# Posebestimmung mittels AprilTags

Marcus Opdenberg
E-Mail: mopdenberg@uni-koblenz.de, Matrikelnummer: 211201917
Seminar Mobile Systems Engineering, Sommersemester 2016, Universität Koblenz-Landau

***Zusammenfassung*—Die folgende Arbeit zeigt eine Möglichkeit auf, die Position von künstlichen Landmarken[1] im Raum mittels je eines Kamerabildes zu ermitteln. Das im Folgenden beschriebene Verfahren verwendet AprilTags als Landmarken und wird hinsichtlich seiner Leistungsfähigkeit mit älteren Verfahren verglichen. Es wird dabei ausgeführt, wie die Tags lokalisiert und verifiziert werden.**

## I. Einleitung

Das von Edwin Olson (Universität Michigan) publizierte Verfahren zur Positionsbestimmung von künstlichen Landmarken verwendet die eigens entwickelten AprilTags, um mit Hilfe eines Kamerabildes von einer kalibrierten Kamera die 6-DOF-Positionen[2] aller sichtbaren Tags zu ermitteln. Die Lokalisation von künstlichen Landmarken ist dabei weniger realitätsnah, defür jedoch simpler und stabiler, als die Positionsbestimmung von natürlichen Merkmalen (features) und wird verwendet, um eine Wissensbasis zu schaffen (ground truthing), auf der wiederum andere Algorithmen getestet werden können. [Ols11]
Ein Tag stellt eine Art 2 dimensionalen Barcode da und muss mit seinem Muster sowohl seine Orientierung als auch seine ID codieren, wobei die ID einen definierten Abstand[3] zu allen anderen verwendeten IDs haben muss, um Fehler kompensieren zu können. Von den Tags wird dabei erwartet, dass diese auch auf große Distanz[4] leicht erkannt werden und sich möglichst gut untereinander unterscheiden. Auch wird vorausgesetzt, dass die Tags möglichst unabhängig von der Beleuchtung oder Verdeckung richtig erkannt werden. Dazu muss das Muster sehr viel gröber sein, als es beispielsweise bei einem QR Code der Fall ist. Beim QR-Code wird vom Nutzer erwartet, dass dieser die Kamera in eine Position bringt, unter der der Code leicht ausgelesen werden kann. Ein Tag soll auch dann gefunden und korekt ausgelesen werden, wenn sich dieser weit weg in einem schlechten Winkel[5] zur Bildebene befindet und dabei unterschiedlich beleuchtet und teilweise verdeckt ist. Die Unterschiede in der Auflösung werden beim Vergleich von QR Codes mit AprilTags deutlich, so verwenden QR Codes bereits 268 Pixel, um die Ausrichtung zu codieren, während ein AprilTag (je nach Größe) 49 bis 100 Pixel verwendet, um Ausrichtung und ID zu codieren.[Ols11]
Der Anwendungsbereich von AprilTags liegt beispielsweise im Bereich Augmented-/Virtual- Reality, wo eine Kamera in einer AR-Brille einen Tag findet und an dessen Stelle eine assoziierte Information einblendet. Invertiert könnte auch eine Person, die eine VR-Brille trägt mit Tags markiert und von Kameras lokalisiert werden, sodass die getätigten Bewegungen in die virtuelle Welt übertragen werden können. Da die Erkennung von Objekten anhand natürlicher Features schwieriger ist, als das Erkennen von künstlichen Tags, werden diese gerne in der Robotik (als Zwischenlösung) eingesetzt, wodurch Roboter unterschiedliche Objekte oder Befehle eines Menschen[6] leichter erkennen können. Für die Lokalisierung der Tags wird vorausgesetzt, dass die verwendete Kamera kalibriert ist und die Größe der zu erkennenden Tags bekannt ist. An der gleichen Universität wurde zudem ein Verfahren entwickelt, dass es erlaubt Kameras mit Hilfe von AprilTags zu kalibrieren, was genutzt werden kann, um die Voraussetzungen für die Lokalisation mit AprilTags zu schaffen.[RSO13] Das vorgestellte Verfahren soll sich durch seine Leistungsfähigkeit, Geschwindigkeit und Offenheit von der Konkurenz absetzen.

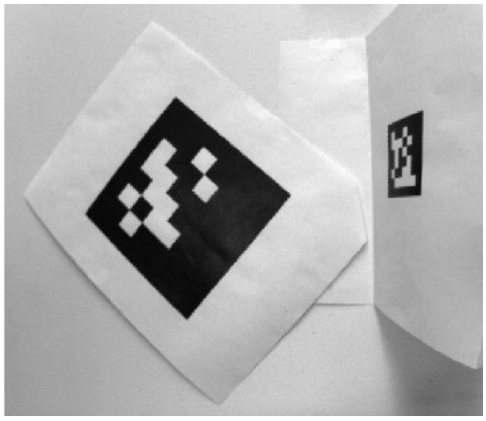

Abbildung 1. Beispiel-Eingabe-Bild für die Lokalisation von AprilTags [Ols11].

## II. Funktion der Lokalisation

Die Lokalisation besteht im Wesentlichen aus zwei Komponenten, die im Folgenden näher beschrieben werden: Der Detektor (Tag-Detector) sucht nach 4-seitigen Regionen im Bild, deren Inneres dunkler ist, als ihr Äußeres. Diese sind Kandidaten für gefundene Tags. Die Decodierung (codingsystem) überprüft daraufhin, ob es sich bei den gefundenen Regionen um gültige Tags handelt und berechnet dessen relative Positionen mit Hilfe von Homographien.

