immer Kontakt zu mindestens vier Satelliten gleichzeitig hat. Jeder Satellit sendet in einem Takt von Einer Millisekunde ein Signal, welches sowohl die Position als auch die genaue Uhrzeit des Senders, sprich des Satellitens, beinhaltet. Mit diesen Daten kann der Empfänger dann seine aktuelle Position, sprich seine Höhe sowie Längen- und Breitengrad berechnen.
Normalerweise sind, um einen Punkt auf der Erde zu bestimmen, nur drei Satelliten notwendig, da aber der Empfänger meist keine ausreichend präzise Uhr besitzt, welche zur Berechnung des Standorts benötigt wird, da das Signal mit Lichtgeschwindigkeit unterwegs ist und somit etwa 300 Meter pro Mikrosekunde zurücklegt, wird ein vierter Satellit zum ausgleichen des Uhrenfehlers benötigt. Aus den vier Signalen wird dann über die Signallaufzeit die aktuelle Position berechnet.

Jedoch hat dieses System auch seine Limitierungen, denn sobald der Empfänger keine uneingeschränkte Sicht mehr zu vier Satelliten auf einmal hat, kann er die genaue Position nicht mehr bestimmen. Das Signal von den Satelliten wird relativ schnell von Hochhäusern Brücken oder Tunneln aufgehalten, weswegen wir auch unsere Tests nicht in unserem Labor durchführen konnten, da dort keine stabile GPS Verbindung garantiert ist.

3.1.3.2 Hardwareimplementierung

Die Positionierung auf der Drohne ist bei einem GPS Empfänger sehr wichtig, da wir hier ein bestmögliches Signal haben wollen. Hierzu positionieren wir das GPS so hoch es geht, um möglichst immer eine uneingeschränkte Sicht zum Himmel zu gewährleisten.
Wir haben bei uns ein GPS Modul verbaut, welches sich direkt mit dem Flight Controller verbinden lässt, somit weiß dieser immer seine genaue Position und teilt sie dann dem Raspi mit.

3.1.4 Kamera

Auch bei der Kamera hatten wir eine sehr große Auswahl von unterschiedlichen Modulen, mit den unterschiedlichsten Spezifikationen. Unsere Anforderungen waren hierbei wie folgt:

- Auflösung von mindestens 720p
- kompatibel mit dem Raspberry Pi
- Farbkamera
- klein und leicht
- preislicher Rahmen maximal 100€
- wenn möglich optische Bildstabilisierung
- wenn möglich gute Softwareunterstützung

Die größten Unterschiede, bei den mit dem Raspberry Pi kompatiblen Kameras liegen in der Größe, der Softwareunterstützung und des Interfaces. Für unseren Anwendungszweck war es von sehr großer Wichtigkeit, dass die Kamera möglichst kompakt aber vor allem leicht ist, da unsere Drohne unter den, von dem Gesetz vorgegebenen 2 kg bleiben soll und ein niedrigeres Gewicht auch eine längere Flugzeit bedeutet, welche ebenfalls sehr wichtig ist.
Bei dem Interface gibt es hauptsächlich zwei unterschiedliche Varianten. Variante Nummer eins ist ein normales USB-Interface, welches den Vorteil hat, dass es hier eine sehr große Auswahl gibt, jedoch sind diese Kameras meistens etwas größer und das wollen wir ja aus den oben gennanten Gründen vermeiden. Variante Nummer zwei ist die Verbindung über den SCI-Port am Raspberry Pi, das hat den Vorteil, dass man noch alle USB-Ports des Raspberry Pi frei für andere Erweiterungen hat aber auch, dass man eine deutlich bessere Kontrolle über die die Kamera hat. Man kann hier zum Beispiel die Belichtungszeit aber auch noch viele andere Parameter verstellen, was bei einer günstigen USB-Kamera meist gar nicht richtig geht oder nur sehr schwer zu realisieren ist.
Bei der Softwareunterstützung war uns wichtig, dass es eine sehr weit verbreitete Kamera ist, da dies meist bedeutet, dass es auch eine große Community gibt, die oft sehr hilfsbereit ist. Außerdem bedeutet das auch, dass es bereits viel Software gibt und man somit einige elementare Funktionen übernehmen kann und man somit mehr Zeit für die Projektspezifische Software hat.

Nach einer gründlichen Recherche hatten wir eine engere Auswahl an Kameras zusammengestellt, bei der wir uns letztendlich für das Raspberry Pi

Camera Module entschieden haben, da diese schon vor Ort war und wir somit direkt mit den Versuchen starten konnten. Ein weiterer Grund war die sehr gute Software Unterstützung, da die Kamera direkt für den Raspberry Pi entwickelt wurde. Außerdem hatten wir somit auch alle Vorteile die mit dem SCI-Interface kommen. Die einzige Anforderung, die nicht erfüllt werden konnte, war die optische Bildstabilisierung. Jedoch war eine Kamera mit ähnlichen Funktionen und optischer Bildstabilisierung um etwa einen Faktor fünf teurer.

Abbildung 3.2: Raspberry Pi Camera Module an einen Raspi angeschlossen[2]

Bei unzureichenden Ergebnissen, sprich unscharfe Bilder aufgrund der Vibrationen der Drohne, hätten wir immer noch die Möglichkeit gehabt, eine andere Kamera zu verwenden. Da die PiCam keine optische Bildstabilisierung hat ist es noch ungewiss ob das Bild stabil genug für unsere Zwecke ist. Das zu Testen lag leider nicht mehr in unseren Möglichkeiten, da auf Grund der Covid-19 Situation keine weiteren Versuche möglich waren.

[2] Quelle: https://www.raspberrypi.org/products/camera-module-v2/

3.2 Aktorik

Um zu wissen welche Aktorik man benötigt, haben wir zuerst eine Liste an Anforderungen erstellt, um zu wissen, was man mit der Aktorik überhaupt erreichen will. Die Aktorik geht außerdem sehr eng mit der Mechanik zusammen, weswegen wir in dem Bereich sehr nahe zusammen gearbeitet haben. Die Anforderungen waren wie folgt:

- Greifen eines $10mm \times 10mm \times 10mm$ großen Pakets
- Drohne in x-, y- und z- Richtung steuerbar
- sowohl Rotation um die z- Achse

Mit dieser Liste haben wir nun begonnen den Greifarm und die Drohne zu konzipieren, wobei das bei der Drohne eine deutlich einfachere Aufgabe war als bei dem Greifarm. Für die Drohne benutzen wir einen Mikrocontroller zur Regelung aller Flugbewegungen.

3.2.1 Hubzylinder

Mit der aus Kapitel 3 bekannten Greifermechanik, stellte sich nun die Frage durch welche Art von Aktorik der Greifarm nun bewegt werden soll, auch hierzu haben wir wieder mehrere unterschiedliche Ideen gesammelt und miteinander verglichen. Bewertet haben wir nach folgenden Kriterien:

- Komplexität
- Robustheit
- Gewicht
- Fehleranfälligkeit
- Effizienz

Zur Auswahl hatten wir schließlich einen Seilzug, angetrieben durch einen Elektromotor und einen Hubzylinder. Die meistgenutzten Hubzylinder sind entweder mit pneumatisch, hydraulisch oder elektrisch angetrieben, jedoch kommt für unsere Anwendung nur der elektrische Antrieb infrage, da Hydraulik und Pneumatik zu groß und schwer sind, denn dafür würde auch noch eine Pumpe benötigt werden.

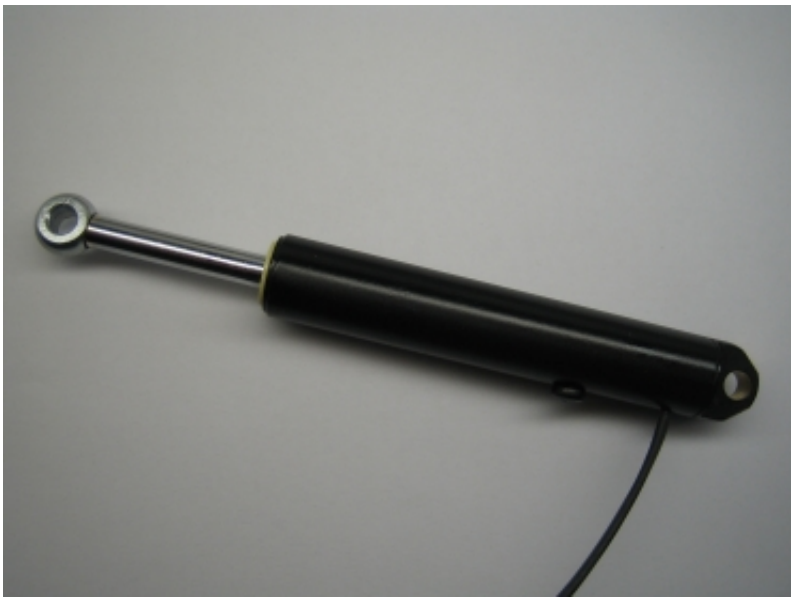

Abbildung 3.3: Elektrischer Hubzylinder[3]

Unsere Entscheidung ist zugunsten des elektrischen Hubzylinders gefallen, da dieser das beste Verhältnis zwischen Robustheit und Gewicht hat und dazu noch einen sehr simplen und robusten Aufbau hat. Der Hubzylinder, den wir in unserem Projekt verbaut haben, kommt aus dem Modellbau, da in diesem Segment Bauteile in der Größenordnung, wie wir sie benötigen oft vorkommen und genau für den Einsatzzweck an ferngesteuerten Modellen, was unsere Drohne im Grunde genommen auch ist.
Der Zylinder wird über einen Regler angesteuert, welcher die Endpositionen des Zylinders erkennen kann und bei einer Last von etwa 60N aufgrund des Überlastungsschutzes abschaltet.

