Embedded Stereokamera-basierte reaktive Kollisionsvermeidung für UAVs

Embedded stereo vision based reactive collision avoidance for UAVs

Michael Grinberg[1], Boitumelo Ruf[1,2], Sebastian Monka[1], Matthias Kollman[1], Aleksej Buller[1] und Igor Tchouchenkov[1]

[1] Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung IOSB, Fraunhoferstr. 1, Karlsruhe

[2] Institut für Photogrammetrie und Fernerkundung, KIT, Englerstr. 7, Karlsruhe

Zusammenfassung Dieser Beitrag beschäftigt sich mit der automatischen Kollisionsvermeidung für UAVs basierend auf einer Stereokamera. Dazu haben wir ein echtzeitfähiges und energieeffizientes System realisiert, das mit beschränkten Ressourcen zurechtkommt. Zur Hinderniserkennung werden aus den Bildern der Stereokamera Disparitätskarten berechnet. Diese werden in die sog. U- bzw. V-Maps konvertiert, mit deren Hilfe mit Methoden der Mustererkennung Objekte erkannt werden. Die Kollisionsvermeidung basiert auf einem reaktiven Ansatz, welcher den kürzesten Weg zum Umfliegen eines sich in kritischen Entfernung befindlichen Hindernisses sucht.

Wir haben unser System auf einer Xilinx Zynq Ultrascale+ Prozessoreinheit mit einer ARM Cortex-A53 Quadcore-CPU und einem FinFET+ FPGA umgesetzt. Die Disparitätsschätzung wurde auf dem FPGA implementiert, während die weniger rechenintensive Kollisionsvermeidung zur Ausführung auf der CPU umgesetzt wurde. Die Portierung der in C/C++ geschriebenen Algorithmen zur Disparitätsschätzung auf den FPGA erfolgte mittels High-Level-Synthese, welche die Lücke zwischen Anwendungsentwicklung und Hardware-Optimierung schließt und damit eine Reduktion der Entwicklungszeit und -Kosten ermöglicht.

Schlagwörter UAV-Navigation, Hindernisdetektion, Kollisionsvermeidung, embedded, Echtzeit-Stereoverarbeitung.

Abstract This contribution deals with obstacle avoidance for UAVs based on a stereo camera setup. We implemented a real-time-capable and energy-efficient system based on disparity map estimation for obstacle detection and a reactive approach for collision avoidance. The basic functionality has been written in C/C++ and optimized for Xilinx Zynq Ultrascale+ with an ARM Cortex-A53 Quad-Core CPU and a FinFET+ FPGA. We aimed to use High-Level Synthesis for porting parts of out code to FPGA in order to close the gap between application development and hardware optimization and reduce development time and costs. We evaluated our implementation of the disparity estimation on the KITTI Stereo 2015 benchmark. The integrity of the overall real-time collision avoidance system has been evaluated by using Hardware-in-the-Loop testing in conjunction with two flight simulators.

Keywords UAV navigation, obstacle detection, collision avoidance, embedded, real-time stereo processing.

1 Einleitung

Um die Benutzerfreundlichkeit und Sicherheit von unbemannten Luftfahrzeugen (UAVs) zu verbessern, können UAVs mit Sensoren, wie z.B. Ultraschallsensoren, Laserscannern und Kameras, ausgestattet werden. Eine wichtige Aufgabe dabei ist die Detektion von Hindernissen und die Vermeidung von Kollisionen. Da ein Großteil der Verbraucher-UAVs unter engen Gewichts- und Leistungseinschränkungen leidet, konzentriert sich unsere Arbeit auf die Hindernisdetektion und Kollisionsvermeidung auf der Grundlage eines leichten Stereokamera-Setups. Dabei sind die Kameras in Flugrichtung ausgerichtet. Die aus den Kamerabildern berechnete Disparitätskarten verwenden wir um Hindernisse im Flugpfad des UAVs zu detektieren und Ausweichmanöver einzuleiten. Dabei erfolgt die gesamte Verarbeitung, welche die Berechnung der Disparitätskarten und Planung des Ausweichmanövers einschließt, an Bord und in Echtzeit.

