Verfolgung von Ballflugbahnen mit einem frei beweglichen Kamera-Inertialsensor

Jörg Kurlbaum

Bremen, 15. März 2007

Diplomarbeit im Fachbereich Informatik
Universität Bremen
unter der Betreuung von Dr. ing. Udo Frese

Gutachter: Dr. ing. Udo Frese
Zweitgutachter: Prof. Dr. rer. hum. biol. Kerstin Schill

Danksagung

Ich möchte mich ganz ausdrücklich bei Udo Frese für den Vorschlag des interessanten Themas sowie die fantastische Betreuung in allen Bereichen und während sämtlichen Abschnitten meiner Arbeit bedanken. Die Beantwortung aller Fragen zu jederzeit, die kleinen Privatvorlesungen, wenn ich mal etwas nicht verstanden habe und die Stunden, die wir gemeinsam vor dem Quellcode auf der Suche nach Fehlern gesessen haben, sind selbst für die engagiertesten Betreuer keine Selbstverständlichkeit. Vielen Dank Udo!
Desweiteren möchte ich mich bei Kerstin Schill für die spontane Zusage zur Zweitgutachterin bedanken.
Außerdem sollen meine beiden Helfer bei den Experimenten nicht unerwähnt bleiben. Ohne euch hätte ich keine Daten zum Verarbeiten bekommen und wäre wahrscheinlich an der Durchführung von Experimenten verzweifelt.

Erklärung

Hiermit versichere ich, Jörg Kurlbaum, diese Diplomarbeit ohne fremde Hilfe angefertigt zu haben. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, sind als solche kenntlich gemacht.

Unterschrift _____

Inhaltsverzeichnis

Kapitel 1

Einleitung

Im Rahmen dieser Diplomarbeit wird ein *frei bewegliches* Kamera-Inertialsensor System entwickelt, welches in der Lage ist, die Flugbahn eines (Fuß)balls im Rahmen einer abgegrenzten und definierten Umgebung zu verfolgen und vorherzusagen, obwohl sich die Kamera im Raum frei bewegt. Dieses System könnte zum Beispiel an einem Helm einer Person oder an einem Roboter montiert sein.

1.1 Einführung und Motivation

> By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

So lautet die Vision der Erfinder des RoboCup Hiroaki Kitano und Minoru Asada [1] [2] [3]. Jedes Jahr stellen Forscher in diesem Bereich ihre Entwicklungen während des RoboCup WorldCup vor und vergleichen ihre Ansätze in Form sportlicher Wettkämpfe. Es werden Simulationen und echte Roboter benutzt darunter seit wenigen Jahren auch vermehrt humanoide Roboter.
Die mechanische Wirklichkeit ist noch weit von der Vision Fußball spielender Roboter auf weltklassigem Niveau entfernt, während die Perzeption ihrer Umgebung, Interaktion mit anderen Robotern und weitere Methoden der autonomen Agenten bereits heute gut entwickelt sind.
Die Aufgaben eines Fußball spielenden Roboters sind vielfältig, neben der Beherrschung des eigenen Körpers, müssen Bälle, Teampartner, Gegner und die eigene Position ermittelt werden. Hinter all diesen Problemen steht methodisch ein Ziel: *Die Schätzung einer Zustandsvariable durch Auswertung unsicherer Sensormessungen*.
In dieser Arbeit sollen zukünftige Szenarien vorweg gegriffen werden, um zu zeigen, dass einzelne Komponenten, wie Bildverarbeitung und Tracking-Algorithmen, derzeit

schon sehr gute Ergebnisse liefern und bereits der notwendigen Leistung eines RoboCup-Spielers von 2050 nahe kommen. Dazu wird ein System entwickelt, welches sich in einer Umgebung lokalisieren kann und Ballflugbahnen verfolgt und vorhersagt. Zum Einsatz kommt eine Kamera als *Auge*, deren Position und Lage im Raum durch einen Inertial-Sensor bestimmt wird. Ein solcher Sensor ersetzt für Roboter das *Gleichgewichtsorgan* des Menschen, mit dem ein Bezug von der *Perzeptionsebene* (Auge/Kamera) zur globalen Umgebung hergestellt werden kann.

1.1.1 Objektverfolgung im RoboCup

Der RoboCup ist ein Anwendungsszenario, in dem die Forschungen zu autonomen Robotern, künstlicher Intelligenz, Bildverarbeitung und anderer Gebiete zusammenfließen und im sportlichen Wettkampf gegeneinander getestet und verglichen werden können. Da es im RoboCup um Fussball spielende Roboter geht, müssen auch Bälle und deren Flugbahn verfolgt und geschätzt werden. Besonders die sich mit Beinen fortbewegenden Roboter werden hier vor besondere Aufgaben gestellt. Sie müssen sowohl sich selbst in ihrer Umgebung lokalisieren, als auch Positionen der Objekte um sie herum ermitteln. Die Schätzungen des Zustandes erfolgen zumeist mit verschiedenen Filtermethoden, neben den klassischen Verfahren wie Kalman-Filter sind im RoboCup auch die relativ neuen Particle-Filter [4] [5] sehr beliebt. Sie sind einfach zu implementieren und kommen gut mit den Besonderheiten im RoboCup zurecht (*Kidnapped Robot*[1], abprallende Bälle, ...). Bei der Bewegung über das Spielfeld verändert sich durch Drehung des Kopfes oder durch Veränderungen der Gelenkwinkel ständig die relative Position des Hauptsensors - der Kamera - zum Boden. In der *Four-Legged League* wird die Position der Kamera zumeist nicht mit Filtermethoden geschätzt, sondern mit einer Transformationsmatrix berechnet, die sich durch die inverse Kinematik aus den Gelenkwinkeln des Roboters ergibt [6] (Abb. 1.1). In der Humanoid-League wird dieser Ansatz ebenfalls realisiert, jedoch u.a. durch die Verwendung günstiger Eigenentwicklungen, die nicht so genaue Informationen über die Stellung der Gelenke liefern können, erschwert. Wenn diese Maschinen beginnen zu laufen, also zeitweise den Kontakt zum Boden verlieren, und so eine inverse Kinematik unmöglich ist, ist es notwendig eine Schätzung der Kameraposition auf anderem Wege zu ermitteln. Ein solches System könnte dann *frei beweglich* sein und zum Beispiel aus einer Kombination von Kamera und Intertial-Sensor bestehen.

[1] Das Kidnapped Robot Problem entsteht, wenn ein lokalisierter Roboter in seiner Position verschoben oder umgesetzt wird, zum Beispiel im RoboCup durch den Schiedsrichter, ohne jedoch irgendeine Aktion ausgeführt zu haben. Je nach Art der Lokalisierung kann das zu einen Totalausfall der Lokalisierung führen. Bei den Particle-Filtern werden für gewöhnlich immer einige Partikel zufällig gestreut, so dass auch eigentlich unmögliche Schätzungen Einfluß auf das System haben können.

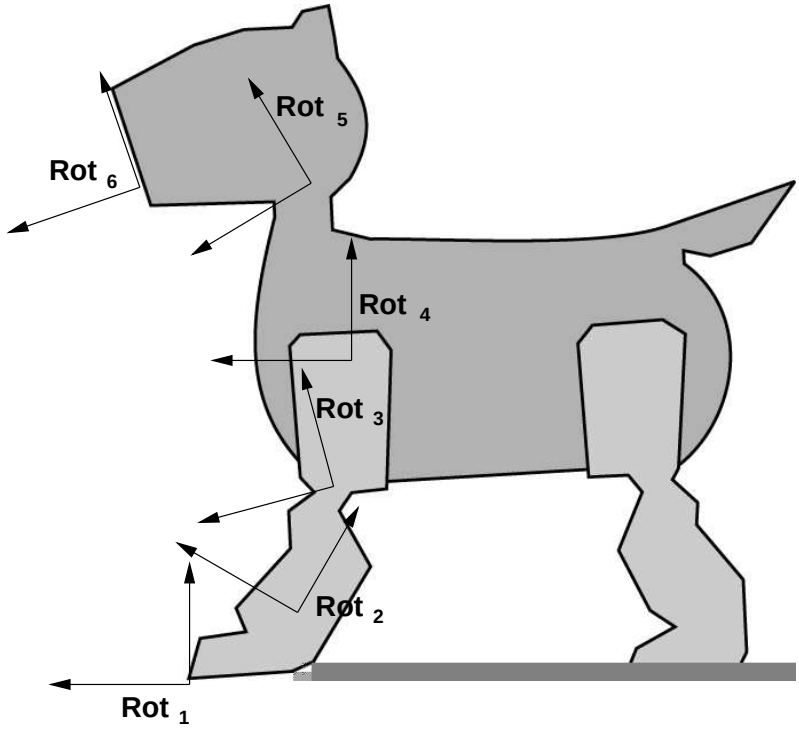


Abbildung 1.1: Inverse Kinematik am Beispiel des Aibo-Roboters der Firma Sony, für die beim RoboCup eine eigene Liga existiert.

1.2 Gliederung der Arbeit

Im weiteren Teil der *Einleitung* werden zunächst die in dem System eingesetzten Sensoren, ihre Eigenschaften und bisherige Verwendung vorgestellt. Danach werden bereits getätigte Ansätze zur Kombination von Kameras und Inertial-Sensoren kurz erläutert und schließlich kurz das Ziel und der Beitrag dieser Arbeit beschrieben.

Im Kapitel *Theorie* werden die mathematischen Grundlagen beschrieben. Es wird kurz auf verschiedene Arten der Darstellungen von Positionen und Orientierungen im dreidimensionalen Raum eingegangen. Schließlich werden vielfach verwendete Tracking-Algorithmen vorgestellt und eine mathematisch definierte Beschreibung des vorliegenden Problems entwickelt. Dazu gehört eine Freiheitsgraddiskussion, sowie Besonderheiten und notwendige Anpassungen des Filters.

Das Kapitel *Implementierung* beschreibt im Anschluß die Umsetzung der Theorie in ein Gesamtsystem und begründet Entscheidungen für eine bestimmte Wahl der Umsetzung. Es wird auch ein Überblick über die verwendeten Software Komponenten gegeben.

Schließlich werden im Kapitel *Ergebnisse* die Resultate der Arbeit vorgestellt und diskutiert.

1.3 Sensoren und ihre Eigenschaften

Das Verfolgen einer Ballflugbahn mit einer frei beweglichen Kamera (oder einem erweiterten Kamerasystem) kann zunächst in zwei Teile zerlegt werden.

Lokalisierung der Kamera

Die Kamera muss in Bezug auf ein globales Koordinatensystem (Weltkoordinatensystem) lokalisiert werden. Nur so können Daten aus dem Kamerabild einem Objekt in der *Welt* zugeordnet werden. Es ist möglich, mit Landmarken eine vollständige Lokalisation der Kamera vorzunehmen, dafür ist es aber nötig jederzeit ausreichend viele Landmarken im Bild zu haben. Dieser Umstand wird in Abschnitt 2.3.2 genauer beschrieben.

Schätzen des Ballzustands und seiner Flugbahn

Der Ball wird ausschließlich im Kamerabild erkannt, seine theoretische Flugbahn ist aber über ein einfaches physkalisches Modell bekannt. Für eine robuste Schätzung seiner Position und Flugbahn können verschiedene Methoden eingesetzt werden. Algorithmen aus der Familie der Kalman-Filter bieten sich wegen ihrer Flexibilität an und sind im Bereich der Robotik neben Particle-Filtern die wohl am häufigsten eingesetzten Schätzsysteme.

1.3.1 Inertial-Sensor

Soll eine Flugbahn durch Nachführen eines Kamerasystems verfolgt werden, kann nicht mehr garantiert werden, dass sich im Bild ausreichend Informationen zur Lokalisierung der Kamera finden. Daher soll ein weiterer Sensor mit der Kamera kombiniert werden. Zur Verfügung steht ein Orientierungs- und Positionstracker (Inertial-Sensor oder *Inertial Measurement Unit* (IMU)) der Firma Xsens (Abb. 1.2). Diese Art von Sensoren liefern Informationen über sechs Freiheitsgrade der Bewegung des Trackers und somit auch des Körper mit dem sie verbunden sind. Sie ähneln damit in der Funktion den Kreiselkompassen in Flugzeugen, ihre Implementierung ist jedoch unterschiedlich und auch preislich sind diese Sensoren im Vergleich eher günstig.
Das vorliegende Gerät bietet die Möglichkeit folgende Messwerte in seinem Körperkoordinatensystem aufzunehmen und auszulesen:

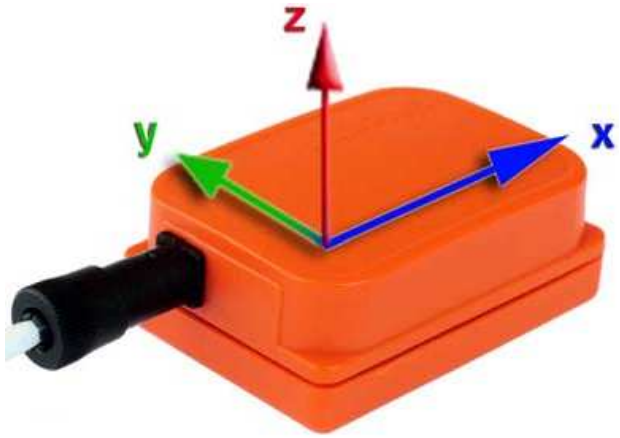


Abbildung 1.2: Inertial-Sensor (MTx) der Firma Xsens. Dieses Modell wurde für die vorliegende Arbeit verwendet.

Accelerometer	
Beschleunigung in x, y, z	in $\frac{m}{s^2}$
Gyroskope	
Winkelgeschwindigkeit $\omega_x, \omega_y, \omega_z$	in $\frac{rad}{s}$
Magnetfeld	
Richtung des Magnetfelds	(dimensionslos) einfacher Vektor

Die IMU ist auch eigenständig in der Lage mit integrierten Algorithmen eine Orientierung zu schätzen. Für die hier vorliegende Anwendung sollen jedoch die „rohen“ aber bereits kalibrierten (temperatur-kompensiert) Dynamikinformationen verarbeitet werden.
Solche Geräte werden bislang häufig für Virtual-Reality-Anwendungen eingesetzt, bei denen aus einer echten Umgebung Bewegungen in die virtuelle computergenerierte Welt umgesetzt werden. Eng damit verbunden ist zum Beispiel auch die Bewegungsanalyse - ähnlich dem Motion-Capturing - für den Hochleistungssport. Die Firma Xsens hat ihre Produkte dafür bereits eingesetzt und auch durch Veröffentlichungen wissenschaftlich fundiert [7]. In Abbildung 1.3 findet sich ein Beispiel für die Anwendung der Tracker bei der Bewegungsanalyse, die Xsens durchgeführt hat (aus [8]).

1.3.2 Kamera

Für die Experimente steht eine digitale Kamera Sony DFW-VL500 mit einer Bildrate von 30Hz und zusätzlichem Weitwinkelobjektiv zur Verfügung. Es werden die Bilder im YUV 4:2:2 Format (siehe auch [9]) über den Firewire-Bus an den Computer übermittelt. Wie aus diesen Bilder für die Verarbeitung geeignete Daten gewonnen werden, kann in Ka-

Abbildung 1.3: Bewegungsanalyse mit Orientierungstrackern (aus [8]). An relevanten Stellen werden am Anzug des Schlittschuhfahrers Orientierungstracker befestigt. Wie bei *Virtual-Reality* Anwendungen wird jede Gliedmaße in ihrer Richtung (Lage im Raum) observiert. Mit einem darübergelegten Modell der Gelenke und des Körpers lassen sich die Bewegungen in den Computer übertragen und analysieren, um beispielsweise den Ablauf der Bewegung zu optimieren oder Unregelmäßigkeiten zu erkennen.

pitel 3 nachgelesen werden.

1.3.3 Kombination von Kamera und IMU

Beide Geräte werden zu einer Einheit (Abb. 1.4) kombiniert. Erste Ansätze zur Kombination von visuellen (Kameras) und günstigen Inertial-Sensoren wurden auf dem InerVis-Workshop der ICRA (International Conference on Robotics and Automation) im Jahr 2004 vorgestellt [10][11][12]. Viele Ansätze beschäftigten sich mit der Kalibrierung eines solches kombinierten Systems, mit der die Verschiebungen von Kamera-Achsen und Drehachsen der Gyro-Sensoren ermittelt werden können. Ein zu der vorliegenden Arbeit ähnliches Ziel wurde in [13] und ebenfalls auf dem Workshop vorgestellt: Eine Kamera wird mit einem Inertial-Sensor verbunden an einem Roboter-Arm befestigt. Das System ist in der Lage, die Position des TCP (Tool Center Points) aus den Informationen der Kamera und des Inertial-Sensors zu bestimmen.

Abbildung 1.4: Die Kombination der verwendeten Sensoren im experimentellen Aufbau

1.4 Begriffsklärung: Tracking

Das aus dem Englischen stammende Wort *Tracking*, welches hier häufig eingesezt wird, besteht im Wortstamm aus dem Wort *track* (Pfad) und wird hier als Verb gebraucht. Es beschreibt in dieser Funktion die Verfolgung einer Bahn eines Objektes. Dabei bezieht sich Tracking in der Informatik zumeist auf die Analyse von Bewegungssequenzen, die beispielsweise aus Bildern gewonnen werden. Tracking kann jedoch Sequenzen von beliebigen Sensoren analysieren. Damit ist auch immer eine Vorhersage verbunden, die beschreibt, wo sich das verfolgte Objekt im nächsten Schritt befinden wird. Bei visueller Sensoreingabe über eine Kamera kann so der Bereich bestimmt werden, in dem ein Objekt im nächsten Schritt zu suchen ist und auf diese Weise die Dauer der Informationsverarbeitung reduziert werden. Im vorliegenden Fall geht es ebenfalls um die Verfolgung eines Objektes, jedoch nur um Positionsbestimmung und Vorhersage ohne direkte Rückkopplung auf die Sensoreingaben.
Allgemein lässt sich Tracking und die Funktion von Tracking Algorithmen in drei Verarbeitungsschritte unterteilen [14]:

Prädiktion Vorhersage von Lage und/oder Position des zu verfolgenden Objektes auf der Basis bekannter physikalischer Gesetzmäßigkeiten für einen Zeitschritt.

Assoziation Assoziiert Messungen mit prädizierten Objekten. Insbesondere bei mehreren Objekten muss eine Verbindung zwischen Messung und Objekt bestimmt werden.

Innovation Bestimmung des Zustands eines Objekt aus der Fusion der Messung und des prädizierten Zustandes. Es werden beide Ergebnisse nach ihrer Gewichtung

zusammengeführt. Dabei hat die Gewichtung Einfluss auf das Verhalten des Filters. Ein Filter der mehr Gewicht auf die Prädiktion legt, *glättet* die Funktion. Eine stärkere Gewichtung der Messung läßt den Filter schneller auf Änderungen in den Messwerten reagieren.

1.5 Ziele

Im Rahmen dieser Arbeit soll mit Hilfe der Kombination des oben vorgestellten Inertial-Sensors und einer Kamera ein Tracking-System entwickelt werden, welches die Flugbahn von Bällen verfolgen und vorhersagen kann. Dabei soll das System zunächst vom Benutzer *von Hand* bewegt und so der ungefähren Flugbahn eines geworfenen Balls hinterhergeführt werden. Es soll ein algorithmischer Filter entwickelt werden, der sowohl den Zustand des Balls, als auch den des Systems selbst nach Möglichkeit in Echtzeit berechnet. Nach einer kurzen Laufzeit soll es außerdem möglich sein, eine Flugbahn und einen Auftreffpunkt des Balls (relativ oder global) zu bestimmen. Zu einem solchen System gehört eine auf das Problem zugeschnittene Bildverarbeitung, die im Rahmen dieser Arbeit ebenfalls entwickelt wird, sowie eine für Tests geeignete Simulationsumgebung. Zum praktischen Teil der Arbeit gehören zusätzlich die Ansteuerung des Sensors und der Kamera sowie die Durchführung von Experimenten.

