Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Gruppenarbeit

Head Tracking durch Bildanalyse auf einem Smartphone

Andreas Marcaletti <marandre@ee.ethz.ch>
Si Sun <sunsi@ee.ethz.ch>

Betreuer:
Samuel Welten
Michael Kuhn
Prof. Dr. R. Wattenhofer

Inhalt

1 Einleitung

Moderne mobile Geräte mit integrierten Orientierungssensoren und Kamera erlauben eine Vielzahl von Anwendungen in den Bereichen Virtual Reality und Augmented Reality. Damit die Illusion einer virtuellen Welt perfekt ist und man wirklich das Gefühl bekommt durch Displays in eine andere Welt zu sehen, muss man die Bewegungen des Kopfes erfassen können.

In dieser Arbeit wollen wir ein System zur Positionsbestimmung mittels des Kamerabildes auf einem Android Gerät implementieren. Damit könnte man die Bewegungen des Kopfes erfassen.

1.1 Problemstellung

Nachdem wir uns in die Materie eingearbeitet hatten und uns die Komplexität des Problems bewusst wurde, war uns klar, dass wir uns im Rahmen dieser Gruppenarbeit einschränken müssen.

Wir haben uns dann auf folgende Einschränkungen geeinigt:

- Wir bewegen uns in einer kontrollierten Testumgebung, welche aus einem Whiteboard besteht, auf welchem wir ein vordefiniertes Muster aus Punkten zeichnen. Die Kamera wird auf das Whiteboard gerichtet und durch Verschiebung der Punkte im Bild wird die Bewegung der Kamera erfasst (Abbildung 1).
- Wir schränken die Bewegung auf Translationen und nur eine Rotation ein .

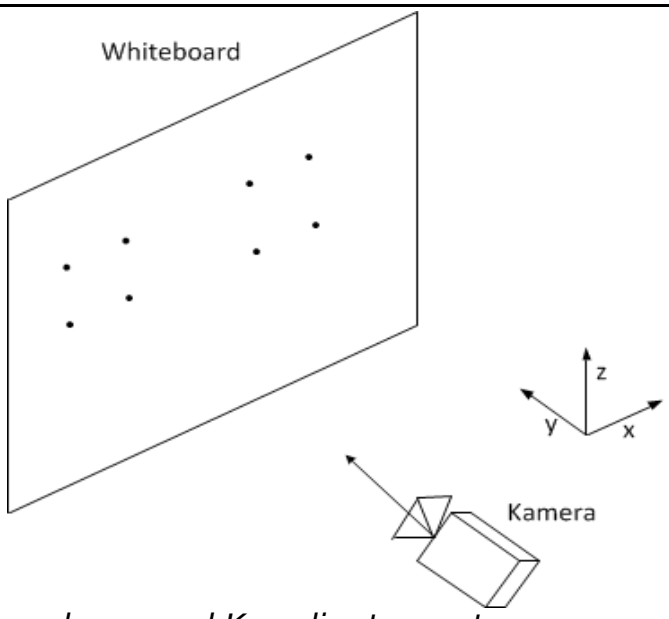

Abbildung 1: Testumgebung und Koordinatensystem.

1.2 Grundsätzlicher Ablauf

Bevor wir einen Algorithmus zur Positionsbestimmung anwenden können müssen wir die gezeichneten Punkte möglichst ressourcensparend erkennen.

Wir haben zwei unterschiedliche Algorithmen zur Positionsbestimmung entwickelt welche mit verschiedene Mustern arbeiten. Beide werden evaluiert und verglichen.

1.3 Notation

Das Bild hat die Dimensionen Breite *w* auf Höhe *h.* Die Koordinaten des Bildes werden *u* und *v* genannt.

Das räumliche Koordinatensystem, auf das wir uns beziehen ist in Abbildung 1 dargestellt.

Im folgenden werden wir Punkte im realen Raum mit *P (x,y,z)* und Bildpunkte mit *P' (u,v)* bezeichnen.

Des weiteren brauchen wir die Brennweite der Kamera, welche wir mit *f* bezeichnen. Wir benützen auch eine Umrechnung dieser Brennweite in Pixel, welche wir f_{px} nennen. Dies entspricht bei unseren Messungen der Breite des Bildes in Pixel.

γ beschreibt den Winkel zwischen der Blickrichtung und einer Senkrechten zur Wand durch die Kamera.

2 Punkterkennung

Der erste Schritt ist ein zuverlässiges Erkennen der Punkte anhand des Kamerabildes, um daraus später mit Algorithmus 1 oder 2 die Position der Kamera zu bestimmen. Die Punkterkennung sollte einfach und schnell sein, da wir eine hohe Framerate gewährleisten wollen. Deshalb geben wir vor, dass die Anzahl der zu erkennenden Punkte konstant ist. Auch benutzen wir nur Helligkeitswerte ohne die Farbe zu betrachten.

Die Android API liefert uns das Kamerabild als ein Bytearray in YCbCr Kodierung. Dabei sind die ersten $w \cdot h$ Einträge die Helligkeitswerte des Bildes. Um die Punkte nun zu detektieren, durchlaufen wir das Bild Pixel für Pixel und speichern für jede Zeile des Bildes die erkannten dunklen Bereiche ab. Am Ende jeder Zeile wird eine Zuordnung durchgeführt, um die erkannten Bereiche mit denen der oberen Zeile zu verknüpfen.

