HAW HAMBURG

Entwicklung und Umsetzung eines Konzepts für ein System mit kollektiver künstlicher Intelligenz und verschiedenen Einsatzmöglichkeiten

Projektdokumentation
zur Erlangung des Prüfungsnachweises für das Projekt C

im Studiengang Media Systems

Christoph Mührke 2278652

Daniel Olhoff 2269742

Frithjof Roscher 226912

Max Kretzschmar 2268538

PrüferIn: Prof. Dr. Schumann

Hamburg, 27.11.2019

Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

Inhalt

1. Einführung

Diese Arbeit hat den Anspruch einen konzeptionellen Vorschlag zur Lösung vieler einzelner Probleme in einem gebündelten anpassbaren System darzustellen. Dabei orientiert und bedient sich diese Arbeit vieler in der Natur eingesetzter Konzepte zur Lösung hochkomplexer Probleme durch die Anwendung simpelster Mechanismen.

Denn ähnlich der (relativ) komplizierten Multiplikation, die man durch eine Vielzahl von Additionen ersetzt, ist es möglich komplexe und arbeitsintensivere Aufgaben durch die Aufteilung in einzelne Bestandteile von einfachsten kompakten Computersystemen erledigen zu lassen.
Am Beispiel Wald Aufforstung wird dies recht schnell sichtbar. Wegen Katastrophen, illegaler Rodung, zur Ertrags Sicherung in Monokulturen[1], oder um den Wald auf den Kilmawandel vorzubereiten[2] muss aufgeforstet werde. Um diese Aufgabe zu erledigen müssen je nach Größe der Fläche einige Hundert bis mehrere Tausend Pflanzen neu eingepflanzt werden. In einem Verbund aus Autonomen Drohnen würde das nach unserem Vorschlag wie folgt aussehen.

Zu Beginn wird in einem Hauptrechner das gewünschte Areal, dessen Größe und Position, sowie die gewünschten Baumarten eingeben. Dann startet ein Schwarm von Scout-Drohnen. Diese untersuchen die Umgebung und fliegen ein, ihnen vom Hauptrechner vorgegebenes Muster ab. Der Hauptrechner analysiert das Empfangene Material und kommandiert jeder Scout-Drohne ihr nächstes Ziel. Sollte die Distanz zwischen Hauptrechner und Scout zu groß werden, startet dieser eigenständig eine Repeater-Drohne, die an Kritischen Punkten zwischen den Drohnen kreist, um das Signal zu verstärken und weiter zu leiten.
Sollte bereits ein geeigneter Ort von dem Hauptrechner entdeckt worden sein, so gibt dieser den Startbefehl für eine Arbeiter-Drohne. Diese ist in der Lage schwere Lasten zu heben und trägt das Saatgut, oder Setzlinge die in speziellen Kapseln zu ihrem Bestimmungsort geflogen und abgeworfen werden.

Mit diesem Schema: Scout sucht, Repeater stellt Verbindungen her, Arbeiter wirft etwas ab, wäre es möglich nur durch die Veränderung der abgeworfenen Ladung und Neubestimmung der Suchparameter mehrere Szenarien zu bearbeiten. So wäre ein Einsatz zur Brandbekämpfung sowohl in Urbanen als auch in ländlichen Gebieten denkbar ähnlich wie der Einsatz zur Seenotrettung.

Unsere Arbeit haben wir in einzelne Kernelemente aufgeteilt und werden diese mit einer kleinen Einleitung jeweils vorstellen und dann genauer auf die jeweiligen Punkte eingehen. Zunächst starten wir mit der Navigation, gefolgt von der Kommunikation und der Datenorganisation und Verarbeitung.

Abschließend folgt ein Fazit, was wir am Ende unserer Arbeit mit

2. Einleitung technische Entwicklung

Ein Kernelement der technischen Entwicklung eines auf Schwarm Intelligenz basierenden Systems ist die Erforschung und Umsetzung effizienter und funktionierender Hindernis Erkennung-, und Vermeidungsalgorithmen. Da die Form der Navigation und Hinderniserkennung von der Verwendung verschiedener spezieller Sensoren abhängt. Beeinflussen solche Entscheidungen auch das letzten ende verwendete Design. Und Entwicklungen von Standards können nur mit der Einbeziehung der Navigationssysteme getroffen werden.

2.1 Werdegang

Die Arbeit begann mit dem groben Ziel eine kostengünstige Drohne zu konstruieren die sich in jeder Umgebung zurecht findet und wegen ihres geringen Preises von jedem in großer Stückzahl gekauft und betrieben werden können.

Um jeder Drohne einen Einsatz unabhängig vom Ort und die Gegebenheit zu ermöglichen ist notwendig ein robustes System zur Erkennung von Hindernissen zu entwickeln.

Zu Beginn wurde dies mit der, theoretischen Kombination eines Raspberry Pi Zero, einer kleinen Kamera und eines GPS Moduls bewerkstelligt. Um möglichst schnell zu arbeiten wurde zunächst nur eine Handelsübliche DSLR mit integriertem GPS benutzt.

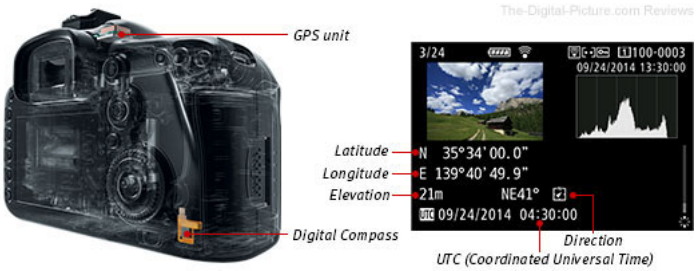


Abbildung 1: Darstellung der Kamera zusammen mit GPS und Kompass Position und beispielhafte Werten

Moderne Kameras speichern in nahezu jedem Bildformat sogenannte „EFIX-daten“, diese tragen Hinweise auf das verwendete Kamera Model, Objektiv, Fotograf und wenn vorhanden Kompass und GPS Daten. Ist bekannt, dass diese Werte Existieren ist es leicht sie sich ausgeben zu lassen.

Eine, mit dieser Kamera, aufgenommen Reihe von Bildern wurde, einen Weg ablaufend angefertigt. Zur Evaluierung wurde nun ein Programm geschrieben, welches diese Reihe von Bildern lädt und immer neu berechnet, um den stetigen Fluss von neuen Bildern zu simulieren.
In jedem neuen Bild wurden dann bis zu 500 Trackingpunkte basierend auf deren Position im letzten Bild, gesucht und die Differenz zusammen mit den GPS Informationen und der bekannten Brennweite in eine Distanz zum Objekt umgerechnet.

Dieses Verfahren, auch wenn es zu Beginn unter bestimmten Bedingungen eine geeignete Geschwindigkeit erreichte offenbarte recht schnell einige Probleme.

- GPS wiederholungsrate: ein zu Beginn nicht bedachter Faktor war die geringe wiederholungsrate des verbauten GPS Moduls (1Hz) so wurden bei Vier geschossenen Bildern pro Sekunde nur einmal die GPS Daten erneuert.
- Die Präzision: Das verbaute GPS Modul hatte eine Varianz von ± 5m auf Längen-, und Breitengrad und eine Varianz von ± 2m in der Höhe.
- Sich bewegende Objekte können mit diesem Algorithmus nicht erkannt werden.

Das erste Problem wäre durch die Verwendung eines anderen GPS Moduls lösbar gewesen. Das zweite hingegen nur in gewissem Maße. Es gibt zwar präzisere Module, allerdings hängt die Präzision immer von den Sichtbaren Satelliten ab und ein kurzer Verlust des Signals könnte zu einer Kollision führen. Dazu Beginn der Arbeit die präzisesten erhältlichen GPS Empfänger lediglich eine Präzision von ± 1m auf Längen-, und Breitengrad ermöglichten, was für diese Aufgaben zu grob ist. War bereits bekannt das nach einer anderen Lösung gesucht werden müsste. Das letzte Problem, die fehlende Fähigkeit sich bewegende Objekte zu erkennen, sorgte für die Abschaffung des Konzepts einer Drohne, die lediglich einen Optischen Sensor benötigt.

Der nächste Schritt war die Implementierung einer Stereokamera, hierfür wurde ein Raspberry Compute Module 3 (RCM) verwendet, da es zwei leicht zugängliche Kamera Interfaces besitzt.

