Praktikumsbericht

Automatische Ausmessung von Stahlrohrstapeln mithilfe von Smartphones

Antonio Noack

Nov. 2019 - Feb. 2020

Inhaltsverzeichnis

1 Einleitung **1**

2 Hauptteil **2**
2.1 Tiefenmessung . . . 3
2.2 Ellipsenfinden und -Vermessen . . . 4
2.2.1 Hough-Transformation . . . 4
2.2.2 Kanten-Läufer . . . 5
2.2.3 Spider-Algorithmus . . . 5
2.2.4 Watershed-Segmentierung . . . 6
2.2.5 Suchen von langen, kontrastlosen Streifen . . . 6
2.2.6 Deep Neural Networks . . . 7
2.2.7 Fast Radial Symmetry . . . 7
2.3 Ellipsenoptimierung aus Einzelpunkten . . . 9
2.4 Eine erste vollständige Messung . . . 9
2.4.1 Motivation . . . 9
2.4.2 Experiment-Aufbau . . . 9
2.4.3 Ergebnisse und Schlussfolgerungen . . . 10
2.5 Testaufbau . . . 12
2.6 Messungen mit dem Intel RealSense D435 . . . 15
2.7 Messungen mit dem Pico Flexx . . . 17
2.8 Textur-Segmentierung . . . 21
2.9 Finden der Rohrstärke . . . 21
2.10 Rohr-Clustering . . . 21
2.11 Ergebnisse im Testaufbau . . . 23
2.12 Zukünftige Verbesserungen . . . 24

3 Schluss **24**

1 Einleitung

Meine Aufgabe bei der Zeiss Gruppe im Bereich Embedded Systems war das Ausmessen und Auszählen von mehreren Dutzend Stahlrohren auf einer Palette mithilfe eines Smartphones. Von Interesse war dabei Radius, Länge und Wandstärke, sofern möglich. Ursprünglich habe ich mich für eine andere Aufgabe in der Abteilung beworben, aber im Vorstellungsgespräch wurde mir noch diese vorgeschlagen, da ich mich für Computer Vision interessiere und das auch in meine Bewerbung geschrieben habe. Die Aufgabe war zu dem Moment noch nicht ausgeschrieben, gerade erst in der Vorbereitungsphase.

Das Praktikum habe ich als großartige Chance gesehen, mich mehr und intensiver mit Computer Vision, teilweise Machine Learning, befassen zu können.

Zeiss als potentiellen Praktikumsbetrieb habe ich mir dabei wegen ihrem tollen Ruf als Hightech-Unternehmen und guter Arbeitsgeber ausgesucht.

2 Hauptteil

Das Problem ist also die Ausmessung von Stahlrohren mit der Hilfe der Smartphone-Kameras. In dem Bereich gibt es bereits einige Apps, die Baumstämme ausmessen. Stahlrohre haben gegenüber Baumstämmen folgende Vor- und Nachteile:

Stahlrohre sind nahezu perfekt rund, ganz im Gegensatz zu Baumstämmen, die oft nicht einmal elliptisch sind. Ihre Kanten sind in der Regel scharf, Ringe nicht.

Dafür sind die Materialien Rinde und Holz selbst generell einfacher als Stahl zu erkennen: Holz hat im Gegensatz zum reflektierenden Stahl eine halbwegs gleichmäßige Färbung, nach der man gut suchen und dank Ringe, wenn der Stammschnitt intakt ist, auch segmentieren kann.

Stahl kann je nach Oberfläche sehr verschieden aussehen: Glatter Stahl reflektiert. Matter, rauer Stahl nicht. Rohre spiegeln diese Eigenschaften auf verschiedene Art und Weise wieder: Rohre aus glattem Stahl erscheinen, wenn man nicht hindurch guckt, in der Mitte sehr dunkel, ohne längerem Helligkeitsverlauf.

Abbildung 1: Das Innere von glatten Stahlrohren ist meist sehr dunkel

Matte Rohre hingegen werden vom Rand zur Mitte gemäßigt dunkler, je nachdem, wie tief man im Rohr ist. Reflektionen spielen kaum noch eine Rolle. Die Außenwand eines matten Rohres hat wenig Kontrast, wodurch es schwierig sein kann, die genaue Kante im Bild zu finden.

Abbildung 2: Ausschnitt eines matten Rohres

Das glatte Rohr hingegen zeigt ein komplett anderes Bild: durch die runde Form des Rohres werden Objekte so in das Auge des Betrachters reflektiert, dass die Objekte, wenn man nicht zu nah dran steht, in der Höhe stark gestaucht werden. Ist man weit genug weg, sind besonders auffällige Objekte nur noch Linien. Das führt zu starken Bildgradienten entlang des Rohres. Schräge Objekte können die Verlaufsrichtung des Rohres jedoch verfälschen, wenn man versucht die genaue Ausrichtung anhand dieser Streifen zu bestimmen.

Stahlrohre haben aber nicht nur diese zwei Typen des Aussehens, sondern auch den ganzen Bereich zwischen ihnen. Glatte oder sehr raue Rohre sind dabei nicht selten, wie man es vom Anfang und Ende eines Spektrums sonst erwarten könnte.

Hölzer haben auch Variationen, besonders in Farbe und Helligkeit, doch die Farbe ist für einen Stamm relativ gleich und es kommt nie zur Reflexion von anderen Objekten im Bild des Holzes.

Beim Vorstellungsgespräch war meine erste Idee zur Vermessung Photogrammetrie, also die Aufnahme mehrerer Bilder und daraus die 3D-Rekonstruktion der Szene. In ihr könnte man dann die Rohre finden und vermessen. Gewöhnliche Methoden werden aber vermutlich Probleme mit diesem Ansatz haben, da die Merkmale auf reflektierenden Oberflächen nicht stillstehen. Echte Kanten bleiben erhalten, aber die Kanten auf der Oberfläche verfälschen die Daten.

2.1 Tiefenmessung

Selbst wenn man eine Methode aus Bilddaten hat, wird einem eine Kamera allein niemals die Skalierung der Szene verraten, weil Kameras die Welt perspektivisch sehen, doppelt so weit entfernte Objekte einfach halb so groß sind.

Die einfachste Lösung ist ein Referenzobjekt. Einen Rohrdurchmesser händisch auszumessen bietet sich an. Das System soll jedoch möglichst automatisch sein, also probieren wir die Skalierung auf anderen Wegen zu bestimmen.

Eine einfach scheinende Lösung dafür ist das Messen der Tiefeninformation während man die Bilder aufnimmt. Hat die Tiefeninformation eine hohe Auflösung, können schon wenige Bilder dazu ausreichen, die Szene komplett zu vermessen.

Ein anderer Ansatz ist die Größe über den Beschleunigungssensor des Smartphones zu bestimmen. Die allermeisten besitzen einen. Die TUM hat im Frühling 2018 dazu eine Photogrammetrie-Methode vorgestellt. Diese basiert jedoch auf direkter Odometrie, die wegen dem Rolling-Shutter-Effekt nicht für Smartphone-Kameras geeignet ist. Der Rolling-Shutter-Effekt verzerrt Objekte, die sich im Bild schnell seitlich bewegen.

Die App "Measure" von Google nutzt vermutlich auch die IMU, um Größen zu messen. Kritiken sagen jedoch, dass die Genauigkeit manchmal stark abweicht, was problematisch für unsere Anwendung ist. Beschleunigungssensoren sind so oder so problematisch zu verwenden, da ihr

Positionsfehler quadratisch mit der Zeit wächst. Gegen Ende meines Praktikums, ausgerechnet in einer Vorführung, hatte ich diese Größenabweichung von gut 20% auch. Vermutlich habe ich ARCore nicht lang und gut genug initialisiert, oder ARCore hat unter manchen Umständen Probleme, die Skalierung der Szene ordentlich zu erkennen.

Also erst mal zurück zum Tiefenmessen: obwohl viele Handys bereits ein Dual-Kamera-Setup besitzen, unterstützen längst nicht alle den Zugriff auf diese aus Apps heraus, die nicht vom Hersteller stammen. Dazu gehört zum Beispiel mein eigenes Honor 10. Glücklicherweise lassen sich auf diesem im "Aperture"-Modus Bilder anfertigen, die an die jpg-Datei hinten die Tiefeninformation in 8-Bit Genauigkeit, leider ohne Dithering, anhängen.

Die Tiefe lässt sich aus zwei Bildern aus versetzen Kameras aus dem Parallaxeffekt und dem Linsenabstand berechnen. So sehen ja auch wir Menschen Tiefe. Zwei andere Methoden zum Tiefenmessen sind Time Of Flight (TOF) und per Dotprojector. TOF sendet Lichtpulse aus, empfängt sie wieder, und misst die Zeit des "Lichtfluges", um daraus die Tiefe zu bestimmen. Das funktioniert zwar gut für matte Oberflächen, aber Spiegel, wie zum Beispiel glatte Stahlrohre, verfälschen das Ergebnis total.

Dotprojectors werden in manchen iPhones verwendet, um das Gesicht zu scannen, und sind kommerziell zum Beispiel mit dem Intel Realsense D435 (siehe Abb. 2.6), den ich zum Testen hatte, erhältlich. Sie projizieren ein infrarotes Punktemuster auf die Szene, das mit üblichen Kameras ohne Filter noch gut sichtbar ist, und versuchen dann im Bild jeweils den Abstand dieser Punkte voneinander zu bestimmen.

Weil dieses Verfahren jedoch aktiv ist, die Szene angeleuchtet werden muss, kann das im direkten Sonnenlicht problematisch sein, was nicht so ein großes Problem in unserem Anwendungsfall wäre. Jedoch reicht die Tiefenmessung wegen der Strahlungsleistung meist auch nur rund 5m, was durchaus ein Problem für lange Rohre wäre, abgesehen davon, dass glatte Rohre die Punkte einfach wieder reflektieren und damit die Messung wieder verfälscht oder gar unmöglich ist.

Das Dual-/Multi-Kamera-Setup ist also erst mal das vielversprechendste.

2.2 Ellipsenfinden und -Vermessen

Am Anfang meines Praktikums habe ich erst mal damit begonnen, verschiedene Methoden zum Kreisfinden und Ausmessen auszuprobieren, da ihr Radius eine Messaufgabe ist und diese Messung zum relativen Radius der Rohre zueinander führt. Das projizierte Bild von Kreisen sind Ellipsen, also habe ich mich zugleich auch eher auf den allgemeineren Fall von Ellipsen konzentriert. Die Parametrisierung von zwei Brennpunkten und den zwei Achsen scheint mir dabei am praktischsten, da die Fläche dadurch am einfachsten Berechenbar und die Ellipse am einfachsten visualisierbar ist.

Praktisch alle Ansätze arbeiten auf den Kanten, da sie in meinen Beispielen oft relativ deutlich zu sehen sind, auch wenn sie praktisch nie nur die Kanten der Kreise sind.

