Dynamische 2D- und 3D-Trajektorienplanung für UAVs

Alfred Schöttl, LFK GmbH, MBDA Systems Deutschland[1]

Wir betrachten das Problem der Trajektorienplanung von unbemannten Luftfahrzeugen (UAVs). Hierfür werden bei statischen Umgebungsbedingungen häufig A*-Algorithmen eingesetzt. Es ist bekannt, dass diese Algorithmenklasse im Allgemeinen brauchbare Näherungen der Optimallösung liefert.

Leider ist die Neuplanung bei Änderungen der Umweltbedingungen sehr zeitaufwändig und ressourcenfordernd. Außerdem kann, da A*-Algorithmen in der Regel auf Gitterpunkten basierende Trajektorien liefern, je nach Anwendungsfall die Güte der Lösung deutlich degradiert sein. Kürzlich wurde die Klasse der Field D*-Algorithmen eingeführt, die diese Probleme umgehen. Im Allgemeinen liefern sie Pfade besserer Güte und bieten einen effizienten Weg zum dynamischen Update der Lösung bei sich ändernden Umweltbedingungen.

Aktuelle Implementierungen betrachten in der Regel den 2-D-Fall und unterstützen nicht typische UAV-Planungsnebenbedingungen wie Bedrohungen, verbotene Gebiete oder die Optimierung bezüglich der Missionserfolgswahrscheinlichkeit. Wir zeigen, dass Field D*-Algorithmen geeignet sind, solche Probleme selbst im 3-D-Fall mit vertretbarem Aufwand zu lösen.

1. Einleitung

Wir betrachten die Planung der Trajektorie eines unbemannten Fluggeräts (UAV). Dabei sei S die Menge aller möglichen Positionen ($S \subseteq \mathbb{R}^d$, $d=2$ oder $d=3$). Mit $s_{start} \in S$ und $s_{goal} \in S$ bezeichnen wir die gegebene *gewünschte Startposition* bzw. *Endposition* der Trajektorie. Die Trajektorien müssen dabei *Nebenbedingungen* π erfüllen (wie z. B. das Umfliegen von verbotenen Gebieten). Eine Trajektorie t, die die Nebenbedingungen $\pi(t)$ erfüllt, wird *zulässig* genannt. Die Menge aller zulässigen Trajektorien von s_{start} nach s_{goal} wird mit T bezeichnet. Ohne Beschränkung der Allgemeinheit sei hier angenommen, dass die Trajektorien t durch eine Folge (s_i) von Stützstellen repräsentiert werden. Es sei weiter $q: T \to \mathbb{R}_+ \cup \{\infty\}$ eine Kostenfunktion, geringere Werte entsprechen dabei günstigeren Trajektorien.

Wir nehmen weiter an, dass die Kostenfunktion *additiv* bzgl. einer *Transitionskostenfunktion* $v: S^2 \to \mathbb{R}_+ \cup \{\infty\}$ ist,

$$q(t) = \sum v(s_i, s_{i+1}).$$

Die klassischen Algorithmen der dynamischen Optimierung liefern exakte Lösungen, sind jedoch aus Rechenzeitgründen in der Regel nicht anwendbar. Man ist deshalb üblicherweise an suboptimalen Lösungen bezüglich q interessiert, (d. h. an einer Trajektorie $t \in T$ mit „fast“ optimalen Kosten $q(t)$).

2. Der A*-Algorithmus im Kontext der UAV-Trajektorienplanung

Der bei der Trajektorienplanung am häufigsten eingesetzte Algorithmus ist der *A*-Algorithmus* [8]. Er basiert auf einer Repräsentation der Umgebung durch Graphen und ist daher besonders geeignet im Kontext von Straßenkarten oder ähnlichen, durch Graphen gut repräsentierbare Strukturen.

Da UAVs in der Regel in offenen Umgebungen ohne enge Begrenzungen fliegen, wird bei UAV-Anwendungen häufig eine adaptierte Version des A*-Algorithmus verwendet: Der Raum $\mathbb{R}^d$ wird hierzu mit Hilfe eines äquidistanten Gitters S diskretisiert. Jeder Gitterpunkt entspricht dabei einem Knoten des Graphen, alle benachbarten Knoten werden durch Kanten verbunden (s. Bild. 1).

