hochschule 21
genial dual

Technical Report Nr. 4
hochschule 21

Einsatz einer Flugdrohne zur Objektlokalisation im Kontext des DLR Spacebot Cup 2013

Christoph Grefe und Thorsten Hermes

hochschule 21 - Technical report, Nr. 4
2015

hochschule 21- Technical report

Herausgeber:
hochschule 21 gemeinnützige GmbH
Staatlich anerkannte private Fachhochschule
Harburger Straße 6

21614 Buxtehude

Telefon: +49 4161 648 0
Fax: +49 4161 648 123
E-Mail: info@hs21.de
http://www.hs21.de

ISSN 2196-5153

Einsatz einer Flugdrohne zur Objektlokalisation im Kontext des DLR SpaceBot Cup 2013

Christoph Grefe & Thorsten Hermes
hochschule 21 gGmbH
21614 Buxtehude
christhoph.grefe@stud.hs21.de & *hermes@hs21.de*

29. Januar 2015

1 Einleitung

Ausgangspunkt für diese Arbeit ist ein Wettbewerb, der vom deutschen Luft- und Raumfahrtzentrum (DLR) 2013 ausgeschrieben wurde. Es handelte sich bei diesem Wettbewerb um einen Wettkampf verschiedener Robotersysteme. Diese sollten in einer unbekannten Umgebung, die zum Beispiel einen fremden Planeten simuliert, unterschiedliche Aufgaben erfüllen. Diesen Aufgaben umfassten, unterschiedliche Objekte zu finden und zu transportieren. An einem vorgegeben Zielort sollte eine Montage der Objekte stattfinden. Weiterhin wurde eine Kartierung des unbekannten Geländes gefordert. Für die Erfüllung der genannten Aufgaben wurde eine Zeitvorgabe von einer Stunde festgelegt, welche nicht überschritten werden darf. Die Aufgaben mussten von dem Robotersystem weitestgehend autonom ausgeführt werden. Zusätzlich war jedoch, ebenfalls in Anlehnung an das Szenario eines fremden Planeten, eine Bodencrew vorgesehen. Die Bodencrew befand sich während der Zeit des Einsatzes des Robotersystems in einem getrennten Raum und hatte in bestimmten Zeitintervallen die Möglichkeit, Befehle an das Robotersystem zu senden. Die Kommunikation war allerdings nur mit einem Delay von zwei Sekunden möglich.

Abbildung 1: Team SpaceBot21

Das Team „SpaceBot 21“ (Abbildung 1), das aus Studenten und Professoren der hochschule 21 bestand, hatte für diesen Wettbewerb ein Konzept für System aus mehreren Robotern entwickelt. Ein Rover und eine Flugdrohne sollten zusammenarbeiten, um die Aufgaben in einem möglichst kurzen Zeitraum bewältigen zu können. Der Rover stellt dabei das Hauptsystem dar, auf dem sich ein Computer mit ausreichender Rechenleistung befindet, um auch die anderen Bestandteile des Systems verwalten zu können. Die Drohne erhält ihre Befehle vom Rover und meldet diesem alle Informationen

zurück, die über unterschiedliche Sensoren aufgenommen werden. Die Hauptaufgabe der Drohne besteht in dem Auffinden der Objekte, die sich an mehreren Stellen des unbekannten Terrains befinden. Die Objekte können durch unterschiedliche Farben selektiert werden. Nach dem Auffinden soll die Drohne die Position der Objekte hinsichtlich einer zu erstellenden Karte an den Rover weitergeben.

Dieser Bericht behandelt die Programmierung der Drohne. Es werden die verwendeten Plattformen und Entwicklungsumgebungen beschrieben. Zusätzlich wird auf die Software-Pakete eingegangen, die zur Ansteuerung der Drohne verwendet wurden. Das Hauptaugenmerk des Berichts liegt dabei auf dem Programm, welches für den Ablauf des Wettbewerbs entwickelt und umgesetzt wurde.

2 Software

Die AR.Drone 2.0 ist zur Freizeitbeschäftigung entwickelt. Die Software, die zur Steuerung der Drohne von der Firma Parrot zur Verfügung gestellt wird, ist primär eine App, die unter Android und iOS auf mobilen Geräten läuft. Zusätzlich existieren für die AR.Drone 2.0 Entwicklungspakete, die einen weitaus größeren Umfang an Möglichkeiten bieten, als die App.

Deutlich mehr Möglichkeiten bietet das Source Developement Kit (SDK), das sowohl für Windows, als auch für Ubuntu zur Verfügung steht. Zu Beginn des Projektes wurde zunächst das SDK für Windows verwendet. Allerdings wurde zu einem frühen Projektzeitpunkt beschlossen, die Ubuntu-version zu verwenden. Zum einen, weil die zu dem Zeitpunkt aktuelle Version des SDK (2.0.1) offenbar einen Fehler enthielt und somit im Verlauf des Projektes nicht mehr online zur Verfügung stand. Zum anderen bereitete die Steuerung über eine PC Tastatur Probleme, da zwar die Steuerung über einen externen Game-Controller realisiert wurde. Der entscheidende Grund für den Wechsel lag jedoch darin, dass das Betriebssystem des Rovers ebenfalls zu Ubuntu gewechselt wurde. Da die Verwendung von unterschiedlichen Systemen zu einem späteren Zeitpunkt einen sehr hohen Aufwand bei der Kommunikation bedeutet hätte, wurde auch die Entwicklungsumgebung für die Drohne umgestellt.

Im weiteren Verlauf des Projektes lief auf dem Hauptrechner Ubuntu 12.04. Für die Kommunikation und die Ausführung einiger Programme wurde außerdem das Robot Operation System (ROS `http://www.ros.org`) verwendet. ROS ist ein Framework, das bei der Entwicklung von Robotersoftware hilft. Es enthält eine Vielzahl verschiedener Bibliotheken. Außerdem stellt es eine einfache Möglichkeit dar, um die Kommunikation in geschlossenen Systemen auf kurzen Wegen zu erleichtern. ROS wurde folglich für die Kommunikation zwischen dem Hauptrechner und der Drohne eingesetzt. ROS enthält bereits einen Treiber für die Kommunikation mit der Drohne. Er ist unter dem Namen „ardrone autonomy driver“ zu finden. Dieser Treiber basiert auf dem SDK und enthält sehr ähnliche Funkionen. Erhältlich ist der Treiber kostenlos von der Seite `https://github.com`. Die Seite kann zur Verwaltung von Softwareprojekten genutzt werden. Außerdem kann Quellcode offen zur Verfügung gestellt werden. Zusätzlich gibt es eine ausführliche Dokumentation, die sowohl die Installation, wie auch die Programmierung mit Hilfe des Treibers beschreibt.

Die Software für die Drohne wurde in der Entwicklungsumgebung Eclipse geschrieben. Eclipse bietet den Vorteil, dass die Dokumente, Abhängigkeiten und Funktionen sehr übersichtlich dargestellt und verwaltet werden. Die Projekte, die in ROS erstellt werden, können in Eclipse eingebunden werden. Somit kann man in der Entwicklungsumgebung mit der gleichen Struktur arbeiten, wie im Dateisystem auch. Zum Kompilieren der Projekte wird jedoch nicht der Compiler von Eclipse genutzt. Um eine vollständige Kompatibilität zu gewährleisten, wurde der in ROS integrierte Compiler verwendet.

2.1 SDK für Windows

Das Source Developement Kit ist eine Softwareumgebung, die den Zugriff und die Steuerung der AR.Drone 2.0 sehr einfach gestaltet. Dennoch bietet es eine große Bandbreite an Möglichkeiten. Die Oberfläche des SDK ist auf der 2 dargestellt.