1.6 Eigener Beitrag der Arbeit

Die Verfahren zur Schätzung von Zuständen, wie Kalman Filter und seine Derivate Extended Kalman Filter (EKF) und Unscented Kalman Filter (UKF) sind bereits bekannt, auch wenn der UKF eine noch junge Methode in der Robotik ist. Ebenso sind Verfahren zur schnellen Objekterkennung in der Bildverarbeitung gerade im Bereich der Echtzeitbildverarbeitung bereits erforscht. Relativ neu ist hingegen die Kombination von Kamera und Inertial-Sensor. Jedoch soll der Beitrag dieser Arbeit nicht nur ausschließlich aus der Implementierung eines Filter-Systems, sondern vor allem in der Anpassung des UKF für das Schätzen von Zustandsräumen mit nicht vektorieller Darstellungform der Orientierung eines Körpers im Raum bestehen. So wird auch im Theorieteil dieser Arbeit ausführlich auf die notwendigen Anpassungen eingegangen und eine generelle Lösung für Mannigfaltigkeiten in Zuständen vorgestellt. Die funktionsfähige Implementierung des Filters ist jedoch auch neben aller Theorie das Ziel der Arbeit. Das Gesamtsystem ist annähernd vollständig im Rahmen der Arbeit entwickelt worden und die Durchführung von Experimenten zeigt, dass das System nicht nur in der Theorie seine Aufgabe erfüllt.

1.7 Experimenteller Aufbau

Da die Experimente schließlich die Funktion des Systems zeigen sollen und der Aufbau auch ganz maßgeblich zur Vorstellung der Funktionsweise des Filter-Systems beiträgt, soll hier ein Eindruck (Abb. 1.5) vermittelt werden, wie die Experimente durchgeführt wurden.

In einem Raum werden an definierten Stellen Landmarken aufgestellt. Ein oranger Fuß-

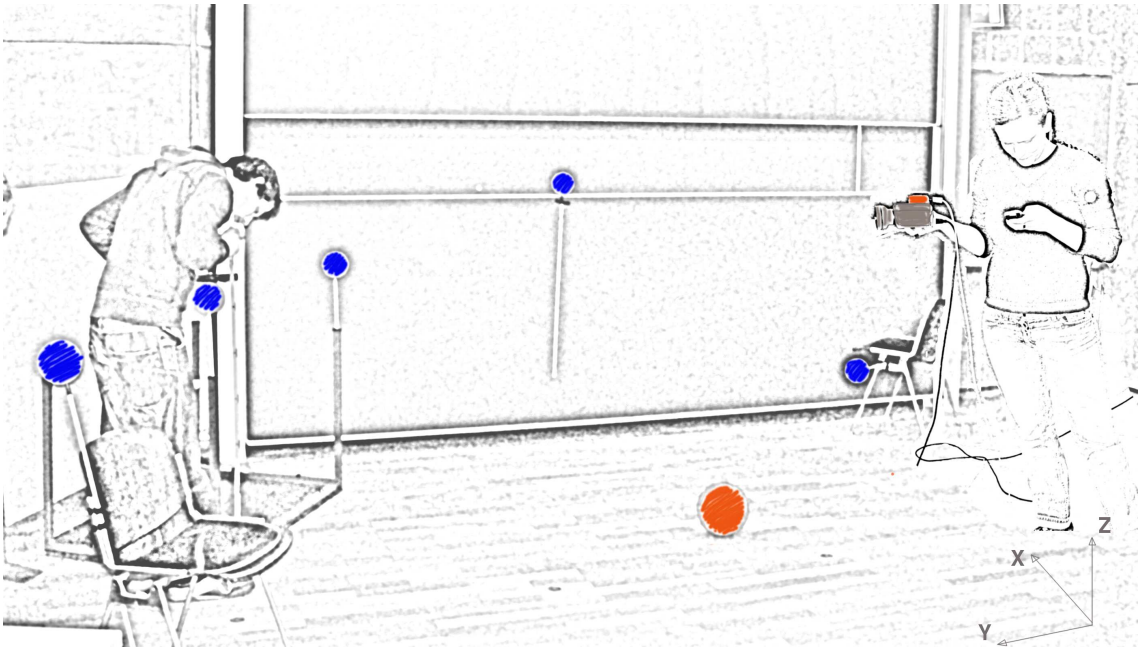


Abbildung 1.5: Eindruck der Szene im Experiment

ball wird im Sichtbereich der Marken durch den Raum geworfen. Die Kamera mit montiertem Inertial-Sensor wird von einer Person so dem Ball nachgeführt, dass nach Möglichkeit der Ball im Zentrum des Bildes zu sehen ist. Diese Daten werden vom Rechner verarbeitet und bei Bedarf aufgezeichnet. Daraus wird schließlich die Flugbahn und der Auftreffpunkt berechnet und am Rechner visualisiert.

Kapitel 2

Theorie

Das Verfolgen von Objekten in einem System setzt eine mathematische Darstellung der Umgebung und der Objekte selber voraus. Für die Position und Orientierung des visuell-inertial Systems im dreidimensionalen Raum muss eine konsistente und handliche Form der Darstellung gefunden werden. Etwas einfacher gestaltet sich die Position von Ball und vorhandenen Marken, die durch ihre Kugelform mit einfachen Vektoren dargestellt werden können. Im Folgenden werden verschiedene mögliche Formen für Rotationen und Orientierungen vorgestellt.

2.1 Darstellung von Positionen, Rotationen und Orientierungen

Im Wesentlichen werden für das vorhandene Szenario folgende Koordinatensysteme verwendet, die zueinander verschoben und verdreht sein können: Im festen Weltkoordinatensystem sollen die Objekte mit Hilfe der Algorithmen verfolgt werden, während im beweglichen Koordinatensystem der Kamera, sowie im fest dazu verschobenen Koordinatensystem des Inertial-Sensors die Messungen stattfinden. Zusätzlich findet die Messung der Kamera (Bildverarbeitung) in einem zwei-dimensionalen Bildkoordinatensystem statt, welches aber durch eine spezielle Abbildungsgleichung in das Kamera-Koordinatensystem übertragen werden kann. Für die Algorithmen ist es wichtig, den Objekten eine Position und Orientierung zu geben, deren Darstellung zudem einfach zwischen den Koordinatensystemen verschoben werden kann. Während die Position in allen Fällen durch einfache Vektoren und deren Arithmetik dargestellt werden können, bestehen für die Orientierung und relative Rotationen mehrere Möglichkeiten der Darstellung, die in den folgenden Absätzen beschrieben werden.

2.1.1 Matrix

Eine Orientierung eines Körpers kann auch mit einer 3×3 Matrix dargestellt werden. Eine solche Rotationsmatrix rechnet einen Richtungsvektor von Körperkoordinaten in Weltkoordinaten um. Die Rotationsmatrix beschreibt drei Freiheitsgerade mit neun Zahlen. Es ist also mit jeder Matrix nötig sechs Randbedingungen zu erfüllen.
Ein großer Vorteil der Matrix-Form ist die Möglichkeit mit einfachen Matrixoperationen Vektoren von einem Referenzrahmen in den anderen zu transformieren. Zudem ist die Darstellung von Orientierungen und Rotationen *singularitätsfrei*. Allerdings muss eine Matrix nach vielen Operationen, wegen der numerischen Ungenauigkeit in der digitalen Darstellung von Fließkommazahlen normalisiert werden, da eine nicht-orthonormale Rotationsmatrix neben einer Rotation auch Scherphänomäne darstellt. Bei der Normalisierung müssen alle sechs Nebenbedingungen der Matrix (*Nebendiagonalen*) berücksichtigt werden.
Ein Kalman-Filter könnte die neun Zahlen der Matrix nur als Vektor verarbeiten und hätte keine Kenntnis über die Verkopplung dieser Zahlen mit ihren Nebenbedingungen. Es wäre also von Vorteil die drei Freiheitsgrade der Orientierung durch drei Zahlen darstellen zu können.

2.1.2 Euler-Winkel

Eine relativ einfache und zunächst sehr intuitiv erscheinende Form der Darstellung von Orientierungen sind dieEuler-Winkel [15]. Diese Darstellung besteht aus drei Komponenten, die jeweils die Drehung des Objektes um eine Achse beschreiben.

$$rot_{euler} = [\phi_x, \phi_y, \phi_z] \tag{2.1}$$

Es handelt sich somit um die gewünschte Darstellung von drei Freiheitsgraden mit drei Zahlen. Die Drehungen werden nacheinander ausgeführt, daraus ergibt sich bereits die erste Nebenbedingung: die Reihenfolge der Anwendung muss vorab definiert sein. Es ist einfach zu zeigen, dass Euler-Winkel keine eindeutige Darstellung sind. So ist es möglich über verschiedene Darstellungen zur gleichen Orientierung zu gelangen. Eine Drehung um $180°$ um die Y-Achse und eine darauf folgende Drehung von $90°$ um die Z-Achse hätte den gleichen Effekt wie eine Drehung um $90°$ um die negative Z-Achse. Zudem haben Euler-Winkel an bestimmten Stellen Singularitäten, das wird zum Beispiel an der Darstellung von Längengraden bei der Weltkugel deutlich: an den Polen laufen die Längengrade zusammen. Wenn man am Pol eine kleinen Schritt Richtung Greenwich geht, würde man auf der $0°$ Länge stehen, während ein genauso kleiner Schritt in die entgegengesetzte Richtung eine Länge von $180°$ bedeutet. Die Parameter der Funktion würden sich stark ändern, während der Zustand sich nur sehr wenig geändert hätte. Für einen schätzenden Filter ist das ein großes Problem, da dieser mit kleinen Änderungen an den

Parametern den Zustand nicht erreichen kann. Aus diesem Grund sind Euler-Winkel zur Darstellung von Orientierung und Rotation ungeeignet. Es ist zwar möglich die Umgebung von Singularitäten gesondert zu behandeln, für die vorliegende Anwendung soll jedoch eine geeignetere Darstellung gefunden werden.

2.1.3 Axis-Angle

Eine verwandte Form der Darstellung von Orientierungen ist die Axis-Angle Schreibweise. Auch mit dieser Darstellung werden drei Freiheitsgrade mit drei Zahlen beschrieben. Dabei wird eine einzelne beliebige Achse definiert, um die sich ein Körper dreht. Die Darstellung besteht aus drei Komponenten $v = [v_0, v_1, v_2]^T$, welche die Richtung der Achse beschreiben. Der Betrag des Vektors $|v|$ entspricht dem Winkel, um den gedreht wird.
Mit Hilfe von Axis-Angle-Darstellungen ist es jedoch nicht direkt möglich Punkte oder Vektoren zu transformieren. Diese Eigenschaft ist aber für die Anwendung von großer Bedeutung, da häufig zwischen den Koordinaten-Systemen gewechselt und transformiert werden muss. Außerdem ist auch die Axis-Angle-Darstellung nicht singularitätfrei. Alle Vektoren v mit $|v| = 2\pi$ stellen $360°$ Drehungen um verschiedene Achsen dar. Die Axis-Angle-Darstellung ist für die vorliegende Anwendung jedoch von direkter Bedeutung, da die Singularität bei kleinen lokalen Drehungen umgangen werden kann. Darauf wird in der Anpassung des Filters noch weiter eingegangen.

2.1.4 Quaternion

Quaternionen sind eine verwandte Weiterentwicklung der komplexen Zahlen, mit denen sich auch Rotationen darstellen lassen. Sie wurden um 1840 von Sir William Hamilton entdeckt [16]. Damals war jedoch noch nicht klar, wie man diese neue Art von Zahlen nutzen könnte. Sie bestehen aus einem 4D-Tupel, mit einem *reellen* und *vektoriellen* Teil und können auch als Summe von vier reellen Zahlen mit imaginären Anteil dargestellt werden.

$$q = [q_r, \ \ q_x, q_y, q_z] = q_r + q_x i + q_z j + q_z k \tag{2.2}$$

Dabei verhalten sich Quaternionen ähnlich wie die komplexen Zahlen. Operationen wie Addition, Multiplikation usw. sind durch die Eigenschaft

$$i^2 = j^2 = k^2 = i \cdot j \cdot k = -1 \tag{2.3}$$

algebraisch definiert. Sie definieren eine Darstellung von drei Freiheitsgraden mit vier Zahlen und einer Nebenbedingung und sind damit einfacher als Rotationsmatrizen. Quaternionen können ähnlich wie Matrizen Punkte in verschiedene Koordinatensysteme drehen. So wird ein Vektor v, der als vektorieller Anteil in ein Quaternion geschrieben wird,

mit

$$[0, v'] = q[0, v]q^{-1} \tag{2.4}$$

in das durch q beschriebene Koordinatensystem gedreht. Aufeinander folgende Rotationen können durch Multiplikation zusammengefasst werden.

$$q_{gesamt} = q_1 \cdot q_2 \tag{2.5}$$

Werden viele Multiplikationen durchgeführt, zum Beispiel bei der Akkumulation von Rotationen, ist die Quaternionen Arithmetik effektiver. Auch Quaternionen müssen wegen der Summierung numerischer Fehler normalisiert werden. Als Nebenbedingung gilt, dass der Betrag eines Quaternions 1 sein muss. Diese Bedingung ist mit den vier Freiheitsgrade des Quaternion wesentlich einfacher zu erfüllen als bei Rotationsmatrizen mit 6 Nebenbedingungen. Zu Rechenregeln auf Quaternionen siehe auch [17].

2.2 Tracking-Algorithmen

Die Verarbeitung unsicherer Informationen kann im Wesentlichen nur eine Schätzung eines Zustands ergeben. Das Ziel dieser Arbeit soll aber sein, aus den fehlerbehafteten Daten und der daraus resultierenden Unsicherheit zu einer Erneuerung der Schätzung eines Zustandes zu gelangen, die besser als die vorherige Schätzung, aber in jedem Fall besser als die einzelnen Messungen ist. Es wurden in der Vergangenheit verschiedene Methoden entwickelt, diesen Bedingungen gerecht zu werden. Besonders in der *Control Theory* (Theorie von Regelkreisen) werden solche Schätzer benötigt, die aus der Messung eines Systemzustandes und einer Steuervariablen einen neuen Zustand schätzen. Darüberhinaus lassen sich aber auch etwas abstraktere Systemzustände schätzen, die nicht direkt einem Regelkreis zuzuordnen sind. Im Folgenden sollen besonders häufig angewendete Verfahren beschrieben werden.
Die vorgestellten Algorithmen funktionieren rekursiv. Als Eingabe erwarten sie den letzten Zustand als Normalverteilung beschrieben durch Mean und Kovarianz (μ_{t-1}, Σ_{t-1}), die aktuelle Messung z_t und Steuereingaben u_t. Die Ausgabe ist wieder ein Zustand mit Mean μ_t und Kovarianz Σ_t. Darüberhinaus habe sie keine weiteren internen Zustände, insbesondere speichern sie keine vorangegangenen Messungen. Sie sind leicht implementierbar, auf das vorhandene Problem anwendbar und mathematisch und statistisch gut fundiert. Diese Klasse von Algorithmen heißen wegen ihres rekursiven Charakters auch *Filter*. Grundlage der im Folgenden vorgestellten Algorithmen ist die Wahrscheinlichkeitsrechnung und im speziellen der *Bayes Filter* (siehe auch Kapitel über Bayes Filter in [18]). Eine wichtige Annahme, die für alle Algorithmen gelten soll, ist, dass der Zustand *vollständig* (complete state) ist und somit die *Markovsche Regel*[19] (ein Zustand hängt nur von seinem Vorgängerzustand ab, aber nicht von dem Weg auf dem dieser

Zustand erreicht wurde) erfüllt ist.
Wie sich aus den Eingaben μ_{t-1}, Σ_{t-1} der aktualisierte Zustand μ_t, Σ_t ergibt, lässt sich aus dem Konditionierungslemma der Wahrscheinlichkeitslehre ableiten. Die einzelnen Filter aus der Familie der Kalman-Filter nutzen alle diese Aktualisierungsregeln. Die beschriebenen Algorithmen sind so formuliert, dass sich diese Gemeinsamkeit in allen Filtern wiederfindet.
Konditionierungslemma:

$$E(X|Z=z) = EX + Cov(X,Z)Cov(Z)^{-1}(z - EZ) \tag{2.6}$$
$$Cov(X|Z=z) = Cov(X) - Cov(X,Z)Cov(Z)^{-1}Cov(X,Z)^T \tag{2.7}$$

Das Lemma 2.6 integriert eine Messung Z in einen Zustand X. Passend dazu wird in 2.7 eine Messung Z in die Kovarianz des Zustands integriert. Daraus lässt sich eine allgemeine Regel für den Mean μ_t (Zustand) ableiten. Mit $EX = \bar{\mu}_t$, $EZ = \hat{z}_t$ und $z = z_t$ gilt demnach:

$$\mu_t = \bar{\mu}_t + \underbrace{\bar{\Sigma}_t^{x,z}(\bar{\Sigma}_t^{z,z})^{-1}}_{\text{Kalman Gain}}(z_t - \hat{z}_t) \tag{2.8}$$

Für die Aktualisierung der Kovarianz kann der zweite Teil des Lemmas angewendet werden. Da die Kovarianz des Zustands $Cov(X)$ nicht bekannt ist, wird hier die prädizierte Kovarianz $\bar{\Sigma}_t$ eingesetzt.

$$Cov(X|Z=z) = Cov(X) - Cov(X,Z)Cov(Z)^{-1}Cov(X,Z)^T \tag{2.9}$$
$$= \bar{\Sigma}_t - \bar{\Sigma}_t^{x,z}\bar{\Sigma}_t^{z,z^{-1}}(\bar{\Sigma}_t^{x,z})^T \tag{2.10}$$

Sowohl Gleichung 2.8 als Gleichung 2.9 beinhaltet den Term $\bar{\Sigma}_t^{x,z}\bar{\Sigma}_t^{z,z^{-1}}$ der auch als Kalman Gain K_t in den Algorithmen bezeichnet wird.

2.2.1 Kalman Filter

Einer der bekanntesten Filteralgorithmen ist der Kalman-Filter, benannt nach seinem Erfinder Ruldolf E. Kalman, der diese Entwicklung 1960 vorstellte [20]. Der Kalman-Filter ist ein *optimaler rekursiver Datenverarbeitungsalgorithmus* [21], was zunächst bedeutet, dass seine Schätzung optimal ist, also alle verfügbaren Eingaben nutzt, um diese zu einem Ergebnis zusammenzuführen und bei gleichen Eingaben zum gleichen Ergebnis kommt. Der Kalman-Filter beschreibt die Schätzung des Zustandes zur Zeit t durch einen Mittelwert μ_t (Mean) und eine Kovarianz Σ_t, wobei der Zustand von der Dimension n und die dazugehörige Kovarianz eine Matrix der Dimensionen $n \times n$ ist.
Zusätzlich zur oben erwähnten Markov-Bedingung müssen folgende drei Bedingungen erfüllt sein:

1. Der Zustandsübergang (auch Dynamik-Schritt oder *dynamic model*) muss eine lineare Funktion sein und die Unsicherheit im Modell (Rauschen) ist additiv:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$$

wobei x_t, x_{t-1} Zustandsvektoren zu den Zeitpunkten t und $t-1$ sind. Die Matrix A_t beschreibt die dynamische Transition des Zustands, die Matrix B_t wie sich eine Steuerungseingabe auf den Zustand auswirkt. ϵ_t ist ein Vektor der Dimension des Zustandes und beschreibt ein Gaussches Rauschen, mit Mean 0 und Kovarianz R_t und damit die Unsicherheit des Dynamik-Modells.

