Diplomarbeit

Portierung und Test von libsift

Evaluierung des Einsatzes zur Roboternavigation

Steffen Planthaber
15544886

Universität Bremen
Fachbereich 3
Informatik

Prüfer:
Prof. Dr. Frank Kirchner
Dr. Udo Frese

12. November 2006

Zusammenfassung

SIFT steht für "Scale-Invariant Feature Transform". Der Algorithmus kann in Bildern markante Punkte erkennen und wiedererkennen. Er kann genutzt werden, um zum Beispiel Panoramabilder aus mehreren Aufnahmen automatisch zu zusammenzufügen. Im Rahmen dieser Diplomarbeit wurde eine bestehende Bibliothek, die den Algorithmus zur Verfügung stellt, von C# nach C++ portiert. Anschließend wurden diverse Tests durchgeführt, die im Ergebnis eine Verwendbarkeit für die Roboternavigation belegen.

Inhaltsverzeichnis

1 Einleitung

SIFT- Features sind markante, wiedererkennbare Punkte in einem Bild. Die Wiedererkennung in anderen Bildern soll dabei unabhängig von Rotation, Größe und Helligkeit der Objekte im Bild funktionieren. Die Features sind aufgebaut aus einer Position im Bild und einem sogenannten "Descriptor". Der Descriptor beinhaltet die Daten für die Wiedererkennung. Diese Daten werden aus einer Umgebung um die Position des Features berechnet und bestehen aus den Richtungen der in dieser Umgebung gefundenen Kanten. Durch die Art der Vorverarbeitung der Bilder können Positionen gesucht werden, die Descriptoren ergeben, die später nur schwer mit denen anderer Features verwechselt werden können.

Bei der Wiedererkennung von Features in einem zweiten Bild werden die normierten Descriptoren als Vektoren betrachtet. Die Abstände zwischen dem gesuchten und allen im dem zweiten Bild vorkommenden Features werden berechnet. Das Feature, dessen Descriptor dem des gesuchten am dichtesten, also am ähnlichsten, ist wird zunächst als Übereinstimmung angenommen, muss aber nicht Korrekt sein.

Sind alle diese Übereinstimmungen gefunden, werden solche entfernt, die mehr als ein Mal dem selben Feature zugeordnet wurden. Es können aber immer noch falsche Zuordnungen zwischen Features existieren. Man kann diese aber entfernen indem man, mittels der unterschiedlichen Positionen der einander zugeordneten Features, Bewegungsmodelle der Kamera berechnet und dann die Zahl der dazu passenden Übereinstimmungen zählt. Es werden nur die Übereinstimmungen als korrekt betrachtet, die zu dem besten gefundenen Modell passen.

1.1 Beitrag der Arbeit

Mit dieser Arbeit wurde eine Grundlage geschaffen, Entscheidungen zu treffen ob die Nutzung SIFT-Features für eine Aufgabe sinnvoll ist.

In den Experimenten werden die Invarianzen der Features gegenüber Perspektivenänderungen, Größe, Helligkeit und Rotation getestet und ausgewertet. Auch wird gezeigt, dass eine Navigationsaufgabe, bei der alle Invarianzen benötigt werden, mittels des Verfahrens gelöst werden kann.

Durch eine Portierung der SIFT- Bibliothek von C# nach C++ wurde die Möglichkeit geschaffen Betriebsystemunabhängig die Algorithmen zur Berechnung, Wiedererkennung und Filterung von SIFT- Features zu verwenden.

1.2 Stand der Forschung

Im Folgenden werden drei Teilbereiche betrachtet. Zunächst wird ein Überblick über den allgemeinen Einsatz von Kameras in der Navigation gegeben (*Visuelle Navigation*). Anschließend wird die Methodik beim Berechnen und Erkennen von Bildmerkmalen vorgestellt (*Erkennung von Bildmerkmalen*) um dann auf die Anwendungsgebiete der Wiedererkennung von Bildmerkmalen einzugehen (*Nutzung von Bildmerkmalen*).

1.2.1 Visuelle Navigation

Man unterteilt die autonome Navigation mit Kamerainformationen in zwei Bereiche: "in-" und "outdoor- Navigation", die sich durch die Gegebenheiten stark voneinander unterscheiden.

Im outdoor Bereich geht es hauptsächlich um autonome Fahrzeuge für Straßen oder die Raumfahrt. Im Straßenbereich werden Kameras häufig zur Erkennung der Straße und ihres Verlaufs genutzt, so dass die Roboter der Straße folgen können. Es werden häufig Textur- und Farbinformationen verwendet [DK02].

Ein großes Problem für die farbbasierten Ansätze ist die Beleuchtung der Umgebung. Dunklere Regionen der Straße, die durch Wolkenschatten hervorgerufen werden, können als Hindernis erkannt werden. Straßen haben abhängig von Bewölkungsgrad und Jahreszeit unterschiedliche Farben, so dass sie bei solchen unterschiedlichen Gegebenheiten nicht zuverlässig erkannt werden können.

Das autonome Fahrzeug STANLEY (ein VW Touareg) [TMD+06] löst dieses Problem indem die Farbinformation für die Wegerkennung per Kamera auf den Wegfarben des Geländes direkt vor dem Fahrzeug beruht, welches in der Reichweite der Laserscanner lag. Wenn dieses gescannte Gelände flach ist, wird die dort aufgefundene Farbe als Farbe der Straße angenommen. Das optische Verfahren wurde verwendet um den Wegverlauf auf längere Strecke zu erkennen als es nur mit den Laserscannern hätte geleistet werden können. Die benutzten Laserscanner hatten nur eine maximale Reichweite von 25 Metern – bei hohen Geschwindigkeiten nicht ausreichend. 2005 gelang es STANLEY eine Strecke von 175 Meilen in der Mojave Wüste (USA) in unter 7 Stunden ohne menschliche Eingriffe zurückzulegen, das entspricht einer Durchschnittsgeschwindigkeit von 19,1 Meilen pro Stunde (ca. 30 km/h), die Höchstgeschwindigkeit betrug 38 Meilen pro Stunde (ca. 61 km/h). Der Weg war durch GPS-Koordinaten (Global Positioning System) vorgegeben, Hauptaufgabe der Kamera war die lokale Navigation, also die Segmentierung des Weges sowie die Erkennung von Hindernissen auf dem Weg.

In komplett unstrukturierten Umgebungen, wie zum Beispiel auf fremden Planeten, stellen sich andere Probleme. Da es keine Straßen und Wege gibt, müssen Zielpunkte gesetzt werden und sich diesen unter Hindernisvermeidung angenähert werden. Es müssen Landmarken gefunden werden, anhand derer der Roboter seine Position berechnen kann. Diese können weit entferne Bergspitzen oder lokale Landmarken sein, wobei die Position meist mittels Triangulation [Jae05, Kap.8.2] errechnet wird.

Innerhalb von Gebäuden gibt es andere Anforderungen. Landmarken

wie Bergspitzen oder GPS sind meist nicht verfügbar oder zu ungenau. So muss die Lokalisierung des Roboters auf eine andere Weise geschehen. Man unterscheidet drei Kategorien: kartenbasierte, kartenaufbauende und kartenlose Navigation.

Bei der kartenbasierten Navigation hat der Roboter eine vorgegebene Karte, auf der er sich mittels seiner Sensoren lokalisiert. Bei der kartenaufbauenden Methode erstellt der Roboter während der Exploration eine eigene Karte aufgrund seiner Sensorinformationen. Die kartenlose Navigation kommt ohne explizite Rauminformationen aus und basiert auf Bewegungen, die aus erkannten Obkjekten oder Landmarken generiert werden [DK02]. Zum Beispiel kann ein Roboter sich einer erkannten Landmarke annähern bis ein bestimmter Abstand erreicht wurde und dann eine Aktion ausgeführen. Der Roboter hat in diesem Fall kein Wissen über seine Umgebung

Innerhalb von Gebäuden werden häufig Ecken und Kanten oder künstliche Markierungen zur visuellen Lokalisierung und Navigation verwendet. Zum Beispiel kann die Position berechnet werden, wenn drei Markierungen an einer Wand sind und diese ein Dreieck bilden. Dann kann die relative Position zu den Markierungen aus den Winkeln und Kantenlängen dieses Dreiecks berechnet werden. Auch kann ein 3D Modell des Gebäudes zur Lokalisierung verwendet werden, wenn man Kanten erkennt und diese versucht in dem 3D Modell wiederzufinden.

Es existieren auch Graphenbasierte Ansätze, dabei repräsentieren Knoten Abzweigungen und Kreuzungen von Fluren. Die Zustandsübergänge stellen zum Beispiel Korridore dar, denen Landmarken wie Türen oder andere Objekte zugeordnet sind. Durch die Landmarken ist eine genaue Positionsbestimmung in dem Korridor oder die Zustandsbestimmung im Graphen möglich [DK02],[WKBH00].

1.2.2 Erkennung von Bildmerkmalen

Bei der Erkennung von Bildmerkmalen gibt es unterschiedliche Algorithmen, alle beruhen aber auf drei gleichen Schritten:

1. Finde Regionen oder Punkte, die als Merkmal geeignet sind
2. Berechne beschreibende Elemente der Markmale
3. Finde die Merkmale in anderen Bildern

Es gibt unterschiedliche Verfahren bei der Suche der Merkmalkandidaten (der Bildpunkte, aus denen ein Merkmal berechnet werden soll). Diese Auswahl ist sehr wichtig für die Qualität der Merkmaldescriptoren [MS05].

Die Spanne der unterschiedlichen Merkmale reicht von einfachen, wie Geraden oder Kanten, über ganze Bildregionen (zum Beispiel die Form einer Region mit gleicher Farbe) bis hin zu einzelnen Pixeln mit einer Umgebung. Die Merkmale unterscheiden sich hauptsächlich in der Art, wie ihr Descriptor aufgebaut ist. Einfache Descriptoren nutzen lediglich Farb-, andre nutzen komplizierte Informationen. SIFT nutzt als Descriptor Richtungsinformationen der gefundenen Kanten im Bild.

Um das Wiederfinden der Merkmale in anderen Bildern zu ermöglichen, muss deren Descriptor vergleichbar sein. So kann man bei einem

Vergleich mit den Descriptoren der Merkmale aus einem anderen Bild den finden, der dem des gesuchten Merkmales am ähnlichsten ist. Eine schnelle Methode ein Merkmal zu finden ist die Nutzung eines kd- tree [Ben75], ein mehrdimensionaler Suchbaum, der auch in der SIFT- Bibliothek verwendet wird.

1.2.3 Nutzung von Bildmerkmalen

Bildmerkmale werden für unterschiedliche Aufgaben verwendet, zum Beispiel zum Berechnen von Panoramabildern, zur Erkennung und Wiedererkennung von Objekten oder zur Navigation.

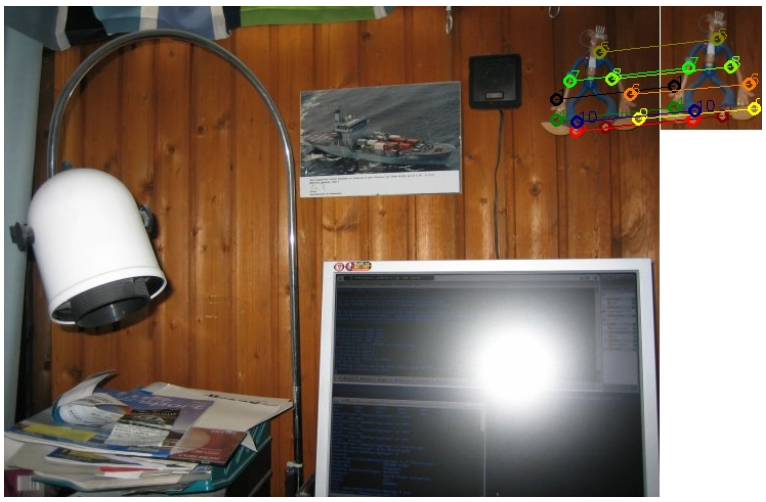


Abbildung 1: Objektsuche mit SIFT- Features

Objekterkennung kann auf der Suche von Merkmalgruppen beruhen, die zu einem Objekt gehören (Clustering). So kann, wenn die Merkmale in gleicher oder ähnlicher Zusammenstellung wieder auftauchen, das Objekt wiedererkannt werden. T. Nathan Mundhenk et al. [MNM+04] benutzen Merkmale wie Richtung, Intensität und Farbe um dann Merkmalcluster zu berechnen, die Objekte in Bildern beschreiben. So ist es möglich diese Objekte in anderen Bildern wiederzufinden.

Auch featurebasierte Navigation wurde bereits erfolgreich umgesetzt. Mit einfachen Features wie Kanten, die für die Exploration eines Raumes verwendet wurden [NBL03]. Oder mit SIFT- Features, bei denen die in dieser Arbeit evaluierten Eigenschaften, also die Robustheit, nicht ins Gewicht fallen. Es wurden Panoramaaufnahmen genutzt, um mittels der wiedergefundenen SIFT- Features die Position und die daraus folgende Richtung zum Ziel zu errechnen [Sch06].

Auch wurde eine Verwendung zum positionieren eines Werkzeuges, das an einem autonomen fahrzeug montiert ist, implementiert. Dafür muss der Roboter sich zu einer geeigneten Stelle bewegen, um das Wekkzeug anzusetzen. Diese Aufgabe wurde mit SIFT- Features in Kombination mit einem umrissbasierten Ansatz gelöst und erfolgreich getestet. Dieses Ver-

fahren soll von der 2009er Version eines Mars Rovers verwendet werden [DKS+05].

Auch kann mittels der SIFT- Features die Bewegung der Kamera zwischen zwei Bildern berechnet werden, wenn die Features der einzelnen Bilder einander zugeordnet werden können.

1.3 Funktionsweise

Dieses Kapitel gibt einen Überblick über die Funktionsweise von libsift. Eine umfassende Beschreibung von SIFT- Features (im Folgenden auch Keypoints genannt) findet sich in [Low04] und [BL02], eine der Suche und Filterung in [Ben75], [FB81] und [Nowa].

1.3.1 Wahl der Kandidaten für Features

Zunächst wird das Bild in mehrere "Oktaven" aufgeteilt, eine Oktave hebt sich durch die Auflösung des Ausgangsbildes hervor. Die Auflösung wird dabei in jeder neuen Oktave halbiert bis die Höhe oder Breite kleiner als 32 Bildpunkte ist. Innerhalb einer Oktave werden mehrere Gauss Filter inkrementell angewendet, die das Bild immer unschärfer werden lassen.

Für die Kandidatensuche werden lokale Maxima bzw. Minima in den Unterschieden zwischen den Gauss Bildern (DoG, Difference of Gaussian, [Jae05, Kap.12.7.6]) gesucht. Dabei werden die Scheitelpunkte in einer Umgebung von 27 Pixeln über 3 Ebenen der DoG- Bilder gesucht, also jeweils in einer 9er Umgebung pro Ebene (Abb.2). Anschließend werden die gefundenen Punkte bis auf Subpixelebene, also zwischen den Bildpunkten, lokalisiert [BL02]. Kandidaten, die problematische Keypoints ergäben, werden anschließend herausgefiltert (zum Beispiel wenn ein Punkt zu nah am Bildrand ist, so dass sein Descriptor abgeschnitten würde).