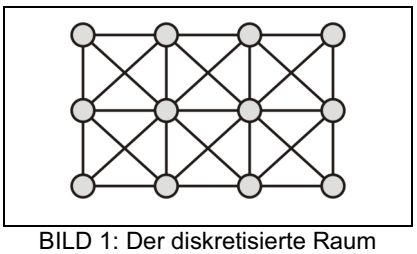

BILD 1: Der diskretisierte Raum

Es seien nun $s, s', s'' \in S$ beliebige Knoten. Mit $f^*(s, s'')$ werden die optimalen Kosten einer Trajektorie von s nach s'' bezeichnet und mit $f^*(s, s', s'')$ die Kosten einer Trajektorie von s über s' nach s'' (falls keine Trajektorie existiert, seien die Kosten ∞). Zur Vereinfachung der Notation setze man $f^*(s') := f^*(s_{start}, s', s_{goal})$.

Das Bellmansche Prinzip der dynamischen Optimierung besagt, dass additive Kostenfunktionen eine

[1] Diese Arbeit nutzt Resultate, die im Rahmen einer Kooperation mit dem ITE Institut für Systemoptimierung, Universität Karlsruhe, Prof. Gert Trommer and Oliver Meister erzielt wurden.

Zerlegung der optimalen Kosten in vergangene und zukünftige erlauben. Für jeden Punkt $s \in S$ gilt somit

$$f^*(s) = f^*(s_{start}, s) + f^*(s, s_{goal}).$$

Die Hauptidee des A*-Algorithmus ist es, eine Approximation f von f^* für die Steuerung der Suchreihenfolge der Knoten zu nutzen. Sei dazu $h(s,s')$, $h: S^2 \to \mathbb{R}_+$, eine grobe Kostenfunktion, die sogenannte *Heuristik*. Es ist für eine sinnvolle Approximation offensichtlich, dass die Heuristik die Dreiecksungleichung erfüllen muss,

$$h(s,s'') \le h(s,s') + h(s',s''). \qquad (1)$$

Wir fordern zusätzlich ein optimistisches Verhalten,

$$h(s,s') \le f^*(s,s').$$

Die A*-Approximation wird dadurch realisiert, dass ein Summand in Gleichung (1) durch die Heuristik[2] ersetzt wird,

$$f(s) = h(s_{start}, s) + f^*(s, s_{goal}).$$

Jedem Knoten $s \in S$ wird einer der Zustände *unbekannt, offen, bearbeitet* zugeordnet. Ein *unbekannter* Knoten wurde durch den Algorithmus noch nicht gefunden. Ein *offener* Knoten wurde bereits besucht. Ein approximierter, d. h. möglicherweise zu großer, Wert $f(s)$ seiner Güte ist somit bekannt. Der optimale Wert $f^*(s, s_{goal})$ eines *bearbeiteten* Knotens ist dagegen bereits bestimmt. Der Algorithmus implementiert die nach f priorisierte Suche aller offenen Knoten, dabei werden die erfolgversprechendsten Knoten (d. h. diejenigen mit dem kleinsten f-Wert) zuerst bearbeitet. Der Algorithmus startet mit dem als *offen* attributierten Zielknoten, alle anderen Knoten werden auf *unbekannt* gesetzt (s. etwa [8]).

Für eine effiziente Implementierung wird die Menge der offenen Knoten zusammen mit ihren f-Werten in einer geeigneten Datenstruktur, üblicherweise einer Prioritätsschlange (mit Prioritätsschlüssel $f(s)$ als Priorität des Knotens s) gespeichert.

Es kann gezeigt werden [7], dass der A*-Algorithmus vollständig, optimal und optimal effizient ist (in der Klasse aller Algorithmen, die dieselbe Heuristik benutzen). Im ungünstigsten Fall ist die Laufzeit von quadratischer Ordnung der Knotenzahl. Ein wesentlicher Faktor bei der Implementierung ist in der Regel der Speicherplatz, in pathologischen Fällen müssen alle Knoten S im Speicher gehalten werden.

[2] In unserem Kontext erweist es sich als günstig, die Suche vom Ziel s_{goal} aus zu starten. Es ist daher sinnvoll, die Kosten $f^*(s_{start}, s)$ in Richtung des Startpunkts durch die Heuristik abzuschätzen. In der Standardformulierung sucht der A*-Algorithmus in umgekehrter Richtung von s_{start} nach s_{goal}.

3. Der D*-Lite-Algorithmus zur Trajektorienplanung eines UAVs

Der D*-Lite-Algorithmus [10] ist vom A*-Algorithmus abgeleitet, speichert jedoch für jeden offenen Knoten s eine Kostenschätzung von s zum Ziel, $g(s) = f(s, s_{goal})$ zusammen mit einer Einschritt-Propagation der Kostenschätzung $rhs(s)$,

$$rhs(s) = \begin{cases} 0 & : s = s_{goal} \\ \min_{s' \in N(s)}(g(s') + v(s,s')) & : \text{else} \end{cases}$$

Hierbei bezeichnet $N(s)$ die Menge der adjazenten Knoten von s.