Zu Beginn der Entwicklungsphase wurde das SDK in der Version 2.0.1 unter Windows installiert und eine Ansteuerung der Drohne erprobt. Die manuelle Steuerung der Drohne mit einem Gamepad stellt keine Probleme dar. Auch die Rückmeldung der Daten während des Flugs an die Software und die Darstellung über die Oberfläche des SDK funktionieren. Speziell die Verwendung der Bilddaten der Frontkamera ist für die Aufgabe der Drohne in dem Projekt entscheidend.

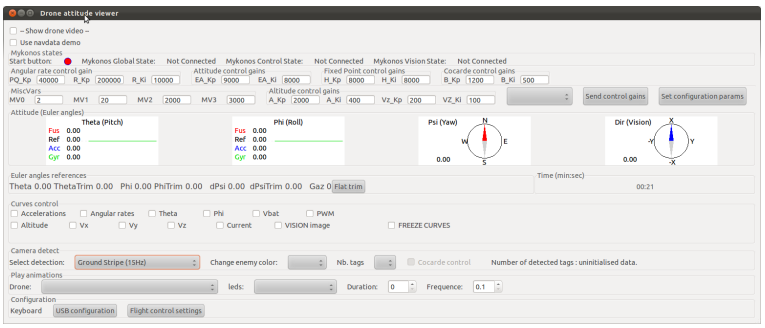

Abbildung 2: SDK Oberfläche

Bereits der Versuch, die Drohne über die Tastatur zu steuern, zeigte jedoch, dass das SDK zu dem Zeitpunkt der Verwendung noch nicht ausgereift war, da hier einige Probleme auftraten. Zu dem Zeitpunkt der Tests wurde das Betriebssystem des Hauptrechners, der sich in dem Rover befindet, von Windows auf Ubuntu 12.04 umgestellt. Somit wurde auch die Umgebung für die Drohne auf Ubuntu umgestellt. Dies war und ist sinnvoll, da die Verarbeitung der Daten, die die Drohne liefert auf dem Hauptrechner des Rovers erfolgen soll. Außerdem erleichtert dies die Kommunikation zwischen der Drohne und dem Hauptsystem.

2.2 Ubuntu und ROS

Mit der Entscheidung, Ubuntu als Betriebssystem zu verwenden, stellte sich die Frage, wie die Daten, die von den unterschiedlichen Sensoren, welche zum Teil von einem eigenen System ausgelesen werden (Raspberry Pi), zusammengeführt werden sollten. Hierfür wurde das Software-Framework ROS (Robot Operation System) ausgewählt. Für die Entwicklung der Drohne wird die Version „Groovy Galapagos“ verwendet. Genauere Informationen zu dieser Version sind unter `http://wiki.ros.org/groovy` zu finden. ROS ist dabei speziell auf die Entwicklung von Software für einzelne Roboter oder ganze Robotersysteme ausgelegt. ROS ist eine Sammlung von Tools und Bibliotheken, die viele Aufgaben, die für Robotersysteme wichtig sind, übernehmen oder erleichtern. Es wird auch die Kommunikation zwischen unterschiedlichen Informationsquellen abgedeckt. Allerdings ist ROS nur für Systeme ausgelegt, die eng miteinander verbunden sind. Die Kommunikation ist problematisch, sobald größere Entfernungen und als Folge auch höhere Latenzzeiten zu überbrücken sind (vgl. [6]). Für das Projekt bedeutet das, dass ROS die Kommunikation zwischen dem Rover, den einzelnen Sensoren und der AR.Drone übernehmen kann. Die Kommunikation des Robotersystems mit der simulierten Bodenstation, für die eine Latenzzeit von zwei Sekunden vorgegeben ist, muss jedoch anders realisiert werden.

Neben der Kommunikation liefert ROS ein Tool zur Visualisierung von dreidimensionalen Daten, die durch die Sensoren der Roboter geliefert werden. Das Tool „rviz“ stellt diese Vielzahl an Informationen bei Bedarf in einem Fenster dar und ermöglicht somit, ein Verständnis dafür zu entwickeln, wie die Sensoren des Roboters zusammenarbeiten. Neben den Daten, die durch die Sensoren aufgenommen werden, kann auch ein Nachbau des Roboters selber dargestellt werden (s. [6]), der sich mit Hilfe der Sensoren in der Umgebung selber lokalisiert. Dies ist auch bei der Fehleranalyse sehr hilfreich. Auf diese Weise kann kontrolliert werden, ob die Positionen und Entfernungen der Sensoren im Verhältnis zum Roboter richtig programmiert sind.

Auf der Abbildung 3 sind zwei Darstellungen zu erkennen, die der Rover von seiner Umgebung aufgenommen hat. In der Mitte der Darstellungen ist der Roboter durch mehrere verknüpfte Knotenpunkte dargestellt, die zuvor definiert wurden. Da sowohl der Rover, als auch die Drohne in einem festgelegten Abstand voneinander starten, ist es möglich, dass die Drohne gefundene Objekte in einer solchen Karte des Rovers positioniert. Sie ermöglicht dem Rover somit, diese Punkte direkt anzufahren.

2.3 Ardrone Autonomy Driver

Nach der Umstellung des Betriebssystems auf Ubuntu und der Einführung von ROS wurden Pakete gesucht, die die Steuerung der Drohne ermöglichen. Die Recherche zu diesem Thema ergab, dass es

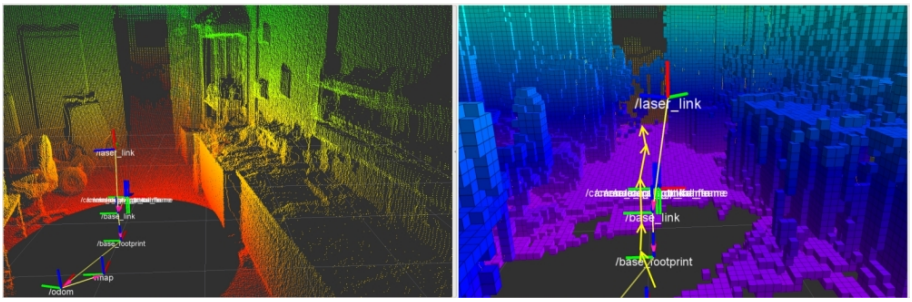

Abbildung 3: Beispiel für die Visualisierung mit RVIZ

neben dem SDK für Ubuntu ebenfalls eine Bibliothek von ROS gibt, die speziell für die AR.Drohne entwickelt wurde, der „ardrone autonomy driver“. Die Bibliothek basiert auf dem SDK, verwendet jedoch nicht die Oberfläche (vgl. Abbildung 2). Der Vorteil des Treibers im Vergleich zu dem reinen SDK, besteht darin, dass der Treiber die unter ROS übliche Kommunikation verwendet, die auch auf dem Rover verwendet wird. Somit können die Daten der Drohne ohne weiteren Verarbeitungsschritt ausgewertet werden [5].

Die Tests zeigten, dass der Treiber leicht zu verwenden ist und ohne großen Aufwand die gewünschten Ergebnisse lieferte. Da für den Treiber eine ausführliche Dokumentation zur Verfügung steht, fiel die Entscheidung, den Treiber für die Ansteuerung der Drohne zu verwenden.