2.1 Erkennen Dunkler Bereiche

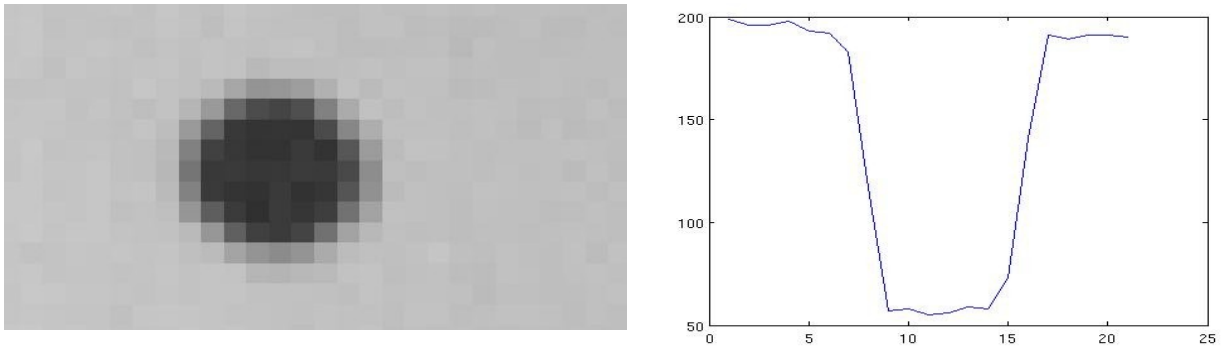

Abbildung 2: Links: Ein typischer Punkt, den wir erkennen möchten. Rechts: Der Plot der Helligkeitswerte einer Zeile vom Bild links. Der dunkle Bereich ist gut erkennbar.

Für das Entdecken eines solchen dunklen Bereichs wie in Abbildung 2 rechts, betrachen wir die Veränderung zwischen zwei Pixel. Dies hat sich als robuster erwiesen als eine fixe Intensitätsgrenze zu wählen, da Schattenverläufe im Bild oder schlechte Lichtverhältnisse das Erkennen weniger stark beeinträchtigen.

Dies sind unsere Bedingungen für die Intensitätswerte I_i beim Pixel i, so dass er als Beginn eines dunklen Bereichs gilt:

$$I_i - I_{i+2} > \delta$$
$$I_i - I_{i+w/32} < \delta/4$$

δ ist ein Parameter, der nach jedem Frame neu angepasst wird. Entdecken wir zuwenig Punkte, verringern wir den Wert, detektieren wir zu viele, wird er erhöht.

Wir verlangen also, dass der übernächte Pixel um δ dunkler ist als der aktuelle. Es

wurde nicht der nächte Pixel gewählt, da sich der Übergang von hell zu dunkel über 2-3 Pixel erstreckt. So kann ein grösseres δ gewählt werden, was zu weniger fehlerhaften Erkennungen führt.

Die Nebenbedingung für $I_{i+w/32}$ wurde eingeführt, um grosse dunkle Bereiche zu ignorieren, wie zB. Schattenverläufe. Wir nehmen also an, dass unsere Punkte immer kleiner als $w/32$ Pixel sind.

Das Ende des dunklen Bereichs an der Stelle j erkennen wir entsprechend:

$$I_j - I_{j+2} < -\delta$$
$$I_i < I_j$$

Die zweite Gleichung sollte sicherstellen, dass der Bereich geschlossen wird, auch wenn die Zunahme der Helligkeit nicht abrupt genug ist für den Parameter δ.

Als Position der dunklen Bereiche $b_{v,1}, b_{v,2}, \ldots$ speichern wir in jeder Zeile v den Mittelwert von i und j ab.

2.2 Zuordnung zwischen den Zeilen

Da wir die dunklen Bereiche nur zeilenweise erkennen, um die Cacheeffizienz zu gewährleisten, müssen wir am Ende jeder Zeile die gefundenen Bereiche der aktuellen Zeile $b_{v,1}, b_{v,2}, \ldots$ mit jenen der vorherigen Zeile $b_{v-1,1}, b_{v-1,2}, \ldots$ zuordnen.

Dabei müssen die b_v im Interval von $[b_{v-1}-w/64, b_{v-1}+w/64]$ befinden, um zum oberen Bereich zu gehören. Nicht zugeordnete b_v gelten als neue Punkte. Die nicht zugeordnete b_{v-1} sind vollständige erkannte Punkte, die geschlossen werden können. Dabei ergibt sich als u-Koordinate der Mittlerwert aller b_v die zur Zurordnung gehören und als v-Koordinate die aktuelle Zeilenzahl minus die halbe Anzahl der b_v: $v-|b_v|/2$.

3 Algorithmus 1

Der Algorithmus 1 nützt die Verzerrung in der *x*-Richtung aus um eine Translation in *x*-Richtung und eine Rotation um die *z*-Achse zu erkennen.

Er braucht vier Punkte, welche in einer Linie angeordnet sind (Abbildung 3). Bei einer Rotation verändern sich die Abstände der Punkte (Abbildung 4).

