Abschlussbericht Projekt Computervision

Teilnehmer:
Tim Jagla
Patrick Nierath

Betreuer:
Sebastian Schäfer

SS2012

Einleitung

Im Zuge der Veranstaltung Computervision im Sommersemester 2012 gab es ein übungsbegleitendes Projekt, in dem eine Software geschrieben werden sollte, welche Verkehrsschilder in Bilddaten erkennt.
Die Vorgehensweise und der genutzte Algorithmus sind hierbei frei wählbar und eigenständig umzusetzen gewesen. Die einzige Limitierung waren die zu evaluierenden Testdaten, welche gestellt wurden, um eine Vergleichsmöglichkeit zwischen den verschiedenen Gruppen zu haben und die Qualität des Ergebnisses bewerten zu können. Unser Ziel in diesem Projekt war natürlich eine möglichst erfolgreiche Templateerkennung mit einem bis dahin für uns unbekanntem Verfahren, welches uns aufgrund seiner vielen Vorteile (im nächsten Abschnitt weiter erläutert) und Komplexität attraktiv erschien - SIFT.
Zur Umsetzung unseres Programmes haben wir folgende Software und Bibliotheken benutzt:

- Microsoft Visual Studio 2010 Ultimate/Professional
- OpenCV
- Gantt (für Projektplanung)
- SVN

Geschrieben ist unserer Software in C++.

Verfahren

Bevor wir hier die genaue Arbeitsweise darstellen, ist es vermutlich sinnvoll zu klären, warum wir uns für diesen Algorithmus entschieden haben. Der Grund dafür sind zum einen das Interesse uns mit etwas Neuem uns bis dahin Unbekanntem zu beschäftigen und zum anderen vor allem die Vorteile des Verfahrens gegenüber vielen anderen, vermutlich etwas einfacher gestrickten Ansätzen Templates in Bildern zu erkennen. SIFT, ausformuliert **S**cale **I**nvariant **F**eature **T**ransform, ist invariant gegenüber:

- Rotation
- Skalierung
- Beleuchtungssituation
- Betrachtungsperspektive

Nachdem wir anfänglich versucht haben, den Algorithmus für uns selber zu implementieren und zu nutzen, dabei jedoch mehr oder weniger gescheitert sind, haben wir letztendlich die von OpenCV vordefinierten Funktionen verwendet.
Der Algorithmus selber von von David Lowe entwickelt und im Jahre 1999 veröffentlicht. Um die Komplexität besser zu erfassen, wird das Verfahren oft in mehrere Teile untergliedert.

Der erste Schritt im Algorithmus ist das Kreieren sogenannter *Scale Spaces*. Dabei nehmen wir das Ausgangsbild und unterziehen es einem Gaußfilter und *blurren* es.progressiv. Danach halbieren wir die Größe des Ausgangsbildes und *blurren* es wieder. So können wir auf mehreren Auflösungsstufen nach unseren Keypoints suchen um gefundene aus vorherigen Auflösungen zu bestätigen oder zu verwerfen. Illustriert ist dieser Teil des Verfahrens in Abb.1. Oft wird das Ausgangsbild in seiner Größe anfangs auch verdoppelt, um im weiteren Verfahren mehr Keypoints zu erhalten.

Im nächsten Schritt würde man die Bilder eigentlich einem *LoG*-Filter unterziehen, um Ecken und Kanten in dem Bild zu lokalisieren. Da dies relativ rechenaufwändig ist, wird bei SIFT etwas getrickst. Von den eben erstellten geblurrten Bildern werden paarweise Differenzbilder bebildet, die letztendlich fast den gleichen Effekt wie ein *LoG*-Filter haben., genannt *DoG* (Difference of Gaussian) - Images. So erhalten wir einfach und schnell unser gewünschtes Ergebnis. Außerdem hat diese Vorgehensweise einen weiteren positiven Effekt, Die *DoGs* sind skalierungsinvariant.