2.2.1 Hough-Transformation

Die wohl bekannteste Methode zum Kreisfinden ist die Hough-Transformation. Für Kreise ist sie jedoch schon nur noch bedingt verwendbar, da die den 3-dimensionalen Parameterraum auffüllen muss. Für Ellipsen ist es deutlich schlimmer, da es hier fünf Parameter sind. Auf einem Smartphone ist dies nur noch mit viel Geduld berechenbar. Ich selbst dachte mir, dass ich ja die Richtung des Gradienten nutzen könnte, um ein besseres Ergebnis zu bekommen; auf Kreise beschränkt, weil weiterhin Ellipsen schwer berechenbar sind. Das Ergebnis davon war zwar besser als die reine Hough-Transformation, aber starke Ellipsen hatten natürlich Probleme. Einiges später bin ich noch auf das Paper "Fast Radial Symmetry" [1] gestoßen, das es noch extremer handhabt: Die Richtung des Gradienten wird nicht nur als Gewicht verwendet, sondern nur ein Punkt pro Radius von Interesse wird gesetzt. "Generalized Fast Radial Symmetry" [2] ist noch ein klein wenig vielversprechender, da es verspricht, auch Ellipsen in kaum mehr Zeit

zu finden. Doch so, wie ich das Paper verstanden habe, wird effektiv nur das Bild transformiert, sodass es wieder nur Voting ist, und je mehr Kreisprojektionen man untersuchen möchte, der Algorithmus wieder viel teurer wird.

2.2.2 Kanten-Läufer

Ein anderer Ansatz, den ich mir ausgedacht habe, ist es, an Kanten entlang zu laufen, Punkte zu sammeln, und dann aus diesen Punkten die Ellipse über einen Optimierer zu berechnen. Für perfekte Kanten erzielt dieser Algorithmus auch sehr gute Ergebnisse mit teilweiser Subpixel-Genauigkeit, doch sind die Kanten oft leider nicht perfekt. Probleme machen auch Kreuzungen durch angrenzende Rohre, denen man jedoch aus dem Weg gehen kann, indem man versucht, die Richtung entlang der Linie beizubehalten.

Findet man auf seinem Weg keine weiteren Kantenpunkte, wird abgebrochen, und versucht mit den gesammelten Punkten die Form zu extrapolieren.

Abbildung 3: Ausschnitt des Ergebnisses des Kanten-Läufers

2.2.3 Spider-Algorithmus

Angelehnt an diese Methode habe ich mir eine besonders schnelle ausgedacht, die jedoch einige Glättung der Kanten erfordert, was wiederum etwas Zeit benötigt: von Punkten, die möglichst weit weg von Kanten sind, geht man in N Richtungen, bis man eine Kante gefunden hat. Von den potentiellen Kandidaten, Rauschen und Unregelmäßigkeiten können zu Ausreißern führen, muss man sich die am besten zusammenpassenden heraussuchen und dann kann man die Punkte wieder zu einer Ellipse optimieren. Ein RANSAC-Algorithmus (Random Sample Consensus) wäre hier vermutlich besonders gut. Etwas später habe ich zwar einen implementiert, aber ich habe ihn noch nicht mit dem Spider-Algorithmus ausprobiert, da ich zufrieden genug mit meinem bis dahin gefundenen Algorithmus war.

Wegen der Form der Pixel, die die Figur abtastet, habe ich sie Spider-Algorithmus genannt. Findet man die richtige Kante, erreicht man auch hiermit Subpixel-Genauigkeit. Ohne RANSAC tendiert der Algorithmus bei teilweise starken und teilweise schwachen Kanten eines Rohres zu versagen. Bei schräger Lichteinwirkung passiert das gerne, 2x oder 4x pro Rohr. Den Median der Längen zwischen den nächsten Nachbarn hilft aber nur bei kleinen Lücken. Ist der Bereich, über dem der Median gebildet wird, zu groß, so wird die Ellipsenform verfälscht. Ich habe hier den Median und nicht den minimalen Wert genommen, weil ich außerdem noch Probleme

mit scheinbaren Kanten in der Mitte des Rohres hatte. Einzelne, kleine Radien wurden damit automatisch mit verworfen.

In der Abbildung 4 sieht man schön die richtig erkannten Ellipsen und die Genauigkeit, wie aber auch Probleme mit Kontrast im Kreisinneren.

Abbildung 4: Ein Debugging-Bild des Spider-Algorithmus. Die schwarzen Punkte sind Marker, wo der Spider-Algorithmus Kanten detektiert hat. Grüne Ellipsen stehen für einen hohen Score, rote für einen niedrigen.

2.2.4 Watershed-Segmentierung

Außerdem habe ich Watershed-Segmentierung probiert, wobei das Ergebnis aber nicht so super toll war. Die Laufzeit vom Watershed-Algorithmus ist zwar $O(Pixelzahl * 2^{BitsProGraustufe})$, aber bei 256 Graustufen ist das trotzdem ziemlich teuer.

Abbildung 5: Original (Ausschnitt) und zugehörige Watershed-Segmentierung.

2.2.5 Suchen von langen, kontrastlosen Streifen

Und von links nach rechts, sowie von oben nach unten zu gehen, und zusammenhängende Bereiche mit wenig Kontrast zu finden; dann die überlappenden Bereiche der beiden zu be-

trachten: leider war mein Testbild, wie vermutlich auch die Bilder der Realität, zu verrauscht, als dass die Methode erfolgreich wäre. Wäre sie es, wäre sie super schnell, mit einer Laufzeit von $O(Pixelzahl)$.

In Abbildung 6 sieht man, wie einige Kreise ohne Störungen im Kreisinneren gut von allen drei Richtungen als Zentrum erkannt werden, und aber auch andere wegen Störungen nicht gefunden werden.

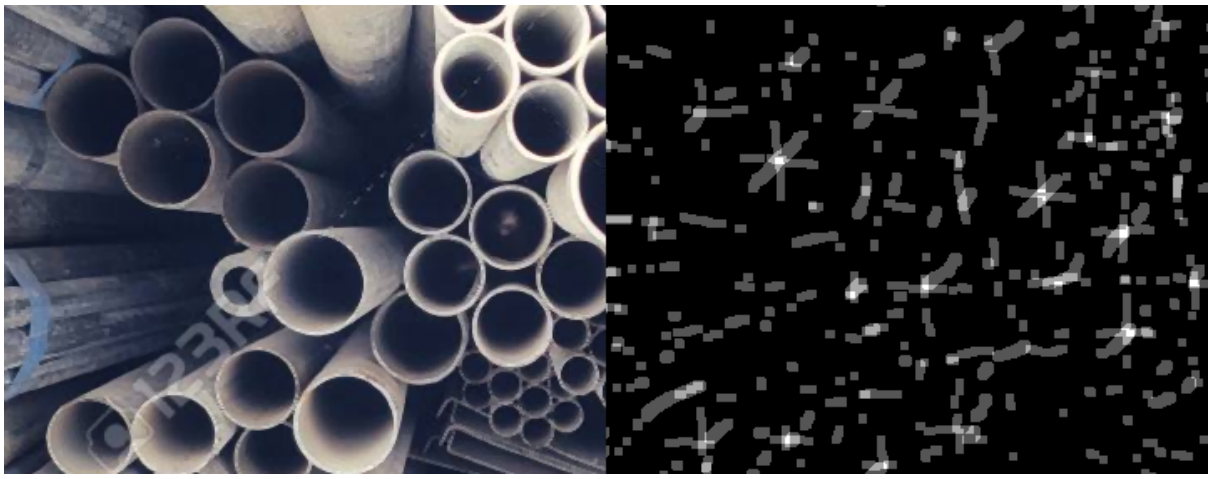


Abbildung 6: Streifen entlang x-Achse, y-Achse, und diagonal; und deren Überschneidungen.

2.2.6 Deep Neural Networks

Ein weiterer Ansatz, den ich ausprobiert habe, weil es mich interessiert hat, sind Deep Neural Networks. Ich hatte viel Gutes über die Netzwerkarchitektur "You Only Look Once" (YOLO) gehört und wollte sie deshalb mal ausprobieren. Ihre Performance soll großartig sein. Zum Trainieren des Netzwerkes braucht man allerdings leider, wie so oft bei Neuronalen Netzwerken, eine Nvidia-Grafikkarte. Mein Arbeitslaptop von Zeiss hatte zwar keine, aber es gab ein paar gemeinsam genutzte Computer für solche Aufgaben und die Nvidia-GPU war aktuell und sehr gut. Tensorflow muss zwar leider genau für eine CUDA-Version kompiliert sein, aber mit der Hilfe von jemanden, der auch schon mit CUDA gearbeitet hat, habe ich es zum Laufen bekommen und konnte das Netzwerk trainieren. Lernbasis war ein auf dem Datensatz Coco trainiertes Modell. Beim Lernen hat YOLO zwar gelernt, die ursprüngliche Klasse nicht mehr zu klassifizieren, aber meine Lerndaten waren vermutlich zu wenig, um ein so kompliziertes Netz zu trainieren: mehr hat es nicht gelernt. Als Trainingsdaten hatte ich 3000 Rohre auf ca. 30 Bildern markiert. 50% fürs Training, 50% zur Validierung. Zwar wäre es möglich gewesen noch mehr Rohre zu markieren, aber vermutlich wären viel zu viele nötig gewesen.

Davor hatte ich, um meine Tensorflow-Installation zu testen, noch ein recht einfaches Netzwerk mit drei Hidden Layers erstellt, allerdings hat auch das nicht erfolgreich genug gelernt. Auf Overfitting habe ich dabei durchaus geachtet: der Trainingsfehler war ungefähr so groß wie der Validierungsfehler. Vermutlich waren auch hier die Trainingsdaten nicht gut genug: meine Testbilder sind alle irgendwo abgeschnitten, sodass man eine Entscheidung treffen muss, ob das Rohr richtig oder falsch erkannt wird. Vermutlich hätte ich den Bereich komplett aus der Bewertung im Training herausnehmen müssen, oder eine dritte Klasse einführen müssen (Rohr, kein Rohr, am Rand).

2.2.7 Fast Radial Symmetry

Die beste Methode habe ich schließlich darin gefunden, Fast Radial Symmetry [1] zu modifizieren: nicht wie im Paper werden die geblurrten Schichten (jeweils für einen Radius) addiert, sondern

direkt auf ihnen wird nach Maxima, also Kreisen gesucht. Dadurch bekommt man zwar deutlich mehr Kreise, aber man bekommt zusätzlich die Information des Radius, die man sonst nicht bekommt.

