Autonomes Fliegen einer Drohne mithilfe einer 3D-Kamera

Matías Spatz und Pablo Fernández

Deutsche Schule Madrid

Inhaltsverzeichnis

Kurzfassung

Wir hatten uns eine günstige Spielzeugdrohne gekauft und nachdem wir etwas damit gespielt haben, ist uns aufgefallen, dass man deren Position ständig korrigieren muss, damit sie auf derselben Position bleibt. Wir fragten uns, ob ein Computer fähig wäre, die Kontrolle zu übernehmen und die Drohne selbstständig zu steuern. Um dies zu erreichen, mussten wir die Drohne zuerst im Raum orten, um sie dann von einem Computer aus zu kontrollieren. Die Erfassung der Drohne erreichten wir mit einer Kinect-Kamera, die nicht die Farbe eines Pixels ermittelt, sondern dessen Entfernung zur Kamera. Damit konnten wir die genaue Position der Drohne in 3 Dimensionen feststellen. Die Drohne wird mit einer kleinen Fernbedienung gesteuert, die die Position der Joysticks mittels Potentiometer misst. Um die Joysticks zu simulieren, benutzten wir einen Arduino als Steuereinheit mit einem digitalen Potentiometer. So haben wir die Drohne gesteuert, ohne die Joysticks bewegen zu müssen. Wir wollen damit die Drohne stabil in der Luft halten und gezielt Positionen im Raum anfliegen.

Einleitung

Heutzutage sind Drohnen der Letzte Ruf, Firmen wie Amazon, Mercedes, Intel und Google investieren in Drohnenentwicklung. Doch alle haben als Ziel den Transport im Freien. Sie benutzen dafür große Drohne die viel wiegen und kraftvolle Motoren besitzen. Wir entgegen möchte eine Drohne für den Innenraum. Unser Vorschlag ist einen System zu entwickeln, in dem der Raum die Drohne steuert. Der Raum soll dafür die Drohne orten können. Das Innovative hierbei ist, dass die Drohne nicht “denken” muss, sie hört nur “blind” zu was der Raum kommandiert. So machen wir die Drohne leichter, da diese keine Wahrnehmungstechnik tragen muss. Dass reduziert auch Stromverbrauch und steigert die Flugzeit.
Als Prototyp-System dachten wir uns, eine günstige Drohne zu kaufen, da diese am “blindesten” war (nur ein Sensor um waagerecht zu bleiben), und eine 3D-Kamera, die die Drohne dreidimensional orten konnte. Damit wollen wir auch zeigen, dass das System auch im schlimmstmöglichen Fall funktioniert.
Es gibt Universitäten (z.B. ETH Zürich), die ein ähnliches System benutzen. Sie verwenden aber ein sehr teures VICON Tracking System, für mehrere zehntausend Euro. Was die ETH eigentlich erforscht sind Manövrieragorithmen und neue Arten von Flugmaschinen. Das “Endprodukt” ist nicht mit dem Trackingsystem gedacht.
Dieses Projekt ist eine Weiterleitung von “Externe Erweiterung der Fähigkeiten eines Spielzeugquadrokopters (Stabilisierung im Raum)”, welches wir im Wettbewerb Jugend forscht Iberia 2016 vorstellten.

Materialien

Für unser Projekt brauchen wir sowohl Hardwarekomponenten, aber auch viele Softwarekomponenten, da es zum Teil ein Informatikprojekt ist.
Warum wir uns für genau diese Komponenten entschieden haben und wofür sie nötig sind, erklären wir im Laufe der Arbeit.

Hardware:
- Drohne: Cheerson CX-10
- 3D Kamera: Kinect 2
- Mikrocontroller: Arduino Mega 2560
- Potentiometer: AD5206

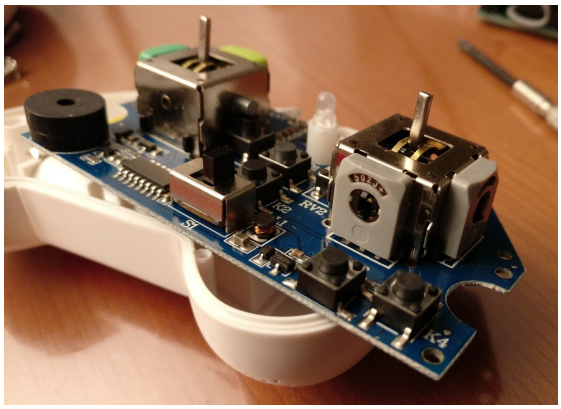
Software:
- Processing
  - Serial Bibliothek
  - Blob detection Bibliothek
  - Kinect Bibliothek
    - Mac Version
    - Windows Version
- Arduino
  - SPI Bibliothek

Durchführung

Unser Projekt besteht aus zwei Teilen, zum einen müssen wir die Drohne vom Computer aus kontrollieren, aber zusätzlich dazu auch deren Position bestimmen um entscheiden zu können, wie sie sich bewegen muss.