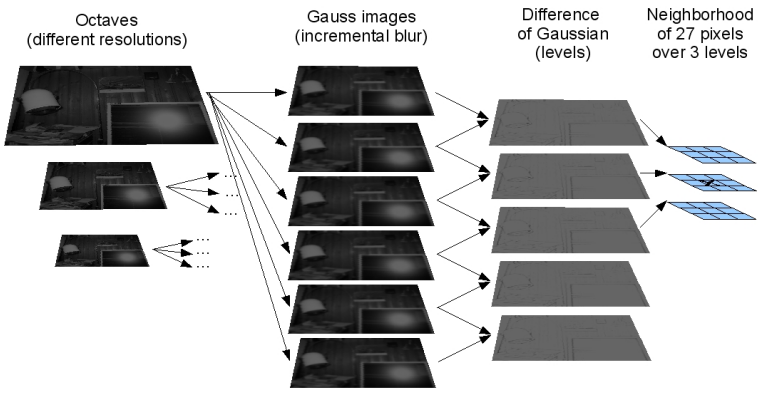


Abbildung 2: Schema der Kandidatensuche für Features

1.3.2 Aufbau der Descriptoren

Die Features sollen invariant gegenüber Rotation, Helligkeit und Größe sein. Um eine Invarianz gegenüber Rotation zu erreichen, wird eine Richtung für den Descriptor aus den Richtungsmagnituden errechnet und die Werte um diesen Wert gedreht. Man kann davon ausgehen, dass bei einer Drehung des Bildes bzw. der Kamera die errechneten Richtungen für den Keypoint sich nicht verändern. Dieses hat zur Folge, dass beide Keypoints des gleichen Punktes aus verschiedenen Bildern in die gleiche Richtung gedreht werden und somit auch drehinvariant sind.

Um Invarianz gegenüber Helligkeit zu erhalten, werden die Richtungsmagnituden der Umgebung des Kandidatenpixels zur Berechnung des Descriptors herangezogen. Diese sind Helligkeitsunabhängig, solange die Kante im Bild erkennbar ist. Die Magnituden und Richtungen berechnen sich wie folgt:

$$M(x,y) = \sqrt{(p(x+1,y) - p(x-1,y))^2 + (p(x,y+1) - p(x,y-1))^2}$$

$$Direction(x,y) = \arctan 2(p(x+1,y) - p(x-1,y), p(x,y+1) - p(x,y-1))$$

Die Features stammen aus unterschiedlichen Oktaven, so beschreiben sie auch unterschiedlich große Flächen im Originalbild. Da für die unterschiedlichen Oktaven das Ausgangsbild immer wieder verkleinert wird, aber alle Featurevektoren auf einer Umgebung von 128 Pixeln beruhen, repräsentieren die Features aus den kleinen Oktaven eine entsprechend große Region im Ausgangsbild. Der Descriptor wird für die folgenden Berechnungen als Vektor angesehen und auf die Länge eins normiert.

1.3.3 Matching von Features

Um die in den Bildern gefundenen Features zu vergleichen und möglichst schnell gute Ergebnisse zu erhalten wird ein kd- tree [Ben75] genutzt. Mit dessen Hilfe der Descriptor gesucht wird, der dem gesuchten am ähnlichsten ist. Es wird auch der Descriptor gesucht, der am zweitähnlichsten ist. Die Abstände der beiden gefundenen Features werden für eine Gewichtung der Güte der gefundenen Matches (übereinstimmende Keypoints in verschiedenen Bildern) herangezogen.

1.3.4 Filtern von gefundenen Übereinstimmungen

Zunächst werden Übereinstimmungen entfernt, die das gleiche Feature aus dem Vergleichsbild als nächsten Nachbarn haben. Da in diesen Fällen mindestens eine falsche Zuweisung erfolgt ist, können diese Matches als inkonsistent betrachtet werden.

Die restlichen Übereinstimmungen von Features werden mit Hilfe des "RANdom SAmple Consensus" (RANSAC [FB81]) gefiltert. Aus einer Menge zufällig gewählter Matches werden Modelle errechnet. Als Modell dient eine Transformationsmatrix von den Koordinaten der Keypoints in dem einen zu den Koordinaten in dem anderen Bild [Nowa].

Matches, die nicht zu dem besten gefundenen Bewegungsmodell passen, werden entfernt.

Die übrigen Matches werden nach einer Gewichtung sortiert. Das Gewicht des Matches setzt sich zusammen aus der Entfernung des nächsten Nachbarn und dem Abstand zwischen dem dichtesten und zweitdichtesten Feature.

$$w(M) = (dist1)^{distExp} * (\frac{1}{dist2 - dist1})^{quotExp}$$

$w(M)$ steht für das Gewicht des Matches, $dist1$ für die Entfernung des Keypoints zu seinem nächsten Nachbarn und $dist2$ für die zum zweitnächsten.

Die Exponenten $distExp$ und $quotExp$ können zur Gewichtung der Berechnungskomponenten genutzt werden. Ein hoher distExp (distance exponent) sorgt für eine höhere Gewichtung des Keypointabstandes bei der Nearest- Neighbour- Suche mittels kd- tree, quotExp für eine höhere Gewichtung des Entfernungsunterschiedes zwischen dem dichtesten und zweitdichtesten Keypoint. Die Exponenten sind für die folgenden Tests jeweils eins.

2 Portierung von C# nach C++

Die bestehende SIFT Bibliothek ist in C# für die Microsoft .NET Laufzeitumgebung implementiert. Die Bibliothek wurde im Zuge dieser Arbeit nach C++ portiert.

2.1 Notwendigkeit der Portierung

Libsift kann nur in einer Virtual Machine (VM) für .NET Programme genutzt werden. Der Vorteil ist, dass die Programmiersprache für die Nutzung der Bibliothek unwichtig ist (der .NET Bytecode ist kompatibel). Man könnte die Bibliothek auch ohne Portierung nutzen, ist dann allerdings dazu gezwungen, immer die VM installiert zu haben und zu benutzen.

Hinzu kommt, dass zur Zeit nur die VMs an sich unter anderen Betriebssystemen als Microsoft Windows nutzbar sind, nicht aber die entsprechenden Compiler. Weder Mono[1], noch dotGNU[2] (welches die .NET VMs für andere Betriebsysteme sind) bieten einen Compiler zu .NET Bytecode an, der dann in der VM ausgeführt werden kann.

Wenn man die Library ohne Portierung nutzen wollte, müsste man das Programm, das die Bibliothek nutzt, unter Microsoft Windows kompilieren um dann das Programm unter anderen Betriebssystemen, beispielsweise Linux, zu testen.

Da die Bibliothek unter Umständen auch auf Robotern zum Einsatz kommen könnte, die einen Microcontroller (zum Beispiel Aimee[ABP+05][3]) nutzen oder mit Betriebssystemen arbeiten, für die es keine .NET VM gibt, wurde trotz der anderen Möglichkeiten zu einer nativen C++ [Str97] Bibliothek portiert. Der SIFT Algorithmus aus libsift scheint zudem recht ausgereift, so dass es in Zukunft wahrscheinlich nicht zu großen Änderungen der libsift kommen wird.

2.2 Unterschiede der Programmiersprachen

Ein großer Unterschied ist das Speichermanagement. Bei der Nutzung von C++ muss die Freigabe von benutztem Speicher manuell erfolgen, wobei C# dieses automatisch regelt, sobald keine Referenzen mehr auf die Daten bestehen. Eine reine Portierung von C# nach C++ zu kompilierenden Code reicht somit nicht aus. Es kommt zu Speicherzugriffsfehlern, die nur durch intensive Suche mittels debugging Wekzeugen behoben werden können. Hervorgerufen werden diese hauptsächlich durch das Freigeben des Speichers, obwohl der Inhalt noch benötigt wird.

Unterschiede gibt es auch bei der Initialisierung von Variablen. C# initialisiert zum Beispiel integer- Zahlen automatisch mit null, C++ lässt sie uninitialisiert – es können bereits Werte enthalten sein. Somit muss die Initialisierung in C++ explizit erfolgen, da ansonsten schon vorhandene Werte Ergebnisse verfälschen können. Diese Fehler äußern sich nur über fehlerhafte Zuordnungen der Keypoints zueinander, deswegen setzen sie

[1] http://www.mono-project.com/
[2] http://dotgnu.org/
[3] Dieser bräuchte dann mehr Rechenleistung

Wissen über die Funktionsweise der Bibliothek voraus, um die entsprechenden Stellen im Quelltext zu finden.

2.3 Probleme der Portierung

Notation:

- "C#::Klasse" bezieht sich auf eine Klasse in der C# Implementierung
- "C++::Klasse" bezieht sich auf eine Klasse in der C++ Implementierung

Ein Problem war, dass viele Methoden ein typenloses C#::object als Parameter hatten, das so in C++ nicht vorhanden ist. Oft wurde es aber direkt nach dem Aufruf in eine "richtige" Datenstruktur umgewandelt, so dass man gleich den Methodenaufruf ändern konnte.

Ein weiteres Problem, das leicht übersehen werden kann, ist die Übergabe von Referenzen. Diese ist in C# nicht sofort durch ein "&" oder Ähnliches ersichtlich. Ob es sich um eine Referenz handelt, hängt in C# von dem Datentyp ab: Handelt es sich um "einfache Typen" (Integer,Floating Point) so wird eine Kopie übergeben, bei höheren Klassen eine Referenz.

Außerdem gibt es in C# eine einfache Art get- und set- Methoden zu definieren zum Beispiel für den Zugriff auf var:

```
int var;

int Var {
    get {
        return (var);
    }
    set {
        var = value;
    }
}
```

Es können in der get- oder set- Methode beliebige Berechnungen ausgeführt werden, der Zugriff erfolgt dann ganz normal mit dem = Operator ($Var = 5$ für set, bzw. $inttmp = Var$ für get). Da meistens Vollzugriff auf die Variablen gewährt wird, wurden die Eigenschaften der entsprechenden Variablen so geändert, dass sie direkt von anderen Objekten aus zugreifbar sind (public).

2.4 Nicht portierte Methoden und Klassen

Zwei Methoden, die für die Panoramaerstellung gedacht sind, wurden nicht portiert. Diese sind MultiMatch::ComponentCheck und MultiMatch::BuildBondBall.

Die Klasse distmetrictest wurde ebenfalls nicht portiert, da es sich um eine reine Testklasse handelt.

2.5 Update auf libsift Version 1.8

Während der Fehlerbehebung nach dem Portieren ist eine neue Version von libsift veröffentlicht worden. Diese beinhaltet abschaltbare Ausgaben, die ansonsten im Eingabefenster gezeigt würden. Ausserdem wurde ein Fehler bei der Generierung der Gauss- Bilder für die Kandidatensuche korrigiert (Kap.1.3 Funktionsweise). Die Änderungen und Korrekturen wurden ebenfalls in die C++ Version eingearbeitet.

3 Experimente

Die Bibliothek soll zum Zwecke der Roboternavigation genutzt werden. Hieraus entstehen besondere Anforderungen, die getestet wurden und in diesem Kapitel beschrieben werden.

Bei autonomen Robotern, die sich auch in unwegsamen Gelände bewegen können, ist vor allem eine Robustheit gegen Verdrehungen der Kamera wichtig. Dieses gilt besonders bei laufenden Systemen (zum Beispiel Scopion [SK02]) oder Systemen mit schwenkbarer Kamera. Auch Unabhängigkeit von Beleuchtungsverhältnissen ist in diesem Szenario wichtig. Die Wiedererkennung aus unterschiedlichen Roboterpositionen (Drehung, seitliche Verschiebung, anderer Blickwinkel) wurde ebenfalls getestet, da zum Beispiel bei der Betrachtung eines Würfels durch eine andere Position auch Seiten sichtbar werden können, die vorher nicht im Bild auftauchten.

Generell können diese Tests in zwei Kategorien eingeteilt werden: *Experimente mit künstlich veränderten Bildern* und *Experimente mit Bildfolgen*. Im ersten Fall wurde zunächst die allgemeine Funktionalität anhand eines Bildes evaluiert, das viele Features aufweist (Abb.3). Danach wurden die Ergebnisse mittels Kamerabildern, die nicht algorithmisch erstellt wurden, verifiziert.

In den Diagrammen gibt es besonders viele Übereinstimmungen wenn ein Bild mit sich selbst verglichen wird. Dieses Phänomen ist nur bei *exakt* gleichen Bildern zu beobachten. Unterschiedliche Aufnahmen führen durch Sensorrauschen oder kleine Bewegungen zu deutlich weniger Übereinstimmungen, auch wenn die Bilder mit einem Stativ aufgenommen wurden. Zur Veranschaulichung kann das Diagramm des Helligkeitsexperimentes mittels Bildfolgen betrachtet werden, da diese Aufnahmen mit Stativ und im kurzen Abstand aufgenommen wurden (Abb.21).

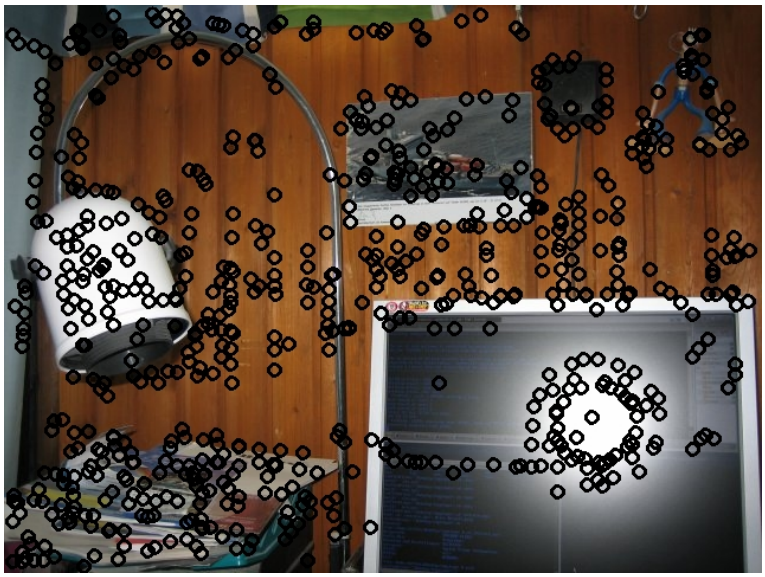


Abbildung 3: Bild mit vielen Features (Mittelpunkte)

3.1 Experimente mit künstlich veränderten Bildern

Für die folgenden Experimente wurde das Referenzbild (Abb.3) mit algorithmisch veränderten Kopien verglichen. Das Referenzbild hat eine Auflösung von 640x480.

Diese Tests können keine Tests in der Realität ersetzen, da die Erkennung aus einem anderem Blickwinkel oder Änderung der Beleuchtung sich nicht wie in der realen Welt verhalten. Beispielsweise ändern sich bei einem Beleuchtungswechsel in der realen Welt auch Schattenverhältnisse und Farben der Objekte.

Um diese Tests durchzuführen wird ein Shellscript und das Programm "convert" (Bestandteil von ImageMagick[4]) genutzt. Mit convert kann man die Eingabedateien verändern, zum Beispiel Auflösung (Größe) und Helligkeit ändern oder Bilder drehen.