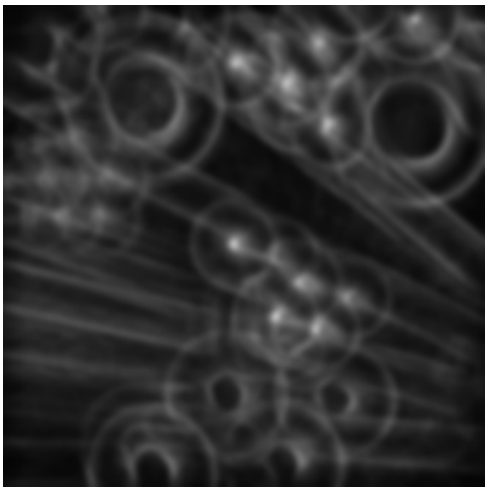


Abbildung 7: Transformation wie Hough-Transformation, aber gewichtet nach Gradient

Abbildung 8: Fast Radial Symmetry, Abtastradius ist 33

Wie man in Abb. 7 und Abb. 8 sieht, ist die Berechnung der Transformation nicht nur deutlich schneller, sondern auch das Ergebnis deutlich sauberer. Bei der Methode

Hat man einen Gradienten und einen Radius gegeben, so gibt es zwei mögliche Kreiszentren: ein Zentrum eines hellen Kreises und einen eines dunklen. Stahlrohre, speziell glatte, haben jedoch oft die Eigenschaft, dass durch die runde Kante diese durch Reflektion von Lichtquellen heller als die Umgebung ist. Zusätzlich sind Stahlrohre im inneren dunkler als außerhalb. Es gibt also zwei Kreise, mit ca. gleichem Mittelpunkt, aber entgegengesetzten Vorzeichen. Im FRS-Paper werden drei Konfigurationen vorgestellt: nur helle Kreise, nur dunkle Kreise, oder beide, wobei sich helle und dunkle auslöschen. Ich separiere, weil ich keine Auslöschung möchte, mich aber für helle und dunkle Kreise interessiere, die Layer, und bekomme aus der Differenz dieser beiden Kreise damit, zusammen mit Tiefendaten, sogar eine Schätzung für die Rohrwandstärke.

Schließlich muss man die Kreise noch nach Güte filtern, und sich die richtigen auswählen. Ich nutze dafür die Kreise, die in ihrem lokalen Gebiet (je nach Radius) ein Maximum darstellen. Durch Schatten führt das nicht immer zur perfekten Korrespondenz zwischen dunklen und hellen

Kreisen. Meist sind die hellen Kreise korrekt, man muss also nur den dunklen auf den bisher minimalen Abstand zum hellen korrigieren.

2.3 Ellipsenoptimierung aus Einzelpunkten

Um aus einer Punktemenge die zugehörige Ellipse zu bestimmen, habe ich erst ein paar eigene, schätzende Algorithmen untersucht, die jedoch nicht korrekt bezüglich ungleichmäßigen Punkteverteilungen waren, oder mit Ellipsendrehungen nicht klarkamen. Da kein Ansatz zufriedenstellend war, habe ich dann den traditionellen Ansatz verwendet: Aufstellen der linearen Form der Ellipsengleichung, Aufstellen eines linearen Gleichungssystemes, und Lösen dieses.

Die lineare, vollständige Ellipsengleichung lautet

$$ax^2 + by^2 + cx + dy + exy = 1.$$

Zur Lösung habe ich die Apache-Math-Bibliothek für Java verwendet. Zur Umwandlung dieser Form in die Form mit Brennpunkten und Achsenlängen, gibt es zum Glück eine Formelsammlung bei Wolfram Alpha, denn die Umrechnung ist nicht ganz trivial.

2.4 Eine erste vollständige Messung

2.4.1 Motivation

Die Methode mit FRS ist zwar zu ca. 3% auf Stereobildern genau, aber braucht auch ca. 12s auf einem 1080p-Bild und Dual-Core (4-Thread), 3-GHz-Prozessor zur Berechnung. Der Aufbau ist deshalb ein Stereobild von vorne, leicht (ca. 10°) von oben und einem symmetrischen von hinten. Anhand der Rohrenden der obersten Rohre und deren aus Stereovision berechneten Tiefe lässt sich die Gesamtlänge der Rohre messen.

2.4.2 Experiment-Aufbau

In meinem Beispiel, das ich in Blender zusammengestellt habe, um Entfernungen und Radien, wie auch Kameraposition und Winkel genau nachschauen zu können, habe ich eine herausfordernde Szene mit Rohren, die bei 3cm Stereo-Kamera-Abstand etwa 6cm Durchmesser haben und 3m lang sind.

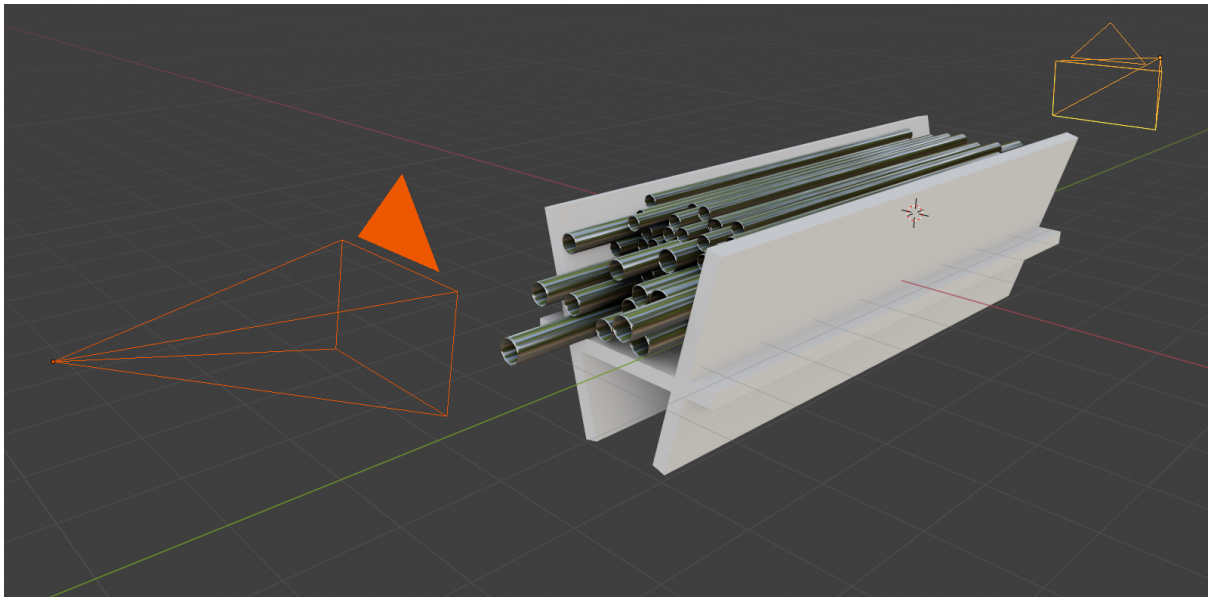


Abbildung 9: Der Aufbau in Blender; die Kameras stellen die Position von vorne und von hinten dar.

2.4.3 Ergebnisse und Schlussfolgerungen

Diese große Länge jedoch hat zu einem entsprechend großen Fehler von ca. 10% in der Gesamtlänge geführt. Ein Mitarbeiter hat deshalb vorgeschlagen, für die Kamerapositionierung ARCore von Google zu verwenden. Von ARCore hatte ich davor schon gehört, allerdings wusste ich nicht, wie genau die Lokalisierung dort ist. Eine Maßstabapp von Google namens "Measure", die auf ARCore basiert, hat im PlayStore eher durchschnittliche Bewertung (3,3 von 5 Sternen), da sie scheinbar gelegentlich 10% falsch misst, was für genaue Messungen nicht zu gebrauchen ist. Der Mitarbeiter meinte die Genauigkeit sei ziemlich gut, vielleicht auf 1% genau. Demnach wäre sie gut genug für meine Anwendung.

In meinen ersten Tests mit dem Hello-World-Beispiel, das ich modifiziert habe, um mir Entfernungen zwischen zwei geklickten Positionen ausgeben zu lassen, und ein paar Tests mit A4-Blättern, bin ich auch auf eine Genauigkeit von ungefähr 1% gekommen.

ARCore scheint also erstmal genau genug zu sein. Zum Aufnehmen von ein paar Positionen und Bildern an diesen Stellen, gab es auch schon eine Zeiss-interne App, was sehr praktisch für ein erstes Testen war. Sie speichert Kameramatrix, FOV, Kamera-Intrinsics, und die Bilder in einem Ordner auf dem Gerät. Ein kleines Problem mit ARCore ist jedoch, dass mindestens Android Version 7 erfordert, auf älteren Geräten also nicht mehr unbedingt läuft. Für zum Beispiel typische Samsung Galaxy SX-Geräte heißt das, dass man mindestens ein S7 benötigt.

Das Stereoproblem ist damit allerdings scheinbar ein bisschen schwieriger geworden: die Kameras sind schauen nicht mehr in die exakt gleiche Richtung. Glücklicherweise lässt sich das Problem jedoch leicht mit den nächsten Punkt zu N Strahlen über einen Least-Squares-Ansatz lösen. Für die QR-Zerlegung verwende ich hier wieder die Apache-Math-Bibliothek. Den Punkt, wo die N Strahlen am nächsten beieinander sind, nenne ich zur einfacheren Vorstellung Schnittpunk, auch wenn es kein wirklicher Schnittpunkt ist.

Unter der Annahme, dass immer der ganze Stahlrohrstapel auf dem Bild ist, lässt sich die Tiefe der Rohre unter Zuhilfenahme einer Verlustfunktion, die auch die Nachbarn mit einbezieht, zuverlässig schätzen. Dabei habe ich die Nachbarschaft als jene Rohre, deren Kante einen Abstand von maximal einem Radius zu dem betrachteten Kreis hat, mit deren relativem Radius und Winkel beschrieben.

Sieht man nur einen Teil des Stapels, steht man vor dem Problem, dass sich die Tiefe in einem sehr regelmäßigen Stapel im Allgemeinen aus zwei Bildern nicht eindeutig bestimmen lässt: Nicht eindeutiger, weil zu regelmäßiger, Aufbau

Dieses Bild könnte von Ort 1 und Ort 2 aufgenommen worden sein, die exakt N Rohrbreiten voneinander entfernt sind. Man könnte zwar annehmen $N = 1$, aber das ist nicht sicher so. Besonders bei vielen, vielen kleinen Rohren wäre das der Fall. Dieses Beispiel ist zwar extrem, es zeigt aber das generelle Problem, dass viele gleiche Rohre, regelmäßig gestapelt, ohne dass man den Rand sieht, effektiv identische Nachbarschaften haben und damit das Mapping sich als sehr fehleranfällig gestaltet.

Meine erste Testszene für das Problem war wieder in Blender erstellt. Mit einem Pythonscript habe ich die exakten Kamerapositionen, wie sie mir auch ARCore geben würde, automatisch als JSON exportiert. Für die Kamerafahrt habe ich einfach Kameraposition und -Drehung animiert. Die Kamerafahrt war erstmal nur von Vorne und umfasst 12 Bilder.