Öffnen der Fernbedienung

Am Anfang haben wir gedacht, dass wir die Radiosignale, die von der Fernbedienung zur Drohne gesendet werden, aufnehmen können um die verschiedenen Kommandos (z. B. wie stark nach vorne zu bewegen) zu verstehen, um diese später beliebig selber zu senden und die Drohne damit steuern.



Als wir aber die Fernbedienung öffneten und herausfanden, dass deren Joysticks, wie in vielen anderen Modellen auch, mit simplen Potentiometern funktioniert, dachten wir uns, dass es eine viel bessere Idee wäre, einfach diese Potentiometer zu simulieren und so der Fernbedienung denken lassen, dass man die Joysticks physisch bewegt.

Dies erreichten wir mithilfe eines digitalen Potentiometers. Dies ist eine Art von IC (integrated circuit) welches digitale Signale in einen analogen Widerstand wandelt.

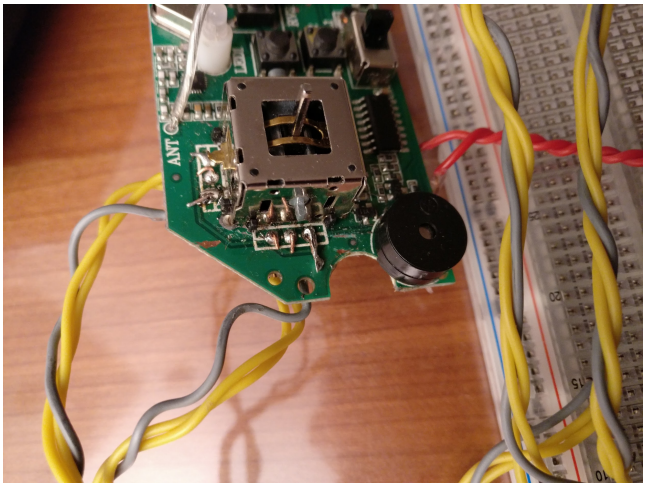
Diese Technik besitzt mehrere Vorteile. Vor allem, da jedes Drohnenmodell höchstwahrscheinlich andere Signale für dieselben Bewegungen benutzt. In diesem Fall wäre es nötig, für jede einzelne Drohne das ganze Prozess zu wiederholen, um sie fliegen zu können, welches nicht das Ziel des Projektes ist. Außerdem ist die Messung dieser Signale auch nicht trivial, da wir spezialisierte Hardware und eine erweiterte Verständigung der Kommunikationsprotokolle bräuchten.

Die Funktionsweise eines digitalen Potentiometer sind kleine Widerstände innerhalb des Chips, die mit Transistoren in Reihe zusammengeschaltet werden, um einen größeren Widerstand zu bilden.

Aus folgenden zwei Gründen entschieden wir uns für den AD5206 als digitales Potentiometer (kurz Digipot). Es ist ein einziger Chip mit 6 Digipots, womit der elektronische Schaltkreis simpler bleibt und es benutzt SPI als Kommunikationsprotokoll, weshalb es sehr einfach mit einem Mikrocontroller zu benutzen ist.

Mikrocontroller

Da der Computer selber nicht SPI “sprechen” kann, fehlt nur noch ein Bauteil. Der Mikrocontroller, welcher die Befehle des Computers auf das Digipot weiterleitet. Dieser wird wie ein Übersetzer benutzt und dank ihm können der Computer und der Digipot kommunizieren. Als Mikrocontroller haben wir ein Arduino Mega ausgesucht, da dieser einfach zu programmieren ist. Arduino ist eine open source Hardware/Software initiative, die die Vereinfachung der Benutzung von Mikrocontrollern anstrebt. Der Arduino Mega spricht mit den Computer durch einen internen seriellen Port über eine USB Verbindung. Der Arduino kann nicht nur SPI, er kann auch programmiert werden. Darum benutzen wir ihn auch um die Verbindung mit der Drohne zu automatisieren, was wir später noch erläutern werden.



Um die Fernbedienung an den Digipot anzuschließen, mussten wir zuerst die “normalen” Potentiometer herauslösen. Es stellte sich heraus, dass der benutzte Lötzinn ein Industrieller war und sich deshalb von unseren einfachen Lötkolben in der Schule nicht löten ließen. Deswegen mussten wir die Joysticks herausreißen.

Danach mussten wir nur noch drei Kabel per Potentiometer von der Fernbedienung zum Digipot und diesen wiederum zum Arduino

verbinden.

Jetzt kann der Computer beliebig die Drohne steuern. Damit sie sich aber komplett autonom bewegen kann, muss der Computer auch wissen, wo die Drohne ist um die gewünschten Anweisungen für die Fernbedienung zu rechnen.