3 Programmierung

Die Drohne hatte vielfältige Aufgaben abzuarbeiten. Um diese Aufgaben zu organisieren, wurden mehrere Teilprogramme verwendet. Hierzu mussten zunächst die Aufgaben und Vorgänge genau definiert werden. Im Folgenden sind das die Aufgaben: *a)* Die Drohne muss auf ein Kommando des Rovers starten. Anschließend fliegt sie *b)* einen vorgegebenen Pfad ab. Während des Fluges müssen *c)* die Bilder der Kamera ausgewertet werden. *d)* Auf den Bildern wird nach den Farbwerten gesucht, die zuvor einprogrammiert wurden. Bei den Farbwerten handelt es sich um einen Blau- und einen Gelbton für die zwei Objekte und um einen Rotton für die Station, an der die Objekte montiert werden müssen. *e)* Nachdem die gesamte Route abgeflogen wurde, muss die Drohne wieder zu ihrem Ausgangspunkt zurückkehren und dort landen. Sofern ein Objekt gefunden wird, muss *f)* die Position des Objektes veröffentlicht werden. Zusätzlich muss *g)* der zurückgelegte Weg veröffentlicht werden, da am Ende des Wettbewerbs eine Karte abgegeben werden soll, auf der sowohl die gefundenen Objekte, wie auch der zurückgelegte Weg markiert sind.

3.1 Kommunikation (Node / Topic / Marker)

Das ROS-Framework bietet eine einfache Methode zur Kommunikation verschiedener Bestandteile eines Systems. Damit die Kommunikation funktioniert, ist ein „Master“ notwendig, der die Kommunikation verwaltet. In dem System ist das der Hauptrechner, der sich auf dem Rover befindet. Jeder Teilnehmer meldet sich an dem Master an. Anschließend kann der Teilnehmer einen „Node“ erstellen. Dieser kann dann Nachrichten an den Master senden („publish“) oder Nachrichten vom Master empfangen („subscribe“). Das Senden von Nachrichten geschieht durch sogenannte „Topics“. Dieses System wird von dem gesamten Robotersystem genutzt. Jeder Mikrocontroller (Raspberry Pi) und auch die Drohne erstellen eigene Nodes. Da diese über den Master verwaltet werden, kann jeder andere Teilnehmer auf die Topics zugreifen, die ein Node sendet. In der Abbildung 4 sind die Topics und die jeweils zu Grunde liegenden Nodes dargestellt, die die Drohne erstellt, um ihre Informationen an das Hauptsystem weiterzugeben. Allein dieser kleine Ausschnitt der Nodes und Topics verdeutlicht, wie umfangreich und verzweigt die Kommunikation des ganzen Systems ist. Eine Übersicht, wie auf Abbildung 4 dargestellt, hilft dabei, die Verbindungen zwischen den einzelnen Teilsystemen zu verstehen. In der Darstellung sind die Nodes durch große Rechtecke dargestellt. In jedem großen Rechteck befinden sich weitere kleine Rechtecke, die alle Messages darstellen, die von dem Node gesendet werden. Die

Ovale stellen die Topics dar. Die Pfeile verdeutlichen den Datenfluss zwischen den Topics und den Nodes [2].

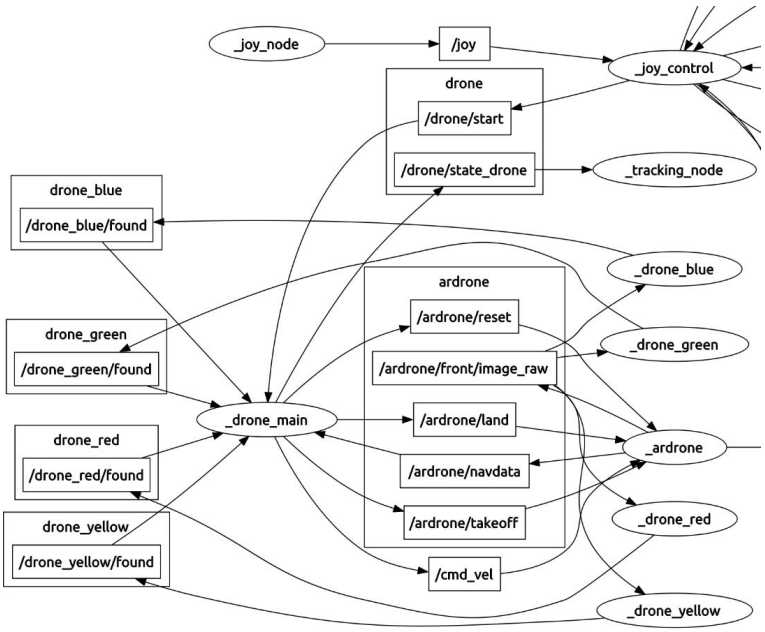

Abbildung 4: Ausschnitt aus einer Übersicht der Topics für die Drohne

In ROS stehen außerdem Marker zur Verfügung. Ein Marker ist ein Objekt, dem unterschiedliche Eigenschaften zugeordnet werden können. Neben einer ID und einem Zeitstempel, die jeden Marker einzigartig machen, können auch Positionen und Orientierungen zugeordnet werden. Diese Eigenschaften werden von der Software für die Drohne genutzt, wenn die Bildverarbeitung ein Objekt erkannt hat. Die Software erstellt dann einen Marker, der die Farbe des erkannten Objektes und die aktuelle Position der Drohne enthält. Dieser Marker wird anschließend in eine Topic gepublisht. Auf diese Weise kann der Rover auf den Marker zugreifen und die Position als nächste Zielposition verwenden. Visualisiert man die Marker mit rviz, kann dies aussehen wie in Abbildung 8 (ein gelber und ein roter Marker). In der Abbildung 5 ist links eine Höhenkarte zu sehen, in die zwei Marker eingezeichnet sind (grün und blau). Auf der rechten Seite der Abbildung 5 ist der Pfad zu sehen, den der Rover geplant hat [2].

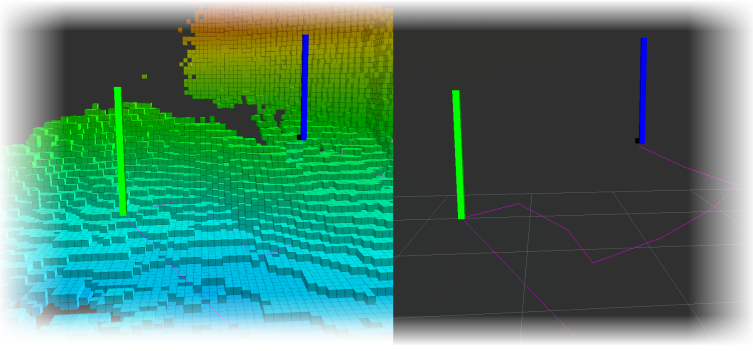

Abbildung 5: Pfadplanung Rover

3.2 Programmablauf

Die Aufgaben werden in vier Programme aufgeteilt. Das Hauptprogramm „fliegen“ (Abschnitt 3.2.1) enthält die autonome Steuerung der Drohne. Für den Fall, dass der Fehler in der Position der Drohne so groß ist, dass die Drohne den Zielpunkt nicht finden kann, lässt das Programm „search home“ (Abschnitt 3.2.2) eine halbautonome Steuerung zu. Das Programm „suchen“ (Abschnitt 3.2.3)ist für die

Bildverarbeitung der Frontkamera zuständig. Das Programm publisht die Positionen der gefundenen Objekte. Das Programm „tracking“ (Abschnitt 3.2.4) ruft in regelmäßigen Abständen die aktuelle Position der Drohne ab und speichert die Koordinaten in einem Format ab, welches von rviz zu einem späteren Zeitpunkt zur Visualisierung als Pfad der Drohne genutzt werden kann.