Die grundsätzliche Eignung von Stereo-Bildverarbeitung für die On-Board Kollisionsvermeidung für UAVs wurde durch zahlreiche Studien gezeigt [1,2]. Für die Berechnung von Disparitäts- bzw. Tiefenkarten auf embedded Systemen hat sich das SGM-Verfahren [3] etabliert. Gründe

dafür liegen nicht nur in dessen Leistungsfähigkeit, sondern insbesondere auch darin, dass sich dieses Verfahren gut für die Parallelisierung auf verschiedenen Rechnerarchitekturen eignet. Zudem zeichnet sich das SGM-Verfahren durch ein gutes Trade-Off zwischen Genauigkeit und Rechenzeit aus. Durch eine Optimierung des Verfahrens für eine CPU wurden für die VGA-Bildauflösung jeweils Verarbeitungsraten von 12 fps [4] und 14 fps [5] erreicht. Durch die Verwendung von General Purpose Computation on GPUs (GPGPU) können deutlich größere Verarbeitungsraten bei höheren Bildauflösungen erreicht werden [6,7].

Im Hinblick auf ihre Energieeffizienz sind FPGAs für ein embedded System jedoch deutlich besser geeignet. So wurde in [1] ein Stereo-Video-Sensor vorgestellt, bei dem eine FPGA-basierte SGM-Implementierung für die Bildauflösung von 320×240 Pixel eine Framerate von 120 fps lieferte. In [8] wurde eine Portierung des SGM-Verfahrens auf ein Xilinx Zynq 7000 FPGA beschrieben. Dabei wurde eine Framerate von 32 fps für eine Bildauflösung von 1280×720 Pixel erreicht. Wie in [9] gezeigt, können je nach FPGA für diese Bildauflösung gar Frameraten von bis zu 197 fps erreicht werden. Um solch hohe Frameraten erzielen zu können, ist allerdings eine ausgeklügelte Optimierung notwendig, die typischerweise Verwendung von Hardware-spezifischen Schnittstellen wie VHDL oder Verilog erfordert.

Eine solche Optimierung erfordert spezielles Wissen sowohl aus dem Bereich der Bildverarbeitung als auch aus dem Bereich der Hardware-Optimierung. Zudem ist sie sehr kostspielig und verlängert die Time-To-Market von Anwendungen erheblich. Daher haben wir uns im Rahmen des TULIPP-Projektes[3] [10] darauf beschränkt, die Bildverarbeitungsverfahren, die in C/C++ entwickelt wurden, alleine mittels High-Level-Synthese (HLS) auf ein FPGA zu portieren. Dabei muss naturgemäß mit einigen Leistungseinbußen gerechnet werden. Wie jedoch in diesem Beitrag gezeigt, reicht die erzielte Genauigkeit und Leistung für den anvisierten Anwendungsfall aus.

Der Rest dieses Artikels gliedert sich wie folgt: Kapitel 2 enthält eine detaillierte Übersicht über den verfolgten Ansatz und dessen Implementierung. Experimentelle Ergebnisse werden im Kapitel 3 vorgestellt. Kapitel 6 gibt eine kurze Zusammenfassung und einen Ausblick.

[3] http://tulipp.eu/

2 Systemrealisierung

2.1 Echtzeit-Disparitätsschätzung

Die Disparitätsschätzung besteht aus den folgenden Schritten:

- Rektifizierung
- Pixel-Matching
- SGM-Optimierung (Regularisierung)
- Links-Rechts-Konsistenzcheck
- Median-Filter

Diese Schritte werden in einer Pipeline angeordnet um eine effiziente Verarbeitung der Pixelströme der Kameras zu ermöglichen. Für ein speichereffizientes Verarbeiten der Daten verwenden wir FIFO-Puffer zwischen dein einzelnen Pipelinestufen. Wie bereits erwähnt, wurden alle Verfahren in C/C++ implementiert, woraus anschließend mittels HLS entsprechender VHDL-Code für die Ausführung auf dem FPGA erzeugt wurde.

Rektifizierung

Für die Disparitätsschätzung müssen die Bildpaare der Stereokamera zunächst rektifiziert werden. Wir verwenden die Standard-Kalibriermethode nach [11] um die Rektifizierungs-Lookup-Tabellen für die beiden Bilder zu berechnen. Diese enthalten für jeden Pixel im rektifizierten Bild die Koordinaten des Pixels im Originalbild. Für die Reduktion der Zugriffszeit haben wir zur Berechnung des rektifizierten Bildes einen Zeilenpuffer für die Zwischenspeicherung der benötigten Zeilen des Eingang-Pixelstroms realisiert.