### A. Tag-Detektor

Die Funktionsweise des Detektors wird anhand von Grafiken aus [Ols11] verdeutlicht. Das Ausgangsbild ist dabei das in Abbildung 1 dargestellte. Besagtes Bild ist, verglichen mit üblichen Szenarien, eher simpel, da hier weder Verdeckung

[1] zwei dimensionale, zwei farbige Muster - Details über die gesamten Anforderungen in der Einleitung
[2] 3D Koordinaten und Rotation
[3] Hammingdistanz zwischen 2 Codewörtern
[4] das Tag-Muster muss deutlich größer gesehen werden, als die Pixel auf die es abgebildet wird
[5] Winkel zwischen Bild und Tag-Ebene geht gegen 90°
[6] der Mensch hält einen Tag hoch, der mit einem Befehl assoziiert wird

noch unterschiedliche Beleuchtung der Tags vorzufinden ist. Es ermöglicht jedoch eine aufs wesentliche reduzierte Veranschaulichung des Vorgehens.
Der Detektor wurde laut [Ols11] derart gestaltet, dass dieser eine überaus geringe falsch-negativ Rate[7] und im Umkehrschluss eine hohe falsch-positiv[8] Rate hat. Die fälschlicherweise gefundenen Tag-Kandidaten werden ein einem späteren Schritt anhand ihrer unpassenden Codierung entfernt.
Im ersten Schritt werden die Intensitäten und Richtungen der Gradienten für jeden Pixel im Eingabebild berechnet (siehe Abbildung 2 oben). Im Folgenden werden Diese anhand ihrer Richtung und Intensität gruppiert. Zum Gruppieren wird ein Graph aufgespannt, in dem jeder Knoten einen Pixel im Bild entspricht. Darin werden benachbarte Pixel durch Kanten verbunden, deren Gewicht die Differenz der Gradienten-Richtung ist. Diese Kanten werden der Reihe nach mit aufsteigendem Kantengewicht abgearbeitet, wobei für jede Kante überprüft wird, ob die Komponenten[9], die sie verbindet eine ähnliche Richtung und Intensität haben. Für eine Komponente $n$ wird die Spannweite der Richtungen (beachte Zyklizität der Richtungen zwischen 0 und $2\pi$, Winkeldifferenz in einer Gruppe kleiner als $\pi$) als $D(n)$ und Die der Intensitäten als $M(n)$ bezeichnet. Ob zwei Komponenten ähnlich genug sind, wird anhand folgender zwei Gleichungen überprüft: [Ols11]

$$\begin{aligned} D(n \cup m) &\leq \min(D(n), D(m)) + K_D/|n \cup m| \\ M(n \cup m) &\leq \min(M(n), M(m)) + K_M/|n \cup m| \end{aligned} \quad (1)$$

Die beiden Kriterien lassen sich natürlich-sprachlich derart formulieren, dass zwei Komponenten vereinigt werden, wenn die Elemente der Vereinigung in etwa so ähnlich zueinander sind, wie die Elemente in den beiden Komponenten jeweils Komponenten-intern zu einander ähnlich sind. $K_D$ und $K_M$ sind dabei Parameter, die eine Verringerung der Ähnlichkeit bei der Kombination zweier Komponenten zulassen. Der Einfluss von $K_M$ und $K_D$ nimmt bei steigender Komponenten-Größe jedoch schnell ab. Das Ergebnis eines solchen Clusterings lässt sich durch die aus Abbildung 1 berechnete Abbildung 2 (unten links) darstellen, wobei eine Farbe je einem Cluster entspricht.
Bei einem Gradienten-basierten Ansatz ist zu beachten, dass die Richtungen der Gradienten benachbarter Pixel durch Fehler und Rauschen im Eingabebild stark variieren können, sodass nur wenige Pixel gruppiert werden. Um das Problem zu mindern, muss das Bild vor der Gradientenbestimmung geglättet werden, wodurch kleine Bild-Störungen herausgefiltert werden. Dazu wird an dieser Stelle ein Gauss-Filter verwendet. Der Einsatz eines Weichzeichners setzt voraus, dass die Codierung des Tags grob genug ist, um anschließend noch ausgelesen werden zu können.
Im Folgenden wird für alle Punkte je einer Komponente eine Linie mit Hilfe einer Kleinste-Fehler-Quadrate-Optimierung angenähert. Jeder Punkt wird dabei mit seiner Gradienten-Stärke gewichtet. Zudem wird die Richtung der zum Gradienten senkrechten Linie derart vermerkt, dass sich die dunklere Seite links von der Linie befindet (siehe Abbildung 2 unten rechts). Dies ermöglicht im nächsten Schritt, Linien so zu einem Quad (4-seitiges Objekt im Bild) zusammenzufassen, dass im Inneren des Quad nur die dunklen (bzw. nur die hellen) Halbebenen der Linien anzutreffen sind. Bei der Quad-

