Navigation eines autonomen Fahrzeugs im Wald - Drohne

Touni Arar, Emanuel Böhm, Martin Heller, Marie Kastning, Jakob Müller

Philipps Universität Marburg
Betreuer: Prof. Thorsten Thormählen

19. Mai 2021

Inhaltsverzeichnis

1 Einleitung

1.1 Thematik und Relevanz

Eine Zukunft in der sich Fahrzeuge komplett eigenständig auf öffentlichen Straßen bewegen werden, scheint nur noch eine Frage der Zeit zu sein. Autonomes Fahren, alternative Antriebe und voll vernetzte Fortbewegungsmittel werden im Individualverkehr die Zukunft sein. Diese Fahrzeuge müssen in der Lage sein maschinell zu lernen und aus diesem Grunde wird die Bedeutung von Software und künstlicher Intelligenz in Verkehrsmitteln immer wichtiger.
Basierend auf dieser Zukunftsvision beschäftigten wir uns in unserem Projekt damit, eine Drohne so zu programmieren, dass sie sich autonom durch einen beliebigen Wald navigieren kann.

1.2 Vorgehensweise und Methoden

Zur Umsetzung wurden uns eine Drohne samt Einplatinencomputer und Tiefenkamera zur Verfügung gestellt. Wir werden, zum Schutz der Drohne anhand einer Simulation, den Path-finding-Algorithmus, der in Python geschrieben ist, testen.

1.3 Ziel

Ziel des Projekts ist es einen Pathfinding-Algorithmus zu schreiben, der es der Drohne ermöglicht ohne einen Zusammenstoß durch einen beliebigen Wald zu navigieren und diesen zu testen.

2 Handhabung / Steuerung der Drohne

2.1 Handhabung der Drohne

Zum Start der Drohne muss der Akku an diese angeschlossen und der Transmitter angeschaltet werden.
An dem Transmitter muss sichergestellt werden, dass sich alle Schalter in hinterer Lage befinden und der rechte Stick (Throttle) sich in unterster Possition befindet.
Die Drohne gibt nach einer kurzen Startfrequenz ein wechselndes Piepen von sich und die LED am Empfänger blinkt blau.
Dann muss der kleine rote Knopf (safety switch) oben am Empfänger betätigt werden. Das Piepen stoppt nun und die Drohne ist einsatzbereit. Der Throttlestick wird für ca. 2 Sekunden nach unten rechts gehalten, um die Drohne in den "Armed" Modus zu versetzen und die Motoren zu starten.

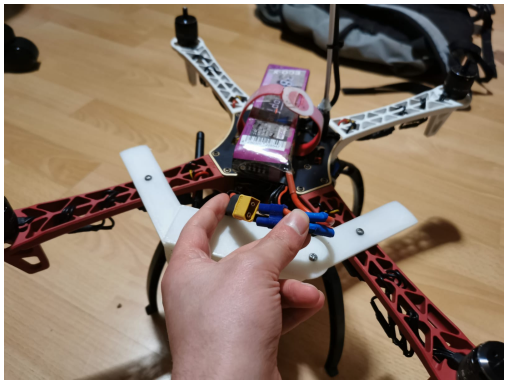
Die Drohne kann in verschiedenen Flightmodi versetzt und gesteuert werden. Zwischen diesen kann mit Hilfe des Transmitters (hintere zwei linke Sticks) oder via Script umgeschalten werden.
Die Flightmodi umfassen unter anderem:

- LOITHER *(Standard Modi für Steuerung via Transmitter)*
- GUIDED *(Standard für Steuerung via Script)*
- LAND *(Die Drohne verliert langsam an Höhe und stoppt die Motoren sobald sie den Boden erreicht hat)*
- ALTHOLD *(Modi zum Halten der Flughöhe, Befehle weiterhin möglich)*

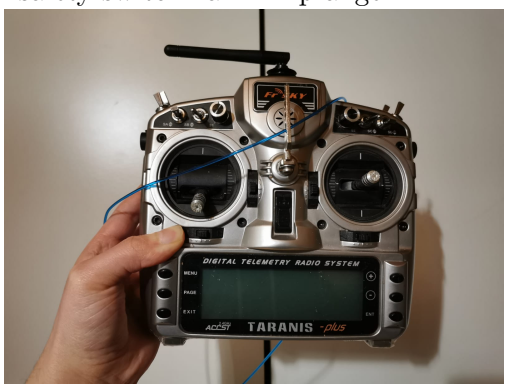
Sollte die Drohne einmal Bruchlanden oder nicht korrekt funktionieren sollte sie sofort über den Transmitter in den Modus *LAND* versetzt werden.