Pixel-Matching

Im nächsten Schritt wird für jeden Pixel in einem der Bilder ein Ähnlichkeitsmaß mit den in Frage kommenden Pixeln des anderen Bildes berechnet. Dadurch entsteht für ein Bild der Größe $W \times H$ ein dreidimensionales Kostenvolumen $\mathcal{C} \in \mathbb{N}_0^{W \times H \times d_{\max}}$. Dabei gibt $d_{\max}$ die maximal mögliche Disparität an. Für jeden Pixel p und jede Disparität d enthält dieses Kostenvolumen das entsprechende Ähnlichkeitsmaß.

In der Literatur wurden mehrere Kostenfunktionen zur Ermittlung der Ähnlichkeitsmaßes zweier Bildpixel vorgeschlagen. Wir haben eine einfache Summe der pixelweise absoluten Differenzen sowie die nicht-parametrische Hamming-Distanz der Census Transformation [12] ausgewählt, da diese in vielen Anwendungen recht beliebt sind. Auch hier werden Zeilenpuffer verwendet, um auf die Pixeldaten aus der unmittelbaren Umgebung des betrachteten Pixels schnell zugreifen zu können.

SGM-Optimierung (Regularisierung)

Das vorberechnete Kostenvolumen $\mathcal{C}$ enthält für jeden Pixel p des linken Bildes und für jede mögliche Disparität entsprechende Plausibilitätswerte. Dies könnte bereits zur Erstellung einer Disparitätskarte genutzt werden, indem man den Winner-Takes-it-All-Ansatz verwendet, bei dem für jeden Pixel die Disparität mit den geringsten Kosten ausgewählt wird. Allerdings liefert dieser Ansatz aufgrund von Mehrdeutigkeiten in der Praxis keine zufriedenstellenden Ergebnisse. Daher wird zur Regularisierung des Kostenvolumens eine Optimierungsstrategie angewandt, die die meisten Mehrdeutigkeiten eliminieren soll. Im Falle des SGM-Verfahrens werden für jeden Pixel die Kosten benachbarten Pixel entlang mehrerer konzentrischen Pfade aggregiert (s. Abbildung 1). Dabei werden die Kosten für die benachbarten Pixel q eines Pixels p entlang eines Pfades jeweils mit dem kleinen Strafterm P_1 bestraft, wenn die Differenz der Disparitäten beider Pixel p und q ± 1 beträgt, und mit einem größeren Strafterm P_2, wenn der Betrag der Differenz größer 1 ist. Aus dem aggregierten Kostenvolumen $\mathcal{C}_{\mathrm{aggr}}$ können dann die endgültigen Disparitäten durch die Anwendung des Winner-Takes-it-All-Ansatzes extrahiert werden.

Die Aggregation entlang jedes Pfades stellt eine unabhängige Aufgabe dar, was eine starke Parallelisierung des Verfahrens für diverse Rechnerarchitekturen erlaubt. Wir haben den SGM-Algorithmus für eine Echtzeit-Anwendung auf dem FPGA optimiert. Dabei wurde die in TULIPP entwickelte Hardware-Instanziierung basierend auf einem embedded Xilinx Zynq Ultrascale+ MPSoC zugrunde gelegt.

In der ursprünglichen SGM-Formulierung [3] wird die Aggregation entlang von 16 konzentrischen Pfaden durchgeführt. Um eine Berechnung der Zwischenwerte für subpixelgenaue Positionen zu vermeiden,

verwenden die meisten Implementierungen jedoch lediglich 8 statt 16 Pfade. Aufgrund der Speicherbeschränkungen von embedded FPGAs, verbunden mit deren Stärken bei gleichzeitiger Verarbeitung mehrerer Datenströme, ist es üblich, bei Portierung des Algorithmus auf FPGAs lediglich 4 Pfade zu verwenden. Wie in [13] gezeigt, führt eine Reduktion von 8 auf 4 Pfade zu einem relativ geringen Genauigkeitsverlust von 1.7% bei gleichzeitiger Reduktion der Laufzeit.