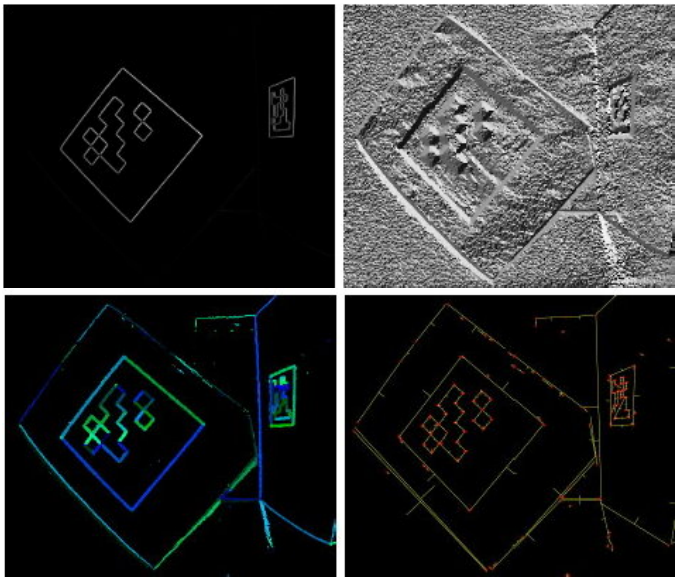

Abbildung 2. oben: Intensitäten (links) und Richtungen der Gradienten; unten: Clustering gleicher Farben (links) und die Linien-Repräsentation von Clustern (die Richtung der Linien wird durch die kleine Normale an einer jeden Linie dagestellt, diese zeigt in die hellere Halbebene) [Ols11].

Berechnung liegt der Anspruch darin, ein Quad auch dann zu finden, wenn eine oder mehrere Seiten des Quads teilweise verdeckt oder von Rauschen gestört werden. Es ist zudem zu beachten, dass die gerichteten Linien nur zyklisch angeordnet werden dürfen, um ein Quad zu formen (vgl. oben: nur helle/ dunkle Halbebenen im Inneren des Quads). Zu diesem Zwecke verwendet der Autor eine rekursive Tiefensuche mit einer Tiefe von 4. Dabei wird auf der obersten Ebene mit allen gefunden Linien begonnen. In den drei darunter liegenden Ebenen werden diese Linien mit anderen Linien verbunden, deren Anfang "nahe genug" am Ende der Linie der oberen Ebene liegt und deren Richtung gegen den Uhrzeigersinn gedreht wurde. Die Definition von "nahe genug" legt dabei fest, wie robust das Verfahren gegen eine teilweise Verdeckung von Linien ist und wird im Abschnitt "Wahl geeigneter Parameter" präzisiert.
Vier Linien mit oben genannten Eigenschaften bilden einen Quad-Kandidaten, dessen Eckpunkte aus den Schnittpunkten der (ggf. verlängerten) Linien berechnet werden. An dieser Stelle ist zu beachten, dass die Linien mit Hilfe einer Kleinsten-Fehler-Quadrate-Optimierung berechnet wurden und somit präziser sind als die einzelnen Pixel, aus denen diese berechnet wurden. Folglich sind auch die Eckpunkte eines Quads Sub-Pixel genau.

### *B. Berechnung der Homographie*

Im Folgenden kann eine Homographie genutzt werden, um die Rotation und Translation des Tags im Kamerabild zu berechnen. Eine Homographie ermöglicht es, die Verschiebung und Drehung mehrerer Kamerabilder zueinander zu berechnen, die eine identische Ebene in der Welt betrachten. Hier wird das Tag von der Kamera an einer beliebigen Stelle gesehen und es ist das Ziel, die Pose[10]-Differenz zu einer virtuellen

[7] der Detektor findet ein Tag im Bild nicht, obwohl es vorhanden ist

[8] Bild-Regionen, die keine Tags sind, werden als Tag detektiert

[9] Komponenten bestehen aus einem oder mehreren Knoten-Pixeln mit ähnlicher Gradienten-Richtung

[10] Position und Drehung

Kamera zu berechnen, die frontal auf das Tag blickt. Die Berechnung der Homographie erfolgt durch das Abbilden von bekannten Punkte eines Bildes auf bekannte Punkte eines weiteren Bildes. Dafür werden die Subpixel-genau berechneten Eckpunkte eines Quads auf die Eckpunkte eines Quadrats der Größe eines Tags abgebildet. Da die Berechnung der Homographie in homogenen Koordinaten[11] statt findet, kann erst durch Wissen über die physische Größe des Tags und die Beschaffenheit des Bild-Sensors eine Aussage über die eindeutige Distanz des Tags getroffen werden. Die Berechnung der 3×3 Homographie-Matrix kann anhand der Direkten-Linearen-Transformation gemäß [HZ04](s.35) erfolgen.
Die auf vorherige Art und Weise bestimmte Homographie-Matrix setzt sich aus der 3×4 Kamera-Projektions-Matrix (intrinsisch) und der 4×4 extrinsischen-Matrix[12] zusammen. Daher liefert die folgende Zerlegung der gegebenen Homographie-Matrix die Pose des AprilTags. Da von einer kalibrierten Kamera (z.B. durch [RSO13]) ausgegangen wird, kann die Kamera-Projektions-Matrix als bekannt vorausgesetzt werden. Dadurch, dass das Referenz-Tag senkrecht von oben betrachtet wird, kann es keinen Bildpunkt im Tag Koordinatensystem geben, dessen Z-Koordinate von 0 verschieden ist. Somit ist es möglich, jede Tag-Koordinate in Form eines homogenen-2D-Punktes auszudrücken, was zur Folge hat, dass die 3. Spalte der extrinsischen Matrix entfällt und eine 4×3 Matrix verbleibt. Im Folgenden wird die intrinsische-Kamera-Matrix $\boldsymbol{K}$ und die extrinsche-Matrix $\boldsymbol{E}$ genannt. Zudem bezeichnet $T_k$ die Translation, $R_{i,j}$ die Rotation, $\frac{f_x}{dx}$ die Brennweite unter Berücksichtigung der Sensor-Größe, $H_x$ den Hauptpunkt und $s$ die unbekannte Skalierung. Damit gilt:

$$\begin{pmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{pmatrix} = s\boldsymbol{K}\boldsymbol{E}$$

$$= s \begin{pmatrix} \frac{f_x}{dx} & 0 & H_x & 0 \\ 0 & \frac{f_y}{dy} & H_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} R_{00} & R_{01} & T_x \\ R_{10} & R_{11} & T_y \\ R_{20} & R_{21} & T_z \\ 0 & 0 & 1 \end{pmatrix}$$