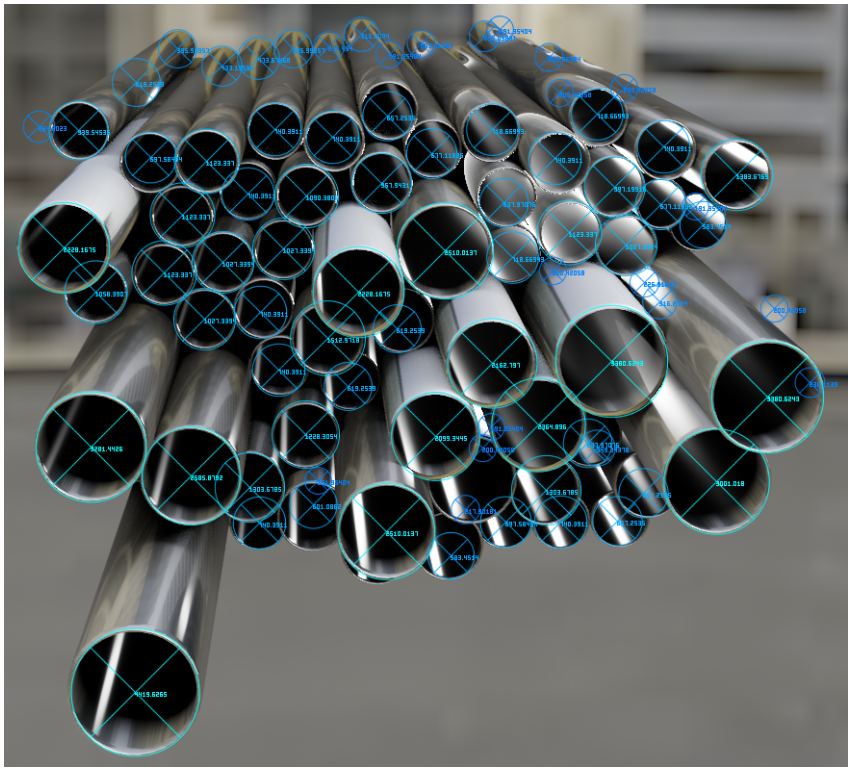


Abbildung 10: Gefundene Kreise in der Blenderszene. Je grüner, desto sicherer ist der Algorithmus, dass es ein Rohr ist.

Zuerst lasse ich die Kreise im Bild finden (Abb. 10), dann werden mit den extrinsischen Kameradaten gute Paare von Bildern gesucht: ähnliche Blickrichtung, nicht zu nah aneinander, nicht zu weit entfernt; je nach durchschnittlichem Abstand zwischen aufeinanderfolgenden Bildern. In den Bilderpaaren werden Rohre anhand ihrer Nachbarschaft zueinander gemappt. Ihre Position in der Welt wird dann per Kameraposition, -Drehung, Fokuslänge der Kamera und Hauptpunkt (principal point) und Pixelpositionen berechnet. Schließlich werden Gruppen von Kreisen gebildet, wo das Mapping übereinstimmt. Diese Gruppen werden sortiert und auf die 2D-x-y-Ebene aufgetragen; Bei größeren Überlappungen wird die bessere Gruppe stehen gelassen.

Die Qualität des Ergebnisses lässt sich ganz gut an den Lücken und Überlappungen der Rohre in der 2D-Ebene erkennen (Abb. 11). Je besser die Position stimmt, umso besser liegen die Kreise von vorne und hinten übereinander.

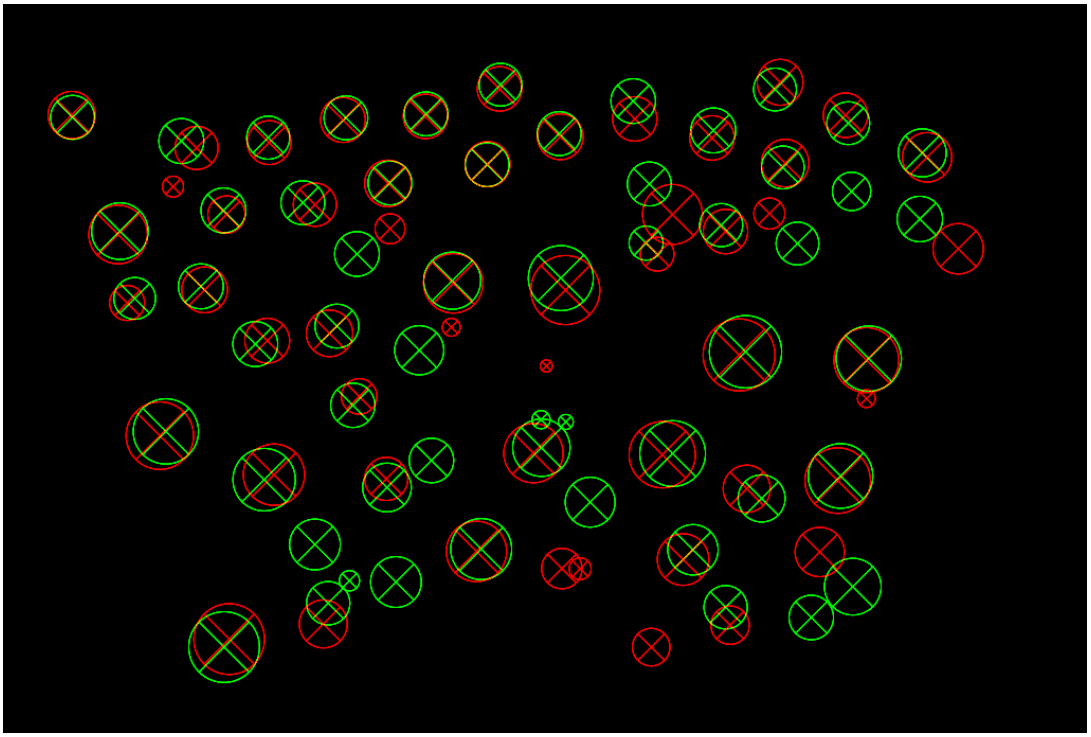


Abbildung 11: Kreise auf die Ebene abgebildet; berechnet aus den Bild- und Kameradaten; Grün und Rot entsprechen von vorne und von hinten.

Das Ergebnis hat sonst außerhalb dieses Bereiches noch ein paar false-positives enthalten, diese lassen sich aber gut daran erkennen, dass sie außerhalb liegen und große Abweichungen der Strahlen vom Schnittpunkt aufweisen. Die Messung des Radius lässt sich verbessern, wenn man mit einbezieht, dass die meisten falschen Kreise wenn dann kleiner sind: durch Verdeckung. Wenn man diese Kreise aus der Rechnung entfernt, bekommt man in ca. 90% der Fälle für die Gruppen bessere Ergebnisse, d.h. in dem Fall kleinere durchschnittliche Abweichung der Strahlen vom Strahlenschnittpunkt.

2.5 Testaufbau

Mein Betreuer, Herr Lakashmanan, hat einen kleinen Aufbau aus 10 Rohren und Stäben zusammengestellt. Davor habe ich mit Bildern aus dem Internet und ein paar Testszenen in Blender, gerendert mit der Renderengine EEVEE, meine Algorithmen ausprobiert.

Zwar war mein Aufbau in Blender oft schon herausfordernd, z.B. durch lange Rohre, die aus dem Haufen hervorstechen, doch der Aufbau meines Betreuers war noch um einiges schwieriger: das Material war nicht auf Stahl beschränkt und neben Rohren gab es auch Stäbe. Ganz abgesehen davon, dass einer der Stahlstäbe so lang war, dass er nicht auf den Tisch gepasst hat, was auch das Abscannen von seiner Rückseite schwierig macht. Die Stahlstärke der Stahlrohre war außerdem deutlich größer, als in meinen bisherigen Tests.

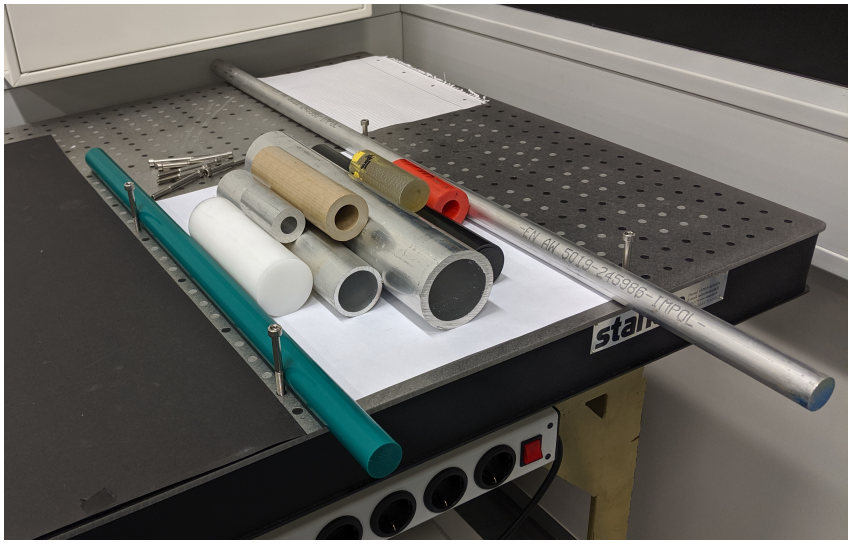


Abbildung 12: Der Aufbau

Die Materialien waren besonders divers: ein schwarzes und ein weißes Rohr, bei denen wegen den extremen Farben Information verloren geht (wegen begrenztem Dynamikumfang des Kamerasensors). Das weiße war zudem etwas transluzent, sodass man seine Kante weniger scharf erkennt. Das Pixel 2 hatte auch Schwierigkeiten, es direkt zu fokusieren. Es gab noch einen weiteren, transluzenteren Stab, doch dessen Kante war glücklicherweise ganz gut zu erkennen.

Des Weiteren gab es ein Stahlrohr, dessen Hauptkontrast nicht entlang des Kreisrandes, sondern wegen des groben Schnittes, in einer Richtung verlief.

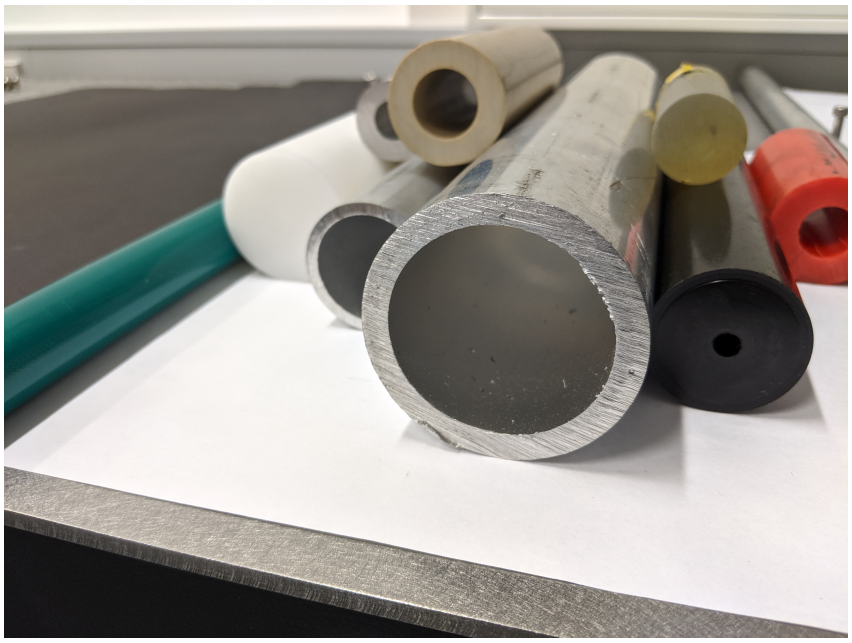


Abbildung 13: Das Rohr mit dem seitlichen Kontrast

In dem Moment war ich froh, nicht auf ein neuronales Netz gesetzt zu haben, denn das hätte die neuen Materialien vermutlich nicht als Stahlrohr erkannt.