Es werden jeweils die besten 10 Übereinstimmungen auf den Bildern dargestellt.

3.1.1 Perspektive

SIFT- Features sollen nach ihrer Definition invariant gegenüber Größe, Helligkeit und Rotation sein. Perspektivenänderungen kommen bei der Roboternavigation aber ebenfalls vor. Es soll getestet werden, ob Keypoints aus unterschiedlichen Perspektiven wiedererkannt werden können, oder dieses aufgrund der veränderten Winkel im Bild nicht möglich ist. Um dieses zu testen, wurde eine Kopie des Ausgangsbildes perspektivisch verzerrt und versucht Übereinstimmungen zwischen den beiden Bildern zu finden.

Es wurden unterschiedliche Stärken der Verzerrung getestet. Die rechte Bildseite wurde von oben und unten gleichmäßig gestaucht, wobei die linke Seite nicht gestaucht wurde. So wird eine Bewegung der Kamera auf einer Kreisbahn nach links simuliert. Da das Ausgangsbild eine vertikale Auflösung von 480 Pixel betrug, ist bei bereits einer Stauchung von 120 Pixel die rechte Bildseite nur noch halb so groß wie die linke, da die Stauchung von oben und unten gleichmäßig angewendet wird (Abb.4).

Abbildung 4: Vergleich mit einer perspektivisch verzerrten Kopie

Selbst bei starken Verzerrungen werden noch richtige Übereinstimmungen gefunden. Bei 30 Pixeln Stauchung $(12,5\%)$ können noch knapp die

[4] http://www.imagemagick.org

Hälfte der Keypoints des Originalbildes wiedergefunden werden. Bei $20,8\%$ (50 Pixel) noch ca. ein Viertel. Zu beobachten ist allerdings, dass sich bei starker Stauchung die gefundenen Punkte auf die Mitte des Bildes konzentrieren, wo die Änderungen durch die Stauchung am geringsten ist.

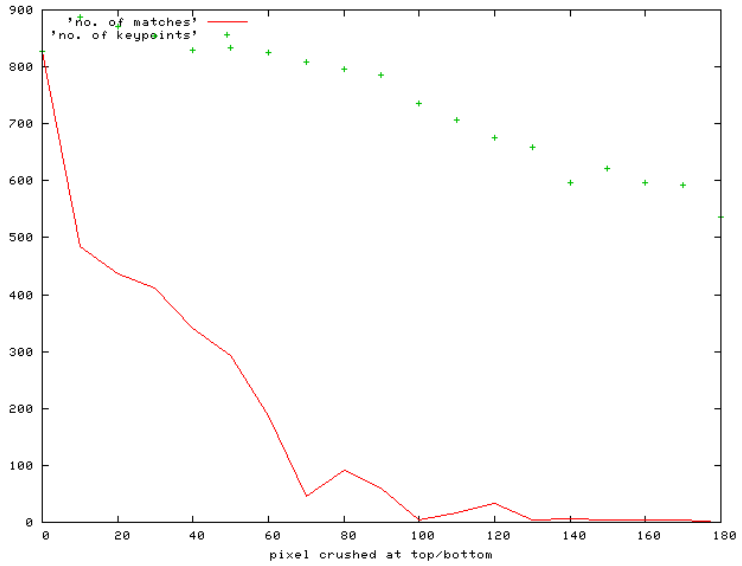


Abbildung 5: Plot: Perspektivenänderung

Da SIFT- Features kantenbasiert sind, war dieses Verhalten zu erwarten, da sich durch perspektivische Änderungen die Winkel optisch ändern. Ein 90^o Winkel hat aus einem anderen Blickwinkel gesehen einen anderen optischen Winkel.

3.1.2 Größe

Dieser Test simuliert eine größere Entfernung zu den Objekten in dem Bild. Das Originalbild wird mit sich selbst, in einer in jeweils 10%- Schritten verkleinerten Version, verglichen (Abb.6). Der Test kann Aufschluss darüber geben, ob der Algorithmus mit unterschiedlichen Größenverhältnissen zurecht kommt. Er spiegelt aber nicht die Realität wieder, da sich im Originalbild näher liegende Objekte bei Vergrößerung der Entfernung in der Realität schneller verkleinern als weiter entfernt liegende Objekte.

Betrachtet man eine Lochkamera, dann errechnet sich die Breite eines Objektes aus $x = b\frac{X}{Z}$, wobei x der Größe im Bild, b der Brennweite, X der realen Breite und Z der Entfernung entspricht. Nimmt man zur Vereinfachung eine Brennweite von eins an und betrachtet zwei Objekte, drei und vier Meter entfernt, deren Breite auch eins ist. Dann ändern sich die Breiten im Bild bei einer um einen Meter größeren Entfernung von $1/3(0.33)$ auf $1/4(0,25)$ und von $1/4(0.25)$ auf $1/5(0.2)$. Das weiter entfernte Objekt hat sich im Bild langsamer (um 0.05) verkleinert als das dichtere (um 0.08). Bei diesem Test aber verkleinern sich alle Objekte gleich schnell, unabhängig ihrer Entfernung zur Kamera.

Abbildung 6: Vergleich mit einer auf 30% verkleinerten Kopie

Trotz der Größenänderung werden Übereinstimmungen gefunden, diese sind aber gemessen an den gefundenen Features im verkleinerten Bild deutlich weniger als bei dem Vergleich mit der Originalgröße. Die Keypoints sind nur begrenzt größeninvariant. Ab einer Göße von 20% des Originals werden kaum noch Übereinstimmungen gefunden (Abb.7). Der Grund liegt anscheinend an der Generierung mittels der Oktaven (Kap.1.3 Funktionsweise). Bei dem Vergleich mit Bildern, deren Größe bei der Generierung nicht in den Oktaven enthalten waren, werden weniger Übereinstimmungen gefunden als bei denen, die enthalten waren. So ist auch die höhere Anzahl der Übereinstimmungen bei 50% zu erklären, da das Bild genau der 2. Oktave entspricht.

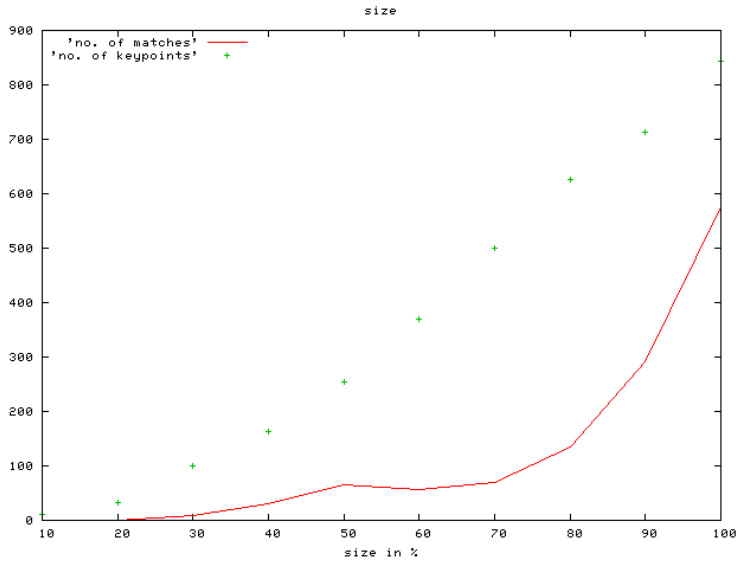


Abbildung 7: Plot: Größenänderung

3.1.3 Helligkeit

Helligkeitsänderungen werden mittels einer Änderung des Gammawertes simuliert, diese berechnet sich nach $i' = i^\gamma$ (i entspricht den Wert des Pixels, also direkt der Helligkeit). Es gibt Unterschiede zu der Realität. Echte Beleuchtungsänderung bewirkt oft auch eine Änderung von Schattenintensität, Schattenposition und der Farben. Bei einem Sonnenuntergang werden Schatten länger, das rötliche Licht ändert auch das Erscheinungsbild von Objekten. Helle Objekte ändern ihre Farbe unter realen Bedingungen besonders stark und somit könnten die für die Keypoints ausschlaggebenden Kanten schlechter zu erkennen sein als es bei dieser Simulation der Fall ist.

Bei einem Gammawert von 0.4 werden durchaus noch brauchbare Übereinstimmungen gefunden (Abb.8). Bei dem Vergleich mit dem Bild mit Gamma $= 0.1$ bleibt zu bezweifeln, ob es nicht auch zu positiven Ergebnissen führt, wenn man ein Bild von einer Lampe in einem abgedunkelten Raum macht (Abb.9). Ein kurzer Test mit einem künstlich erstellten Bild zeigte, dass selbst dann keine Übereinstimmungen gefunden werden, auch wenn die Kontur der hellen Bereiche dem Original ähnlich ist (Abb.10). Hieraus kann man folgern, dass nur dann Übereinstimmungen bei unterschiedlicher Helligkeit gefunden werden, wenn eine "echte" Ähnlichkeit der Bilder besteht.

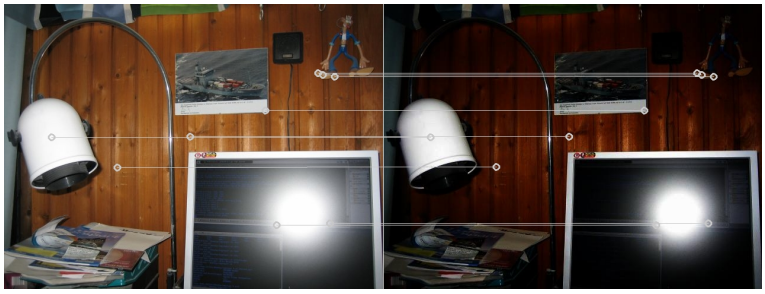


Abbildung 8: Vergleich mit einer Kopie mit $\gamma = 0.4$

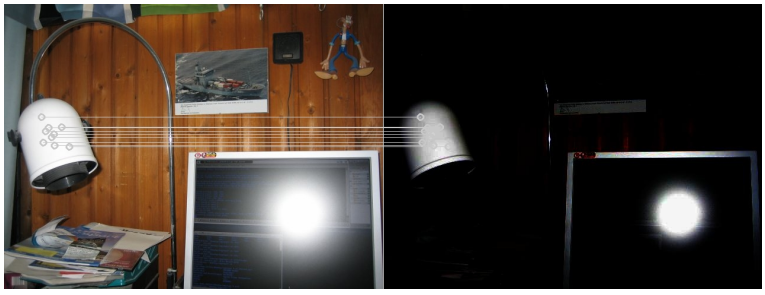


Abbildung 9: Vergleich $\gamma = 0.1$

Die Anzahl der gefundenen Übereinstimmungen nimmt, wie zu erwarten war, mit zunehmender Dunkelheit ab (Abb.11). Die Abnahme ist da-

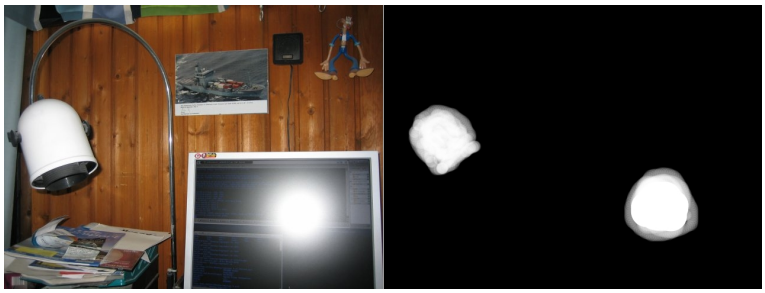


Abbildung 10: Versuch mit einem ähnlichen Bild

bei aber nahezu linear. Dieses liegt an dem Aufbau der Keypoints, die als beschreibendes Element nicht Farbe oder Helligkeit sondern Magnituden, also vereinfacht gesagt Kantenrichtungen, aufweisen. Solange also Kanten und Konturen im Bild erkennbar sind, bleiben die dazugehörigen Keypoints auch ähnlich und so können auch Übereinstimmungen noch hinreichend gut gefunden werden.

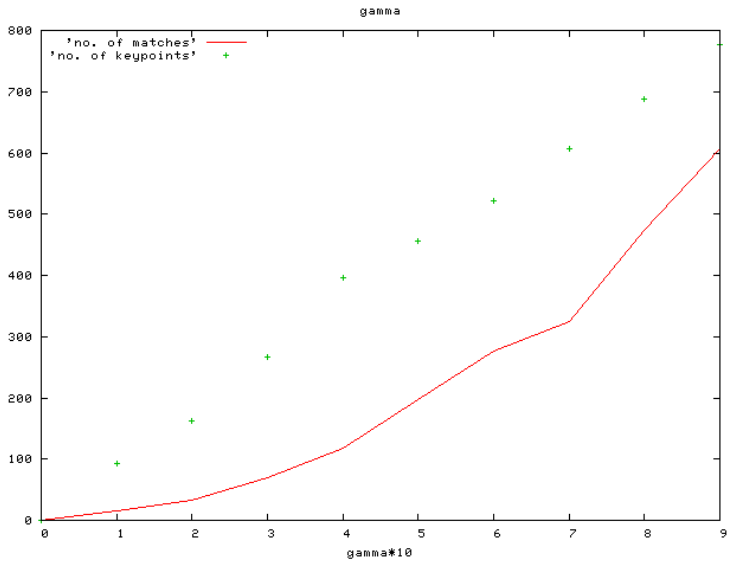


Abbildung 11: Plot: abnehmende Helligkeit

3.1.4 Drehung

Dieser Test prüft Invarianz gegenüber Drehung der Kamera um die optische Achse. Eine Kopie des Originals wird in 10^o Schritten gedreht und mit dem Original verglichen (Abb.12). Dieser Test ersetzt nicht den in der Realität, da zur Drehung immer exakt dasselbe Bild genutzt wird und es somit keine Änderungen in der Kameraposition, Drehachse oder Beleuchtung geben kann. Hinzu kommt, dass durch das Drehen oft ein Rand um das gekippte Bild entsteht, so dass auch außerhalb des Originalbildes des Keypoints gefunden werden könnten. Deswegen wird für diesen Test ein Bildausschnitt aus der Mitte des Bildes verwendet. Dieses bewirkt allerdings, dass es je nach Drehung mehr oder weniger Keypoints in dem gedrehten Bildteil gibt.

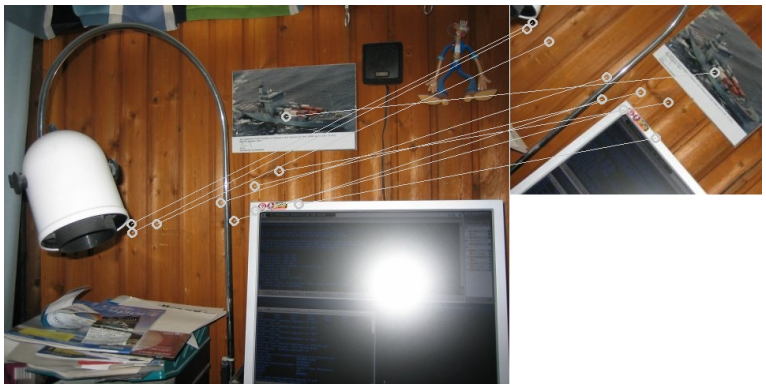


Abbildung 12: Vergleich mit einer um 50^o gedrehten Kopie

Bei der Durchführung wurden für jeden getesteten Winkel Übereinstimmungen gefunden (Abb.13), allerdings nimmt die Anzahl der Matches gegenüber eines nicht gedrehten Bildes deutlich ab. Ein Problem ist die Auswirkung der bei der Drehung notwendigen Interpolation [Jae05, Kap.10.5]. Diese wird für jeden Keypoint sowie für das Drehen des Testbildes verwendet und bewirkt eine Verfälschung. Bei dem gedrehten Bild sind die Eingangsdaten bereits interpoliert worden, bevor sie bei der Keypointgenerierung erneut interpoliert werden. Bei dem Vergleich mit dem Originalbild gibt es mehr Matches, da die Eingangsdaten nicht interpoliert wurden und die interpolierten Daten der Keypoints somit exakt die gleichen sind.

