SLAM

Einer der technischen Hauptbestandteile unseres Projekt ist mit den Daten des Lidars unsere Umgebung zu erfassen und sich ihr zu lokalisieren. 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 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 entspricht unserer Vorstellung, von diesem Projekt und so haben wir in Kauf genommen, dass unser dies unser Projekt wesentlich schwieriger machen würde. Wissenschaftliche Paper hingegen konnten wir nur teilweise nutzen, da die verwendeten Sprache und Mathematik für uns schwer bis unverständlich ist. Es fehlt uns Wissen in Themen wie, Differentialgleichung, Vektorfeldern und Kernelfunktionen, um mal paar Begriffe zu nennen über die wir gestolpert sind und erstmal nichts mit anfangen konnten. In den nächsten Seiten werden die Ansätze erklärt, die wir genutzt haben, um unseren Algorithmus zu erarbeiten. Außerdem wird auf die Hürden und Probleme auf diesem Weg eingegangen.

Um in der Corona Zeit unabhängig von ein Ander zu arbeiten 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, nur Verschobene Punkte, nur gedrehte Punkte, und verschobene und gedrehte Punkte. Dies half uns den Algorithmus in Teilschritten 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 zu großen Fehlern. Der 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. Diese gehören meistens zu festen großen Objekten, welche man auf mehreren Messungen aufgenommen haben muss.

Der Algorithmus zum finden dieser Flächen arbeitet wir folgt:

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 die Liste durch bis der Abstand dieser Punkte zur Geraden größer als ein Max Wert ist.

Danach 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 zu wenig punkte hat oder zu kurz ist. 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, da die teilweise neue Flächen erkannt werden, welche nicht zu ortbar sind. Tim hatte nun die Idee statt den Flächen die Ecken im Raum zu vergleichen. Mit Ecken haben wir mehr Daten zum mappen. Ecken haben eine Richtung und Winkel. Diese Informationen helfen zwei gleiche Ecken auf den Messdaten zu finden.

Also haben wir ein Algorithmus geschrieben, der alle Flächen vergleicht. Wenn die Endpunkte der beiden Flächen nah aneinander dran sind und der Winkel zwischen den beiden Geraden die die Flächen 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. Diese haben wir für die neue Position und Drehung des Roboters genutzt.

Weitere Probleme:

In unserem Ansatz haben wir oft mit bruth force wegen gearbeitet, welches teilweise zu Performance Problemen geführt haben. Gerade das errechnen alle möglichen Überlagerungsmöglickeiten 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 Errechnung der neuen Position in einen Hintergrund Thread, welcher über mehrere Frames arbeiten kann. Hierfür haben wir ein Task System gebaut. Dieses nutzt enum Flags um anzugeben, in welchem Schritt der Errechnung das Programm ist und ob wir die Daten in den Hauptthread einbinden können.

Außerdem haben wir ausprobiert bei dem errechnen alle möglichen Überlagerungsmöglickeiten 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 Hauptthread entschieden. Der zweite Grund für diese Entscheidung war, die Überlegung, das unser Programm später auf Handys oder Ipads laufen sollen und die haben nicht genug Kerne, als das sich tausende an Parallele Aufgaben lohnen würden.

Was bisher beschrieben wurde ist Version 1 unseres Algorithmus. Sie funktioniert zuverlässig wen bestimmte Anforderung gegeben sind:

1. Wenn zu jedem Zeitpunkt drei oder mehr Ecken des Raumes von dem Lidar aufgenommen werden.
2. Wenn teile des Raums nicht durch bewegende Objekte wie Menschen verdeckt werden.
3. Wenn der Raum nicht spiegelbar ist.

Zu diesem Zeitpunkt in unserem Projekt wurde uns klar, dass wir auf schon vorhandene Forschung zurückgreifen müssen, um die obigen Probleme zu lösen. So kamen wir zu dem oben genannten Problem, dass die meisten Informationen in Form von Wissenschaftlichen Papern vorhanden ist. Diese verstehen wir teilweise nicht. Trotzdem haben wir ein Paper gefunden, welches ein relative simplem, aber funktionsfähigen Algorithmus vorstellt.

Wir werden hier nicht auf den gesamten Inhalt des Papers wieder geben, sondern nur erklären welche Ideen und Ansätze wir von diesem in Version 2 unseres Algorithmus übernommen haben. In dem Paper wurden alle Messpunkte in einer Liste an zwei Dimensionalen Texturen gespeichert. Die Texturen unterscheiden sich in ihrer Auflösung. Die Auflösungen gehen von der maximalen Genauigkeit in unserem Fall 1 cm pro Pixel zu der maximalen Verschiebung des Lidars pro update, in unserem Fall 64 cm pro Pixel. Jede Textur hat die halbe Auflösung als die davor.

Mit zwei Funktionen aus dem Paper kann man einem Vector durch Differentiale Ableitung errechnen, der in die Richtung der nächsten Wand zeigt. Die einzige Vorgabe, ist, dass der Messpunkt nur ein Pixel von der Wand entfernt ist. Deshalb werden die verschiedenen Auflösungen benötigt. Wir fange mit der gröbstem an und verschieben bis wir die Position der geringsten Verschiebung gefunden haben. Dann gehen wir zunächst genaueren Textur und wiederholen die Prozedur. Durch jede Textur wird die Position genauer, bis sie schließlich bis auf 1 cm der Wirklichkeit entspricht.

Der Algorithmus hat die Verschiebung sehr genau errechnet, aber wir hatte keine Ahnung wie wir die Drehung errechnen sollten. In dem Ansatz des Papers wurde die Drehung als dritte Dimension behandelt und genauso optimiert, wie die beiden Karten Dimensionen. Dies hat bei uns aber nicht funktioniert und so waren wir in einer Sackgasse. Deswegen haben wir nie eine Funktionierende Version mit diesem Ansatz gebaut, dennoch sind die Ansätze in dem Paper sehr viel versprechend und es ist unser Plan sie in späteren Versionen so weit wie wir sie verstehen zu nutzten.