Da zu Beginn des Wettbewerbs eine Vielzahl von Programmen gleichzeitig gestartet werden müssen, wird der Startvorgang durch die Verwendung von launch-Files realisiert. In dem launch-File wird festgelegt, welche Programme in welcher Reihenfolge ausgeführt werden. In dem launch-File können zu jedem Programm Parameter angegeben werden, die bei der Ausführung des Programms verwendet werden. Jeder Programmaufruf erstellt eine Node. Dies ermöglicht es, ein Programm mehrmals mit unterschiedlichen Parametern zu starten. Im Listing 1 sind beispielhaft die Programmaufrufe der Programme „tracking“ und „fliegen“ dargestellt. Angegeben wird das „Paket“ der zu erstellenden Node. Der „Typ“ ist der Name der auszuführenden Datei. Das Attribut „Name“ legt den Namen der Node fest, die erstellt wird. Ein „Output“ kann optional festgelegt werden und bestimmt, ob Daten auf den Bildschirm ausgegeben werden oder ob diese in eine Datei geschrieben werden. Das Attribut „respawn“ legt fest, ob die Node automatisch neu gestartet wird, falls diese unerwartet geschlossen wird. Dies ist bei dem Tracking-Programm beispielsweise gewünscht, da der gesamte Weg aufgezeichnet werden soll. Wird die Node während des Wettbewerbs beendet, so soll ab der aktuellen Position weiter aufgezeichnet werden. Ein Neustart des Programms „fliegen“ ist hingegen sinnlos, da der Ablauf nur von einer definierten Startposition aus abgearbeitet werden kann.

Listing 1: Start der Programme „tracking“ und „fliegen“ im launch-File

```
        <!--Pfaddarstellung der Flugroute-->
    <node pkg="drone_auto" type="pfaderstellung" name="tracking_node" output="screen" respawn="true"/>

        <!--Main (fliegen/suchen)-->
    <node pkg="drone_auto" type="fliegen" name="drone_main" output="screen" respawn="false"/>
```

3.2.1 fliegen

Das Programm „fliegen“ enthält den eigentlichen Ablauf, den die Drohne während des Wettbewerbs ausführen soll. In dem Programm sind auch die Koordinaten festgelegt, die die Drohne anfliegen soll. Weitere Daten, wie die maximale und minimale Geschwindigkeit oder die Toleranz der Drohne, werden aus einer Konfigurationsdatei gelesen.

Das Programm beginnt mit einer Initialisierung, in der sichergestellt wird, dass alle Parameter geladen sind. Außerdem wird festgestellt, ob sich die Drohne in einer startbereiten Position befindet, also nicht auf dem Kopf liegt und die Rotoren nicht blockiert sind. Gibt die Drohne keine Fehler oder Probleme zurück, startet sie und bleibt über dem Boden schweben.

Ein wichtiger Bestandteil des Programms ist die Funktion „position bestimmen“. Der Treiber der Drohne publisht nur die Beschleunigungswerte, die der IMU der Drohne ausgibt. Die Position der Drohne muss jedoch selber bestimmt werden. Der Programmausschnitt für die Positionsberechnung ist in Listing 2 abgebildet. In den Zeilen 224 bis 227 werden die aktuellen Zeiten abgefragt und die Differenzzeit berechnet. In den Zeilen 228 und 229 wird die aktuelle Position in Abhängigkeit von der letzten Position, der vergangenen Zeit und der aktuellen Beschleunigung, sowie dem Winkel, in dem sich die Drohne zu dem festgelegten Koordinatenkreuz bewegt, berechnet. Die Berechnung ist nur eine Näherung, da davon ausgegangen wird, dass die Beschleunigung über den gemessenen Zeitraum konstant ist. Diese Näherung ist jedoch zulässig, da die Funktion mit 100Hz aufgerufen wird und der Fehler somit sehr klein ist. Die Position in Z-Richtung, also die Höhe, wird nicht auf diese Weise berechnet, da die Drohne über zwei Ultraschallsensoren verfügt, die direkt den Abstand zum Boden messen. Eine Berechnung wäre nicht sehr hilfreich, da die Berechnung immer nur die Höhendifferenz zum Startpunkt berechnen könnte, die Höhe des Geländes sich jedoch ändert.

Listing 2: Berechnung der Position der Drohne

```
void position_bestimmen() {
  end = ros::Time::now().toSec();
  begin = ros::Time::now().toSec();
  diff_time = end - begin_2;
  begin_2 = begin;
  pos_x=pos_x+(vx/1000)*diff_time*cos(((rotZ_drone)*pi)/180)-(vy/1000)*diff_time*sin(((rotZ_drone)*pi)/180);
  pos_y=pos_y+(vx/1000)*diff_time*sin(((rotZ_drone)*pi)/180)+(vy/1000)*diff_time*cos(((rotZ_drone)*pi)/180);
  if ( erstes == 0 || status == 6 ){
```

```
        pos_x = 0.0 * map_size;
        pos_y = 0.0 * map_size;
        rotZ_start = rotZ;
        erstes++;
  }
}
```

In dem Programm ist auch die Funktion für das Fokussieren eines gefundenen Objektes vorhanden. Die Funktion wertet dabei die Daten aus, die von dem Programm „suchen“ gepublisht werden. Je nachdem in welchem Bereich des Bildes sich das gefundene Objekt aktuell befindet, soll die Drohne eine Bewegung ausführen, die das Objekt nach Möglichkeit in der Mitte des Kamerabildes zentriert. Wichtig hierbei ist es, einen guten Wert für die Toleranz dieser Bedingung zu finden. Ohne eine Toleranz wird die Drohne den Zentriervorgang nicht abschließen, da es bei einem fliegenden Objekt nahezu unmöglich ist, dass Objekt exakt mittig im Bild zu fokussieren. Ist die Toleranz jedoch zu groß, würde die Drohne auch ein Objekt am Bildrand akzeptieren. Dies würde eine höhere Fehlerquote bei der Objekterkennung zur Folge haben. Zum anderen wäre die Positionsangabe des Objektes sehr ungenau, da eine Berechnung der Position, auf Grund der Verwendung der Linse ([4]), sehr schwierig ist. In Listing 3 sieht man den Ausschnitt des Programms, indem die Flugrichtung der Drohne, in Abhängigkeit von der Position des Objektes, im Bild festgelegt wird. X- und Y-Koordinatensystem sind bei der Analyse des Bildes und der Drohne unterschiedlich.

Listing 3: Entscheidung der Flugrichtung bei der Zentrierung eines Objektes

```
if ( pos_x_image[farbe] < 100 ) {
        vy_soll = v_max;
} else if ( pos_x_image[farbe] < 280 && pos_x_image[farbe] > 100 ) {
        vy_soll = v_min;
} else if ( pos_x_image[farbe] > 360 && pos_x_image[farbe] < 540 ) {
        vy_soll = -v_min;
} else if ( pos_x_image[farbe] > 540 ) {
        vy_soll = -v_max;
} else {
        vy_soll = 0.0;
}

if ( pos_y_image[farbe] < 100 ) {
        vx_soll = v_max;
} else if ( pos_y_image[farbe] > 100 && pos_y_image[farbe] < 200) {
        vx_soll = v_min;
} else if ( pos_y_image[farbe] > 280 && pos_y_image[farbe] < 380 ) {
        vx_soll = -v_min;
} else if ( pos_y_image[farbe] > 280) {
        vx_soll = -v_max;
} else {
        vx_soll = 0.0;
}
```

Ist der Zentriervorgang erfolgreich abgeschlossen, wird ein Marker erstellt und gepublisht, der die Farbe und die Position des Objektes enthält. Die Anzahl der Marker, die pro Objekt erkannt werden können, ist auf drei begrenzt, da die Gefahr besteht, dass die Drohne zum Beispiel den sandigen Untergrund als gelbes Objekt erkennt. In diesem Fall sollen nicht unbegrenzt viele Objekte an den Rover gesendet werden.