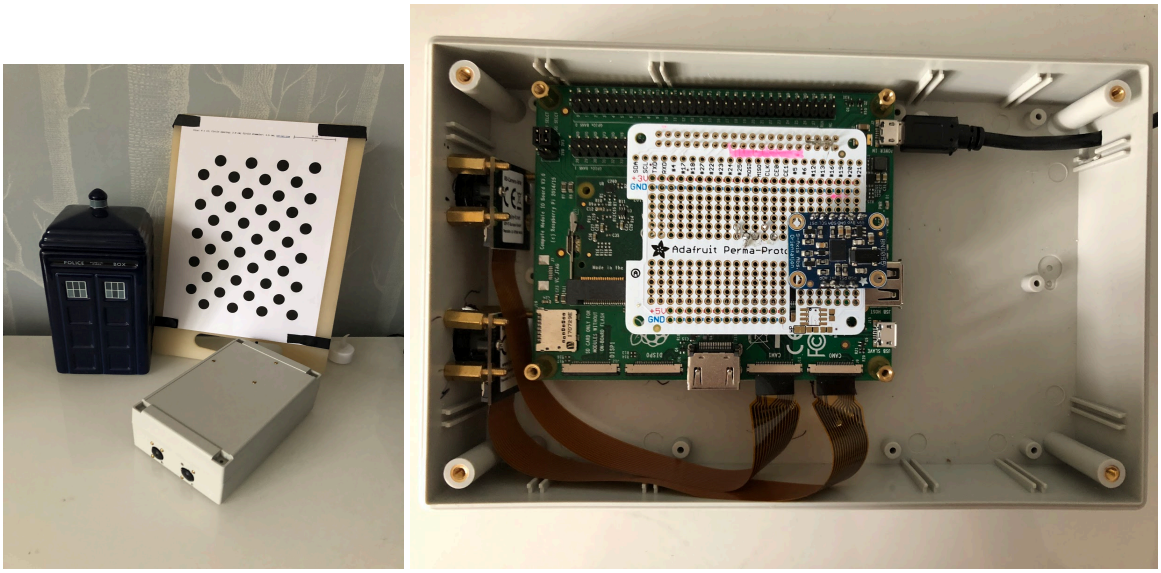


Abbildung 2. & 3 RCM mit IMU Sensor, in Projekt Box zsm. mit Kalibrier Matrix

Bei einer solchen Stereokamera werden zu jederzeit zwei Bilder gleichzeitig aufgenommen und dann verglichen. Bei dem vergleich analysiert ein Algorithmus zunächst ein Bild und sucht „Pixelfelder“ die später im zweiten Bild wiederentdeckt werden müssen. Nachdem die Felder gefunden wurden, wird durch deren unterschiedliche Position in beiden Bildern auf ihre Tiefeninformation zurückgeschlossen. Dies wird dann mit der bekannten Distanz zwischen den beiden Kameras, sowie der bekannten

Brennweite kombiniert und zu einem „Depth Image“ zusammengefasst. Hierbei handelt es sich um ein Bild, bei dem jeder Pixel einen die Distanz zu diesem Punkt gespeichert hat. Diese kann man beim Betrachten dann als Graustufen oder Farbwerte erkennen. Die Ergebnisse dieses Geräts waren vielversprechend. Doch wegen Hardware Limitierungen war es nicht möglich an diesem Gerät festzuhalten.

Das erste Problem war die begrenzte Anzahl von Seriellen Schnittstellen (CSI & I2c). Je zwei dieser Schnittstellen sind ohne Probleme zu erreichen, doch benötigt jede Kamera die komplette Bandbreite ihrer Schnittstellen so, dass für die Anbindung von Sensoren über I2c keine Bandbreite mehr zur Verfügung steht. Zur Lösung dieses Problems wurde von den einzelnen Kameras abgesehen und eine im Handel erhältliche Lösung angeschafft. Die sogenannte Intel Realsense d435. Eine kompakte Stereokamera mit eingebauter VPU (video processing unit). Die VPU entlastet das Hauptsystem, da die tiefen Berechnungen auf der Kamera durchgeführt werden und das Hauptsystem wie eine USB Kamera nur noch die Bilder anfragen muss.

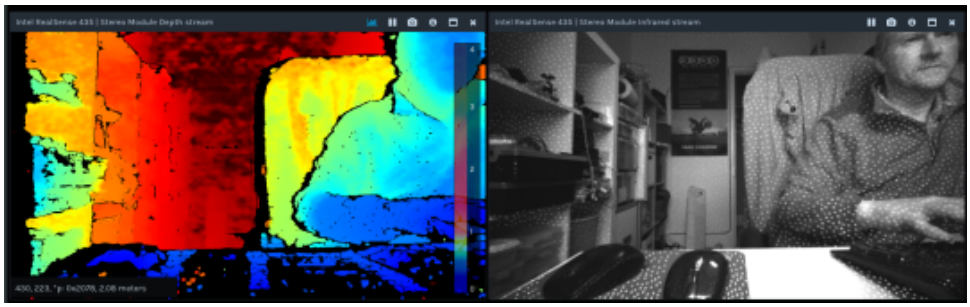


Abbildung 4. Links Depth Image in Farbe, rechts Schwarz-weiß Kamera bild

Wegen der von der Kamera generierten hohen Datenrate ist eine USB3.0 Schnittstelle erforderlich, diese wird aber nicht von dem RCM angeboten. Da die Treiber aber nach einer CPU mit x86 Architektur verlangen und ARM (vom RCM benutzt) nicht unterstütz wird, musste das RCM ersetzt werden.

Der erste Ersatz war ein von Intel empfohlene Entwicklerboard das AAEON UP. Dieses musste aber auch nach langem testen, wegen fehlendem Support und wiederkehrenden Problemen bei der Stromversorgung durch ein anderes System ersetzt werden.

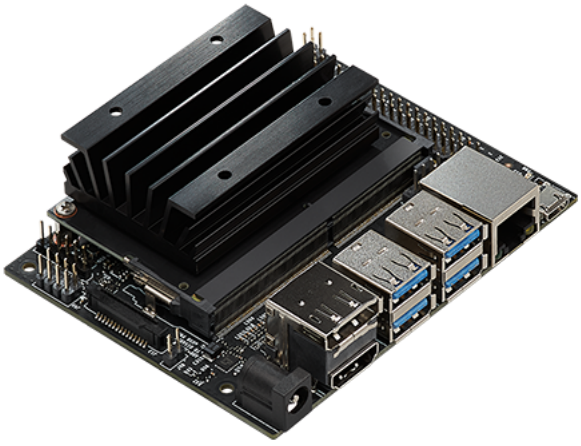


Abbildung 5. Jetson Nano

Das NVIDIA Jeston Nano, ein kompaktes Board mit 4 Kern ARM Prozessor und einer integrierten Grafikkarte mit 128 CUDA Cores. Auch wenn ARM nicht von Intel selbst unterstütz wird, führte eine große Community und guter Support seitens Nvidia dazu, dass dieses Board in Kombination mit einer Reihe an Sensoren die gewünschten Ergebnisse liefert und die Entwicklung voranschreiten kann.

3. Verwendete Verfahren

Auch nach einigen Iterationen ist diese Arbeit darauf fokussiert flugfähige Endergebnisse zu ermöglichen. Darum ist das Endgewicht wichtig und die Rechenleistung somit limitiert.

So musste bei jedem Verfahren nach Optimierungspunkten gesucht werden und nach Möglichkeiten diese auch umzusetzen. Da die native Auflösung der Intel Realsense d435 848x480 Pixel, was 407.040 einzelne Bildpunkte bedeutet, war offensichtlich das eine große Menge Optimierungspotential in der Art und Weise der Bildverarbeitung liegt. Von sich aus kann die d435 das Bild intern auf die halbe Größe skalieren. So ist die Anzahl der Pixel zwar geringer, aber im Hinblick auf verwendete Algorithmen, eine zu große Anzahl, als dass diese so verwendet werden kann.

Um die benötigte Menge der Pixel zu minimieren wurde sich dafür entschieden 1000 Punkte gleichmäßig auf dem Bild zu verteilen. Dies geschah unter zu Hilfenahme des Goldenen Schnitts ((1+sqrt(5))/2). Dieses Verhältnis kann häufig beobachtet werden, schaut man sich die Verteilung von Blättern an einem Blumenstängel und deren relativen Winkel zu einander an, oder im Beispiel der Sonnenblume, deren Kerne einen sehr an die in Abb3. gezeigte Spirale erinnern. In der Natur passiert das, um den platz möglichst effizient auszunutzen und gleichzeitig jedem Blumenkern, oder Blatt das Maximum an Sonnenstrahlen zu garantieren.