Die Aggregation kann effizient durch eine Zwischenpufferung und Akkumulation der vorberechneten Kosten für jeden Pixel durchgeführt werden. Wie in [3] vorgeschlagen, werden nach Prozessierung jedes Pixels bei der Kostenaggregation die bisherigen Mindestkosten des Pfades subtrahiert um einen numerischen Überlauf zu vermeiden.

(a) (b)

Abbildung 1: Kostenaggregation (4 vs. 8 Pfade). Während die Aggregation entlang von 8 Pfaden (a) den Zugriff auf das komplette Bild für die Kostenberechnung einzelner Pixel erfordert, ist bei der Verwendung von 4 Pfaden (b) eine effiziente Implementierung durch Akkumulation der bereits berechneten Kosten des Pixelstroms (hellblau) möglich.

Links-Rechts-Konsistenzcheck

Zur Eliminierung von Ausreißern wird ein Links-Rechts-Konsistenzcheck durchgeführt. Dazu muss neben der Disparitätskarte für das linke Bild eine Disparitätskarte für das rechte Bild berechnet werden. Bei korrekter Schätzung sollen die für die Pixeln im linken Bild geschätzten Disparitäten mit den für das rechte Bild geschätzten Disparitäten an den jeweils entsprechend versetzten Pixelpositionen übereinstimmen. Ist dies nicht der Fall, wird die geschätzte Disparität als ungültig markiert.

Median-Filter

Um weitere Ausreißer zu eliminieren, wird abschließend eine Median-Filterung durchgeführt. Ähnlich zur Berechnung der Kostenfunktion, sind für die FPGA-Implementierung eines $k \times k$-Median-Filters k Zeilenpuffern notwendig.

2.2 Hindernisdetektion und Kollisionsvermeidung

Unser Ansatz zur UAV-Navigation basiert auf einer reaktiven Kollisionsvermeidung. Dabei werden die wie beschrieben errechneten Disparitätskarten verwendet, um die Umgebung vor dem UAV zu erfassen. Diese werden zunächst jeweils in ein horizontales und vertikales Disparitätshistogramm konvertiert (U- bzw. V-Map), wodurch die Komplexität deutlich verringert und die Robustheit gegenüber Messungenauigkeiten verbessert wird [2, 14, 15]. Bei einer U-Map wird jede Spalte der Disparitätskarte durch das zweidimensionale Histogramm der Größe $W \times d_{\max}$ repräsentiert. Entsprechend wird bei einer V-Map jede Zeile durch ein Histogramm der Größe $d_{\max} \times H$ repräsentiert. Mittels einer anschließenden Dilatation, sowie einer Gauß-Filterung, werden Messungenauigkeiten und Rauschen unterdrückt. Im nächsten Schritt werden beide Maps mittels Schwellwertbildung in ein Binärbild umgewandelt. Das Ergebnis der Berechnung sowie seine Interpretation sieht man in der Abbildung 2. Die V-Map wird dazu verwendet die Lage der Bodenebene und die Höhe der Objekte in einer bestimmten Entfernung zu schätzen. Die U-Map kann als eine Draufsicht auf die Szene interpretiert werden. Dabei können größere Objekte in der Umgebung anhand dominanteren Strukturen in der U-Map ermittelt werden.

Zur Navigation werden potentielle Hindernisse gemäß dem Ansatz in [16] anhand ihrer Konturen in der U-Map erkannt. Anschließend werden deren Schwerpunkte berechnet und sie in Ellipsen eingeschlossen. In der V-Map kann für die gegebene Disparität die vertikale Ausdehnung der Objekte berechnet werden. Dadurch werden Objekte durch ein zylindrisches Modell repräsentiert. Wird ein Objekt in einem kritischen Abstand zum UAV entdeckt, wird der kürzeste Weg nach links, rechts, oben oder unten gefunden, sodass das Hindernis umflogen werden kann. Durch eine periodische Ausführung des Algorithmus können falsche Entscheidungen erkannt und revidiert werden.