Eine Untersuchung des Quelltextes der Bibliothek ergab, dass es zusätzlich einen Fehler in C#::DScaleSpace::CreateDescriptors Implementierung gibt, der sich auf die Rotationsinvarianz auswirkt. Bei der Generierung des Desciptors für den Keypoint wurde die Drehrichtung unter gewissen Umständen fehlerhaft normiert. Nach Behebung des Fehlers in C++::DScaleSpace::CreateDescriptors gab es ein deutlich besseres Ergebnis (Abb.14), bei dem die unterschiedliche Anzahl an Übereinstimmungen hauptsächlich auf die verschiedenen Bildausscnitte zurückzuführen ist. Die zuvor beschriebene Interpolationswirkung ist allerdings auch erkennbar, die Anzahl der Übereinstimmungen ist bei nicht interpolierten Bildern höher

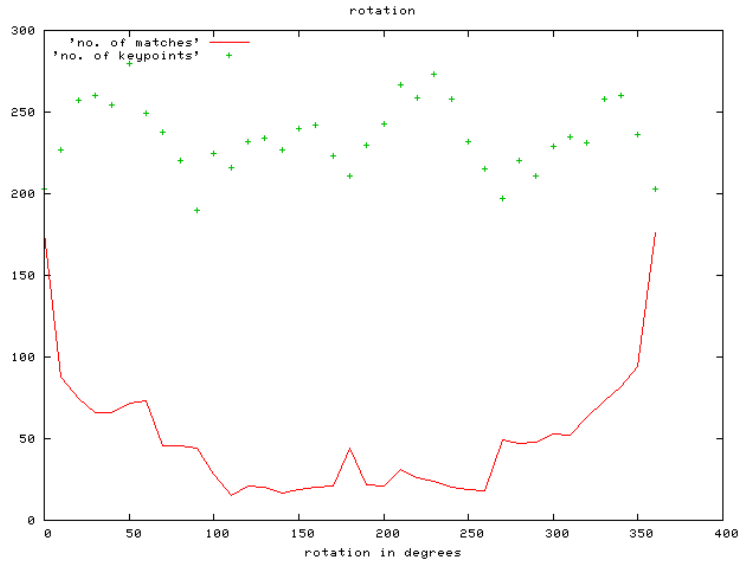


Abbildung 13: Plot: Drehung (vor Korrektur)

(90,180 und 270^o).

Um die Auswirkungen des Fehlers zu verdeutlichen wurde ein erneuter Test durchgeführt. Dieses Mal wurde das komplette Bild gedreht und in diesem ein kleiner Bildausschnitt des Originalbildes gesucht um vergleichbare Ergebnisse zu erhalten. Das Randproblem ist nicht eleminiert, tritt aber bei beiden Kurven gleich stark auf, so dass es vernachlässigbar ist. Im Bildausschnitt befanden sich 193 Keypoints. Man kann deutlich einen Einbruch der gefundenen Übereinstimmungen erkennen, der nach der Fehlerkorrektur nicht mehr auftritt (Abb.15)

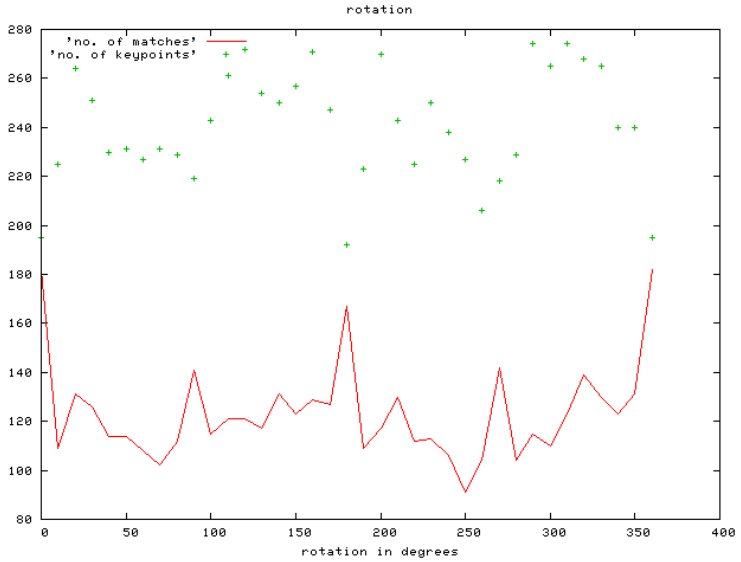


Abbildung 14: Plot: Drehung

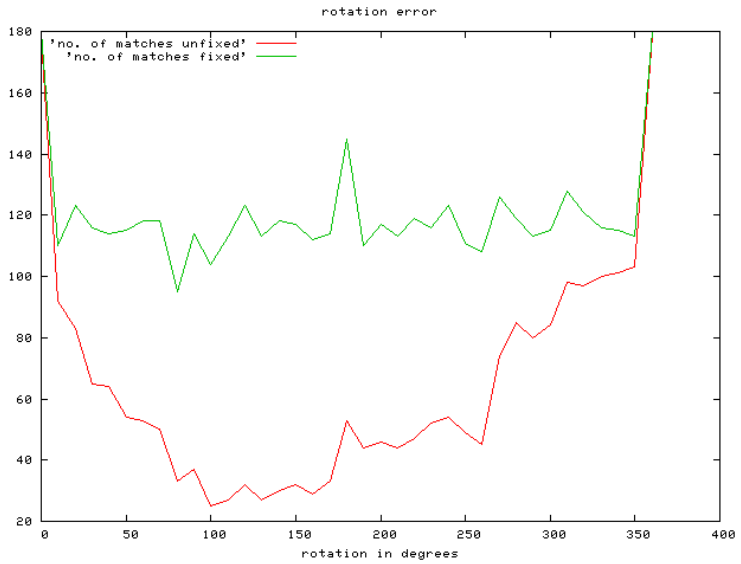


Abbildung 15: Plot: Gegenüberstellung vor und nach Korrektur

3.2 Experimente mit Bildfolgen

Für die meisten dieser Tests wird eine Stereokamera (STG-DCSG von Videre Design[5]) genutzt. Es handelt sich um eine Firewire Kamera, die eine maximale Auflösung von 640x480 liefert. Für diese Tests werden die Bilder auf 320x240 verkleinert und keine Informationen aus der zweiten Linse genutzt. Diese können aber später dazu genutzt werden, die Keypoints genauer im Raum zu lokalisieren um eine bessere Navigation zu ermöglichen. Wenn ein Punkt in beiden Bildern der Kamera gefunden wird, kann auch dessen Entfernung zur Kamera mittels Triangulation [Jae05, Kap.8.2] ermittelt werden.

Die Tests wurden mit Hilfe von zuvor aufgenommenen Videos und Bildern durchgeführt, da so jedes einzelne Bild in die Berechnungen einbezogen werden konnte. Durch die relativ lange Wartezeit während der Keypointberechnung hätten sonst die Bewegungen zwischen den berechneten Bildern ungleichmäßig werden können.

Die Videos wurden im DivX[6] Format gespeichert, wobei das Video komprimiert wird. Die genutzten Bilder sind also von schlechterer Qualität, als würden die Kamerabilder direkt genutzt. Somit ist damit zu rechnen, dass die Ergebnisse bei direkter Nutzung der Kamerabilder besser ausfallen als die Ergebnisse dieser Tests (Kap.3.5 Bildkompression).

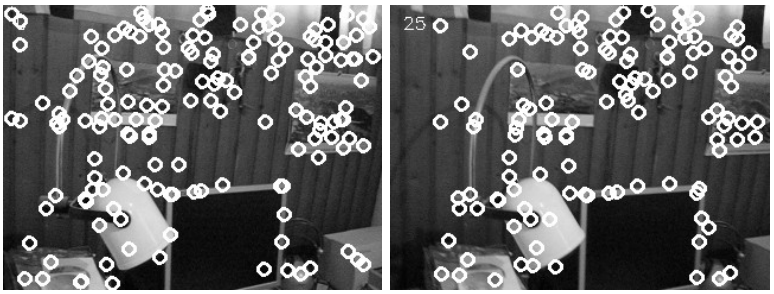
In einigen Fällen gibt es in den Plots eine große Anzahl von falsch positiven Matches, also Zuordnungen, die der Algorithmus für richtig hält, obwohl sie es nicht sind. Diese können durch die Funktionsweise des RANSAC Filters verursacht werden (Kap.1.3 Funktionsweise). Falls in der Menge der Bewegungsmodelle, die aus zufällig gewählten Matches errechnet wurden, keines der realen Bewegung der Kamera entspricht, werden Matches als falsch positiv angesehen, obwohl sie es unter Umständen nicht sind.

Die weißen Ringe in den Bildern zeigen die wiedergefundenen Keypoints aus dem Referenzbild des jeweiligen Experimentes.

[5] http://www.videredesign.com
[6] http://www.divx.com

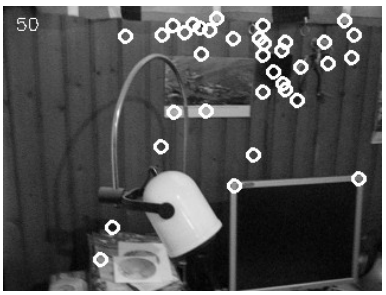
3.2.1 Perspektive

Bei diesem Test wurde die Kamera von links nach rechts um ein Objekt bewegt, verglichen wurde mit dem ersten Bild (Abb.16(a)).

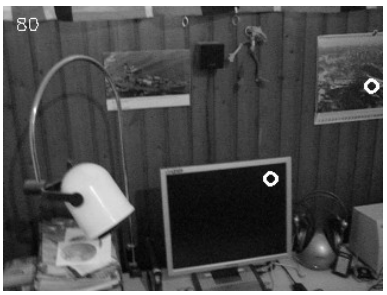


(a) Frame 0

(b) Frame 25



(c) Frame 50



(d) Frame 80

Abbildung 16: Video: Perspektive

Das Diagramm zeigt bis um Frame 25 gute Ergebnisse, bei denen noch über ein Drittel der Keypoints wiedergefunden werden (Abb.17). Um Frame 50 ist die Matchanzahl zwar noch ausreichend, aber deutlich gesunken. Ab Frame 80 kommt es zu sehr vielen falsch Positiven, so dass man ab diesem Bereich wohl keine brauchbaren Informationen aus den Matches herleiten kann.

Die großen Unterschiede zu der simulierten Verzerrung basieren auf der Tatsache, dass in der Realität sich die Positionen von Objekten zueinander ändern, wenn sich der Blickwinkel ändert. Je weiter ein Objekt entfert ist, desto kleiner ist dessen Bewegung im Bild. Features, die im Vergleichsbild auf Kanten von unterschiedlich weit entfernten Objekten beruhen, können bei einem anderen Blickwinkel nicht wiedererkannt werden, da sich die Position dieser Kanten zueinander ändert. Dieses ist gut an dem Abstand zwischen Lampe und Monitor erkennbar (Abb.16)

Dieses Ergebnis zusammen mit dem der künstlichen Verzerrung deutet darauf hin, dass Features aus flachen Gegenständen wie zum Beispiel Bilder oder flache künstliche Landmarken an Wänden auch aus anderen Perspektiven erkennbar sind. Weniger größeninvariant sind Features, die aus Objekten unterschiedlicher Entfernung zusammengesetzt sind.

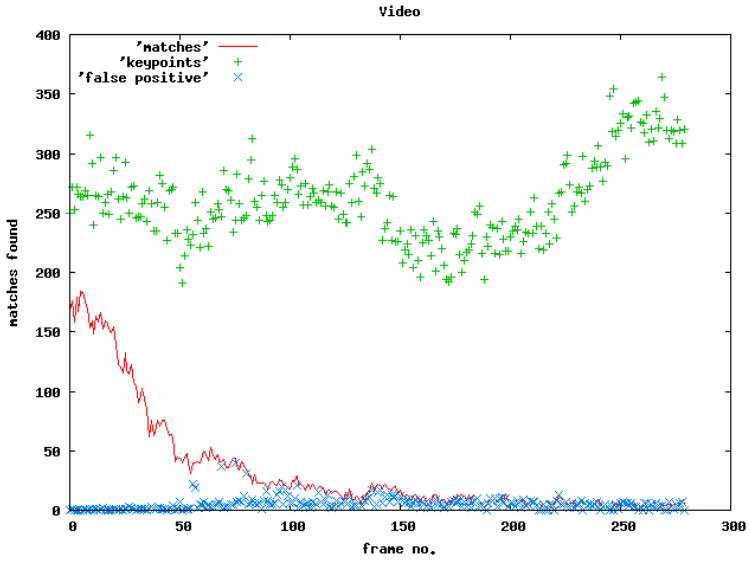
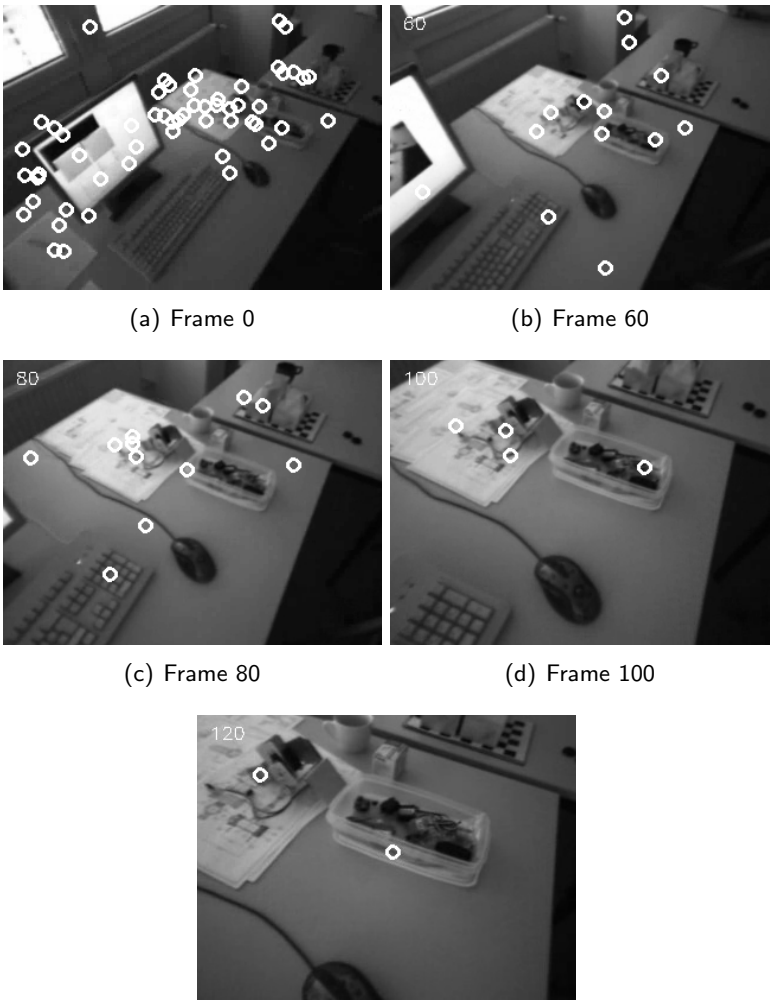


Abbildung 17: Plot: Perspektive

3.2.2 Größe

Bei diesem Test wird die Invarianz gegenüber der Größe, an realen Bildern, erneut überprüft. Die Kamera wurde manuell bewegt. Der RANSAC Filter war bei diesem Test aktiv. Die Kamera wurde auf ein Objekt zubewegt, so dass sich dieses vergrößert und die Randbereiche dementspechend aus dem Blickfeld der Kamera verschwinden.