Da dieses Verhalten dem Behandelten Problem, die möglichst gleiche Verteilung von Punkten auf einer gegebenen Fläche sehr ähnelt, wurde entschieden diese Spirale nachzuempfinden. [3]

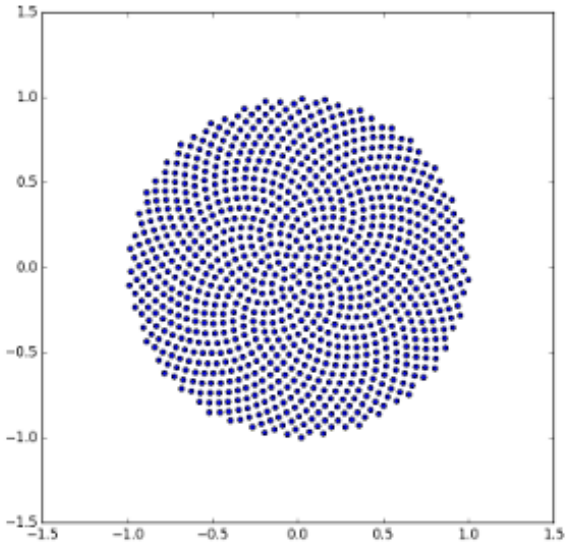


Abbildung 6. 1000 Punkte spiralförmig angeordnet

Um die vorher bestimmten Punkte zu benutzten wird zunächst jeder Punkt geprüft und mit dem Label Obstacle oder Free gekennzeichnet. Danach wird durch die liste der Free Punkte iteriert. Dabei wird ein Rechteck berechnet dessen Kantenlänge sich nach der, im behandelten Punkt gespeicherten, Distanz (**D**) ergibt. Dabei wird dieses so konstruiert das es dieselbe Pixel Mengen wie ein Objekt mit den Maßen 1m x 0,5m in der Entfernung D in dem Depth Image besitzt. Diese Pixelmenge wird nun mit der Liste der Obstacles verglichen. Sollten keine Übereinstimmungen gegeben sein, wird der Punkt als Potantial Target (pt) markiert.

Diese Verfahren wurde in großen Teilen durch die Arbeit „Depth Image Processing for obstacle avoidance VTOL UAV“ [4] eines Teams des DLRZ insperiert.

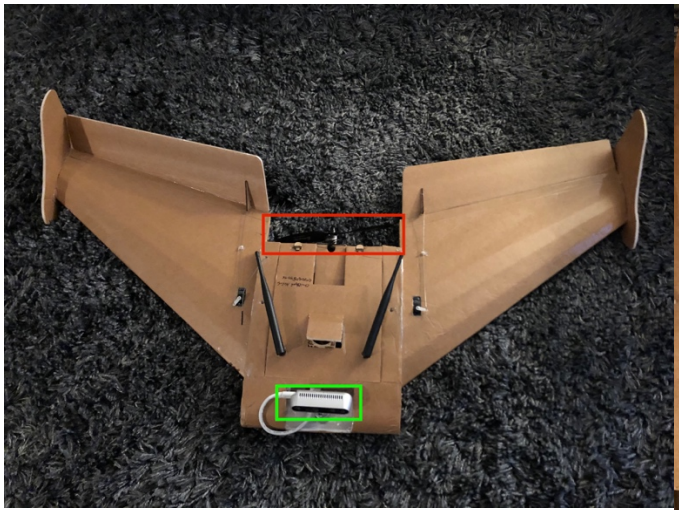
Wenn alle Free Punkte behandelt wurden, wird der Punkt aus pt verwendet der wahlweise dem Bildmittelpunkt oder dem Zielpunkt am nächsten liegt.

Um bei 1000 Punkten die Berechnung so schnell wie möglich durchzuführen werden die CUDA Cores des Jetson Nano verwendet. Um für jeden freien Punkt diese schleife einzeln durch zu laufen.

Die nächste Routine konvertiert dann die Bild Koordinaten in Steuerbefehle zusammen mit einer Schleife, die beendet wird, wenn die Ziel Ausrichtung erreicht, oder eine Kollision bevorsteht.

4. Verwendetes Design

Zu Beginn war wegen der höheren Manövrierfähigkeit und der freien Sicht in Flug Richtung ein sogenannter Quadrocopter geplant. Da diese wegen der geringen Effizienz der Fortbewegungs Methode verhältnismäßig geringe Lasten tragen können und die Flugzeit mit jeder zusätzlichen Beladung deutlich abnimmt wurde entschieden, einen Flugzeug Model als Träger zu verwenden. Flugzeuge, können wegen der effizienteren Methode Tragkraft zu erzeugen und bei deutlich geringerem Verbrauch mehr Last für längere Strecken tragen. Um die Verluste der Wendigkeit gering zu halten wurde sich für ein „Nurflügler“ Design entschieden. Diese können wegen dem geringen Aerodynamischen Profil hohe Geschwindigkeiten erreichen und sind gleichzeitig wegen großer Steuerflächen immer noch sehr wendig. Ein entscheidender Vorteil ist, dass diese Flugzeuge nicht wie typische Propellermaschinen gezogen werden, also einen Propeller und Motor an der Spitze besitzen. Sondern die meisten Modelle ihren Antrieb hinter der Hauptkammer tragen, also geschoben werden. Dies ermöglicht die Anbringung der Tiefenkamera in der Mitte des Flugzeugs ohne, dass ein Propeller das Bild verfälscht.



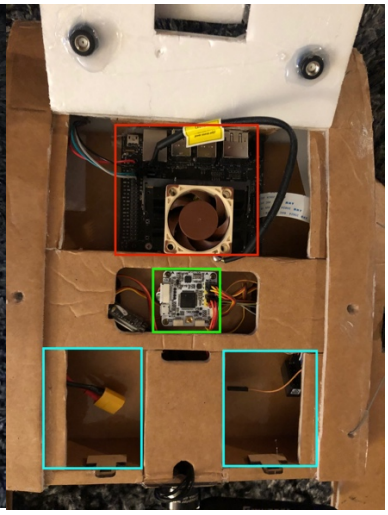


Abbildung 7 (links) Das Flugzeug Modell, Rot Antrieb, Grün Intel Realsense D435

Abbildung 8 (rechts). Rot Jetson Nano, Grün Flugcontroller, Blau Akku Fach

Durch den Wandel der Plattform von einem Quadrocopter hin zu einem Flugzeug ist es auch möglich für jeden benötigten Typ die notwendigen Vorrichtungen anzubringen. So kann das Testen in Zukunft erleichtert werden.

5. Praktische Umsetzung

In dem Fall, dass es zu einer Umsetzung unseres Konzepts kommt, können, sofern keine alternativen vorhanden sind, die von dieser Arbeit angeführten Algorithmen bei einer gleichen Implementierung benutzt werden. Sie wurden so wie der Großteil des geschriebenen Programmcode mit einer Adaptierung an verschiedene Szenarien als kern Kriterium entwickelt.

6. Einleitung Kommunikation

Neben der technischen Realisierung der Drohne und der Bilderkennung beschreibt ein weiteres Kernelement der Arbeit die Kommunikation und das Verhalten von Drohnen im Kollektiv. Im Einzelnen beschreiben wir in welcher Form die Drohnen Datenpakete austauschen und wie sie sich im Schwarm möglichst effizient verhalten.

Für unser Modell benutzen wir ein Szenario, in welchem der Drohnenschwarm dazu eingesetzt wird, ideale Plätze in einem Wald auszusuchen, um dort neue Bäume zu pflanzen. Um dieses Ziel zu erreichen, weisen wir den Drohnen jeweils eine von drei möglichen Rollen zu. Die Scout-Drohne besitzt eine Kamera, um Landschaftsdaten zu erheben, welche nach der Auswertung zur Bestimmung geeigneter Orte benutzt wird. Die Arbeiter-Drohne verfügt über technische Apparaturen, um Baumsaat zu verteilen und ist – bedingt durch ihre Rolle – materialtechnisch verstärkt. Um die Verbindung innerhalb des Schwarms aufrecht zu erhalten benutzen wir Repeater-Drohnen, welche den einzigen Zweck der Signalverstärkung verfolgen.

6.1 Kommunikation

6.1.1 Technik

Jede Drohne ist mit einer WLAN-Karte ausgestattet. Dieses Modul unterstützt alle marktüblichen Standards (IEEE 802.11) zur drahtlosen Kommunikation mittels WiFi. Für die Übertragung von Datenpaketen können wir sowohl auf das 2.4 Ghz als auch 5.0 Ghz ISM Frequenzband zugreifen, welches uns unterschiedliche maximale Reichweiten bei variierenden Datenübertragungsraten ermöglicht.

Bei einer Übertragungsrate von 11 Mbps (Download und Upload) ist zum Beispiel eine maximale Distanz zwischen zwei Teilnehmern von rund 2200 Metern möglich[5].