Außerdem stellte der Aufbau mich vor ein paar weitere Herausforderungen, die nicht von den Rohren selbst, sondern aus der Umgebung stammten: der Tisch, auf dem der Aufbau lag, ein optischer Tisch, hat als Oberfläche eine Stahlplatte mit vielen, regelmäßig angeordneten,

Kreisen. Mit teilweise dunklem in der Mitte, sehen sie Stahlrohren ziemlich ähnlich, die gehen bloß in eine andere Richtung.

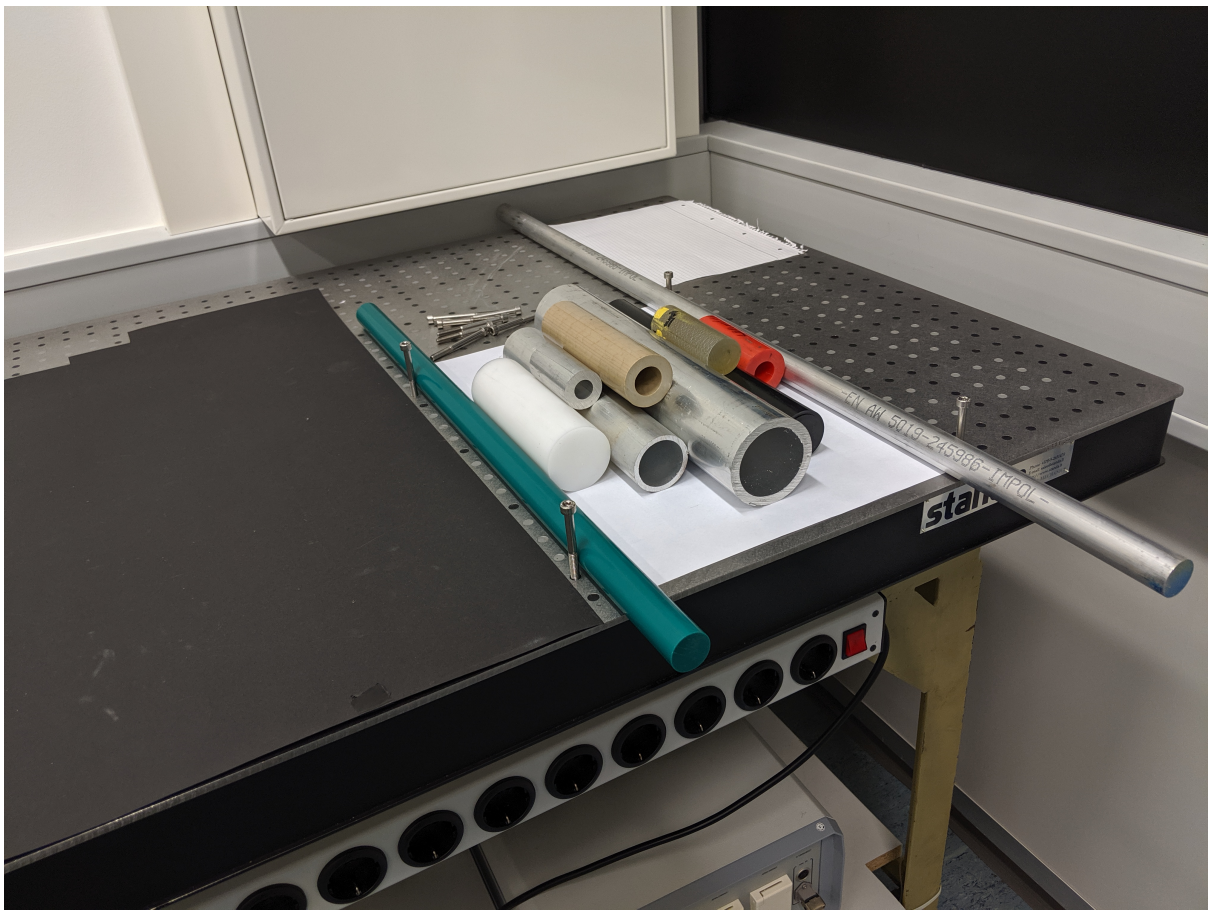


Abbildung 14: Gesamter Aufbau, mit Schrauben und Steckdosenleiste, Blätter als kontrastlose Unterlagen

Außerdem gab es eine Steckdose mit schwarzen Steckplätzen: ohne genügend Licht sehen sie dem schwarzen Rohr bis auf den hinteren Teil und den einen metallischen Kontakt zum Verwechseln ähnlich. Eine kleine weitere Störung, die sich aber leicht beseitigen ließ, war ein Haufen Schrauben, der hinter dem Aufbau lag: metallisches Material und viel Kontrast.

Sind die Rohre immer farbig, ließe sich die Farbe als weiteres Segmentierungsmerkmal wählen, aber die eigentliche Aufgabe sind ja Stahlrohre, sodass es sich für den Aufbau zwar nutzen ließe, im eigentlichen Anwendungsfall aber nicht.

2.6 Messungen mit dem Intel RealSense D435

Abbildung 15: Bild des D435; Von links nach rechts: rechte Stereo-IR-Kamerahälfte, IR-Projektor, linke IR-Stereohälfte, Farbkamera

Der D435 ist ein kommerzieller Tiefensensor, der mit Infrarot-Dot-Projector (Abb. 16) und Stereokameras arbeitet. Die Auflösung von dem einen Farbbild, das mit dem Intel RealSense Viewer anschaubar ist, beträgt, wie auch das Tiefenbild, 1280px mal 720px. Die Android-Integration ist leider nicht so einfach, weil man dafür sein Smartphone rooten muss. Das hätte Intel besser lösen können. Immerhin ist der Anschluss USB-C, das sich in modernen Geräten immer weiter verbreitet.

Die Reichweite für gute Tiefenergebnisse bei Stereokameras hängt vom Abstand der Kameras ab. Die Kameras beim D435 sind 5cm entfernt, was für den Testaufbau ziemlich viel, und meist mehr ist als bei üblichen Smartphones, aber für größere Stapel sicher von Vorteil wäre. Die Qualität der Tiefenbildes ist für die Erkennung von Stahlrohren im Bild aber leider problematisch:

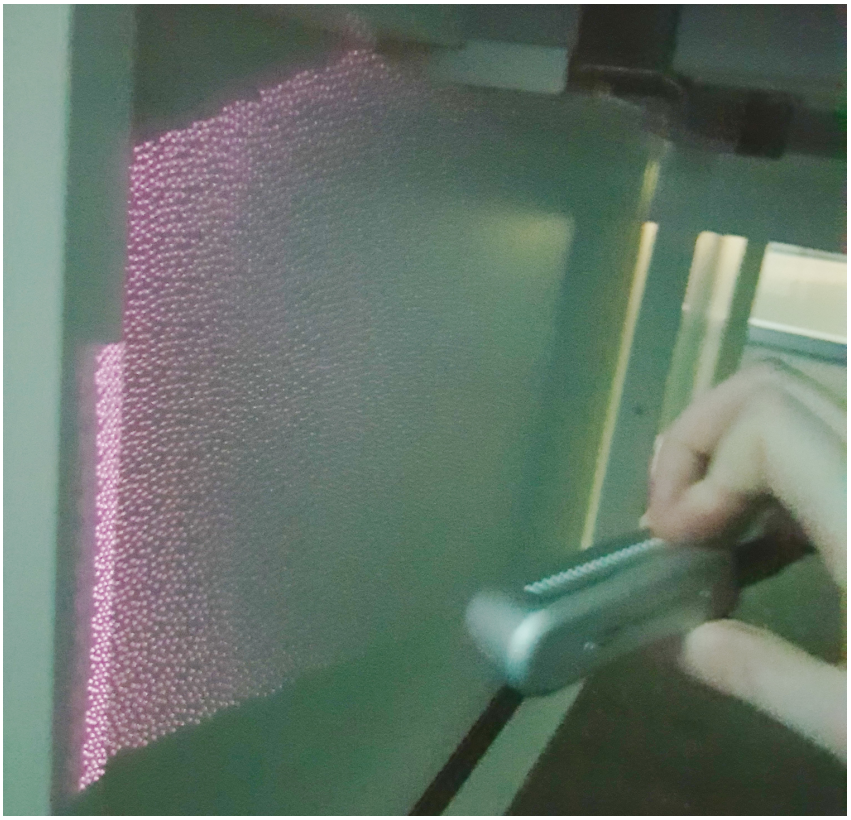


Abbildung 16: IR-Punkte des D435, aufgenommen in Dunkelheit mit dem S8+, nachträglich aufgehellt

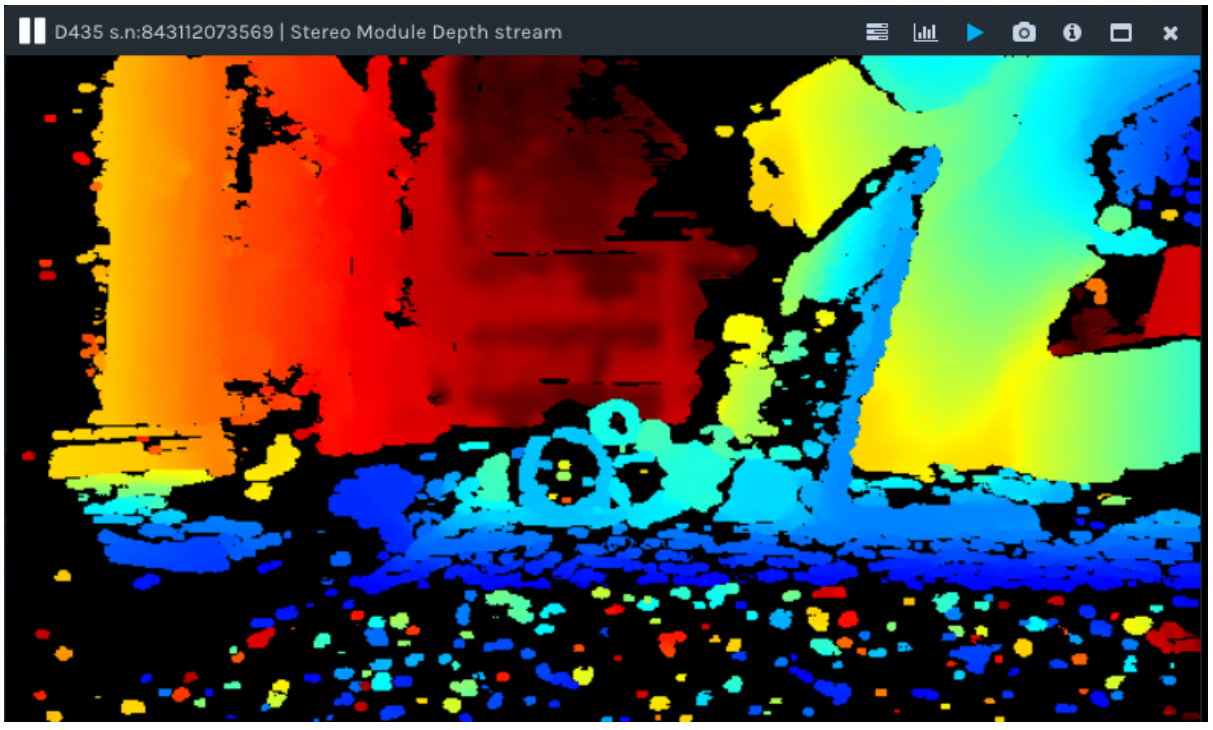


Abbildung 17: Bunte Flecken sind falsch erkannte Tiefen. Schwarz heißt, dass keine Tiefe zugeordnet werden konnte.