(a) Frame 0 (b) Frame 60

(c) Frame 80 (d) Frame 100

(e) Frame 120

Abbildung 18: Video: Größe

Die Zahl der Keypoints vermehrt sich, obwohl sich weniger Gegenstände die Kanten verursachen im Kamerabild befinden (Kap.3.4 Auswirkungen der Bildgröße auf die Zahl der Keypoints). Trotz der Zunahme an Keypoints nimmt die Anzahl der gefundenen Matches ab, obwohl noch ein recht großer Ausschnitt aus dem Vergleichsbild sichtbar ist (Abb.18(e)). Dieses ist ein Zeichen für eine gute Invarianz der Keypo-

ints gegenüber unterschiedlicher Größe (Abb.19), es werden wenig falsche Matches gefunden, allerdings nimmt gleichzeitig die Anzahl der gefundenen Übereinstimmungen ab.

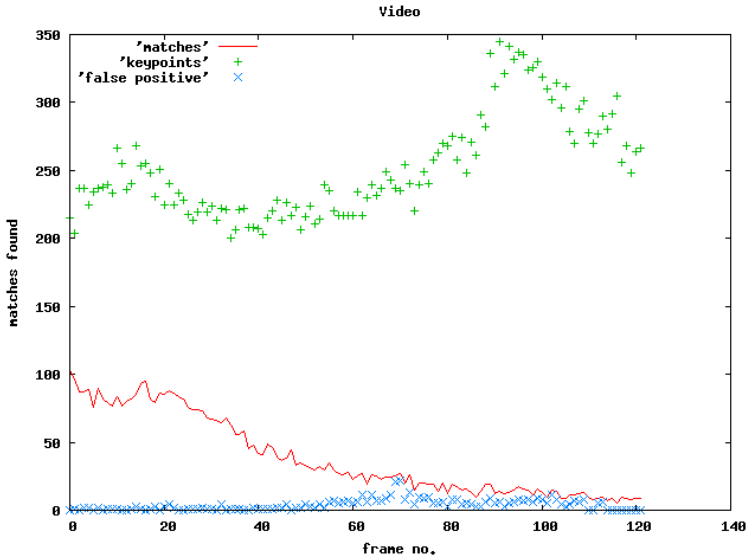
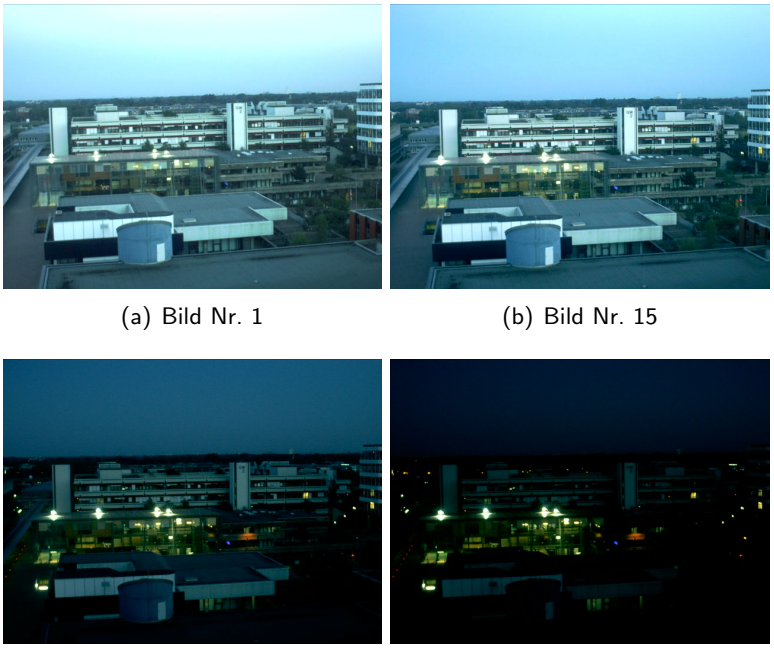


Abbildung 19: Plot: Größe

3.2.3 Helligkeit

Bei diesem Test wurde kein Video verwendet sondern eine Fotoreihe, die während der Dämmerung aufgenommen wurde. Die Fotos wurden in einem Intervall von 30 Sekunden aufgenommen (Abb.20).



(a) Bild Nr. 1 (b) Bild Nr. 15

(c) Bild Nr. 35 (d) Bild Nr. 48

Abbildung 20: Bildreihe: Helligkeitsinvarianz

Es wurde jedes Bild mit dem ersten aufgenommenen verglichen, das 1407 Keypoints enthält. Der Helligkeitswert im Diagramm ist die durchschnittliche Helligkeit der Pixel im Bild. Schwankungen in den Helligkeitswerten lassen sich durch unterschiedlichen Schattenwurf oder Objekte, die in den anderen Bildern nicht sichtbar sind, erklären. Der Test bestätigt den mittels Gammawerten durchgeführten Test (Kap.3.1.3 Helligkeit). Die Anzahl der Keypoints sowie die der Übereinstimmungen nehmen fast linear ab, trotzdem werden noch bei starken Veränderungen der Farben ausreichend Übereinstimmungen gefunden, zum Beispiel bei Bild 35 (Abb.20(c)) noch ca. 100.

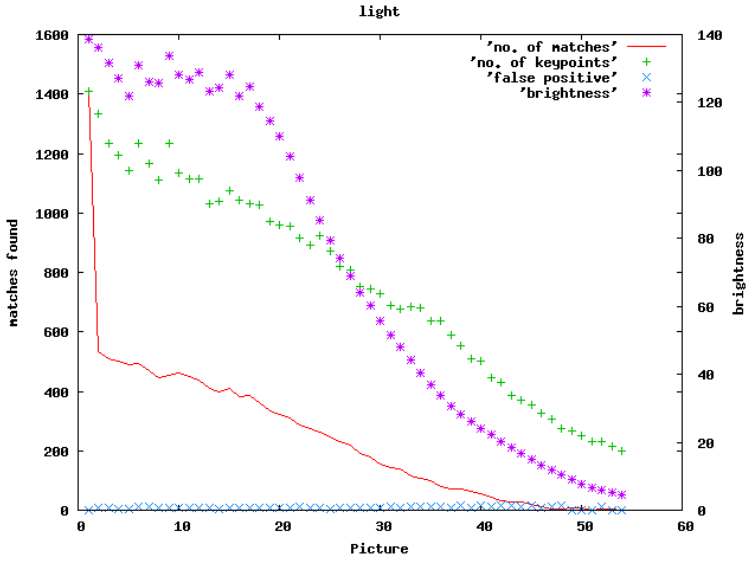
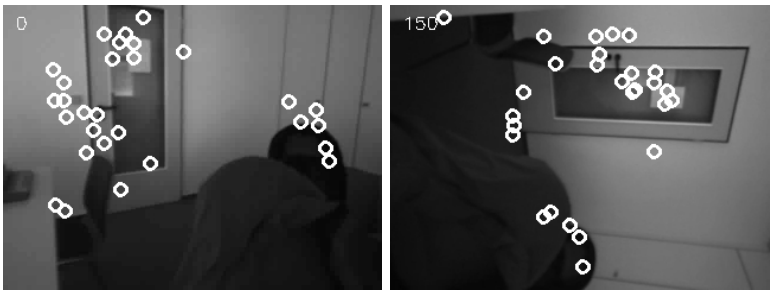


Abbildung 21: Plot: Helligkeitsinvarianz

3.2.4 Drehung

Drehungen einer Kamera können in zwei Arten aufgeteilt werden, die um die optische Achse (um die Blickrichting, wobei die Bildinhalte nahezu gleich bleiben) und die um andere Achsen, bei denen sich die Bildinhalte ändern. Es wird exemplarisch die Drehung um die Hochachse verwendet, damit nicht der Boden oder die Decke komplett ins Blickfeld gelangen. Es soll die Neigung zu falsch positiven Übereinstimmungen getestet werden, somit sollten auch ähnliche Bereiche sichtbar bleiben und nicht Features der Decke oder des Bodens mit denen von Schrankwänden verglichen werden.

3.2.4.1 Drehung um die optische Achse Dieser Test entspricht der Rotation bei den vorherigen Tests ohne Kamera (Kap.3.1.4 Drehung). Die Kamera wurde um ca. 90^o seitlich nach links verdreht. Im Gegensatz zu dem zuvor durchgeführten Test tritt das Randproblem etwas anders auf. Es können Bildteile auftauchen, die im Vergleichsbild nicht vorhanden sind. Dieses Problem bedarf allerdings keiner gesonderten Behandlung, da es sich um reale Bildelemente handelt, die keine unnatürlichen Kanten einfügen und somit das Ergebnis auch nicht verfälschen. Die Kamera wurde freihändig gedreht, so dass es sich nicht immer um die gleiche Drehachse gehandelt hat, wie es bei dem vorherigen Experiment der Fall war. Die RANSAC Filterung war aktiv.



(a) Frame 0 (b) Frame 150

Abbildung 22: Video: Drehung um die optische Achse

Es ist erkennbar, dass sich die Anzahl der Keypoints ständig ändert, je nachdem welcher Bildausschnitt sichtbar ist (Abb.23). Erwartungsgemäß müsste die Anzahl der gefundenen Übereinstimmungen bei 90^o abnehmen. Wenn die Kamera um 90^o gedreht ist, dann ist der maximal in beiden Bildern sichtbare Teil 480x480 Pixel bei einer Videoauflösung von 640x480. Somit sollte dementsprechend auch die Anzahl der gefundenen Übereinstimmungen sinken. Dieser Effekt ist in diesem Fall allerdings nicht stark, da sich die meisten Keypoints des Ausgangsbildes mittig befinden und somit auch sichtbar bleiben.

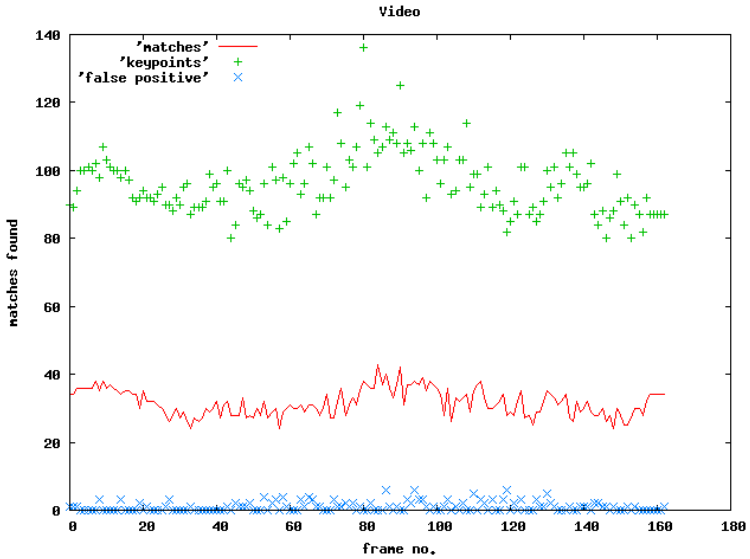


Abbildung 23: Plot: Drehung um die optische Achse

3.2.4.2 Drehung um die Hochachse Dieser Test wurde nicht vorher simuliert, das verwendete Video zeigt eine 360^o Aufnahme eines Raumes. Die Kamera wurde dabei auf einem Stativ gedreht und versucht die Keypoints des ersten Bildes wieder zu erkennen. Da die Anfälligkeit für falsch positive Matches getestet werden sollte, wurde auf die RANSAC Filterung verzichtet.

(a) Frame 0

(b) Frame 120

(c) Frame 320

(d) Frame 740

Abbildung 24: Video: Drehung um die Hochachse

Für ein paar Frames am Ende des Videos gibt es einen Abfall von Matches. Dieses liegt an einer partiellen Verdeckung des Hintergrundes, so dass einige Keypoints nicht mehr sichtbar sind (Abb.24(d)).

Es kommt nur zu wenig falsch positiven Matches. Diese sind unabhängig von der Anzahl der Keypoints im Bild (Abb.25). In Frame 120 ist links bereits ein Bereich ohne Übereinstimmungen zu erkennen, obwohl dieser dem Ausgangsbild sehr ähnlich sieht.

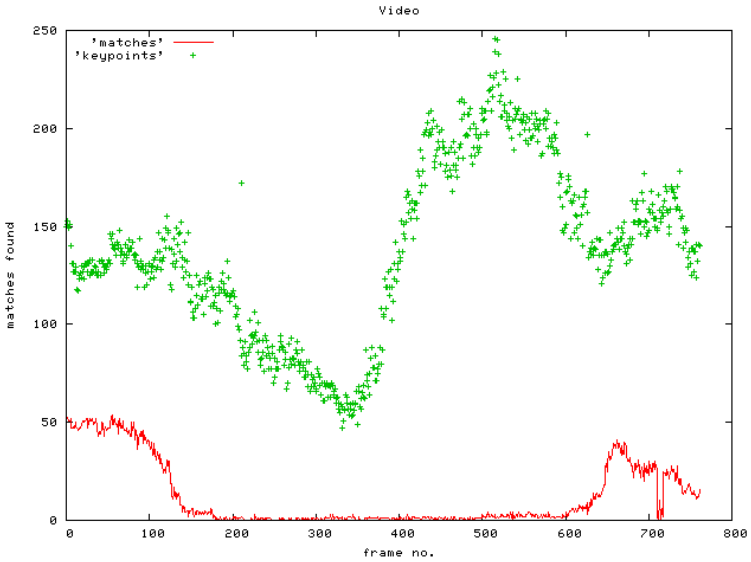


Abbildung 25: Plot: Drehung um die Hochachse

3.3 Geschwindigkeit und Speicherverbrauch

Bei Navigationsaufgaben bekommt auch die Berechnungsgeschwindigkeit eine Bedeutung. Roboter, die sich schnell bewegen können, sollten nicht stehen bleiben müssen um sich zu orientieren. Sie sollten ebenfalls nicht mit veralteten Informationen navigieren.