Kinect

Als erstes brauchten wir eine 3D-Kamera. Das ist eine Kamera die anstatt die Farbe von einem Pixel zu messen, die Distanz zu diesem Punkt ermittelt. Wir brauchen so eine Kamera um wissen zu können, wo genau sich die Drohne im 3-dimensionalen-Raum befindet. Außerdem ist es damit im Gegensatz zu einer "normalen" Farbkamera einfacher den Quadrokopter vom Hintergrund zu unterscheiden. Wir entschieden uns unter den vielen Modellen für die Kinect aus mehreren Gründen. Sie ist mit 170 € eine der günstigsten Kameras überhaupt und hat eine sehr präzise theoretische Auflösung von 2 Millimetern in der Tiefe.

Die Kinect wurde eigentlich von Microsoft für die XBOX Spielkonsolen entwickelt, kann aber mit Adaptern auch mit einem Computer benutzt werden.

Die Funktionsweise der Kinect ist ähnlich eines Radar-System, doch anstatt Mikrowellen benutzt es Photonen höhererch Frequenz, also infrarotes Licht. Darum werden Kameras, die mit diesem Prinzip funktionieren, auch *LIDAR* genannt (light-radar). Als Sender werden drei infrarot LEDs benutzt und als Empfänger wird ein spezieller Time-of-flight (Tof) sensor eingesetzt. Dieser misst die Zeitverschiebung zwischen dem Senden und Empfangen. Der Sensor wurde von Microsoft selbst entwickelt und besitzt ein Auflösung von 512 mal 424 pixel (0,217 Megapixel). Der Sensor kann auch pro Pixel das Infrarot-Umgebungslicht des Raumes, welches im Sonnenlicht enthalten ist, berücksichtigen. Die Reichweite des Sensors ist durch die LEDs limitiert, dafür bieten diese einen breiten Beleuchtungsraum.

Datenerfassung

Die Kinect gibt als Ausgangswerte einen eindimensionalen Array d_i, welches die Distanzen in Millimetern enthält. Obwohl der Sensor eigentlich bidimensional ist, werden die Daten in ein einziges Array zusammengefasst, da dies effizienter ist. Durch die Benutzung dieser Speichermethode verliert sich aber die Intuition eines bidimensionalen Arrays in den zwei Indexen, die die Position des Pixels bestimmen. Doch das eindimensionale Array ist eigentlich nur die Aneinanderreihung der Zeilen des Sensors: $d_i = d_{zeile\,1} + d_{zeile\,2} + \ldots + d_{zeilen\,n}$. Mit hilfe der Formel $d_{i,j}^{bi.} = d_{i+B*j}^{uni.}$ kann sie als eine bidimensionale gelesen werden. Hierbei sind i und j die zwei Indizes und B der maximale i Wert. Im Falle der Kinect würde i die X-Achse des Sensors sein, j die Y-Achse und B würde den Wert 512 entsprechen.

Perspektivkorrektur

Jetzt muss man aber bedenken, dass die Kinect eine Kamera ist und darum ihre Daten Perspektive beinhalten. Um diese Perspektive zurückzurechnen benutzt man ein Punktkamera-Perspektivenmodell (Fig.1): $(-p^p_{x,y} - M_{x,y}) = \frac{f_{x,y}}{p^r_z} * p^r_{x,y}$. Hierbei ist p^pder projektierter Punkt auf der Projektionsebene (Sensor), p^rder reelle Punkt im Raum, $f_{x,y}$ ist die Brennweite und $M_{x,y}$ die optische Mitte.
In einer normalen Kamera ist p^pbekannt aber p^rnicht. Dies ist so, da $p_z$und $p^r_{x,y}$unbekannt sind ($f_{x,y}$ ist eine optische Eigenschaft der Kamera, mathematisch gesehen eine Konstante). Doch in

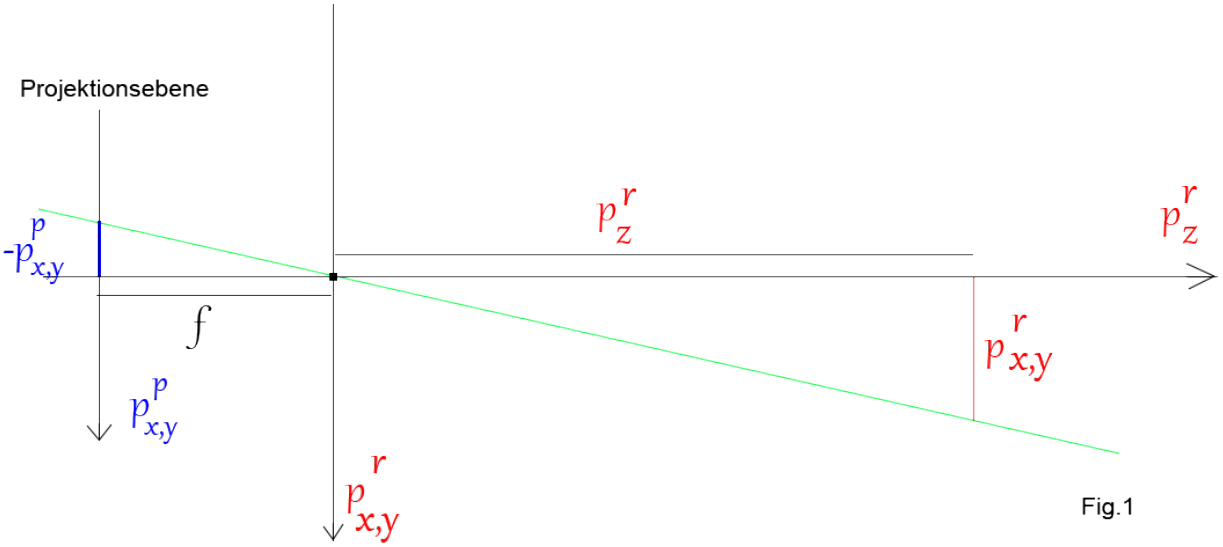
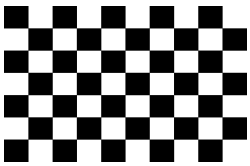