Nahe Objekte sind immer problematisch, da der Parallaxeneffekt dort zu groß wird, um ihn

schnell und zuverlässig berechnen zu können. Beim D435 zeigt sich das in Bereichen mit bunten Tiefenflecken, umgeben von schwarz, was heißt, dass er dort keine Werte gefunden hat. Doch auch bei den Stahlrohren hat er Probleme mit den Reflektionen: die Ringe konnten einigermaßen gefunden werden, die Tiefe vom Körper hingegen ist unbekannt.

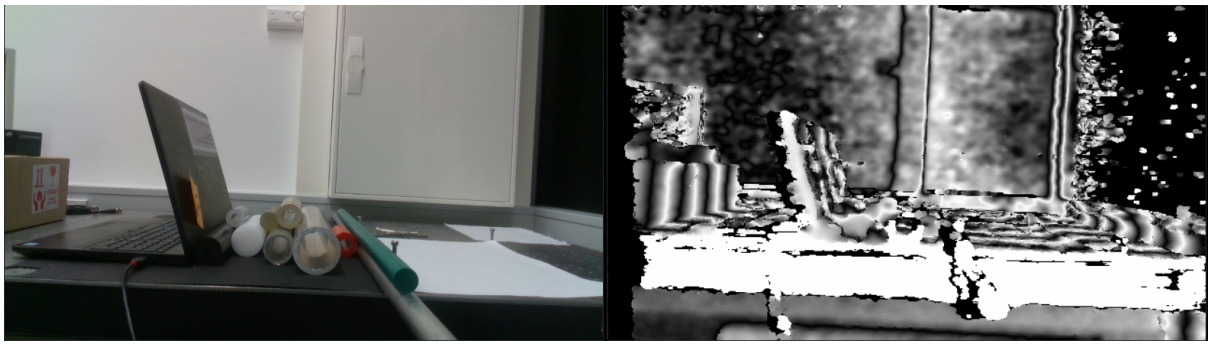


Abbildung 18: Die Streifenvisualisierung ist gut zur Fehlerabschätzung: auf Ebenen sollten die Streifen möglichst gerade sein.

Das Tiefenergebnis ist an sich vermutlich genau genug, und nur etwas verrauscht, wie man an der Visualisierung mit den Streifen (Abb. 18) sehen kann: wäre die Messung perfekt, so wären die Linien auf dem Tisch und an der Wand gerade. Leider fehlt ja aber einige Information. Findet man die Kreise in den Bildern aus ARCore zuverlässig, ist das Ergebnis von ARCore vermutlich genauer; denn man darf nicht vergessen, dass man aus den Bildern vom D435 genauso die Kreise finden müsste.

2.7 Messungen mit dem Pico Flexx

Abbildung 19: Bild des Pico Flexx

Der Pico Flexx von PMD Technologies AG ist ein Time-Of-Flight-Sensor (TOF), d.h. er sendet mit einem Laser IR-Pulse aus und misst die Zeit zwischen dem Absenden, bis der Laserstrahl zurückkommt, und errechnet daraus die Tiefe in der Szene. Wegen der überschaubaren Framerate von 5 FPS sieht man mit einer IR-sensitiven Kamera die Pulse. Der Anschluss an den PC läuft über ein Adapterkabel von Mikro-USB mit 3.0-Erweiterung zu normalem USB. Die Auflösung ist auf 352px x 287px begrenzt. Zum Anschauen wird der Royale Viewer empfohlen, auf

den man mit dem Kauf des Pico Flexx Zugriff erhält. Der Bereich der Messung wird auf 10cm bis 6m angegeben, d.h. der Sensor ist nicht für besonders kleine oder große Stapel geeignet. Die Technologie hat mit spiegelnden Oberflächen entsprechend Probleme. Die Rohre im Testaufbau spiegeln nur etwas, aber sie spiegeln. Komplett schwarze Körper können genauso ein Problem sein, da sie das Licht verschlucken, statt es zurück zu senden. In meinem Test mit meinem Testaufbau hat sich außerdem gezeigt, dass der TOF-Sensor auch Probleme mit sehr hellen Gebieten hat. Interessanterweise hat der TOF-Sensor aus Sicht von vorne eher Probleme mit den Rohr- und Stabenden, als mit den Korpussen (Vgl. D435 in Abb. 2.6). Von oben hat er, vermutlich wegen den Reflektionen der Lampen, auch mit denen große Probleme.

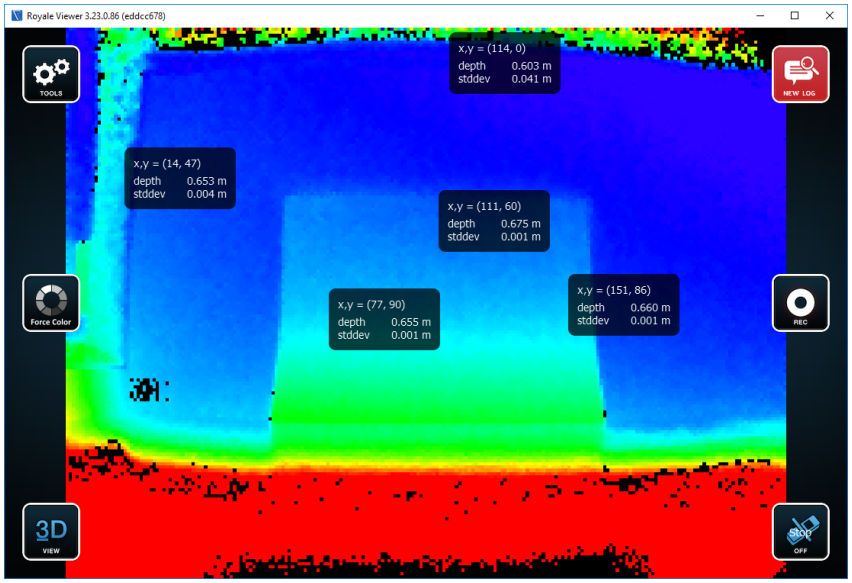


Abbildung 20: Messung und Abweichung einer statischen Szene

Abbildung 21: Die Szene der Messung

In Abbildung 20 sieht man den Fehler. Wie schon früher beobachtet, ist er vom Material abhängig. Besonders fehleranfällig war die Aluleiste, bei der ca. die Hälfte der Pixel gefehlt

haben. Gut hingegen wurde das Papier mit seiner sehr rauen Oberfläche erfasst. Die Standardabweichung war zudem für die ganze Szene, außer der Aluleiste, $< 1\%$.

Allerdings sind das Primärziel Stahlrohre, sodass sich dafür der Pico Flexx nicht wirklich eignet.

Im Vergleich zum Pico Flexx liefert der D435 bei dieser Szene ohne temporalen Filter Standardabweichungen von ca. 5-10mm. Hält man den Sensor ruhig, kann man mit temporalem Filter aber durchaus auch Standardabweichungen von nur 1-2mm erreichen.

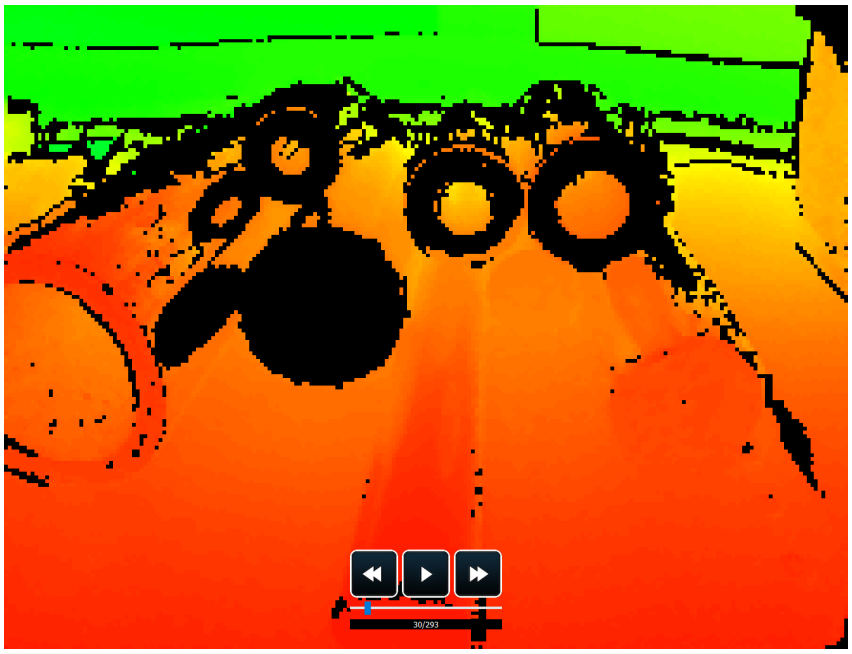


Abbildung 22: Bild des Pico Flexx: besonders die fehlenden Vorderseiten der Rohre sind problematisch. Schwarz heißt wieder, dass keine Tiefe zugeordnet werden konnte.

Abbildung 23: Auch die Oberseite lässt sich nicht optimal ausmessen. Für die Seiten zeigt sich ein äquivalentes Bild.

Außerdem konnte ich beobachten, dass der Sensor offenbar Probleme mit meiner Haut hatte, teilweise der Strickjacke. Vielleicht liegt es an der Transluzenz, denn meine Haut ist zwar sehr hell und der Sensor hatte ja auch mit sehr hellen Objekten Probleme. Meine Stickjacke jedoch war blau, und nicht sonderlich hell, also vermute ich eher, dass die Transluzenz das Problem war.

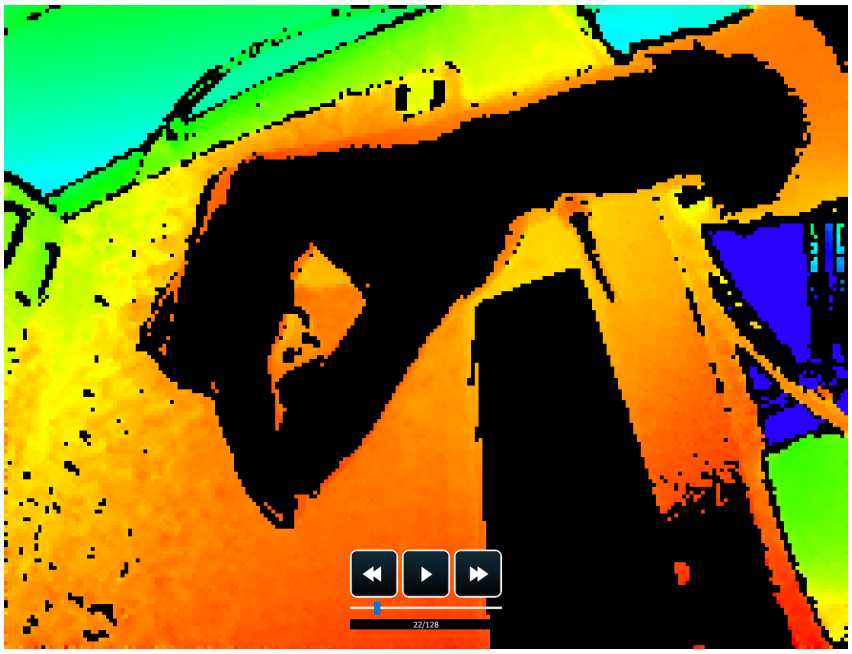


Abbildung 24: Probleme mit meiner transluzenten Hand.