Die folgenden Diagramme (Abb.26,27) zeigen auf der x-Achse die Pixelzahl des Bildes in der Horizontalen (die Auflösung lässt sich aus (Tab.1) entnehmen), auf der y-Achse sind die Anzahl der gefundenen Keypoints bzw. die Zeit in Millisekunden, die für die Berechnung benötigt wurde, abgetragen.

Tabelle 1: Auflösungen

Bildauflösung	Pixelzahl
320x240	76800
640x480	307200
800x600	480000
1024x768	786432
1600x1200	1920000

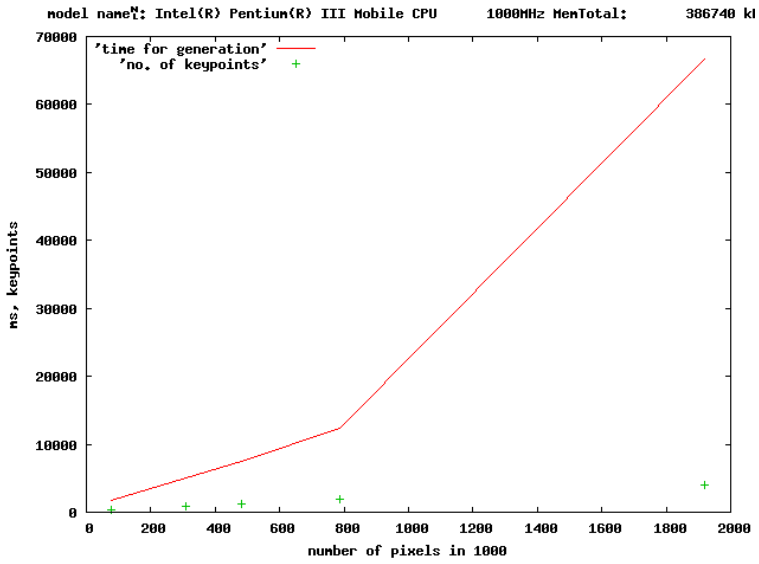


Abbildung 26: Geschwindigkeit Mobile Pentium 3

Wenn die Berechnungen auf einem Mobile Pentium 3 mit 1GHz Prozessortakt und 384 MB Arbeitsspeicher durchgeführt werden, dann sind die Grenzen schnell erreicht. Auf dem Plot erkennt man einen deutlichen Anstieg der Berechnungsdauer ab einer Bildgröße von 1024x768 (Abb.26). Es ist davon auszugehen, dass dieser von einem Mangel an Arbeitsspei-

cher herrührt und der Rechner an dieser Stelle begonnen hat Teile des Arbeitsspeichers auf die Festplatte auszulagern. Programme, die auf die ausgelagerten Bereiche zugreifen, werden dadurch deutlich verlangsamt.

Der hohe Speicherbedarf begründet sich durch die Generierung der Keypointkandidaten. Die errechneten Oktaven werden im Speicher gehalten, bis alle Keypoints generiert wurden. Eine Oktave beinhaltet das Originalbild in Graustufen, jeder Bildpunkt wird als double Wert (8 Byte) repräsentiert. In jeder Oktave befinden sich zusätzlich zu dem Originalbild 6 Gauss und 5 DoG Bilder, welches den Speicherverbrauch vervielfacht. Außerdem werden noch Oktaven in jeweils halber Auflösung errechnet, bei einem Bild von 1024x768 Pixeln gibt es zum Beispiel 6 Oktaven (1024x768, 512x384, 256x192, 128x96, 64x48, 32x24).

Den Speicherverbrauch kann man der Tabelle entnehmen (Tab.2). Es wird nur die Größe der Bilddaten ansich behandelt, nicht enthalten sind die Größe der Klassenstrukturen, die die Bilder beinhalten oder temporäre Daten, die nicht für die gesamte Keypointberechnung im Speicher gehalten werden. Ein Beispiel für solche temporären Daten sind die Magnituden und Richtungen für die Keypointberechnung, die ebenfalls in einer Bildstruktur gespeichert werden.

Tabelle 2: Speicherverbrauch in MB ($Basis : 1024$)

Bildauflösung	Speicher pro Bild (Originalauflösung)	Speicherverbrauch Alle Oktaven (je 12 Bilder)
320x240	$\approx 0,5$	$\approx 9,3$
640x480	$\approx 2,4$	$\approx 37,5$
800x600	$\approx 3,7$	$\approx 53,8$
1024x768	≈ 6	$\approx 96,1$
1600x1200	$\approx 14,6$	$\approx 229,6$

Bei einem besser ausgestatteten Rechner gibt es keinen deutlichen Anstieg, da ausreichend Speicher zur Verfügung steht (Abb.27). Dafür wird deutlich, dass die Berechnungsdauer stärker steigt als die Anzahl der im Bild gefundenen Keypoints. Die längere Berechnungsdauer resultiert aus der quadratisch steigenden Pixelzahl bei höheren Auflösungen, da beim Anwenden des Gaussfilters Berechnungen für jedes einzelne Pixel des Bildes durchgeführt werden müssen.

Die Berechnungsdauer steigt fast linar gegenüber der Pixelzahl bei einer unterschiedlichen Zahl der gefundenen Keypoints (exponentiell gegenüber Verdoppelung der Auflösung). Dieses zeigt, dass die Kandidatensuche der Keypoints die meiste Rechenzeit in Anspruch nimmt, während die Generierungszeit der Keypoints vergleichsweise gering ist.

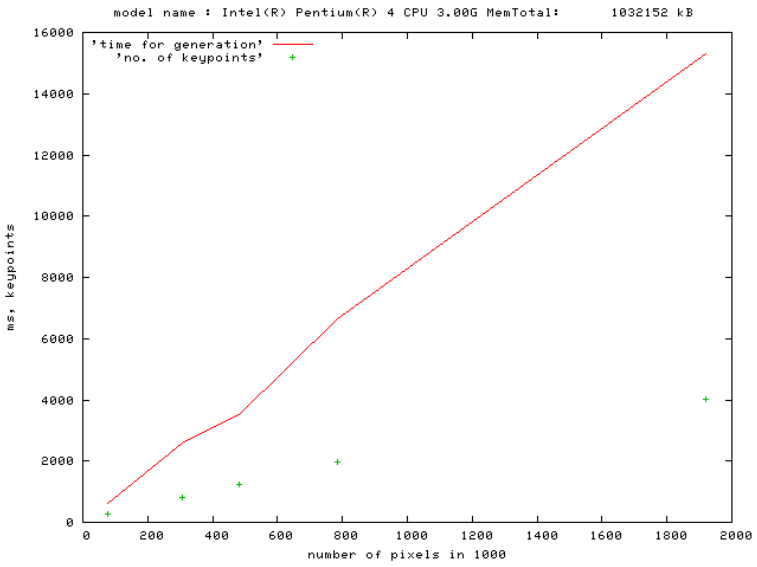


Abbildung 27: Geschwindigkeit Pentium 4

3.4 Auswirkungen der Bildgröße auf die Zahl der Keypoints

Der Test wurde mit einem Ausgangsbild von 640x480 durchgeführt, welches für die Berechnungen mit anderen Auflösungen dementsprechend vergrößert bzw. verkleinert wurde. Selbst bei einem algorithmisch vergrößerten Bild steigt die Zahl der gefundenen Keypoints mit der Größe des Bildes (Abb.28). Bei der Benutzung mit gering auflösenden Kameras könnte sich eine Vergrößerung des Bildes vor den Berechnungen lohnen.

Diese Option ist bereits in der SIFT library [Nowb] implementiert und wurde ebenfalls portiert. Es bleibt aber zu bezweifeln, dass dieses sehr hilfreich ist und das Vorhandensein von mehr Keypoints auch zu mehr Qualität in den richtig zugeordneten Übereinstimmungen führt. Auch möglich ist, dass die ohnehin schon vorhandenen Keypoints vervielfacht werden, wofür die Ergebnisse im Test sprechen (Kap.3.2.2 Größe). Das Diagramm (Abb.28) zeigt nur einen leichten Anstieg an Übereinstimmungen zu dem Originalbild ab einer Größe von 160%. Da aber die Geschwindigkeit darunter leidet und der Speicherverbrauch stark ansteigt, ist die Vergrößerung eine Entscheidung, die im Einzelfall getroffen werden muss.

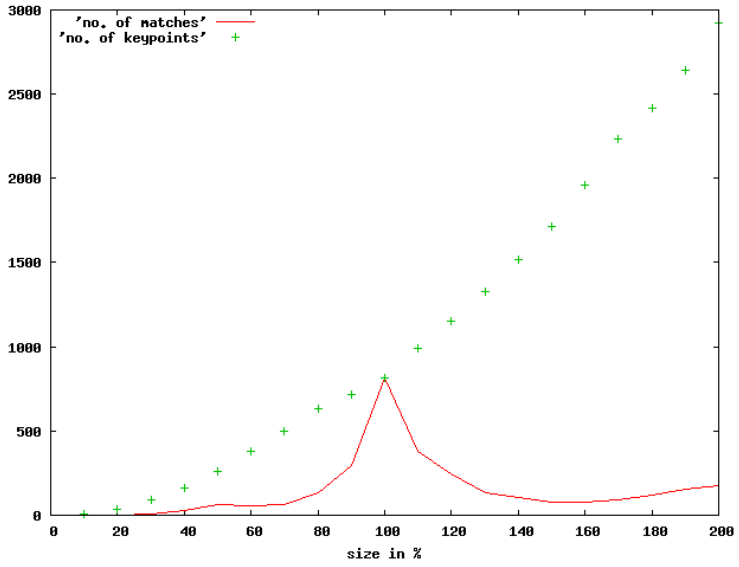


Abbildung 28: Auswirkung Bildgröße auf Keypointzahl

Der Abfall bei Bildern, die größer sind als das Referenzbild lässt sich aus der Tatsache Erklären, dass bei der Keypointgenerierung nur kleine Oktaven berechnet werden, nicht aber solche, die größer sind als das Referenzbild.

3.5 Bildkompression

3.5.1 JPG

Da das Finden der Keypointkandidaten sehr rechenaufwendig ist, könnte es nötig sein die Berechnungen auf einem externen Rechner durchzuführen. Es könnte sinnvoll sein das zu übertragende Bild zu komprimieren. Nicht alle Komprimierungsverfahren sind verlustfrei wie png [WWWC03]. JPEG Komprimierung [Uni92] ist nicht verlustfrei, aber recht verbreitet. Sie soll exemplarisch auf ihre Auswirkungen auf die Wiedererkennung der Features untersucht werden.

Abbildung 29: Bildqualität Stufe 85 und 10

Für den Test wurde ein Originalbild (JPEG) in unterschiedlichen Kompressionsstufen gespeichert und versucht die errechneten Keypoints im Originalbild wiederzufinden. Die RANSAC Filterung war deaktiviert, da es sich um das gleiche Bild in verschiedenen Größen handelt und somit auch keine Transformationen des Bildes vorkommen können, die für RANSAC inkonsistent sind. Doppelt zugewiesene Keypoints wurden allerdings entfernt.

Als Ausgangsbild wurde ein Bild mit der Qualität 85 gewählt. JPEG definiert die Qualitätseinstellung von eins (schlechteste) bis 100 (beste).

Der Plot zeigt eine deutliche Abnahme der wiedergefundenen Punkte gegenüber dem Originalbild bei höherer Kompression. Bemerkenswert ist, dass bei gleicher Kompression (Qualitätsstufe 85) deutlich mehr Matches gefunden werden als bei anderen Kompressionsstufen (Abb.30). Sogar bei besseren Qualitäten werden nicht mehr oder gleich viel Übereinstimmungen gefunden. Dieses suggeriert eine gute Performanz bei der Wiedererkennung von Keypoints solange die Kompressionsstufe gleichbleibend ist.

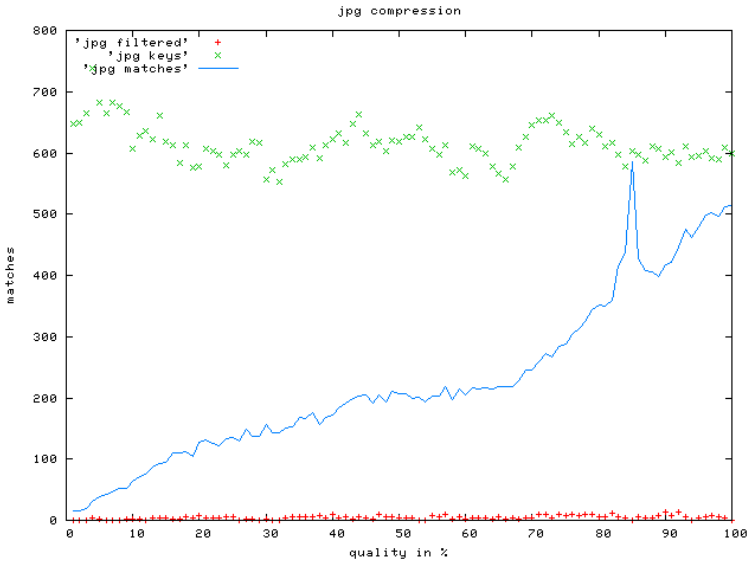


Abbildung 30: Bildqualität JPG

3.5.2 Video

Für den Test der Videokompression wurden künstlich zwei Videos erstellt, die keine Bewegungen beinhalten. Jedes Videoframe zeigt das Bild, das auch zuvor genutzt wurde (Kap.3.1 Experimente mit künstlich veränderten Bildern). Die beiden Videos wurden mit unterschiedlichen Methoden komprimiert: DivX und Motion JPEG.

Auch bei Videokompression kommt es zwangsläufig zu Veränderungen in der Bildqualität, allerdings können die Keypoints weiterhin einander zugeordnet werden. Es wurden unterschiedliche Tests durchgeführt (Abb.31).

Da Videokompression teilweise auf Bewegungen der Kamera aufbaut, kann es sein, dass dieses Experiment nicht repräsentativ für in der Realität aufgenommene Videos ist. Trotzdem lassen sich Rückschlüsse auf die Auswirkungen der Kompression auf die Eigenschaften der SIFT- Features ziehen.

Das aktuelle Videobild wurde mit dem Originalbild, mit dem das Video erstellt wurde, verglichen. Die Kurven im Plot ('divx' und 'mjpg') repräsentieren diesen Test. Auch wurde versucht, die Keypoints des jeweils erstes Bildes der Videos in den nachfolgenden wiederzufinden ('divx first' und 'mjpg first'). Die Kurve 'bmp' zeigt die Keypoints des Originalbildes, 'divx keypoints' und 'mjpg keypoints' die Keypointzahl im aktuellem Bild.