Fig.1

der Kinect ist $p_z$bekannt und $p^r_{x,y}$ kann mit einfachem Umformen mit der Formel $(-p^p_{x,y} - M_{x,y}) * \frac{p^r_z}{f_{x,y}} = p^r_{x,y}$ berechnet werden.

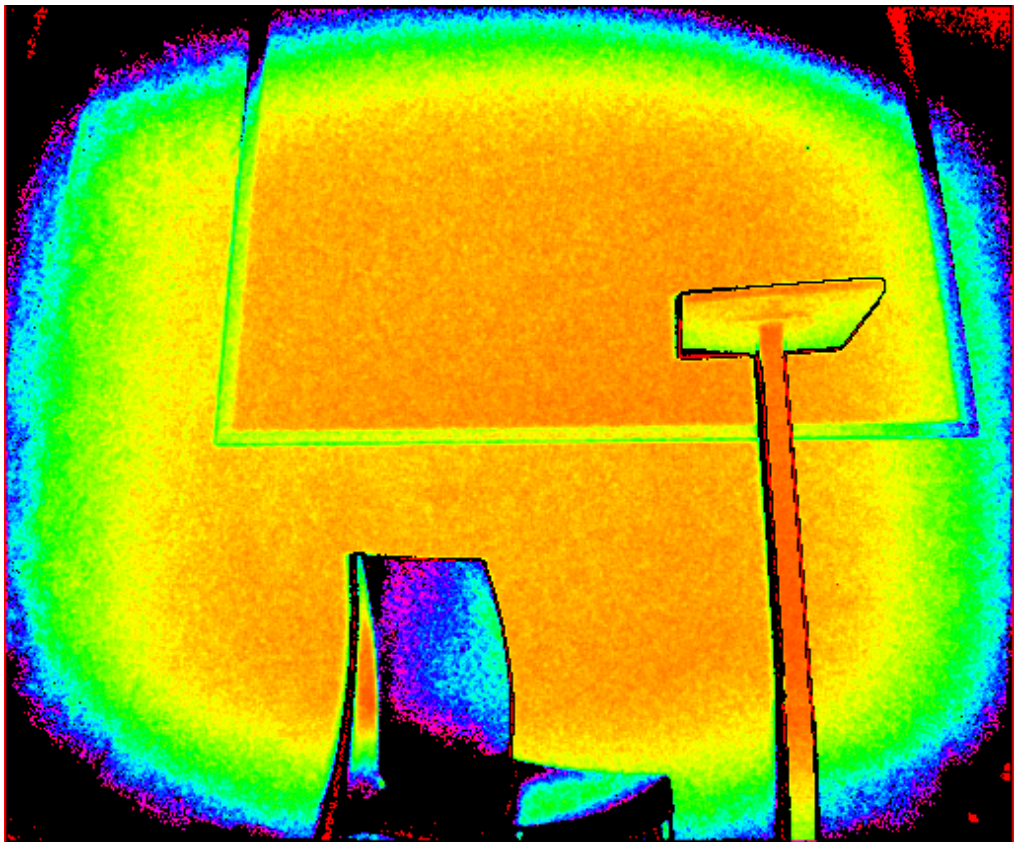
Kalibrierung

Ein wichtiger Aspekt wenn man mit Kameras arbeitet, ist ihre Kalibrierung. Dabei werden die Brennweite f und die optische Mitte des Sensors gesucht. Wir benutzten das Programm [GML C++ Camera Calibration Toolbox] von der *Graphics and Media Lab* an der Staatlichen Universität Lomonosov in Moskau. Das Programm nimmt Fotos von Kalibrierungsmustern und berechnet von ihnen die nötigen Parameter. Unsere Gab für die Brennweite ist $f_x = 364.18$ $f_y = 364.33,$ für die optische Mitte ergab sich die geometrische Mitte also $M_{x,y} = \frac{max_{x,y}}{2}$ (maxist der Maximalwert für x oder y, also die Breite und Höhe des Sensors).