Das Programm enthält außerdem die Funktion, die für das Abfliegen der geplanten Route geschrieben wird. Die Funktion berechnet aus den Koordinaten der eigenen Position und den Koordinaten des nächsten Wegpunktes den Abstand zwischen der Drohne und dem Wegpunkt. Abhängig vom Abstand und der Richtung wird die Geschwindigkeit in X- und Y-Richtung bestimmt. Die Geschwindigkeit ist von der Entfernung des nächsten Zielpunktes abhängig, ist jedoch sowohl nach oben, wie auch nach unten begrenzt. Ist das Gebiet um den nächsten Zielpunkt erreicht, so wird der nächste Zielpunkt geladen und angeflogen. Es wird eine Toleranz angegeben, sodass die Drohne die Koordinaten des Zielpunktes nicht exakt anfliegen muss. Für ein exaktes Anfliegen der Koordinaten kann die Drohne nicht genau genug positioniert werden. Der Versuch würde mehr Leistung benötigen, als angemessen ist, da ein Erreichen der exakten Position keinen Vorteil für die Aufgabe der Drohne hat. Listing 4 zeigt die Berechnung für die Geschwindigkeit der Drohne. Die Berechnung enthält auch die Vorschrift für eine Rotation um die Z-Achse. Diese wird im Wettbewerb jedoch nicht genutzt, da dies die Genauigkeit der Positionsbestimmung nicht zulässt (vgl. Kapitel 3.3.2).

Listing 4: Geschwindigkeitsberechnung bei dem Anflug des nächsten Wegpunktes

```
while ( ( erreicht_x != true || erreicht_y != true ) && ros::ok() ) {
        abw_x = ziel_x - pos_x;
        abw_y = ziel_y - pos_y;
        vx_soll = ( abw_x * cos(rotZ_drone) + abw_y * sin(rotZ_drone) ) * faktor;
```

```
        vy_soll = ( abw_x * sin(rotZ_drone) * (-1) + abw_y * cos(rotZ_drone) )  * faktor;
        vz_soll = (hoehe - altd) * faktor_z;

        if ( ( altd < ( hoehe + 500 ) ) && ( altd > hoehe ) ) {
                vz_soll = 0.0;
        }

        if ( vx_soll > v_max ) {
                vx_soll = v_max;
        }
        else if ( vx_soll < -v_max) {
                vx_soll = -v_max;
        }
        else if ( vx_soll > 0 && vx_soll < v_min){
                vx_soll = v_min;
        }
        else if ( vx_soll < 0 && vx_soll > -v_min) {
                vx_soll = -v_min;
        }


        if ( vy_soll > v_max ) {
                vy_soll = v_max;
        }
        else if ( vy_soll < -v_max) {
                vy_soll = -v_max;
        }
        else if ( vy_soll < 0 && vy_soll > -v_min) {
                vy_soll = -v_min;
        }
        else if ( vy_soll < 0 && vy_soll > -v_min) {
                vy_soll = -v_min;
        }


        if ( fabs(abw_x) < toleranz ){
                erreicht_x = true;
        }
        if ( fabs(abw_y) < toleranz ){
                erreicht_y = true;
        }
```

Der letzte automatisierte Ablauf, den das Programm „fliegen“ enthält, ist der Landevorgang. Die Funktion wird aufgerufen, wenn die Drohne den letzten Wegpunkt erreicht hat. Der letzte Wegpunkt entspricht dabei dem Startpunkt der Drohne, da die Aufgabenstellung des Wettbewerbs vorgibt, dass das gesamte Robotersystem nach Beendigung der Aufgabe wieder an seinem Ausgangspunkt ankommen soll. Dafür wird auf der Startplattform eine grüne Platte platziert, an der sich die Drohne orientieren soll. Diese wird mit dem gleichen Algorithmus erkannt, wie die anderen Objekte. Sobald die Drohne ihre letzte Position erreicht hat, beginnt sie nach der Farbe der Platte zu suchen. Dabei fliegt sie langsam rückwärts, bis sie die Platte gefunden hat. Sollte sie die Platte auch fünf Meter hinter dem Ausgangspunkt nicht gefunden haben, landet sie vorläufig an der Stelle, an der sie sich befindet. In diesem Fall kann das Programm „search home“ (Kapitel 3.2.2) zum halbautomatischen Auffinden der Startplattform verwendet werden.

Der Ablauf der oben beschriebenen Funktionen wird von der Funktion „main“ vorgegeben. Der Hauptbestandteil ist in Listing 5 dargestellt. Bei jedem Durchlauf der Schleife wird zunächst überprüft, ob der Status des ROS in Ordnung ist. Sollte dies nicht mehr der Fall sein, ist davon auszugehen, dass das System abgestürzt ist, oder aber wenigstens die Kommunikation nicht mehr funktioniert. In diesem Fall wird das Programm beendet. Solange der Status in Ordnung ist, wird abgefragt, ob bereits alle Wegpunkte angeflogen wurden. Ist dies nicht der Fall, erhält die Drohne die Kommandos, um den nächsten Wegpunkt anzufliegen. Diese Routine enthält, wie oben beschrieben, auch die Suchalgorithmen. Ist der letzte Wegpunkt erreicht, wird der Landeprozess eingeleitet, der oben beschrieben ist. Nach Abschluss des Landevorgangs wird das Programm beendet. Das Programm wird mit maximal 100 Hertz ausgeführt.

Listing 5: Funktion „Main“

```
while(ros::ok())
{
        if ( wegpunkt < anzahl_wegpunkte ) {
                fliegen(ziel_x[wegpunkt], ziel_y[wegpunkt], hoehe_soll);
                wegpunkt++;
        } else if ( wegpunkt >= anzahl_wegpunkte ){
                landen();
                exit(0);
                printend();
        }

        ros::Rate r(100);
        r.sleep();
}
```

Neben der Steuerung des Ablaufs ist in der Funktion „Main“ auch festgelegt, welche Subscriber

abgerufen werden und welche Publisher veröffentlicht werden. Die folgende Auflistung zeigt, welcher Datenaustausch mit dieser Node stattfindet.

- **Publisher**
  - Ziel: Ardrone Autonomy Treiber; Information: TakeOff
    Wird diese Message gesendet, startet die Drohne.
  - Ziel: Ardrone Autonomy Treiber; Information: Reset
    Wird diese Message gesendet, führt die Drohne einen Software-Reset durch.
  - Ziel: Ardrone Autonomy Treiber; Information: Land
    Wird diese Message gesendet, landet die Drohne.
  - Ziel: Ardrone Autonomy Treiber; Information: Twist_Msgs
    Diese Message sendet der Drohne die Sollgeschwindigkeiten.
  - Ziel: Routenaufzeichnung; Information: Navigationsdaten
    Diese Message enthält Daten zur Flugbahn der Drohne.
  - 6 Publisher: Ziel: Hauptsystem (Rover); Information: Koordinaten
    Positionen der gefundenen Objekte. Für jede Farbe ein Publisher mit sicher gefundenen Objekten und ein Publisher mit möglichen Fundorten.
- **Subscriber**
  - Herkunft: Ardrone Autonomy Treiber; Information: Navigationsdaten
    In dieser Message sind alle Statusdaten der Drohne enthalten.
  - 4 Subscriber: Herkunft: Bildverarbeitung; Information: Koordinaten
    In diesen Messages sind die Koordinaten der gefundenen Objekte in einem Bild vorhanden (blau, gelb, rot, grün).

3.2.2 search home

Bei einem optimalen Ablauf des Wettbewerbs, kommt das Programm „search_home“ nicht zum Einsatz. Es ist stattdessen als Absicherung gedacht. Für den Fall, dass die Drohne nach der Erledigung ihrer Aufgabe den Ausgangspunkt nicht findet, soll das Programm manuell aufgerufen werden. Da dies einen Punkteabzug zur Folge hätte, ist die Funktion „search_home“ dafür ausgelegt, die Drohne halbautonom zu ihrem Ausgangspunkt zurückfliegen zu lassen.