Man beachte, dass bei exakter Schätzung g, $g(s) = f^*(s, s_{goal})$ für alle $s \in S$, die Einschritt-Propagation identisch wäre, $g = rhs$. Umgekehrt ist die Schätzung exakt, falls die beiden Funktionen g und rhs übereinstimmen. Aus diesem Grund werden Knoten s, die die Bedingung $g(s) = rhs(s)$ erfüllen, *konsistent* genannt. Alle anderen Knoten heißen *überkonsistent* genau dann, wenn $g(s) > rhs(s)$ und sonst *unterkonsistent*. Inkonsistente Knoten werden wie beim A*-Algorithmus in die Prioritätswarteschlange aufgenommen. Statt des A*- Prioritätsschlüssels

$$f(s) = h(s_{start}, s) + f^*(s, s_{goal})$$

wird nun der zweidimensionale Schlüssel (k_1, k_2),

$$k_1(s) = h(s_{start}, s) + \min(g(s), rhs(s)),$$
$$k_2(s) = \min(g(s), rhs(s))$$

verwendet, versehen mit der lexikographischen Ordnung: Ein inkonsistenter Knoten s besitzt höhere Priorität als ein inkonsistenter Knoten s', falls $k_1(s) < k_1(s')$. Wenn k_1 für beide Knoten denselben Wert liefert, wird das zweite Schlüsselelement zum Vergleich verwendet. Wenn auch dieses übereinstimmt, wird eine beliebige Reihenfolge für den Eintrag in die Prioritätsschlange zugelassen.

Der Updateprozess unterscheidet zwischen *überkonsistenten* und *unterkonsistenten* Knoten. Überkonsistente Knoten werden einfach durch die Zuweisung $g(s) = rhs(s)$ zu konsistenten Knoten. Im unterkonsistenten Fall wird hingegen $g(s) = \infty$ gesetzt. Danach sind die rhs-Werte der Nachbarknoten von s neu zu berechnen und evtl. hierdurch nicht mehr konsistente Knoten in die Prioritätsschlange einzufügen. Gilt in beiden Fällen $g(s) = rhs(s)$, wird der Knoten aus der Prioritätsschlange entfernt.

Der Updateprozess startet mit dem einzigen Knoten s_{goal} in der Prioritätsschlange mit $g(s_{goal}) = \infty$. Er wird so lange wiederholt, bis s_{start} konsistent ist.

Man kann zeigen [9], dass der D*-Lite-Algorithmus die vorteilhaften Eigenschaften des A*-Algorithmus behält: Er ist vollständig, optimal und optimal effizient unter allen Algorithmen, die dieselbe Heuristik verwenden.

Ein Vorteil des D*-Lite-Algorithmus ist, dass die Pfadneuberechnung bei einer Änderung der Umweltbedingungen mit weniger Aufwand erfolgen kann als beim A*-Algorithmus. Für typische UAV-Anwendungen, in denen zur Erfassung der Umwelt bordeigene Sensorik mit verwendet wird und sich das Abbild der Umgebung während der Mission häufig verändert, ist somit der D*-Lit-Algorithmus besonders geeignet.

Neuerkenntnisse zur Umwelt spiegeln sich in einer Änderung der Kantenkosten v wider. Die Neuberechnung der Trajektorie erfolgt durch Anpassung der rhs-Werte der zugehörigen Knoten s, was gegebenenfalls zum Update der Prioritätsschlange führt.

4. Planung in einer kontinuierlichen Umgebung

In den meisten UAV-Anwendungen wird davon ausgegangen, dass das UAV vom Start- zum Zielpunkt durch einen kontinuierlichen Raum $C \subseteq \mathbb{R}^d$ fliegt. Die Trajektorie $t:[0,l] \to C$ ist damit ein kontinuierlicher Pfad, parametriert durch seine Länge l. Seine Kosten sind das Linienintegral

$$q(t) = \int_0^l c(t_s)\, ds ,$$

mit den durch die Funktion c definierten *spezifischen Pfadkosten*.

Die Trajektorienplanung mit Hilfe von Graph-basierten Algorithmen wie dem D*-Lite-Algorithmus wird durch Rasterung des kontinuierlichen Raums C erreicht. Die Gitterpunkte des üblicherweise equidistanten Rasters werden dabei als die Knotenmenge S des Graphen definiert, die Kanten ergeben sich aus den Verbindungslinien zu den Nachbarknoten (8 in 2D and 26 in 3D). Ohne Beschränkung der Allgemeinheit nehmen wir an, dass $S \subseteq \mathbb{Z}^d \cap C$.