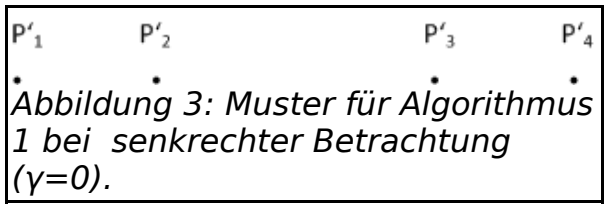

Abbildung 3: Muster für Algorithmus 1 bei senkrechter Betrachtung (γ=0).

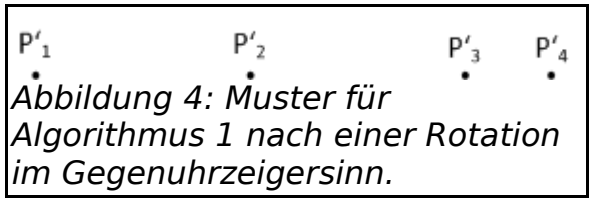

Abbildung 4: Muster für Algorithmus 1 nach einer Rotation im Gegenuhrzeigersinn.

Da die Punkte, welche unser Punkterkennungsalgorithmus liefert, nicht immer in der gleichen Reihenfolge sind, müssen diese zuerst nach ihren x-Koordinaten sortiert werden.

Die Distanz zwischen den Punkten P_1 und P_2 bzw. P_3 und P_4 sowie der Abstand von der Kamera zur Wand müssen vorher bekannt sein. Wir nennen die Distanz der Punkte *pointDistance* und der Abstand zur Wand ist *y.*

Der Ursprung des Koordinatensystems befindet sich in P_1.

3.1 Berechnung

Der erste Schritt ist aus den Koordinaten der Bildpunkte etwas zu berechnen, das uns hilft die Situation in der wirklichen Welt zu beschreiben. Dafür wollen wir β_i als Öffnungswinkel zwischen der Mitte des Bildes und dem Punkt P_i berechnen.

In der Abbildung 5 sieht man, eine schematisch Darstellung der Kamera. Der Punkt P_1 wird auf den Kamerasensor im Abstand *f* projiziert. Dieser Abstand ist die Brennweite der Linse und somit ein konstanter Kameraparameter. Der Bildpunkt *M*' bezeichnet den Mittelpunkt des Bildes, somit bezeichnet also *M* den Punkt, auf den die Kamera gerichtet ist. Der Punkt *C* bezeichnet die Position der Kamera. Die Breite des Bildes in Pixel wird als *w* bezeichnet.

Der Winkel β kann mit folgender Formel über das rechtwinklige Dreieck aus *M, P_i* und *C* oder aus dem Dreieck aus *C* und den Bildpunkten *M*' und *P*'$_i$ bestimmt werden.

$$\beta_i = \arctan\left(\frac{\left|\overline{P'_i M'}\right|}{f}\right)$$

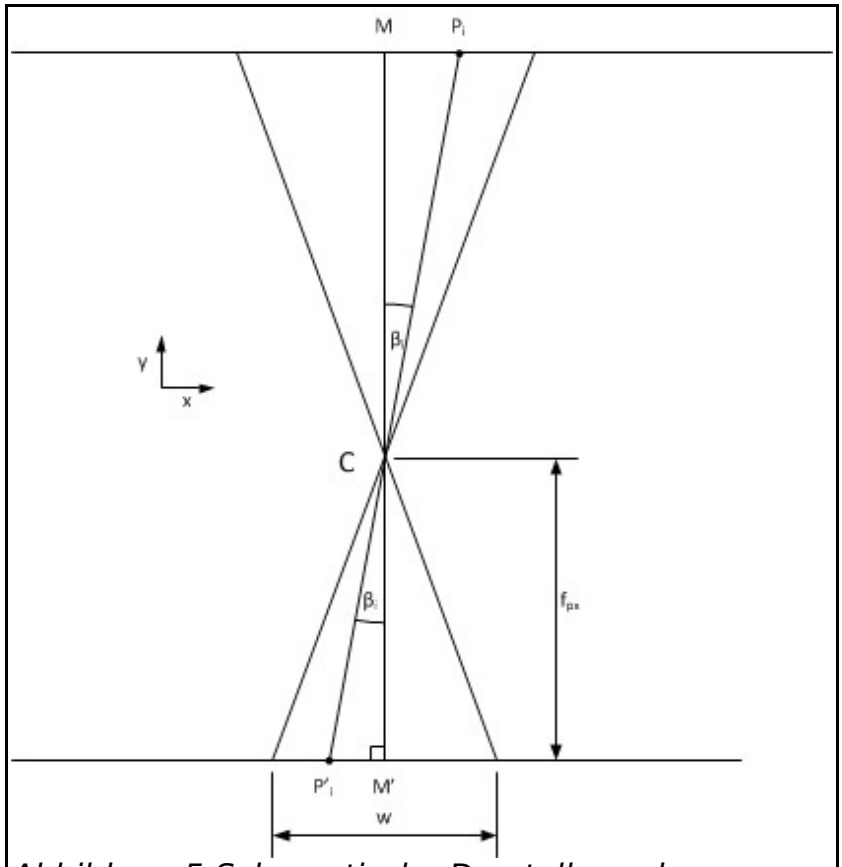

Abbildung 5:Schematische Darstellung der Kamera zur Berechnung von β_i

Da die Position von P_1 in Pixel gegeben ist, ist die Messeinheit der Strecke $\overline{P'_i M'}$ auch Pixel. Deshalb müssen wir die Brennweite ebenfalls in Pixel umrechnen und nennen diese f_{px}. Da M' in der Mitte des Bildes ist, ist seine u-Koordinate die Hälfte der Breite des Bildes, also $w/2$. Wenn wir wir die u-Koordinate von P'_i mit u_i bezeichnen, können wir β_i folgendermassen ausdrücken:

$$\beta_i = \arctan\left(\frac{u_i - w/2}{f_{px}}\right)$$

Nach dieser Definition ist β_1 negativ, wenn P_1' weiter links ist als M'. Damit können wir zu jedem Punkt P_i ein Winkel β_i berechnen.