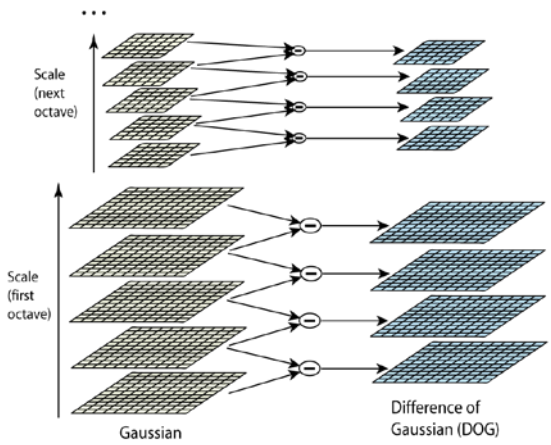

Abb.1(links)) Die verschiedenen Auflösungsstufen mit Blur an einem Testbild. Abb. 2(rechts))Erstellen der DoG-Images.

Im weiteren Verlauf suchen wir nun die Maxima bzw. Minima unseres Bildes, und zwar mit Hilfe der eben erstellten *Difference of Gaussian Images*. Um die Keypoints auch zu verifizieren nehmen wir zum einen die benachbarten Pixel zum Prüfen dazu, sowie auch die benachbarten Pixel aus dem Bild über dem aktuellen und unter dem aktuellen. Das heißt wir erstellen ein Maxima/Minima-Bild aus 3 *DoGs*. Folgende Abbildung verdeutlicht dies in einfacher Form.

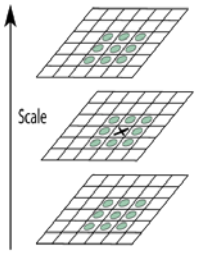

Figure 2: Maxima and minima of the difference-of-Gaussian images are detected by comparing a pixel (marked with X) to its 26 neighbors in 3x3 regions at the current and adjacent scales (marked with circles).

Abb.3) Bestimmen der Keypoints

Auf diese Art und Weise erhalten wir eine Menge Keypoints, die aber nicht alle relevant für uns sind. Viele der Punkte liegen auf Kanten oder an Ecken oder aber auf flachen Regionen, die aufgrund des geringen Kontrastes ebenfalls uninteressant für uns sind. Insofern werden diese Punkte eleminiert.
Um nun auch unsere gewünschte Invarianz gegenüber Rotation abzusichern, bestimmen wir für jeden Keypoint eine Orientierung. Diese Keypoints erhalten in einem weiteren Schritt, dessen Erläuterung an dieser Stelle nicht unbedingt nötig ist und den Rahmen sprengen würde, einen persönlichen Fingerprint. Schließlich können die Keypoints zweier Bilder verglichen und auf Matches geprüft werden.

Evaluierung

Für die Evaluierung der Testdaten haben wir unsere Ergebnisse in eine Textdatei geschrieben. Diese hatte in etwa das Format von der Textdatei für die Übersicht der Testdaten, sodass wir auf einen Blick genau sagen konnten, welche Schilder wir im jeweiligen Testbild gefunden haben, und sie somit auch auswerten konnten. Hier erst einmal eine Übersicht über unsere Ergebnisse:

Image	Name Schild	korrekt positiv	falsch negativ	Sensitivität (%)	# falsch positiv / DS
1	Stop, 30, 120	1	0	100	2
2	Stop	0	2	0	1
3	Gefahr	1	1	50	0
4	60,120	1	1	50	1
5	30	1	0	100	0
6	Gefahr, Vorfahrt	1	1	50	1
7		0	0	100	0
8	Gefahr	1	1	50	0
9	Stop	1	0	100	0
10		0	2	0	0
11	30	1	1	50	0
12	Gefahr, Vorfahrt	1	2	33,3	1
13	120	0	0	100	1
14		0	3	0	0
15	Stop	0	2	0	1
16	Stop, 120	1	0	100	1
17	Stop, 30, 60, 120, Gefahr	1	0	100	4
18	120	0	1	0	1
19	Stop	1	0	100	0
20	120	0	1	0	1
21	30	0	1	0	1
22	Stop, Vorfahrt, Gefahr	1	0	100	2
23	120	1	1	50	0
24	Gefahr	0	0	100	1
25	Stop, 60	1	1	50	1
26	Stop, 120	1	0	100	1
27		0	2	0	0
28		0	1	0	0
29	60	1	0	100	0
30		0	2	0	0
				52,7766666667	0,7

Abb. 1) Übersicht über unsere Ergebnisse in Form einer Excel-Tabelle.