Im Folgenden werden die Quadrate (bzw. Würfelflächen) mit Kantenlänge 1, die durch 2^d benachbarte Knoten definiert werden, *Rasterzellen* genannt (s. Bild 2). Man beachte, dass jeder Punkt $r \in G$ auf einer Rasterzelle G als Linearkombination seiner 2^{d-1} benachbarten Knoten $(s_k) =: N(r)$ repräsentiert werden kann,

$$r = \alpha_1 s_1 + (1-\alpha_1) s_2 =: [\alpha, (s_1, s_2)]$$

für $d = 2$ und analog für $d = 3$

$$\begin{aligned} r &= \left(\alpha_1 s_1 + (1-\alpha_1) s_2\right) \alpha_2 \\ &\quad + \left(\alpha_1 s_3 + (1-\alpha_1) s_4\right)(1-\alpha_2) \\ &:= [(\alpha_1, \alpha_2), (s_1, s_2, s_3, s_4)] \end{aligned}$$

mit der Kurzschreibweise $r =: [\alpha, (s_k)]$ für Vektoren $\alpha \in [0,1]^{d-1}$ und $s_k \in S$. Innerhalb einer Rasterzelle seien die spezifischen Kosten konstant. Auf der gemeinsamen Linie zweier benachbarter Rasterzellen wird dabei das Minimum der spezifischen Kosten der beiden Zellen angenommen. Im einfachsten Fall von konstanten spezifischen Zellkosten $c \equiv 1$ ergibt sich als Trajektorienkosten q die Pfadlänge.

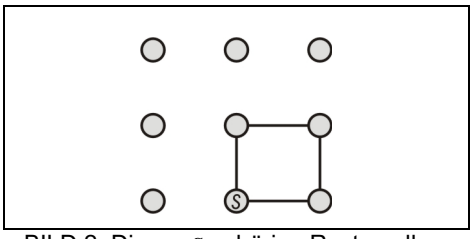

BILD 2: Die zu s gehörige Rasterzelle

Wie der A*-Algorithmus berechnet der D*-Lite-Algorithmus die optimale Lösung unter allen durch Graphen dieser Struktur repräsentierbaren Trajektorien. Die optimale Trajektorie kann, abhängig von der Kostenfunktion) weit entfernt liegen. Selbst im einfachsten Fall der Optimierung der Pfadlänge ($c \equiv 1$) kann die Lösung bis zu

$$\frac{1+\sqrt{2}}{\sqrt{5}} - 1 \approx 8\%$$

zu lang sein (s. Bild 3).

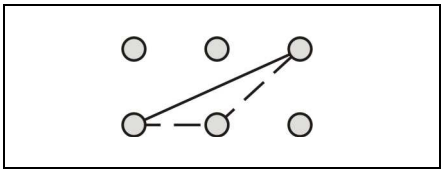
BILD 3. Worst-Case Szenario

Dieses Problem wird beim Field-D*-Algorithmus umgangen, indem man zulässt, dass die Trajektorie die Rasterzelle beliebig schneidet. Es sei $r = [\alpha, (s_k)]$ ein beliebiger Rasterzellenpunkt und seien $g(s_k)$ die optimalen Kosten einer Trajektorie, die bei Knoten $s_k \in S$ startet. Die optimalen Kosten $g(r)$ einer Trajektorie, die bei r startet, werden linear interpoliert bzgl. der optimalen Kosten der benachbarten Gitterpunkte,

$$g(r) = [\alpha, (g(s_k))] \qquad (2)$$

(s. etwa [4]). Die Transitionskosten $v(s,r)$ eines Gitterpunkts $s \in S$ zu einem beliebigen Rasterzellenpunkt r derselben Rasterzelle werden als die minimalen Pfadkosten aller innerhalb der Zelle verlaufenden Trajektorien von s nach r definiert.

Es lässt sich leicht zeigen, dass im 2D-Fall diese Trajektorie nur aus einer Strecke auf dem Raster von s zu einem Zwischenpunkt $s_x = [x, (s, s_1)]$, gefolgt von einer Diagonalen von s_x nach r bestehen kann, hier bei sei $r = [\alpha, (s_1, s_2)]$ mit $s \neq s_1$,

$s \neq s_2$ and $\alpha \in (0,1)$ (s. Bild 4). Die Trajektorie degeneriert, falls $r = [\alpha, (s, s_1)]$ oder $r = s_1$. Die Situation ist im 3D-Fall analog. [3] zeigt einen kurzen Algorithmus zur effizienten Bestimmung der optimalen Kosten $v(s,r)$.