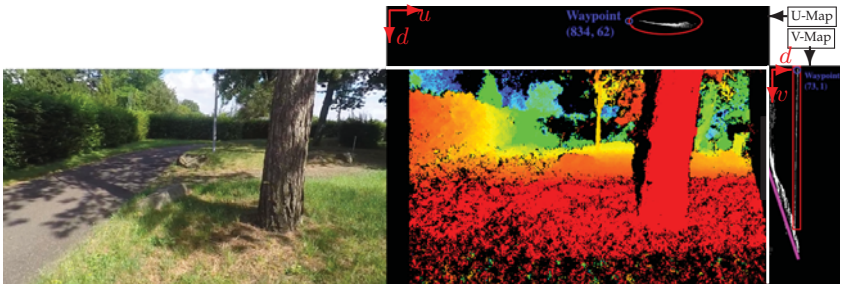


Abbildung 2: U-/V-Map-basierte Hinderniserkennung. Das linke Bild ist das Referenzbild der linken Kamera. Das rechte Bild zeigt die farbig kodierte Disparitätskarte. Dabei steht Blau für eine geringe Disparität (große Bildtiefe bzw. Entfernung zur Kamera) und Rot für eine große Disparität (kleine Bildtiefe bzw. Entfernung). Über der Disparitätskarte, sowie rechts daneben, befinden sich jeweils die U- und die V-Map. Die weiße Kontur in der U-Map entspricht der sichtbaren Seite des Baums in der Draufsicht, die interne Repräsentation des Baums geschieht durch die rot gefärbte Ellipse. In der V-Map ist der Baum durch das rot gefärbte Rechteck repräsentiert. Die pinkfarbene Linie dient zur Hervorhebung der Bodenebene (diagonale weiße Kontur). Blaue Kreise zeigen den Wegpunkt, der genutzt wird um dem Hindernis auszuweichen.

3 Evaluation

Wir haben unser System zur Kollisionsvermeidung auf einem embedded Xilinx Zynq Ultrascale+ MPSoC mit einer ARM Cortex-A53 Quadcore-CPU und einem FinFET+ FPGA implementiert. Während die Disparitätskarten auf dem FPGA berechnet werden, wird der Algorithmus zur Kollisionsvermeidung auf der CPU ausgeführt. Der Ultrascale+ wird als System-on-a-Module (SOM) auf dem EMC^2-DP Trägerboard von Sundance betrieben. Für den Bildeinzug wurden zwei Industriekameras von Sentech verwendet, die über die CameraLink-Schnittstelle mit der Trägerplatine verbunden sind.

3.1 Evaluation der Disparitätsschätzung

Sowohl die Rektifizierung der Eingangsbilder als auch die Berechnung der Disparitätskarte wurde in C/C++ implementiert. Dabei haben wir unseren Algorithmus wie folgt konfiguriert: Bei Berechnung der Sum-

me der absoluten Differenzen (SAD), der Census-Transformation (CT) sowie der Median-Filterung wird eine Nachbarschaft der Größe 5×5 verwendet. Die maximale Disparität wird auf 59 Pixel gesetzt. Die SGM-Strafterme wurden auf $P_1 = 200$ und $P_2 = 800$ für die SAD-Kostenfunktion bzw. auf $P_1 = 8$ und $P_2 = 32$ bei Verwendung der Hamming-Distanz der Census-Transformation gesetzt.

Zur automatischen Generierung des VHDL-Codes aus dem C/C++-Code für die Prozessierung auf dem FPGA haben wir Xilinx Vivado High-Level Synthesys verwendet. Die damit erzielten Ergebnisse werden in den nachfolgenden Tabellen dargestellt. Tabelle 1 zeigt den Ressourcenverbrauch des Algorithmus zur Berechnung der Disparitätskarte auf dem Ultrascale+, dessen FPGA 154.000 Logikzellen hat. Tabelle 2 zeigt die Laufzeitmessungen der einzelnen Systemkomponenten und des Gesamtsystems bei einer FPGA-Taktung von 200 MHz.

Tabelle 1: Ressourcenverbrauch des Algorithmus zur Disparitätsschätzung auf dem Zynq Ultrascale+.