Die erste Spalte der obigen Tabelle enthält lediglich die Nummer des Testbildes aus dem uns vorgegebenen Datensatz. In der zweiten Spalte haben wir aufgeführt, welche Schilder im jeweiligen Bild gefunden wurden. Die anderen Spalten stellen dann die Werte gemäß der Beschreibung im Tabellenkopf dar.
Aus den Ergebnissen wird ersichtlich, dass wir im Durchschnitt eine Sensitivität von 52,77% haben, also immerhin noch bei mehr als der Hälfte der Testfälle die Schilder erkennen, und 0,7 falsche, also nicht vorhandene, Schilder in den Bildern finden.
Generell müssen wir sagen, und es wird ja auch in der Tabelle deutlich, dass die Resultate sehr schwankend waren. Allerdings konnten wir in manchen sehr schwierigen Fällen auch nur staunen, dass unser Algorithmus korrekt positive Schilder gefunden hat, wie z.B. in Abb.4.

Abb. 4a) Gefundene Matches des Schildes mit dem Bild ohne Filterung der Ergebnisse.

Abb. 4b) nach Filterung der Matches finden wir tatsächlich das schwer erkennbare 60km/h-Schild.

Nun stellt sich die Frage, warum denn unser Algorithmus nicht immer funktioniert hat, trotz seiner vielen Vorteile? Das hat mehrere Gründe:
Ein Aspekt ist, dass wir gewisse Feinheiten nicht abfangen bzw. im Rahmen dieses Projektes nicht weiter bearbeiten konnten. Da unser Algorithmus auf Feature-Vektoren beruht, gibt es gerade bei den Geschwindigkeitsbegrenzungsschildern ein paar Probleme. So werden markante Stellen der NULL sehr gut gefunden, allerdings kommt diese nicht nur bei 30, 60, 120km/h vor, sondern auch bei allen anderen Geschwindigkeitsbegrenzern, weshalb wir z.B. im folgenden Bild die 20 fälschlicherweise als 30, 60 und 120 erkennen.

Abb. 5) Ein für uns problematisches Bild. Es werden alle Schilder erkannt, obwohl keines der geforderten in dem Bild vorhanden ist.

Ein weiteres Problem ist in diesem Fall unser eigentlicher Vorteil der Invarianz gegenüber Rotationen. Dadurch gibt es sehr viele Verwechslungen und falsch positive Matches mit Schildern, die die Grundform des "Vorfahrt beachten" - Schildes haben. Hier dafür ein Beispiel:

Abb. 6) Da sich die Grundformen gleichen, gibt es auch hier einen falsch positiven Fund des Vorfahrt-Schildes.

Die grundlegenden Formen des Schildes in Abb.4 werden richtig erkannt, allerdings fällt das innere unter den Tisch und wird nicht weiter beachtet, sodass dieser Fund für richtig befunden wird.

Sicherlich ist zu bedenken, dass SIFT als Basisalgorithmus für diese Art von Problemstellungen noch viel mehr Potential besitzt, als das, was wir im Rahmen dieser Veranstaltung umsetzen konnten. Für uns waren die größten Hindernisse zum einen das Finden der Keypoints, die die erste wichtige Stufe im Algorithmus darstellen. Wenn man in den relevanten Bereichen des Bildes (also dort wo Schilder sind) bereits wenig Keypoints hat, wird es natürlich umso schwieriger dort am Ende auch Matches mit dem Template zu bekommen. Aus unserer subjektiven Sicht hat allerdings das Verändern der Parameter, welche sowieso schon relativ uneinsichtig und schwer zu beurteilen sind, kaum signifikante Verbesserung bzw. überhaupt Veränderung gebracht. Die Anzahl der Keypoints geht dabei auch einher mit der Performanz, je mehr relevante Punkte untersucht werden, umso langsamer wird der Algorithmus, welcher ohnehin nicht besonders performance-optimiert ist.
Weitere Probleme waren die Qualität und Menge der Matches, sowie auch die Bewertung der Relevanz dieser Matches.

Schluss

Abschließend können wir sagen, dass der SIFT-Algorithmus durchaus geeignet ist für die Art Problemstellung, wie wir sie hatten. Aber gerade aufgrund seiner Komplexität ist es zum einen nicht einfach mit ihm umzugehen und zum anderen erfordert es gerade im Bereich des *Matchings* eine gute Implementation, die auch die aufgezeigten Sonderfälle berücksichtigt. Gerade hier ist noch einiges an Verbesserungspotential vorhanden, was uns im Rahmen dieses Projektes leider nicht möglich war.

Quellenangabe