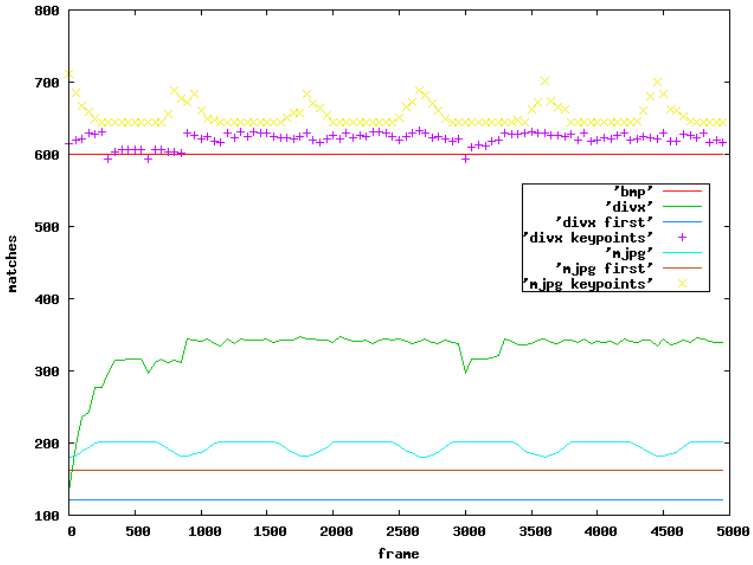


Abbildung 31: Bildqualität Video

Die DivX Kompression zeigt die höhere Qualität in Bezug auf die Wiedererkennung und im Vergleich mit dem MotionJPEG Video. Bei DivX steigt die Bildqualität und die Menge der gefundenen Übereinstimmungen zu Anfang des Videos an. Die höhere Anzahl von Keypoints in den Videos lässt sich durch die Kompression hervorgerufener Artefakte erklären. Oft sind in komprimierten Videos größere Quadrate einer Farbe erkennbar, welche Kanten verursachen, die im Originalbild nicht vorhanden sind.

Diese können, wie echte Kanten auch, in einem Keypoint resultieren. Dieses Phänomen muss nicht weiter berücksichtigt werden, da die Vergleiche mit dem ersten Bild des Videos konstant die gleiche Anzahl an Übereinstimmungen finden. Dieses zeigt, dass die "falschen Keypoints" keine Auswirkung auf das Endergebnis haben.

In den vorangegangenen Versuchen (Kap.3.2 Experimente mit Bildfolgen) wurde häufig ein ausreichendes Ergebnis erzielt, obwohl mit dem 1. Bild verglichen wurde. Ohne Kompression ist von einer deutlich höheren Menge an invarianten Keypoints im Bild auszugehen. Die Anfangsbilder in den Tests, die mit Video durchgeführt wurden, beinhalten alle unter 200 Keypoints. In diesem Experiment stieg die Matchzahl, also die Zahl der wiedererkennbaren Keypoints, bei ca. 150 im ersten Bild auf über 300. Somit ist bei der realen Anwendung der Bibliothek mit besseren Ergebnissen zu rechnen, als bei den besagten Tests mit komprimierten Videos.

Um dieses zu verdeutlichen wurde der Größentest (Kap.3.2.2 Größe) erneut durchgeführt. Allerdings wurde dieses Mal mit dem 120. Frame verglichen (Abb.32) anstatt mit dem ersten (Abb.19). Es gibt einen Unterschied zwischen den Diagrammen, da Frame 120 auch mit sich selbst verglichen wird, Frame 0 im anderen Test nicht. Bei dem Vergleich mit Frame 120 sind vor und nach diesem deutlich mehr Übereinstimmungen gefunden worden (ca. 150) als bei dem Test, der mit Frame 0 verglichen hat (ca. 100).

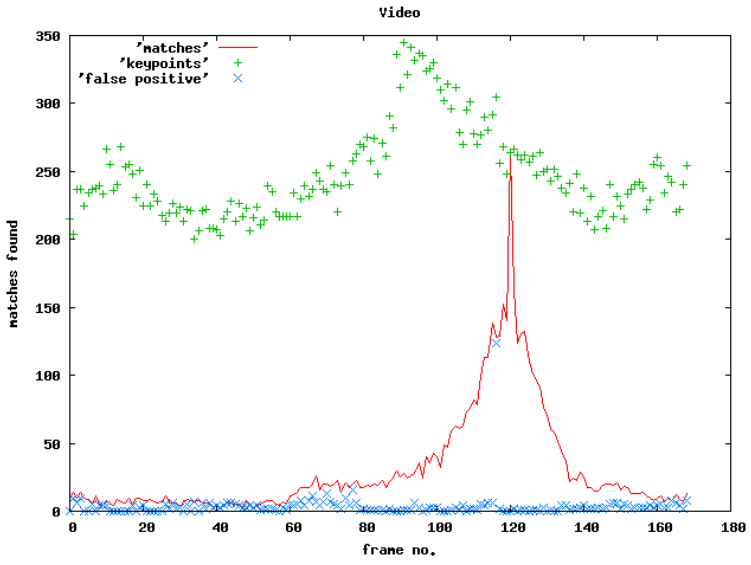


Abbildung 32: Plot: Größe, Vergleich mit Frame 120

4 Navigation

In den vorangegangenen Experimenten wurde jeweils nur ein Teilaspekt der SIFT- Features betrachtet. Bei diesem Test kommen alle diese zusammen.

Um eine Verwendbarkeit zu Navigationszwecken zu demonstrieren wurde eine Implementierung in Routen umgesetzt. Eine Route besteht hierbei aus Zuständen, denen Aktionen zugeordnert werden können, die dann in einen neuen Zustand führen. Ein Zustand definiert sich aus den Keypoints eines Kamerabildes. Ein Roboter könnte dieser Route folgen, indem er die Aktionen ausführt, die zu dem von ihm erkannten Zustand gehören.

4.1 Vorgehensweise

Die Keypoints von zuvor aufgenommenen Bildern wurden in der Routenstruktur gespeichert. Bei dem Versuch die Zustände zu erkennen, wurde das Kamerabild mit den Keypoints aus der Routenstruktur verglichen. Die Zustandsunterscheidung beruhte ausschließlich auf der Anzahl der gefundenen Übereinstimmungen mit den Keypoints der Zustände. Bei weniger als 10 gefundenen Übereinstimmungen, wurde der Zustand als nicht erkannt betrachtet.

Für den Test wurde ein Video einer Webcam genutzt und mittels DivX komprimiert. Es hat eine Auflösung von 320x240 Pixeln. In dem Video treten starke Verzerrungen auf und die Bildqualität ist relativ schlecht bezeichnen (Abb.33,35). Das Video wurde bewusst so gewählt, um die Robustheit der Zustandsbestimmung mittels SIFT- Features beurteilen zu können. Für die Zustände wurden größtenteils Bilder verwendet, die möglichst wenig Verzerrungen beinhalten (Abb.35).

Die für diesen Test verwendete Kamera war nicht kalibriert. Aufgrund der Linsenverzerrung kann es sein, dass bei schon leichten Drehungen sich gerade Kanten krümmen bzw. nicht mehr gerade erscheinen [Jae05, Kap.7.4.5]. Eine Kalibrierung hätte es ermöglicht es die Verzerrungen algorithmisch zu entfernen. Da Kanten maßgeblich für die Keypoints sind, ist die Navigationsaufgabe durch die fehlende Kalibrierung zusätzlich erschwert. Übereinstimmungen, die nicht gefunden wurden, hätten unter Umständen mit einer kalibrierten Kamera gefunden werden können.

Der RANSAC Filter konnte auch nicht verwendet werden, da duch die Verzerrung die Keypoints ihre relative Position zueinander verändern, wenn die Kamera bewegt wird. Es werden mit dem aktuellem Transformationsmodell (affine Transformation) des RANSAC Filters bei einer unkalibrierten Kamera zu viele eigentlich richtige Übereinstimmungen entfernt.

Zum Testen der Wiedererkennung der Zustände wurde das gleiche Video genutzt, mit dem die Zustände auch definiert wurden. Es kam also zu exakten Übereinstimmungen zwischen den Zuständen und den Bildern des Videos im Testlauf. Dieses stellt für die Ergebnisse des Tests aber kein Problem dar. Die Videobilder vor und nach dem Zustand unterscheiden sich von dem Zustandsbild – sie könnten auch aus einer zweiten Aufnahme stammen.

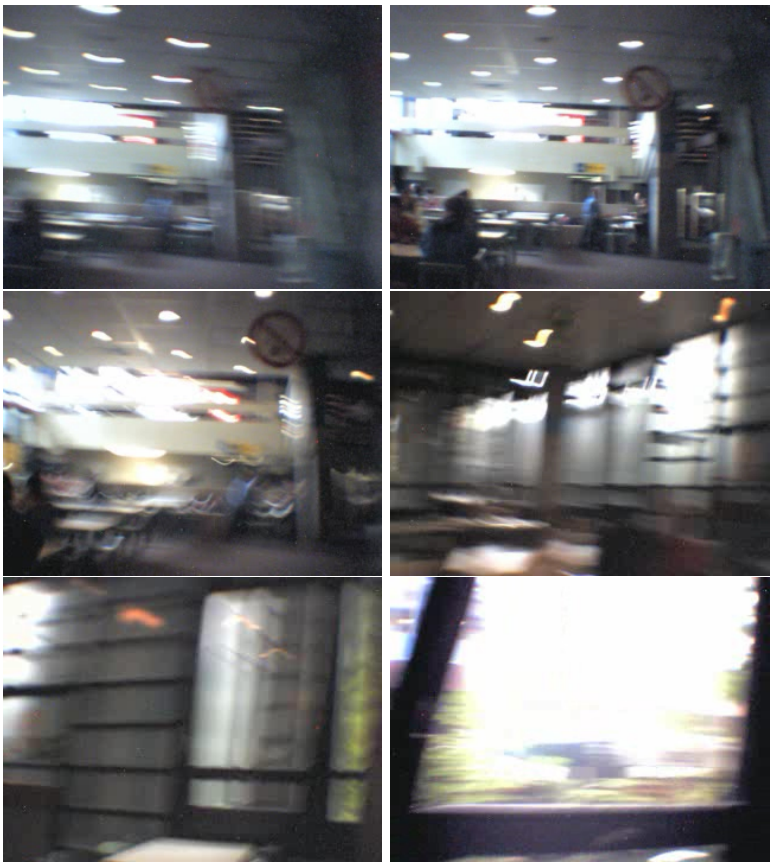


Abbildung 33: Navigation: Testvideo für die Navigation

4.2 Navigationstest

Getestet wurde auf zwei Arten:

Erkenne den aktuellen Zustand : Verglichen wurde das zu testende Bild mit den Keypoints aller Zustände. Der Zustand mit den meisten Übereinstimmungen wird als der aktuelle angesehen, falls mehr als 10 Matches gefunden wurden.

Prüfe ob der nächste Zustand erreicht wurde : Verglichen wurde das zu testende Bild mit den Keypoints des aktuellen Zustandes und denen des nächsten. Falls es mit dem nächsten Zustand mehr Übereinstimmungen gab als mit dem aktuellen und diese mehr als 10 waren wurde in den folgenden Zustand gewechselt.

Obwohl die Videoqualität sehr schlecht ist, funktioniert die Bestimmung des Zustandes recht zuverlässig. Die Diagramme zeigen die Zuordnung des aktuellen Videoframes zu dem Zustand. Die Frames, welche die Zustände definieren, sind durch drei übereinander angeordnete Sterne markiert. Einige Zustände werden schon vor Erreichen des zustandsdefinierenden Frames erkannt, da schon die für den Zustand relevanten Bildelemente

sichtbar sind.

Bei der Erkennung des aktuellen Zustandes wird deutlich, dass es zu Fehlzuordnungen kommt und häufig auch gar kein passender Zustand gefunden werden kann (Zustand = -1) (Abb.34). Dieses ist auf die Verzerrungen der Kanten (Abb.33) und auf Drehungen um die Hochachse zurückzuführen, bei denen neue Bildbereiche ins Blickfeld gelangen, die nicht in einem Zustand vorkommen.

Trotzdem funktioniert in der Umgebung des zustandsdefinierenden Frames die Zustandszuordnung hinreichend gut. Es kommt nur selten zu Zuordnungen zu falschen Zuständen, die Zuordnung bleibt eher ergebnislos.

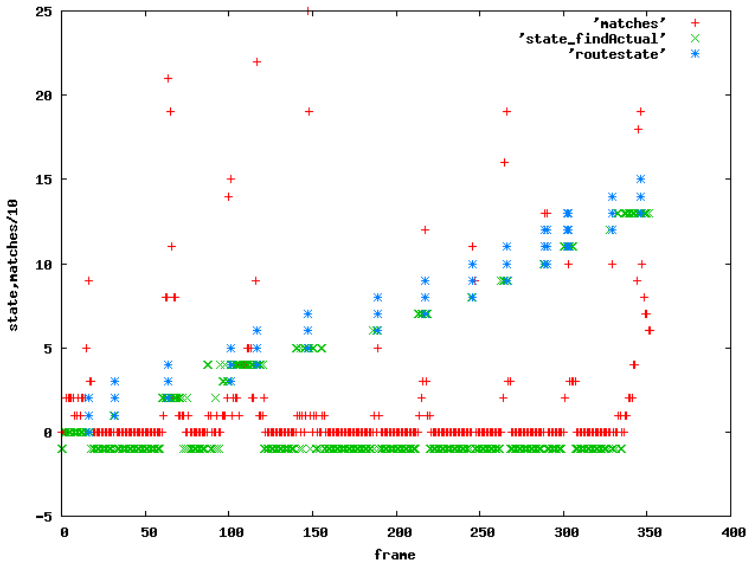


Abbildung 34: Navigation: Erkenne aktuellen Zustand

Zwischen Zustand drei und vier (ca. Frame 100) kommt es zu den größten Irritationen in der Zuordnung, da die Bilder der Zustände sehr ähnlich sind (Abb.35).

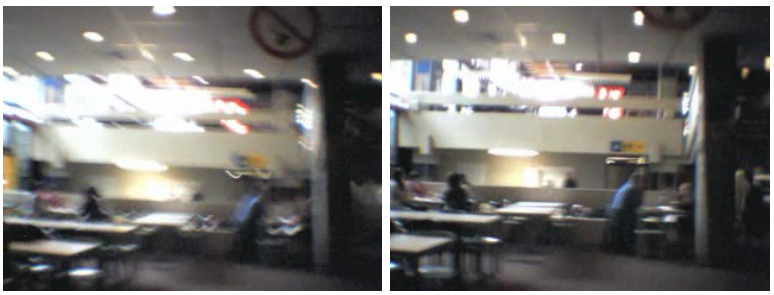


Abbildung 35: Navigation: Zustände drei und vier

Bei der Zuordnungsweise, bei der nur geprüft wurde ob der nächste Zustand erreicht ist, wurden deutlich bessere Ergebnisse erzielt (Abb.36).

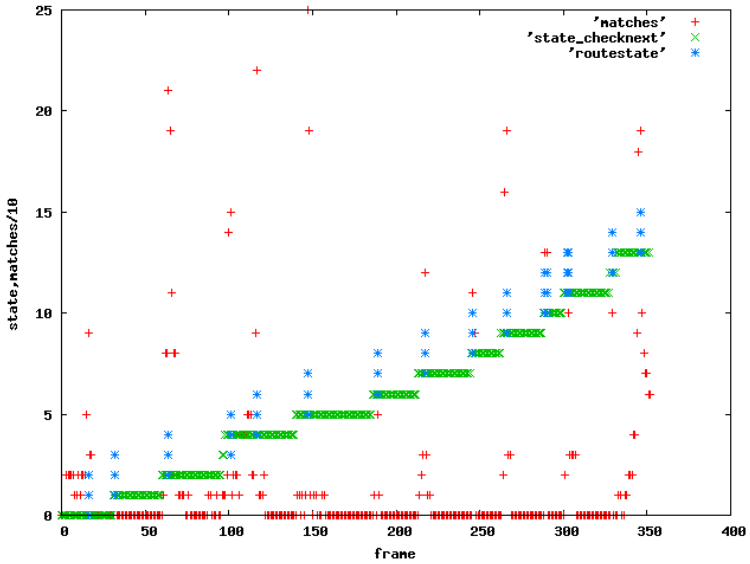


Abbildung 36: Navigation: Prüfung auf den nächsten Zustand

Bei dem Übergang von Zustand drei nach vier kommt es wie auch beim ersten Test zu einer falschen Zuordnung, da auch hier schon die relevanten Bildteile aus dem vierten Zustand sichtbar sind und es somit auch zu einem vorzeitigen Zustandsübergang kommt. Sonst gibt es nur eindeutige Zuordnungen von Kamerabildern zu den Zuständen.