Ressource	Insgesamt verfügbar	Verwendet	Ausnutzung
DSP48E	360	22	6 %
BRAM_18K	432	132	31 %
FF	141120	12561	9 %
LUT	70560	27063	38 %

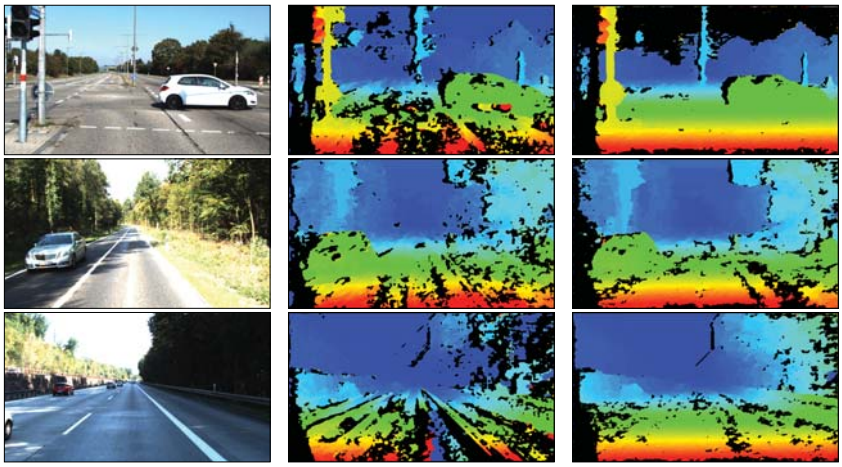
Tabelle 2: Laufzeitmessungen des Systems auf dem Xilinx Zynq Ultrascale+ bei einer FPGA-Taktung von 200 MHz.

Verfahren	Laufzeit (s)	Mittlere Laufzeit (s)
Schätzung der Disparitätskarte	0,0314 - 0,0361	0,0345
Kollisionsvermeidung	0,0027 - 0,0110	0,0042
Σ	0,0341 - 0,0471	0,0387

Die komplette Pipeline zur Disparitätsschätzung erreicht somit eine mittlere Verarbeitungsgeschwindigkeit von 29 fps bei einer Bildgröße von 640×360 Pixeln. Die Latenz der Verarbeitung liegt bei 28,5 ms. Eine Reduktion der Taktrate des FPGAs auf 100 MHz reduziert die Verarbeitungsgeschwindigkeit auf ca. 20 fps und erhöht die Latenz auf 53,2 ms.

Unser Algorithmus zur Kollisionsvermeidung läuft lediglich auf der ARM-CPU. Aufgrund der geringen Komplexität des Algorithmus liegt dessen Laufzeit weit unter der Laufzeit zur Berechnung der Disparitätskarte.

Eine quantitative Evaluation der Implementierung der Disparitätsschätzung erfolgte anhand des KITTI Stereo 2015 Benchmark-Datensatzes [17]. Für die Evaluation haben wir die Bilder auf eine 640×360 Pixel große ROI beschnitten. Tabelle 3 zeigt die Ergebnisse für die Benchmark. Unter anderem kann man daraus entnehmen, dass die Verwendung der CT ein robusteres Matching gegenüber der Verwendung der SAD Kostenfunktion erlaubt. Dies führt zu genaueren Ergebnissen und dichteren Disparitätskarten. Abbildung 3 zeigt exemplarische Ergebnisse der Disparitätsschätzung, die diese Schlussfolgerungen veranschaulichen. Eine Demonstration der Ergebnisse in Form einer Videosequenz befindet sich unter `https://youtu.be/gzIFqUmqM7g`.



(a) Originalbild (b) Disparitätskarte (Pixelmatching mit SAD) (c) Disparitätskarte (Pixelmatching mit CT)

Abbildung 3: Exemplarische Visualisierung der mit unterschiedlichen Pixel-Matching-Verfahren geschätzten Disparitätskarten für drei Stereo-Bildpaare aus dem KITTI Stereo 2015 Benchmark-Datensatzes [17].

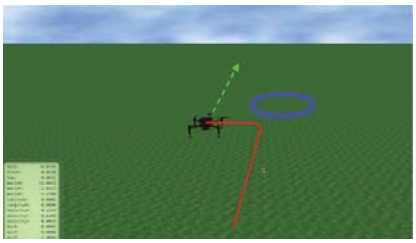
Tabelle 3: Ergebnisse der Disparitätsschätzung bei Verwendung unterschiedlicher Kostenfunktionen für die KITTI Stereo 2015 Benchmark. Die Dichte bezeichnet den Prozentsatz der Pixel, für welche eine gültige Disparität berechnet werden konnte.Die Disparitätsschätzung eines Pixels wird als korrekt betrachtet, wenn ihre Differenz zur Groundtruth weniger als 3 Pixel ist.