Das Programm verfügt über die gleichen Funktionen, die auch in dem Programm „fliegen“ zur Verfügung stehen. Allerdings ist die Funktion „drehen“ hinzugefügt. Die Funktion lässt die Drohne um einen vorgegebenen Winkel um die Z-Achse drehen. Der Ablauf des Programms ist in Listing 6 dargestellt.

Listing 6: Funktion „Main“

```
int main ( int argc, char** argv ) {
        init();
        steigen();
        drehen(90);
        drehen(90);
        drehen(90);
        drehen(90);
        landen();
        while ( instruction == 0 ) {
        }
        init();
        conf_read_ziel();
        drehen(winkel);
        fliegen(distanz, 0, 1000);
        landen();

        ros::waitForShutdown();
        return 0;
}
```

Bei Aufruf der Funktion startet die Drohne, nach Durchführung der Initialisierung, neu. Anschließend steigt sie auf eine festgelegte Höhe, damit die Kamera einen weiteren Bereich des Spielfeldes übersehen kann. Es werden vier Drehungen um je 90 Grad durchgeführt. In der Funktion „drehen“ ist

ein Delay vorhanden, sodass die Drohne nach jeder Drehung für ein paar Sekunden verharrt. Dies ist notwendig, da die Übertragung der Daten von der Drohne in die Bodenstation durch die Aufgabenstellung nur sehr begrenzt möglich ist. Auf diese Weise sendet die Drohne einige Sekunden das gleiche Bild, sodass dieses in der Bodenstation gut empfangen werden kann. Anschließend landet die Drohne und wartet auf eine Eingabe der Bodencrew. Diese muss nun den Winkel und die Entfernung, in der sich die Zielplattform befindet, in eine Datei schreiben. Anschließend erhält die Drohne den Befehl, die Daten zu laden. Die Drohne dreht sich anschließend um den vorgegebenen Winkel und legt die vorgegebene Strecke zurück. Dort landet sie erneut. Die Probleme, die durch die Rotation entstehen (vgl. Kapitel 3.3.2), können hier vernachlässigt werden, da die Distanz, die während dieser Routine zurückgelegt werden muss, nur sehr gering ist.

3.2.3 suchen

Das Programm „suchen“ ist für die Bildverarbeitung der Frontkamera verantwortlich. In dem Programm werden Grenzen für die Farbwerte der Objekte festgelegt. Außerdem wird die Anzahl der Pixel festgelegt, die mindestens vorhanden sein müssen, damit das Objekt erkannt wird. Sind genug Pixel in dem vorgegebenen Farbspektrum gegeben, berechnet die Funktion den Flächenschwerpunkt des erkannten Objektes. Diese Bildkoordinaten werden an die Funktion „fliegen“ gesendet. „fliegen“ nutzt die Koordinaten, damit sich die Drohne schräg über dem Objekt ausrichten kann. Eine genauere Beschreibung der Funktionsweise der Bildverarbeitung ist in [7] dargestellt. Auf der Abbildung 6 sind die Objekte zu sehen, die das Team auf Grund der Aufgabenbeschreibung nachgebaut und zu Testzwecken verwendet hat. Diese Objekte müssen unter Anderen von der Drohne erkannt werden.

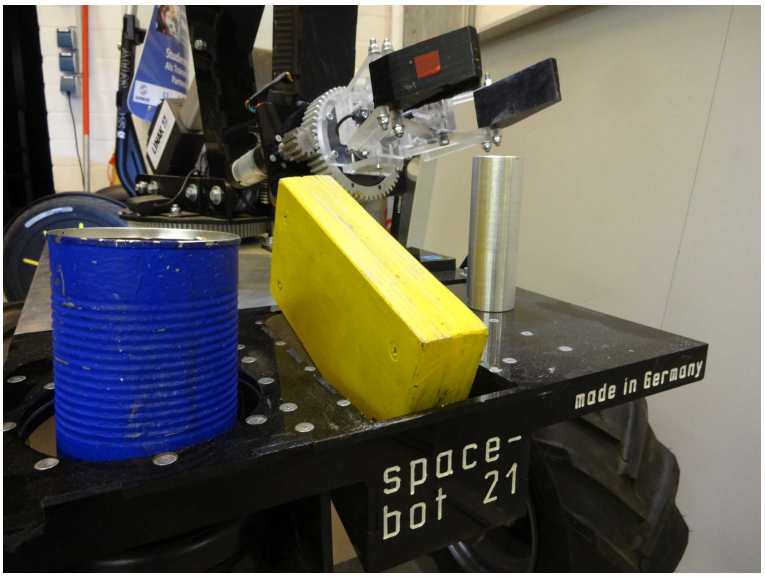

Abbildung 6: Nachgestellte Objekte

3.2.4 tracking

Das Programm „tracking“ ist dafür verantwortlich, die Strecke der Drohne aufzuzeichnen. Laut der Aufgabenstellung müssen die zurückgelegten Wege von Drohne und Rover nach dem Wettbewerb in die selbst erstellte Karte des Geländes eingezeichnet und abgegeben werden. Für die Aufzeichnung der zurückgelegten Route wird das Paket „tf“ verwendet. Der ardrone autonomy Treiber publisht automatisch eine tf-Transformation der Drohne. Diese enthält die Ausrichtung und Position in Bezug auf den Punkt, an dem sich die Drohne befunden hat, als der Treiber gestartet wurde. Die Positionsdaten, die das Paket „tf“ liefert, werden sekündlich abgerufen und in eine Variable vom Typ „Path“ gespeichert. Dieser Variablentyp ist in ROS vordefiniert und kann mit Hilfe von rviz visualisiert werden. Der Pfad wird gepublisht, sodass die Information von der Crew der Bodenstation empfangen und für die Abgabe verarbeitet werden kann.

3.3 Tests im Vorfeld

Im Vorfeld des Wettbewerbs, während der Entwicklung des Programms, werden bereits zahlreiche Tests durchgeführt. Dafür wird sowohl eine Testbed verwendet, die zu diesem Zweck aufgebaut wurde, sowie eine Sporthalle. In der Sporthalle kann die Drohne längere Strecken zurücklegen, wodurch Tests zu der Genauigkeit der Drohne durchgeführt werden können. Auf der Testbed hingegen können nur kurze Strecken zurückgelegt werden. Hier besteht jedoch der Vorteil, dass Höhenunterschiede von bis zu zwei Metern vorhanden sind, sodass die „Geländefähigkeit“ der Drohne getestet werden kann. Außerdem entspricht der Untergrund der Testbed eher dem Untergrund, der auf dem Wettbewerbsareal erwartet wird.

3.3.1 Kalibrierung der Farben

Ein wichtiger Punkt, auf den einige Zeit verwendet wird, ist die Kalibrierung der Drohne auf die Farben der Objekte. Zur Verarbeitung der Bilder wird die Bibliothek „OpenCV“ genutzt. Grundlagen zu der Verwendung der Bibliothek „OpenCV“ können [1] entnommen werden. Auf der Abbildung 7 ist zu sehen, wie die Farbe des Basisobjektes eingestellt wird. Eine Software, die sowohl den Treiber