Um Daten der Drohne (z.B mit Hilfe der Software Mission Planer) auszulesen kann der USB Port des Autopilotenmodul genutzt werden (siehe Bild 4 unten).

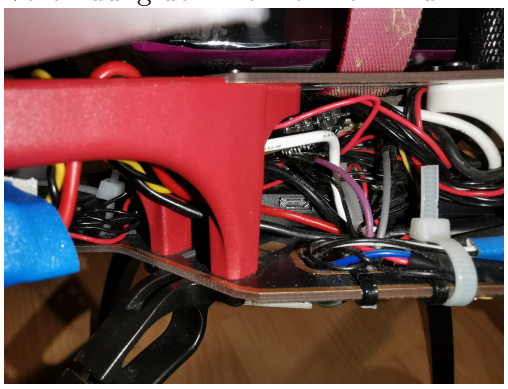
"safety switch" am Empfänger



Verbindung der Drohne mit Akku



Transmitter mit Throttle Stick links



Micro USB Port

2.2 Steuerung der Drohne via Python

Der Raspberry Pi startet sich mit der Stromversorgung der Drohne über den angeschlossenen Akku. Die Stromversorgung über den USB Anschluss funktioniert gegebenenfalls nicht oder nur schlecht, da die weiteren elektronischen Bauteile der Drohne ebenfalls an den selben Stromkreis geschlossen sind und über USB eventuell nicht ausreichend Stromstärken abgegeben werden kann.
Der Raspberry verbindet sich automatisch mit einem Verfügbaren W-Lan mit dem Namen "Kies" und dem Passwort "123456789". Über diese Verbindung kann dieser dann auch über SSH oder VNC mit einem weiteren Gerät im selben Netzwerk (z.B. Notebook) verbunden werden.

Unter ~/Desktop/ss2021/ befinden sich die von unserer Gruppe erstellten Scripts zum Steuern der Drohne. Unter ~/Desktop/Philipp/ befinden sich die wichtigsten Scripts unserer Vorgängergruppe sowie eine Kopie unserer Scripts.
Im Ordner ss2021 sind 6 Dateien zu finden.
Darunter 2 Python scripts welche anschliesend beschrieben werden, daneben 2 log dateien für die jeweiligen Scripts und 2 bash Dateien, welche bei Ausführung das jeweilige Script starten und alle Print Ausgaben in die die jeweilige Log Datei schreiben (zusammen mit dem Datum und der Uhrzeit)

Die beiden Scripts arbeiten jeweils mit der dronekit libary, welche die benötigten Befehle zum einfachen handeln der Drohne und der kommunikation mit dem Autopiloten zur Verfügung stellt. Zusätzlich nutzen wir noch pymavlink über welche man via Mavlink befehle in Abhängigkeit der aktuellen Position der Drohne übergeben kann.

In dem Script testmain.py führen wir einen einfachen Arm befehl durch, welcher nach herstellen einer Verbindung (kann ca 30 Sekunden dauern) die Motoren startet. Anschliesend hebt die Drohne auf eine Höhe von 1,5 Metern ab und hällt diese bis das Script beendet oder der Flugmodus gewechselt wird.

Im Script simpleGoTo.py wird die Drohne ebenfalls gestartet und hebt auf eine Höhe von 2 Metern ab. Anschliesend wird nach kurzer Zeit erst ein SimpleGoTo Befehl ausgeführt welcher die Drohne nur um 1 Meter nach oben fliegen lassen sollte, dann ein Befehl über Mavlink welcher die Drohne erst 2 Meter nach Norden und dann wieder 2 Meter nach Süden fliegen lässt.