3.2.2 Flight-Controller

Für die Steuerung der Drohne wird ein Controller benötigt, welchen wir über den Raspberry Pi ansteuern können. Hierfür haben wir den bereits am Institut vorhandenen Pixhawk Flight Controller benutzt. Dieser muss lediglich mit dem TX- und RX-Pin des Paspberry Pi verbunden werden. Außerdem empfiehlt es sich ebenfalls die Ground Verbindung des Raspberry Pis und die des Flight Controllers miteinander zu verbinden, da das die Signalqualität verbessert und es sonst zu Problemen kommen könnte.

Nun stellt sich aber erstmal die Frage, warum wir überhaupt einen sol-

[3]Quelle: https://www.cti-modellbau.de/CTI-Hubzylinder/titan/

chen Flight Controller benötigen und die Drohne nicht einfach direkt über den Raspberry Pi ansteuern. Das hat mehrere Gründe, zum einen ist die linux Version, welche sich auf dem Raspi befindet, nicht echtzeitfähig und somit kann es zu Problemen bei der Regelung kommen, wenn Beispielsweise der Raspi ausgelastet ist, würde die Regelung, welche der Quadrokopter benötigt, zu kurz kommen und somit der Flug sehr instabil werden. Zum anderen sparen wir uns damit sehr viel Zeit und Arbeitsaufwand, da der Flight Controller bereits die ganze Regelung für die Drohne besitzt und bereits mit unterschiedlichen Sensoren verbunden ist. Ein weiterer Vorteil dieser Methode ist, dass man die Drohne mit dem extra Flight Controller auch Manuel über eine Fernbedienung steuern kann, was bei einem Raspi nicht unmöglich ist, aber nochmal eine Menge mehr Aufwand bedeuten würde.

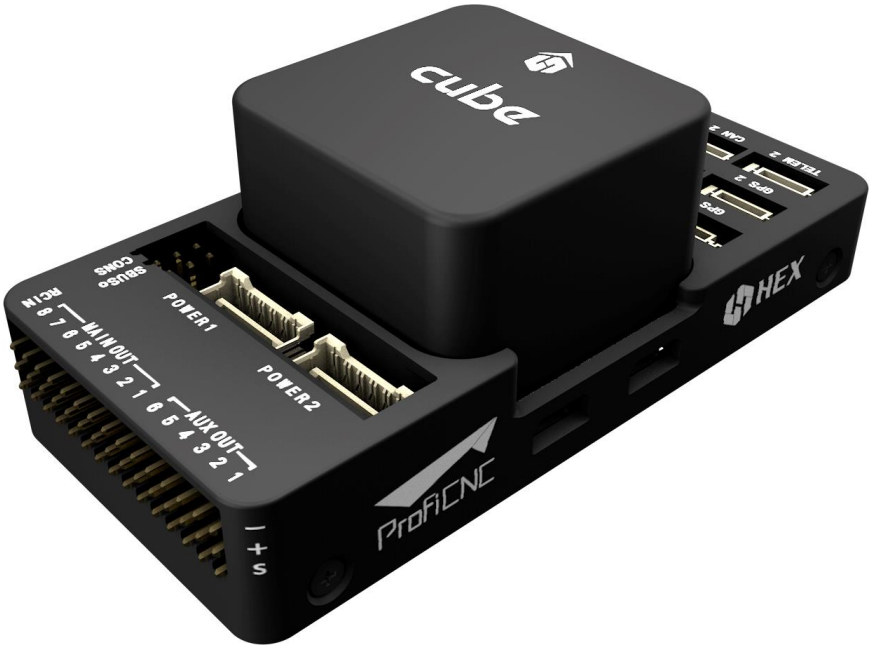

Abbildung 3.4: Flight Controller Pixhawk 2[4]

An diesem Controller sind nun die vier Elektromotore über ESCs (Electronic speed control) verbunden. Außerdem wird noch ein Empfänger für das Empfangen des Signales der Fernsteuerung angeschlossen, sowie bereits oben erwähnt das GPS Modul. Der Flight Controller bekommt später also nur noch Angaben wie zum Beispiel bewege dich zwei Meter nach oben und sorgt dann dafür, dass der Kopter sich auch genau zwei Meter nach oben bewegt. Das entlastet den Raspberry Pi extrem, sodass dieser genug Rechenleistung für die Bilderkennung übrig hat.

[4] Quelle: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk-2.html

Kapitel 4

Softwareimplementierung

4.1 Softwareanforderung

Ein weiterer Teil des Projektes ist die Softwareimplementierung. Die Aufgabe dieser erstreckt sich von der Bildverarbeitung über den Objekterkennungsprozess bis hin zu der Regelung und Steuerung der Drohne. Dabei muss sie verschiedensten Kriterien genügen.

Da sie Signale von einem Externen Bauteilen wie dem PX4 oder dem Abstandssensor über das Interface ROS (siehe Kapitel 6) kann es zu Datenverlust kommen. Bedingt durch die unterschiedlichen Taktrate der Prozessoren. Deshalb muss auch bei gelegentlich fehlenden Datenpaketen der Rest verarbeitet und die Drohne auf Grundlage dieser geregelt werden.

Der PX4-Controller schreibt eine Datenrate von 20 Hz vor. Also muss in dieser Datenrate die Sollposition der Drohne geschickt werden. Fallen diese aus, so geht die Drohne in den Offboard-Modus und steigt auf 10 m Höhe, um dann automatisch zur Startposition zurückzukehren. Dies wäre bei einer Raumhöhe von 2,5 m fatal. Leider kann man diese „Schutzfunktion“ nicht abschalten da sie fest im Betriebssystem des PX4-Controllers verankert ist. Demzufolge darf das Signal der Position nie ausbleiben.

Um dies zu erreichen, darf sich die Regeleinheit der Drohne nie aufhängen. Alle Schleifen müssen deshalb ein zeitliches Abbruchsignal haben, um eine Auslastung des Arbeitsspeichers zu verhindern. Aufgrund des Gewichtes muss ein leichter Controller gewählt werden(siehe Hardware Komponenten). Da wir aufgrund der Software Inkompatibilität des Pi-4 mit dem PX4-Controller einen Pi-3 gewählt haben steht ein maximaler Arbeitsspeicher von 1 Gb zur

Verfügung. Dabei hat er 2 Kerne und kann somit effektiv 2 Aufgaben gleichzeitig ausführen. Die Software muss also sehr ressourcensparend geschrieben sein.

Aufgrund der Struktur von ROS empfiehlt es sich, einen Event basierten Programmablauf zu schreiben. Dabei löst ein Event, also eine Aktion die stattfindet, einen Programmablauf aus. In diesem Fall sind die Events eingehende Datenpakete für die entsprechende Node (Siehe Programmstruktur). Diese werden verarbeitet. Dadurch kann sichergestellt werden, dass jedes Datenpaket verarbeitet wird, unabhängig in welchem Abstand es ankommt. Durch diese Warte-Bedien-Struktur wird eine maximale Effizienz erreicht, da das Programm keine Schritte doppelt auf nicht aktualisierte Werte anwendet und so Prozessorzeit spart.

4.2 Python als Programmiersprache

Wie es häufig in der Entwicklung der Fall ist, stehen auch hier für die Implementierung der Software auf dem RPi verschiedene Plattformen, Frameworks und Programmiersprachen zu Verfügung. Bei der Wahl der Programmiersprache ist zu beachten das sie möglichst einfach und bereits relativ alt ist. Der Vorteil von alten Programmiersprachen ist, dass es eine umfangreiche Beispielsammlung gibt. Dadurch kann man sich leicht an bereits vorhandenem Code orientieren. Anforderungs-bedingt muss die Programmiersprache möglichst ressourcenschonend sein. Dadurch fallen alle grafischen Programmiersprachen wie Matlab, Simulink, Labview etc. heraus. Diese haben aufgrund ihres unstrukturierten Programmablaufs, eine eher schlechte Laufzeit.