Die tatsächliche Kommunikation läuft mittels Datenpaketen ab. In diesen Datenpaketen befinden sich zu jedem Zeitpunkt die Positionsdaten der Drohne, ihre Rolle im Schwarm, notwendige Parameter für die Ausführung des Fluges, Anweisungen für eine Positionsänderung und Ausführung einer Arbeit und, abhängig von ihrer Rolle, Rohbilddaten. Diese Datenpakete werden von Drohne zu Drohne verschickt bis sie schlussendlich im Hauptrechner ankommen, um verarbeitet zu werden. Dabei dienen Repeater-Drohnen als Router zwischen den Drohnen.

6.1.2 Grundprinzip

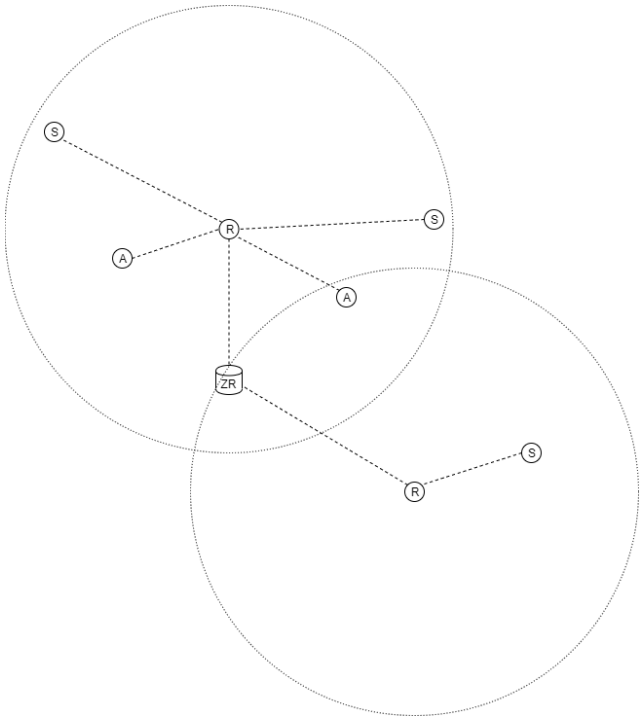


Abbildung 9: Schematische Darstellung des Schwarmnetzwerkes

Die Abbildung 1 zeigt den schematischen Aufbau eines Schwarmnetzwerkes, wie es in der Praxis zum Einsatz kommen würde. Als Ausgangspunkt dient der Zentralrechner, welcher eine hohe Rechenleistung zur Bildauswertung besitzt. Zusätzlich dazu verfügt er über Machine-Learning Algorithmen, um die Eignung aufgenommener Landabschnitte zu prüfen und Steuerbefehle zu erzeugen. Die durch die Scout-Drohnen generierten Bilddaten werden als Datenpakete zwischen den Drohnen and den Zentralrechner geschickt, welcher genaue Anweisungen als Basis für Arbeiter-Drohnen erstellt.

Entfernen sich Arbeiter- oder Scout-Drohnen zu weit von ihrem Hauptrechner, werden automatisch Repeater-Drohnen losgeschickt, um die Verbindung aufrecht zu erhalten und die maximale Reichweite und damit gleichzeitig das maximal abzudeckende Gebiet zu erweitern. Sollte es zu einer Störung während des Betriebes kommen, zum Beispiel durch Umwelteinflüsse oder durch zu niedrigen Akkuladestand, werden die Drohnen im laufenden Betrieb durch Reserveeinheiten ersetzt.

In der Theorie kann durch dieses Verhalten ein großes Netzwerk mit vielen Drohnen über einem Gebiet aufgebaut und über einen langen Zeitraum aufrechterhalten werden.

7. Verhalten

7.1. Grundlagen

Um ein möglichst effizientes Verhalten zu erreichen, wurde sich an Beispielen aus der Natur orientiert. Schwarmverhalten und Schwarmintelligenz lässt sich bei vielen Insektenvölkern wie beispielsweise Bienen, Wespen oder Ameisen feststellen. Diese Insekten bilden teils riesige Kolonien, welche einem bestimmten übergeordneten Zweck dienen. Um diesen Zweck zu erreichen, hat jeder Teilnehmer in diesem Konstrukt eine genau definierte Aufgabe. Die Parallelen zu unserem Projekt sind damit eindeutig. Im Zuge des Projektes wurden 2 Modellgrundlagen entwickelt, welche im Folgenden näher beschrieben werden.

Aufgrund der technischen Einschränkungen des Projekts wurde für die Darstellung des Verhaltens eine rein modellhafte Simulation in Python geschrieben. Einzelne Codeabschnitte, wie beispielsweise die Wegfindung, können jedoch in abgewandelter Form als Grundlage für praxistaugliche Algorithmen benutzt werden.

7.1.1 Baugleiche Drohnen

Die Idee baugleicher Drohnen basiert auf diversen Untersuchungen, welche bei Ameisen durchgeführt wurden. Man unterscheidet grundsätzlich zwischen Arten, bei denen die individuellen Insekten durch ihre körperlich ausgeprägten Merkmale unterschiedliche Rollen innerhalb einer Kolonie einnehmen und jene Arten, die absolut identische Arbeiter ausbilden. Bei der letzteren Art werden die Aufgaben und Arbeiten innerhalb eines Ameisenstaates situativ verteilt. Kommt es zu einer Störung und damit beispielsweise zu einem Verlust eines Teils der Ameisen, können die Insekten die Rollen tauschen und so die fehlende Arbeitskraft in einem Bereich ausgleichen.

Wenn wir dieses Muster auf unseren Drohnenschwarm übertragen, erreichen wir die größtmögliche Flexibilität. Bei der Durchführung einer Aufgabe kommen identisch aufgebaute Drohnen zum Einsatz, von denen jede einzelne alle Teilaufgaben übernehmen kann. Gibt es einen störungsbedingten Ausfall von Drohnen, gibt es – abhängig von der Anzahl an Reserve-Drohnen – unmittelbaren Ersatz.

Abhängig davon, welche Teilaufgaben zu erfüllen sind, gibt es hierbei allerdings erhebliche Nachteile. Bedingt durch den identischen strukturellen und technischen Aufbau der Drohnen werden unter Umständen niedere Aufgaben, wie beispielsweise das alleinige Verstärken eines Signals, von Drohnen durchgeführt, welche auch höhere Arbeiten ausführen könnten. Das ist sowohl kosten- als auch effizienztechnisch nicht optimal.

7.1.2 Spezialisierung

Für das tatsächliche Konzept wurde sich für eine bautechnische Spezialisierung entschieden. Ähnlich der körperlich ausgeprägten Merkmale bei Ameisen, kommen dabei bautechnisch angepasste Drohnen zum Einsatz. Eine Arbeiter-Drohne besitzt ein verstärktes Gerüst und entsprechende Apparaturen, um eine ausführende Aufgabe zu übernehmen. Scout-Drohnen verfügen über eine gesteigerte Flexibilität und Geschwindigkeit und Repeater-Drohnen haben ein sehr konstantes

Flugverhalten sowie eine hohe Flugdauer. Der Vorteil an diesem Konzept ist die Einsparung von Kosten und Material. Es werden keine redundanten Systeme an Drohnen benötigt, welche nicht zu der tatsächlichen Aufgabe beitragen. Auf der anderen Seite fehlt bei einer Störung unter Umständen die nötigte Reserve, da spezialisierte Drohnen nicht jede Arbeit übernehmen können.

7.2. Modell

Um die Ergebnisse der Recherchen und Untersuchungen zu visualisieren, wurde im Zuge des Projektes eine Simulation mithilfe von Python und PyGame erstellt. In dieser sieht man eine modellhafte Darstellung der Drohnen und deren Verhalten im Schwarm bei der Ausführung einer Aufgabe. Bei der Aufgabe handelt es sich um das Analysieren von Landabschnitten und das darauffolgende Pflanzen von Bäumen.

