Jonglieren mit der Kinect

ein Softwareprojekt im Projekt Bildverarbeitung

Projektbericht

Rolf Boomgaarden
Florian Letsch
Thiemo Gries

11. April 2014

Unter Aufsicht von: **Benjamin Seppke**
Arbeitsbereich Kognitive Systeme
Fachbereich Informatik, Universität Hamburg

Inhaltsverzeichnis

1 Einleitung

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed aliquam, ligula vitae condimentum malesuada, turpis nisi placerat eros, vel facilisis mi neque quis nulla. Aenean eleifend risus id dolor ultricies scelerisque. Phasellus venenatis libero enim, vel lacinia massa interdum nec. Quisque a euismod ligula. In eget mattis orci. Integer vitae enim ac nisl scelerisque luctus ut et nibh. Quisque ut odio ultrices, consequat mi vel, accumsan metus. Donec faucibus, nulla vel mattis euismod, felis leo accumsan tortor, et congue turpis leo et elit. Proin gravida mollis facilisis. In enim nisi, pellentesque id tincidunt a, accumsan eget elit. Aliquam erat volutpat. In quam ante, accumsan eu est a, molestie euismod neque. Proin porta rhoncus nisl sed dignissim. Aliquam lacinia sed libero et eleifend. Ut placerat tortor eget augue pellentesque rutrum.

2 Motivation

Mit den technischen Möglichkeiten eines Tiefen- und Bilddaten liefernden Systems (konkret: Microsoft Kinect) soll in dieser Arbeit versucht werden, das Wurfmuster eines mit Bällen jonglierenden Akteurs zu analysieren.

Ein Jongleur wirft Jonglierbälle in einem Muster, das möglichst gleichmäßig ist. So ist der Höhepunkt der Flugbahn idealerweise konstant auf der gleichen Höhe. Zum Analysieren des Jongliermusters wäre dies also bereits ein erstes Kriterium, die *Güte eines Jongliermusters* automatisiert zu bewerten.

Denkbar sind auch weitere Anwendungen, wie etwa das automatische Zählen von erfolgreich gefangenen Würfen. Eine computergesteuerte Erfassung der insgesamten Wurfzahl ist ein einfaches Kriterium für eine *Leistungsbewertung des jonglierenden Benutzers*.

Die genaue Anwendung ist jedoch nicht Ziel dieser Arbeit. Stattdessen verfahren wir in einem bottom-up Herangehen, um von den rohen Bild- und Tiefendaten der Kinect ausgehend Informationen über sich im Bild befindliche Objekte (Jonglierbälle) zu erfassen und deren Bewegung zu erkennen. Das Ergebnis ist dann ein Fundament, auf dessen Grundlage konkrete Anwendungen entwickelt werden können.

3 Zielsetzung

Am Ende dieser Arbeit soll eine Anwendung stehen, die mit Hilfe der Kinect Daten über die Flugbahnen dreier jonglierter Bälle liefert.

Ein Akteur befindet sich hierbei im Bildzentrum in einem wohl definierten Abstand zur Kinect. Es werden drei matte Bälle beliebiger Farbe jongliert. Um das Ergebnis unabhängig von der Szenenbeleuchtung zu halten, sollen die Tiefendaten ausreichend Information für das eindeutige Identifizieren der Bälle liefern.

4 Möglichkeiten der Kinect

Die Kinect ist eine von Microsoft zur Spielekonsole Xbox 360 vertriebene Erweiterung, die den Spieler mit einem RGB- und einem Tiefensensor erfasst und diese beiden Datenströme an die Konsole liefert. Da die Kinect über einen USB-Anschluss verfügt, kann sie an konventionellen Rechnern angeschlossen und betrieben werden. Eine quelloffene Implementierung zur Unterstützung der Kinect ist das freenect Projekt, das für Linux, Windows und MacOS zur Verfügung steht und im Rahmen dieser Arbeit als Bibliothek für Python verwendet wurde. FIXME: Quellen.

Die Videoquelle der Kinect liefert standardmäßig 30 (JA?) Bilder pro Sekunde mit einer Auflösung von 640x480 und 8 bit Farbtiefe. Die Tiefendaten stammen von einer Infrarot-Kamera und liefern bei gleicher Bildfrequenz 2048 verschiedene Tiefenwerte (11bit). Aus der Funktionsweise der Infrarotkamera ergibt sich, dass die Kinect in Umgebungen starker Infrarotstrahlung (beispielsweise im Tageslicht) nur beschränkt einsatzfähig ist.

FIXME: Erklären wie das mit diesem projizierten Punktemuster funktioniert.