In Abbildung 6 sieht man eine allgemeine Situation. Die Blickrichtung der Kamera (Pfeil zu M) steht im Winkel γ zur Senkrechten durch die Kameraposition C und dessen Projektion auf die Wand C^*. γ ist negativ, wenn der Punkt M weiter links ist als C. Das Ziel ist nun den Winkel γ und die x-Koordinate der Kameraposition zu bestimmen.

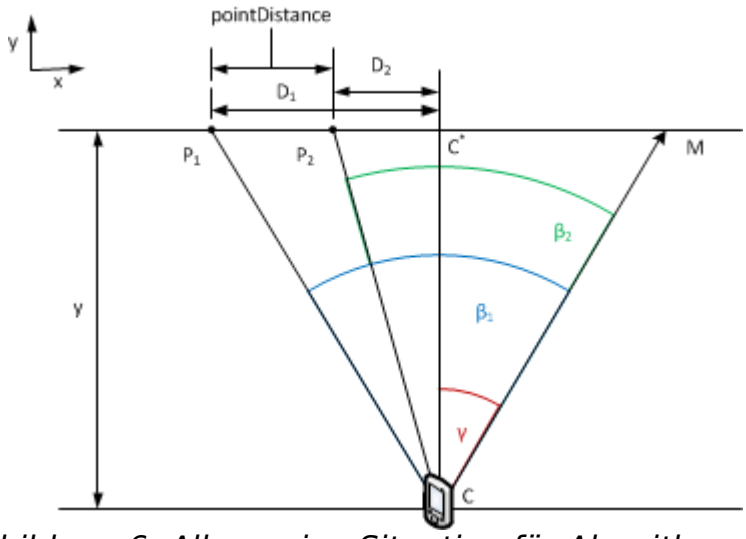

Abbildung 6: Allgemeine Situation für Algorithmus 1

Den Abstand D_i vom Punkt P_i zum Punkt C^* entspricht der gesuchten Kamerapostion und ist folgendermassen definiert:

$$D_i = y \cdot \tan(\gamma + \beta_i)$$

Nach dieser Definition ist D_i negativ, wenn β_i grösser ist als γ, was der Fall ist, wenn P_1 weiter links ist als C^*.

Des weiteren ist die Strecke *pointDistance* konstant und immer Positiv. Somit kann folgende Gleichung aufgestellt werden:

$$pointDistance = D_2 - D_1$$

Setzt man nun die Definitionen von D_1 und D_2 ein, erhält man diese Gleichung für γ:

$$pointDistance = y \cdot (\tan(\gamma + \beta_2) - \tan(\gamma + \beta_1))$$

Beschränkt man γ auf den Bereich von $-\pi/2$ bis $\pi/2$ ist diese nicht lineare Gleichung analytisch lösbar, ergibt aber 2 Lösungen. Um ohne weitere Einschränkungen entscheiden zu können, welches die richtige Lösung ist kann man γ auf die gleiche Weise aus P_3 und P_4 berechnen. Dann erhält man 4 Lösungen, von welchen 2 übereinstimmen. Dies ist der Grund weshalb 4 Punkte benötigt werden.

In der Praxis stimmen diese Lösungen wegen Messfehler nicht exakt überein. Um trotzdem zuverlässig die richtigen Lösungen identifizieren zu können, muss der Abstand zwischen P_2 und P_3 genügend gross sein. Ein grosser Abstand schränkt aber die Bewegungsfreiheit ein. Die doppelte *pointDistance* hat sich als guter Kompromiss erwiesen.

Der gesuchte x-Wert ist dann:

$$x = -D_1 = -y \cdot \tan(\gamma + \beta_1)$$

3.2 Probleme

Es traten vor allem 2 Probleme auf.

Das erste Problem ist, dass nie garantiert werden kann, dass man immer die beiden richtigen Lösungen findet. Liegen die Fehler ungünstig, kann es zu Situationen kommen, in denen fälschlicherweise eine korrekte Lösung ignoriert und eine falsche Lösung benutzt wird.

Das zweite und grössere Problem ist, dass die Lösung sehr stark davon abhängt wie weit man von der Wand entfernt ist. Diese Entfernung sollte vorher bekannt sein. Falls diese aber nicht exakt stimmt kommt es zu grossen Ungenauigkeiten oder gar zu komplexen und deshalb unbrauchbaren Lösungen.

Solange man das Mobiltelefon mittels eines Stativs ruhig hält, hält sich das Problem in Grenzen. Sobald man es aber in die Hand nimmt, ist es sehr schwer die richtige Distanz zu finden und zu halten, weshalb sich grosse Fehler ergeben, oder gar keine Lösung gefunden wird.

Aus diesem Grund finden wir, dass der Algorithmus 1 praktisch nicht benutzbar ist, obwohl er auf dem Stativ zufriedenstellende Genauigkeiten liefert.