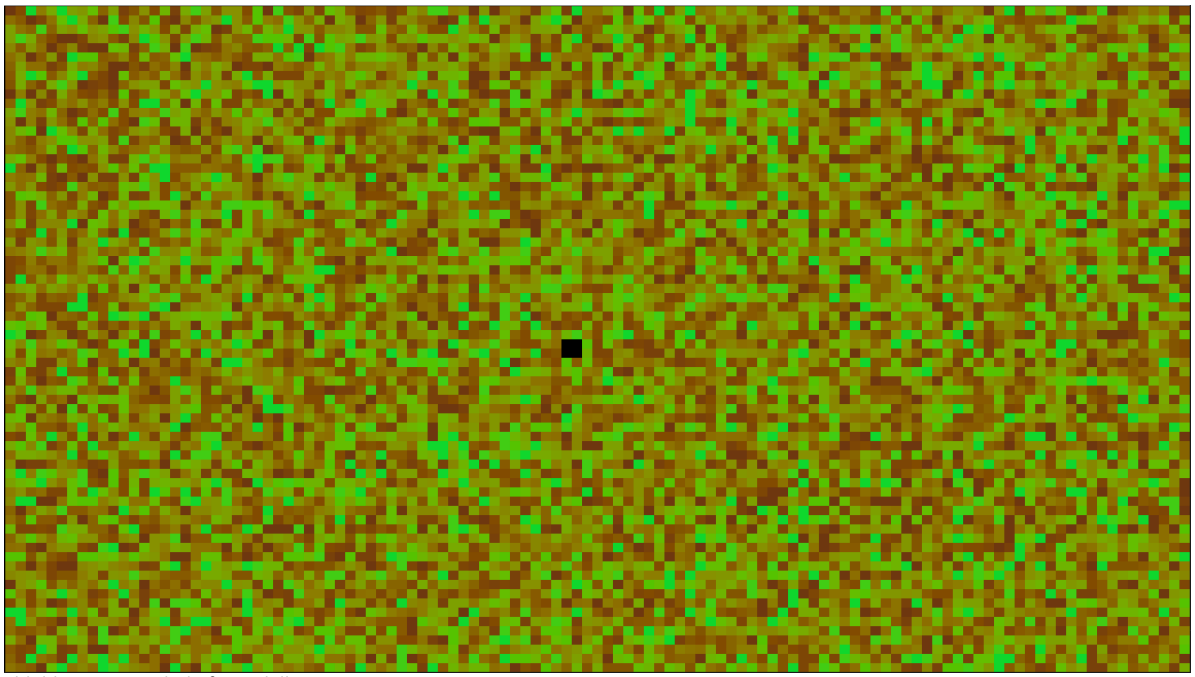


Abbildung 10: Landschaftsmodell

In Abbildung 10 ist das Modell der Landschaft in der Draufsicht dargestellt. Diese besteht aus einem zweidimensionalen Array, in welchem jede Zelle eine zufällige Farbe aus einem Farbpool zugewiesen bekommt. Grüne Zellen sind dabei besonders zur Bepflanzung geeignete Flächen, braun hingegen sind unfruchtbare Bereiche. Der zentrale schwarze Bereich stellt den Hauptrechner dar, welcher in einem Landschaftsabschnitt in die Mitte des abzudeckenden Bereiches platziert wird. Das ist der Startpunkt der Drohnen.

Nun können wir die Anzahl der Drohnen pro Rolle bestimmen. Nachdem wir einen Schwarm erzeugt haben, wird die Simulation gestartet und die Drohnen beginnen einen vorher ausgewählten Bereich zu analysieren (Abbildung 11).

In Abbildung 11 sind Repeater-Drohnen ausgerückt, um das Funksignal zu verstärken. Dieses Verhalten läuft automatisch ab, sobald die Drohnen Gefahr laufen aus dem Funkradius des Hauptrechners zu fliegen.

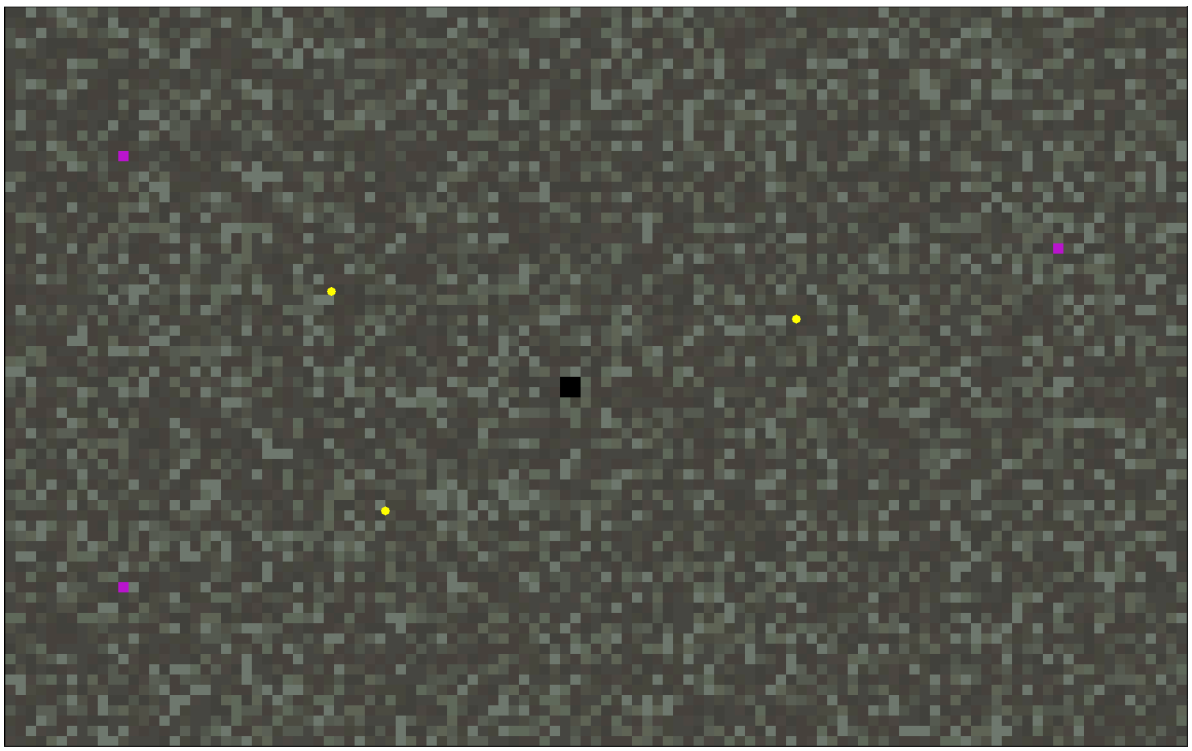


Abbildung 11: Simulation mit ausgewähltem Bereich (pink) und Scout-Drohnen (gelb), entsättigt

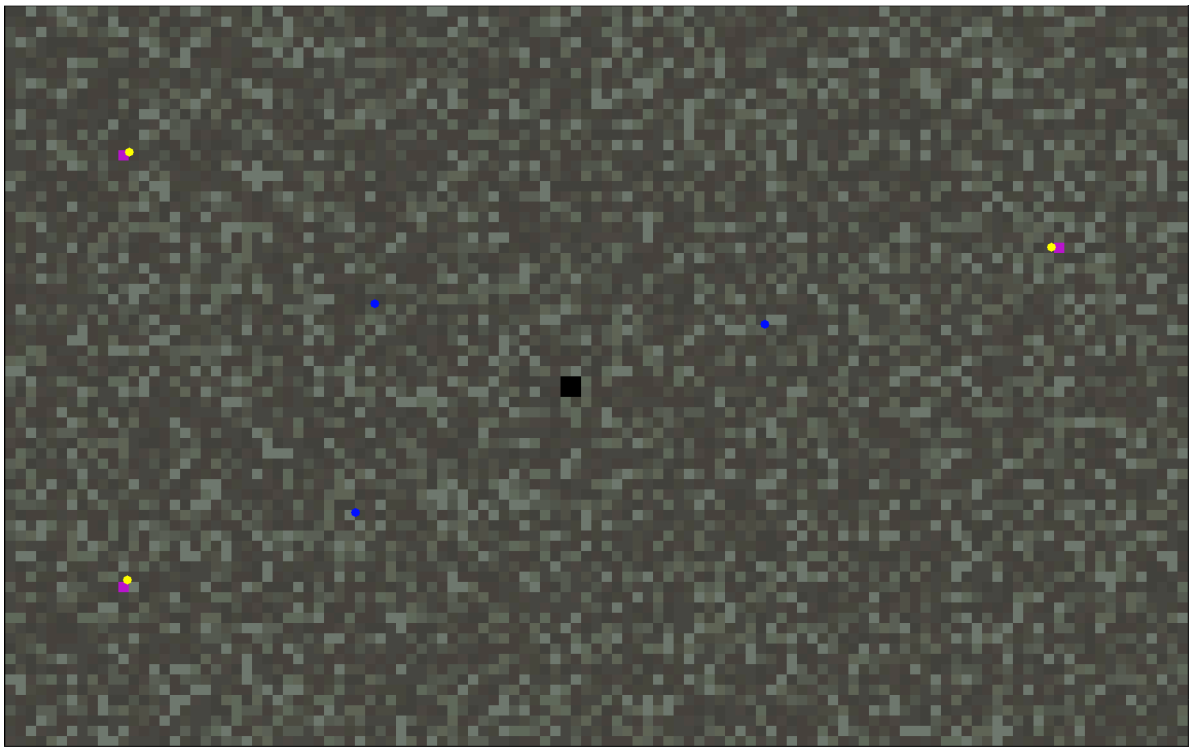
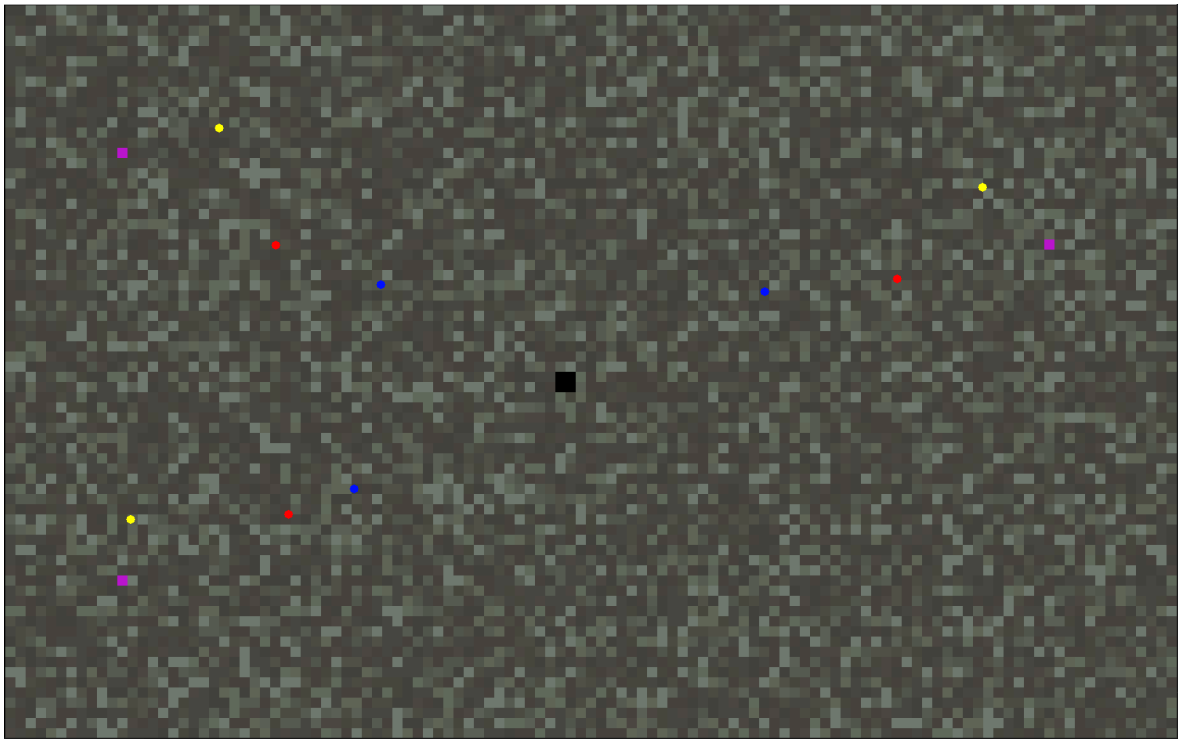