2. Linearität wird auch für das Messmodell gefordert. Auch die Unsicherheit des Messmodells wird durch Addition eines Fehlervektors ausgedrückt.

$$z_t = C_t x_t + \delta_t$$

wobei z_t die Messung ist und C_t die Gleichung für die Messfunktion, gegeben einen bestimmten Zustand x_t, in Matrix-Form darstellt. δ_t beschreibt die Ungenauigkeit des Messmodells.

3. Der initiale Zustand (Belief) muss bereits eine Gaussche Verteilung sein, damit alle weiteren errechneten Zustände ebenfalls dieser Annahme genügen.

Der Filter (Abb. 2.1) macht zunächst eine Vorhersage $\bar{\mu}_t, \bar{\Sigma}_t$ des Zustands auf Basis des Dynamik-Modells (auch: *Prediction* oder Prozessmodell), in den auch Steuervariablen einfließen können (Zeilen 1+2). Der Fehler des Dynamik-Modells wird über die Matrix R_t modelliert und auf die Kovarianz addiert.
Der Kalman-Filter korreliert die Messung und die Vorhersage des Dynamik-Modells nach der eingangs beschriebenen Aktualisierungsregel. Dafür werden die Kovarianzen $\bar{\Sigma}_t^{x,z}$ und $\bar{\Sigma}_t^{z,z}$ in den Zeilen 3+4 berechnet. Diese werden zum Kalman Gain K_t kombiniert (Zeile 6). In den Zeilen 7+8 finden sich die Regeln des Updates wieder. Zeile 5 berechnet zuvor den Mean der erwarteten Messung $\hat{z}$, der in der Update-Regel von der wahren Messung z_t subtrahiert wird. Dieser Term wird auch *Innovation* genannt. Schließlich wird das Ergebnis des Filter, der neue Zustand (μ_t, Σ_t), zurückgegeben.

2.2.2 Extended Kalman Filter

Weil der Kalman Filter Linearität in seinen Modellen vorraussetzt, ist seine Anwendung relativ stark beschränkt. Die wenigsten natürlichen physikalischen Phänomene folgen tatsächlich einer linearen Funktion. Der *Extended Kalman Filter* (EKF) erweitert den Kalman-Filter um die Möglichkeit auch nicht-lineare Modelle zu berechnen. Die Matrizen A_t und B_t werden zunächst durch die nicht-lineare Funktion $g(\mu_{t-1}, u_t)$, welche die Dynamik

KALMAN-FILTER ALGORITHMUS$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1 $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$
2 $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$
3 $\bar{\Sigma}_t^{z,z} = C_t\bar{\Sigma}_t C_t^T + Q_t$
4 $\bar{\Sigma}_t^{x,z} = \bar{\Sigma}_t C_t^T$
5 $\hat{z} = C_t\bar{\mu}_t$
6 $K_t = \bar{\Sigma}_t^{x,z}\bar{\Sigma}_t^{z,z^{-1}}$
7 $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z})$
8 $\Sigma_t = \bar{\Sigma}_t - K_t\bar{\Sigma}_t^{z,z}K_t^T$
9 **return** μ_t, Σ_t

Abbildung 2.1: Der Kalman Filter für lineare gausssche Zustandsübergänge

EXTENDED KALMAN-FILTER ALGORITHMUS$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1 $\bar{\mu}_t = g(u_t, \mu_{t-1})$
2 $\bar{\Sigma}_t = G_t\Sigma_{t-1}G_t^T + R_t$
3 $\bar{\Sigma}_t^{z,z} = H_t\bar{\Sigma}_t H_t^T + Q_t$
4 $\bar{\Sigma}_t^{x,z} = \bar{\Sigma}_t H_t^T$
5 $\hat{z} = h(\bar{\mu}_t)$
6 $K_t = \bar{\Sigma}_t^{x,z}\bar{\Sigma}_t^{z,z^{-1}}$
7 $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z})$
8 $\Sigma_t = \bar{\Sigma}_t - K_t\bar{\Sigma}_t^{z,z}K_t^T$
9 **return** μ_t, Σ_t

Abbildung 2.2: Der Extended Kalman Filter Algorithmus

des Systems beschreibt, und C_t durch $h(x_t)$ ersetzt. Der Extended Kalman Filter linearisiert die Funktionen g und h am aktuellen Schätzwert durch Taylor Expansion. Es werden die partiellen Ableitungen

$$g'(u_t, x_{t-1}) = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}}\Big|_{x_{t-1}=\mu_{t-1}} \tag{2.11}$$

und

$$h'(x_t) = \frac{\partial h(x_t)}{\partial x_t}\Big|_{x_t=\bar{\mu}_t} \tag{2.12}$$

gebildet und als Matrix G_t bzw. H_t zusammengefasst. Diese Matrizen sind ebenfalls unter dem Namen *Jacobian* (Jacobi-Matrix) [22] bekannt und stellen die beste lineare Approximation einer differenzierbaren Funktion an einer bestimmten Stelle dar. Nach dem Ersetzten der linearen Zustandsübergangsmatrizen durch ihre lineare Approximation,

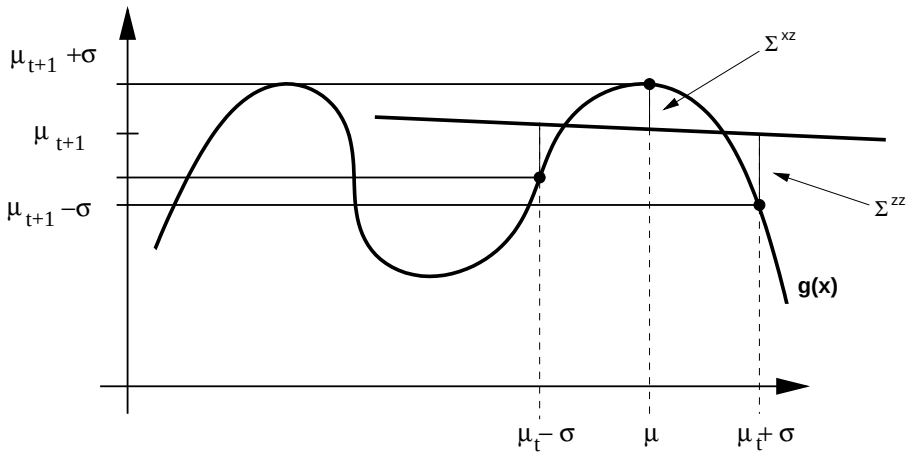


Abbildung 2.3: Prinzip der Sigma-Punkt Approximierung. Es wird eine Gerade, ähnlich einer Ausgleichsgeraden, durch die drei Punkte $\mu, \mu + \sigma, \mu - \sigma$ gelegt.

verhält sich der EKF algorithmisch wie der reguläre Kalman-Filter. Die Aktualisierungsregeln sind identisch, der EKF unterscheidet sich nur in der Art wie die Kovarianzen $\bar{\Sigma}_t^{x,z}$ und $\bar{\Sigma}_t^{z,z}$ erstellt werden.
In praktischen Anwendungen ist jedoch das Abbilden der Dynamik- und der Messfunktion und damit auch das Bilden ihrer Ableitungen oft nicht möglich, weil das reale Verhalten eines Systems nicht in Form von Gleichungen vorliegt, sondern nur durch Sensoren oder durch eine software-technische Black-Box (Bibliothek, etc.) beschrieben wird. Liegen dennoch systembeschreibende Gleichungen vor, ist das Erstellen einer Jacobi-Matrix oft nicht trivial und fehlerträchtig.

2.2.3 Unscented Kalman Filter

Eine ebenfalls für den nicht-linearen Fall angepasste Version des Kalman Filter ist der von *Julier und Uhlmann* 1997 [23] vorgestellte *Unscented Kalman Filter* (UKF), damals jedoch noch nicht unter diesem Namen. Auch der UKF versucht die nicht-lineare Funktion durch Linearisierung anzupassen, um die regulären Schritte des Kalman-Filters anwenden zu können. Im Unterschied zum EKF werden jedoch keine Ableitungen der Modell- und Messfunktionen gebildet; stattdessen wird die Normalverteilung des Zustands approximiert, indem zwischen Punkten, die mit der aktuellen Verteilung getreut wurden, eine Linearisierung ähnlich einer Ausgleichsgeraden (Abb. 2.3) durchgeführt wird. Diese Familie von Filtern wird in der Literatur auch oft *Sigma-Punkt Filter* genannt.

Unscented Transform

Der Kern des UKF ist die deterministische Transformation von Mean und Kovarianz der letzen Schätzung ($t-1$) in den aktuellen Zeitrahmen (t). Dazu wird mit einer bestimmten Anzahl von *Sigma-Punkten* die Verteilung der Schätzung approximiert und diese mit Hilfe des Prozessmodels in den aktuellen Zeitrahmen (t) transferiert (Abb. 2.4). Die Wahl der

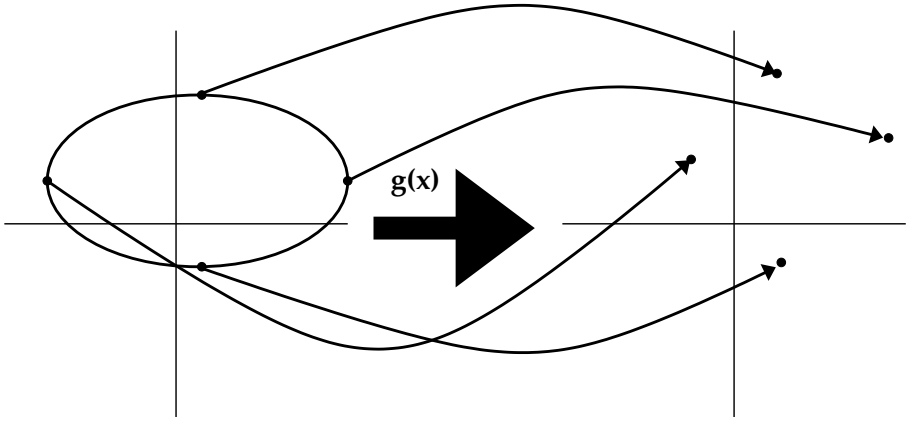


Abbildung 2.4: Die Unscented Transformation in 2D (nach [23]). Rund um den Mean im A-priori Zustand werden Sigma-Punkte gelegt, welche dann mit der Dynamik-Funktion transformiert werden.

Sigma-Punkte wird in [23] beschrieben, ist jedoch nicht fest und kann zum Beispiel mit verschiedenen Parametern verändert werden. In der vorliegenden Anwendung werden $2n+1$ Sigma-Punkte X wie folgt erzeugt:

$$X_0 = \bar{x} \qquad\qquad w_m = \frac{1}{2n+1}$$

$$X_i = \bar{x} + (\sqrt{\Sigma_t^{x,x}})_i \qquad\qquad w_c = \frac{1}{2}$$

$$X_{i+n} = \bar{x} - (\sqrt{\Sigma_t^{x,x}})_i$$

wobei $(\sqrt{\Sigma_{xx}})_i$ die Wurzel einer symmetrischen Matrix symbolisiert. Diese wird aus der $i$ten Spalte einer durch Cholesky-Zerlegung erstellten Matrix $\Sigma_{xx} = LL^T$ erzeugt. W_m und W_c sind die Gewichtungen der Punkte bei der späteren Zusammenfassung zum Mean ($_m$) und zur Kovarianz ($_c$). Auch die Berechnung der Gewichte kann auf unterschiedliche Art geschehen. In [23] werden unterschiedliche Gewichte der Punkte vorgestellt, so wird zum Beispiel der Mean stärker gewichtet als die außenliegenden Sigma-Punkte. In dieser Arbeit sollen die Sigma-Punkte jedoch gleich gewichtet werden.
Nachdem die Menge der Sigma-Punkte $\bar{X}_t^*$ durch das Prozessmodell propagiert wurde, kann sie wieder zu einem Mean $\bar{\mu}_t$ und einer Kovarianz $\bar{\Sigma}_t$ berechnet werden.

Gleiches geschieht bei der Anwendung des Messmodells $h(x_t)$. Anstelle einer Linearisierung der Funktion wird die Verteilung approximiert und alle Sigma-Punkte durch die Messfunktion propagiert. Diese (Unscented-) Transformation kann anstelle der linearisierten Funktionen in Form der Jacobi-Matrizen im EKF eingesetzt werden. Tatsächlich kann durch weitere Feinabstimmungen in der Auswahl der Sigma-Punkte die Genauigkeit erhöht werden. Die hier vorgestellte Variante approximiert die Funktionen ebenso genau wie der EKF. Es können jedoch auch höhere Ordnungsterme modelliert werden.
Die Autoren von [23] vergleichen ihre Methode auch mit der Monte-Carlo-Methode für Particle Filter, wie sie zum Beispiel in [24] oder für den Anwendungsbereich der mobilen Roboterlokalisierung in [25] beschrieben werden. Die Partikel werden jedoch nicht zufällig verteilt, sondern nach deterministischen, statistisch fundierten Verfahren ausgewählt. Daher kommt diese Methode mit sehr viel weniger zu transformierenden Punkten aus, wenn die Schätzung tatsächlich einer Gaussverteilung ähnelt.

UKF-Algorithmus

Mit den Erkenntnissen zur Unscented Transformation kann der Kalman-Filter Algorithmus abgewandelt werden. Die hier vorgestellte Variante entstammt hauptsächlich [18]. Eingabe und Ausgabe des Filters unterscheiden sich nicht vom EKF. Zunächst erzeugt der Filter eine Menge von Sigma-Punkten aus dem alten Mean μ_{t-1}, diese werden in Abbildung 2.5 Zeile 2 durch die rauschfreie Dynamikfunktion (Prozeßmodell: $g(u_t, X_{t-1})$) propagiert. Bei der statistischen Aufbereitung der neuen Kovarianz $\bar{\Sigma}_t$ wird eine Rauschmatrix R_t addiert (Zeile 4). Die Gaussverteilung mit $\bar{\mu}_t$ und $\bar{\Sigma}_t$ wird erneut mit Hilfe von Sigma-Punkten approximiert (Zeile 5) und durch die Messfunktion $h(\bar{X}_t)$ propagiert (Zeile 6). Der Mean der erwarteten Messung $\hat{z}$ wird mit der Gewichtung w_m aus den propagierten Sigma-Punkten berechnet (Zeile 7), im Vergleich wird beim EKF die Messfunktion in Kombination mit dem prädizierten Mean benutzt. Die Kovarianz der erwarteten Messung $\bar{\Sigma}_t^{z,z}$ wird über die Differenz des Sigma-Punktes zu der erwarteten Messung $\hat{z}$ erstellt und das Messrauschen Q_t addiert (Zeile 8). Die *Cross-Variance* $\bar{\Sigma}_t^{x,z}$ wird analog dazu erzeugt (Zeile 9). Der Kalman-Gain K_t wird in Zeile 10 korrespondierend zu der Zeile 6 im EKF berechnet. Der weitere Algorithmus unterscheidet sich nicht mehr von EKF und KF.
Der Unscented Kalman Filter sieht zunächst etwas aufwendiger als die vorhergegangenen Filer aus, ist jedoch leichter zu implementieren, da keine Jacobi-Matrix wie für den EKF erstellt werden muss. Die Komplexität wird von den Autoren von [18] als gleichwertig zum EKF angegeben, in der Praxis kann der EKF jedoch wegen dünner Jacobi-Matrizen (*sparse-matrix*) etwas schneller sein, bzw. läßt sich die Implementierung auf das entsprechende Problem optimieren. Der große Vorteil des UKF liegt in der direkten Verwendung der Modellfunktion ohne Ableitungen. So können zum Beispiel auch analy-

UNSCENTED KALMAN-FILTER ALGORITHMUS$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1 $X_{t-1} = (\mu_{t-1} \quad \mu_{t-1} + \sqrt{\Sigma_{t-1}} \quad \mu_{t-1} - \sqrt{\Sigma_{t-1}})$
2 $\bar{X}_t^* = g(u_t, X_{t-1})$
3 $\bar{\mu}_t = \sum_{i=0}^{2n} w_m \bar{X}_t^{*[i]}$
4 $\bar{\Sigma}_t = (\sum_{i=0}^{2n} w_c(\bar{X}_t^{*[i]} - \bar{\mu}_t)(\bar{X}_t^{*[i]} - \bar{\mu}_t)^T) + R_t$
5 $\bar{X}_t = (\bar{\mu}_t \quad \bar{\mu}_t + \sqrt{\bar{\Sigma}_t} \quad \bar{\mu}_t - \sqrt{\bar{\Sigma}_t})$
6 $\bar{Z}_t = h(\bar{X}_t)$
7 $\hat{z}_t = \sum_{i=0}^{2n} w_m \bar{Z}_t^{[i]}$
8 $\Sigma_t^{z,z} = (\sum_{i=0}^{2n} w_c(\bar{Z}_t^{[i]} - \hat{z}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T) + Q_t$
9 $\Sigma_t^{x,z} = \sum_{i=0}^{2n} w_c(\bar{X}_t^{[i]} - \bar{\mu}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T$
10 $K_t = \Sigma_t^{x,z}(\Sigma_t^{z,z})^{-1}$
11 $\mu_t = \bar{\mu}_t + K_t(z_t - \hat{z}_t)$
12 $\Sigma_t = \bar{\Sigma}_t - K_t \Sigma_t^{z,z} K_t^T$
13 **return** μ_t, Σ_t

Abbildung 2.5: Der Unscented Kalman Filter Algorithmus

tisch nicht bekannte Funktionen als Mess- oder Modellfunktionen verwendet werden.

2.3 Schätzung eines Zustands im Rahmen eines Kamera-Inertialsensor-Systems

Für die Schätzung einer Ballflugbahn mit einem frei beweglichen Kamera-System müssen mehrere Variablen mit Hilfe eines hier vorgestellten Filters bearbeitet werden. Dazu muss zunächst entschieden werden, wie der Systemzustand repräsentiert werden soll. Es sind bereits einige Repräsentationen für Orientierungen und Rotationen vorgestellt worden, von denen nun eine geeignete ausgewählt werden muss. Zusätzlich muss entschieden werden, wie das Gesamtsystem repräsentiert werden soll. Es sollen zwei zunächst unabhängige Zustände geschätzt werden:

1. Das Kamera-Inertialsensor-System mit seinen Zuständen:
   Position: $[x_c, y_c, z_c]^T$
   Geschwindigkeit: $[v_{xc}, v_{yc}, v_{zc}]^T$
   Orientierung: $[q_0, (q_x, q_y, q_z)]$ bei Darstellung mit einem Quaternion
2. Der Ball mit folgenden Zuständen:
   Position: $[x_b, y_b, z_b]^T$
   Geschwindigkeit: $[v_{xb}, v_{yb}, v_{zb}]^T$
   Luftwiderstand: $drag$ wurde jedoch aus Zeitgründen in der Arbeit nicht mehr berücksichtigt.

Es ist jedoch möglich, dass die Position des Balls, da die Flugbahn einem festen Modell folgt, ebenfalls Informationen über die Position der Kamera (und umgekehrt) liefert. Würden die Zustände als ein Vektor in einem Filter repräsentiert, kann sich eine eventuelle Verkopplung dieser Informationen in der Kovarianzmatrix in den Nebendiagonalen niederschlagen. Als Nachteil wächst allerdings die Größe des Zustands an. Da der Luftwiderstand aus Zeitgründen nicht mehr implementiert wurde (jedoch vorgesehen ist) würde ein kombinierter Zustand aus 15 Komponenten bestehen (auch wenn ein Quaternion aus vier Komponenten besteht, hat der Zustandsraum tatsächlich aber nur 15 Elemente, siehe hierzu Abschnitt 2.3.1). Es läßt sich schwer abschätzen, wie stark die verkoppelte Information tatsächlich dem Gesamtsystem helfen kann, jedoch sollte die Möglichkeit nicht ausgeschlossen werden. Folglich arbeitet das System mit einem kombiniertem Zustand.