4 Algorithmus 2

Der Algorithmus 2 ist in der Lage alle drei Translationsrichtungen und eine Rotation zu erkennen. Dabei müssen vier Punkte erkannt werden, die auf der Wand quadratförmig angeordnet sind. Bei einer Rotation, werden die Seiten des Quadrats verkürzt dargestellt im Kamerabild.

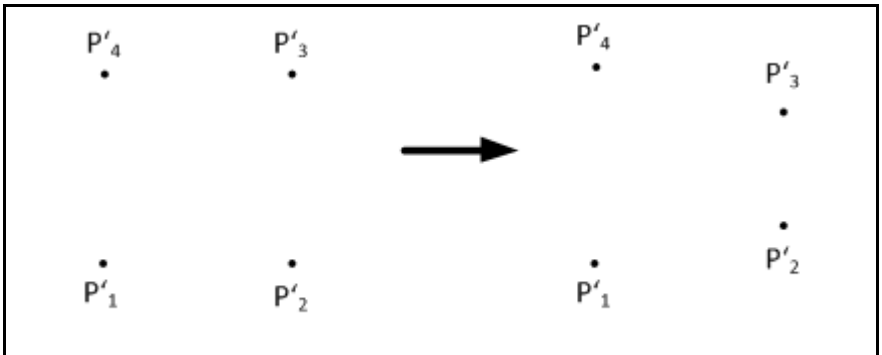

Abbildung 7: Links sind die erkannten vier Punkte, wenn man senkrecht auf die Wand blickt. Bei einer Rotation der Kamera im Uhrzeigersinn, verkürzt sich nun die Strecke $|P_2'P_4'|$

Die Abstände $|P_1P_2|=|P_2P_3|=|P_3P_4|=|P_4P_1|$ bezeichnen wir als *pointDistance*. Der Koordinatenursprung haben wir in der Mitte von P_1 und P_4 gewählt.

4.1 Grundlagen der Berechnung

Um die nachfolgende Berechnung zu verstehen, möchte wir erst die folgende Formel einführen.

$$\frac{d}{\left|\overrightarrow{P_1}-\overrightarrow{P_2}\right|}=\frac{f_{px}}{\left|\overrightarrow{P'_1}-\overrightarrow{P'_2}\right|} \quad \text{(I)}$$

Sie besagt, dass wir die Strecke *d* berechnen können, wenn wir von zwei beliebigen Punkten den Abstand kennen und sie auf dem Bild detektieren. Dies gilt jedoch nur, wenn die beiden Punkte in einer Ebene liegen, die senkrecht zur Blickrichtung steht.

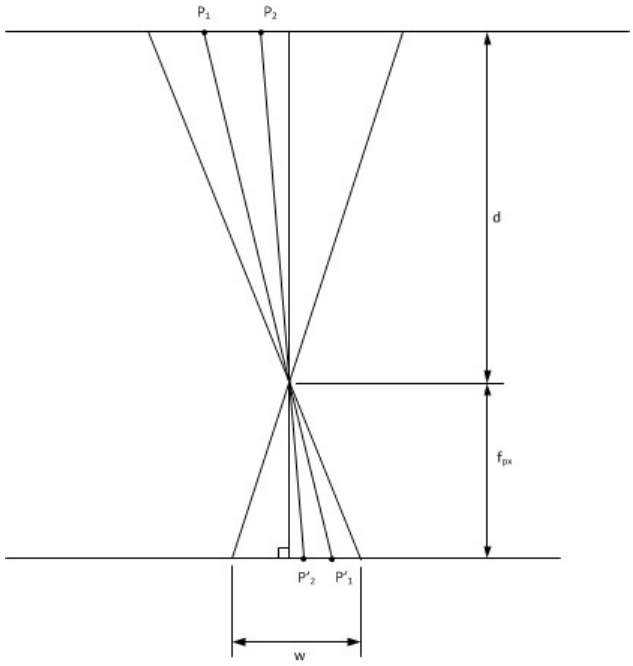

Abbildung 8: Skizze zu Formel (I)