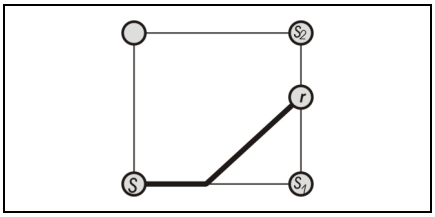

BILD 4. Die optimale Trajektorie innerhalb einer Rasterzelle

Die Idee des Field-D*-Algorithmus ist es, den D*-Lite-Algorithmus auf die Gitterpunkte S anzuwenden, die Kosten jedoch über alle möglichen Rasterzellenpunkte r einer Rasterzelle zu optimieren. Somit erhält man für rhs mit $s, r \in G$

$$rhs(s) = \begin{cases} 0 & : s = s_{goal} \\ \min_{r \in G}(g(r) + v(s,r)) & : \text{else} \end{cases}$$
$$\approx \begin{cases} 0 & : s = s_{goal} \\ \min_{\alpha \in [0,1]}(\alpha g(s_1) + \\ \quad (1-\alpha) g(s_2) + v(s,r)) & : \text{else} \end{cases}$$

für die oben beschriebenen Transitionskosten v und den interpolierten Kosten g.

Es besteht Einigkeit unter einer Reihe von Autoren ([2], [9]), dass der Field-D*-Algorithmus eine gute Approximation $g(s)$ der optimalen Kosten $f^*(s, s_{goal})$ für Trajektorien im kontinuierlichen Raum C liefert.

Die Konstruktion einer suboptimalen Trajektorie kann unter Verwendung der erhaltenen g-Werte auf unterschiedlicher Weise erfolgen. Eine häufig angewandte Methode ist es, in einem ersten Schritt die suboptimale Trajektorie auf dem Raster S zu rekonstruieren und in einem zweiten Schritt die optimalen Rasterzellenpunkte r jeder Zelle zu bestimmen. Diese Rasterzellenpunkte werden durch Geradenstücke verbunden (s. Bild 5a, 5b). Da diese Methode in wenigen, pathologischen Fällen keine brauchbaren Trajektorien liefert, sollte die konstruierte Trajektorien gegen andere Trajektorien getestet werden (z. B. gegen die D*-Lite-Trajektorie, s. [6]). Eine andere Methode nutzt lokale Optimierer (die die Dynamik des UAV berücksichtigen) zur Bestimmung von realisierbaren Trajektorien. Hierzu wird die Definitionsmenge der Funktion g durch Interpolation auf C erweitert. Durch Anwendung klassischer Optimierungsmethoden (Gradientenmethoden) wird nun unter Berücksichtigung der Nebenbedingungen ein „Abstieg“ auf der durch g definierten Fläche von s_{start} nach s_{goal} ermittelt.

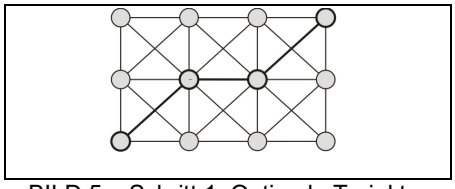

BILD 5a. Schritt 1: Optimale Trajektorie auf S (mit interpolierten Kosten)

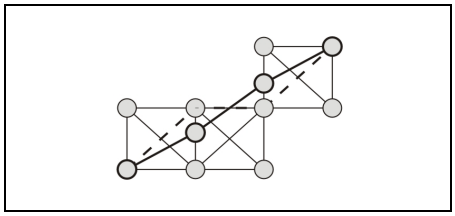

BILD 5b. Schritt 2: Verbesserung durch Interpolation der Knoten

5. UAV-spezifische Optimierungskriterien und Nebenbedingungen

UAV-Missionen haben neben klassischen Anforderungen wie der nach minimaler Pfadlänge eine Vielzahl weiterer Anforderungen zu erfüllen.