Eine der am weitesten verbreiteten Programmiersprachen ist Java. Der Vorteil von Java ist, dass es plattformübergreifend funktioniert. Leider verwendet Java dafür eine Virtuelle Java Maschine. Also ein Betriebssystem, dass noch mal über dem Betriebssystem liegt. Dies verringert die Laufzeit von Java enorm. Ein weiteres Problem liegt darin, dass Schnittstellen immer plattformabhängig sind(USB, D-Sub, Ethernet etc.). Dadurch werden sie durch Java nur schlecht unterstützt. Aus diesen Gründen konnte Java ebenfalls nicht gewählt werden.

Eine weitere Alternative ist eine C-Programmiersprache(C++, C etc.). Diese haben den großen Vorteil, dass sie echtzeitfähig sind. Sie können also unter Umständen garantieren, dass das Ergebnis des Algorithmus nach einer bestimmten Zeit feststeht. Dies ist für eine Drohnensteuerung sehr geeignet, da man garantieren sollte, dass sie Drohne in einem bestimmten Zeitintervall überwacht wird. Außerdem sind C-Sprachen sehr Hardware nah geschrieben, wodurch die

Laufzeit sehr gut ist. Allerdings wird die Hought-Transformation der Bibliothek Open-CV genutzt. Diese Bibliothek ist allerdings nicht echtzeitfähig, wodurch der große Vorteil von C verloren geht. Des Weiteren ist es sehr schwer/mühselig Open-CV in C zu implementieren.

Außerdem ist Python eine Alternative. Python ist leider nicht echtzeitfähig. Jedoch ist es sehr benutzerfreundlich, hinsichtlich des Importierens von Bibliotheken. Es ist nicht so hardwarenah wie C, allerdings immer noch deutlich näher als Java oder Labview. Außerdem gibt es zahlreiche Beispiele für Python und das Ansteuern von Schnittstellen geht problemlos. Aus all diesen Gründen ist die Wahl auf Python gefallen.

4.3 Vergleich Monolithische Single Application und Service Oriented Architecture

4.3.1 Monolithische Single Application

Bei der Softwarearchitektur wurde sich zunächst an einer normalen objektorientierten Struktur bedient: Es gibt eine main-Datei, welche vergleichsweise klein ist und nur die grobe Struktur bzw. den Gesamtablauf darstellt. Diese ruft dann weitere Softwaremodule auf, die die Unteraufgaben, wie beispielsweise das Empfangen und Verarbeiten von Bilddaten, implementiert. Ein Problem, das hier recht schnell auftritt ist, dass theoretisch zwei Prozesse gleichzeitig ablaufen müssen. Wenn ein Motormodul beispielsweise gerade darauf wartet, dass die Sollposition erreicht wurde, müsste in einem Programmierstrang, in dem ein Befehl nach dem anderen ausgeführt wird, das Kameramodul mit der Verarbeitung des Bildes auf das Motormodul warten, obwohl das Motormodul gerade gar nichts tut, außer auf den Motor in der echten Welt zu warten, bis er die gewünschte Position hat. Da das Kameramodul aber an das Motormodul eventuell kommunizieren muss, dass das Paket gar nicht mehr an der richtigen Stelle ist, weil der Quadrokopter aufgrund äußerer Umstände nicht ganz stillsteht, müssen beide Module gleichzeitig arbeiten und miteinander kommunizieren können.

In der traditionellen Softwareentwicklung gibt es dafür sogenannte Threads. Diese sind separate Ausführungsstränge, die quasi gleichzeitig ausgeführt werden und sich dann beispielsweise ein Datenobjekt im Speicher teilen und über dieses miteinander kommunizieren können. Das entspricht dann einer Dreischicht-Architektur (siehe Abb. 4.1).

Wichtig bei dieser Art der Architektur ist, dass jede Schicht nur auf die nächst untere zugreifen darf, um einen reibungslosen Ablauf garantieren zu können. In diesem Fall greifen nun die Threads T_1, T_2 ... T_n nicht direkt auf den Speicher zu, sondern müssen den Weg über die Datenzugriffsschicht gehen. Diese stellt unter anderem mittels Semaphoren sicher, dass immer nur ein Thread auf die entsprechende Ressource zugreifen kann. Damit wird verhindert, dass beispielsweise T_1 schreiben und T_2 gleichzeitig lesen möchte. Dies würde zu einem Speicherkonflikt führen und das Programm wird abstürzen. [?]

Abbildung 4.1: Dreischichtarchitektur

4.3.2 Service Oriented Architecture

Bis zu einem gewissen Grad war es möglich die geforderten Funktionalitäten mit Multithreading zu implementieren. Allerdings wurde es im Laufe der Entwicklung immer komplizierter, da Bilddaten beispielsweise von zwei Ressourcen abhängen, die nicht immer verfügbar sind: zum einen der Speicherplatz im Datenobjekt, das andere Threads lesen und damit belegen wollen und zum anderen von der Kamera selbst, die natürlich auch nicht willkürlich Bilddaten liefert. Das zu synchronisieren ist zwar theoretisch möglich, praktisch aber mit einer Menge schwer wartbarem Code verbunden, der das Problem unnötig kompliziert löst.

4.3.3 Robot Operating System

Deshalb basiert die endgültige Software auf ROS, das genau diese Probleme adressiert. ROS steht dabei für Robot Operating System und ist ein Framework, das auf dem Publisher-Subscriber-Modell (P/S-Modell) basiert. Die Idee ist dabei, dass es verschiedene Themen (Topics) gibt, in die verschiedene dezentrale Skripte, die gleichzeitig laufen, Informationen zu einzelnen Themen veröffentlichen oder abonnieren können. So gibt es beispielsweise ein Kameraskript, das das Bild

und die erkannte Position des Pakets veröffentlicht und ein Motorskript abonniert die Paketposition und kann daraus dann weitere Schlüsse ziehen. Ein anderes Skript könnte dann auf einem anderen Rechner (z.B. Laptop) im gleichen (WLAN-) Netzwerk das Kamerabild abbonnieren und mit sehr wenig Programmieraufwand anzeigen. Das elegante dabei ist, dass sämtliche vorher beschriebene Multithreading-Probleme dabei vom sogenannten ROS-Core, der zentrale Schaltstelle des ROS, übernommen werden und damit die einzelnen (selbstgeschriebenen) Skripte sehr entschlackt werden. Das macht es deutlich einfacher diese zu debuggen. ROS ist zudem sehr gut dokumentiert und für viele Probleme gibt es bereits vorgefertigte Skripte, die Dank des einfachen P/S-Modells auch einfach anzubinden sind.

4.3.4 ROS als Beispiel einer Service Oriented Architecture (SOA) / Microservices

Die Architektur hinter ROS ist dabei keine neue Erfindungen, sondern basiert im Gegensatz zum Ansatz der Schichtenarchitektur auf dem Prinzip der serviceorientierten Architektur (SOA).

Bei der SOA spielt der (Enterprise) Service Bus eine zentrale Rolle. Er ist das Bindeglied zwischen allen anderen Knotenpunkten im Netzwerk und stellt sicher, dass die Kommunikation funktioniert. Bei ROS ist das der ROS-Master, der über den Befehl

```
1 $ roscore
```

gestartet werden kann. Über ihn kommunizieren die Knoten miteinander, wobei jeder Knoten eine logisch abgeschlossene Aufgabe übernimmt und von außen über eine API angesprochen werden kann. Im Fall des Quadrokopters existiert nun beispielsweise ein Knoten, der das Kamerabild publiziert, einer, der das Bild verarbeitet und einer, der die Steuerung auf Basis der Bilddaten durchführt. Durch diese Modularität gibt es einige Vorteile. So ist es damit sehr leicht einen Knoten auszutauschen, zu erweitern oder neue Knoten hinzuzufügen. Zudem ist das System deutlich stabiler als eine monolithische Lösung, da bei einem fehlerhaften Knoten nur die Knoten ausfallen, die von diesem Ausfall logisch betroffen sind. Alle unabhängigen Knoten werden weiterhin funktionieren. Bei einer monolithischen Lösung führt ein Modulausfall zum Ausfall des Gesamtsystems.

Aber einer gewissen Granularität spricht man von Microservices. Diese folgen der Strategie “Erledige nur eine Aufgabe und erledige sie gut” Bei “normalen” SOAs sind die Knoten tendenziell groß und implementieren viele Funktionalitäten auf einmal. Microservices hingegen sind sehr klein und erledigen nur eine bestimmte Aufgabe. In der finalen Gesamtarchitektur kommen Knoten zum Einsatz, die

sowohl den normalen SOAs als auch den Microservices zugewiesen werden können.