Da $\boldsymbol{K}$ nicht vollen Rang hat, kann die oben genannte Gleichung nicht sofort gelöst werden und es ist nötig, zunächst die Matrizen auszumultiplizieren. Danach lässt sich für jede Komponente von H eine eigene Gleichung aufstellen (hier nur die ersten 3 aufgeführt):

$$h_{00} = sR_{00}\frac{f_x}{d_x}; h_{01} = sR_{01}\frac{f_x}{d_x}; h_{02} = sT_x\frac{f_x}{d_x}; ...$$

Die oben genannten Gleichungen liefern jeweils Komponenten der $\boldsymbol{E}$ Matrix, die zunächst bis auf den Skalierungsfaktor $s$ genau bestimmt sind. Da die Spalten einer Rotationsmatrix (orthogonale) Einheitsvektoren sein müssen, lässt sich der Betrag der Skalierung $s$ hieraus in Form des geometrischen Mittels zwischen den beiden berechneten Spalten der Rotationsmatrix ableiten. Das Vorzeichen von $s$ folgt wiederum aus der Annahme, dass das Tag vor der Kamera gesehen wurde und folglich gilt: $T_z < 0$. Die 3. Spalte der Rotationsmatrix folgt wiederrum aus der Bedingung, dass alle 3 Spalten der Rotationsmatrix orthogonal zueinander sind und somit jede Spalte aus dem Kreuzprodukt der anderen beiden Spalten berechnet werden kann. Die oben gennante Berechnung der Homographie, sowie der Extraktion der Rotation daraus stellt nicht sicher, dass die Vektoren der Rotationsmatrix tatsächlich orthogonale Einheitsvektoren sind, weswegen der Autor an dieser Stelle noch eine Polarzerlegung vorschlägt, die eine best mögliche (gemessen an der Frobenius Norm des Fehlers) Rotationsmatrix mit den oben genannten Attributen liefert.

### C. Decodierung der Pixel

Nun da Kandidaten-Tags und deren Homographien bekannt sind, können die korrespondierenden Daten extrahiert werden. Um die Daten innerhalb eines Kandidaten-Tags auszulesen, werden die Positionen, an denen sich ein Bit-Feld befindet, mit Hilfe der Homographie in das Bildkoordinatensystem umgerechnet. Dort werden die korrespondierenden Pixel mit einem Schwellwert verglichen, um zu entscheiden, ob es sich um einen schwarzen oder weißen Pixel handelt. Im Zuge der Kompensation von inhomogener Beleuchtung werden räumlich variierende Schwellwerte verwendet, jeweils Einer für weiß und Einer für schwarz. Die Schwellwerte werden mit Hilfe des Randes eines Tags ermittelt, da dieser bekanntermaßen außen weiß und innen schwarz ist. Es kommt folgendes Intensitätsmodell zur Anwendung:

$$I(x, y) = Ax + Bxy + Cy + D$$

Die Konstanten $A, B, C, D$ werden nun anhand aller Pixel entlang des Randes mit Hilfe eines kleinsten-Fehler-Quadrat Optimierungsverfahren berechnet. Für einen auszulesenen Pixel im Tag wird der Durchschnitt der beiden Intensitätswerte (schwarz/weiß) für die fragliche Position als Schwellwert angenommen.

### D. Codierungs-System

Nachdem nun ein Tag-Kandidat (mit einer geringen falsch-negativ, aber einer hohen falsch-positiv Rate) gefunden und mit Hilfe einer Homographie die frontal-Ansicht berechnet wurde, obliegt es dem Codierungs-System, die falsch-positiv Rate auf ein vertretbares Level zu reduzieren und zugleich die richtige Rotation innerhalb der Bildebene des Tags zu ermitteln. Die Anforderungen an das Codierungs-System sind wie folgt:

- Maximierung der Anzahl unterscheidbarer Codes (auch hinsichtlich der Rotation)
- Maximierung der Anzahl von Bit-Fehlern die entdeckt oder sogar korrigiert werden können
- Minimierung der Rate, mit der Tags untereinander verwechselt oder fälschlicherweise erkannt werden
- Minimierung der totalen Anzahl von Bits pro Tag, was sich positiv auf die Größe eines Tags auswirkt

Da die oben genannten Anforderungen im Gegensatz zueinander stehen, muss dies bezüglich ein Kompromiss gefunden werden. [Ols11] stellt dabei ein auf Lexicodes basierendes Verfahren vor, bei dem der Nutzer verschiedene Parameter

[11] diese bestehen aus den Koordinaten in der gewünschten Dimension zuzüglich eines skalierungs Faktors