2.3.1 Wahl der Filter-Methode

Die Vergleiche und Beschreibungen von Kalman-Filter, EKF und UKF haben schon einen guten Einblick gegeben, so dass hier der Unscented Kalman Filter als Auswahl nur kurz begründet wird. Die relativ einfache Implementierung und die Flexibilität des UKF stellen alleine schon ausreichend Vorteile dar.

Auschluss anderer Verfahren

Die bereits häufig erwähnten Particle-Filter scheinen auch geeignet zu sein ein Tracking-Problem dieser Form zu lösen. Die Beschreibung des Systemzustandes macht aber deutlich, dass es sich hier nicht um ein einfaches kleines System handelt, sondern mit seinen 15 Freiheitsgraden schon relativ komplex erscheint. Um eine ausreichende Genauigkeit mit Particle-Filtern zu erreichen, muss eine entsprechend hohe Menge an Samples (Partikeln) benutzt werden. Diese steigt mit jeder weiteren Dimension an. Da der Systemzustand, aufgrund der Verkopplung der Informationen, die sich aus der modellhaft bekannten Flugbahn und der Umgebung ergeben, nicht getrennt werden soll, kann hier auch nicht für einen Teil ein anderer Filter eingesetzt werden. Es besteht die Befürchtung, dass sich ein Particle-Filter mit der Menge an Freiheitsgraden nicht sinnvoll und effizient implementieren läßt. Die Ähnlichkeit des UKF zum Particle-Filter wurde bereits beschrieben. Der UKF basiert auf einer ähnlichen Funktionionalität, ist jedoch durch die statistisch fundierte Generierung der Sigma-Punkte (die vergleichbar mit den Partikeln des Particle-Filters sind) wesentlich effizienter.

Quaternionen als Orientierungsrepräsentation und Anpassung des Filters

Der Kalman-Filter erwartet, dass die Parameter des Zustandvektors unabhängig voneinander sind. Möchte man nun Orientierungen im Zustandsvektor des Filter beschreiben, so braucht man eine geeignete Repräsentation. Allgemein ist die Orientierung im Raum ($SO(3)$) eine 3-*Mannigfaltigkeit* [26], das heißt eine Orientierung läßt sich lokal als Vektor mit drei unabhängigen Parametern darstellen. Es kann jedoch bewiesen werden, dass es keine Darstellung für $SO(3)$ mit drei Parametern gibt, die singularitätsfrei ist (Der Beweis wird in [27] geführt). Letzteres ist jedoch eine Bedingung für den Kalman-Filter: Kleine Änderungen im Zustand erfordern auch nur kleine Änderungen in den Parametern. Es muss demnach eine andere Form der Repräsentation gefunden werden. Nach der Diskussion verschiedener Darstellungen in Abschnitt 2.1 soll hier ein Quaternion als Repräsentation gewählt werden.
Mit einem Quaternion und der 3-Mannigfaltigkeit der Orientierung kann ein Kalman-Filter oder ein UKF nicht umgehen. Die Übergänge der Repräsentation in die verschiedenen Räume $\mathbb{R}^4 \Leftrightarrow \mathbb{R}^3$ müssen daher vor dem Algorithmus verborgen bleiben. Das wird durch eine Kapselung des Zustands erreicht, was sich in der softwaretechnischen Implementierung in einer eigenen Klasse widerspiegelt. Dieser neue Zustand wird mit neuen Funktionen versehen, die zu $+$ und $-$ äquivalente Operationen $\oplus$ und $\ominus$ ausführen können und so die vektoriellen Operationen imitieren. Die speziellen Funktionen Dynamik- und Messfunktion, die ohnehin für jeden neuen Filter gesondert implementiert werden müssen, werden ebenfalls Teil des Zustands und können so auf die Besonderheiten der Quaternionen-Darstellung angepasst werden. Generische Funktionen des

Filters wie Sigma-Punkt Erzeugung, Kalman Gain oder Zustandsaktualisierung nutzen die *überladenen* Operationen $\oplus$ und $\ominus$.
Die Verwendung von Quaternionen in Kalman-Filtern ist bis jetzt wenig in der Literatur behandelt worden. Die Idee für diesen Ansatz und eine auf Quaternionen spezialisierte Lösung liefert [28]. Ähnliche Ansätze und Vergleiche von Methoden auch für den Extended Kalman Filter beschreiben [29] und [30]. Hier wird mit den äquivalenten Operationen $\oplus$ und $\ominus$ jedoch eine generelle Lösung der Zustandsübergänge und der Anpassung des UKF vorgestellt. Jede andere Form der Mannigfaltigkeit, beispielweise eine Darstellung mit Rotationsmatrizen, kann ebenso mit diesen Operationen in den Filter integriert werden.
Im Folgenden sollen die Besonderheiten bei der Anpassung des Filters behandelt werden.

Einführung der $\oplus$- und $\ominus$-Operationen

Die Operationen $\oplus$ und $\ominus$ verbergen vor dem Filter das Problem, dass im vorliegenden Fall zwischen Vektorraum $\mathbb{R}^n$ und Zustand S mit Quaternion unterschieden werden muss. Oder allgemein: Zwischen Vektorraum und einen Zustandsraum mit weiteren undefinierten Nebenbedingungen, für die aber eine Umrechnung existiert. Die Operationen sind dann wie folgt definiert:

$$\oplus \ : \ S \times \mathbb{R}^n \rightarrow S \tag{2.13}$$

$$\ominus \ : \ S \times S \rightarrow \mathbb{R}^n \tag{2.14}$$

Es werden mit $\oplus$ kleine parameterisierte Drehungen im Vektorraum auf den Zustandraum S addiert, während mit $\ominus$ die Differenz zweier Zustände in den Vektorraum übertragen werden. Insbesondere gilt dann, mit $s_n \in S$:

$$s_1 \oplus (s_2 \ominus s_1) = s_2 \tag{2.15}$$

Es muss für jede Form der Mannigfaltigkeit eine besondere Parametrisierung gefunden werden.

Zustandsvektor

Der Zustandsvektor und gleichzeitig der Mean μ_t im Filter wird durch eine eigene Abstraktion dargestellt. Er besteht aus einem vektoriellen Teil v und dem Quaternion q für die Orientierung der Kamera:

$$\mu_t = \begin{pmatrix} v \\ q \end{pmatrix} \tag{2.16}$$

Die Operationen $\ominus$ und $\oplus$ sind dann wie folgt definiert:

OPERATOR $\oplus$ $(s : S,\ r : \mathbb{R}^n)$	OPERATOR $\ominus$ $(s_1 : S,\ s_2 : S)$
1 $q : Q,\ v : \mathbb{R}^{n-3}$	1 $r : \mathbb{R}^n$
2 $v \leftarrow s.v + r[0,..,n-3]$	2 $r[0,..,n-3] \leftarrow s1.v - s2.v$
3 $dR \leftarrow$ QUATERNION$(r[n-3,..,n])$	3 $dR \leftarrow s2.q \cdot s1.q^{-1}$
4 $q = s.q \cdot dR$	4 $r[n-3,..,n] \leftarrow$ AXIS-ANGLE(dR)
5 **return** $S(v,q)$	5 **return** r

Abbildung 2.6: Die Operationen $\oplus$ und $\ominus$ für den Zustand aus vektoriellem Anteil und einem Quaternion.

Prozessmodell

Das Prozessmodell $g()$ (auch: Dynamikmodell) muss für jeden Filter gesondert implementiert werden. Es spiegelt das dynamische Verhalten des Systems über eine gewisse Zeit δt wider. Das Modell des Balls ist durch die Gleichungen

$$x_t = x_{t-1} + v_x\ \delta t \tag{2.17}$$
$$y_t = y_{t-1} + v_y\ \delta t \tag{2.18}$$
$$z_t = z_{t-1} + v_z\ \delta t \tag{2.19}$$
$$v_{xt} = v_{xt-1} \tag{2.20}$$
$$v_{yt} = v_{yt-1} \tag{2.21}$$
$$v_{zt} = v_{zt-1} + g\ \delta t \tag{2.22}$$

beschrieben. Wobei g die Erdgravitation ist. Der Einfluß des Luftwiderstandes wird hier wegen der Größe und des Gewichtes des Ball vernächlässigt. Sein Einfluß kann aber ähnlich wie die Gravitation auf die Geschwindigkeiten addiert werden. Er wirkt wie eine Kraft, die auf den Ball entgegen der Bewegungsrichtung ausgeübt wird.

$$F_d = -\frac{1}{2} p A C_d v^2 \tag{2.23}$$

p : Dichte(für Luft$1.29\frac{kg}{m^3}$)
A : Anströmfläche (reference Area) für Kugel: $\pi\ r^2$
C_d : Widerstandsbeiwert (drag coefficient), nichtC_w
v : Geschwindigkeit

Für langsame Geschwindigkeiten kann auch folgende Formel verwendet werden:

$$F_d = -br \tag{2.24}$$

wobei $b = 6\pi r\eta$ und $\eta = 17,1$ für Luft.
Die Dynamikfunktion des Kamera-Sensor Systems wird durch die IMU vorgegeben. Die Beschleunigung liefert nach zweifacher Integration die Positionsänderung des Sensors. Die Integrationen werden durch Summieren und Akkumulation im Zustand umgesetzt, sie werden vektoriell addiert und nutzen nur die Information über die Orientierung q. Dafür wird zunächst der Beschleunigungsvektor dA_c in Weltkoordinaten gedreht und die Gravitation herausgerechnet.

$$dA'_w = q \cdot dA_c \tag{2.25}$$
$$dA_w = dA'_w + g \tag{2.26}$$
$$dV_w = dA_w \cdot \delta t \tag{2.27}$$

dV_w beschreibt schließlich die Änderung der Geschwindigkeit pro Zeitschritt in Weltkoordinaten. Damit läßt sich der Zustand aktualisieren.

$$vx^c_t = vx^c_{t-1} + dV^{[x]}_w \tag{2.28}$$
$$vy^c_t = vy^c_{t-1} + dV^{[y]}_w \tag{2.29}$$
$$vz^c_t = vz^c_{t-1} + dV^{[z]}_w \tag{2.30}$$
$$x^c_t = x^c_{t-1} + vx^c_t \cdot \delta t \tag{2.31}$$
$$y^c_t = y^c_{t-1} + vy^c_t \cdot \delta t \tag{2.32}$$
$$z^c_t = z^c_{t-1} + vz^c_t \cdot \delta t \tag{2.33}$$

Die Änderung der Orientierung des Sensors wird durch die gemessenen Winkelgeschwindigkeiten bestimmt. Bei kleinen Änderungen, die im Fall einer 100Hz Abtastrate vorliegen, kann die Änderung näherungsweise mit der Geschwindigkeit gleichgesetzt werden. Die drei Messungen des Sensors beschreiben daher in Axis-Angle Form die lokale Rotation. Diese wird in ein Quaternion umgewandelt, um dann auf den Zustand multipliziert werden zu können. Im Zeitintervall δt gilt:

$$angle: \ \alpha_\delta = \ |\vec{\omega}| \ \delta t \tag{2.34}$$
$$axis: \ \vec{e}_\delta = \ \frac{\vec{\omega}}{|\vec{\omega}|} \tag{2.35}$$

Das korrespondierende Quaternion bildet sich dann wie folgt:

$$q_\delta = \ \left[\cos(\frac{\alpha_\delta}{2}), \ \vec{e}_\delta \sin(\frac{\alpha_\delta}{2})\right] \tag{2.36}$$

Diese Relativrotation kann auf das Zustands-Quaternionen mit einer rechtseitigen Multiplikation *akkumuliert* werden.

$$q_{t+1} = q_t \ q_\delta \tag{2.37}$$

Messfunktion

Die Messfunktion für einen Kalman-Filter muss ebenfalls für jeden Filter individuell angepasst werden. In der vorliegenden Anwendung hat das Messmodell keinen direkten Einfluß auf die Orientierung, diese wird aber benötigt um zum Beispiel eine Marken- oder Ballobservation in das Körperkoordinatensystem zu drehen. Wie bereits im Abschnitt 2.1 erwähnt, kann ein Quaternion einen Vektor in das repräsentierte Koordinatensystem drehen. Das Quaternion im Zustand beschreibt in dieser Logik die Relativdrehung aus dem Körperkoordinatensystem des Sensors in das Weltkoordinatensystem. Die Messung findet im Körperkoordinatensystem statt, während die Repräsentation der Objekte in Weltkoordinaten vorliegt. Das Messmodel muss eine erwartete Messung berechnen und dazu ein Weltobjekt in Körperkoordinaten wandeln. Die Gleichung 2.38 wendet diese Vorgabe auf einen beliebigen Vektor v an.

$$[0, v'] = q_{x_t}^{-1} \, [0, v] \, q_{x_t} \tag{2.38}$$

Die tatsächliche Messung misst jedoch in der 2D-Ebene des Bildes. Das bedeutet zunächst, dass die Entfernung eines Objekte nur über seinen Radius (Durchmesser) bestimmt werden kann. Alle Punkte, die in den Bildkoordinaten gemessen werden, werden mit einer Kameraabbildungsgleichung zunächst in das Kamerakoordinatensystem (und damit auch von 2D nach 3D) übertragen. Kamerakoordinatensystem und das Koordinatensystem werden hier zunächst als gleich behandelt.

Generierung der Sigma-Punkte

Bei der Zerlegung einer Normalverteilung in Sigma-Punkte können die oben beschriebenen besonderen Operationen angewandt werden. Die Erzeugung der Sigma-Punkte aus dem UKF (Abb. 2.5, Zeile 1)

$$X_{t-1} = \; (\; \mu_{t-1} \qquad \mu_{t-1} + \sqrt{\Sigma_{t-1}} \qquad \mu_{t-1} - \sqrt{\Sigma_{t-1}}) \tag{2.39}$$

ändert sich mit einfacher Ersetzung des Operators $+$ in

$$X_{t-1} = \; (\; \mu_{t-1} \qquad \mu_{t-1} \oplus \sqrt{\Sigma_{t-1}} \qquad \mu_{t-1} \oplus -(\sqrt{\Sigma_{t-1}})) \tag{2.40}$$

Die Spalten der Kovarianz (siehe Abschnitt 2.2.3), die auf den Mean (also den Zustandsvektor) addiert oder subtrahiert werden, haben eine andere Dimension als der Zustand. Eine Vektoroperation kann wegen der im Zustand enthaltenen Quaternionen nicht direkt durchgeführt werden. Abbildung 2.7 zeigt in Pseudo-Code nochmal, wie der Algorithmus angepasst werden muss. Zeile 4 und 5 benutzen die Operatoren $\ominus$ und $\oplus$, ansonsten bleibt der Algorithmus gleich.

```
GENERATE-SIGMA-POINTS(μ, Σ)
1  sigmapoints[] ← μ
2  U ← CHOLESKY(Σ)
3  for i = 0 to N
4  do sigmapoints[] ← μ ⊕ U_{i−1}
5     sigmapoints[] ← μ ⊖ U_{i−1}
6  return sigmapoints
```

Abbildung 2.7: Pseudo-Code: Erzeugen von Sigma-Punkten mit den speziellen Operationen des Zustands

Berechnung des Mean

Nach der Propagierung der Sigma-Punkte durch die Dynamikfunktion müssen diese wieder zu einem Mean und einer Kovarianz zusammengefasst werden. Dabei muss beachtet werden, dass im Zustandsraum nicht summiert werden kann und so eine einfache Mittelwertbildung für den Mean nicht möglich ist. Eine allgemeine Lösung findet sich in der Parametrisierung aller Sigma-Punkte zu einem Referenzwert und der Mittelwertbildung über diese Werte, die dann wieder im Vektorraum vorliegen und so eine Mittelwertbildung zulassen. Der Mittelwert wird dann wieder als Referenzwert benutzt und solange weitere Iterationen durchgeführt, bis der Fehler ausreichend klein ist, die Funktion also konvergiert. Unter Verwendung der Operationen $\ominus$ und $\oplus$ ist der Algorithmus wie folgt:

$$\bar{\mu}_{t_0} = \bar{X}_t^{*[0]} \tag{2.41}$$

$$\bar{\mu}_{t_{i+1}} = \bar{\mu}_{t_i} \oplus \frac{1}{n} \sum_n \bar{X}_t^{*[n]} \ominus \bar{\mu}_{t_i}, \qquad \text{mit } \bar{\mu}_t = \lim_{i \to \infty} \bar{\mu}_{t_i} \tag{2.42}$$

Dieser Algorithmus stellt eine Lösung für beliebige Mannigfaltigkeiten dar und ist eine Verallgemeinerung des Algorithmus aus [28] zum iterativen Finden des Means einer Menge von Quaternionen, welcher an die Untersuchungen aus [31] zur allgemeinen Berechnung des Mean von geometrischen Merkmalen angelehnt ist. Die Implementierung macht davon Gebrauch, indem nicht der gesamte Zustand bis zur Konvergenz iteriert wird, sondern der vektorielle Teil des Zustands mit normaler Mittelwertbildung auf Vektoren berechnet wird und nur der Mean des Quaternionenanteils über Iteration berechnet wird. Die Implementierung findet sich in Pseudocode in Abbildung 2.8.

```
ITERATIVE-MEAN-FINDING(S : set of sigma-points)
1  q ← S[0]  ; use some value as starting point
2  while error > ε
3  do for i in S
4     do e ← sq^-1
5        errorvec[] ← AXIS-ANGLE(E)
6        error ← (1/i) Σ_i errorvec[i]
7     q_e ← QUATERNION(ERROR)
8     q ← q_e q
9  return q
```

Abbildung 2.8: Finden des Mean für einen Satz von Quaternionen wie es nach der Generierung von Sigma-Punkten notwendig ist.

Es werden zu einem gewählten Quaternion (hier bietet sich als beste Näherung der alte Mean an, also $S[0]$), die relativen Drehungen der einzelnen Sigma-Punkte berechnet (Zeile 4). Diese Relativdrehungen werden in Axis-Angle Form gebracht (Zeile 5). In dieser Darstellung kann mit vektoriellen Operationen ein Mittelwert gebildet werden, der die Fehlerabweichung beschreibt (Zeile 6). Dieser wird in ein Quaternion gewandelt und auf q angewendet (Zeile 8). Wenn der Fehler klein genug ist, wird die Iteration abgebrochen und der gefundene Mean zurückgegeben.

Ermitteln der Kovarianz

Für die Berechnung von $\bar{\Sigma}_t$ und $\bar{\Sigma}_t^{x,z}$ wird ebenfalls die Operation $\ominus$ benutzt. Sie bildet die Differenz zweier Zustände in den Vektorraum ab. Das ist an zwei Stellen im UKF notwendig. Für den a priori Zustand $(\bar{\mu}_t, \bar{\Sigma}_t)$ wird die Kovarianz $\bar{\Sigma}_t$ mit

$$\bar{\Sigma}_t = (\sum_{i=0}^{2n} w_c^{[i]} (\bar{X}_t^{*[i]} - \bar{\mu}_t)(\bar{X}_t^{*[i]} - \bar{\mu}_t)^T) + R_t \tag{2.43}$$

berechnet (Zeile 4, Abb. 2.5). Da sowohl $\bar{X}_t^*$ als auch $\bar{\mu}_t$ keine rein vektoriellen Größen sind, ist eine Substraktion und anschließende Multiplikation mit dem transponierten Vektor nicht möglich. Es wird daher die Ersetzung mit der bereits eingeführten Operation $\ominus$ zu

$$\bar{\Sigma}_t = (\sum_{i=0}^{2n} w_c^{[i]} (\bar{X}_t^{*[i]} \ominus \bar{\mu}_t)(\bar{X}_t^{*[i]} \ominus \bar{\mu}_t)^T) + R_t \tag{2.44}$$

vorgenommen. Für die Berechnung von $\bar{\Sigma}_t^{x,z}$ wird die gleiche Ersetzung vorgenommen. Aus der Berechnung von $\Sigma_t^{x,z}$ im UKF (Abb. 2.5, Zeile 9)

$$\Sigma_t^{x,z} = \sum_{i=0}^{2n} w_c(\bar{X}_t^{[i]} - \bar{\mu}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T \tag{2.45}$$

wird unter Verwendung von $\ominus$

$$\Sigma_t^{x,z} = \sum_{i=0}^{2n} w_c(\bar{X}_t^{[i]} \ominus \bar{\mu}_t)(\bar{Z}_t^{[i]} - \hat{z}_t)^T \tag{2.46}$$

Rauschen im Prozess- und Messmodell

Der UKF behandelt Rauschen in Prozess- und Messfunktion additiv. Es werden die Kovarianzen R_t für das Prozessrauschen und Q_t für das Messrauschen bei der Berechnung der jeweiligen Kovarianzen addiert. Es ist jedoch auch möglich im Falle des Prozessrauschens den Fehler bereits vor der Erzeugung der Sigma-Punkte auf die Kovarianz des Zustands zu addieren und so eine Menge von bereits verrauschten Zuständen zu erlangen der Vorteil liegt in der möglichen Wiederverwendung des Sigma-Punkt-Arrays, was wegen der Cholesky-Zerlegung und der Wechsel zwischen Vektroraum und Zustandsraum (also zwischen Vektordarstellung und Zustand mit Quaternion), zu großen Effizienzsteigerungen des Algorithmus führen kann. Dieser Ansatz wurde von den Autoren von [28] gewählt und sollte zunächst auch für die vorliegende Implementierung verwendet werden. Tatsächlich wird die Implementierung durch diesen Ansatz erheblich komplizierter und wurde deshalb verworfen.