5 Ergebnis

SIFT Features sind durchaus für die Navigation verwendbar, wenn auch einzelne Features bzw. Matches nicht für eine Lokalisierung oder Feststellung, wie zum Beispiel der Blickrichtung, ausreichen. Die Features sind nur in einer Menge aussagekräftig, da es bei der Suche zu falsch positiven Übereinstimmungen kommen kann. Mittels der RANSAC Filterung für die Matches können aber inkonsistente Übereinstimmungen entfernt werden. Starke Größenänderungen machen die Wiedererkennung schwieriger.

Zu beachten ist, dass der RANSAC Filter eine homogene Transformation für das gesamte Bild annimmt. Es können allerdings mehrere der errechneten Bewegungsmodelle des Filters der Realität entsprechen. Dieses wird hervorgerufen durch verschieden weit entfernte Objekte im Bild, die sich bei Bewegung der Kamera unterschiedlich schnell im Bild bewegen. Diese unterschiedlichen Bewegungen würden in der gleiche Anzahl von richtigen Bewegungsmodellen resultieren. Libsift verwendet nur das Modell, dass die für meißten Übereinstimmungen gültig ist. Alle anderen werden als falsch betrachtet.

Es wurden keine Navigationsalgorithmen auf einem Roboter getestet, da diese für die Aussagen über die Verwendbarkeit nicht notwendig sind. Bei dem Navigationstest ohne Roboter wurde gezeigt, dass die SIFT-Features gut dafür geeignet sind, Zustände oder Positionen in einem vorgegebenen Pfad zu erkennen, so dass man diesen gegebenenfalls auch mit einem Roboter folgen kann.

Problematisch für die Navigation bleiben Geschwindigkeit und Speichernutzung. Um Navigationsalgorithmen auf Robotern direkt auszuführen, muss auch dessen Steuerungsrechner schnell genug sein, um neben den Steuerungsaufgaben auch die SIFT Freatures zu berechnen. Viele Roboter werden aber von Rechnern gesteuert, die auf geringen Energieverbrauch optimiert sind und somit auch nicht viele Rechenkapazitäten zur Verfügung haben. Um eine gute Wiedererkennung der Features zu gewährleisten, sollten Bilder in einer möglichst großen Auflösung verwendet werden.

Trotzdem werden auch bei geringen Auflösungen ausreichend Keypoints gefunden, um eine Navigation möglich zu machen oder die Selbstlokalisation unter Zuhilfenahme von anderen Sensoren mittels Sensorfusion [CD93] zu unterstützen.

6 Ausblick

Bei der Navigation kann eine Einbindung von Keypointentfernungen die Entscheidung für einen Zustand stark verbessern. Vor allem wenn der Roboter sich längere Zeit geradeaus bewegt und somit über mehrere Zustände hinweg die gleichen Punkte im Bild sichtbar sind.

Die Berechnung der Entfernung könnte mittels einer Stereokamera geschehen. Zunächst könnten die Features beider Bilder berechnet und deren Übereinstimmungen gesucht werden. Da die Keypointpositionen auf Subpixelebene berechnet werden [BL02] kann so die Entfernung eines Keypoints von der Kamera ermittelt werden. Ebenso könnte man für die Zustandsbestimmung eines Routengraphen [WKBH00] Panoramabilder einer 360^o Kamera zur exakten Positionsbestimmung benutzen [Sch06].

Um eine höhere Invarianz gegenüber Blickwinkeländerungen zu erreichen könnte man Objekte, die auch aus leicht verschiedenen Blickwinkeln unterschiedliche Keypoints ergeben entfernen. Wie zum Beispiel die von Pflanzen, bei denen die Blätter schon aus leicht unterschiedlichen Blickwinkeln ein anderes Kantenmuster haben. Auch Keypoints die leicht zu falsch positiven Übereinstimmungen führen, könne man so von den Berechnungen ausnehmen. Dieses könnte ebenfalls mit einer Stereokamera geschehen, indem man nur die Keypoints, die in dem linken und rechten Bild gefunden wurden, in die weiteren Berechnungen mit einbezieht.

Eine Beschleunigung der Bildverarbeitungsroutinen, insbesondere des häufig verwendeten Gaussfilters, könnte das gesamte Verfahren deutlich beschleunigen. Hierzu könnte man eine manuelle Geschwindigkeitsoptimierung des Quelltextes vornehmen oder Bibliotheken verwenden, die häufig verwendete Aufgaben schneller lösen können (zum Beispiel das Errechnen der Gaussbilder). Ebenso wäre es möglich den Speicherverbrauch bei der Suche der Keypointkandidaten verringern, indem man Bilder (Gauss oder DoG) erst erstellt, wenn sie benötigt werden und wieder löscht, wenn sie nicht mehr gebraucht werden. Momentan werden diese von libsift und der Portierung komplett im Vorraus berechnet und bis zum Ende aller Berechnungen im Speicher behalten.

Desweiteren ließen sich dicht beieinander liegende Features zu Featuremengen zusammengefasst werden, so dass nicht ein gesamtes Bild erkannt werden muss. Auf diese Weise wäre es möglich, eine autonome Landmarkenerkennung auf Basis der Keypoints zu realisieren.

Mit einer Stereokamera kann eine direkte Positionsbestimmung der Featuremengen im Raum realisiert werden. So könnten Featuremengen aus unterschiedlichen Blickwinkeln, die an gleichen Position im Raum sind, einem "Objekt" zugeordnet werden. Das Objekt könnte nun in unterschiedlichen Positionen erkannt werden. Vorraussetzung hierfür ist eine genaue Positionierung des Roboters im Raum während der Objekterkennung. Eine solche Objektrepräsentation kommt wahrscheinlich auch im menschlichen Gehirn vor. Versuche mit Menschenaffen deuten darauf hin, dass für das Erfassen unterschiedlicher Ansichten des gleichen Objektes, die zuständigen Areale im Hirn nebeneinander liegen. Somit kann eine Wiedererkennung des Objektes unabhängig des Blickwinkels gewährleistet werden [Tan02].

Mit dieser Art der Repräsentation von Objekten könnte es einem Roboter möglich sein Schlüsse zu ziehen, wie sich die Keypoints eines Objektes bei Bewegung verändern, ohne dass er diese Bewegung in der Realität ausführt. So könnte er prüfen, ob seine realen Bewegungen mit den "imaginär" durchgeführten übereinstimmen.

A Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

B Änderungen an libsift

Hier werden nur Änderungen erwähnt, die über eine reine Portierung hinaus gehen. Es handelt sich allerdings nicht um Änderungen der Funktionalität oder Funktionsweise, sondern um interne Änderungen an Speicherung und Zugriff der Daten.

Neue Klassen: Da die C#::ArrayList recht häufig genutzt wird, wurde eine Klasse C++::ArrayList hinzugefügt, die das Verhalten der C#::ArrayList nachempfindet. Somit musste nicht jede Stelle einzeln geändert werden. Die ArrayList erbt dabei von der C++::vector Klasse und bildet dabei größtenteils die Methodennamen der C#::ArrayList auf die vererbten Methoden von C++::vector ab (z.B. C#::ArrayList.Add() auf C++::vector.push_back()). Die ArrayList wurde zunächst nur für Objekte der Klasse "Point" vorgesehen, im späterem Verlauf wurde sie in ein Template umgeschrieben, damit sie auch mit beliebigen anderen Klassen genutzt werden kann.

ImageMap: Die interne Datenhaltung der Pixel wurde von einem zweidimensionalem Array auf ein ein- dimensionales umgestellt.

KeypointXML: Die Klasse nutzt im Original die vom .NET Framework bereitgestellte C#::XmlSerializer Klasse um KeypointN Objekte in Dateien zu speichern. Da der C#::XmlSerializer nicht zur Verfügung steht, werden die Objekte binär und nicht in einer XML [WWWC04] Struktur gespeichert.

MultiMatch (Matches.h/.cpp): Die von C#::Array erbende "KeypointXMLList" mit dem Namen "keysets", wurde in eine C++::ArrayList geändert, da es kein C++::Array mit dynamischer Größenänderung gibt.

MatchKeys: Die Funktion C++::FilterJoins benutzt eine C++::map<int*,int> anstatt des C#::hashtable bei dem die (Referenzen) auf Keypoints genutzt wurden um dopplet zugewiesene Matches zu entfernen. Zur Identifizierung der einzelnen Keypoints wird nun die Speicheradresse des jeweiligen Descriptors verwendet.

C CD

D Referenzen

Literatur

[ABP+05] Albrecht, Martin ; Backhaus, Till ; Planthaber, Steffen ; Stöppler, Henning ; Spenneberg, Dirk ; Kirchner, Frank: AIMEE: A Four-Legged Robot for RoboCup Rescue. In: *CLAWAR 2005*, 2005

[Ben75] Bentley, Jon L.: Multidimensional binary search trees used for associative searching. In: *Communications of the ACM* 24 (1975), Nr. 9, 509 - 517. http://portal.acm.org/

[BL02] Brown, M. ; Lowe, David G.: Invariant features from interest point groups. In: *British Machine Vision Conference*, 2002

[CD93] Crowley, J. L. ; Demazeau, Y.: Principles and Techniques for Sensor Data Fusion. In: *Signal Processing* 32 Nos 1-2 (1993), May, 5-27. http://www-prima.inrialpes.fr/Prima/Homepages/jlc/papers/SigProc-Fusion.pdf

[DK02] Desouza, G. N. ; Kak, A. C.: Vision for mobile robot navigation: a survey. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), Nr. 2, 237–267. http://www.cs.uml.edu/~holly/91.549/readings/desouza-kak-vision-survey.pdf

[DKS+05] Deans, J. ; Kunz, C. ; Sargent, R. ; Park, E. ; Pedersen, L.: Combined Feature Based and Shape Based Visual Tracker for Robot Navigation. (2005). http://marstech.jpl.nasa.gov/tdaPublications/CombindFeatureBasedandShapeBasedVisualTracker.pdf

[FB81] Fischler, Martin A. ; Bolles, Robert C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. In: *Communications of the ACM* 24 (1981), Nr. 6, 381 - 395. http://portal.acm.org/

[Jae05] Jaehne, Bernd: *Digitale Bildverarbeitung*. 6., berarbeitete und erweiterte. Springer-Verlag Berlin Heidelberg, 2005. – ISBN 3–540–41260–3

[Low04] Lowe, David G.: Distinctive image features from scale-invariant keypoints. In: *International Journal of Computer Vision, 60, 2 (2004)* (2004), 91-110. http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf

[MNM+04] Mundhenk, T. N. ; Navalpakkam, Vidhya ; Makaliwe, Hendrik ; Vasudevan, Shrihari ; Itti, Laurent: Biologically inspired feature based categorization of objects. In: *SPIE Human Vision and Electronic Imaging IX* Bd. 5292, 2004, 330-341

[MS05] Mikolajczyk, K. ; Schmid, C.: A performance evaluation of local descriptors. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27 (2005), Nr. 10, S. 1615–1630

[NBL03] Newman, P. ; Bosse, M. ; Leonard, J.: Autonomous feature-based exploration. In: *ICRA '03. IEEE International Conference on Robotics and Automation* Bd. 1, 2003, 1234-1240

[Nowa] Nowozin, Sebastian: *Image Matching Model*. http://user.cs.tu-berlin.de/~nowozin/autopano-sift/matchmodel.pdf. – Model for RANSAC used in libsift

[Nowb] Nowozin, Sebastian: *libsift*. http://user.cs.tu-berlin.de/~nowozin/libsift/. – sift library for the .NET framework

[Sch06] Schatz, Alexej: *Visuelle Navigation mit "Scale Invariant Feature Transform"*, Universität Bielefeld, AG Technische Informatik, Diplomarbeit, 2006. http://www.ti.uni-bielefeld.de/html/people/froeben/diploma_theses_pdfs/da_15_visuelle_nav_sift_050614.pdf

[SK02] Spenneberg, Dirk ; Kirchner, Frank: SCORPION: A Biomimetic Walking Robot. In: *Robotik 2002 No. 1679* (2002), 677-682. http://ag47.informatik.uni-bremen.de/downloads/papers/robotik2002-spenneberg.pdf

[Str97] Stroustrup, Bjarne: *The C++ Programming Language*. 3rd. Addison-Wesley, 1997. – ISBN 0–201–88954–4

[Tan02] Tanaka, Keiji: Neuronal Representation of Object Images and Effects of Learning. In: *Perceptual Learning*. A Bradford Book, the MIT Press, 2002. – ISBN 0–262–06221–6, S. 67–82

[TMD+06] Thrun, S. ; Montemerlo, M. ; Dahlkamp, H. ; Stavens, D. ; Aron, A. ; Diebel, J. ; Fong, P. ; Gale, J. ; Halpenny, M. ; Hoffmann, G. ; Lau, K. ; Oakley, C. ; Palatucci, M. ; Pratt, V. ; Stang, P. ; Strohband, S. ; Dupont, C. ; Jendrossek, L.-E. ; Koelen, C. ; Markey, C. ; Rummel, C. ; Niekerk, J. van ; Jensen, E. ; Alessandrini, P. ; Bradski, G. ; Davies, B. ; Ettinger, S. ; Kaehler, A. ; Nefian, A. ; Mahoney, P.: Winning the DARPA Grand Challenge. In: *Journal of Field Robotics* (2006). http://robots.stanford.edu/papers/thrun.stanley05.pdf. – accepted for publication

[Uni92] Union, International T.: *DIGITAL COMPRESSION AND CODING OF CONTINUOUS - TONE STILL IMAGES - REQUIREMENTS AND GUIDELINES*. http://www.w3.org/Graphics/JPEG/itu-t81.pdf. Version: 1992

[WKBH00] Werner, S. ; Krieg-Brückner, B. ; Herrmann, T.: Modelling navigational knowledge by route graphs. In:

Freksa, Ch. (Hrsg.) ; Brauer, W. (Hrsg.) ; Habel, Ch. (Hrsg.) ; Wender, K.F. (Hrsg.): *Spatial Cognition II*, 2000, S. 295 – 317

[WWWC03] World Wide Web Consortium, (W3C): *Portable Network Graphics (PNG) Specification*. http://www.w3.org/TR/2003/REC-PNG-20031110/. Version: 2, 2003

[WWWC04] World Wide Web Consortium, (W3C): *Extensible Markup Language (XML) 1.0*. http://www.w3.org/TR/2004/REC-xml-20040204/. Version: 3, 2004