Präzision der Kinect

Wie schon erwähnt wurde, misst die Kinect Distanzen durch Reflexion von Licht. Das führt manchmal zu fehlerhaften Messungen an reflektierenden Gegenständen, da das Licht nur dann gelesen werden kann, wenn es den Sensor auch erreicht. Ein anderes Problem sind dunkle Gegenstände, da diese Photonen teilweise absorbieren. Das hat zur Folge, dass weniger Photonen den Sensor erreichen und die Messung mehr Rauschen enthält. Noch ein Problem ist die Lichtquelle die den Raum nicht gleichmäßig beleuchtet und am Bildrand weniger Lichtstrahlen ankommen. Wir haben auch die durchschnittliche Abweichung jedes Pixels berechnet. Dafür haben wir für jeden Pixel d_i, n Messungen durchgeführt und in d_{i_j} gespeichert. Mit der Formel $\overline{d_i} = \frac{1}{n} * \sum_{j=1}^{n} \left| d_{i_j} - md_i \right|$ kann dann die durchschnittliche Abweichung berechnet werden. Wobei der Durchschnitt des Pixel md_i mit der Formel $md_i = \frac{1}{n} * \sum_{j=1}^{n} d_{i_j}$ bestimmt wird. Der Resultat kann in einem Foto dann visualisiert werden. Hierbei ist :
orange ≈1,27mm,
gelb ≈2,8mm,
grün ≈ 5 mm,
blau ≈ 10mm und
rot ≈ 16mm.
Ab 16mm sind die Pixel Schwarz angezeigt. Die Ränder und insbesondere die Ecken haben ein sehr hohes Rauschen. Die größte Fläche hat aber ein geringes Rauschen mit nur 1.3 mm Abweichung.



Programmierumgebung

Als Programmierumgebung entschieden wir uns für Processing, eine auf Java basierte Sprache die für uns viele Vorteile hat. Wir haben bereits mehrere Jahre Erfahrung damit und sie besitzt die nötigen Bibliotheken für die serielle Kommunikation mit Arduino und der der Kinect. Sie ist außerdem open-source und cross-platform, funktioniert also auf jedem Betriebssystem.
Die Entfernungswerte der Kinect werden mithilfe einer Bibliothek erfasst. Diese Bibliothek ist jedoch für Mac und Windows eine andere, weshalb wir den gleichen Code nicht in verschiedenen Computern benutzen konnten. Wir schrieben deshalb zwei Wrapper-Klassen, eine für jede Plattform, die die gleichen Methodennamen benutzt, damit der Programmiercode austauschbar ist.
Außerdem schrieben wir noch eine weitere Klasse, die ein vorher aufgenommenes Video von der Kinect abspielt. Damit können wir Programme austesten, wenn wir die Kinect nicht dabei haben oder die physische Umgebung ungeeignet ist.
Die eigentliche Erfassung der Position funktioniert in mehreren Schritten, sie benutzt also verschiedene Algorithmen je nach Zustand des Programmes.

Finden der Drohne

Am Anfang, wenn die Drohne noch nicht gefunden wurde, wird diese mit dem sogenannten Mindestabstand erfasst. Mit dieser Methode wird alles, was näher als ein vorbestimmter Wert ist, als Drohne erkannt. Diese Mindestdistanz muss klein genug sein, so dass der Hintergrund nicht erfasst wird und mit der Drohne verwechselt wird. Die Drohne wird näher zur Kamera gebracht, bis sie vom System erkannt wird und mit diesem Wert zur nächsten Erkennungsmethode weitergeleitet wird.

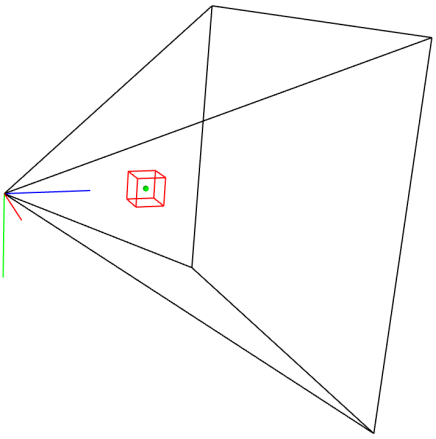
Blob detection

Blob detection (BD) ist ein Algorithmus im Bereich des computer vision, welches Gruppen von Pixeln in einem Foto, die zum selben Objekt gehören, erkennen kann. Dies funktioniert in einem üblichen Farbbild, indem man alle Pixel die eine ähnliche Farbe haben und nah beieinander sind, zusammen gruppiert.

Finden am Anfang

Wir können diese Methode benutzen um die Position der Drohne im Bild der Kinect zu finden. Dafür erstellen wir ein eigenes Bild mit weißem Hintergrund, in welchem wir nur die Pixel färben, die Teil der Drohne sein könnten. Um dann auf dieses Bild BD auszuführen und die mittlere Position der Drohne zu bestimmen. Im ersten Teil unseres Systems wären diese gefärbten Pixel genau die, deren Distanz zur Kamera kleiner als ein bestimmter Wert ist.
Um die Drohne danach weiter lokalisieren zu können, benutzten wir eine Reihe verschiedener Algorithmen, unter anderem auch unser selbstentwickeltes Box-tracking.