2.3.2 Freiheitsgrade

Für ein funktionierendes Tracking-System müssen über die Sensoren ausreichend viele Informationen erlangt werden können, so dass alle Freiheitsgrade im System abgedeckt werden. Ein freibewegliches Kamerasystem hat demnach sechs Freiheitsgrade, drei translatorische (Position in x, y, z) und drei Orientierungsfreiheiten (Drehung um die Achsen u, v, w) (Abb. 2.9). Das System wird als erstes über den Inertial-Sensor mit Informationen versorgt, es werden dabei im Prinzip alle Freiheitsgrade abgedeckt. Die eingebauten Accelerometer geben Auskunft über die translatorischen Bewegungen, während die Gyroskope die Bewegungen um die Körperachsen registrieren. Die ersten praktischen Erfahrungen mit dem Sensor haben jedoch ergeben, dass die Messung der Beschleunigung zu ungenau ist und die notwendige zweifache Integration, um eine Position aus der Beschleunigung zu erhalten, diesen Effekt noch verstärkt. Die Translation

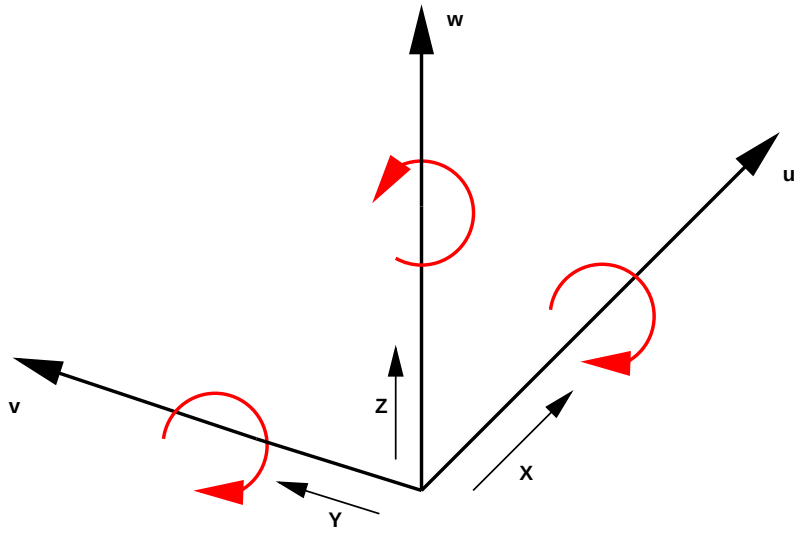
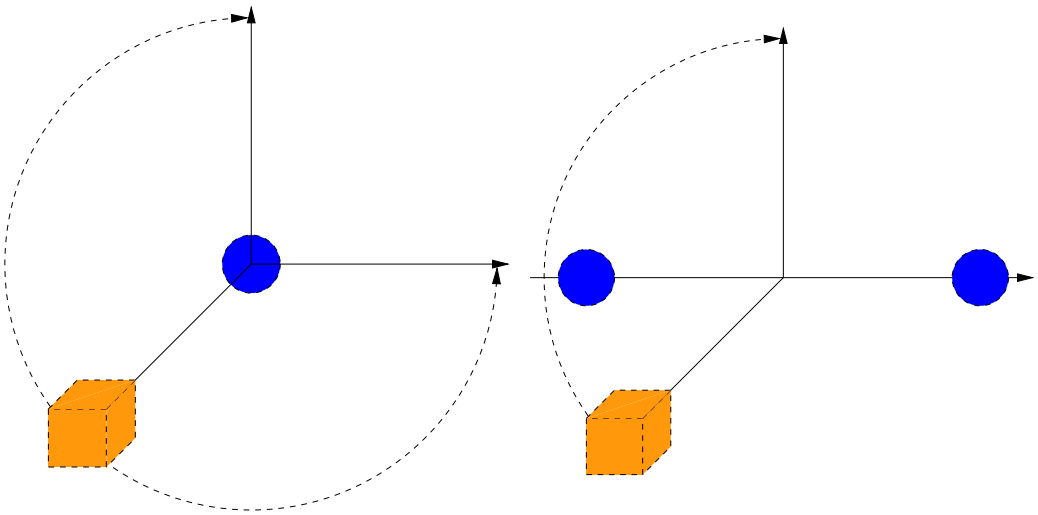


Abbildung 2.9: Freiheitsgrade

driftet nach kurzer Zeit (2-3 Sekunden) um mehrere Meter, da sich der Fehler in der Berechnung akkumuliert. Es ist also notwendig, eine weitere Beschränkung der translatorischen Freiheitsgrade einzuführen. Die Gyroskope scheinen sehr genau zu funktionieren, die Orientierung kann über mehrere Minuten ohne große Abweichung verfolgt werden. Um die Schätzung der Translation mit weiteren Informationen versorgen zu können, werden kugelförmige Marken, die über den zweiten Sensor - die Bildverarbeitung - in das System integriert werden, hinzugefügt. Die Marken sollen anhand ihrer Farbe erkannt werden, sind jedoch untereinander (außer durch ihre Position) nicht unterscheidbar. Eine einzelne Marke kann jedoch noch keine vollständige Korrektur der Position leisten. Über den erkannten Durchmesser der Kugel kann die Entfernung ermittelt werden, das Objekt kann sich aber immer noch auf einem Orbit um die Kugel bewegen (Abb. 2.10(a)). Es ist also nur ein Freiheitsgrad direkt abgedeckt und durch die Bedingung, dass das Objekt auf die Kugel ausgerichtet sein muss, eine Kopplung mit zwei Achsen der Orientierung vorhanden. Das bedeutet, dass für jeden Punkt im Orbit genau eine Orientierung der zwei Drehachsen möglich ist. Um die Achse von Objekt zu Marke kann das Objekt immer noch beliebig gedreht werden. Können zwei Marken zur Positionsbestimmung genutzt werden, liefern diese bereits Informationen über zwei Translationsachsen und zwei Drehachsen, das Objekt kann sich nur noch auf einer Kreisbahn um die beide Marken verbindende Achse drehen. Bei drei nicht in einer Ebene (oder nahezu einer Ebene) liegenden Marken ist eine vollständige Positionsbestimmung über Triangulation möglich. Im vorliegenden System liefert natürlich auch der Ball durch seine modellhaft beschriebene Flugbahn weitere Informationen über die Position des Kamera-IMU Systems.



(a) 2/3 (b) 4/5

Abbildung 2.10: Mit einer einzelnen Marke (a) ist die Bewegungsfreiheit des Sensors auf eine Kugelbahn rund um die Marke beschränkt. Wird der Radius der Kugel ebenfalls hinzugenommen, kann sich die Kamera auch nicht mehr auf ihrer z-Achse bewegen. Mit zwei Marken (b) und Radius werden zwei translatorische und zwei rotatorische Freiheitsgrade abgedeckt.

In der tatsächlichen experimentellen Bestimmung der Position und Orientierung liefern jedoch nicht alle Messungen die gleiche Qualität. So ist die Bestimmung einer Entfernung zu einer Marke über den Radius im Bild sehr schwach (ungenau), während zum Beispiel die Orientierungsmessung mit den Gyroskopen der IMU recht stark (genau) ist. Ein ungefähre Abschätzung der Qualität der Sensoren findet sich in Tabelle 2.3.2. Die Optionen *Orientierung mit einer oder zwei Marken* macht zu der Orientierung nur mit

Freiheitsgrade und Sensoreingabe	**Qualität**
Tanslation mit Accelerometer:	sehr schwach
Translation mit Acc. und 1 Marke:	schwach
Translation mit Acc. und 2 Marken:	stark
Translation mit Acc. und 3 Marken (Triangulation):	sehr stark
Orientierung mit Gyro:	stark
Orientierung mit Gyro und 3 Marken:	sehr stark

Tabelle 2.1: Grob qualitative Abschätzung der Genauigkeit von Messungen zur Abdeckung der Freiheitsgrade.

Gyroskop-Messungen keinen großen Unterschied und ist daher weggelassen worden.

Deutlich wird jedoch, dass die Translation, also die Positionsbestimmung im Wesentlichen „schwach“ ist und daher die Umgebung nach Möglichkeit so gestaltet werden muss, dass immer zwei Marken oder mehr im Bild zu sehen sind.
Mit dieser Abschätzung wird erneut deutlich, warum ein optimaler Schätzer wie der Kalman-Filter oder eben der UKF eingesetzt werden sollte. Der optimale Schätzer integriert alle Messungen und liefert ein optimales Ergebnis auf der Basis ihrer Gewichtung. Es können die Qualitäten der Messungen in der Kovarianz des Fehlers im Prozessmodell (im Falle der IMU-Messungen) oder in der Kovarianz des Fehlers im Messmodell (Marken, Messen von Durchmessern) in den Filter eingebracht werden.

2.3.3 Nicht behandelte Themen

Kalibrierung der Kamera

Für ein Trackingsystem mit einer Kamera müssten auch noch weitere Themen behandelt werden. So ist die Kalibrierung der intrinsischen Kameraparameter und damit die Bestimmung einer Abbildungsgleichung der Kamera von elementarer Bedeutung. Jede Messung mit der Kamera muss mit dieser Gleichung aus dem zwei-dimensionalen Bild in das drei-dimensionale Kamera-Koordinatensystem transformiert werden. Es wird eine quadratische Ausgleichsrechnung auf einem Schachbrettmuster zur Ermittlung der Parameter benutzt, die aber aus einer externen Quelle [32] übernommen werden konnte (Abb. 2.11).

Kalibrierung der Position der IMU

Da der Inertial-Sensor alle Messungen in Körperkoordinaten durchführt, ist es wichtig, seine relative Position zum zweiten Sensor, der Kamera, und dort insbesondere zur optischen Achse zu ermitteln. Die eingangs erwähnten bei dem Workshop *InerVis* vorgestellten Artikel beschreiben verschiedene Methoden eine solche Kalibrierung vorzunehmen. Aus Zeitgründen wurde jedoch kein Verfahren zur Kalibrierung entwickelt und während der Experimente versucht den Fehler gering zu halten. Es findet daher auch keine theoretische Diskussion der Verfahren statt.
Der daraus resultierende Fehler wurde mit einem größeren Messrauschen modelliert, wie auch im Abschnitt 3 nachgelesen werden kann.

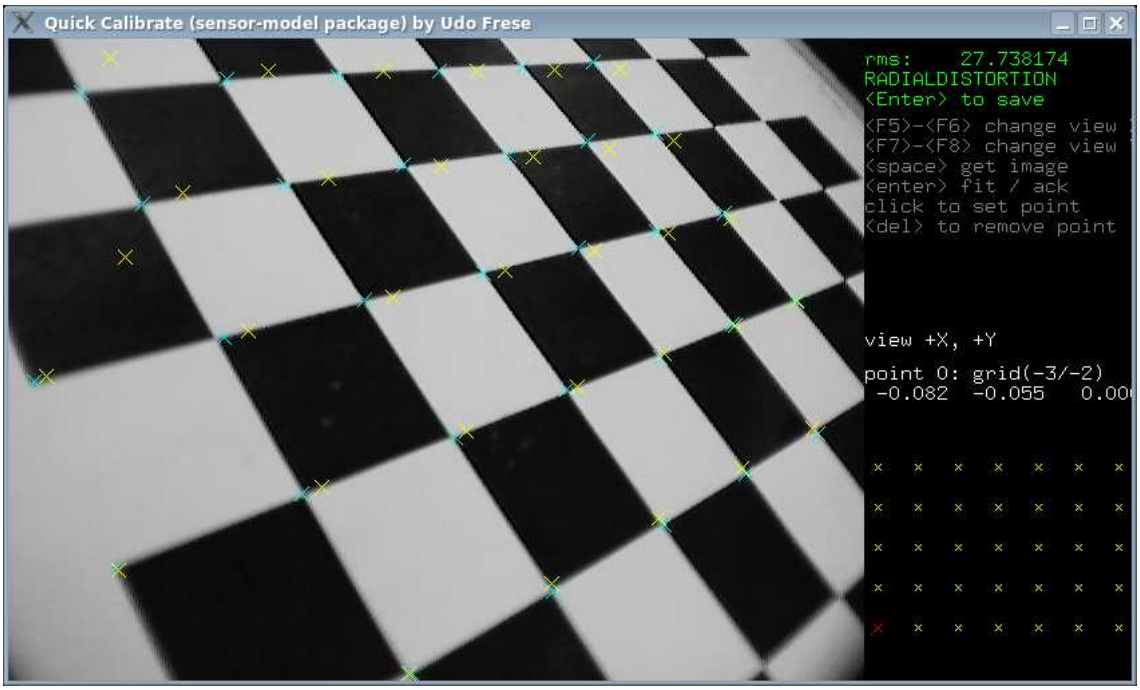


Abbildung 2.11: Quickcalibrate von Udo Frese.

Kapitel 3

Implementierung

Die Implementierung eines Trackingsystems für das Verfolgen einer Ballflugbahn benötigt verschiedene Komponenten: Es werden Programme zur Aquirierung der Daten des Inertial-Sensors sowie eine auf das Problem angepasste Bildverarbeitung benötigt. Das eigentliche Trackingsystem muss diese Daten fusionieren und zur Überprüfung der Ergebnisse eine visuelle Darstellung des geschätzen Zustandes bereitstellen. Die verschiedenen Funktionalitäten wurden bewußt in alleinstehene Programme getrennt, um so Übersichtlichkeit und Flexibilität zu wahren. Die in Abbildung 3.1 dargestellten Komponenten wurden im Rahmen der Arbeit erstellt, auf ihre Funktionalität wird im weiteren Verlauf eingegangen.

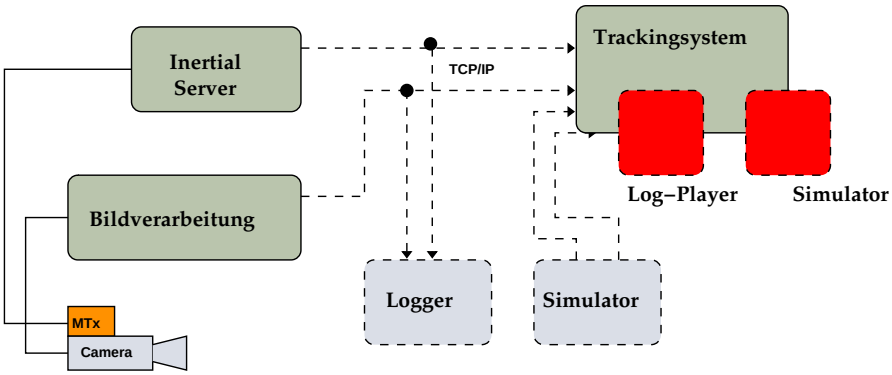


Abbildung 3.1: Komponenten des Trackingsystems. Die roten Kästen, sind Komponenten, die in den Tracker integriert wurden, jedoch ebenfalls eigene Komponenten darstellen. Sie nutzen die bereits vorhandene Umgebung des Trackingsystems zur Visualisierung.

Die Sensordaten werden von zwei Serverprozessen erzeugt, die Zugriff auf die jeweilige Hardware-Komponente haben. Sie schreiben ihre Ausgabe in einen TCP-Stream. Ein

beliebiger Client bekommt diese Daten geschickt, wenn er sich mit dem Server verbindet.

3.1 Trackingsystem

Die Tracking-Komponente implementiert den Unscented Kalman Filter, so wie er im vorherigen Kapitel beschrieben wurde und bereitet die Daten für die Visualisierung auf.
Die Messdaten werden über eine Netzwerkverbindung zu den Hardwareservern oder wahlweise vom internen Log-Player bezogen. Dynamik- und Messschritt sind unabhängig voneinander implementiert. Je nachdem welche Information zur Verfügung steht, wird der passende Teil des Algorithmus ausgeführt.

3.1.1 Parameter, Kovarianzen

Für die Implementierung müssen insbesondere die zu erwartenden Fehler in den Messungen vorab abgeschätzt werden, um daraus geeignete Kovarianzen der Fehler (R_t, Q_t zu erstellen.
Auf der Diagonalen der jeweiligen Matrix findet sich das Quadrat des Fehlers σ^2 für den Mess- oder Dynamikschritt.