[12] diese enthält Rotation und Translation zwischen den Bildebenen und ist Ziel der Berechnung

verändern kann, um das Verfahren an seine Bedürfnisse anzupassen.
Klassische Lexicodes werden für gewöhnlich durch zwei Parameter beschrieben: Durch die Anzahl an Bits $n$ die zur Codierung zur Verfügung stehen und durch den minimalen Hamming-Abstand $d$ zwischen zwei beliebigen Codewörtern. Lexicodes können dabei $\lfloor (d-1/2) \rfloor$ Bit-Fehler korrigieren und $d/2$ Bit-Fehler erkennen. Im Folgenden werden Lexicodes anhand ihrer Eigenschaften verkürzt bezeichnet als $n$ h $d$-Codes (für Codes mit 36 Bit und minimaler Hamming-Distanz von 10 z.B. 36h10). Um den Anwendern die Zeit zur Erzeugung der Tags zu ersparen, stellt der Autor einige Tag-Familien bereit, die durch oben genannte Kurz-Bezeichnung charakterisiert werden.
Ein Lexicode wird, wie der Name impliziert, mit Hilfe einer Heuristik in lexikographischer Ordnung erstellt. Dabei werden alle in Frage kommenden Codewörter aufsteigend sortiert und zum Codebuch hinzugefügt, sofern sie zu allen anderen Codes im Codebuch einen Mindestabstand gemäß der gewünschten Hamming-Distanz haben. [Ols11] verweist an dieser Stelle auf einen Artikel in dem bewiesen wurde, dass dieses Verfahren bereits nahe der Optimalität agiert.
Da die Anordnung der Bits zwei-dimensional ist und auch die Tags beliebig gedreht werden können, müssen die Codewörter folglich auch in allen möglichen Rotationen die minimale Hamming-Distanz gewährleisten. Die übliche Erzeugung von Lexicodes kann diesen Abstand nicht garantieren. Die Erzeugung kann jedoch leicht um eine Prüfung auf alle Rotationen beim Hinzufügen von neuen Codewörtern erweitert werden.
Darüber hinaus muss beachtet werden, dass ungünstige Codewörter im Einsatzgebiet auftreten können. So wäre ein komplett schwarzes Tag (alle Bits sind 0) häufig fälschlicherweise in einer natürlichen Szenerie anzutreffen, dessen Rotation ließe sich zudem nicht feststellen.
Zu wahrscheinliche, primitive Muster werden heuristisch dadurch ausgeschlossen, dass diese sich durch zu wenige Rechtecke bilden lassen. Von [Ols11] durchgeführte Tests zeigen, dass bei zunehmender Anzahl von Rechtecken, die benötigt wird, um ein Tag zu beschreiben, die Wahrscheinlichkeit sinkt, ein solches Muster in einer natürlichen Umgebung zu finden, ohne, dass es sich dabei um ein Tag handelt.
Auch diese Prämisse wird der Erzeugung von neuen Codewörtern hinzugefügt: Ein Codewort wird nur dann zum Codebuch hinzugefügt, wenn die Anzahl der Rechtecke, die zur Erzeugung des Tags notwendig ist, oberhalb eines Schwellwertes liegt.

## III. Wahl geeigneter Parameter und Optimierung

Im Vorfeld wurden bereits einige Parameter angesprochen, für die jedoch keine Standardwerte genannt wurden. Diese werden in dem folgenden Abschnitt zusammengefasst behandelt. Der Detektor gruppiert einzelne Pixel anhand der Richtung und Intensität ihrer jeweiligen Gradienten, wenn diese zu einander ähnlich genug sind. Das Ähnlichkeitsmaß wurde unter (1) definiert, wobei [Ols11] die Werte $K_D = 100$ und $K_M = 1200$ vorschlägt, gleichzeitig aber nahelegt, dass der Gruppierungs-Algorithmus über einen großen Bereich der eben genannten Werte gut funktioniert. Die Kantengewichte des zur Gruppierung notwendigen Graphen werden vorab quantisiert und als Fix-Punkt-Zahlen gespeichert. Dies erlaubt es, die Kanten mit linearem Aufwand zu sortieren, sodass das Gruppieren im Folgenden von dem effizienten "union-find"-Algorithmus durchgeführt werden kann. Dabei werden die oberen und unteren Grenzen der Gradienten-Richtungen und -Intensitäten in einem Array gespeichert, das von einem eine Gruppe repräsentierenden Element indexiert wird.
Der Gaußfilter der Vorverarbeitung wird mit einem Wert $\sigma = 0,8$ vorgeschlagen. Da das Gruppieren der einzelnen Pixel den größten Anteil an der Laufzeit des Detektors ausmacht, besteht die Möglichkeit, diese Berechnung um ein vierfaches zu beschleunigen, indem das Bild auf die Hälfte [13] der Auflösung reduziert wird, was im Zuge der empfohlenen Tiefpass-Filterung keinen nennenswert größeren Aufwand bedeutet, jedoch auch eine Verringerung der maximalen Detektions-Reichweite mit sich bringt.
Im Kontext der Quad-Bildung schlägt [Ols11] bei der Gruppierung von einer Linie zur nächstgelegenen Linie vor, dass eine Linie dann nahe genug bei der Vorherigen liegt, wenn die Lücke dazwischen nicht größer als das Doppelte der Linienlänge (erstere) zuzüglich 5 Pixel ist. Dieser großzügig dimensionierte Schwellwert sorgt dafür, dass auch große Verdeckungen des Randes ignoriert werden können, führt allerdings auch dazu, dass die falsch-positiv Rate potentiell hoch ausfällt.
Bei der Erzeugung des Codebuchs werden solche Codes verworfen, die nur eine geringe geometrische Komplexität besitzen, die also durch eine Kombination von zu wenigen Rechtecken repräsentiert werden können. Hierzu wurde eine Mindestanzahl von 10 Rechtecken empirisch als ein guter Wert ermittelt (vgl. Abschnitt: Empirische Tests).
Während oben genannte Parameter in der Referenz-Implementation[14] von [Ols11] bereits gute Ergebnisse in Vergleichen erzielen, folglich nicht mehr angepasst werden müssen (abgesehen vom möglichen Herunterskalieren des Eingabebilds), ist die Wahl der Tag-Parameter Anwendungsspezifisch und somit eine Entscheidung des Nutzers. Dieser hat dabei die Möglichkeit unter verschiedenen, bereits erzeugten und zur Verfügung gestellten Tag-Familien zu wählen oder gar neue Tags zu berechnen, was jedoch mit einem erheblichen[15] Zeitaufwand verbunden ist. In einem Vergleich zwischen den Tag-Familien 36h10 und 36h15[16] macht der Autor die Abwägung deutlich, die bei der Wahl der Tags gemacht werden muss. So haben 36h10 Tags 2221 voneinander unterscheidbare Muster, während bei der 36h15 Familie nur auf 27 verschiedene Exemplare zurückgegriffen werden kann. Auf der anderen Seite können durch 10 Redundanz-Bits nur 4 Bit-Fehler korrigiert werden, wogegen 15 redundante Bits eine Korrektur von bis zu 7 Bit-Fehlern ermöglichen. Es muss im Vorfeld also abgewogen werden, wie viele verschiedene Muster benötigt werden und wie stark die Sicht auf diese gestört sein wird. Auch die maximale Distanz der Tags zur Kamera muss bei der Wahl berücksichtigt werden, da, bei gleicher Fläche der Tags, die Tag-Pixel bei steigender Anzahl kleiner werden. Dies hat zu Folge, dass Tag-Familien mit wenigen Pixeln (z.B. 16) über eine größere Distanz funktionieren als Solche mit vielen Pixeln (z.B. 36). Der Autor[Ols11] hat in Tests herausgefunden, dass die