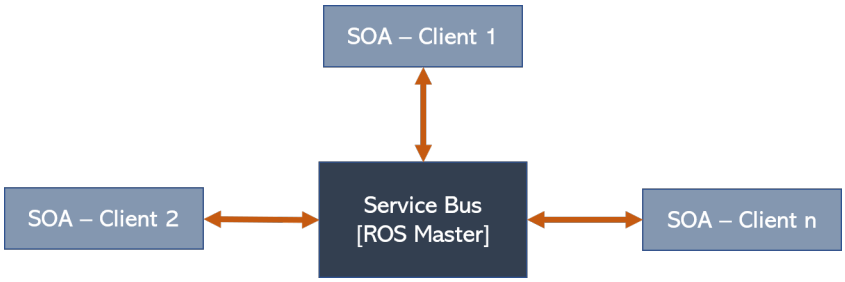

Abbildung 4.2: Service Oriented Architecture

4.4 Finale Gesamtarchitektur - Übersicht

In der finalen Gesamtarchitektur (siehe Abb. 4.3) gibt es insgesamt drei ROS Instanzen: Die erste läuft auf dem Quadrokopter selber und dient als Schnittstelle zur Firmware des Quadrokopters. Der dazugehörige Knoten heißt Mavros, welcher mit MAVLink (Micro Air Vehicle Link) kommuniziert, das die gesamte tatsächliche Steuerung des Quadrokopters übernimmt. Dabei hat der ROS-Core jedoch keine Master-Funktion, sondern verbindet sich über eine serielle Schnittstelle mit dem ROS-Core auf dem RaspberryPi, auf dem der tatsächliche Master-ROS-Core läuft. Dieser ist in einem WLAN-Netzwerk mit einem Laptop, der für Debugging-Zwecke und zum Konfigurieren und Kontrollieren der Bildverarbeitung benötigt wird. Auch auf dem Laptop läuft ein ROS-Core, wobei auch dieser sich mit dem Master auf dem Raspberry Pi verbindet. Im folgenden werden nun die einzelnen Knoten genauer vorgestellt.

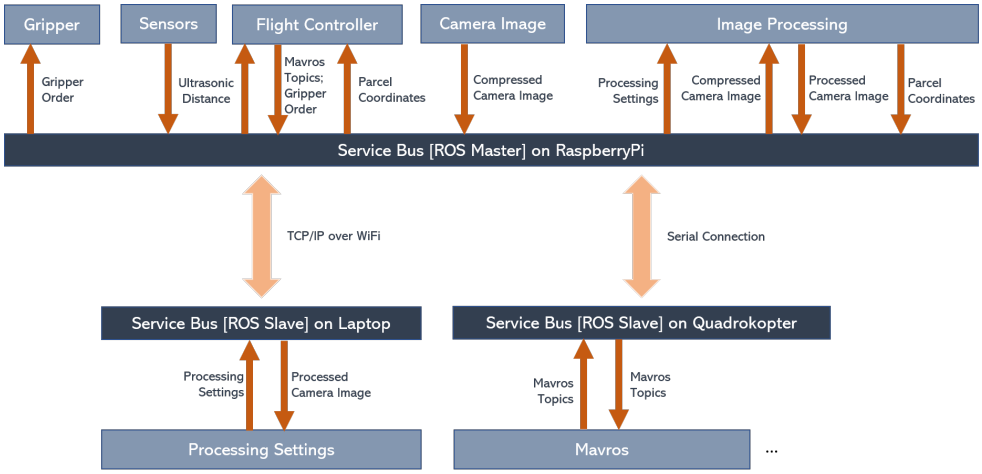

Abbildung 4.3: Service Oriented Architecture

4.4.1 Camera Image

"Camera Image" ist ein einfacher Publisher, der das Bild der Dronenkamera komprimiert und über den Service Bus den anderen Knoten im Netzwerk als serialisiertes numpy-Array zu Verfügung stellt. Aufgrund der später folgenden rechenintensiven Bildverarbeitung sendet dieser in der finalen Version das Bild jedoch nur mit 10 Hz. Dies is jedoch immer noch ausreichend, da sich die Geschwindigkeit der Drone nicht schnell ändern wird.

4.4.2 Image Processing

"Image Processing" empfängt die Bilddaten von Camera Image und wir zudem über das Topic Processing-Settings vom Laptop aus konfiguriert. Dieser Knoten filtert dann das Bild um anschließend die Position und Drehung des Pakets zu ermitteln. Das gefilterte Bild und die Koordinaten als Typ

```
1 PoseStamped
```

des Pakets werden publiziert. Dieser Nachrichtentyp enthält unter

```
1 geometry_msgs/Point
```

drei Koordinaten (x, y, z) wobei in diesem Fall x und y die Position und z die Drehung um die z-Achse angibt.

4.4.3 Sensors

"Sensors" verwaltet alle Sensordaten. In der aktuellen Version ist das jedoch nur der Ultraschallsensor. Die Daten werden gesammelt, aufbereitet, validiert und publiziert. So werden dann beispielsweise die Ultraschalldaten unter der Topic

```
1 /sensor_dist
```

im Datenformat Float32 mit einer Frequenz von 20 Hz veröffentlicht.

4.4.4 Flight Controller

"Flight Controller" ist die zentrale Steuereinheit der neuen Funktionalität. Er empfängt alle aufbereiteten Sensordaten wie die Koordinaten des Pakets, Ultraschalldistanz, Sensordaten der Drone etc. und führt die Suche und Anflug des Pakets durch. Er gibt die Flugsteuerbefehle an Mavros über die serielle Schnittstelle weiter und gibt Greifbefehle weiter. Mehr dazu ist im Kapitel 6 zu finden.

4.4.5 Datenübertragungsformate

Es war sehr schwierig, die richtigen Formate zu wählen da Python im Allgemeinen eher unsauber mit den verschiedenen Datentypen umgeht. So gibt es z. B. keinen Unterschied zwischen einem 32-Bit oder einem 64-Bit Integrer. Falls die Zahl außerhalb des darstellbaren Bereiches des 32-Bit Interger liegt, so erweitert der Python-Interpreter es selbstständig zu einem 64 Bit Integer. ROS ist hingegen sehr typenspezifisch, wodurch es zuerst einige Unklarheiten bezüglich der Typen gab und diese dementsprechend verändert werden mussten.

4.5 Implementierung der Kommunikation in ROS

Wie bereits erwähnt basiert ROS auf dem Publisher/Subscriber Prinzip. Ein Knoten kann in ROS sowohl Publisher als auch Subscriber sein und das auch für mehrere Topics. Die Topics sind dabei hierarchisch aufgebaut und gehen vom Groben ins Feine. So publiziert beispielsweise der "Camera Image" Knoten das komprimierte Bild in

```
1 /camera/image/compressed
```

.

Anhand dieses Beispiels soll nun dargelegt werden, wie man prinzipiell bei der Erstellung eines Nodes vorgeht.

4.5.1 Kamera Publisher

Zunächst muss eine Datei mit dem Namen des Knotens und der Endung ".py" im entsprechenden Paketordner im src-Ordner des Catkin-Workspaces erstellt werden. Diese kann dann in PyCharm geöffnet werden, wenn nicht schon das gesamte Paket als Projekt geöffnet wurde. Um klarzumachen, dass es sich hierbei um ein Python File handelt, muss die Datei mit

```
1 #!/usr/bin/env python3
```

starten, um die Pythonumgebung auszuwählen. Es folgt nun für gewöhnlich die Lizenzerklärung zur Datei. Es sollten nun

```
1 numpy, time, cv2, roslib, rospy
```

importiert werden. "numpy" ist eine Bibliothek zum effizienten Arbeiten mit numerischen Arrays. "time" gibt Zugriff auf die aktuelle Systemzeit. "cv2" ist die

Python-Version von OpenCV, mit der die Bildverarbeitung implementiert wird. “roslib” und “rospy” sind die Schnittstellen zum ROS-Bus. Außerdem muss für dieses Skript

```
CompressedImage
```

aus der ROS-Bibliothek

```
sensor_msgs.msg
```

importiert werden.

Es empfiehlt sich nun die gesamte Funktionalität mittels

```
def function_name
```

in eine Funktion zu packen. Dort muss dann zunächst mit

```
pub = rospy.Publisher("", DataType)
```

ein Objekt erstellt werden, wobei hier das Topic

```
"/camera/image/compressed"
```

,

sowie der Datentyp

```
CompressedImage
```

dem Konstruktor übergeben werden müssen. Mit

```
rospy.init_node('node_name',
        anonymous=True)
```

registriert sich der Knoten nun am ROS-Service-Bus. Nun wird die Computerbildverarbeitungsbibliothek OpenCV genutzt, um mit

```
cap = cv2.VideoCapture(0)
```

ein Objekt zu erstellen, dass auf den Datenstrom der ersten angeschlossenen Kamera zugreift. Der nun folgende Code soll so lange ausgeführt werden, wie ROS aktiv ist. Dies lässt sich mit

```
while not rospy.is_shutdown():
```

implementieren. Mit

```
ret, frame = cap.read()
```

wird nun der Datenstrom der Kamera ausgelesen und das aktuelle Bild in die Variable

```
frame
```

geschrieben. Jetzt muss mit

```
msg = CompressedImage()
```

ein Nachrichtenpaket des Typs

```
CompressedImage
```

erstellt werden, das mit

```
msg.header.stamp = rospy.Time.now()
```

um einen Zeitstempel erweitert wird und mit

```
msg.format = "jpeg"
```

das richtige Dateiformat erhält. Die Daten des Pakets werden mit der Property

```
msg.data
```

gesetzt. Diese müssen nun aus dem vorher gelesenen

```
frame
```

erstellt werden. Dafür wird zunächst mit

```
cv2.imencode('.jpg', frame)[1]
```

das

```
frame
```

mithilfe von OpenCV in ein JPEG umgewandelt und wird dann mit

```
np.array()
```

zu einem Bildarray transformiert wird und mit

```
tostring()
```

serialisiert wird. Nun ist Paket bereit mit

```
pub.publish(msg)
```

an den Service Bus gesendet zu werden. Die Funktion sollte nun mit

```
if __name__ == '__main__': function_name()
```

aufgerufen werden, wobei es Sinn macht den Funktionsaufruf mit einer Ausnahmeregelung für die

```
rospy.ROSInterruptException
```

zu erweitern.