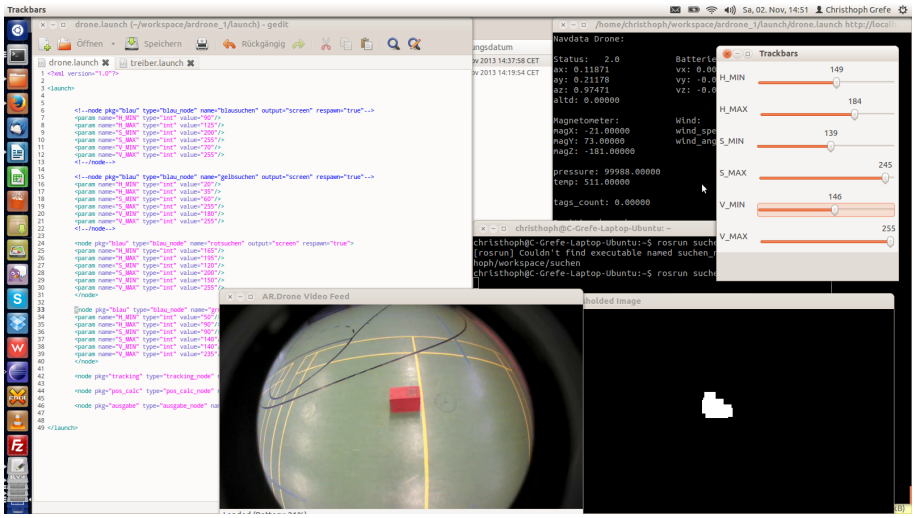

Abbildung 7: Kalibrierung der Drohne auf das rote Basisobjekt

für die Drohne, wie auch die Bibliothek „OpenCV“ verwendet, stellt das Kamerabild der Drohne dreimal dar. Es ist das Originalbild (Abbildung 7 mitte / unten) zu sehen. Das gleiche Bild wird zusätzlich noch als „Thresholded Image“ (Abbildung 7 unten / rechts) dargestellt. Auf diesem Bild wird dargestellt, welcher Bereich des aktuellen Bildes von der OpenCV-Bibliothek erkannt wird, sich also in dem angegebenen Farbbereich befindet. Als drittes wird das Kamerabild als HSV-Image dargestellt. Ein viertes Fenster ist für die Auswahl der HSV-Werte verfügbar. Hier können die Grenzen verändert werden, während die Drohne über einem Objekt schwebt. Die Grenzen müssen möglichst eng gefasst werden, damit die Bibliothek keine falschen Objekte erkennt. Dennoch dürfen die Grenzen nicht zu eng sein, da die Farben je nach Beleuchtung und Blickwinkel variieren.

3.3.2 Genauigkeit

Auch zu der Genauigkeit der Drohne werden einige Tests durchgeführt. Am wichtigsten ist die Genauigkeit in x- und y-Richtung (horizontale Ebene). Aber auch die Genauigkeit in z-Richtung ist nicht zu vernachlässigen. Da die Gegebenheiten des Wettbewerbsortes nicht genau bekannt sind, darf die Drohne die Grenzen des vorgegebenen Spielfeldes nicht überschreiten. Außerdem darf sie nicht zu stark absinken, um nicht mit Hindernissen, im schlimmsten Fall dem Rover, zusammenzustoßen. Eine zu große Flughöhe ist auch nicht möglich, da nicht bekannt ist, ob das Spielfeld nach oben hin begrenzt ist. Außerdem können die Objekte aus einer Höhe von mehr als drei Metern nicht mehr erkannt werden. Als Grundlage zur Berechnung der Position der Drohne dient ein Inertialsensor (engl. inertial

measurement unit, IMU). Grundlagen zu Inertialsensoren sowie typisch auftretende Fehler werden in [8] und [3] betrachtet.

Um die Genauigkeit zu testen, werden in einer Sporthalle die drei Testobjekte auf dem Hallenboden verteilt. Die Drohne wird auf einen Startpunkt gesetzt, der mit einer grünen Platte markiert ist. Die grüne Platte soll auch im Wettbewerb zum Einsatz kommen und soll das präzise Auffinden des Startplatzes sicherstellen. Anschließend wird ein Programm gestartet, in dem eine feste Route einprogrammiert ist. Während des Fluges werden sowohl die Flugroute, wie auch die gefundenen Objekte mit dem Programm „rviz“ dargestellt (siehe Abbildung 8 unten / mittig). Auf diese Weise kann ermittelt werden, wie genau die Drohne Koordinaten anfliegen kann.

Abbildung 8: Genauigkeitstest mit der Drohne

Das Ergebnis der Tests ist, dass die Drohne mehrere Ziele in einem Abstand von bis zu 20 Metern mit einer Genauigkeit von maximal zwei Metern anfliegen kann. Dies funktioniert jedoch nur, sofern die Drohne während dieser Zeit keine Rotation um die Z-Achse ausführt. Da die Rotation nicht exakt ausgeführt wird, ergibt sich hierbei eine Abweichung in dem Winkel der Flugrichtung, was zu einer höheren Abweichung bei dem Erreichen der Zielpunkte führt.

Das keine Rotation ausgeführt werden kann, hat in zweifacher Hinsicht einen negativen Einfluss auf den Ablauf des Wettbewerbs. Zum einen ist laut Aufgabenbeschreibung eines der Objekte unter einem kleinen Vorsprung versteckt. Dieses Objekt kann von der Kamera der Drohne möglicherweise nur dann erfasst werden, wenn sich die Drohne aus der Richtung nähert, in die der Vorsprung geöffnet ist.

Der zweite Nachteil besteht darin, dass für die Ausrichtung der Drohne die Verwendung eines Symbols als Möglichkeit herangezogen wird, an welchem sich die Drohne ausrichten soll. Das Symbol ist in der Firmware der Drohne hinterlegt und die notwendigen Befehle werden von dem Ardrone Autonomy Treiber bereitgestellt. Allerdings nutzt eine Kalibrierung an dem Symbol nur dann etwas, wenn sich die Drohne auch in ihrer Ausrichtung an dem Symbol orientieren kann. Hierfür wäre eine Rotation um die Z-Achse jedoch unvermeidbar. Es muss folglich auf die Verwendung des Symbols als Orientierungshilfe verzichtet werden.

4 Probleme

Während der Entwicklung des Robotersystems und der Programmierung der Drohnen-Software traten einige Probleme auf. Diese führten teilweise dazu, dass Ideen nicht umgesetzt werden konnten. Auch während des Wettbewerbs traten Probleme auf, die dazu führten, dass die Drohne ihre Aufgabe nicht erfüllen konnte. Diese Probleme werden im Folgenden dargestellt und analysiert.

4.1 Nicht umgesetzte Ansätze

In [4] werden mehrere Möglichkeiten genannt, die Grundausstattung der AR.Drone 2.0 zu erweitern, um diese für die Erfüllung der gestellten Aufgaben zu optimieren. Für Wettkampf wurden jedoch nur

zwei dieser Optimierungen vorgenommen. Neben der Verwendung eines Akkus mit größerer Kapazität wurde außerdem noch die Linse verwendet, die das Sichtfeld der Kamera vergrößert.

Der Hauptgrund dafür, dass die Drohne weder mit den LEDs, noch mit den Ultraschallsensoren ausgestattet wurde, besteht darin, dass die Flugzeit der Drohne dadurch negativ beeinflusst würde. Der Nutzen der LEDs ist zusätzlich nicht so stark, wie dies zu Beginn der Überlegung erhofft wurde. Der Grund dafür liegt in der Ausleuchtung der Wettbewerbshalle mit diversen Scheinwerfern. Die LEDs sind nicht stark genug, sodass die Scheinwerfer dennoch eine unterschiedliche Beleuchtung der Objekte verursachen.

Neben dem Energieverbrauch der Ultraschallsensoren konnte hier auch das Problem der Befestigung nicht zufriedenstellend gelöst werden. Da die Hülle der Drohne nicht verwendet werden soll, um die Flugeigenschaften zu verbessern, bietet die Drohne keinen Platz, um die Sensoren zu montieren, ohne, dass das Signal durch die Rotoren verfälscht wird. Zusätzlich müsste eine Auswerteeinheit (Mirkocontroller) auf der Drohne transportiert werden. Auch diese erhöht das Gewicht und den Energiebedarf. Unter Betrachtung all dieser Faktoren ist der Vorteil der einzelnen Verbesserungsmöglichkeiten im Vergleich zu den Problemen und Nachteilen nicht groß genug.