Messfehler

Die Messung erfolgt durch eine Kamera mit angeschlossener Bildverarbeitung und anschließender Anwendung eines Kameramodells, welches die zwei-dimensionalen Ergebnisse der Bildverarbeitung in das drei-dimensionale Koordinatensystem der Kamera überträgt. Das Kameramodell konnte mit einer externen Bibliothek implementiert werden und wird so als *Black-Box* behandelt.
Die Genauigkeit der Bildverarbeitung wird in Pixeln angegeben. Eine Messung besteht aus der Bounding-Box und der x,y-Position einer erkannten Kugel (alle relevanten Objekte in der Bildverarbeitung sind kugelförmig). Die Bounding-Box wird zu einem Durchmesser umgerechnet. Die Position im Bild leitet sich aus dem Schwerpunkt der erkannten Region ab (siehe Abschnitt 3.2) und kann als zuverlässig angenommen werden. Daher wird hier eine Abweichung von 2 Pixeln im Bild in jede Richtung angenommen ($\sigma^2 = 4$).
Die Genauigkeit der Bounding-Box ist geringer und der daraus resultierende Durchmesser kann einen etwas größeren Fehler als die Position ausweisen. Eine empirische Schätzung beläuft sich hier auf etwa 3 Pixel Fehler im Durchmesser ($\sigma^2 = 9$). Damit ergibt sich

die Kovarianz des Fehlers Q_t zu

$$Q_t = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{pmatrix} \tag{3.1}$$

Fehler im Prozessmodell

Die Kovarianz des Fehlers im Prozessmodell R_t beschreibt die Zuverlässigkeit des Modells innerhalb eines Aktualisierungsschritts. Auf der Diagonalen werden die geschätzten Fehler als σ^2 angegeben. Es kann nur ein qualitativer und daraus resultierend nur ein geschätzer Fehler der Komponenten angegeben werden. Grundsätzlich kann angenommen werden, dass keine Abhängigkeit der Fehler existiert, sie also unkorreliert sind und daher auf den Nebendiagonalen alle Einträge mit 0.0 festgelegt werden können. Die Positionen von Ball und Kamera hängen mit der Geschwindigkeit zusammen. Würden jeweils beide Zustandsteile mit Fehlern behaftet werden, ist es für den Filter schwierig zu entscheiden, welche Komponente tatsächlich einen Fehler aufweist. Daher werden die Positionsfehler ebenfalls mit 0.0 angegeben und versucht den Fehler über die Änderungen der Geschwindigkeiten zu modellieren. R_t gilt für den gesamten Zustand, kann aber in seine Komponenten unterteilt werden.
Das Dynamik-Modell des Balls ist relativ einfach, liegt aber dennoch im Bereich der wirklichen Bewegung auf einer Flugbahn. Die Information zur Position wird hauptsächlich durch die Messung der Kamera eingeholt, auf die bereits ein Rauschen addiert wird. Es wird daher kein weiterer Fehler im Modell addiert ($\sigma = \sigma^2 = 0$). Die Ballgeschwindigkeit wird mit zwei Messungen zu Beginn des Wurfes initialisiert und kann daher noch stark fehlerbehaftet sein. Es kann also angenommen werden, dass die Geschwindigkeit einen großen Fehler enthält. Es wird von einem Fehler in der Geschwindigkeit von $10\frac{m}{s}$ ausgegangen. Bei einer Update-Rate des Dynamik-Schritt von 100 Hz ergibt sich ein $\sigma^2 = (10 \cdot 0.01)^2 = 0.01$
Die Fehler in der Dynamik der Kamera werden nach ähnlichen Prinzipien ermittelt. Die Position wird ebenfalls nicht mit einem Fehler belegt, alle Fehler werden über die Geschwindigkeit modelliert. Die Geschwindigkeit wird mit einem Fehler von $\sigma^2 = (0.7 \cdot 0.01)^2 = 0.0049$ angeben und entspricht so einem Fehler von 70cm in der Sekunde. Der Fehler der Drehgeschwindigkeit des Sensor setzt sich aus zwei Teilen zusammen: zum einen ist der Sensor selbst ungenau und zum anderen ist keine Kalibrierung der Position relativ zur Kamera vorgenommen worden. Um diesen Fehler ebenfalls für die Auswertung der Experimente zu berücksichtigen, wurde von einen maximalen Fehler in der Ausrichtung von $10^\circ = 0.17\ rad$ ausgegangen. Der maximale Ausschlag des Sensors liegt in den Messreihen bei etwa $0.82\frac{rad}{s}$. Daraus kann man einen Fehler von $\sigma^2 = (0.82 \cdot 0.17 \cdot \sqrt{0.01})^2$ ableiten. Insgesamt ergeben sich die Fehler nach Tabelle 3.1

$\mathbf{R_t(i,i)}$	σ
0	0.0
1	0.0
2	0.0
3	$10.0\frac{m}{s} \cdot 0.01s$
4	$10.0\frac{m}{s} \cdot 0.01s$
5	$10.0\frac{m}{s} \cdot 0.01s$
6	0.0
7	0.0
8	0.0
9	$0.7\frac{m}{s} \cdot 0.01s$
10	$0.7\frac{m}{s} \cdot 0.01s$
11	$0.7\frac{m}{s} \cdot 0.01s$
12	$0.7\frac{m}{s} \cdot 0.01s$
13	$0.7\frac{m}{s} \cdot 0.01s$
14	$0.7\frac{m}{s} \cdot 0.01s$

Tabelle 3.1: Die Diagonale der Fehlermatrix R_t.

3.1.2 Daten-Assoziation

Um zwei Messungen zu einer Abfolge zusammenfassen zu können, ist eine Assoziation einer Messung mit dem aktuellen Systemzustand notwendig.
Die Marken sind untereinander nicht unterscheidbar, jedoch in ihrer Position dem System bekannt und können nach diesem Kriterium zugeordnet werden. Der Ball ist zunächst nur über den Zustand des Filters bekannt, das ist in der Regel aber ausreichend um Phantombälle in der Messung als wahre Messungen auszuschließen.
Es existieren einige Methoden zur Daten-Assoziation, die hier zur Anwendung kommen könnten. Jedoch hat sich gezeigt, dass eine einfache *Nearest-Neighbor*-Assoziation ausreichend ist. Es werden bei den vorhandenen Datensätzen keine Markenmessungen einer falschen Identität zugeordnet. Genausowenig werden Phantombälle als Messung in den Filter integriert. Die verwendete Nearest-Neighbor-Assoziation berechnet den Abstand eines Objektes aus der Messung zu den Position der Marken im aktuellen Zustand. Die Marke mit der kürzesten Entfernung zur Messung wird mit dieser Messung assoziert (siehe Abb. 3.2).

```
NEAREST-NEIGHBOR ALGORITHMUS(z1 : Observation )
1  for i ∈ beacon
2  do
3      z2 ← MEASUREMENT-MODEL(I)
4      d ← √((z1_x − z2_x)² + (z1_y − z2_y)² + (z1_z − z2_z)²)
5      if d < d_min
6        then
7              d_min ← d
8      return d_min
```

Abbildung 3.2: Der Nearest-Neighbor Algorithmus für die Assoziation der Landmarken.

3.1.3 Visualisierung

Um die Funktion des Filters testen zu können und im Verlauf der Entwicklung die Möglichkeit zu haben einzelne Abschnitte der Implementierung visuell überprüfen zu können, wurde eine Anwendung mit grafischer Oberfläche programmiert. Es können die eingehenden Messungen mit dem zugehörigen Kamerabild dargestellt werden (Abb. 3.3, rechts oben). Die dynamischen Informationen des Inertial-Sensor werden in einen Graph geplottet (rechts unten). Der geschätzte Zustand wird in eine 3D-Umgebung eingezeichnet (links oben). Die Umgebung beeinhaltet ebenso die wahren Positionen der Landmarken. Der rote Ball entspricht dem geschätzten Ball zum aktuellen Zeitpunkt t, während die grün eingezeichneten Bälle die Zustände $t + n$ darstellen, bis die z-Koordinate des Balls gleich der z-Koordinate des Bodens plus den Radius des Balls ist. In der Abbildung wird gerade eine aufgezeichnete Messreihe abgespielt, es kann noch das Fenster des *Log-Players* gesehen werden (links mitte). Die Trackingkomponente kann zusätzlich Informationen über ihre Umgebung einlesen um so die Marken im Modell zu positionieren und die Nullposition der Kamera richtig zu initialisieren. Zudem ist es möglich einen Durchlauf eines Experimentes als Folge von Bilddateien zu speichern.

3.2 Bildverarbeitung

Das Trackingsystem benötigt für die Schätzung eine Messung in Form von Positionen und Größen der relevanten Objekte (Marken, Ball) in Bildkoordinaten.
Das Bildverarbeitungssystem wird eigens für diesen Zweck entwickelt und kann daher speziell auf die Aufgabe angepasst werden. Es werden für den experimentellen Aufbau geeignete kugelförmige Marker und ein farbiger Ball verwendet. Eine Extraktion aufgrund der farblichen Klassifizierung erscheint auch für die Forderung nach Echtzeit

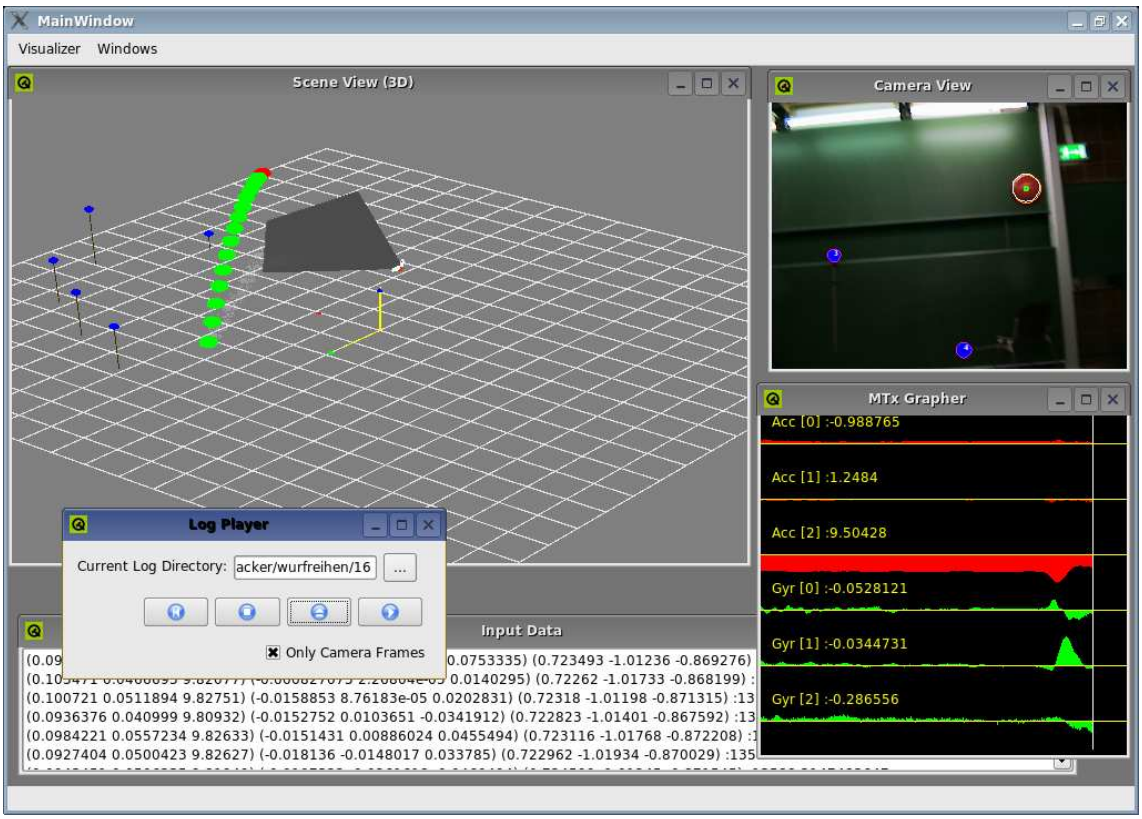


Abbildung 3.3: Die grafische Oberfläche des Trackingsystems. Links oben: Die Darstellung des Zustand in einer OpenGL-Szene. Rechts oben: Das Kamerabild mit darin eingezeichneten Annotationen zu erkannten Objekten.

als am einfachsten zu implementieren, zumal vorangegangene Versuche einer robusten Kreiserkennung auf der Basis einer Hough-Transformation nicht die gewünschten Erfolge erzielt haben.

3.2.1 Verwendete Algorithmen

Die Erfahrung mit der vorhandenen Kamera (Sony DFW-VL500) aus abgeschlossenen Projekten (RoboCup: B-Smart) legt eine Schwellwert-basierte Lösung im YUV-Raum nahe. Für die Aufgabe sollten ein Ball sowie verschiedene Marker erkannt werden. Werden diese im YUV-Raum mit ausreichend Abstand zueinander gewählt, kann eine einfache und robuste Lösung erstellt werden. Geplant war eine Klassifizierung von drei verschiedenen Farben: Blau und gelb für die Marker und orange für den Ball. Im Experiment wächst jedoch die Unsicherheit nie soweit, dass Marken tatsächlich verwechselt werden

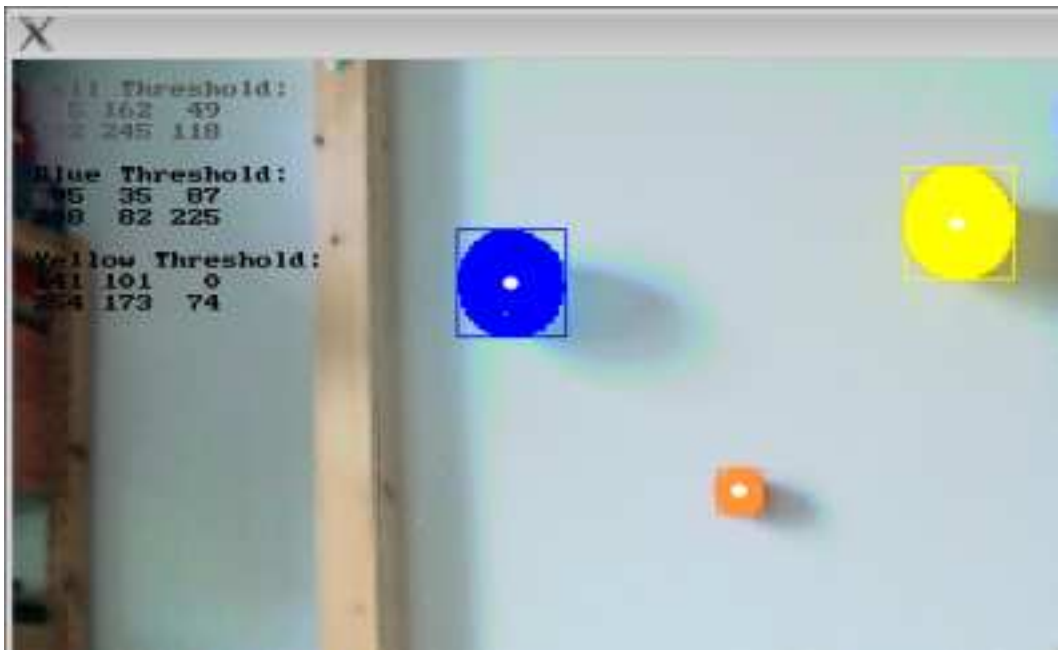


Abbildung 3.4: Interface zur Bildverarbeitung: Links sieht man die Einstellungen für die Thresholds der Farben im Y, U, und V Kanal. Es werden drei Objekte erkannt. Die übergemalten gelben, blauen oder orangen Flächen visualisieren das zugrundeliegende RLE-Bild. Die rechteckigen Kästen mit weißem Punkt in der Mitte beschreiben Position und Ausdehnung der erkannten Objekte.

können, es genügt demnach nur eine Sorte Landmarken zu verwenden. Es werden blaue Marker benutzt, die bei wechselnden oder schwierigen Lichtverhältnissen etwas robuster erkannt werden. Die Software ist jedoch in der Lage drei Farben zu klassifizieren.

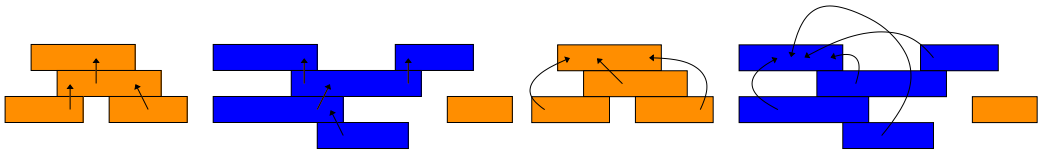
RLE-Komprimierung

Im ersten Verarbeitungschritt wird jedes Bild nach den drei gewählten Farben klassifiziert und dann als *Run-Length Encoded* Bild gespeichert.
Zur Klassifizierung wird für die gewünschten Farben ein Würfel im YUV-Raum aufgespannt. Liegt der Wert einer Farbe innerhalb dieses Würfels gehört er zu der Farbklasse. Die Größe des Würfels wird über einfache Thresholds in der Benutzungsoberfläche der Bildverarbeitung eingestellt. Der Nutzer bekommt sofort ein visuelles Feedback (Abb. 3.4), ob die Klassifizierung (Segmentierung) ein gewünschtes Ergebnis erzielt.
In den RLE-Bildern werden zusammenhängende Regionen mittels *Union-Find* gefunden und mit einer Pfad-Komprimierung (*path-compression*) zur besseren Verarbeitung in Sequenzen zusammengefasst. Der *Union-Find*-Algorithmus verbindet alle zusammenhän-

```
UNION-FIND()
 1  ip1 ← BEGIN(RLE)
 2  ip2 ← BEGIN(RLE)
 3  while ip1 < end(RLE)
 4  do
 5     while ip2.y < ip1.y − 1 ∨ (ip2.y == ip1.y − 1 ∧ ip2.xHigh < ip1.xLow)
 6     do ip2 ← NEXT-INTERVALL()
 7        if ip2.y == ip1.y − 1 ∧ ip2.xLow <= ip1.xHigh
 8           then
 9                if ip2.cc == ip1.cc
10                   then UNITE-REGIONS(IP1, IP2)
11                if ip2.xHigh > ip1.xHigh
12                   then ip1 ← NEXT-INTERVALL()
13                   else ip2 ← NEXT-INTERVALL()
14           else ip1 ← NEXT-INTERVALL()
```

Abbildung 3.5: Union-Find Algorithmus für mehrere Farbklassen in einem RLE-Bild. Ein RLE-Intervall hat die Attribute *xLow* für den Beginn eines Intervalls, *xHigh* für das Ende, *y* für die Zeile und *cc* für die Farbklasse, dem das Intervall zugeordnet ist.

genden Intervalle zu Regionen. Dabei wird mit zwei Zeigern $ip1$ und $ip2$ durch die RLE-Intervalle gelaufen. Wenn sich zwei Intervalle überschneiden, wird in *unite-regions* das `parent`-Attribut des Intervall gesetzt, gleichzeitig wird eine Pfadkomprimierung durchgeführt, die alle Intervalle einer Region genau einem Elternintervall zuordnet und nicht nur den *Parent* auf das direkt berührte Intervall setzt (siehe auch Abb. 3.6(b)), so ist das Auffinden des obersten Elementes einer Region einfacher und schneller.



(a) Run-Length-Encoding: Union-Find. Sich berührende Intervalle einer Farbklasse werden zusammengefasst.

(b) Run-Length-Encoding: Path-Compression. Alle Intervalle einer Region werden einem Parent zugewiesen.

Abbildung 3.6: Schritte zur Regionenerkennung.

Extrahieren von geeigneten Blobs

Die im vorangegangenen Schritt erkannten Regionen werden nun auf ihre Eigenschaften überprüft. Insbesondere werden die Momente $0.$ und $1.$ Ordnung ausgerechnet, um Fläche und Schwerpunkt zu bestimmen. Das Moment 0. Ordnung ergibt sich aus dem Integral (oder im diskreten Fall der Summe) über x und y, wenn $f(x, y)$ die Zugehörigkeit eines Pixels zur gewünschten Region beschreibt ($f(x, y) = 1$ für Pixel der Region $f(x, y) = 0$ für Pixel, die nicht zu der Region gehören).

$$m_{(0,0)} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \tag{3.2}$$

Das Moment 1. Ordnung läßt sich in ein Zeilen- und Spaltenmoment aufteilen, deren Kombination die Berechnung der Schwerpunkte erlaubt.

$$m_{(1,0)} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot x \tag{3.3}$$

$$m_{(0,1)} = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) \cdot y \tag{3.4}$$

$$\bar{x} = \frac{m_{(1,0)}}{m_{(0,0)}}, \qquad \bar{y} = \frac{m_{(0,1)}}{m_{(0,0)}} \tag{3.5}$$

Eine Reihe von weiteren *Constraints* wird schließlich geprüft, um sicherzustellen, dass eine Region auch das gewünschte Objekt sein kann. Dazu gehören Höhen- und Breitenverhältnis, da die relevanten Objekte im Bild alle rund erscheinen, können stark rechteckige Objekte ausgeschlossen werden. Besonders kleine Flächen können zum Beispiel durch Rauschen im Kamerabild entstehen, diese verworfen.

Verschicken der Informationen

Die Informationen über erkannte Objekte werden zusammen mit einem Zeitstempel von der Aufnahme des Bildes in einem String kodiert und an die angeschlossenen Netzwerkclients verschickt.
Bei Bedarf kann auch eine Bilddatei des aktuellen Kameraframes gespeichert werden, um diese später im Trackingsystem, zum Beispiel beim Abspielen eines Logs wieder anzuzeigen.