4.5.2 Kamera Subscriber

Der dazugehörige Subscriber ist im Knoten "Image Processing" implementiert und ist dem Publisher gegenüber recht ähnlich aufgebaut. Hier wird jedoch ein Subscriber-Objekt mit

```
rospy.Subscriber("topic",
        DataType,
        callback)
```

erstellt, wobei callback die Callback-Funktion ist, die aufgerufen wird, wenn eine neue Nachricht in der Topic ankommt. In den Argumenten dieser Funktion eine Variable

```
msg
```

definiert sein, in der die Nachricht bei Empfang gespeichert wird. Der Inhalt der Nachricht kann mit

```
msg.data
```

aufgerufen werden und mit

```
np.fromstring(data, np.uint8)
```

zuerst zu einem

```
numpy
```

-Array deserialisiert und dann mit

```
cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
```

zu einem für OpenCV verarbeitbarem Format dekodiert werden. Der darauf folgende Code zur Bildverarbeitung ist in Kapitel 5. beschrieben.

4.6 Starten und Automation eines ROS-Systems

Damit die Knoten auch lauffähig sind, müssen die Dateien zunächst mit

```
chmod +x
```

für den Kernel ausführbar gekennzeichnet werden. Nun können, nachdem

```
roscore
```

ausgeführt wurde, in einem jeweils neuen Terminaltab mit

```
rosrun paketname knotenname.py
```

die verschiedenen Knoten ausgeführt werden. Mit

```
rostopic echo /topic/name
```

können die Nachrichten der verschiedenen Topics überwacht werden. Um das alle nicht jedes Mal von Hand machen zu müssen, gibt es bei ROS sogenannte Launch-Files, die, wenn ausgeführt, nicht nur einen ROS-Core ausführen, sondern auch alle notwendigen Knoten startet, Umgebungsvariablen setzt etc. Das Launch File basiert auf XML und enthält innerhalb des

```
<launch></launch>
```

Tags alle Komponenten. Ein Knoten ist dabei recht selbsterklärend aufgebaut:

```
<node name="CameraImagePub"
        pkg="parcelcopter"
        type="CameraImagePub.py"/>
```

Innerhalb des Tags können außerdem Startparameter mit

```
arg
```

definiert werden. Mit dem "env"-Tag können zudem Umgebungsvariablen gesetzt werden.

Man bemerke hier die Modularität und Stabilität von ROS. Denn die Knoten geben keinen Fehler aus, obwohl ein anderer Knoten, der eigentlich für die Ausführung von Nöten ist, noch gar nicht gestartet ist. Sobald dieser bereit ist, fangen die davon abhängigen Knoten automatisch an zu arbeiten.

4.7 Mavros und Simulation

Die Simulation der Drohne erfolgt in Gazebo. Dafür wird ein PX4-Controller simuliert. Gazebo greift die Signale von dem ROS Interface ab und verarbeitet sie. Dafür wird eine FPL-Drohne simuliert. Ihre Daten sendet dann Gazebo wieder an den PX4-Controller. Weiterhin sendet es einen Video-Stream mit den

simulierten Bildern. Dieser wird dann von der Node ImagePub erkannt und in ROS gestreamt. Eigentlich kommt er schon über ROS aber auf diesem Wege ist es realitätsgetreuer. (Die Alternative wäre einfach eine andere Quelle für den ImagePub anzugeben.) Auf diesem Wege kann die Drohne nahezu eins zu eins simuliert werden.

Auf diesem Wege wurden bereits viele Fehler wie z. B. Vorzeichenfehler bei den Berechnungen gefunden. Jedoch ist es etwas kompliziert neue Drohnen zu implementieren, da dies erst die Umgebung programmiert werden muss. Unten sieht man einen Kurzfilm, welcher eine Simulation zeigt.

Auch bei der Simulation ist ROS von Vorteil, da man im Testmodus einfach einen Sensor simulieren kann ohne die eigentlichen funktionsverarbeitenden Programme umschreiben zu müssen. Diese Programme laufen dann auf einem externen PC, der sich im gleichen Netzwerk wie die der Pi befindet, und senden regelmäßig "Fake" Nachrichten (auch als Fake News) bekannt. Außerdem kann man so über einen externen PC einfach den Datenfluss kontrollieren, da man quasi die Rohdaten abrufen kann, um die Funktion der einzelnen Programme zu überprüfen. Das ist natürlich auch ein gewisses Sicherheitsrisiko, weshalb die eingehende Kommunikation in den Service Bus beispielsweise durch ein sicheres WiFi-Netzwerk restriktiert werden muss.

Kapitel 5

Bilderkennung

Bei dem Bildverarbeitungsprozess muss das Paket vor verschiedenen Hintergründen, erkannt werden. Dabei darf die Beleuchtung keine Rolle spielen. Die charakteristischen Eigenschaften sind neben Farbe somit ebenfalls Form und Größe.

Zuerst wurde RGB(Red, Green, Blue) Farbraum gefiltert. Jedoch ergaben Test's das der RGB-Farbraum sehr belichtungsempfindlich ist. Deswegen wird das Bild in den HSV(Hue, Saturation, Value) Farbraum konvertiert. Dieser ist deutlich belichtungsstabiler.

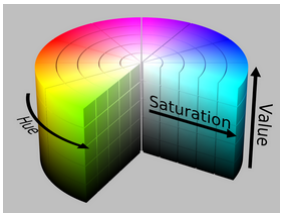

Abbildung 5.1: HSV-Raum[1]

Als Farbe wäre Rot sehr geeignet, da es in unsere Umgebung sehr selten vorkommt. Allerdings gibt es keine Roten-Pakete. Außerdem ist Rot für eine Kamera sehr schwer zu erkennen, da es eine der Grundfarben ist und damit in fast jeder Farbe vorkommt. Deswegen wurde als Farbe weiß gewählt. Durch den Filter werden nun weiße Flächen weiß und der Rest wird schwarz. Da das Experiment auf einem dunklen Boden stattfindet, bietet sich dies an.

Weiterhin muss auf dem Paket mittig, ein weiteres schwarzes Viereck sein. Dies ist nötig um eine Orientierung bei naher Distanz zu gewährleisten. Befindet sich die Drohne nahe über dem Paket, so füllt es das komplette Kamerasichtfeld aus.

[1] Quelle: http://www.subcolors.de/content/public/colorsystems/rgb.html

Um sich nun orientieren zu können ist ein weiteres Viereck nötig. Dies muss mittig sein, da sich die Drohne im Verlauf des Anflugs parallel zu dem Rechteck ausrichtet. Damit bei allen Rotationen der Versatz immer gleich ist, muss es mittig sein. Dadurch kann man ihn in der Kameraregelung berücksichtigen.

Für die Form wird der Hought-Transformation verwendet. Bei dieser wird das Bild zuerst abgeleitet. Also werden Pixel neben den sich ein andersfarbiges befindet weiß und der Rest(Weiß neben Weiß, Schwarz neben Schwarz) wird schwarz. Nun sind die Kanten deutlich zu sehen. Anschließend wird durch jeden weißen Punkt eine Linie mit verschieden Anstiegen zwischen - unendlich und + unendlich gelegt. Schneidet man dabei mit der Linie einen anderen weißen Punkt, so wird sich diese Linie gemerkt. Linien, die durch mehrere weiße Punkte gehen werden, so als Linien detektiert. Der Algorithmus wird aus der OpenCV Bibliothek implementiert. Er hat die Komplexität: n! (n Fakultät). [**?**]

Anschließend wird das Bild nach Konturen durchsucht. Dabei sind nur die Konturen mit 4 Ecken von Bedeutung. Diese Konturen werden nun nach minimaler und maximaler Größe gefiltert. Die Intervallbreite variiert abhängig von der Flughöhe. Außerdem spielt die Auflösung der Kamera ein Rolle, diese ist 640*480p (3.1.4)

Von den gefilterten Rechtecken wird nun der Mittelpunkt berechnet. Ist mit einem Umkreis von 10 Pixeln bereits ein weiteres Rechteck gefunden worden, so wird das Rechteck verworfen. Dadurch ist sicher gestellt, dass Rechtecke nicht doppelt erkannt werden. Erfüllt ein Rechteck all diese Bedingungen so wird noch die Rotation der oberen Kante berechnet. Diese wird dann zusammen mit den Mittelpunktkoordinaten an den Controller über ROS weitergeleitet. Es ist das Problem aufgetreten, dass das Viereck um 45° gedreht war. In diesem Fall gibt es 2 oberste kanten und der Algorithmus erkennt schlimmstenfalls abwechselnd beide. Um dies zu vermeiden, wurde eine 2 Regel eingeführt. Es wir immer die oberste Kante genommen. Gibt es 2, so wird die Linke davon genommen. Da es nicht 2 Oberste und Linkeste geben kann, ist der Algorithmus so eindeutig definiert und terminiert.[**?**]

Um die Rechenzeit zu verringern, wird das Bild auf 640x480 Pixel herunter skaliert, da dies für einfache Vierecke völlig ausreicht. Außerdem wird der Hough Algorithmus bewusst erst nach dem Filtern ausgeführt, da seine Komplexität fakultativ mit der Anzahl der erkannten Objekte ansteigt. Auf diesem Wege ist eine Auswertung mit mehr als 5 Frames pro Sekunde möglich, was unseren Anforderungen genügt.