4.2 Grund für Scheitern

Im Wettkampf zeigt sich, dass die Drohne ihre Aufgabe nicht erfüllen konnte. Zu Beginn des Wettkampfes startete die Drohne, nachdem sie ein Signal von dem Hauptprogramm, welches auf dem Rover läuft, erhalten hatte. Der Startvorgang lief fehlerfrei ab. Anschließend stieg die Drohne, bis sie ihre Flughöhe erreicht hatte. Sie begann die einprogrammierte Route abzufliegen. Bereits nach weniger als drei Metern änderte sie jedoch ihre Flugrichtung. Sie flog entgegengesetzt ihrer geplanten Route und überschritt die Spielfeldbegrenzung, wo sie durch ein Schutznetz abgefangen wird. Der Grund für das ungewollte Verhalten wird im Folgenden beschrieben.

Der einzige Grund dafür, dass die Drohne von der programmierten Route abweichen darf, ist das Auffinden eines Objektes.

Die Bildverarbeitung muss also bereits kurz nach dem Starten ein Objekt erkannt haben (obwohl sich in dem einsehbaren Feld der Kamera kein Objekt befand). Die Auswertung der Karte (auf der gefundene Objekte eingezeichnet werden) zeigte, dass die Drohne tatsächlich ein gelbes Objekt in der Nähe des Startpunktes identifiziert hatte. Die Ursache hierfür kann auf die beim Wettbewerb vorherrschenden Lichtverhältnisse zurückgeführt werden. Da ein Test auf dem Gelände des Wettbewerbs vor dem Lauf nicht möglich war, konnten die Farbwerte nicht auf die entsprechenden Lichtverhältnisse eingestellt werden. Dies hatet zur Folge, dass die Bildverarbeitung die Farbe des Sandes, in einer schlecht ausgeleuchteten Ecke, als die Farbe des gelben Objektes erkannte. Es zeigte sich, dass sich eine Klassifizierung nur auf Grund der Farbe eines Objekts als sehr schwierig erwies. Um hierbei auf zufriedenstellende Ergebnisse zu kommen, müssten einige Bedingungen (z.B. Ausrichtung, Beleuchtung) bekannt sein. Die Methode eignet sich nicht für unbekanntes Terrain und ist für schlecht ausgeleuchtete Szenen (z.B. Nacht, Weltraum) nicht geeignet. Eine Methode, die unter diesen Bedingungen vermutlich zuverlässiger arbeitet, ist die Erkennung von Oberflächenstrukturen oder Umrissen.

Wie die Aufzeichnungen der Drohne im Nachgang zeigen, wurde das gelbe Objekt noch innerhalb des Spielfeldes erkannt. Es muss also ein zweites Problem vorgelegen haben, da die Drohne das Spielfeld verlassen hat. Die Ursache ist, wie oben bereits beschrieben, erneut das Erkennen eines Objektes. Die Aufzeichnungen der Drohne zeigen, dass außerhalb des Spielfeldes ein blaues Objekt erkannt wurde. Betrachtet man die Umgebung des Terrains, ist zu vermuten, dass es sich hierbei um ein blau beleuchtetes Redepult handelt, welches in der Nähe des Spielfeldes aufgestellt war.

5 Fazit

Das Robot Operating System (ROS) bietet ein einfaches Baukastensystem, mit dem unterschiedliche Komponenten eines Robotersystem zusammengeführt werden können. Es dient dem Austausch von Daten, Informationen und Befehlen bei direkten Verbindungen. Dabei ist das System so aufgebaut, dass nahezu alle Systemkomponenten unabhängig voneinander agieren können. Fällt also ein Teil des Systems aus, hat dies nicht zwangsläufig eine Auswirkung auf die anderen Komponenten des Systems,

solange zwischen den Komponenten keine direkte Abhängigkeit besteht. ROS eignet sich hingegen nicht für die Übertragung von Daten über große Entfernungen oder bei hohen Latenzzeiten. Folglich kann die Kommunikation eines lokal operierenden Robotersystems über ROS erfolgen. Um Daten an ein Kontrollzentrum zu übertragen, ist jedoch eine andere Kommunikationsschnittstelle zu wählen.

Für die Parrot AR.Drone gibt es einen ROS-Treiber, der die Funktionen der Drohne nahezu vollkommen ausschöpft und der kontinuierlich weiterentwickelt wird, um Fehler zu beheben und den Funktionsumfang noch weiter zu steigern. Die Einbindung der Drohne in ein Robotersystem funktioniert mit Hilfe dieses Treibers gut. Dafür wird die Drohne in das Netzwerk eines Routers eingebunden und darüber mit dem Computer verbunden, der die Steuerung der Drohne übernimmt.

Nach den ersten Tests wurde festgestellt, dass die Grundfunktionen der Drohne ausreichen, um die Aufgaben zu erfüllen, für die sie eingesetzt werden sollte. Sie kann mit Hilfe des internen Beschleunigungssensors eine Route, die aus mehreren Koordinaten besteht, abfliegen. Außerdem ist die Auflösung der Kamera hoch genug, um ein Objekt anhand der Farbe und der Größe zu erkennen und die Drohne darüber auszurichten. Die Flugdauer, die durch den Akku begrenzt ist, ist ausreichend, um das Spielfeld mehrmals komplett abzusuchen.

Im weiteren Verlauf der Wettbewerbsvorbereitungen zeigte sich jedoch, dass die Qualität der Drohne nicht die Präzision zulässt, die für die zufriedenstellende Erledigung der Aufgaben notwendig ist. Die Kamera lässt es nicht zu, die programmierten Farben bei unterschiedlichen Lichtverhältnissen zu erkennen. Hierzu wäre zum Beispiel eine Ausrüstung der Drohne mit einer ausreichenden Beleuchtung notwendig. Zum anderen lässt die Genauigkeit der Drohne bei langen Flugstrecken mit häufigen Richtungswechseln nach, wobei eine Drehung um die Z-Achse gar nicht möglich ist, ohne die Position komplett zu verlieren. Zusätzlich sind die Zahnräder und Motoren, über die die vier Rotoren angetrieben werden, so empfindlich, dass schon kleine Staub- oder Sandkörner die Stabilität der Drohne zerstören.

Während des Wettbewerbs zeigt sich, dass das Programm der Drohne nicht vorsieht, ein Objekt außerhalb des Spielfeldes zu ignorieren. Dies führte dazu, dass die Drohne versuchte, die Spielfeldgrenzen zu überfliegen, sofern die Farbe eines Objektes erkennbar war.

Blickt man auf das Gesamtkonzept des Robotersystems zurück, so lässt sich zusammenfassend sagen, dass das System aus vielen Einzelkomponenten bestand, die für ihre Aufgaben geeignet sind. Das Zusammenspiel dieser einzelnen Komponenten wurde jedoch nicht ausreichend getestet und konnte so auch nicht unter den Bedingungen funktionieren.

Literatur

[1] Gary Bradski Adrian Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[2] Aaron Martinez Enrique Fernández. *Learning ROS for Robotics Programming*. Packt Publishing Ltd., 2013.

[3] Warren S. Flenniken IV John H. Wall David M. Bevly. Characterization of various imu error sources and the effect on navigation performance. Technical report, Auburn University.

[4] Christhoph Grefe. Konfiguration einer flugdrohne zur objektlokalisation. Technical report, hochschule 21, 2014.

[5] Mani Monajjemi. ardrone_autonomy : A ros driver for ardrone 1.0 & 2.0, 2013.

[6] Jason M. O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, 2013.

[7] Benjamin Schoof. Objekterkennung und lagebestimmung mit hilfe von farbe und punktwolken. Technical report, hochschule 21, 2014.

[8] Oliver J. Woodman. An introduction to inertial navigation. Technical Report 696, University of Cambridge, 2007.