[13] jeweils Höhe und Breite
[14] erhältlich unter: https://april.eecs.umich.edu/wiki/index.php/AprilTags
[15] [Ols11] gibt dazu eine Rechenzeit in der Größenordnung von Tagen an
[16] zur Erinnerung: Anzahl Bits = n=36; 10 und 15 jeweils Umfang der Redundanz

Distanz, in der 16-Bit-Tags wahrgenommen werden können, um 25% größer ist, als die Distanz unter der 36-Bit-Tags erkannt werden können.
Beim Auslesen eines Tag-Inhalts wird dessen Code mit jedem gültigen Code im Codebuch verglichen und jeweils die Hammingdistanz berechnet. Die geringste gefundene Distanz wird mit einem vom Nutzer definierten Schwellwert verglichen und anhand Dessen weitergeleitet oder verworfen. Somit hat der Nutzer direkt Einfluss auf die Balance zwischen falsch-negativ- und falsch-positiv- Rate. Der Decodiervorgang wächst linear mit der Größe des Codebuches, was im Vergleich zu der Komplexität der Bildverarbeitungsschritte zu vernachlässigen ist.

## IV. Empirische Tests und Vergleiche mit anderen Verfahren

Zur Evaluation des vorgeschlagenen Verfahrens wurde eine Bilder-Sammlung aus dem LabelMe Datensatz[RTMF07] verwendet, die 180.829 Bilder aus unterschiedlichen Bereichen innerhalb und außerhalb von Gebäuden enthält. Dieser Datensatz ist überaus gut geeignet, um die falsch-positiv Rate des Verfahrens zu untersuchen, da keines der verwendeten Bilder die hier vorgestellten, künstlichen Tags enthält.
Die ersten Tests untersuchen dabei die These, dass Muster, die sich aus einer zunehmenden Anzahl von Rechtecken ausdrücken lassen, eine abnehmende falsch-positiv Rate haben. Mit Hilfe einiger 25h9 Tag-Familien mit Komplexitäten (Anzahl Rechtecken) von 2 bis 10 wurde, im Rahmen des LabelMe Datensatzes, ein geeigneter Wert für die Mindestanzahl von Rechtecken, aus denen ein Muster mit geringer falsch-positiv Rate erzeugt werden kann, abgeleitet. Das Ergebnis besagten Tests ist in Abbildung 3 ersichtlich. Dabei fällt auf, dass eine zunehmende Komplexität der Tag-Familie die falsch-positiv Rate sukzessive senkt. Ab Komplexitäten von 9 aufwärts, ist es sogar möglich, eine von [Ols11] berechnete, theoretische falsch-positiv Rate zu unterbieten[17]. Eine Anhand dieser Tests ermittelte Komplexität von 10 wird von [Ols11] bevorzugt und findet entsprechend in den folgenden Tests Anwendung.

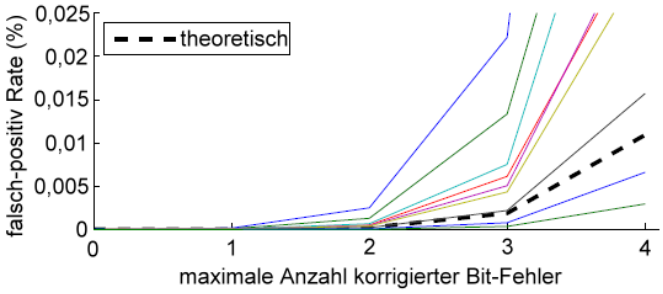


Abbildung 3. Darstellung für 25h9 Tag Familien mit unterschiedlicher Komplexität von 2 bis 10. Komplexität 2 entspricht der (steileren) blauen Linie, während die weiteren Komplexitäten sukzessive in der Fehlerwahrscheinlichkeit pro korrigiertem Bit-Fehler sinken. Komplexitäten 9 und 10 liegen dabei sogar unter einer von [Ols11] berechneten Wahrscheinlichkeit für natürlich vorkommende Muster mit zufälligem Bitmuster. Übersetzt aus [Ols11].