4.2 Winkel Berechnung

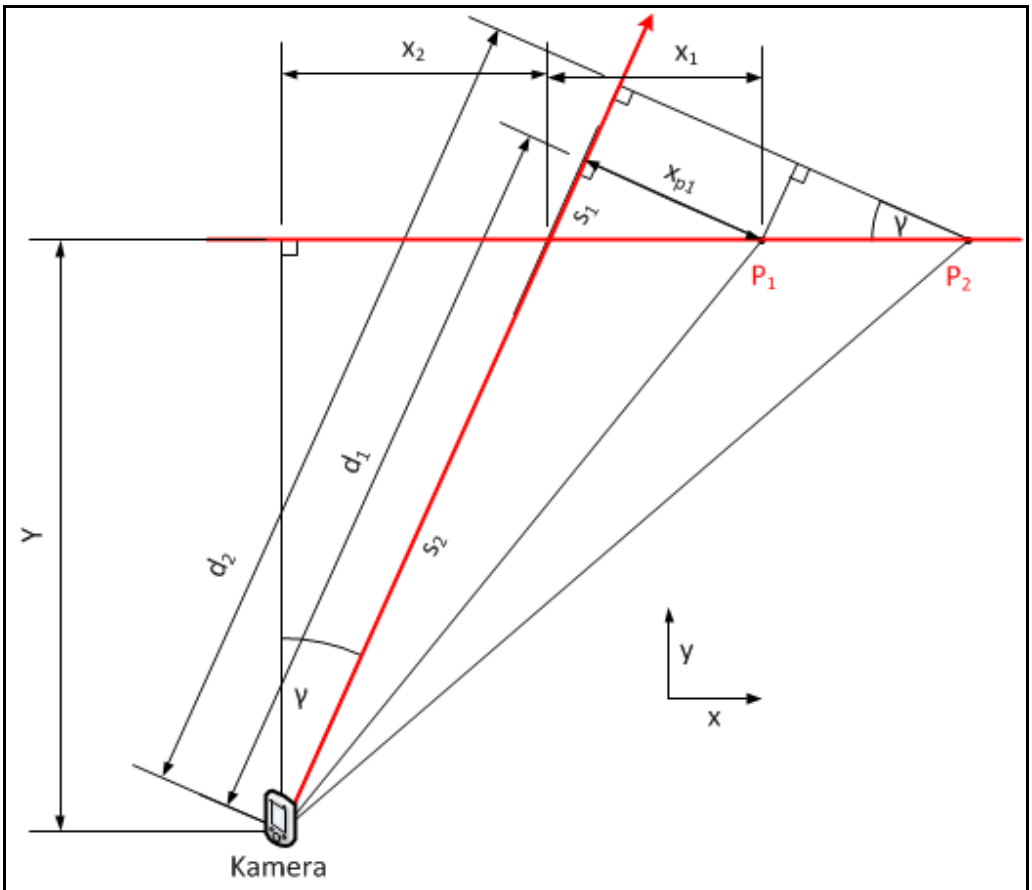

Abbildung 9: Skizze Algorithmus 2

Als erstes berechnen wir die Rotation der Kamera, also den Winkel γ. Dazu verwenden wir die vorherige Formel (I) für d_1 und d_2.

$$\frac{d_1}{pointDistance} = \frac{f_{px}}{\left|\overrightarrow{P'_1} - \overrightarrow{P'_4}\right|} \qquad \frac{d_2}{pointDistance} = \frac{f_{px}}{\left|\overrightarrow{P'_2} - \overrightarrow{P'_3}\right|}$$

Zu beachten ist, dass in der Abbildung 9 die Punkte P_4 und P_3 in z-Richtung über P_1 und P_2 liegen und deshalb nicht dargestellt sind. Dies ist auch der Grund, weshalb d_1 und d_2 länger sind als s_2 und durch die Wand hindurch verlaufen. d_1 ist definiert als der Abstand zur Ebene, die senkrecht zur Blickrichtung steht und die beiden Punkte P_1 und P_4 enthält. Für d_2 gilt entsprechend das selbe mit P_2 und P_3.

Nun kann man den Winkel γ mit dem Sinus ausrechnen.

$$\sin\gamma = \frac{d_2 - d_1}{\left|\overrightarrow{P_1} - \overrightarrow{P_4}\right|} = \frac{d_2 - d_1}{pointDistance}$$

$$\boxed{\gamma = \arcsin\left(f_{px}\left(\frac{1}{\left|\overrightarrow{P'_2} - \overrightarrow{P'_3}\right|} - \frac{1}{\left|\overrightarrow{P'_1} - \overrightarrow{P'_4}\right|}\right)\right)} \quad \text{(II)}$$

4.3 Translations Berechnung

Nun möchten wir die Translationskoordinaten *x*, *y* und *z* berechnen, dazu brauchen wir erst x_{p1} (siehe Abbildung 9). Dazu verwenden wir nochmals die Formel (I). Jedoch wählen wir die beiden Punkte für die Formel: $(P_1+P_4)/2$ und den Mittelpunkt des Bildes.

$$\frac{|x_{p1}|}{d_1} = \frac{\left|\begin{pmatrix} w \\ h \end{pmatrix}/2 - (\overrightarrow{P'_1} + \overrightarrow{P'_4})/2\right|}{f_{px}} \quad \rightarrow \quad |x_{p1}| = \frac{d_1\left|\begin{pmatrix} w \\ h \end{pmatrix} - \overrightarrow{P'_1} - \overrightarrow{P'_4}\right|}{2 f_{px}}$$

Da durch den Betrag das Vorzeichen wegfällt, wird nicht mehr unterschieden, ob sich x_{p1} links oder rechts vom Mittelpunkt befindet. Das Vorzeichen muss man wieder hinzufügen.

$$x_{p1} = signum(w - \overrightarrow{P'_1} + \overrightarrow{P'_4}) \cdot |x_{p1}|$$

Nun kann man weitere trigonometrische Beziehungen nutzen, um die restlichen Strecken zu berechnen:

$$x_1 = \frac{x_{p1}}{\cos\gamma} \quad , \quad s_1 = x_{p1}\tan\gamma \quad , \quad s_2 = d_1 + s_1 = d_1 + x_{p1}\tan\gamma \quad , \quad x_2 = -s_2\sin\gamma$$

Zu Beachten ist, dass in der Skizze x_{p1} negativ ist. Auch sollten x_1 und x_2 negativ sein. Deshalb ergeben sich die Vorzeichen in der s_2 und x_2 Formeln.