Der Bildverarbeitungsprozess läuft in der Image Progressing Node. Er wird maximal mit 10 Hz ausgeführt. Das stellt sicher das der Pi nicht überlastet und noch genügend Rechenkapazität für die anderen Codes/Prozesse hat. Da die Drohne nicht zu schnell fliegt, reicht dieses Schrittweite aus um ein stabiles Systemver-

halten des PT1-Glied(Regler mit proportionaler Vergrößerung erster Ordnung). Es wurde experimentell in der Gazebosimulation herausgefunden das bei dem PX-4 Controller sogar eine Geschwindigkeit von 3 HZ ausreichend würde, da die Änderungsraten klein sind 0.1. Diese 0.1 steht dafür das, wenn der Regler eine Abweichung von 1 m erkennt, er der Drohne als neue Zielkoordinaten nur Koordinaten gibt, die 0,1 m entfernt sind. Dadurch fliegt sie langsamer und kippt nicht so stark. Durch das kippen wird die Kamera bewegt und ändert somit ihren Winkel auf den Boden. Dadurch wiederum wird das Paket falsch erkannt. (siehe Kapitel 7)

Kapitel 6

Gesamtintegration

6.1 Sensordatenverarbeitung

Prinzipiell gibt es 3. Sensoren. Zum einen der Ultraschallsensor, um die Höhe genau und jederzeit bestimmen zu können. Außerdem gibt es einen Kamerasensor, um die Position des Paketes zu bestimmen. Das dritte Sensorsystem ist da, um die Position der Drohne zu bestimmen. Dieses wird vom Institut für Technische und Numerische Mechanik gestellt. Es ist für das fliegen der Drohne existenziell wichtig.

Leider ist ein Fliegen der Drohne ohne dies nicht möglich, weil die Drohne mit globalen Koordinaten arbeitet. Das heißt, dass System bestimmt die Position der Drohne im Raum, anschließend berechnet der Flight Controller die nächste Sollposition und schickt sie an den PX4-Controller. Dieser regelt dann seine Lage dem entsprechend. Der Vorteil an diesem Vorgehen ist, dass die Drohne leicht auf eine externe Positionsbestimmung(z. B. GPS) umgestellt werden kann. Mit dem Ultraschallsensor ist eine genaue Höhenbestimmung immer noch möglich. Der Nachteil ist, dass man die Drohe ohne das System nicht fliegen kann. Die Umstellung auf GPS oder Galileo wäre z. B. ein denkbares Folgeprojekt. Mehr dazu ist unter Auswertung und Fazit zu finden.

Ein weiteres wichtiges Merkmal ist die Einheit der Sensordaten. Zu dem der Ultraschallsensor liefert dabei sein Ergebnis in mm. Näheres dazu findet man unter dem Kapitel “Sensorik, Aktorik und andere Hardware”.

Die Kamera bestimmt die Bildposition in Pixel. Abhängig vom Öffnungswinkel der Kamera ist die Position des Paketes daraus zu berechnen. Die Formel für die

x/y-Koordinate ist:

$$x = \frac{x^*}{a^*} \tan(\frac{a}{2})h$$
$$y = \frac{y^*}{b^*} \tan(\frac{b}{2})h \tag{6.1}$$

wobei:
a/b* = Bildpunkte in x/y Richtung
x/y^* = erkannte Punkte
a/b = Öffungswinkel

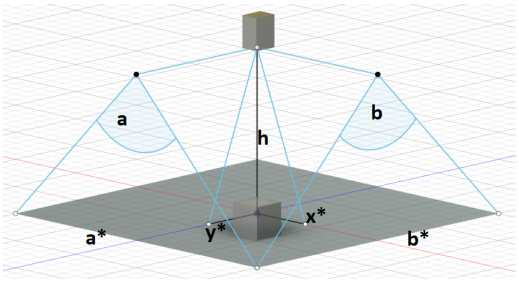

Abbildung 6.1: Kamerawinkel

6.2 Programmablauf

Prinzipiell ist jede Node gleich aufgebaut. Zuerst gibt es eine Initialisierung und Verbindung mit der Masternode. Anschließend deklariert die Node welche Streams die entsprechende Node benötigt und welche nicht. Nun wird gewartet, bis eine neue Nachricht auftaucht und die diese wird dann verarbeitet. Geprüft wird in diesem Fall mit einer Rate von 20 Hz.

Der Flight-Controller verhält sich jedoch leicht anders. Die PX-4 muss erst in den Offboard-Modus versetzt werden. Dieser bedeutet, dass die Drohne extern gesteuert werden kann. Außerdem muss der Arming-Modus aktiviert werden. Damit sie Befehle annimmt. Leider hat das einmalige Senden nicht gereicht. Sie hat nicht zuverlässig beide Befehle erkannt. Damit dies gewährleistet ist, sendet der Pi nun aller 5 Sekunden den Offboard Befehl, solange bis sich der Status der Drohne geändert hat, sodass sie sich nun im Offboardmodus befindet. Anschließend sendet der Controller das Arming Signal, solange bis die Drohne sich im Arming Modus befindet.

Wichtig ist das gleichzeitig mit 20 Hz, eine Sollposition gesendet werden muss, damit die Drohne im nicht wieder aus dem Offboardmodus geht. Dies hat am Anfang große Probleme bereitet, da man so schnell die Kontrolle verliert.

Anschließend wird zuerst eine Sollposition angeflogen. Aus dieser Position muss die Drohne, das Paket sehen. Die Abbildung 6.2 zeigt links oben das gefilterte Bild, links unten das Bild der Kamera mit dem erkannten Paket und rechts ein Bild der Drone in der Simulation.

Anschließend richtet sich die Drohne mithilfe der Kamera mittig über dem Paket aus. Dabei vergleicht der Kontroller die aktuelle Position mit der Paketposition. Die Paketposition wird dabei mithilfe der Formel(siehe Sensorerfassung) berechnet. Um keine zu schnellen Änderungen zu fliegen wird ein P-Regler verwendet. Ist das Paket mittig und die Drohne nicht zu schnell(dies wird mithilfe der alten Position und der neuen Position bestimmt) so sinkt die Drohne. Hat sie eine Höhe von 60 cm erreicht, so dreht sie sich und richtet sich entlang des Paketes aus. Dabei verfährt sie nach dem gleichen Verfahren wie bei der Positionsausrichtung und wartet immer, bis das Paket wieder mittig ist. Siehe Abb. 6.3

Ist die Verdrehung kleiner als 3 Grad, sinkt sie weiter.

Ist sie nur noch 20 cm über dem Boden, so hat sie das Paket gefunden und schließt den Greifer. Anschließend könnte sie wieder abheben und das Paket zur Zielposition bringen. Der Prozess ist in dem unteren Strukturgramm Abb. 6.5 dargestellt. In der Simulation hält sie ihr Position weiterhin. Siehe Abb,