Abbildung 12: Simulation mit Scout-Drohnen im Zielbereich und Repeater-Drohnen (blau), entsättigt

In Abbildung 11 sind Repeater-Drohnen ausgerückt, um das Funksignal zu verstärken. Dieses Verhalten läuft automatisch ab, sobald die Drohnen Gefahr laufen aus dem Funkradius des Hauptrechners zu fliegen.

Abbildung 13: Simulation mit Arbeiter-Drohnen (rot



Sobald sich der analysierte Bereich als geeignet erweist, werden Arbeiter-Drohnen losgeschickt, um die Aufgabe durchzuführen. In diesem Fall bedeutet das, dass der markierte Bereich mit einem Baum bepflanzt wird.

Dieses Muster wird so lang wiederholt, bis jeder Bereich analysiert und gegebenenfalls bepflanzt wurde. Ist die Aufgabe erfolgreich ausgeführt, fliegen die Drohnen automatisch in die Station zurück. Sollten Drohnen während der Durchführung ihrer Arbeit durch zu wenig Akkuladung bedroht werden, fliegen sie ebenso automatisch zurück und werden durch gleichartige Drohnen ersetzt. Dadurch wird eine kontinuierliche Bearbeitung der Aufgabenstellung gewährleistet.

7.3. Anwendung in der Praxis

Wie bereits beschrieben können Teile des Codes in tatsächliche Drohnen eingesetzt werden. Die Berechnungen, welche für das Zurückfliegen bei zu niedrigem Akkuladestand oder das radiale Absuchen eines Zielgebietes nach geeigneter Fläche verantwortlich sind, bestehen aus grundlegenden mathematischen Formeln, die sehr leicht in praxistauglichen Code überführt werden können. Die Analyse der aufgenommen Bilder geschieht in der Realität mithilfe von Machine-Learning Bilderkennungsverfahren. In der Simulation werden zwischen den Drohnen keine Datenpakete ausgetauscht, da sämtliche Parameter wie Position und Landschaft in einer sich wiederholenden Schleife Bild für Bild ausgewertet werden. In der Realität gibt es keine übergeordnete Mechanik hierfür und es müssen Daten zwischen Drohnen und Hauptrechner ausgetauscht werden.

8. Organisation und Datenverarbeitung der Drohnen

Ein weiteres Kernelement des Projekts lag außerdem in der Verarbeitung von Datenströmen und Klassifizierung von eingehenden Bildern sowie eine Möglichkeit auszuloten die Daten zu speichern für die spätere Auswertung. Dazu haben wir uns mit verschiedenen Speicher- und Visualisierungsmethoden beschäftigt und recherchiert. Eine unserer ersten Ideen hier war der Data Lake für die Datenspeicherung. Jedoch stellten wir schnell fest, dass diese Option zu groß für unser Projekt war, da uns die notwendigen Datensätze fehlten und es auch einen gewissen Overkill darstellte. Eine weitere Idee war die Benutzung von Neo4J als Datenbank und Apache Spark als zentrale Datenverarbeitungsmaschine. Wobei unsere Hauptaufgabe die Entwicklung einer Machine-Learning Applikation zur Klassifizierung von Bildern war. Diese sollte im Fall eines empfangen Bildes einer Drohne jenes Klassifizieren und der Drohne mitteilen ob ein Baumsetzling abgeworfen werden sollte oder nicht. Unser Antrieb bestand darin eine möglichst skalierbare Architektur zu entwickeln, um möglichst viele Drohnen Anfragen an den Zentralrechner verarbeiten zu können.

Probleme hatten wir vor allem im Hinblick auf unsere KI bei der Recherche nach einem geeigneten Datensatz und auch die Performanz unsere Computer. Das Training eines CNN würde sehr lange dauern wie wir feststellen mussten, mehrere Stunden bis wir neue Anpassungen an unserem Model machen konnten. Wir benötigten eine geeignete Menge an Bilder unterschiedlichster Areale zum Beispiel: Wasserareale -> Fluss, See, Meer; Bewohnte Gebiete; Ackerland; Wald usw. Die möglichen Klassen zur Klassifizierung schienen schier unendlich. Schließlich begnügten wir uns mit dem frei verfügbaren Eurosat Datensatz, welcher sicher nicht optimal ist, aber dennoch zehn unterschiedliche Klassen bzw. Areale zur Verfügung stellt.

9. Neo4J:

- ✓ Verwendung einer Graphen-Datenbank
- ✓ Schema passt, da es variabel ist
- ✓ Gute Möglichkeit, die Drohne zu visualisieren
- ✓ Beziehungen zwischen den Drohnen sind erkennbar

Kerngedanke war hierbei die erhöhte Lesegeschwindigkeit von Graphen Datenbanken[6] gegenüber anderen zu nutzen umso zum Beispiel die aktuelle Position einer Drohne im Drohnenschwarm zu Broad casten. Dadurch wollten wir im Falle eines Setzlingabwurfs anderen Drohnen „bescheid geben“ diesen Ort nicht mehr besuchen zu müssen. Jedoch war uns nicht klar in wie weit die Datenbandbreite bzw. das Intervall in dem Drohnen annehmen und senden können, also vernachlässigten wir zunächst diese Funktion und bastelten eine generische Datenbank in dem die Koordinaten möglicher Abwurfsorte gespeichert werden.

Wir hatten zuvor ein Toolkit benutzt aus dem Netz benutzt, um die Relationen und Attribute der jeweiligen Drohnen zueinander festzulegen und darstellen zu können.

