SLAM

Der einer technischen Hauptbestandteile unseres Projekt ist mit den Daten des Lidars unsere Umgebung zu erfassen und sich ihr zu lokalisieren. Die dieser Prozess nennt sich SLAM (Simultaneous Localization and Mapping) und wir haben in einigen Versuchen probiert einen einfachen und zuverlässigen Algorithmus zu finden, der diese Aufgabe löst. Von Anfang an haben wir uns entschieden, dass wir keine vorgefertigte SLAM Algorithmen nutzen wollen. Aktive Forschung in SLAM Algorithmen gibt es schon zeit den 90 igern bis heute, und so haben wir uns entschieden, dass wir uns nicht in die tiefen der aktuellen Forschung werfen, sondern unseren eigenen weg finden wollen. Einer der Hauptgründe für diese Entscheidung ergab sich aus der Tatsache, dass es hauptsächlich entweder nur fertige Algorithmen oder Wissenschaft Paper zu dem Thema gibt. Einfach nur etwas Fertiges zu nehmen war uns zu einfach und langweilig, da wir so keine Ahnung haben würden, wie unser Algorithmus überhaupt funktioniert. Wissenschaft Paper hingegen konnten wir nur teilweise nutzen, da sie für und in der verwendeten Sprache und Mathematik schwer bis unverständlich sind. Es fehlt uns einfach Wissen in Themen wie, Differentialgleichung, Vektorfeldern und Kernelfunktionen. Um mal paar Begriffe zu nennen über die wir gestolpert sind aber nichts mit anfangen konnten. In den nächsten Seiten werden die Ansätze erklärt, die wir genutzt haben, um unseren Algorithmus zu arbeiten. Außerdem wird auf die Hürden und Probleme auf dem Weg eingegangen.

Um in der Corona Zeit unabhängig zu arbeiten können, hat Tim einige roh Daten von seinem Zimmer mit dem Lidar Sensor aufgenommen und in CSV Format abgespeichert. Wir haben pro Messung 100 Umdrehungen aufgenommen, um Messfehler zu vermeiden. Wir haben einem Start Position aufgenommen, und dann weitere punkte mit nur Verschiebung, nur Drehung oder beiden genutzt. Dies half uns den Algorithmus in teil schritten zu testen, indem man sich erst nur auf die Verschiebung oder nur die Drehung konzentrieren konnte.

Unser alle erste Ansatz war die Verschiebung zu errechnen, indem wir das Delta der beiden Mittelpunkte der Punktwolken nutzen.

Hier steht d für das errechnete Delta, n für die menge an Messpunkten in den Punktwolken und p für die Position des Messpunktes.

Dies führe aber schon bei Verschiebungen über 10 cm großen Fehlern. Die Lidar nahm zu viele Punkte auf, die schon bei kleinen Bewegungen nicht mehr gemessen werden. Unsere Idee, um diese Problem zu lösen war nicht alle Punkte zu beachten, sondern nur Gerade Flächen zu nutzen, da diese meistens zu festen Objekten die Wände oder Schränke gehören.

Um die längsten Geraden Flächen im Raum zu finden, haben wir ein Algorithmus geschrieben, der wie folgt arbeitet:

Wir gehen durch jeden Punkt in den Messpunkten. Erzeugen eine Gerade mit dem Punkt und den nächsten in der Liste. Dann gehen wir nach vorne und nach hinten durch das Array durch bis der Abstand dieser Punkte zur Geraden größer als ein Max Wert ist.

Dann gehen wir durch alle möglichen Linien an Punkten durch und nehmen immer die längste, während wir alle doppelten Punkte aus allen anderen Linien raus löschen. Das wiederholen wir bis die größte Linie bis wir alle Linien haben, die mindestens eine bestimmte Anzahl an punkten haben. Durch diesen Algorithmus haben wir nun alle gerade Flächen die der Lidar aufgenommen hat. Doch auch hier macht unsere Delta Rechnung große Fehler, da teilweise verschiedene Messdaten verschiedene Flächen des Raumes enthalten.

Zu diesem Zeitpunkt kamen wir erstmal nicht weiter, bis Tim die Idee hatte, ob wir vierleicht nur die Ecken im Raum vergleichen wollen. Wenn wir die Ecken vergleichen, haben wir mehr daten, da wir Ecken eine Richtung haben, in die sie schauen, so wie den Winkel in dem die beiden Geraden zu einander stehen. Mit diesen Daten kann man probieren immer zwei Ecken paare übereinander zu legen und dann alle anderen Ecken vergleichen.

Also haben wir ein Algorithmus geschrieben, der alle Linien vergleicht. Wenn die Endpunkte der beiden Linien nah aneinander dran sind und der Winkel zwischen den beiden Geraden die die Linien bilden größer als 20° und kleiner als 160° ist haben wir diese als Ecke zusammen gefasst.

Mit der Liste alle Ecken beider Messdaten, haben wir nun die Verschiebung und Drehung ausgerechnet, indem wir alle möglichen Überlageungsmöglickeiten ausprobiert haben. Für jede Möglichkeit haben wir einen Wert ermittelt, der den Fehler angibt und sich aus dem Abstand der anderen Ecken ergibt. Schließlich ist nun die Überlagerungsmöglickeit die den kleinsten Fehler hat die hoffentlich richtige, so haben wir diese genutzt, um die Position des Roboters relativ zu seiner letzten Position zu ermitteln.

Weitere Probleme:

In unserem Ansatz haben wir oft mit bruth force wegen gearbeitet, welches teilweise zu performace Problemen geführt hat. Gerade das errechnen alle möglichen Überlagerungsmöglickeit braucht seine Zeit. Daher haben für diese Rechnungen ein Multithreadsystem genutzt. In Unity ist der Hauptthread für Graphik und alle Scripte verantwortlich, daher Verlagen wir die ganze Errchnung der neuen Position in einen Hintergrund Thread, welcher über mehrere Frames arbeiten kann. Hierfür haben wir ein Task System gebaut mit Flags die angeben in welchem Schritt die Errechnung ist. Außerdem haben wir ausprobiert bei dem errechnen alle möglichen Überlagerungsmöglickeit nochmal ein Thread pro Möglichkeit zu starten. Das starten der Threads hat aber länger gedauert, als die wirklich Rechnung, daher haben wir uns auf eine langsame Hintergrund Aufgabe geeinigt. Der zweite Grund für diese Entscheidung war, die Überlegung, das unser Programm später auf Handys oder Ipads laufen sollen und die haben eh nicht so viele Kerne, als das sich tausende an Parabeln Aufgaben lohnen würde.