5.1 Verbotene Gebiete

Verbotene Gebiete sind eine Teilmenge $F \subseteq C$ des kontinuierlichen Raums, die die Trajektorie nicht schneiden darf, $t \cap F = \varnothing$. Diese Nebenbedingung kann auf zweifache Weise berücksichtigt werden: die zugehörige Knotenmenge kann um die Knoten in F reduziert werden, $S_{neu} = S \setminus F$, so dass keine Rasterzelle, die die verbotenen Gebiete berührt, im Optimierungsproblem enthalten ist. Diese Methode ist besonders im Fall großer verbotener Gebiete und zeitkritischer Anwendungen günstig. Sie erfordert einen zusätzlichen Verwaltungsaufwand für den Aufbau des nun weniger regulären Graphen. Eine andere Möglichkeit besteht in der Zuweisung von unendlichen spezifischen Kosten für die betroffenen Rasterzellen, $c(s) = \infty$. Diese Methode ist besonders für kleinere und geometrisch schwer zu erfassende Gebiete geeignet.

Ein offensichtlicher Spezialfall der verbotenen Gebiete ist die Berücksichtigung der Terrainhöhe im 3D-Fall.

5.2 Missionserfolgswahrscheinlichkeit

In einer großen Zahl von Anwendungen ist das entscheidende Kriterium an die UAV-Mission die Wahrscheinlichkeit, zu der die Mission erfolgreich abgeschlossen werden kann. In unserem Modell wird die Erfolgswahrscheinlichkeit durch Bedrohungen beeinträchtigt. Die Entdeck- und Zerstörwahrscheinlichkeit eines UAVs durch ein Luftverteidigungssystem hängt vom Abstand von der Bedrohung (genauer: vom Sensor) und der Streckenlänge durch den Auffassungsbereich des Systems ab.

Unter der (gewagten!) Annahme, dass das Ereignis, dass das UAV aufgefasst und zerstört wird, zu jedem Zeitpunkt von anderen Zeitpunkten stochastisch unabhängig ist, ist die Missionserfolgwahrscheinlichkeit p_{succ} gegeben durch

$$p_{succ}(t) = \exp\left(\int_0^l \ln(p_{spec}(t(x)))\, dx \right)$$

wobei die spezifische Wahrscheinlichkeit p_{spec} (Wahrscheinlichkeit pro Pfadlänge) die Wahrscheinlichkeit darstellt, an der Position x nicht vom erfolgreichen Abschluss der Mission gehindert worden zu sein. Wieder wird angenommen, dass die spezifische Wahrscheinlichkeit pro Rasterzelle konstant ist. Man beachte, dass wegen der Monotonie der Exponentialfunktion das lineare Kostenfunktional

$$\int_0^l \ln(p_{spec}(t(x)))\, dx$$

ein äquivalentes Kriterium definiert.

Falls nicht alle Bedrohungen im Voraus bekannt sind, wird im 3D-Fall üblicherweise zusätzlich verlangt, dass der Flugpfad nahe am Boden realisiert wird, um die Entdeckungsgefahr zu minimieren. Ein zusätzlicher höhenabhängiger Kostenterm, der die Wahrscheinlichkeit für die Entdeckung und Bekämpfung aufgrund zu hohen Flugs beschreibt, kann wie oben zur Gesamtkostenfunktion (s. Abschnitt 5.3) hinzugefügt werden. Er steuert die Höhe der Trajektorie über Boden im 3D-Fall.

5.3 Die Gesamtkostenfunktion

In unseren Anwendungen nutzen wir die spezifischen Kosten

$$\begin{aligned} c(x) &= c_1 + c_2 \ln(p_{spec}(t(x))) \\ &\quad + c_3 b(x_3 - h_{ground}) lrf, \end{aligned}$$

$$lrf := \frac{\max\left(\min\left(\left\|x - s_{start}\right\|, \left\|x - s_{goal}\right\|\right), lr\right)}{lr}$$

mit einer Funktion b, die die Kosten für zu hohen Flug und für Bodenberührungen zusammenfasst, z. B.

$$b(x) = \begin{cases} \infty & : \quad x < SHeight \\ x - SHeight & : \quad \text{sonst} \end{cases}.$$

Die Koeffizienten $c_1, \ldots, c_3$ ermöglichen das Tuning des Optimierungsergebnisses bezüglich der Kriterien Pfadlänge, Bedrohungen und Flughöhe, der Parameter lr wird für Start und Landung verwendet und bestimmt den Bereich um Start- und Zielpunkt, in dem ein Unterschreiten der Mindestflughöhe $SHeight$ erlaubt ist.

6. Ergebnisse

Wir betrachten die Algorithmen-Ergebnisse für eine künstliche Umgebung mit hügeligem Terrain, zwei Bedrohungen und einem verbotenen Gebiet (s. Bild 6). Verglichen wurden die mit dem Field-D*-Algorithmus für verschiedene Tuningkoeffizienten $c_1, \ldots, c_3$ erzielten Trajektorien sowie die mit dem D*-Algorithmus erzielten Trajektorien.