$$\boxed{\begin{array}{c} x = x_1 + x_2 = \frac{x_{p1}}{\cos}\gamma - \sin\gamma(d_1 + x_{p1}\tan\gamma) \\ y = s_2\cos\gamma = \cos\gamma(d_1 + x_{p1}\tan\gamma) \end{array}}$$

Für die *z*-Koordinate verwenden wir wiederum Formel (I). Jedoch ersetzen wir *d* durch *y* und für die Bildpunktdistanzen setzen wir die *v*-Abweichung des Musters zur Mitte ein.

$$\frac{z}{y} = \frac{(height - \overrightarrow{p'_1}[2] - \overrightarrow{p'_4}[2])/2}{f_{px}} \quad \rightarrow \quad \boxed{z = \frac{y(height - \overrightarrow{p'_1}[2] - \overrightarrow{p'_4}[2])}{2 f_{px}}}$$

4.4 Theoretische Genauigkeit

Die theoretische Genauigkeit des Algorithmus 2 ist bestmögliche Genauigkeit, die durch die Auflösung des Bildes bestimmt wird. Für die Berechnung betrachten wir den senkrechten Fall ($\gamma=0$) und bestimmen, wie viel ein Messfehler von einem Pixel den Winkel γ beeinflusst. Dazu führen wir diesen Parameter ein:

$$d_{ratio} = \frac{y}{pointDistance}$$

Dieses Verhältnis gibt die Entfernung zur Wand normiert auf die *pointDistance* an. Auch bestimmt es, wie gross wie den Abstand der Punkte im Bild ist.

$$\frac{f_{px}}{d_{ratio}} = \frac{f_{px} \cdot pointDistance}{y} = \left|\overrightarrow{P'}_1 - \overrightarrow{P'}_2\right| \quad \text{Nach Formel (I)}$$

Nun können wir dies in die Formel (II) einsetzen und addieren zu einer Strecke den Fehler e_{px} hinzu.

$$\gamma_{error} = \arcsin\left(f_{px}\left(\frac{1}{f_{px}/d_{ratio}} - \frac{1}{f_{px}/d_{ratio} + e_{px}}\right)\right) = \arcsin\left(\frac{d_{ratio}^2 e_{px}}{f_{px} + d_{ratio} e_{px}}\right)$$

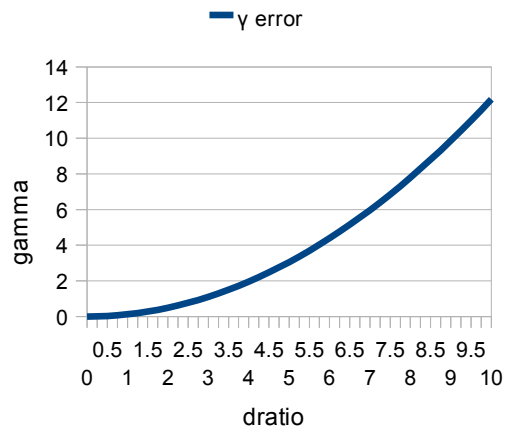

Abbildung 10: Dies ist der Plot der γ_{error} Funktion mit e_{px}=1 und f_{px}=464. Die Kurve beschreibt, wie gross die Abweichung vom gewünschten γ=0 ist, wenn ein Punkt um ein Pixel falsch erkannt wurde.

5 Messergebnisse

5.1 Algorithmus 2

Um die theoretische Genauigkeit zu illustrieren haben wir ein Messung gemacht, bei welcher wir die Distanz der Kamera zur Wand erhöht haben. Wir haben bei einem d_{ratio} 4.7 angefangen und am Ende hatten wir einen Wert von 9.4. Dabei haben wir uns um 15 cm bewegt und währenddessen das Verhalten des Winkels γ aufgezeichnet. Da wir keine Rotation durchgeführt haben, sollte γ konstant sein (Abbildung 11).

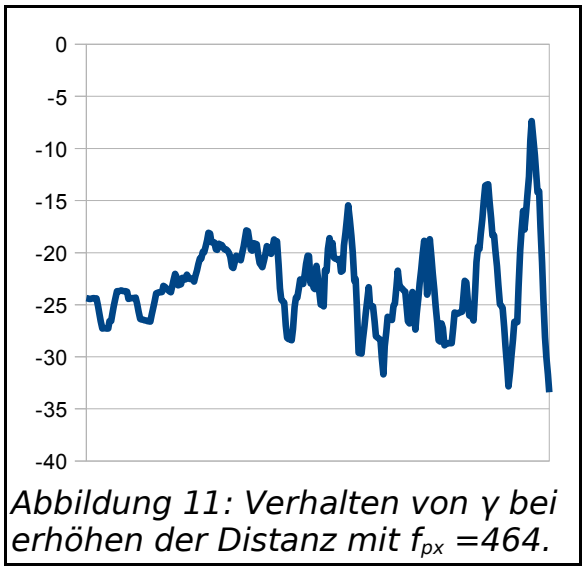

Abbildung 11: Verhalten von γ bei erhöhen der Distanz mit f_{px} =464.

Man sieht dass sich erwartungsgemäss die Schwankungen erhöhen, der Mittelwert sich aber nicht gross verändert.

5.2 Gegenüberstellung von Algorithmus 1 und Algorithmus 2

Weiter haben wir versucht das Verhalten der beiden Algorithmen zu vergleichen.