An dieser Stelle werden erstmals Vergleiche zu anderen Systemen angebracht, die mit Hilfe von künstlichen Landmarken Positionsbestimmung realisieren. Dabei werden die hier vorgestellten AprilTags mit den 1999 vorgestellten ARToolkit, ARTags (2003) und ARToolkit Plus (2005) verglichen. Da ARToolkit und ARTag nicht Quell-offen sind, lassen sich diese kaum anhand ihres Aufbaus vergleichen und können somit nur in ihren Ergebnissen (und offensichtlichen technischen Aspekten, wie der Tag-Beschaffenheit) gegenüber gestellt werden.
Stellt man die Marker der unterschiedlichen Verfahren gegenüber, so kann festgestellt werden, dass alle Verfahren auf quadratische Objekte mit schwarzer und weißer Farbe zurückgreifen.
ARToolkit verwendet jedoch keine direkte, binäre Codierung sondern stellt Symbole, wie z.B. den lateinischen Buchstaben A dar. Zur Decodierung dieser Symbole muss das Muster eines gefundenen Markers mit einem Codebuch verglichen werden, wobei das Symbol aus dem Buch gewählt wird, das die höchste Korrelation zum gefundenen Zeichen inne hat. Der oben gennante Vorgang der Decodierung ist, in Abhängigkeit von der Größe des Codebuchs, potentiell langsamer als das im Rahmen von AprilTag vorgestellte Verfahren. Auch die Erzeugung möglichst vieler, gut unterscheidbarer (orthogonaler) Codes erweist sich bei Symbolen als schwieriger, da diese nicht lexikografisch aufgezählt werden können. Die Auswertung des Kamerabildes ist bei ARToolkit störanfälliger (gegen variable Beleuchtung) aber schneller als beim AprilTag-System, da bei Ersterem das Eingabebild lediglich anhand eines Nutzer-definierten Schwellwertes binarisiert wird. Zudem unterstützt ARToolkit laut [Ols11] keinerlei Verdeckung des Randes.
ARTags verwenden dagegen binärcodierte Marker (mit orthogonaleren Codes als ARToolkit), die, Dank Vorwärts-Fehlerkorrektur, auch dann noch erkannt werden können, wenn sie teilweise verdeckt sind. ARTags verwenden zur Detektion der Tags ein Gradienten-Verfahren, über dessen genaue Beschaffenheit jedoch nicht viel bekannt ist. ARTag verbietet bei der Erzeugung von Markern 2 mögliche Codes (Alle Bits 0 oder alle 1), da diese zu wahrscheinlich in der Realität sind.
Im Folgenden ist ein Vergleich der Marker der unterschiedlichen Verfahren abgebildet (aus [Ols11]). Der Vorteil der AprilTags besteht in der deutlich größeren min. Hammingdistanz zwischen den Codewörtern eines Codebuchs bei ähnlichen oder teilweise sogar größeren Codebüchern. Dies spiegelt sich in der niedrigeren Verwechselungsrate zwischen verschiedenen Markern wieder und unterstützt zudem die Decodierung von großflächigerer Verdeckung der Tags, verglichen mit den anderen genannten Verfahren.

| Verfahren | Länge | # Codes | Min. Hamming |
|---|---|---|---|
| ARToolkit+ (simple) | 36 | 512 | 4 |
| ARToolkit+ (BCH) | 36 | 4096 | 2 |
| ARTag | 36 | 2046 | 4 |
| AprilTag(36h9) | 36 | 4164 | 9 |
| AprilTag(36h10) | 36 | 2221 | 10 |

Der nächste Test (Abbildung 4) vergleicht alle genannten Verfahren anhand ihrer falsch-positiv Rate im Rahmen des LabelMe Datensatzes (wie schon in Abb. 3 zu Ermittlung einer geeigneten Komplexität). Hier haben Codes mit weniger umfangreichen Codebüchern einen Vorteil, da nur wenige

[17] Da die von [Ols11] berechneten Werte im Rahmen dieser Arbeit jedoch nicht reproduziert werden konnten, werden diese hier nicht explizit aufgeführt.

Codes existieren, mit denen eine natürliche Szene verwechselt werden kann. AprilTag 36h15 ist hierbei das einzige Schema, das kein Mal fälschlicherweise einen Marker findet, obwohl keiner vorhanden ist (das Schema enthält dafür auch nur 27 unterscheidbare Codes). Das nächst (geringfügig) schlechtere Ergebnis wird von ARToolkit Plus mit einfachem Schema erzielt (512 Codes). AprilTag 36h10 (2221 Codes) folgt mit einem etwas größeren Abstand, dabei ist jedoch zu beachten, dass das Codebuch von AprilTag 36h10 4 Mal mehr unterscheidbare Codes leifert, als das des ARToolkits (einfach). ARToolkit (BCH) fällt bereits bei 2 korrigierten Bit-Fehlern aus, was primär daran liegt, dass die minimale Hammingdistanz 2 ist und bei 2 korrigierten Bitfehlern folglich nicht immer eine eindeutige Fehlerkorrektur möglich ist. Die anderen Verfahren (ARTag und ARToolkit Plus standard) fallen dagegen erst bei 4 korrigierten Bit-Fehlern auf, was daran liegen dürfte, dass die minimale Hammingdistanz mit 4 in diesem Fall auch nicht ausreicht. Lediglich das Ergebnis von AprilTag 36h10 verwundert, da dieses mit 10 Redundanz-Bits 4 Bit-Fehler gänzlich korrigieren können sollte. Zur