Box-tracking



Um den aktuelle Standort der Drohne zu bestimmen, berechnen wir zuerst die vorhergesehene nächste Position mithilfe der vorherigen Zwei. Wir subtrahieren also die vorletzte Position von der Letzten um die Geschwindigkeit zu bekommen und addieren diese letztendlich zur letzten Position. Jetzt haben wir eine Vorhersage von dem Ort, an dem sich die Drohne höchstwahrscheinlich befindet, welche dem Box-tracking als Input gegeben wird.
Wie dieser Algorithmus funktioniert, kann man sich relativ einfach vorstellen. Um die Vorhersage, also den Punkt im dreidimensionalen Raum, zeichnet man eine Kiste. Alle Objekte, die sich innerhalb dieser Kiste befinden, werden als Drohne erkannt. Ist in dieser Kiste gar nichts zu finden, wird deren Größe ein Stück erweitert, und solange wiederholt, bis etwas gefunden wurde oder Kiste zu groß geworden ist. In diesem Fall wird die Drohne als verloren angesehen und es wird von neuem angefangen.
In der Praxis wird nur über die Werte im Array der Kinect iteriert, die sich in der Kiste befinden. Davon werden nur die Werte, die in der z-Achse auch innerhalb der Kiste sind, auf das neue Bild für das BD übertragen.
Nun können wir zuverlässiger Weise die Drohne im Raum orten. Doch um sie zu steuern, müssen wir auch berechnen, wie sich die Joysticks bewegen müssen, damit die Drohne zu dem gewünschten Punkt fliegt.

Berechnung der Potentiometer

Um dies zu erreichen, benutzen wir einen sogenannten PID-Regler. Das ist ein Algorithmus für die Regelung von Werten, die man nicht direkt ändern kann und vielleicht Störungen von außen haben (z.B. Wind, Stöße). Eine Praktische Anwendung dafür wäre ein autonomes Fahrzeug, welches das Gaspedal des Autos, also die Beschleunigung v steuern kann, aber nicht die genaue Position (s). Deshalb muss es die Beschleunigung immer weiter ändern, bis die aktuelle Position dem Gewünschten (Setpoint) entspricht. Für unsere Drohne hat jede Achse einen eigenen PID, welcher die aktuelle Position der Drohne und den Setpoint bekommt und damit Widerstandswert für den Digipot berechnet.
Der PID Regelungsalgorithmus besteht aus drei verschiedenen Teilen, die versuchen den Fehler zu minimieren. Das P steht für Proportional zum Fehler, I für das Integral des Fehlers mit der Zeit und D für Derivativ (in Deutsch Ableitung), die momentane Änderung des Fehlers. Diese Teile werden zusammen addiert und die resultierende Summe ist die die nötige Korrektur. Hierbei ist jeder Teil gewichtet um mehr oder weniger Anteil in der finalen Summe zu haben. Die PID Formel lautet $PID = K_p * F(t) + K_i * \int_0^t F(t)\,dt + K_d * \frac{dF(t)}{dt}$. Mit den Gewichten K_p,K_i,K_d.
Bei der Implementation ist die Berechnung des PIDs einfach, da der proportionale Teil eine einfache Multiplikation ist. Der Integrale Teil ist eine Variable die den Fehler addiert bekommt

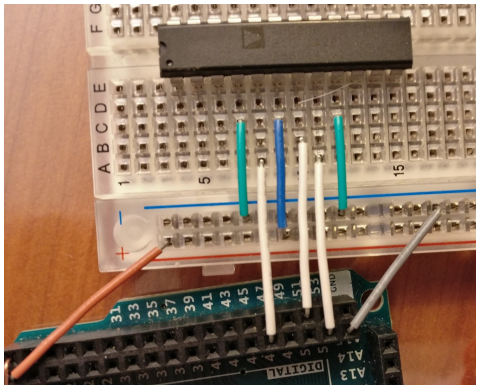
und so den kumulativen Fehler beinhaltet und Derivative Teil ist die Differenz des letzten Wertes und des Neuesten.
Das Komplizierte an dem PID sind die korrekte Gewichtung zu finden. Diese sind nur experimentell zu bestimmen,was hat uns viel Zeit gekostet hat, da bei einer Drohne alle drei PIDs gleichzeitig bestimmt werden müssen.

Processing zum Arduino

Um die berechneten Werte letztendlich wieder zur Fernbedienung zu senden, kommunizieren wir zuerst mit dem Arduino über einen seriellen USB-Port. Es gibt für Processing eine leicht zu benutzende Bibliothek, die fast alles automatisch macht. Da alles im Processing-Programm berechnet wird, werden nur die endgültigen Werte gesendet, welche auf den Digipot weitergeleitet werden. Diese Werte sind ein Array mit vier Variablen des Typs Byte, da die Potentiometer nur 255 verschiedene Positionen haben können. Wir versuchen die kleinste Anzahl an Daten zu senden damit sie schneller übertragen werden kann.
Im Arduino wird dieses Array empfangen und die Werte werden dann einfach auf den Digipot kopiert. Danach wird ein einziges Byte zurückgesendet, welches Processing wissen lässt, dass der Arduino bereit ist die nächsten Werte zu empfangen.