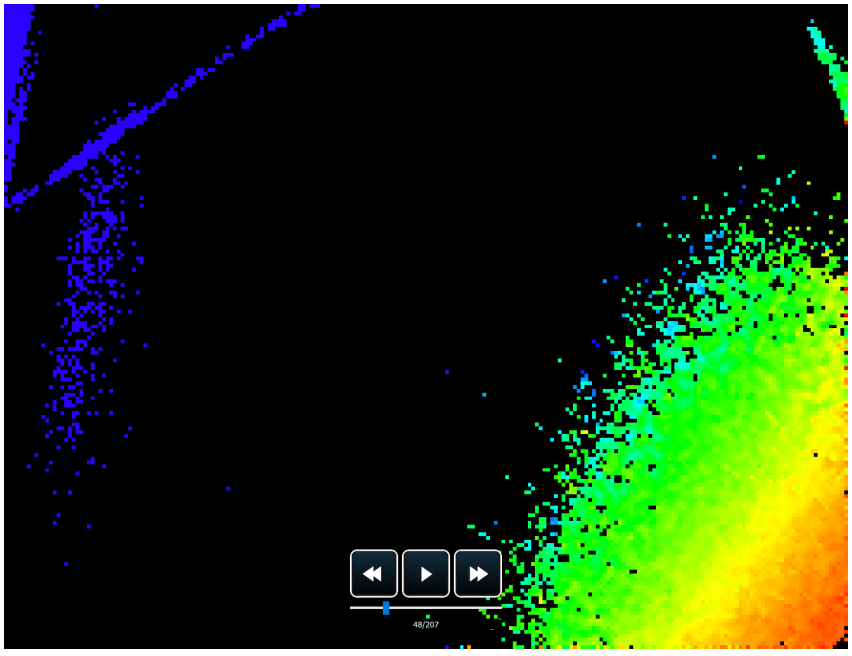


Abbildung 25: Die schwarze Wand im Laserlabor erfüllt ihre Aufgabe des Absorbierens der Strahlung: sie ist für den Pico Flexx unsichtbar/unverständlich.

2.8 Textur-Segmentierung

Die Testszene hat ganz im Gegensatz zu meinem Test in Blender viele verschiedene Materialien, also habe ich mich etwas über Textursegmentierung informiert, um damit die Kanten der Textur loszuwerden. Das Forschungsfeld ist zwar schon ziemlich alt, aber es gab auch ein Paper zu einem neuronalen Netz aus 2018 dazu, weil es immer noch eine wichtige Aufgabe, z.B. zum Auszählen von Feldern auf Satellitenaufnahmen, ist. Senthil hat mir dann aber den ausgezeichneten Tipp gegeben, es mal mit Median-Blur zu probieren. Ich kannte ihn so noch nicht und war mir seiner Eigenschaften nicht so bewusst; und er ist super für die Aufgabe: grobe Kanten werden erhalten, während kleine, die der Textur, geglättet werden. Gaussian-Blur hätte das zwar auch in etwa gekonnt, doch dort verliert man die Schärfe der Kante und kann damit nicht mehr so genau die Rohre ausmessen.

Der Algorithmus, der in OpenCV dafür implementiert ist, ist "Median Filtering in Constant Time" und damit einer der schnellsten Algorithmen, den es für dieses Problem gibt. Der Algorithmus setzt auf Histogramme und möglichst gutes Caching, da man sonst für große Radien für jeden einzelnen Pixel hunderte Pixel im Originalbild absuchen und deren Median für alle drei Farbkanäle bestimmen muss. Bevor ich auf das Paper gestoßen bin, habe ich die primitive Variante selbst implementiert, die dadurch ca. 100x langsamer war, als das, was ich in Gimp, wo ich den Median-Blur zuerst ausprobiert habe, gesehen habe.

2.9 Finden der Rohrstärke

Um die Stärke der Rohre universell, also nicht nur für Stahl-, sondern auch für Plastikrohre, zu finden, habe ich ein Histogramm der Gradienten, abhängig von notwendiger Skalierung der Ellipse, damit sie diesen Punkt schneidet (Radius-Äquivalent) aufgestellt, mit Gaussian-Blur das Rauschen etwas unterdrückt, und dann nach Maxima gesucht. Die Ringe wurden dabei nach Wert relativ zum Maximum bewertet. Gibt es zu viele gute, so ist es wohl kein richtiges Rohr; sondern ein falsch detektierter Kreis, oder Baumstamm. Meine Aufgabe sind ja aber Rohre, weshalb ich diese Ellipsen aussortiert habe. Dann habe ich die stärkste, möglichst weit außen liegende Ellipsenlinie als Radius des Kreises und wenn es ein zweites, ähnlich starkes Maximum gab, dieses als inneren Radius eingeordnet. Oft funktioniert das ziemlich zuverlässig und lässt einen somit auch die Wandstärke schätzen. Gibt es keinen zweiten Ring, so muss es wohl ein Stab sein.

2.10 Rohr-Clustering

Hat man die Ellipsen in den einzelnen Bildern zuverlässig gefunden, steht man wie auch schon bei der Stereoansicht vor der Aufgabe, jedem Kreis in einen Bild den passenden Partner in den anderen Bildern zuzuordnen.

Aus Interesse, wie gut es funktioniert, habe ich versucht mit k-Means-Clustering und einer Distanzfunktion die Strahlen (von Kameraposition zu Kreismitte) zu clustern. Die Distanzfunktion hat dabei den Abstand der Strahlen, den Radius (wäre der Schnittpunkt die Kreismitte in 3D), die durchschnittliche Farbe, und den relativen inneren Radius bewertet.

Das Ergebnis war leider nicht so gut, sodass ich auf eine übliche Distanzfunktion umgestiegen bin, die wieder Farbe, Radius, Kreisfinder-Score, usw. in Betracht zieht. Kleine Ungenauigkeiten fälschen jedoch gute Überschneidungen vor, die eigentlich nicht so gut sind. Deshalb bin ich darauf umgestiegen, nicht Paare, sondern Tripel aus drei Bildern miteinander zu vergleichen. Aus der Testszene meines Betreuers ließen sich so aus der einen Richtung 9/10 Rohre, und aus der anderen 6/10 finden. Die beiden besonders langen Stäbe waren dabei mit am problematischsten, da die kreisförmige Seiten des längsten nur von vorne sichtbar waren und vom zweit-längsten zwar aus beiden Richtungen, aber von hinten nur in der Hälfte der Bilder.

Insgesamt ergaben sich 6/10 richtig gefundene Rohre und nicht mal so viele false-positives,

weil diese in den ganzen Filterungsschritten zuvor heraussortiert wurden. Der Radius vom problematischen, schwarzen Rohr, weicht nur 5% vom echten Wert ab.

Die Zuordnung hatte ich erst recht pessimistisch mit Paaren und 1:1 Korrespondenz der besten Partner probiert, doch werden dann viele Rohre vergessen. Deshalb habe ich mich entschieden, iterativ die besten Paare der Szene hinzuzufügen und Überlappungen zu verbieten. Das führt zu ein paar False-Positives, aber man bekommt die Messungen, die man erwarten würde.

Wählt man schließlich noch Bilder, auf denen möglichst alle Rohre zu sehen sind in der App aus, so schafft man es auch, dass die App alle Rohre findet:

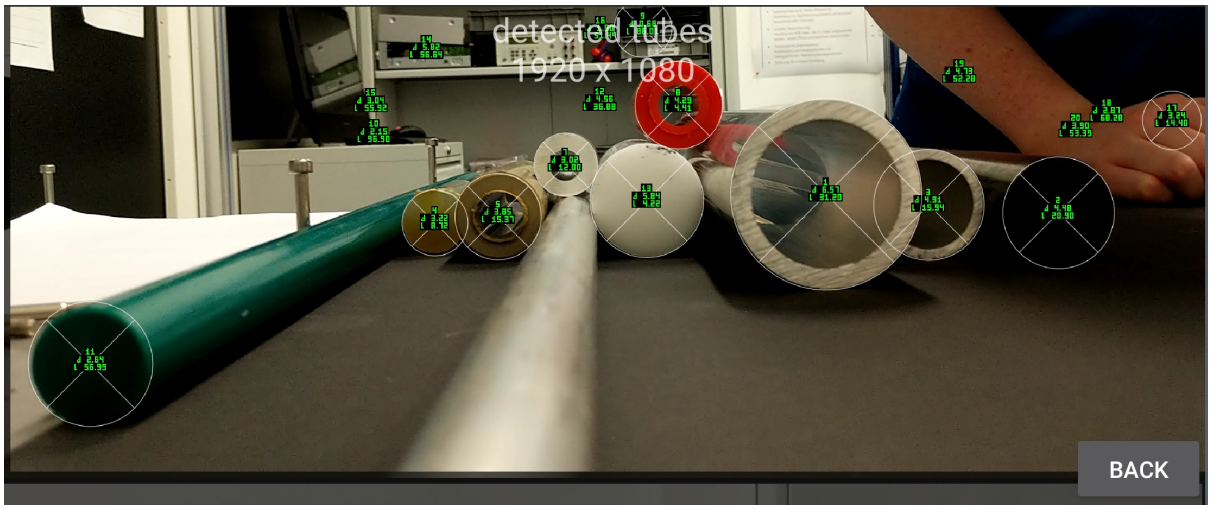
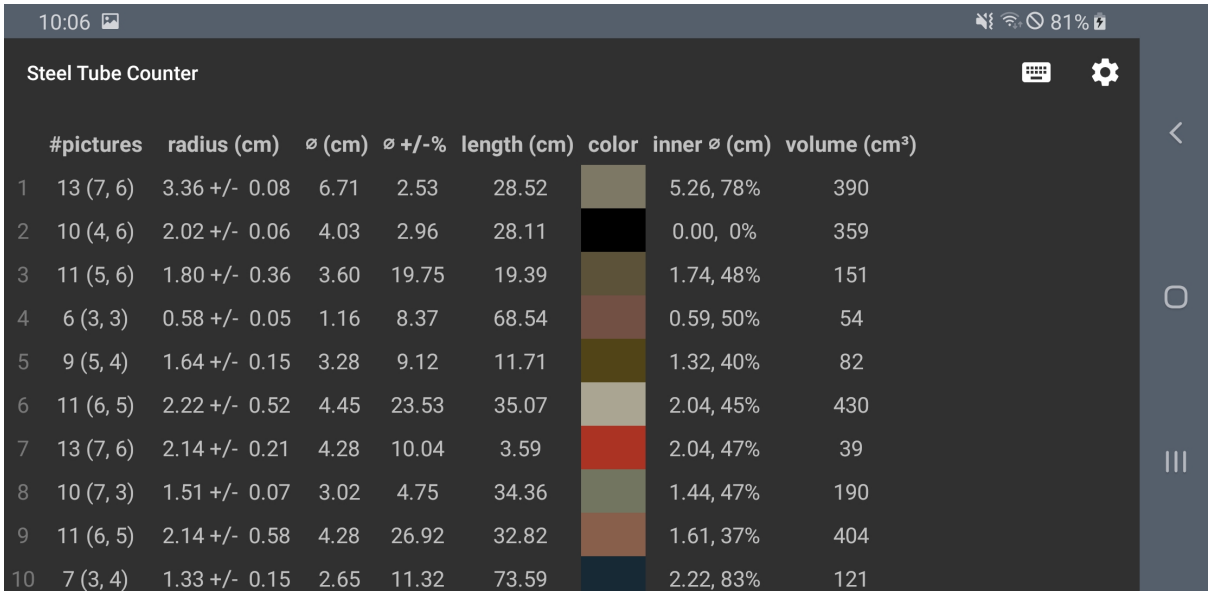


Abbildung 26: Alle Rohre gefunden; jeweils von oben nach unten: Rang, dann Durchmesser in cm, und Länge in cm.