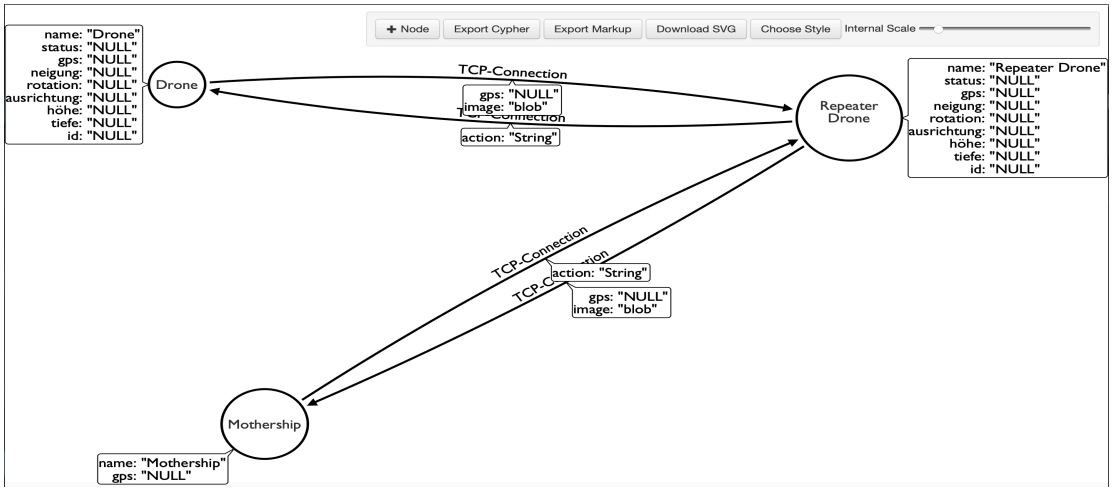


Abbildung 14: Grafische Darstellung der Beziehungen

Diese festgelegten Daten wollten wir mithilfe der Cypher Export Funktion uns kopieren und diese in Neo4J einbinden.

Leider mussten wir schnell feststellen, dass wir in Neo4J nicht das für uns optimale Ergebnis graphisch darstellen konnten, weshalb wir uns an dem Ergebnis aus dem Toolkit orientiert haben.

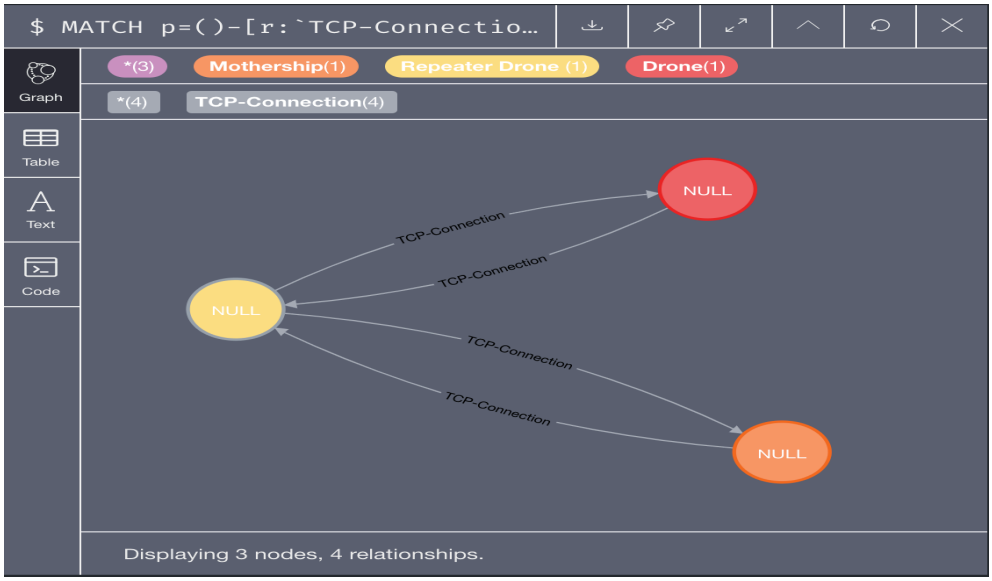


Abbildung 15: Grafische Darstellung in Neo4J

10. Architektur des zentralen Rechners

Im laufe des Projekts hatten wir einige Ideen wie wir eine möglichst skalierbare Architektur entwickeln wollten, um möglichst wenig Wartezeiten für eine Drohne zu gewährleisten. Jedoch starben viele dieser Ideen recht schnell und zwei Möglichkeiten kristallisierten sich heraus. Zum einen das Apache Spark Framework und seine Master-Slave-Architektur und die darin enthaltene Hilfsmittel zur Verarbeitung kontinuierlich anfallender Daten, auch Data-Streaming genannt und verteilte

Datenverarbeitung. Die Zweite Idee war die Entwicklung eines simplen Lokalen REST Servers mit Python Flask und unser Machine-Learning Model mit Kubernetes verteilt zur Verfügung zu stellen. Bei genügend Kubernetes Container könnte so beinahe jede Drohne ihr „eigenes“ Machine-Learning Model zur freien Verfügung haben. Drohnen könnten also einfach einen GET Anfrage an den lokalen Server, da wir ja uns immer im gleichen W-Lan befinden, schicken mit dem Bild als Inhalt und den jeweiligen Koordinaten um diese zu Speichern. Der Server würde dann einfach eine POST Antwort schicken mit dem Inhalt {Abwerfen: „Ja/Nein“}

Wir haben uns recht lange mit Apache Spark beschäftigt und erst etwas später festgestellt das es dabei doch eine recht steile Lernkurve gibt und wir uns eventuell früher mit der REST-Server Option hätten beschäftigen sollen. Nichts desto trotz konnten wir Spark dazu verwenden unser Neural-Net verteilt zu trainieren und so schneller dieses zu testen.

10.1 Apache Spark

Apache Spark ist ein Framework, welches wir nicht nur aus eigenem Interesse gewählt haben, sondern es durchaus viele Gründe gibt, weshalb sich Apache Spark für solch ein Szenario gut eignet:

- ✓ verteilte Verarbeitung von Daten über ein Master-Slave System und so auf herkömmlichen mehreren Computern realisierbar.
- ✓ Trainierte Modelle können über UDFs (User defined functions) als SQL Statement einfach abgerufen werden
- ✓ „In Memory“ Verarbeitung -> hält Daten im RAM, anstatt ständig von der Festplatte zu lesen
- ✓ Stream Processing -> kann kontinuierlich Daten empfangen und automatisch auf Worker Computer verteilen

Graphisch dargestellt müsste man sich den Structured Data Stream etwa so vorstellen:

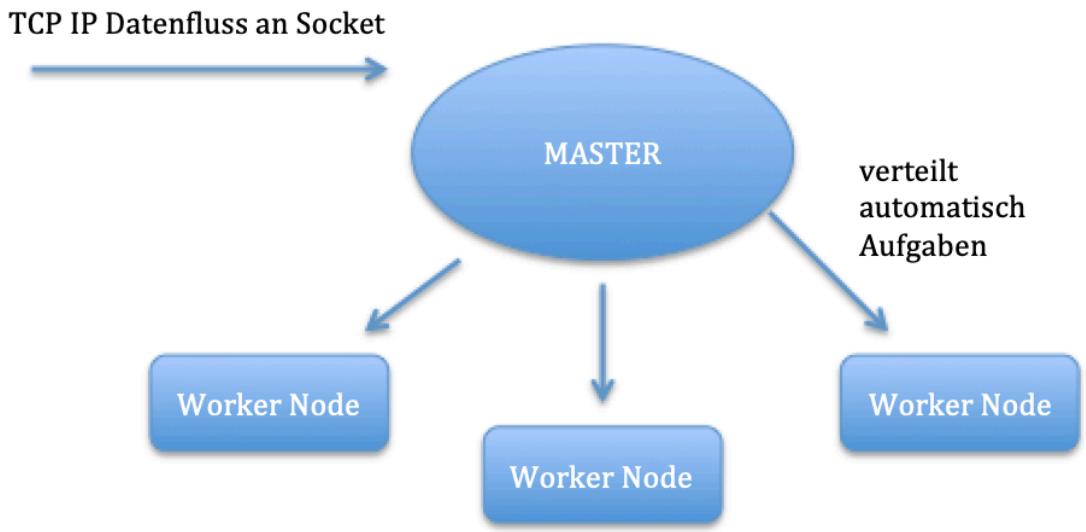


Abbildung 16: Übersicht Structured Data Stream

11. Deep Convolutional Neural Network

Um verstehen zu können was, ein Deep Learning Model macht, muss man zunächst verstehen, was ein Convolutional Neural Network oder auch kurz gesagt CNN ausmacht und wie es aufgebaut ist. Ein CNN ist ein eine Deep Learning Architektur, die sich sehr gut für die Verarbeitung von Bildmaterial eignet. Mithilfe von Convolotional Neural Networks sind wir in der Lage, den Input in Form einer Matrix zu verarbeiten und als Matrix dargstellte Bilder als Input für die Verarbeitung zu verwenden. Normale neuronale Netzwerke haben nicht die Fähigkeit, Objekte in einem Bild unabhängig der Position des Objekts im Bild zu erfassen und zu verarbeiten.