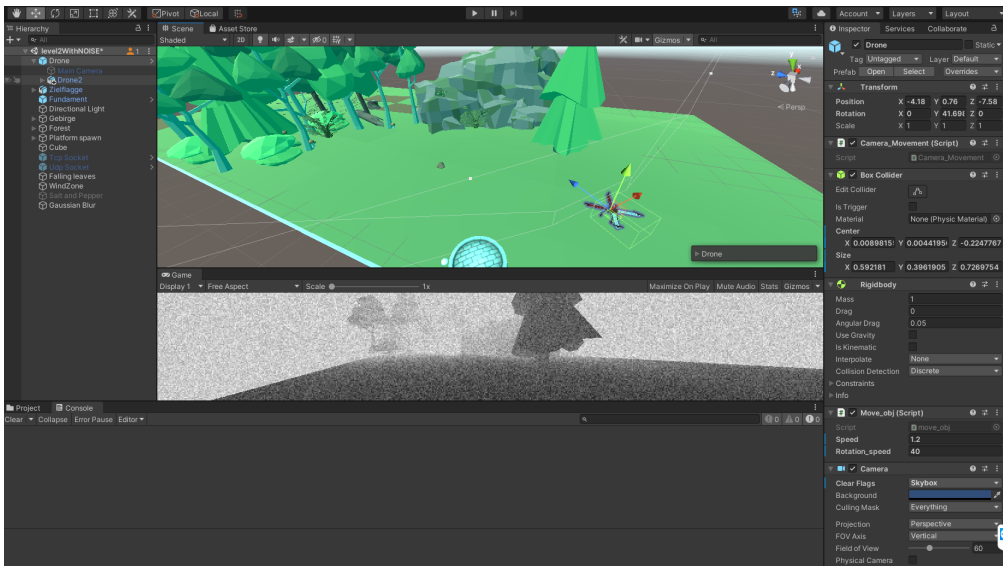
Wichtig ist für alle Befehle, dass die Drohne nur Startbereit ist, wenn der safety switch betätigt wurde und die Drohne über ein ausreichendes GPS Signal verfügt.

3 Unity-Simulation

3.1 Simulation via Unity

Um den Algorithmus des Hindernis-Ausweichens nicht direkt an der Drohne zu testen und somit mögliche Schäden bei auftretenden Fehlern zu riskieren, haben wir beschlossen, mithilfe der Gameengine Unity eine Simulation des Drohnenflugs zu erstellen.
Zu diesem Zweck haben wir eine Testszene erstellt, welche unsere Drohne, eine Fahne als Zielobjekt für den Pathfinding-Algorithmus und ein paar Objekte als Waldlandschaft enthält.
Der Drohne haben wir ein Kameraobjekt zugewiesen, welches mithilfe eines Shaders (siehe nächstes Kapitel) den Output der Tiefenkamera simuliert und direkt an das Pathfindingscript übergibt. Somit sorgen wir für realitätsnahe Bedingungen beim Testen der Fähigkeiten der Hinderniserkennung.

Die Unity Simulation kommuniziert direkt via TCP und UDP (siehe entsprechendes Kapitel) mit dem Pythonscript zum Umfliegen der Hindernisse und bekommt von diesem auch den Input zum Bewegen der Drohne (Wahlweise per Pfeiltasten zum Testen oder automatisiert über Koordinaten) während der Hinderniserkennung.



Unity Simulation der Drohne in der Szene

3.2 Unity Shader

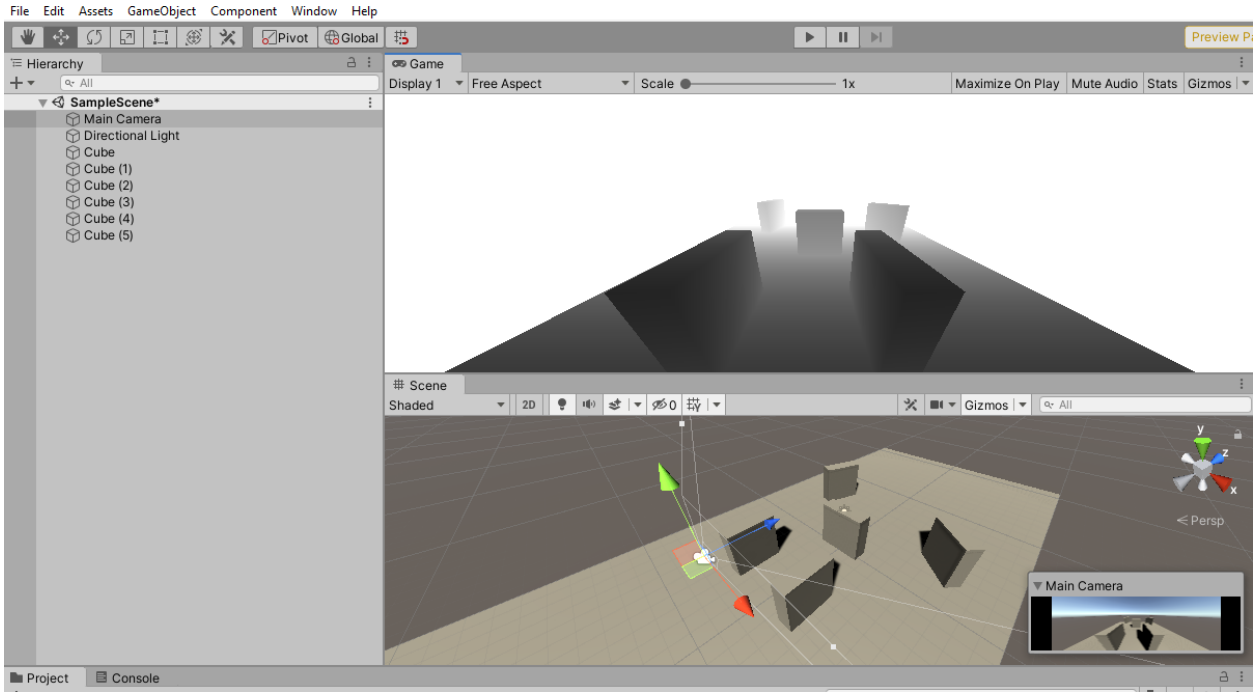
Dem Kameraobjekt werden via C# Script "*ScreenDepthNormal*" zwei Materialien zugewiesen, welche durch Imageshader manipuliert werden. Das erste davon "*Depth_Buffer_Shader*" , welches die Depthnormal Ausgabe als Farbtextur für alle Meshes innerhalb der Near-Far Plane zuweist. Hierbei werden Objekte immer heller, je weiter diese von der near-plane der Kamera entfernt sind. Die Far-Plane markiert die maximale Sichtweite der Kamera, welche von uns nach Tests mit der realen Kamera gesetzt wurde.