Arduino zu Digipot

Die Kommunikation mit dem Digipot wird mit SPI realisiert. SPI ist ein Kommunikationsprotokoll, welches mit nur drei Kabeln funktioniert und mehrere verschiedene Endgeräte (slaves), in unserem Fall mehrere digitale Potentiometer auf einmal unterstützt. Für uns ist das nützlich, wenn wir mehrere Drohnen fliegen wollen, weil wir dann einfach ein zweites Digipot anschließen können und schon funktioniert es. Um mit dem AD5206 einen Widerstand einzustellen, stellt man den slave-Auswahlpin auf 0 Volt, um das Protokoll wissen zu lassen, an welchen slave die Werte zu senden sind. Mit unserem aktuellen Aufbau gibt es nur ein Digipot, aber dieser Schritt ist wichtig, wenn man mit mehreren Endgeräten arbeitet. Danach werden mit der SPI Bibliothek zwei Werte gesendet (mit einem anderen Modell als der AD5206 funktioniert dieser Schritt vielleicht anders), zuerst eine Zahl von 0 - 5, welches ein einzelnen simuliertes Potentiometer (im Chip gibt es sechs) auswählt und danach den Widerstandswert von 0 - 255, welchen diese Potentiometer haben sollen. Dieser Wert wird dann in Ohm umgewandelt, in unserem Fall also von 0Ω - 10kΩ.



Automatisierung der Verbindung

Nach mehreren Flügen ist uns aufgefallen, wie zeitverschwendend es ist, die Drohne am Anfang mit der Fernbedienung zu verbinden. Um dies zu tun, muss man den linken Joystick

zuerst ganz nach unten bewegen, dann ganz nach oben und zuletzt wieder nach unten. Dies wird “Arming” genannt.
Es ist dafür gedacht, dass die Fernbedienung weiß, welches die Maximal- und Minimalwerte dieses Joysticks sind, sie werden also geeicht.
Nach jedem dieser Schritte piepst ein kleiner Lautsprecher und wir dachten uns, dass wir diesen ganzen Prozess automatisieren könnten, indem der Arduino sozusagen “hört” wann es piepst und entsprechend dazu die ausgehenden Werte anpasst. Damit ist es nur noch nötig, Arduino, Fernbedienung und Drohne einzuschalten und das “Arming” geschiet automatisch.

Dafür mussten wir noch ein Kabel von dem Piepser zum Arduino löten, der dann die Stromspannung misst und damit weiß, ob der Piepser gerade an ist oder nicht.

Open-source

Alle unsere geschriebenen Programme sind open source und auf GitHub unter der Adresse https://github.com/MaPaFe/Drone erhältlich. Es ist mit der Mozilla Public License 2.0 lizensiert, was heißt, dass jedem erlaubt ist, den Code zu sehen und sogar selbst zu benutzen und erweitern, solange wir dafür Anerkennung bekommen und Patentrechte behalten.

Messungen

Um unser System zu bewerten, müssen wir Messungen zu machen. Eine erste logische Messung ist, die Drohne auf einen Punkt im Raum zu halten und Messungen der echten Drohnen-Position durchführen. Nun kann man die durchschnittliche Abweichung berechnen: $\overline{a_x} = \frac{1}{n} * \sum_{i=1}^{n} |x_i - m_x|$. Wobei die Rechnung für jede Achse separat durchgeführt wird, $_x$ die Achse bezeichnet und m_x der Durchschnitt der Werte ($m_x = \frac{1}{n} * \sum_{i=1}^{n} x_i$) ist. Nun hat man einen Wert für jede Achse, den man mit anderen Systeme vergleichen kann. Man kann dann auch den Wert der drei Achsen in einem zusammenfassen: $\overline{a_s} = \frac{\overline{a_x}+\overline{a_y}+\overline{a_z}}{3}$. Diese Rechnung kann mit ein bisschen Modifikation auch benutzt werden, um die Genauigkeit der Drohne auf ihrem Pfad zu verfolgen. Anstatt den Wert gegen den Durchschnitt zu vergleichen, wird er gegen die theoretisch korrekte Position verglichen: $\overline{a_x} = \frac{1}{n} * \sum_{i=1}^{n} |x_i - T_i|$. Zu bedenken ist, dass beim Generieren der korrekten Positionen T_i auch die Geschwindigkeit mitbeachtet wird. Dadurch kann auch die Geschwindigkeit gegen die Präzision untersucht werden um die optimale Geschwindigkeit zu suchen.