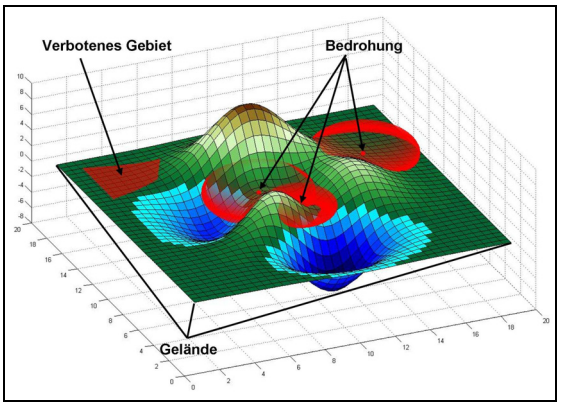

BILD 6. Das künstliche Testszenario

Bild 7 zeigt die deutliche Verbesserung der Trajektorienlänge und der „Glattheit“ der Trajektorie bei der Verwendung des Field-D*-Algorithmus im Vergleich zum D*-Algorithmus bei der Verwendung

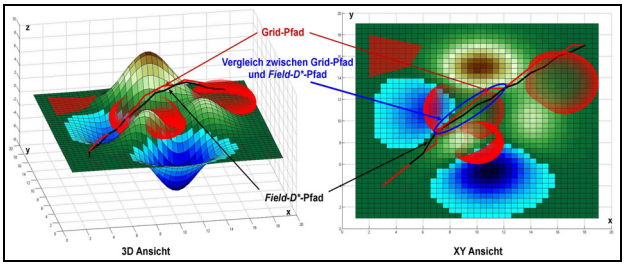

BILD 7. D*-Lite und Field-D*-Pfad

desselben Optimierungskriteriums.

Durch Tuning der Parameter kann das Verhalten des Algorithmus gezielt beeinflusst werden. Wird z. B. der Koeffizient c_2 erhöht, wird der Erfolgswahrscheinlichkeit eine höhere Bedeutung zugemessen. In der entstehenden Trajektorie wird die Bedrohung nun komplett umflogen (s. Bild 8).

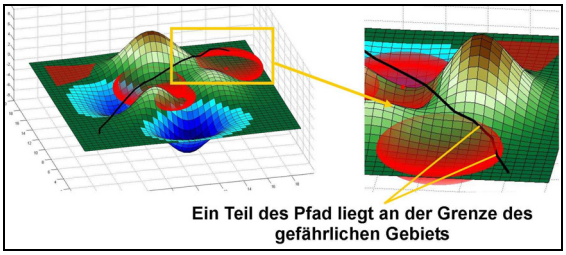

BILD 8. Effekt der Höhergewichtung der Missionserfolgswahrscheinlichkeit

Der bestimmende Faktor ist dabei die Berechnung der optimalen Trajektorien innerhalb einer Rasterzelle. Es zeigt sich (siehe [3]), dass im 3D-Fall die Verwendung einer Approximation die Rechenzeit massiv reduzieren hilft. Die Rechenzeitanforderungen bleiben dann vergleichsweise moderat. Die hier präsentierten Ergebnisse wurden auf einem Standard-PC innerhalb einiger Sekunden ermittelt.

Bild 9 zeigt die Einbettung der Algorithmik in das Missionsplanungssystem MoPS. Es stellt neben geometrischer Bahnplanungsmethoden (kreissegmentbasiert, splinebasiert etc.) automatische (d. h. optimierende) Modi zur Verfügung. Verwendet wird nun ein reales Terrain.

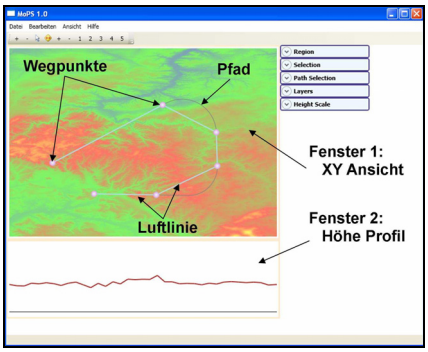

BILD 9. Das Missionsplanungssystem MoPS

Ein einfacher Field-D*-optimierter Flugpfad ist in Bild 10 dargestellt, das zugehörige Planungsgitter wurde überlagert. Die Trajektorien des geplanten Flugpfads schneiden die Rasterzellen an beliebigen Stellen. Unterhalb der 2D-Trajektorie ist das zugehörige Geländeprofil abgebildet.