5 Recherche: Ein jonglierender Roboter

Im Vorfeld der ersten eigenen Implementierungsversuche sind wir bei der Paper-Recherche auf eine Arbeit von FIXME gestoßen, die zumindest in Teilen eine ähnliche Aufgabenstellung verfolgte. Unter dem Titel "Playing catch and juggling with a humanoid robotüntersuchte die Gruppe des Disney Research Center (FIXME) einen humaoiden Roboter, der auf ihn zugeworfene Bälle fangen und zurückwerfen soll. In der Entwicklung dieses Systems war ein Teil der Gesamtaufgabe ein bild- und tiefendatenverarbeitendes System, das mit einer der Kinect sehr ähnlichen Kamera arbeitete.

Kernidee dieses Systems war eine *Image Processing Pipeline*, also eine Kette von Verarbeitungsschritten, die als Eingabe eine Folge von RGB- und Tiefendaten nahm und als Ausgabe die aktuelle Position und zukünftige berechnete Flugbahn liefern.

FIXME: Processing Pipeline

Die gelöste Aufgabe in dieser Arbeit ist also von der Grundidee her also eine ganz ähnliche, weshalb wir den Grundaufbau der Verarbeitungsschritte auch in unserem Vorgehen übernehmen wollten. Wie wir im Laufe der Programmierung aber feststellten, hatte die Arbeit einige Rahmenbedingungen anders gesetzt, so dass wir auf Probleme stießen, die sich in der Arbeit mit dem Roboter offensichtlich nicht so deutlich gezeigt haben.

Hierbei ist zuerst zu nennen, dass die betrachteten Bälle bei uns sehr viel kleiner waren, da wir einen tatsächlich jonglierenden Menschen betrachtet haben. In der Arbeit mit dem Roboter wurden sehr große Bälle verwendet, die von den werfenden Menschen mit beiden Händen gehalten und geworfen wurden.

Hierdurch ergeben sich auch längere Wurfbahnen, die tendenziell auch weiter auseinander liegen. Wie später in der Arbeit zu sehen sein wird, ist das normale Jongliermuster mit zwei Händen teilweise so klein, dass es schwer wird, dicht aneinander vorbei fliegende Bälle voneinander zu unterscheiden.

Zusätzlich wurde in der Arbeit mit dem Roboter auf verschiedenfarbige Bälle zugegriffen, um diese voneinander zu unterscheiden. Dies ist eine Rahmenbedingung, die wir so nicht wählen wollten, da die RGB-Werte, die die Kinect liefert, sehr stark von den Beleuchtungsbedingungen abhängen und bei den verschwischten Objekten, wie wir sie in den Kinect-Aufnahmen sehen, keine robuste Erkennung zu ermöglichen versprechen.

6 Lösungsidee

image Processing Pipeline

7 Umsetzung

7.1 Programmstruktur

Die ersten Tests, die wir mit der Kinect und bildverarbeitenden Verfahren ausprobiert haben, bestanden aus wenigen Schritten und wurden in einer simplen Schleife gelöst, die bei Druck der ESC Taste verlassen wurde. Zu Beginn eines Schleifendurchlaufes wird ein neuer RGB Frame und die zugehörigen Tiefendaten geholt. Danach folgt schrittweise Verarbeitung dieser Daten je nach gewünschtem Zweck.

FIXME: Einfacher Programmablauf, Blockdiagramm? Nur wenn wir noch Platz brauchen

Nun möchten wir je nach betrachtetem Arbeitsschritt aber verschiedene Verarbeitungsstufen ausführen oder ausklammern, so dass wir irgendwie eine Parametrisierung finden mussten. Wir haben uns für ein Konzept von Filtern entschieden, wobei zu Programmstart eine Liste mit gewünschten Filtern erstellt wird und diese in der Hauptschleife nacheinander ausgeführt werden (die Originaldaten werden also in den ersten Filter gegeben, das Ergebnis dieses Filters wird als Eingabe für den zweiten Filter verwendet und das Ergebnis des letzten Filters in der Liste wird dann als Gesamtausgabe visuell dargestellt.