Per Buttonpress kann die Ausgabe des Shaders dann als PNG (quasi Screenshot) gespeichert werden (Button x), sowie der Shader komplett deaktiviert werden (Button y)

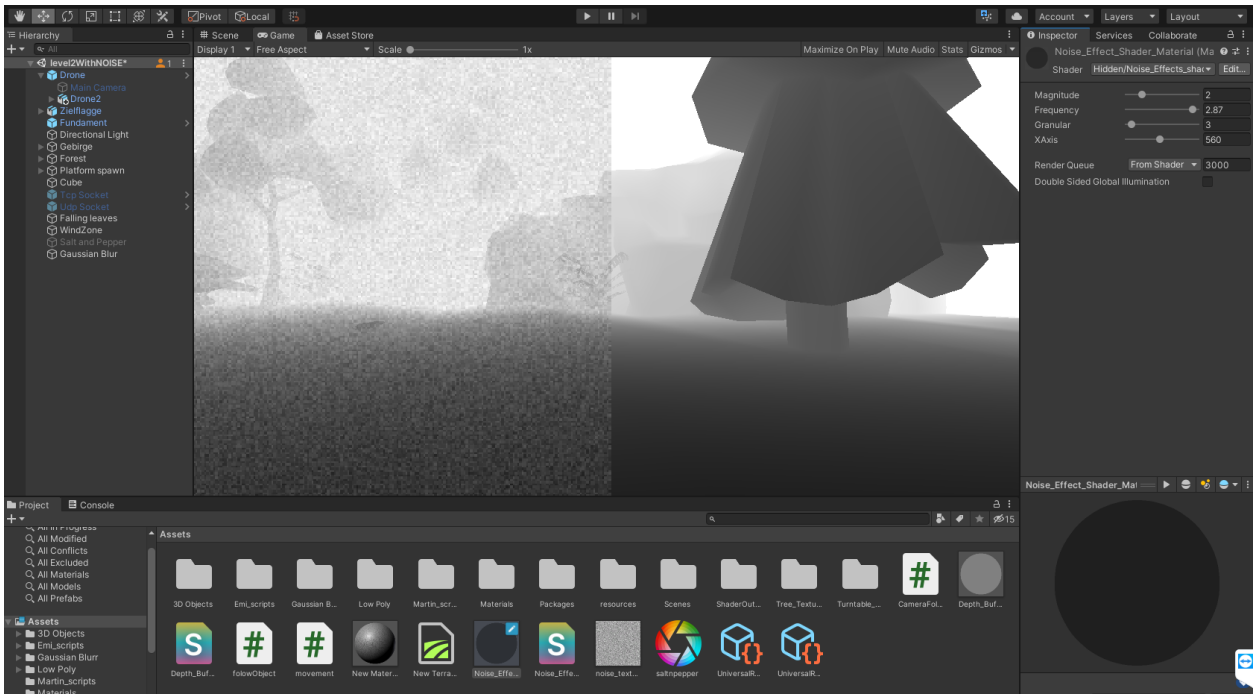
Zusätzlich haben wir, um realgetreue Simulationsverhältnisse zu schaffen, einen weiteren Shader