Die Ergebnisse werden zusätzlich in einer Tabelle in der App dargestellt:



	#pictures	radius (cm)	⌀ (cm)	⌀ +/-%	length (cm)	color	inner ⌀ (cm)	volume (cm³)
1	13 (7, 6)	3.36 +/- 0.08	6.71	2.53	28.52		5.26, 78%	390
2	10 (4, 6)	2.02 +/- 0.06	4.03	2.96	28.11		0.00, 0%	359
3	11 (5, 6)	1.80 +/- 0.36	3.60	19.75	19.39		1.74, 48%	151
4	6 (3, 3)	0.58 +/- 0.05	1.16	8.37	68.54		0.59, 50%	54
5	9 (5, 4)	1.64 +/- 0.15	3.28	9.12	11.71		1.32, 40%	82
6	11 (6, 5)	2.22 +/- 0.52	4.45	23.53	35.07		2.04, 45%	430
7	13 (7, 6)	2.14 +/- 0.21	4.28	10.04	3.59		2.04, 47%	39
8	10 (7, 3)	1.51 +/- 0.07	3.02	4.75	34.36		1.44, 47%	190
9	11 (6, 5)	2.14 +/- 0.58	4.28	26.92	32.82		1.61, 37%	404
10	7 (3, 4)	1.33 +/- 0.15	2.65	11.32	73.59		2.22, 83%	121

Abbildung 27: Beispiel der Tabelle, gehört nicht direkt zu Abb. 26, ist aber die gleiche Szene; Berechnet aus vier Bildern von vorne und vier von hinten.

Die Rohr-Gesamtlänge ist besonders anfällig für Fehler, wenn das betrachtete Rohr relativ zu den anderen Rohren besonders kurz ist: ein $\pm 3cm$ Fehler in der Tiefe für ein $10cm$ langes Rohr ergibt $4cm$ bis $16cm$ Länge. Das zeigt sich am Beispiel 27 z.B. beim roten Rohr: es ist nur $9cm$

lang, während das längste der Szene einen ganzen Meter lang ist. Der tatsächliche Radius von diesem Rohr ist $4.2cm$, also nicht so weit vom geschätzten Wert entfernt. Der innere Radius ist $2.0cm$.

2.11 Ergebnisse im Testaufbau

Die finalen Ergebnisse hängen von der Qualität des Trackings und von der Aufwahl der Bilder zur Berechnung ab. Im Nachfolgenden ist eine Tabelle, die die realen Werte mit denen der App vergleicht. Die Rohre mit Längen bis 15 cm habe ich dafür noch einmal extra aufgenommen und ausrechnen lassen, da ihre Längenfehler sonst deutlich größer sind.

Farbe, Material	Durchmesser, GT (cm)	Durchmesser, App (cm)	Fehler
Stahl	6.60	6.27	5.3%
Stahl	3.20	3.10	3.2%
Stahl	4.50	4.59	2.0%
Weiß, transluzent	5.10	5.81	13.9%
Schwarz	4.10	3.83	7.0%
Dunkel-Türkis	2.03	2.62	29%
Rot	4.20	4.31	2.6%
Holzimitat	4.10	3.75	9.3%
Gelb-grün, transluzent	3.10	3.10	<1%
Stahl, blauer Fleck	2.5	2.61	4.4%

Farbe, Material	Länge, GT (cm)	Länge, App (cm)	Fehler
Stahl	33.1	35.4	6.9%
Stahl	6.42	5.16	24.4%
Stahl	25.0	24.2	3.3%
Weiß, transluzent	15.1	12.5	20.8%
Schwarz	29.1	36.5	25%
Dunkel-Türkis	ca. 65	62	5%
Rot	9.00 bis 9.15	8.22	10.3%
Holzimitat	14.1 bis 14.3	14.0	1.6%
Gelb-grün, transluzent	9.05	7.53	20%
Stahl, blauer Fleck	ca. 100	Rückseite verdeckt	-

Das rote Rohr und das Holzimitat-Rohr wurden nicht sauber abgeschnitten, sodass ihre Länge nicht ganz eindeutig ist. Das längste Rohr war so lang, dass man es auf dem Boden hätte vermessen müssen. Es war länger als der Tisch tief.

Zuletzt noch der innere Radius, der durch die Maxima im Histogramm der Gradienten, abhängig von der Skalierung der Ellipse bestimmt wird. Er ist für die Abschätzung der Masse der Rohre interessant. 0 wird vom Algorithmus verwendet, wenn er keinen zweiten Ring findet, der stark genug ist.

Farbe, Material	Relativer innerer Radius, GT (cm)	R.i.R., App (cm)	Fehler
Stahl	0.79	0.78	1%
Stahl	0.50	0.47	6%
Stahl	0.81	0.80	1%
Weiß, transluzent	-	0.83	falsch
Schwarz	-	0.00	richtig
Dunkel-Türkis	-	0.67	falsch
Rot	0.48	0.47	2%
Holzimitat	0.55	0.55	$< 2\%$
Gelb-grün, transluzent	-	0.00	richtig
Stahl, blauer Fleck	-	0.11	falsch

Der innere Radius vom weißen, transluzenten Rohr wurde falsch eingeschätzt, weil oft die obere Kante nicht richtig erkannt wurde. Auch für das menschliche Auge und Verständnis sieht man diese Kante sehr schlecht im Vergleich zu den Kanten der anderen Rohre. Das dunkeltürkise Rohr und das lange Stahlrohr sind wegen ihrer Länge in den Aufnahmen oft unscharf. Beim Stahlrohr mit dem blauen Fleck war vermutlich der Fleck detektierten, inneren Radius schuld.

2.12 Zukünftige Verbesserungen

Ein Loop-Closure-Algorithmus, d.h. Neuberechnung der Positionen, sobald man alle Bilder kennt, könnte die Positionen besser bestimmen, was zu besseren Tiefenmessungen führen würde. Da man mit der Berechnung im Nachhinein so oder so nicht an Real-Time-Algorithmen gebunden ist, könnte man sich zum Tracken auch mehr Zeit lassen und bessere, teurere Algorithmen einsetzen. Außerdem wäre es für problematische Rohre vielleicht eine ganz gute Idee, nicht die Rohre mit Ringen in den Bildern zu clustern, sondern erst mal die Ringe selbst. Beim weißen, transluzenten Rohr, würde das zwar genauso zum Finden eines inneren Ringes führen, aber der Radius wäre korrekter. Außerdem könnte man, wenn die Rohre dafür lang genug sind, sodass man nicht hindurch schaut, vom Rohrinneren verlangen, dass es dunkler ist, als der Rand vom Rohr.

Im Fast-Radial-Symmetry-Teil meiner App gehe ich 5%-Schritte für den Radius, weil es sich als gut genug herausgestellt hat, wenn ich danach den Ellipsenoptimierer verwende. Im Kreisfinden ist das System so oder so ziemlich gut, Verbesserungen außerhalb vom Kreisfinden und der Kamera-Positionsbestimmung wären nur noch im Clustering der Kreise aus den einzelnen Bildern möglich. Der aktuelle Algorithmus funktioniert meist, ist aber nicht perfekt.

Für reine Ausmessung von Stahlrohren sollte man zudem in den Loss-Funktionen zum Clustern weniger Gewicht auf die Farbe legen. Hat man hingegen nur wenige oder keine Stahlrohre, könnte man den Kreisfinde-Abschnitt beschleunigen, indem man "helle" und "dunkle" Kreise gemeinsam zählt.

Hat man besonders viele Rohre, mehr als 400 pro Bild, wäre es vermutlich angebracht, den zur Performance-Optimierung verkleinerten Votingraum von FRS zu vergrößern.

3 Schluss

Wie ich gehofft hatte, habe ich viel über Computer Vision gelernt: ich habe viele Algorithmen und Methoden kennengelernt und zudem einen tieferen Einblick in Photogrammetrie und Augmented-Reality bekommen. Besonders Photogrammetrie ist auch für mich privat sehr interessant, da ich gerne kleinere Spiele entwickle: wenn die low-cost Photogrammetrie gut genug ist, kann man beliebige Objekte aus der echten Welt scannen, und in seinem Spiel verwenden.

Berechnungszeit, Aufwand und Qualität vom Ergebnis waren es bisher aber noch nicht wirklich wert.

Außerdem habe ich ARCore näher kennengelernt. Dass es noch keine längerfristigen Ankerpunkte gibt, ist wirklich schade, aber die Depth-API, wenn sie veröffentlicht und nicht verworfen wird, könnte coole Dinge mit AR ermöglichen.

Für die Auswahl von zukünftigen Arbeitsplätzen ist es zudem nützlich zu wissen, dass Großraumbüros, zeitweise sehr nervig sein können, besonders, wenn die Arbeiter an sehr unterschiedlichen Themen arbeiten. Bei einer Diskussion eines einem fremden Themas kann man nicht mal mit-diskutieren, aber Ausblenden ist anstrengend. Als Hilfe habe ich da zwar meine Noise-Cancelling-Kopfhörer, aber Musikhören lenkt ja auch ein klein wenig ab. Ohne Musik ist ihr Effekt deutlich schlechter, da man sich an Lautstärken gewöhnt und auch Reden bei 30dB ablenken würde. In meinem Fall saßen im gleichen Raum neben uns vier Studenten noch acht Optik-Entwickler. Die Grundlagen der Optik kennt man zwar, aber das ist selbstverständlich auf einem komplett anderem Level.

References

[1] Gareth Loy and Alexander Zelinsky. A fast radial symmetry transform for detecting points of interest. volume 2350, pages 358–368, 01 2002.

[2] Jie Ni, Maneesh Singh, and Claus Bahlmann. Fast radial symmetry detection under affine transformations. 06 2012.