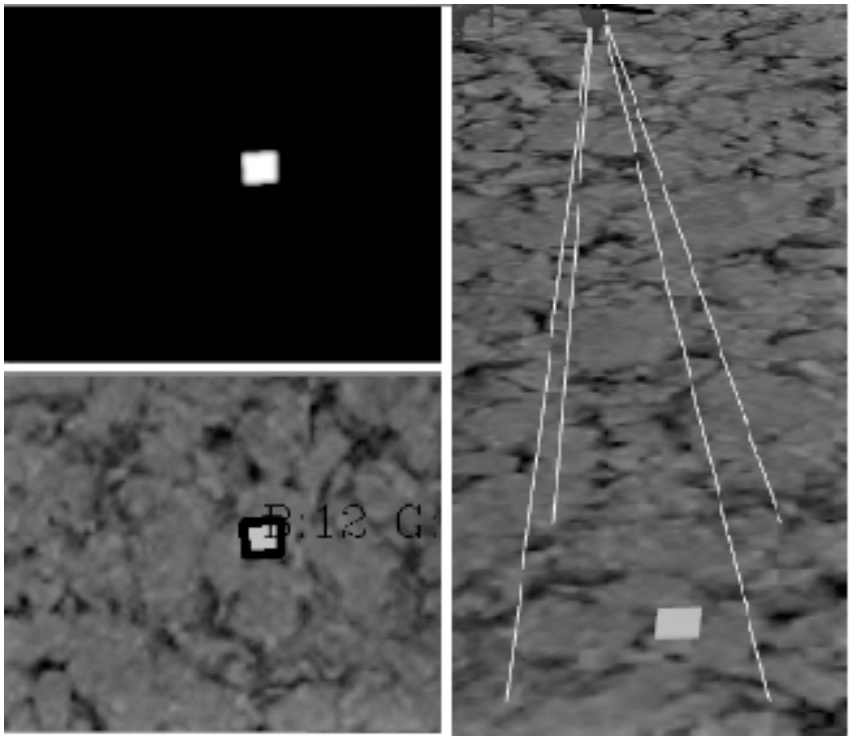

Abbildung 6.2: Simulation der Drohne

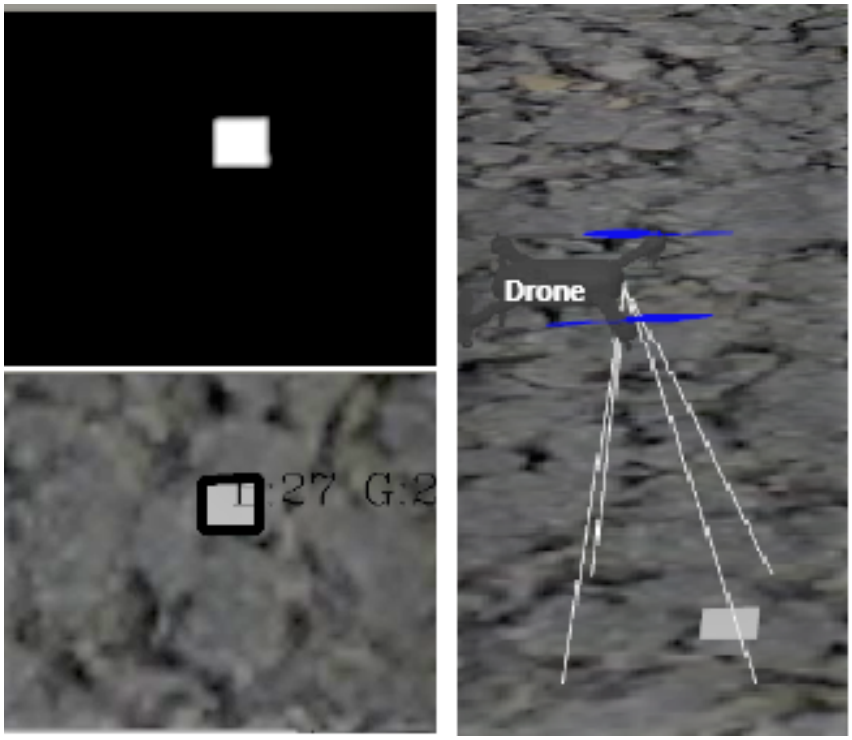

Abbildung 6.3: Ausrichten über dem Paket

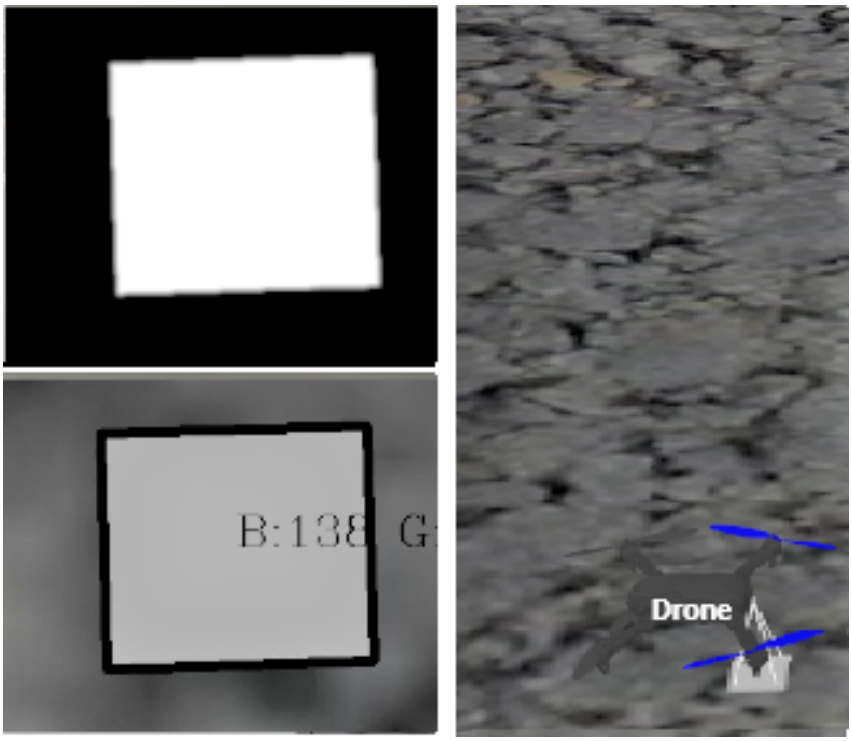

Abbildung 6.4: Halten der Position

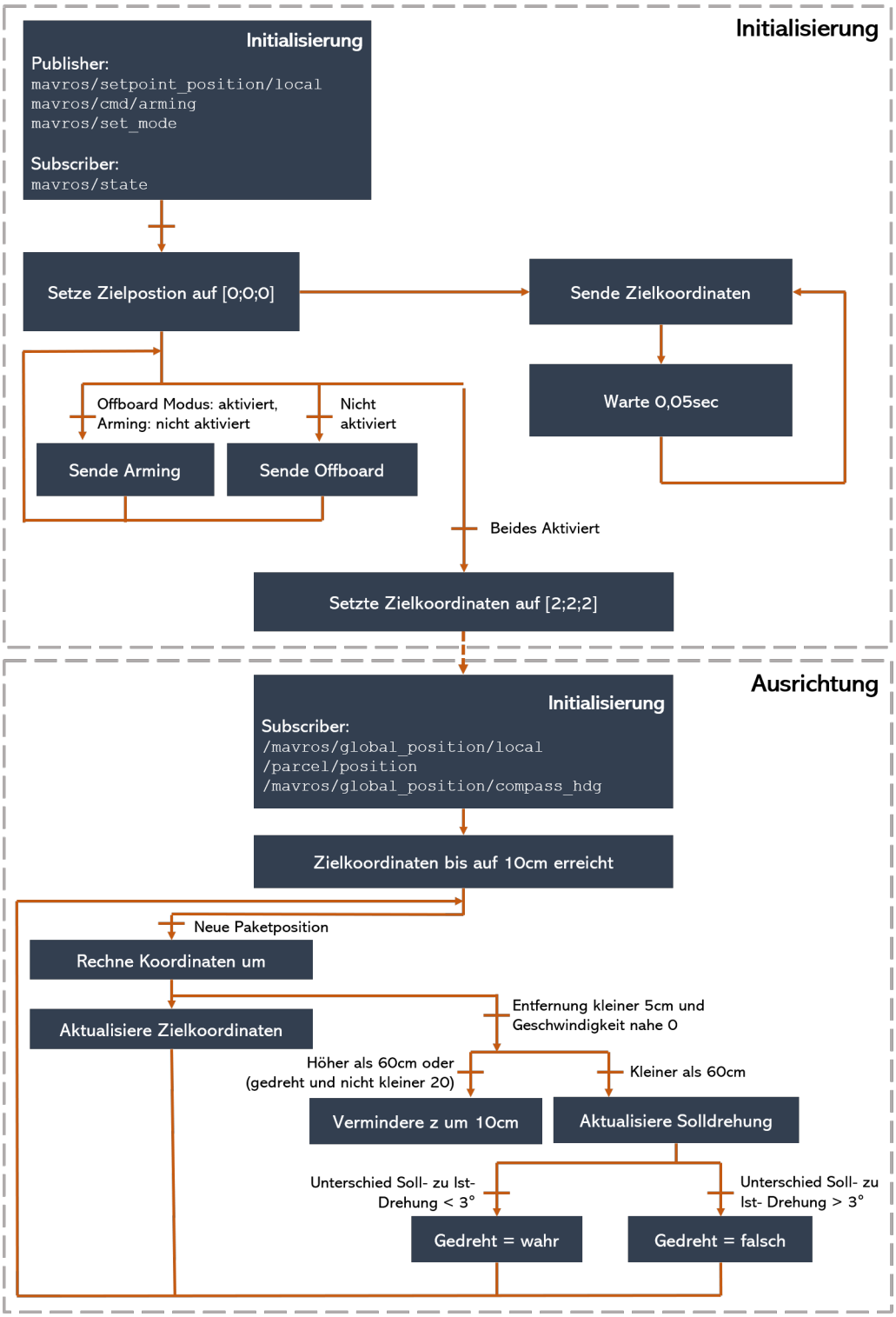

Abbildung 6.5: Funktionsablaufdiagramm

Kapitel 7

Fazit und Ausblick

Zusammengefasst ist zu erkennen, dass eine Drohnenregelung durchaus auch mit einfachen Mitteln machbar ist. Besonders überraschend war jedoch, dass die Bilderkennung nicht so gut funktioniert wie gedacht. Es ist sehr schwierig einzelne äußere Faktoren zu Eliminieren und immer zu dem „richtigen“ Ergebnis zu kommen. Jedoch funktioniert das Ansteuern des PX-4 Controllers erstaunlich problemlos. Es hat uns große Schwierigkeiten bereitet die Umgebung zu installieren da diverse Packages manuell nachinstalliert werden müssen. Gerade den Catkin-WS zu erstellen und in diesem die richtigen Pakete zu installieren hat sehr viel Zeit in Anspruch genommen. In den nächsten Wochen wollen wir die Software dann vollständig integrieren und testen. Wenn diese Test's erfolgreich verlaufen so ist die Projektarbeit damit abgeschlossen.