3.3 Anbindung an den Inertial-Sensor

Der Inertial-Sensor (IMU) liefert die Messungen über einen Seriell-USB-Konverter an den USB-Port des Rechners. Xsens stellt ein Software Development Kit (SDK) für Windows bereit, sowie zum Zeitpunkt der Entwicklung der Anbindung des MTx eine binäre, proprietäre Bibliothek für Linux. Da die Implementierung mit Linux als Basissystem stattfinden sollte, kam nur die mitgelieferte Bibliothek in Frage. Es stellt sich jedoch schnell heraus, dass auch diese nicht für die Entwicklung geeignet war. Der Binärblob der Bibliothek setzte veraltete Compiler voraus und unterlag einer sehr restriktiven Lizenz, die eine Weiterverbreitung der Software zu einem späteren Zeitpunkt erschwert hätte. Die Dokumentation des MTx (IMU) ist jedoch hervorragend, so dass eine eigene Implementierung des Protokolls, welches auf der seriellen Schnittstelle gesprochen wird, möglich war. Aus Zeitgründen wurde jedoch nur eine Untermenge der Funktionen des Trackers implementiert, jedoch ist der Server um weitere Teile des Protokolls erweiterbar.
Der Sensor kann eigenständig Algorithmen auf den gemessenen Daten anwenden und so zum Beispiel eine Orientierung in Weltkoordinaten als Messung herausgeben. Für die Anwendung in der hier vorliegenden Software sind jedoch die tatsächlichen Messwerte der internen Sensoren, Gyroskop und Accelerometer, notwendig.
Aus technischer Sicht ist die serielle Schnittstelle nur ein Bytestrom. Pakete müssen demnach selber zusammengebaut und geparst werden und nach Möglichkeit mit einer Checksumme überprüft werden. Der MTx liefert vor jedem Paket eine Premable von einem Byte und danach die ID des Gerätes. Falls mehr als ein Sensor am Bus benutzt wird, können diese über die BID identifiziert werden (Abb. 3.7). Dann folgt die Message-ID,

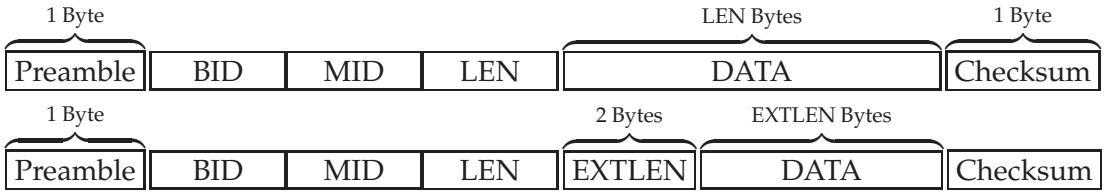


Abbildung 3.7: Die Variationen der Datenpakete des MTx: Oben findet sich ein normales Paket mit Preamble, Geräte ID, Message ID sowie den Daten (Payload). Am Ende wird noch ein Byte als Checksumme herangehängt. Unten das gleiche Paket mit vergrößertem Payload und einem Extrafeld für die Länge.

die beschreibt, welche Daten sich im Payload befinden. Es kann sich um tatsächliche Messdaten handeln, als auch um Steuer-, Bestätigungs- oder Einstellungsdaten. Ist der Payload länger als 254 Bytes, wird das LEN-Feld auf 255 gesetzt und ein EXTLEN die Länge des Datenfeldes mit 2 Bytes angegeben.

Wenn der Sensor startet, wartet er zunächst für 500ms im *Wake-up*-Zustand. Wird keine Nachricht geschickt, wechselt das Gerät direkt in den letzten Messzustand. Es werden dann je nach eingestellter Messfrequenz die Daten auf die serielle Schnittstelle geschrieben und können am Computer ausgelesen und geparst werden. Es kann mit einer Nachricht jederzeit in den Konfigurationsmodus gewechselt werden, um beispielsweise ein *Reset* einzuleiten oder die Messfrequenz zu ändern.
Für die Anwendung werden die kalibrierten (temperatur-kompensierten) Sensordaten benötigt. Ein solches Paket ist im Datenfeld, wie in Abbildung 3.8 dargestellt, organisiert. Ein solches Datenpaket wird als String verpackt und an angemeldete Clients verschickt.

Abbildung 3.8: Das Datenfeld eines Paketes mit den kalibrierten Messungen. Neben den Messwerten der Sensoren, wird auch ein fortlaufender Zeitstempel (TS) mitgeschickt, der es erlaubt, die Pakete zu sortieren und fehlende Pakete zu erkennen.

3.4 Simulator

Da der experimentelle Aufbau zur Verfolgung von geworfenen Bällen sehr aufwendig ist und wegen der notwendigen Wiederholbarkeit von Messreihen, wurde ein Simulator implementiert, der die Schnittstellen der Bildverarbeitung und des Intertial-Sensors emuliert.
Der Simulator wurde mit der Skriptsprache Python und einer Python-Anbindung an die *Open Dynamics Engine* (ODE) [33] programmiert. Es ist möglich eine Beschreibung der Umgebung und der in der Szene enthaltenen Objekte in einer XML-Datei zu spezifizieren. Es wurde ein einfaches Kamera-Modell implementiert, so dass aus der dreidimensionalen Umgebung ein Kamerabild abgeleitet und diese Information über die Netzwerkverbindung an das Trackingsystem gesendet werden kann.

Kapitel 4

Ergebnisse

4.1 Experimente

Zum Testen der implementierten Algorithmen wurde einige Testreihen aufgezeichnet und mit der Software verarbeitet. Da die Implementierung bisher ohne Optimierungen nicht echtzeitfähig ist, wurden während der Experimente nur Daten aufgezeichnet, die im *Log-Player* der Anwendung beliebig häufig abgespielt werden können.

4.1.1 Aufbau und Durchführung

In einem Raum wurde die Marken bestehend aus blau angeprühten Styroporkugeln sinnvoll verteilt und ihre Position zu einem festgelegten Koordinatenursprung ausgemessen. Es wurde ein Startpunkt für das Kamera-Inertial-System festegelegt und ebenfalls ausgemessen. Diese Positionen wurde in die Software eingegeben und bilden so die Grundlage für die Umgebung des Modells.
Nach der Kalibierung der Kamera und Einstellen der Parameter zur Farbklassifizierung wurde die Kamera von einer Person von ihrer Startposition aufgenommen und eine weitere Person hat den Ball entlang einer sinnvollen Bahn vor den Marken geworfen. Die Person mit der Kamera versuchte den Ball im Kamerabild zu halten, indem sie das System der Flugbahn des Ball hinterher führte.

4.1.2 Probleme

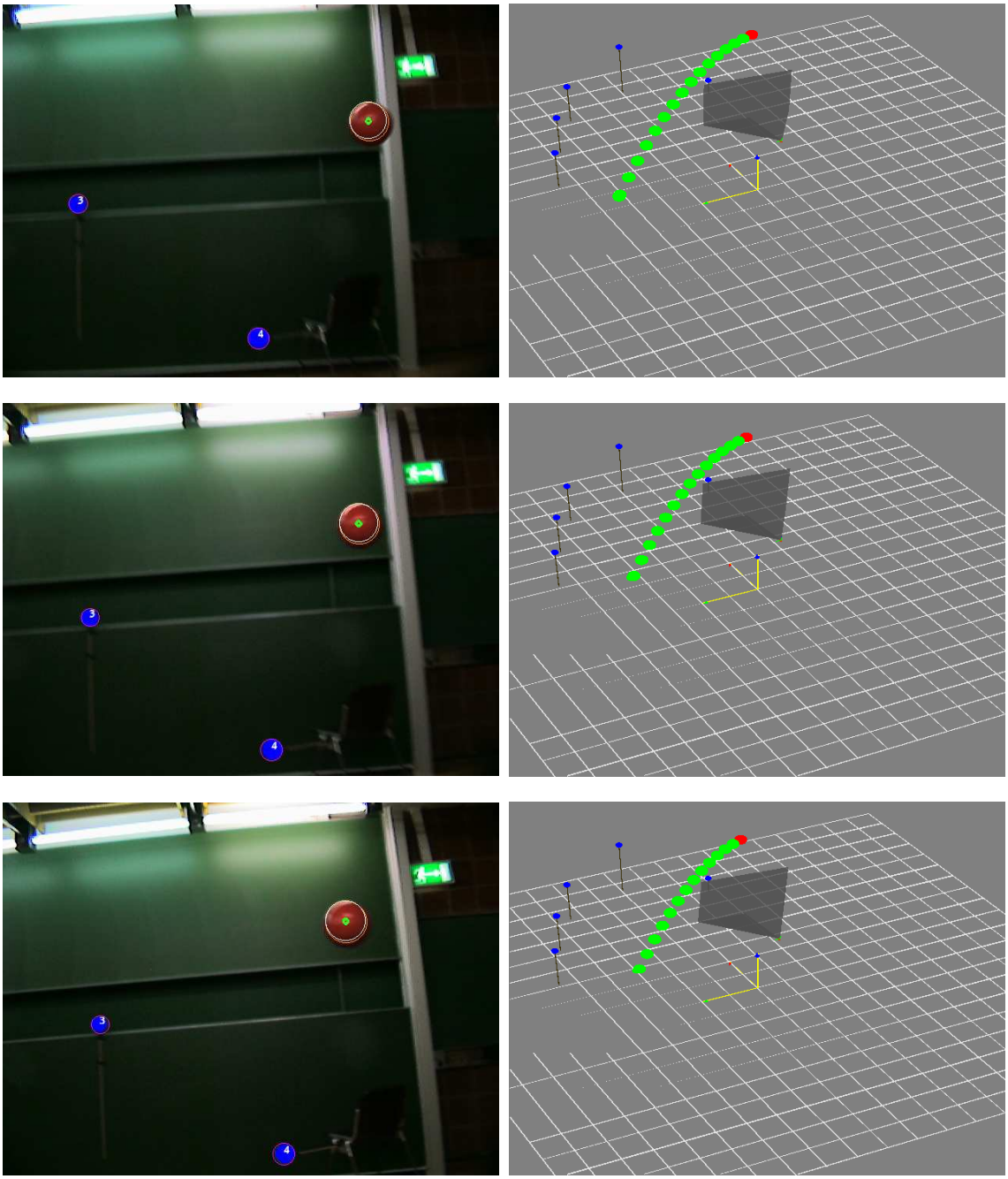
Für eine gut funktionierende Bildverarbeitung sind optimale Lichtverhältnisse von Vorteil. Das Labor in dem die Experimente durchgeführt wurden, verfügt über keine Fenster, was den Vorteil hat, dass eine gleichmäßige Beleuchtung vorhanden ist. Die Leuchtstoffröhren des Raumes haben aber nur eine geringe Lichtintensität und werden über

ein pulsierendes Signal mit Strom versorgt. Durch die relative Dunkelheit musste die Shutter-Zeit der Kamera relativ lang gewählt werden, um noch ein ausreichendes Bild zu erlangen. Das führt jedoch zu einer Verwischung von schnellen Bewegungen im Bild. Hier musste ein Kompromiss zwischen guter Erkennung der Farben und klar abgegrenzter Formen der Marken und des Ball gefunden werden.
Die Startposition der Kamera wurde markiert und ausgemessen, allerdings weist der Kamerakörper nur eine gerade Fläche auf (Stativansatz), so dass sich die Kamera auf ihrer Startposition nicht 100%ig wiederholbar positionieren ließ. Durch Hilfskontruktionen wurde versucht diesen Fehler zu minimieren.
Auf die nicht vorhandene Kalibrierung der Lage der Kamera zum Inertial-Sensor ist bereits eingegangen worden. Auch hier war eine Ausrichtung nicht zuletzt wegen fehlender gerader Kanten an der Kamera schwierig.
Die Experimente konnte dennoch mit einen großen Grad an Erfolg durchgeführt werden, wie der nächste Abschnitt zeigt.

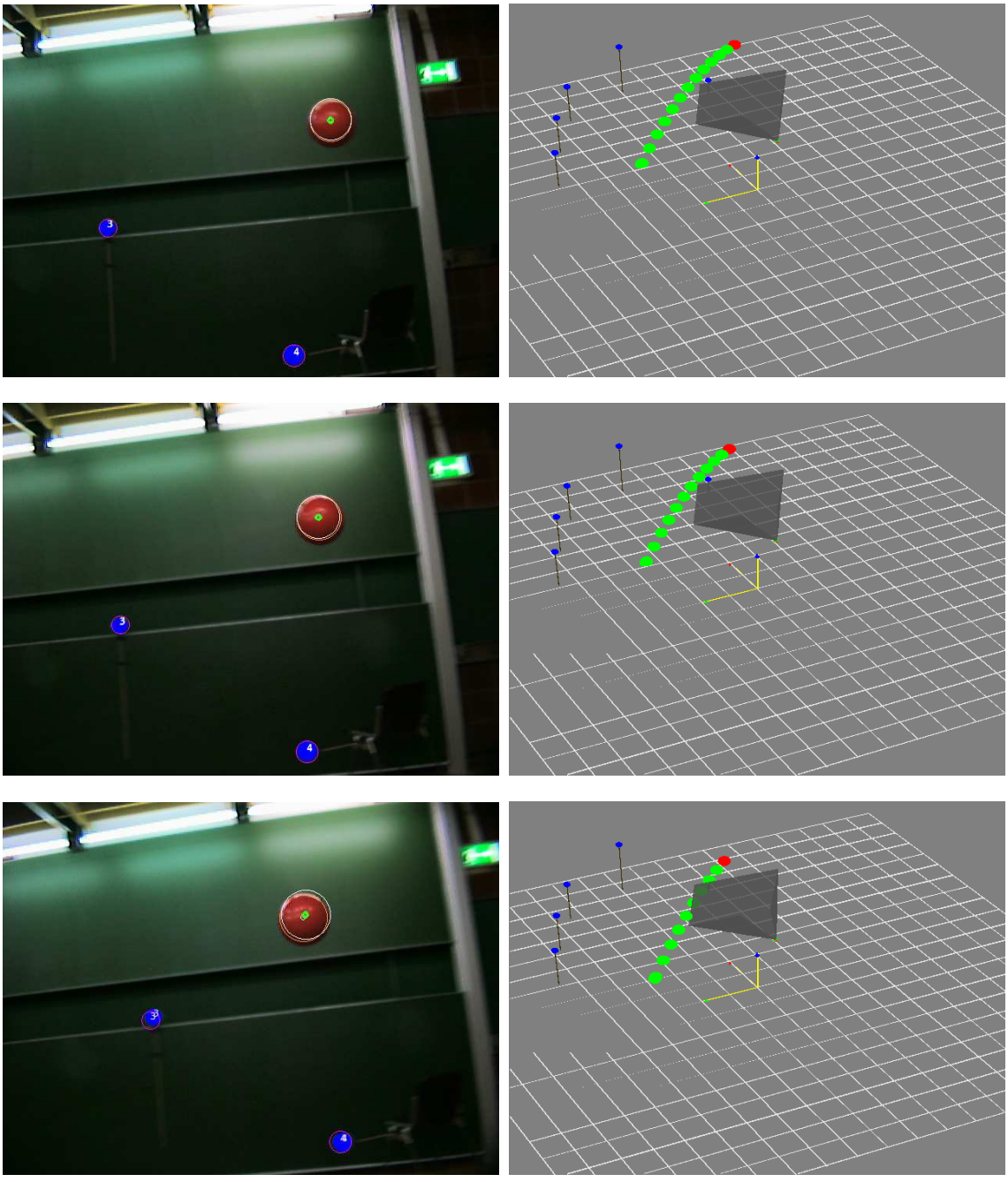
4.2 Visuelles Ergebnis

Die folgende Bildreihe zeigt die Vorhersage der Flugbahn eines geworfenen Balls. Links ist das aktuelle Kamerabild zu sehen, inklusive der Annotierungen der Bildverarbeitung und des Algorithmus. Die Marken, wie sie die Bildverarbeitung erkennt sind gleichmäßig blau übermalt. Die Schätzung der Position der Marken ist mit einem magentafarbenen Kreis versehen. In den meisten Fällen liegen diese korrekt übereinander.
Auf der rechten Seite findet sich passend zu dem Kamerabild, eine dreidimensionale Darstellung des geschätzen Zustands. Der rote Ball, stellt die aktuelle Schätzung des Balls dar, während in grün die Schritte bis zum Auftreffpunkt markiert sind. Der graue Kegel beschreibt den Öffnungwinkel der Kamera und visualisiert so die Blickrichtung der Kamera. Am spitzen Ende des Kegels befindet sich der Sensor. Das gelbe Koordinatensystem beschreibt die Weltkoordinaten.