"*Noise_Effects_shader*" implementiert, um das natürliche Rauschen der Tiefenkamera zu simulieren, um damit zu testen, wie unser Algorithmus auf diese erschwerten Umstände reagieren würde. Hierfür wird ein weiterer Image-Shader auf die Kamera gelegt, welcher (im Inspector Tab des Materials) über weitere Anpassungsmöglichkeiten verfügt.
Es ist möglich die Transparenz des Rauschens (quasi die Stärke) sowie die Grobheit der Körnung, und die Frequenz des Effekts stufenlos anzupassen.

Die beiden Shader manipulieren jeweils ein gleichnamiges Material welches über das oben genannte C# Script mit dem Befehl "Graphics.Blit(source, destination, material)" als Image-Effekt auf die Kamera angewandt wird.



Imageshader für Depth-Effekt ohne Rauschen



Noise Shader zum erstellen eines Rauscheffekts mit Slidern zur Manipulation des Effekts (rechts)
Hier zur Veranschaulichung nur auf halben Bild angewendet

4 Python Code

4.1 Die Main Methode

Wir beginnen damit die *Drohnen* Klasse zu initialisieren. Hierbei wird ein TCP und UDP Kernel gestartet, die nach der Unity Simulation suchen.
Der Timer gibt uns darauf hin Zeit die Simulation zu starten.
Also nächstes initialisieren wir die *PathMap*. Diese verwaltet die zwei Dimensionale-Karte, die sich im Hintergrund merkt, welche Hindernisse wir bereits gefunden haben.

- *pixel_size* gibt an wie viel Meter durch einen Pixel auf der *PathMap* dargestellt werden. Ein Wert von 0.5 bedeutet also, dass $1m^2$ durch 4 Pixel dargestellt wird.
- *radius* gibt an wie viel Rand in alle Richtungen eingefügt werden soll. Die Angabe findet in Metern statt.

Zusätzlich werden zur Initialisierung die Startposition der Drohen und die Zielposition benötigt. Durch diese bestimmt sich die Größe der *PathMap*.

pos_points ist ein Array in dem alle Checkpoints gespeichert werden. Bei der ersten Erstellung wird nur die Zielposition (*dest*) hinzugfügt.

Nun startet die Haupt while-Schleife. Diese Läuft solange, bis die Drohne *dest* erreicht hat.

Die innere for-Schleife geht nun alle vorhandenen Checkpoints durch. Die sleep timer sind nötig, da Unity nur in festen Abständen die Drohnen Daten sendet. Um sicher zu stellen, dass diese immer aktuell sind, finden wir im Code immer wieder sleep times von ca. 0.5sec.

Sollte sich die Drohne nicht in Blickrichtung bewegen können, so rotieren wir sie zunächst. Hier ist zu beachten, dass die Methode *rotate_by()* einen Winkel in Grad nimmt und die Drohne um diesem Winkel rotiert. Explizit wird also nicht einfach der *yaw*, der Drohne, auf den übergebenen Winkel gesetzt. Besitzt die Drohne einen *yaw* von 90 Grad und die Methode *rotate_by(45)* wird auf ihr aufgerufen, so hat die Drohne danach einen *yaw* von 135 Grad. Dieser Winkel würde eine Blickrichtung nach negativ, negativ entsprechen.

4.2 Pos-Points und Checkpoints

Der Pathfinding Algorithmus (AStar) findet in der 2-Dimensionalen Projektion einen Weg von der aktuellen Position zur Zielposition. Ein **Checkpoint** ist eine Pixelkoordinate, an der die Drohne ihre Richtung ändern muss um die Pfad, welcher vom AStar Algorithmus berechnet wurde ab zufliegen. Die Methode *create_checkpoints(path)* (in *aStar.py*) nimmt eine Liste von Tupeln entgegen. Diese Liste ist der Output von AStar. Als Rückgabe erhalten wir eine Liste, in der nur noch die Pixelkoordinaten enthalten sind, an welchen eine Richtungsänderung stattfindet.
Pos-Points werden in *simPathMap.py* berechnet. Die Methode *checkpoints_to_pos(checkpoints, drone_cord)* berechnet die Punkte welche in der Unity-Simulation tatsächlich angeflogen werden müssen. Da wir wissen, wie groß ein Pixel ist und die aktuelle Drohnen Position (in Unity-Einheiten) übergeben, können wir so einen Pfad in *x,y,z* Koordinaten erstellen. Als letztes fügen wir an die Rückgabe noch die Zielposition als finalen Punkt an. Dies wird in der Main-Methode gemacht.