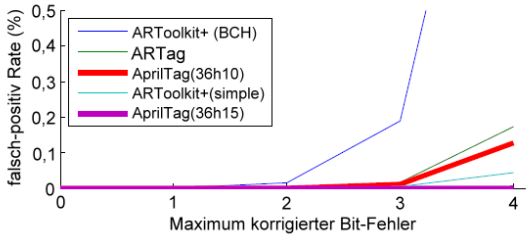


Abbildung 4. Ergebnisse der verschiedenen Verfahren im Rahmen des LabelMe Datensatzes. Beim direkten Vergleich von Verfahren mit gleich großer Anzahl Codes schneiden die AprilTags deutlich besser ab. Die Korrektur von 4 Bit-Fehlern macht AprilTag 36h10 größere Probleme, als man angesichts von 10 Paritäts-Bits erwarten würde. Grafik übersetzt aus [Ols11].

Ermittlung der Präzision der einzelnen Verfahren wurden synthetische Bilder mit Hilfe von Raytracing konstruiert, die als Ground-Truth dienen. Diese Bilder weisen gültige Tags, Reflexionen und ein Schachbrett-Muster auf. Besagter Tests wurd zweigeteilt: einerseits mit veriierender Distanz des Tags von der Kamera (Abbildung 5), andererseits mit veriierender Ausrichtung[18] (Abbildung 6).

## V. Fazit

AprilTags sind ein quelloffenes System, mit dessen Hilfe künstliche Landmarken erzeugt und in einem Kamerabild gefunden werden können. Dabei kann die Pose des Tags im Kamerabild anhand von Größeninformationen des Tags und der intrinsischen Kameraparameter berechnet werden. Das vorgestellte Verfahren kann sich in Punkto Entdeckungs-Wahrscheinlichkeit von tatsächlichen Tags, Präzision (sowohl in der Distanz als auch im Winkel) und falsch-positiv Rate (natürliche Muster werden mit einem Tag verwechselt) von älteren, konkurierenden Systemen absetzen. Die Beispielimplementation in Java lieferte dabei eine Wiederholrate von 30 Bildern pro Sekunde für Bilder in VGA-Auflösung. Die später folgende C-Implementation dürfte diese Leistung noch

[18] gemessen wird der Winkel zwischen Tag-Normale und Blickrichtung der Kamera

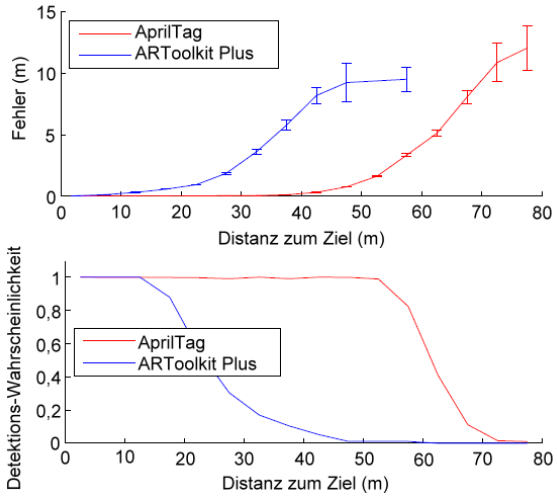


Abbildung 5. Vergleich eines unbekannten AprilTag Schemas mit ARToolkit Plus. Die Messung erfolgte mit synthetischen Bildern bei unterschiedlicher Distanz aber gleicher Rotation des Tags. Grafik übersetzt aus [Ols11].

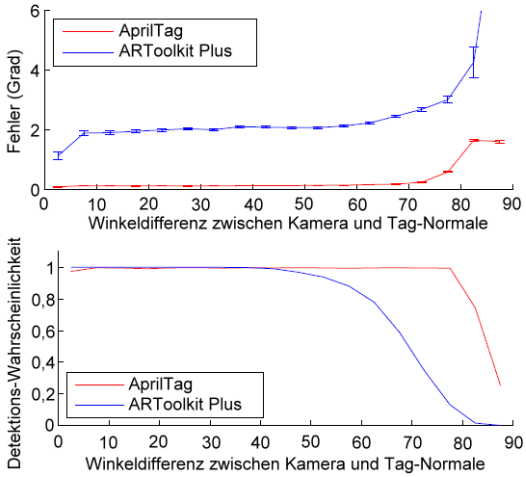


Abbildung 6. Vergleich eines unbekannten AprilTag Schemas mit ARToolkit Plus. Die Messung erfolgte mit synthetischen Bildern bei unterschiedlichen Winkeln aber gleicher Distanz des Tags. Grafik übersetzt aus [Ols11].

deutlich steigern. Vorgefertige Tag-Familien sowie der offene Quellcode mit Dokumentation sollen eine kurze Einarbeitung und schnelle Verwendung des Systems ermöglichen.

## Literatur


[HZ04] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[Ols11] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[RSO13] Andrew Richardson, Johannes Strom, and Edwin Olson. AprilCal: Assisted and repeatable camera calibration. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, November 2013.

[RTMF07] Bryan C. Russell, Antonio Torralba, Kevin P. Murphy, and William T. Freeman. Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1):157–173, 2007.