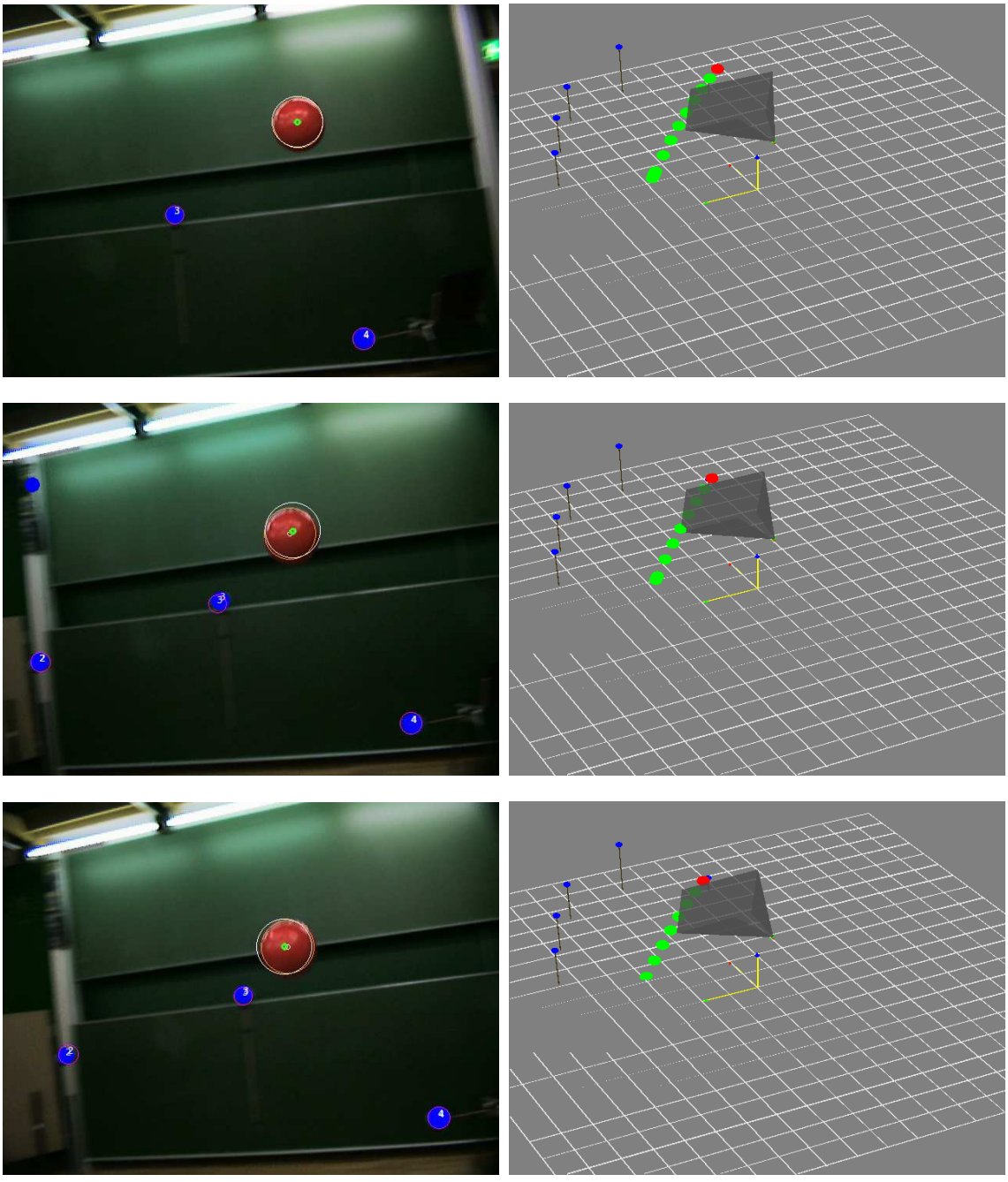
4.2.1 Bildfolge



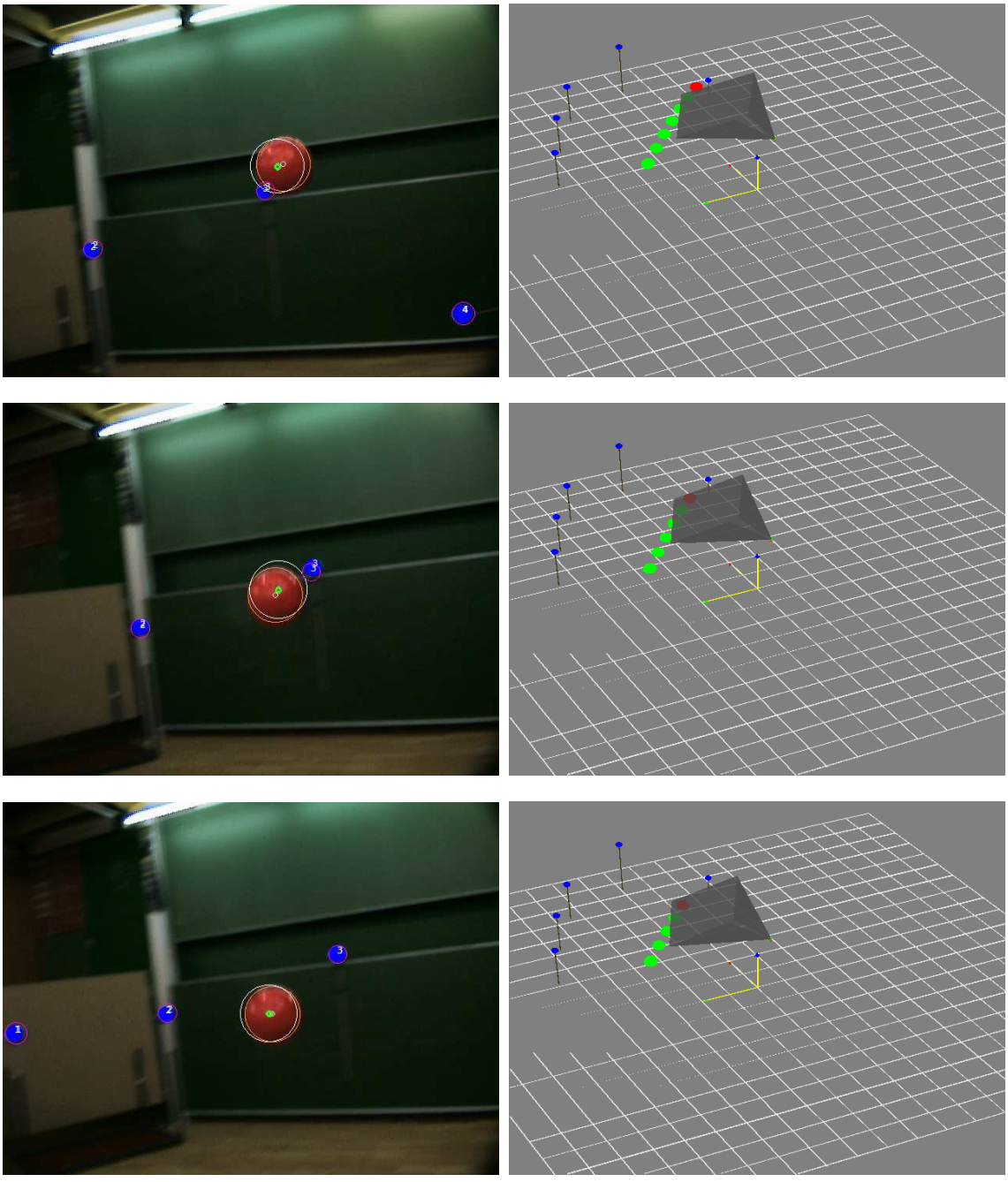
Die ersten drei Messungen der Kamera werden noch benötigt, um den Zustand insbesondere die Geschwindigkeiten von Ball und Inertialsensor zu initialisieren. Die Vorhersage der Flugbahn springt stark, der Auftreffpunkt ist noch nicht korrekt.



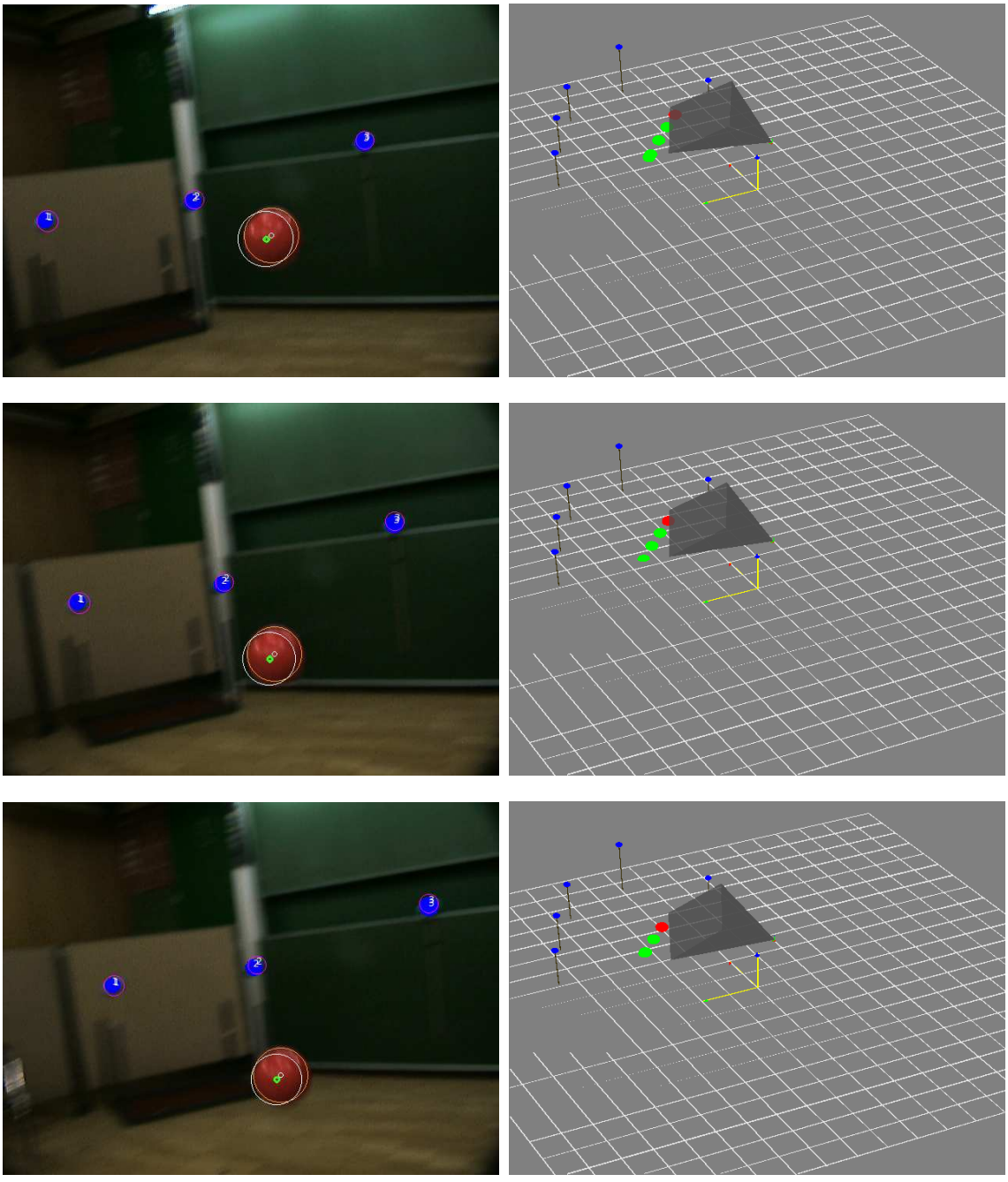
In den nächsten drei Bildern verändern sich die Flugbahn und der Auftreffpunkt kaum noch. Der prädizierte Auftreffpunkt pendelt in der Genauigkeit innerhalb einer Fläche von etwa 60×60 cm. Ein Quadrat in der Visualisierung entsprechen einer Fläche von 50×50 cm.

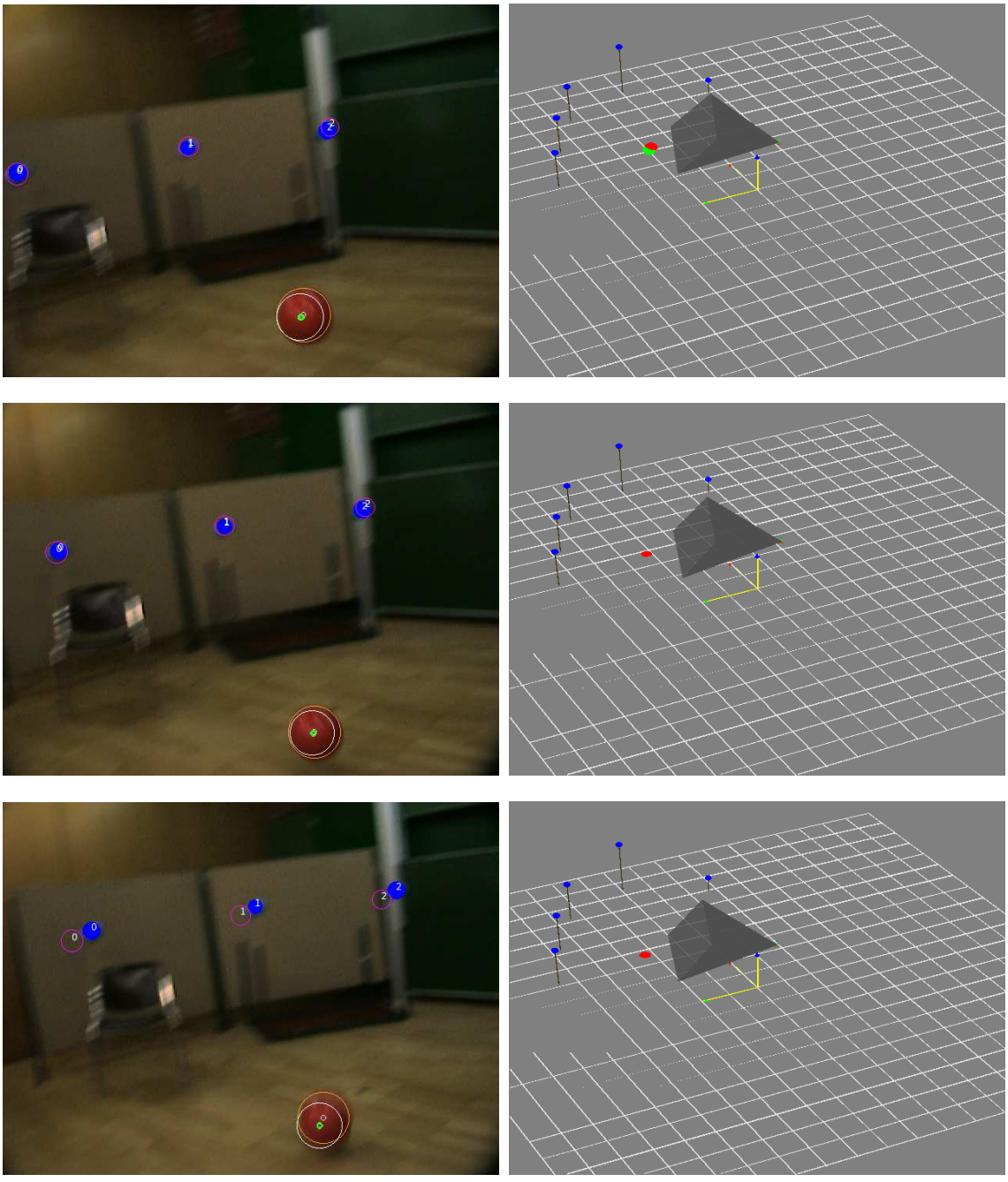


In den letzten beiden Bildern kann im Kamerabild schön gesehen werden, wie der Filter die Schätzung der drei Marken anpasst. Im vorletzten Bild sind die magenta-farbenden Kreise noch etwas neben den erkannten Marken, im nächsten Bild wurde die Schätzung bereits angepasst. Außerdem sieht man im mittleren Bild ein Beispiel für die funktionierende Daten-Assoziation: Die Marke links oben im Bild wird ignoriert.



In der Bildfolge kann auch schön erkannt werden, dass der Ball und die Marken trotz der relativ schlechten Lichtverhältnisse immer gut erkannt werden.





Schließlich kommt der Ball am Aufftreffpunkt an.

4.3 Zusammenfassung und Ausblick

Es konnte gezeigt werden, dass die Komponenten *Bilderverarbeitung* und *Tracking mit einem Unscented Kalman Filter* für ein frei bewegliches Kamera-Inertial-System angewendet werden können und sich mit diesem System die Flugbahn eines Balls vorhersagen läßt.
Es wurden einige Anpassungen des Filter vorgenommen, die eine algorithmische Behandlung der Mannigfaltigkeit der Orientierung erst möglich machen. Die korrekte Funktion dieser Anpassung wurde durch die Implementierung bestätigt.
Die implementierte Bildverarbeitung ist simpel, konnte aber im Experiment durch ihre Robustheit überzeugen.
Die Experimente konnten zeigen, dass das System in der Lage ist seine Aufgabe zu erfüllen. Jedoch läßt sich rückblickend sagen, dass die die Durchführung der Experimente noch zu viele Unsicherheiten offen gelassen hat. So ist es praktisch schwierig die Startposition der Kamera immer exakt einzuhalten. Die Lage von Sensor zu Kamera muss kalibriert werden. Der relativ große Fehlerwert für die Orientierung in der Fehlermatrix R_t könnte dadurch minimiert werden. Die Biase der Sensoren müssten sorgfältig zur Zeit des Experiments bestimmt werden. Damit ließe sich ein konstanter Fehler des Sensors praktisch eliminieren.
Die Fehlerterme in R_t sind in der vorliegenden Anwendung häufig empirisch mit theoretischen Einflüssen ermittelt worden. Diese Fehler könnten aber zum Beispiel über die *offline* Anwendung anderer Verfahren (Least-Square-Algorithmen), genauer bestimmt werden. Auch analytisch lassen sich die Fehlerterme sicherlich noch exakter bestimmen. Würden diese Aspekte in weiteren Experimenten sorgfältig beachtet, könnte sich sicherlich die Genauigkeit der Vorhersage deutlich verbessern.
Die nächsten Schritte für die Weiterentwicklung des Systems könnten folgende sein:

- Kalibrierung der Lage von Sensor zu Kamera.
- Echtzeitfähigkeit.
- Austauschen der künstlichen Landmarken durch echte Marken in einer Fußballumgebung (Linien, Tore, Ecken).
- Montage des Systems am Helm eines menschlichen Spielers. Das Bewegungsprofil wird hier nochmal deutlich anders sein als bei der Nachführung von Hand.
- Weiteres Fein-Tuning der vorhandenen Parameter insbesondere die Streuung der Sigma-Punkte (vielleicht lassen sich noch Terme höherer Ordnung modellieren) und Erstellung der Kovarianz des Fehler R_t

Literaturverzeichnis

[1] Hiroaki Kitano and Minoru Asada. Robocup humanoid challenge: That's one small step for a robot, one giant leap for mankind. In *IEE/RSJ International Conference on Intelligent Robots and Systems 1998 (IROS '98)*, pages 419–424, 1998.

[2] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997. ACM Press.

[3] RoboCup Federation. RoboCup-Homepage, 2007. `http://www.robocup.org`.

[4] Thomas Röfer, Tim Laue, and Dirk Thomas. Particle-based self-localisation using landmark and directed lines. Technical report, Universität Bremen, 2005. `http://www.informatik.uni-bremen.de/kogrob/papers/rc06-selfloc.pdf`.

[5] Luca Marchetti, Giorgio Grisetti, and Luca Iocchi. A comparative analysis of particle filter based localization methods. Technical report, Universitá „La Sapienza", 2006. Extended Version from Robocup Symposion 2006.

[6] T. Röfer, R. Brunn, H.-D. Burkhard, I. Dahm, U. Düffert, K. Engel, D. Göring, J. Hoffmann, M. Jüngel, M. Kallnik, M. Kunz, M. Lötzsch, A. Osterhues, S. Petters, M. Risler, C. Schumann, M. Stelzer, O. von Stryk, M. Wachter, and J. Ziegler. GermanTeam RoboCup 2004, 2004. `http://www.germanteam.org/GT2004.pdf`.

[7] Hendrik Johannes Luinge. *Inertial Sensing of Human Movement*. PhD thesis, Universiteit Twente, 2002.

[8] Xsens. `http://www.xsens.com/download/metro_thialf_speedskating.pdf`, visited 2.2.2007.

[9] fourcc.org. `http://www.fourcc.org/yuv.php#YUY2`, 03 2007.

[10] Jorge Lobo and Jorge Dias. Relative pose calibration between visual and inertial sensors. In *ICRA Workshop InerVis*, 2004.

[11] Peter Lang and Axel Prinz. Calibration of hybrid vision / inertiral tracking systems. In *ICRA Workshop InerVis*, 2004.

[12] Pierre Chalimbaud, François Berry, François Marmoiton, and Serge Alizon. Design of a hybrid visuo-inertial smart sensor. In *ICRA Workshop InerVis*, 2004.

[13] Leopoldo Armesto, Josep Tornero, and Markus Vincze. Fast ego-motion estimation with multi-rate fusion of inertial and vision. In *ICRA Workshop InerVis*, 2004.

[14] Wikipedia. Wikipedia: Tracking. `http://de.wikipedia.org/wiki/Tracking`, visited 11.3.2007.

[15] Weisstein, Eric W. „Euler Angles." From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/EulerAngles.html`, 2 visited 1.3.2007.

[16] Sir William Hamilton. Lecture on quaternions: A new mathemtical method, 1853. `http://historical.library.cornell.edu`.

[17] R. Mukundan. Quaternions: From classical mechanics to computer graphics, and beyond. In *Asian Technolgy Conference in Mathematics*, 7th, 2002.

[18] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[19] Wikipedia. Markov-chain. `http://en.wikipedia.org/wiki/Markov_chain`, visited 1.3.2007.

[20] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[21] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. Academic Press Inc., 1979.

[22] Wikipedia. Jacobian. `http://en.wikipedia.org/wiki/Jacobian`, visited 2.3.2007.

[23] S. Julier and J. Uhlmann. A new extension of the Kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls, Orlando, FL*, 1997.

[24] A. Doucet, S. J. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and Computing*, 2000.

[25] Dieter Fox, Wolfram Burgard, and Frank Dellaert. Monte carlo localization: Efficient position estimation for mobile robots. In *AAAI*, 1999.

[26] Wikipedia. Mannigfaltigkeit. http://de.wikipedia.org/wiki/Mannigfaltigkeit, visited 28.2.2007.

[27] Udo Frese and Lutz Schröder. Theorie der sensorfusion: Skript zur vorlesung. http://www.tzi.de/~ufrese/teaching/tds06/tds06skript.pdf, visited 2.3.2007.

[28] Edgar Kraft. A quaternion-based unscented kalman filter for orientation tracking, 2003.

[29] João Luís Marins, Xiaoping Yun, Eric R. Bachmann, Robert McGhee, and Michael Zyda. An extended kalman filter for quaternion-based orientation estimation using marg-sensors. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[30] Joseph J. La Viola. A comparision of unscented and extended kalman filtering for estimating quaternion motion. *Advanced Scientific Computing and Visualization*, unknown.

[31] Xavier Pennec. Computing the mean of geometric features: Application to the mean rotation. INRIA: Rapport de Recherche, 03 1998.

[32] Udo Frese. Quickcalibrate: Software zur ermittlung von kamera-parametern. not published, 2007.

[33] Russel Smith. Www: Open dynamics engine. http://www.ode.org, visited 2007.