Kostenfunktion	Dichte	Anzahl korrekter Pixel
SAD	64,1 %	76,0 %
CT	74,2 %	95,4 %

Insgesamt zeigt sich somit, dass die erzielte Verarbeitungsgeschwindigkeit und Genauigkeit für unseren Anwendungsfall ausreichen.

3.2 Evaluation der Hindernisdetektion und Kollisionsvermeidung

Die Funktionalität der Kollisionsvermeidung wurde anhand von Hardware-in-the-Loop-Tests für zwei unterschiedliche UAV-Systeme validiert, nämlich für die DJI Matrice 100 und für ein Pixhawk-basiertes System. Dazu haben wir die UAV-Flugsimulator-Software von DJI sowie Microsoft AirSim [18] verwendet. Abbildung 4 zeigt Screenshots der beiden Flugsimulatoren.



(a) Screenshot der Simulationssoftware von DJI

(b) Screenshot der AirSim Simulationssoftware von Microsoft

Abbildung 4: Screenshots von zwei Flugsimulatoren, die für das Hardware-in-the-Loop-Testen verwendet wurden. Das Hauptfenster in (b) zeigt die Sicht eines externen Beobachters, während die drei kleine Fenster am unteren Rand die Disparitätskarte, die Objektsegmentierung und das Kamerabild des UAV beinhalten.

Beide Flugcontroller waren über die USB-Schnittstelle mit dem PC verbunden, auf dem die Simulationssoftware ausgeführt wurde. Ausgehend von den aus der Simulations-Software erhaltenen Daten, sollte unser System eine Flugplanung durchführen und die entsprechenden Steuerbefehle an die Simulations-Software zurückliefern. Durch diese Hardware-in-the-Loop-Anbindung konnte eine automatische Navigation des UAV simuliert und getestet werden. Die Tests wurden im autonomen Flugmodus durchgeführt. Dabei haben wir das folgende Verhalten des UAV zugrunde gelegt: Nach dem Start steigt das UAV in 1m Höhe und fliegt dann geradeaus. Befindet sich ein Hindernis im Flugpfad, findet das System den kürzesten Weg zu seinem Rand und bewegt sich entsprechend nach links, rechts, oben oder unten. Sobald das Hindernis aus dem Sichtfeld verschwindet, bewegt sich das UAV wieder geradeaus.

Für die Tests der grundlegenden Funktionalität der Kollisionsvermeidung haben wir den proprietären Flugsimulator von DJI verwendet. Dieser ist zwar sehr minimalistisch, enthält jedoch die für die ersten Versuche notwendige Funktionalität. Eine Demonstration der durchgeführten Simulation befindet sich unter `https://youtu.be/pMDMTUCVwKc`.

Für realistischere Tests haben wir den Pixhawk-Flugcontroller verwendet, welches durch den Microsoft AirSim-Flugsimulator unterstützt wird. Der AirSim-Simulator basiert auf der Unreal Engine, welche fotorealistische Modellierung der Umgebung bietet. Auch erlaubt AirSim Tiefen- und damit auch Disparitätskarten der simulierten Umgebung zu generieren. Dadurch ist es möglich die Leistungsfähigkeit der Hindernisausweichung zu testen und sein Verhalten in verschiedenen Szenarien zu analysieren. Zu beachten ist allerdings, dass die durch den Simulator generierten Disparitätskarten keine in realen Szenarien auftretenden Artefakte wie Rauschen und Ausreißer enthalten, so dass die in der Simulation erreichten Ergebnisse in ähnlichen Szenarien in der realen Umgebung überprüft werden müssen.

4 Fazit & Ausblick

In diesem Beitrag haben wir uns mit der Entwicklung eines Systems zur Detektion von Hindernissen und Kollisionsvermeidung für UAVs

beschäftigt. Dazu haben wir eine Optimierung des bekannten SGM-Verfahrens für die Ausführung auf embedded Hardware durchgeführt und einen reaktiven Ansatz zur Kollisionsvermeidung umgesetzt. Unsere Experimente zeigen, dass mit unserem System eine echtzeitfähige Detektion von Hindernissen und eine Kollisionsvermeidung für UAVs, welche starken Gewichts- und Leistungseinschränkungen unterliegen, möglich ist.