Mögliche Anwendung

Wir wollten dieses System bauen, um die Drohne von einem Computer aus kontrollierbar zu machen und so zu automatisieren.
Dies könnte industrielle Anwendungen haben, wie zum Beispiel im Gütertransport in Innenräumen. Wie kann eine Firma eine Drohne als Transportmittel im selben Firmengebäude benutzen, um z.B. Kleinteile zu transportieren und dafür nicht ein eigenes Fließband bauen zu

müssen? Eine Drohne könnte diese Arbeit verrichten. Sie kann mit sehr wenig Aufwand neue Wege gehen und flexibel neue Aufgaben absolvieren.
Doch Drohnen haben ein Nachteil, sie können nur entsprechend ihrer Größe Nutzlast transportieren. Größere und darum schwere Drohnen könnten zu Problemen führen, da sie sehr unkontrolliert und in geschlossenen Räumen fliegen. Große Motoren erzeugen auch starke Turbulenzen. Darum sollte eine Indoor-Drohne klein sein und wenig wiegen. Hier kommt unser System ins Spiel. Es macht die Drohne sehr leicht, weil die Wahrnehmungssensoren nicht von der Drohne selber getragen werden müssen. So können Drohnen ihre Motorkraft benutzen um die Nutzlast zu tragen, anstatt der zusätzlichen Sensoren.
Man muss auch berücksichtigen, dass die Sensoren und die Datenverarbeitung Energie verbrauchen und damit die Flugzeit verkürzt würde. Um diese zu halten, müsste man eine größere Batterie benutzen, die wiederum die Drohne schwerer mache würde und sie bräuchte dann stärkere Motoren. Diese wiegen mehr und verbrauchen mehr Strom und man braucht ebenfalls eine größere Batterie. Dieses Prozesse können einfach mit unserem System gestoppt werden.
Das System hat auch Vorteile beim hochskalieren, da die externen Sensoren für mehrere Drohnen genutzt werden können. Dies würde ebenfalls die Kosten vermindern. Je mehr Drohnen man verwendet, umso besser ist das Preis-/Leistungsverhältnis des Systems.

Pläne für die Zukunft

Wir haben viele geplante mögliche Erweiterungen für die Zukunft. Als Erstes würden wir weiter die PIDs kalibrieren, damit die Drohne stabiler fliegt.
Darüber hinaus wäre es dank unseres Box-Trackings keine große Herausforderung, zwei oder mehr Drohnen gleichzeitig fliegen zu lassen. Danach könnten wir mehrere Kinect-Kameras anschließen, um ein größeres Sichtfeld zu haben auf dem mehr und größere Drohnen erfasst würden.
Eine Limitierung von unserem Aufbau ist, dass die Drohne in einer bestimmten Ausrichtung sein muss um zu funktionieren, nämlich vorwärts ausgerichtet. Wenn die Drohne aber z.B. 90º nach rechts gedreht ist und den Befehl bekommt, sich nach vorne zu bewegen, würde sie nach rechts gehen, die PID-Regler versuchen diese Abweichung zu korrigieren, aber dies macht sie nur größer. Wir könnten einen Algorithmus schreiben, welches diese Drehung erkennt und korrigiert.

Quellen

Arduino Mega 2560 datasheet 6.11.17
https://store.arduino.cc/arduino-mega-2560-rev3
AD5206 datasheet 6.11.17
http://www.analog.com/en/products/digital-to-analog-converters/digital-potentiometers/ad5206.html

Processing homepage 7.11.17

https://www.processing.org/
Serial Bibliothek Referenz 8.11.17
https://www.processing.org/reference/libraries/serial/
Blob detection Bibliothek Referenz 8.11.17
http://www.v3ga.net/processing/BlobDetection/
Kinect Bibliothek für Mac Tutorial 7.11.17
http://shiffman.net/p5/kinect/
Kinect Bibliothek für Mac Quellcode 7.11.17
https://github.com/shiffman/OpenKinect-for-Processing
Kinect Bibliothek für Windows Referenz 8.11.17
http://codigogenerativo.com/kinectpv2/
Arduino homepage 6.11.17
https://www.arduino.cc/
Arduino SPI Bibliothek Referenz 6.11.17
https://www.arduino.cc/en/Reference/SPI

Kinect offizielle Dokumentation 12.11.17
ttps://msdn.microsoft.com/en-us/library/windowspreview.kinect.depthframe.aspx
Kinect Perspektivkorrektur Tutorial 11.11.17
https://threeconstants.wordpress.com/2014/11/21/kinect-point-cloud-normals-rendering-part-1/
Punktkamerapespektivenmomdell 11..11.17
https://en.wikipedia.org/wiki/Pinhole_camera_model
Kinect Kalibrationssoftware 11.11.17
http://graphics.cs.msu.ru/en/node/909

MIT PID Erklärungsvideo 5.11.17
https://www.youtube.com/watch?v=4Y7zG48uHRo
Wikipedia PID 5.11.17
https://de.wikipedia.org/wiki/Regler#PID-Regler