Die RGB- und Tiefendaten werden vom freenect Modul als numpy Arrays zurückgegeben. Die Ausgabe erfolgt über ein von OpenCV erzeugtes Fenster, welches Daten für OpenCV erwartet. Die numpy Arrays müssen also an einer Stelle umgewandelt werden. Zusätzlich haben wir schnell festgestellt, dass einige gewünschte Operationen entweder nur in numpy oder nur in OpenCV zur Verfügung stehen. Ein mehrfaches

Umwandeln von einem Format in das jeweils andere ist aus Performanz-Gründen natürlich zu vermeiden. Um in der Entwicklung nicht immer darauf achten zu müssen, welcher Filter welche Eingabe erwartet und welche Ausgabe liefert, haben wir uns dazu entschieden, jeden Filter als Eingabe Daten in numpy Darstellung zu liefern und auch für die Ausgabe numpy zu erwarten. Dies erlaubt uns eine von außen identische Betrachtung der Filter, auch wenn einige Filter intern eine Umwandlung durchführen müssen. Bei etwaigen Problemen wollten wir die Performanz unserer Lösung getrennt betrachten, um zu Beginn lediglich über eine funktionierende Lösung nachdenken zu müssen.

Waren die Filter ursprünglich als isolierte Einheiten mit simplem Input-Output-Verhalten von Bild- bzw. Tiefendaten gedacht, fiel uns schnell auf, dass einige Filter Zusatzinformationen liefern, die von anderen Filtern gebraucht werden. Um Filter nicht intern Unteraufrufe von von anderen Filtern oder Komponenten ausführen zu lassen (dies wäre ebenfalls eine denkbare Lösung), entschieden wir uns für ein Objekt, das von der Hauptkomponente des Programms in jeden Filter hineingereicht wird und in dem jeder Filter Informationen ablegen und abfragen kann. Natürlich ist dies abhängig von der Reihenfolge der Filter und einige Filter haben als implizite Vorbedingung, dass andere Filter bereits ausgeführt wurden, dies haben wir der Einfachheit halber aber nicht expliziert formuliert, im Zweifelsfall wird beim ersten Ausführen einer nicht korrekten Filterkombination oder -reihenfolge ein Laufzeitfehler geworfen, da Informationen in dem übergebenen Objekt noch nicht vorhanden sind. Dieses Objekt ist ein schlichtes Python dictionary.

FIXME: Diagramm Programmstruktur (so cool skizziert, nicht ganz formell UML)

FIXME: bisschen Python Listing um diese Filter-Konzepte und das args dictionary zu zeigen

7.2 Programmfluss

7.2.1 Schritt 1: Tiefendaten vorverarbeiten

Normieren aux XXX Tiefenwerte

7.2.2 Schritt 2: Regions Of Interest isolieren

Hintergrund entfernen, Bälle freistellen. Zwei Ansätze:

A. keine Objekte auf Tiefenebene zwischen Spieler und Kinect, auch nicht am Rand. Tiefenwerte aber einem gewissen Wert einfach abschneiden. Annahme: Spieler steht auf Linie oder ähnlich. Tiefendaten binarisieren.

B. Mit temporalem Filtering sich bewegende Regionen isolieren. Erlaubt auch störende Objekte wie Stühle am Rand, Erfahrung aber nicht so gut, da das Verfahren bei schneller Bewegung (Ballwurf) nicht zuverlässig ist. Außerdem Probleme mit unscharfen

Objekträndern und Rauschen. Außerdem technische Hürden (Vigra als weitere Abhängigkeit).

Bewertung: Ansatz A völlig ausreichend für unsere Zwecke. Einschränkung der Spielerpositition nicht störend, da dies sogar interkativ durchgeführt werden (so lange nach vorne gehen, bis System vernünftige Werte liefert - kein aufwändiges Abmessen nötig).

7.2.3 Schritt 3: Bälle in Frame-Folgen einander zuordnen

Kurze Vorverarbeitung: Rechtecke erkennen aus binarisiertem Bild. Annahme: der Mittelpunkt jedes Rechtecks ist ein Kandidat für eine Ballposition. Die Ausdehnung und somit der Ballradius werden vorerst ignoriert.

Dies ist der aufwändigste Schritt wie sich herausgestellt hat, zumindest der, mit dessen Lösung wir die meiste Zeit verbracht haben.

Erste Idee: Regionen mit Tiefenbild bestimmen, tatsächliche Bälle von Händen etc unterscheiden, indem Kreise in den RGB Bildern gesucht werden. Dies war aber rechenaufwändig und wegen Bewegungsunschärfe sehr unzuverlässig (auch noch unterschiedlich stark je nach Fortschritt des Ballwurfs).