Unsere aktuellen Arbeiten konzentrieren sich auf die Durchführung der Experimente in einer realen Umgebung. Parallel dazu sind wir dabei, eine Evaluation unserer Implementierung der Disparitätsschätzung in Bezug auf ihre Energieeffizienz auf unterschiedlichen Plattformen, nämlich FPGA, CPU und GPU, durchzuführen.

Literatur

1. A. J. Barry, H. Oleynikova, D. Honegger, M. Pollefeys, and R. Tedrake, "FPGA vs. Pushbroom Stereo Vision for MAVs," in *IROS Workshop on Vision-based Control and Navigation of Small Lightweight UAVs*, 2015.
2. H. Oleynikova, D. Honegger, and M. Pollefeys, "Reactive Avoidance Using Embedded Stereo Vision for MAV Flight," in *Proc. IEEE International Conference on Robotics and Automation*, 2015, pp. 50–56.
3. H. Hirschmueller, "Stereo Processing by Semi-Global Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.
4. R. Spangenberg, T. Langner, S. Adfeldt, and R. Rojas, "Large Scale Semi-Global Matching on the CPU," in *Proc. IEEE Intelligent Vehicles Symposium*, 2014, pp. 195–201.
5. S. K. Gehrig and C. Rabe, "Real-Time Semi-Global Matching on the CPU," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2010, pp. 85–92.
6. C. Banz, H. Blume, and P. Pirsch, "Real-Time Semi-Global Matching Disparity Estimation on the GPU," in *Proc. IEEE International Conference on Computer Vision Workshops*, 2011, pp. 514–521.
7. I. Haller and S. Nedevschi, "GPU Optimization of the SGM Stereo Algorithm," in *Proc. IEEE International Conference on Intelligent Computer Communication and Processing*, 2010, pp. 197–202.

8. J. Hofmann, J. Korinth, and A. Koch, "A Scalable High-Performance Hardware Architecture for Real-Time Stereo Vision by Semi-Global Matching," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2016, pp. 27–35.
9. Y. Li, Z. Li, C. Yang, W. Zhong, and S. Chen, "High Throughput Hardware Architecture for Accurate Semi-Global Matching," *Integration*, 2017.
10. T. Kalb, L. Kalms, D. Göhringer, C. Pons, F. Marty, A. Muddukrishna, M. Jahre, P. G. Kjeldsberg, B. Ruf, T. Schuchert, I. Tchouchenkov, C. Ehrenstrahle, F. Christensen, A. Paolillo, C. Lemer, G. Bernard, F. Duhem, and P. Millet, "TULIPP: Towards Ubiquitous Low-power Image Processing Platforms," in *Proc. International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, July 2016, pp. 306–311.
11. Z. Zhang, "A Flexible New Technique for Camera Calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, 2000.
12. R. Zabih and J. Woodfill, "Non-parametric Local Transforms for Computing Visual Correspondence," in *Proc. European Conference on Computer Vision*, 1994, pp. 151–158.
13. C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch, "Real-Time Stereo Vision System using Semi-Global Matching Disparity Estimation: Architecture and FPGA-Implementation," in *Proc. International Conference on Embedded Computer Systems*, 2010, pp. 93–101.
14. R. Labayrade, D. Aubert, and J. P. Tarel, "Real Time Obstacle Detection in Stereovision on Non Flat Road Geometry Through „V-disparity" Representation," in *Proc. IEEE Intelligent Vehicle Symposium*, vol. 2, Jun. 2002, pp. 646–651 vol.2.
15. Y. Li and Y. Ruichek, "Occupancy Grid Mapping in Urban Environments from a Moving On-Board Stereo-Vision System," *Sensors*, vol. 14, no. 6, pp. 10 454–10 478, 2014.
16. S. Suzuki and K. Abe, "New Fusion Operations for Digitized Binary Images and Their Applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, no. 6, pp. 638–651, Nov. 1985.
17. M. Menze and A. Geiger, "Object Scene Flow for Autonomous Vehicles," in *Proc. Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3061–3070.
18. S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles," in *Proc. Field and service robotics*, 2018, pp. 621–635.