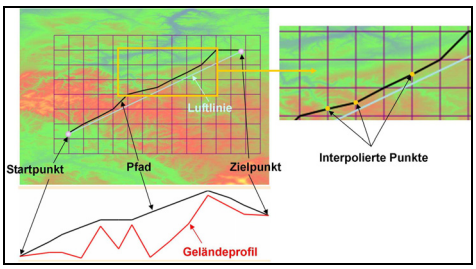

BILD 10. Ein optimierter 3D-Flugpfad

Bild 11 vergleicht den in MoPS Field-D*-optimierten Flugpfad mit einem D*-Lite-optimierten Flugpfad.

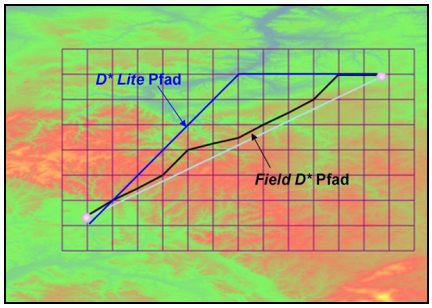

BILD 11. Vergleich zwischen Field-D*-Pfad und D*-Lite Pfad in MoPS

Nach Durchführung einer ersten Pfadoptimierung wird noch ein verbotenes Gebiet eingeführt (s. Bild 12). Die Pfadänderung erfolgt dabei fast in Echtzeit mit dem Verschieben der Ecken des verbotenen Gebiets.

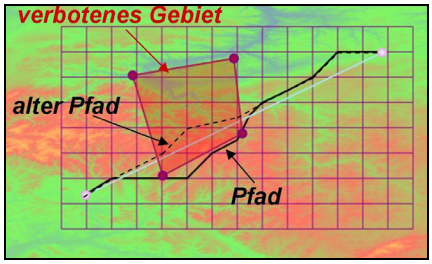

BILD 12. Nachträgliche Einfügung eines verbotenen Gebiets

7. Zusammenfassung

Wir haben die Eignung des Field-D*-Algorithmus zur Planung von UAV-Trajektorien untersucht. Hierbei konnte, bei geeigneter Wahl der Kostenfunktionen und einer Approximationen bei der Ermittlung der Kosten der optimalen Trajektorie innerhalb einer Zelle im 3D-Fall realisierbare und nahe am Optimum liegende Trajektorien generiert werden.

Die Familie der (Field)-D* Algorithmen (s. z. B. auch die „anytime"-Varianten [1], die zu jedem Zeitpunkt der Berechnung einen gültigen Pfad liefern) erweist sich als beachtenswerte Algorithmenfamilie für die Planung von UAV-Trajektorien.

Literaturverzeichnis

[1] van den Berg J., Ferguson D., Kuffner J. 2006: *Anytime Path Planning and Replanning in Dynamic Environments*. Proc. IEEE International Conference on Robotics and Automation (ICRA). Florida, USA. 2006.

[2] Blard D. 2008*: Implementierung eines Missionsplanungssystem zur Flugwegrepräsentierung*, Universität Karlsruhe, Institut für Systemoptimierung (ITE), Diplomarbeit.

[3] Carsten J., Ferguson D., Stentz A. 2006: *3D Field D*: Improved Path Planning and Replanning in Three Dimensions*. Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, Beijing, China 3381-3386.

[4] Ferguson D., Stentz A. 2005: *Field D*: An Interpolation-based Path Planner and Replanner*. Proc. International Symposium on Robotics Research (ISRR), San Franzisco, USA.

[5] Ferguson D., Stentz A. 2006: *Using Interpolation to Improve Path Planning: The Field D* Algorithm*. Journal of Field Robotics 23(2), 79–101.

[6] Ferguson D., Stentz A. 2005: *The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-uniform Cost Environments.* Technical Report CMU-TR-RI-05-19, Carnegie Mellon University.

[7] Gelperin, D. 1977. On the optimality of A*. Artificial Intelligece 8(1): 69-76.

[8] Hart, P., Nilsson, N., Rafael, B. 1968: *A formal basis for the heuristic determination of minimum cost paths*. IEEE trans. Sys. Sci. and Cyb. 4:100–107.

[9] Koenig, S., Likhachev, M. 2002: *Improved fast replanning for robot navigation in unknown terrain.* In Proceedings of the IEEE International conference on Robotics and Automation (ICRA).

[10] Koenig, S., and Likhachev, M. 2002: *D* Lite.* Proceedings of AAAI/IAAI (Innovative Applications of Artificial Intelligence), Edmonton, Canada, pp. 476-483.