11.1 Aufbau Convolutional Neural Network

Der Aufbau eines Convolutional Neural Network besteht aus folgenden Komponenten[7]:

- Convolutional Layer
- Pooling Layer
- Dense Layer
- Input, Output

Der Input der Matrix wird von einer festgelegten Anzahl sogenannter Filter analysiert. Dabei wird der Filter von Links nach Rechts über die Input Matrix bewegt und der Output ist das Ergebnis von Filter und Input Matrix.

Der Pooling Layer dient zur Aggregation der Ergebnisse der Convolutional Layer, indem er nur das stärkste Signal weiterschickt. Bei unserem Beispiel hat uns der Max Pooling Layer interessiert, da er einfach die höchsten Werte ausgibt und alle anderen und überflüssigen Informationen verwirft. Bei unserem Beispiel von Bilddaten und Standorten bedeutet dies eine Menge an Daten, die einfach nicht mehr gespeichert und verarbeitet werden müssen.

Die Klassifikation findet abschließend mit dem Dense Layer statt. Hierbei ist jeder Knoten im Netzwerk mit der davorigen Schicht verbunden. Bei dieser Klassifikation wird ein individuelles Feature benötigt, ein sogenannter Feautre Vector. Beim sogenannten Flattening findet die Überführung des so entstanden mehrdimdimensionalen Output in einen eindimensionalen Vector statt.

12. Funktionsweise von Convolutional Neural Networks

Mit seinen Filtern erkennt ein Convolutional Neural Network ortsabhängig Strukturen in den Input Daten. Zunächst werden auf der ersten Schicht die Filter von einfachsten Strukturen wie Kanten, Linien und Farbtupfern aktiviert. Hierbei wird die Art der Filter vom Netz selbst gelernt. In der nächsten Schicht werden Strukturen wie Kurven und einfache Formen gelernt. Das Abstarktionslevel des neuralen Netzes ist mit der Filterebene gekoppelt und erhöht sich mit fortschreiten der Filterebene. Backpropagation ist ein Algorithmus, der häufig verwendet wird, um neuronale Netzwerke zu trainieren. Wenn das neuronale Netzwerk initialisiert wird, werden Gewichtungen für seine einzelnen Elemente, sogenannte Neuronen, festgelegt. Eingaben werden geladen, sie werden durch das Netzwerk der Neuronen geleitet, und das Netzwerk stellt eine Ausgabe für jede, unter den anfänglichen Gewichtungen. Backpropagation hilft, die Gewichte der Neuronen so anzupassen, dass das Ergebnis dem bekannten wahren Ergebnis immer näherkommt.

Nun werden wir kurz auf die von uns verwendeten Algorithmen eingehen und warum wir genau diese uns ausgesucht haben.

Activation Functions:

Activation Functions[9] sind ein entscheidender Bestandteil des Deep Learning und bestimmen die Ausgabe eines Deep-Learning-Modells, seine Genauigkeit und auch die Recheneffizienz des Trainings eines Modells, das ein großes neuronales Netzwerk bilden oder brechen kann. Activation Functions haben auch einen großen Einfluss auf die Fähigkeit des neuronalen Netzwerks, zu konvergieren und die Konvergenzgeschwindigkeit, oder in einigen Fällen könnten diese Funktionen verhindern, dass neuronale Netzwerke überhaupt konvergieren.

Activation Functions sind mathematische Gleichungen, die die Ausgabe eines neuronalen Netzwerks bestimmen. Die Funktion wird an jedes Neuron im Netzwerk angehängt und bestimmt, ob sie aktiviert werden soll ("gefeuert") oder nicht, je nach, ob die Eingabe jedes Neurons für die Vorhersage des Modells relevant ist. Activation Functions helfen auch, die Ausgabe jedes Neurons auf einen Bereich zwischen 1 und 0 oder zwischen -1 und 1 zu normalisieren.

Wir verwenden in an allen Layer bis auf den Output Layer die ReLu Funktion und in letzterem die Softmax Funktion. ReLu oder auch Recified Linear Unit ist praktisch DIE Funktion da sie dem Netz ermöglicht schnell zu konvergieren und Backpropagation ermöglicht. Softmax wird immer bei Multi-Klassen Klassifikationen verwendet da sie die Outputs der Knoten auf einen Wert zwischen 0 und 1 normalisiert und diese durch die Summe aller Outputs teilt. So bekommen wir ein Wahrscheinlichkeitsverteilung zwischen 0 und 1. Wobei der Knoten als prognostizierte Klasse ausgewählt wird, der den größten Anteil an der Gesamtverteilung besitzt.

Loss Function:

Als Teil des Optimierungsalgorithmus[8] muss der Fehler für den aktuellen Zustand des Modells wiederholt geschätzt werden. Dies erfordert die Wahl einer Fehlerfunktion, die konventionell als Loss Function bezeichnet wird und die verwendet werden kann, um den Verlust des Modells abzuschätzen, so dass die Gewichtungen aktualisiert werden können, um den Verlust bei der nächsten Auswertung zu reduzieren. Dabei verwenden wir die Categorial Cross Entropy Funktion die kurzgesagt die durchschnittliche Differenz zwischen vorhergesagter und tatsächlicher Klasse summiert. Also je näher die Vorhersage an der tatsächlichen Klasse dran ist desto geringer ist der Verlust bzw. neu Justierung des Netzes.

Optimization Function:

Optimierungsalgorithmen helfen uns, unsere Loss Function zu minimieren oder zu maximieren, die einfach eine mathematische Funktion ist, die von den internen lernfähigen Parametern des Modells abhängt, die bei der Berechnung der Zielwerte (Klassen in Klassifikationsmodellen) aus dem Satz von Vorhersage Variablen im Modell verwendet werden. Zum Beispiel – wir nennen die Gewichte und die Bias Werte des neuronalen Netzwerks als internen lernbaren Parameter, die bei der Berechnung der Ausgabewerte verwendet werden und in Richtung einer optimalen[8] Lösung erlernt und aktualisiert werden, d.h. die Minimierung des Verlusts durch den Trainingsprozess des Netzwerks. Wir verwenden dabei den Adam Algorithmus da er grundsätzlich performanter ist als andere ist.

13. Fazit

Mit Beendigung dieser Arbeit stellt sich die Frage nach Erkenntnissen und potenziellen Aussichten. So gehen wir davon aus, dass diese Technologie, richtig eingesetzt, ein großes Potential hat Probleme verschiedenster Arten zu lösen. Leider war sowohl die Zeit als auch unsere Mittel begrenzt und so mussten oft abstriche gemacht werden oder es kam zu erzwungenen Entwicklungspausen. Aber alles in allem sind wir, den Umständen entsprechend, zufrieden mit der geleisteten und gelieferten Arbeit. Es war für uns von Anfang an klar, dass wir nicht in der Lage sein werden eine ausgereifte Schwarmintelligenz zu kreieren, aber wir haben einen Grundstein gelegt und sind in einigen Bereichen auch weit genug gekommen um es als persönlichen Erfolg zu verbuchen.

Als Grundstein betrachtet glauben wir, dass wir mit dieser Arbeit in der Zukunft in der Lage sein werden sie erfolgreich weiter zu entwickeln.

14. Abbildungsverzeichnis

15. Quellen

[1] https://www.youtube.com/watch?v=a9_0-VKx7D4 (DW Deutsch Youtube Channel, „Erhalt des Baumbestandes: Aufforstung", 20.09.2019)

[2] https://www.tagesschau.de/multimedia/video/video-563147.html (tagesschau.de, „Studie zu Klimaschutz durch Aufforstung: Neue Wälder könnten CO2 speichern", 04.07.2019)

[3] https://stackoverflow.com/questions/9600801/evenly-distributing-n-points-on-a-sphere ("evenly distributing n points on a sphere", answer by CR Drost, 24.05.17)

[4] https://core.ac.uk/download/pdf/11131792.pdf (Franz Andert, Gordon Strickert und Frank Thielecke DLRZ Braunschweig 2006)

[5] Guillen-Perez, Antonio & Sanchez-Iborra, Ramon & Sanchez-Aarnoutse, Juan & Cano, Maria-Dolores & Garcia-Haro, Joan. (2016). WiFi networks on drones. 183 - 190. 10.1109/ITU-WT.2016.7805730.

https://neo4j.com (20.11.19)

[6] Neo4J, Blogs and Tutorials

https://jaai.de/convolutional-neural-networks-cnn-aufbau-funktion-und-anwendungsgebiete-1691/ (22.11.19)

[7] Roland Becker, Februar 2019 Concolutional Neural Networks – Aufbau, Funktionen und Anwendungsgebiete

https://medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c (15.10.19)

[8] Apoorva Agrawal, September 2017, Loss Functions and Optimization Algorithms. Demystified.

https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d (16.10.19)

[9] Uniqtech, Januar 2018, Understand the Softmax Function in Minutes