Als erstes haben wir γ um 13° verringert und wieder das Verhalten aufgezeichnet Man sieht, dass der Algorithmus 1 die Bewegung ohne grössere Schwankungen aufzeichnet, während es beim Algorithmus 2 zu starken Schwankung kommt. (Abbildung 12 und 13)

Die zweite Messung war eine Bewegung um 5 cm in x-Richtung wobei wir diesmal das Verhalten des x-Wertes aufgezeichnet haben. Man sieht wieder, dass der Algorithmus 1 die Bewegung kontinuierlicher aufzeichnet als der Algorithmus 2. Man sieht aber auch einen grossen Ausreisser. Dieser könnte von einer falschen Lösung handeln. (Abbildung 14 und 15)

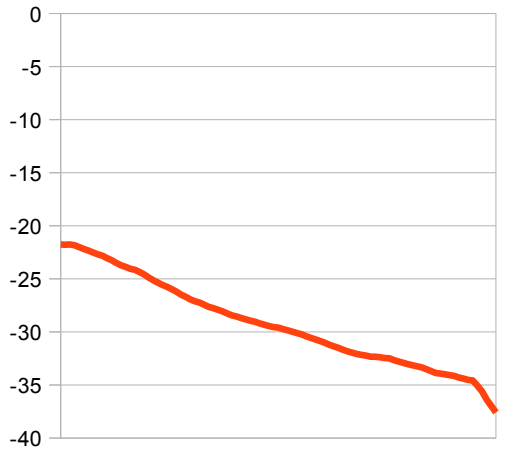

Abbildung 12: Verhalten von γ bei Algorithmus 1 bei einer Rotation.

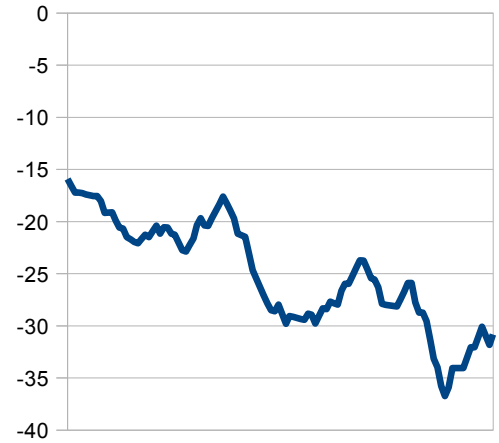

Abbildung 13: Verhalten von γ bei Algorithmus 2 bei einer Rotation.

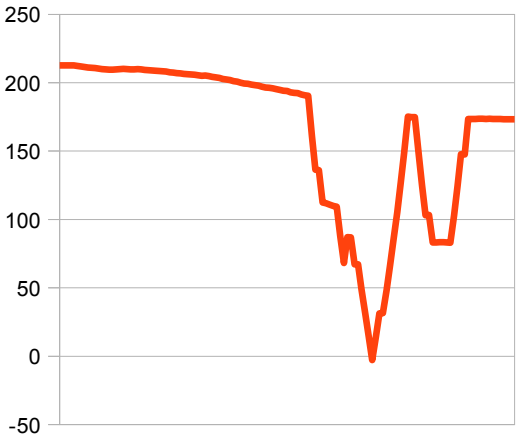

Abbildung 14: Verhalten der x-Koordinate beim Algorithmus 1 bei einer Translation.

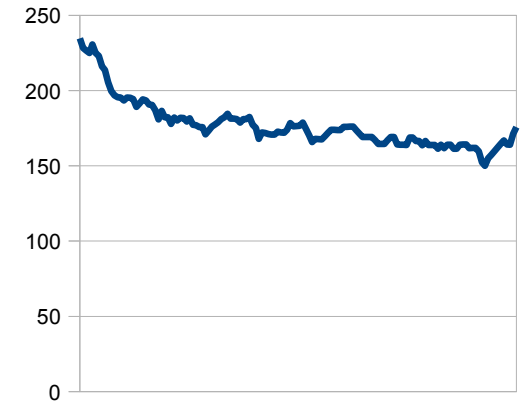

Abbildung 15: Verhalten der x-Koordinate beim Algorithmus 2 bei einer Translation.

6 Ausblick

Zum Schluss wollen wir uns noch Gedanken machen, wie man unser System weiterentwickeln könnte.

6.1 Farbkodierung

Wir haben mit schwarzen Punkten auf weissem Hintergrund gearbeitet. Mit Farbigen Punkten könnte man die Zuverlässigkeit des Punkterkennungsalgorithmus erhöhen und somit die Robustheit des ganzen Systems verbessern.

6.2 Erweiterung des Musters zu einem Gitter

Man könnte sich vorstellen das Muster vielmals zu wiederholen und zu einem Gitter erweitern, um so mehr Bewegungsfreiheit zu schaffen. Man müsste dann, zum Beispiel mit Farbigen Punkten, gewisse Elemente schaffen, welche sich Abheben, um sich zu Orientieren.

6.3 In die reale Welt

Die interessanteste Weiterentwickelung wäre, wenn man sich von der Testumgebung verabschieden würde und sich in die reale Welt begeben würde. Man könnte mittels eines Feature Detection Algorithmus (z.B. SURF, SIFT) Punkte entdecken und aus diesen die Position schätzen.

Die Perfomance wird wahrscheinlich ein Problem sein. Man könnte aber versuche die Feature Detection in der GPU mittels Shader zu rechnen.