4.3 Simulation-Python Kommunikation

Zur Kommunikation zwischen Unity(C#) und Python wird leider nach momentanen Stand udp **und** tcp genutzt.
tcp wird benutzt um die Kamerabilder zum python script zu schicken
udp wird benutzt um:
-die momentanen Positionsdaten von der Drohne an das python script zu schicken
-Befehle vom python script zu der Unity drohne zu verschicken

Hierbei wurde als Startpunkt für den udp teil das Repository
Python-Unity-Socket-Communication genutzt.

Hierbei haben beide Protokolle die Möglichkeit Bytearrays zu versenden. UDP stellt nicht sicher, dass Packete unbeschädigt, noch dass das Packet überhaupt ankommt; TCP schon. TODO: Der Ursprungsplan war den String im json format zu schicken. Die Json implementation auf der Seite von C# macht hierbei aber einige Probleme und scheint insbesondere nur für Windows zu funktionieren.

In Python kann nun aus der Datei drone.py ein Drohnen Object erstellt werden, welches ein Interface für die Kommunikation zu Unity bereitstellt
auf der Drohne können folgende Befehle ausgeführt werden.

Python → Unity:

- rotate_by angle(float)
- move_by_relative x(float) y(float) z(float)
- fly_forward dist(float)
- move_by x(float) y(float) y(float)

Hierbei sieht zum Beispiel der intern verschickte String um zur Position (0 0 0) zu fliegen folgendermaßen aus:
"{"name":"move_by", "x":0, "y":0, ß":0} "

Unity → Python:

Unity hat die Möglichkeit in regelmäßigen Abständen sowohl das momentane Bild der Camera sowie Positionsdaten zu verschicken. Positionsdaten werden hierbei als 'pos:(x,y,z), angle: a' verschickt.

In Unity findet die Kommunikation zu python durch das Script "move_obj.pyßtatt.
Hier kann z.B auch definiert werden wie häufig Kamerabilder und Positionsdaten versendet werden sollen (InvokeRepeating(delay,delta))

Unity ohne Python:

Sollte aus irgendeinem Grund die Unitysimulation ohne die Pythonscripte benutzt werden wollen, empfiehlt es sich stark das Script "moveobj"von der Drohne, sowie TCPserver und UDPserver zu deaktivieren.

Bugs und Bugfixes:

- Von Zeit zu Zeit kann es sein, dass das System den Port für den TCP Server noch aus dem letzten Durchlauf blockiert. Hier kann entweder der Port in Tcpserver.py sowie Tcpsocket.cs auf einen anderen gesetzt werden, oder der Rechner neu gestartet werden.
- Mit dem Voranschreiten des Projektes kann es sein, dass die Größe des Buffers beim übertragen der Nachrichten nicht ausreicht. Diese können in der jeweiligen Datei geändert werden.

move_obj.cs

Die Datei stellt die Schnittstelle zwischen Unity und dem Tcp, sowie Udp server da. Hauptbestandteil ist die MoveQueue, welche eine beliebige Abfolge an einzelnen
Befehlen(MoveEnums) nacheinander abarbeiten kann.
Kommt ein neuer Befehlt von der Pythonseite,
kann die momentane MoveQ mit dem Befehl this.moveQ.clear() geleert werden.

In der Start methode wird festgesetzt in welcher Frequenz das Bild, sowie die Positionsdaten geschickt werden sollen. Diese können angepasst werden, es sollte folglich aber getestet werden ob das Protokoll mit der neuen Sendegeschwindigkeit funktioniert.

5 Fazit

5.1 Ergebnis

Es wurde eine Simulation erstellt, die eine Waldlandschaft darstellt und mithilfe von “Salt and Pepper-Noise” Störeffekte simuliert.
Der Path-finding-Algorithmus navigiert die Drohne in der Simulation erfolgreich und ohne jeglichen Zusammenstoß zum Ziel.
Für den Notfall wurde ein Abbruchbefehl implementiert um die Drohne auf der Stelle stoppen und den Algorithmus beenden zu können.

5.2 Perspektive

Es bleibt nun noch das Testen und Optimieren des Algorithmus unter wahren Begebenheiten im Wald.