Problemquellen:

- Hände sind auch als Rechtecke enthalten
- Bälle fliegen sehr nah, teilweise überschneiden sich die Rechtecke zweier Bälle, so dass nur ein großes zu sehen ist und als eine mögliche Ballposition untersucht wird
- Ball legt in einem Frame (1/30 Sekunde) unterschiedlich lange Strecken zurück, teilweise sehr große (Pixelanzahl angeben?)
- Mindestabstand zur Kinect resultiert in kleinem Jongliermuster, das verstärkt die problematischen Faktoren
- teilweise fehlt eine Region in einem erkannten Frame

Ansätze:

Konsumierende Ansätze, feste Ballanzahl:

1. Nächste Punkte in zwei aufeinander folgenden Frames werden als der identische Ball aufgefasst. Nicht so zuverlässig, vor allem wegen schneller Ballbewegung und nah aneinander fliegender Bälle. Schwierig auch, wenn ein erkannter Ball fehlt -> Beachtung von ßpringenden"Bällen.
2. Verbesserungsansatz: erwartete Ballposition wird approximiert mit vorheriger Bewegung (linearer Bewegungsvektor). Teilweise besser, aber schwierig, die initiale Bewegung zu Erkennen, auch weiterhin Probleme mit Lücken in den Informationen.

3. Verbesserung: Nicht linear, sondern Flugbahn vorberechnen. Linearer Bewegungsvektor wird als Tangente an Steigung der Wurfparabel zu Grunde gelegt. (Verbesserung nochtmal gut angucken, aber gefühlt hat das erstaunlich wenig unterschied gebracht)

Nicht konsumierende Ansätze, variable Ballanzahl:

1. wenn langsame Aufwärtsbewegung in aufeinander folgenden Frames erkannt wird: als Beginn eines Wurfes auffassen und an dieser Stelle einen Ball mit identischer Geschwindigkeit starten und dessen Flugbahn ab dort schrittweise simulieren. In jedem Schritt mit aktuell vorhandenen Bällen abgleichen und Wurfparameter anpassen. (hier schrittweise Bilder zeigen: Feuerwerk etc)

7.2.4 Schritt 4: Bereinigte Wurfparabel

Das fehlt uns noch. Aber aus den gelieferten Daten wollen wir dann höher-levelige Informationen abstrahieren. Objektanzahl, Wurfhöhen, Würfe zählen, etc.

7.3 Erläuterung verwendeter Bildverarbeitungsverfahren

7.3.1 Kalman Filter

FIXME: alles überarbeiten, mehr, nochmal nachlesen, wie's wirklich ist, Schaubilder

Der Kalman Filter kann sich bewegende Objekte beobachten und Schätzungen zur aktuellen oder auch zukünftigen Position machen.

In diesem Projekt wird er dazu verwendet, die Bälle zu beobachten und die weitere Flugbahn zu bestimmen. Damit kann eine voraussichtliche Flugbahn in die Ausgabe gezeichnet werden, aber auch in der internen Verfolgung und Zuordnung der Bälle spielen die Ergebnisse eine wichtige Rolle.

Der Kalman Filter besteht aus zwei Funktionen, einem `predict` und einem `update`. Im `predict` wird mit Informationen zur Art der Bewegung und mit mindestens einer Ortsinformation zum Zeitpunkt `t` eine Schätzung zum Ort im Zeitpunkt `t+1` gemacht. Ein `predict` kann beliebig oft hintereinander ausgeführt werden.

Im `update` wird die aktuell gespeicherte Ortsinformation mit extern gewonnen Daten aktualisiert. Die hier gemessene Abweichung zwischen Schätzung und tatsächlichen Daten kann natürlich auch zur weiteren Schätzung eingebracht werden. Ein `update` wird nicht mehrmals hintereinander ausgeführt.

7.4 Herausforderungen

Probleme aus den Ansätzen noch mal aufgreifen. Noch irgendwas abstrakteres dazu schreiben? Vielleicht dass wir uns nicht doll genug getracked haben dier Projektzeit

über?

7.5 Bewertung der Umsetzung

Robustheit.

Effizienz. Speedup-Möglichkeiten?

Anwendungsrelevanz.

8 Anwendungsmöglichkeiten

Projekte nehmen als Grundlage für einfache Programme / Spiele.

- Objekte zählen
- Würfe zählen
- ... ?

9 Fazit

Lerneffekt, Frustration, Bewertung des Endprodukts

Quellen

[1] Paul Viola, Michael Jones,
Robust Real-time Object Detection
Vancouver, Canada, 13.07.2001.
http://research.microsoft.com/en-us/um/people/viola/Pubs/Detect/violaJones_IJCV.pdf

[2] Ole Helvig Jensen,
Implementing the Viola-Jones Face Detection Algorithm
IMM-M.Sc.: ISBN 87-643-0008-0 ISSN 1601-233X
Technical University of Denmark, Informatics and Mathematical Modelling
Kongens Lyngby, Denmark, 2008.
http://www.imm.dtu.dk/English/Research/Image_Analysis_and_Computer_Graphics/Publications.aspx?lg=showcommon&id=223656