Trotzdem bleiben einige Themen leider offen. Vielleicht für nachfolgende Gruppen. Die größten softwareseitigen Probleme sind zum einen die Bilderkennung, dort wären mögliche Lösungsansätze das Auslagern des Prozesses der Bilderkennung. Dadurch könnte massiv Laufzeit gewonnen werden, weil dieser Anspruchsvolle teil auf einem Externen leistungsfähigeren Rechner gemacht werden könnte. Weiterhin könnte die Bilderkennung dahin verbessert werden das zusätzliche starke Lampen an der Unterseite befestigt werden, um das Paket bei verschiedenen Belichtungen erkennen zu können.
Ein weiterer Punkt ist das Herausrechnen des Drohnenwinkels bei der Bestimmung der Position. Der Vorteil dabei wäre, dass man deutlich schneller fliegen könnte, da die Drohne sich mehr neigen könnte.

Insgesamt könnte noch einiges an Laufzeit gewonnen werden, indem man Prozesse beschleunigt, jedoch sind dafür ausführliche Test notwendig. Am Anfang des Projektes war es angedacht ein GPS-Modul zu integrieren, um die Drohne auch draußen fliegen zu können. Leider ging dies Sicherheitsbedingt

nicht. Trotzdem wäre es sinnvoll, da der Einsatz der Paketdrohne erst draußen richtig sinnvoll ist.

Trotz aller dieser Punkte die noch Fehlen, ist der Parcelcopter ein gutes Stück weiterentwickelt worden. Sowohl Hardware technisch mit einem neuen Greifer als auch Softwareseitigen mit einer Bilderkennung und Drohnensteuerung.

Anhang

A.1 ROS auf Ubuntu

Für eine effektive und effiziente Entwicklung des Softwaresystems ist die Nutzung geeigneter unterstützender Software unabdingbar. Daher soll nun die verwendete Toolchain kurz vorgestellt werden.

A.1.1 Setup

Sowohl die Entwicklung als auch die Runtime läuft auf Ubuntu 18.04. Das gilt also sowohl für Entwicklungsrechner, als auch für den Raspberry Pi, wobei bei diesem Ubuntu Mate eingesetzt wird. Der Vorteil ist, dass ROS nativ darauf funktioniert und sehr viele Bibliotheken für ROS bereits vorkompiliert zur Verfügung stehen. Es ist dabei für die aktuelle ROS-Version "Melodic"dringend davon abzuraten, das System auf einer Raspian-Installation laufen zu lassen, da man einerseits alle Pakete selber kompilieren muss und darüber hinaus viele der Abhängigkeiten von MAVROS nicht zu Verfügung stehen. Wichtig ist auch, dass "Melodicäusschließlich von der Ubuntu-Version 18.04 unterstützt wird. Die Installation von ROS lässt sich auf Ubuntu nach Hinzufügen des Repositories über

```
1 $ sudo apt install ros-melodic-ros-base
```

einfach durchführen. Hierbei ist es empfehlenswert für den Raspberry Pi die

```
1 $ ros-base
```

Version und für den Entwicklungsrechner die

```
1 $ desktop-full
```

Version zu nehmen, da diese bereits die wichtigsten grafischen Werkzeuge installiert hat. Nachdem man nun noch rosdep initialisiert, welches das Arbeiten mit Softwareabhängigkeiten deutlich vereinfacht, müssen nun die mitinstallierten ROS-Pakete im aktuell genutzten Terminal mit Hilfe des

```
$ source
```

Befehls geladen werden. Bei einer Standardinstallation geht das mit

```
$ source /opt/ros/melodic/setup.bash
```

. [?]

A.1.2 Workspaces

Alle eigenen Entwicklungen finden im sogenannten catkin-Workspace statt, worin sich die eigenen Pakete befinden. Dieser kann im home-Verzeichnis des Benutzers nach Erstellen des Verzeichnisses "catkin_ws"mit

```
$ catkin_make
-DPYTHON_EXECUTABLE
=/usr/bin/python3
```

erstellt werden. Dabei muss

```
$ catkin_make
```

jedes mal ausgeführt werden, wenn ein neues Paket erstellt wurde. Das kann mit dem Befehl

```
$ catkin_create_pkg my-package-name
std_msgs rospy roscpp
```

erreicht werden. Dadurch wir ein gleichnamiger Ordner im “src”-Ordner des Workspaces erstellt, in dem die Sourcedateien abgelegt werden können. Um die neuen Pakete nun auch in ROS zu Verfügung zu haben, müssen diese auch im Terminal geladen werden. Das geht analog zum Laden der vorinstallierten Pakete mit

```
$ source ~/catkin_ws/devel/setup.bash
```

.

A.1.3 .bashrc

Um nicht bei jedem neuen Terminal erst die Pakete manuell laden zu müssen, bietet Ubuntu die Möglichkeit dies automatisch zu tun. Dafür ist die Datei .bashrc zuständig, die sich im home-Verzeichnis eines jeden Benutzer befindet. In diese können die beiden Befehle einfach angehängt werden. Auch die Konfiguration

für Gazebo (siehe Kapitel 6) kann hier bereits erfolgen. Es sei aber zu erwähnen, dass dies nur so lange sinnvoll ist, wie ROS hauptsächlich auf dem System genutzt wird.

A.2 Entwicklungsumgebung und Versionsverwaltung

Eine (integrierte) Entwicklungsumgebung sollte den Entwickler in seiner Arbeit unterstützen und ihm sich wiederholende oder logisch einfache, aber zeitaufwändige Schritte abnehmen. Für diese Entwicklung fiel daher die Wahl auf "PyCharm"der Firma JetBrains. Dieses bietet neben dem obligatorischen Texteditor mit integriertem Auto-Complete und Compiler Vorschläge zu Codeverbesserung Refactoring etc. Es macht hier jedoch Sinn sich eine der ROS-Erweiterungen für PyCharm zu installieren, damit die ROS-eigenen Python Bibliotheken auch korrekt erkannt werden. PyCharm arbeitet zudem mit Virtual Environments (venv), also einer abgekapselten, meist projektspezifischen Python-Installation, die nur die zusätzlichen Bibliotheken enthält, die für das aktuelle Projekt benötigt werden. Diese sollte bei allen Entwicklern identisch sein, um Versionsinkompatibilitäten zu vermeiden. Ein weitere Funktion von PyCharm ist die integrierte Versionsverwaltung, kurz VCS (Version Control System). Diese bietet eine direkte Anbindung an Git, das Versionen des Sourcecodes verwaltet und Teilentwicklungen in den verschieden Entwicklungszweigen (branch) der Entwickler effektiv zum Hauptzweig (master) zusammenbringt (merge). Konflikte, wenn beispielsweise eine Datei von zwei zu vereinenden Zweigen bearbeitet wurden, müssen jedoch oft von Hand gelöst werden. Deshalb macht es Sinn Funktionalitäten weitestgehend in einzelne Dateien zu unterteilen.

A.3 Inhalt der CD-ROM

Die beigelegte CD-ROM enthält in der obersten Dateistruktur die Einträge

- **stud_26.pdf**: das PDF-File zur Studienarbeit STUD–26.
- **STUD_26/**: ein Verzeichnis mit den TEX-Dateien des in LaTeX verfassten Berichtes zur Studienarbeit STUD–26 sowie alle dazugehörigen Grafiken als *.eps und *.svg Dateien.
- **DATA/**: ein Verzeichnis mit den für diese Arbeit relevanten Daten, Hilfsprogrammen, Skripts und Simulationsumgebungen.

Zusätzliche Informationen stehen in den readme.txt-Dateien der jeweiligen Verzeichnisse zur Verfügung.

Erklärung

Hiermit versichere ich, dass

- ich die vorliegende Arbeit selbständig verfasst habe,
- ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- ich die eingereichte Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe,
- das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Datum	Unterschrift