Beschreibung der Matlab-Simulationsumgebung

1 Funktionale Beschreibung

Die Simulationsumgebung ermöglicht es, innerhalb von Matlab das zeitliche Verhalten einer Vielzahl von Systemen zu untersuchen. Sie besteht aus einem Simulator-Kern und einer Sammlung von Blöcken, welche die zu simulierenden Modelle implementieren. Das vorliegende Dokument beschreibt nur den Simulator-Kern sowie die formale Blockstruktur. Die Implementierung konkreter Blöcke erfolgt typischerweise im Kontext der jeweiligen Simulationsanwendung.

Der primäre Einsatzbereich der Simulationsumgebung ist die Robotik. Da der Funktionsumfang jedoch im Wesentlichen durch die zur Verfügung stehenden Blöcke bestimmt wird, kann prinzipiell jedes technische System behandelt werden, für welches geeignete Modellbeschreibungen vorliegen, die mit den Möglichkeiten von Matlab umsetzbar sind.

1.1 Überblick

Abbildung 1 zeigt die wesentlichen Elemente der Simulationsumgebung sowie deren Zusammenwirken. Der Simulator erfordert eine Experiment-Beschreibung in Form einer verschachtelten Matlab-Struktur (struct). Deren Aufbau wird in Abschnitt ??? detailliert beschrieben. Der Aufruf des Simulators erfolgt über einen der folgenden Befehle:

```
simulate(<experiment-struct>);
results = simulate_unattended(<experiment-struct>);
```

Die erste Variante startet den Simulator einschließlich der Visualisierung mit vollständiger grafischer Benutzeroberfläche. Der zweite Befehl startet eine Basisversion ohne Visualisierung und eignet sich daher für die Einbindung in eigene Skripte. Da dieser Befehl nur eine Variation des ersten Befehls mit reduziertem Funktionsumfang darstellt, bezieht sich die nachfolgende Beschreibung nur auf die erste Variante.

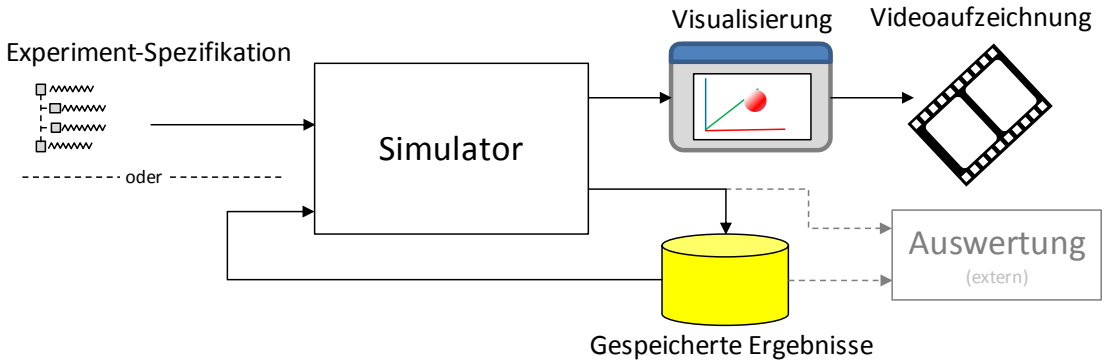


Abbildung 1: Elemente der Simulationsumgebung

Bereits berechnete Simulationsergebnisse werden während der Simulation visualisiert. Die Art der Visualisierung ist durch die am Experiment beteiligten Blöcke vorgegeben. Für weitere Informationen zur Visualisierung sei auf Abschnitt ??? verwiesen. Falls nötig, wird die Simulation dabei auf Realzeit abgebremst, um eine sinnvolle Darstellung zu ermöglichen. Bei entsprechend einfachen Modellen verlängert sich hierdurch die Simulationsdauer. Bei Verwendung der Funktion

`simulate_unattended` (ohne Visualisierung) erfolgt keine derartige Geschwindigkeitsreduktion und somit auch keine Verlängerung der Simulationsdauer. Werden die Modellberechnungen für eine Realzeitdarstellung zu aufwendig, erfolgt die Visualisierung entsprechend verzögert. In diesem Fall kann selbstverständlich auch durch die Verwendung von `simulate_unattended` keine signifikante Reduktion der Simulationsdauer erreicht werden.

Sämtliche während der Simulation anfallenden Daten können auf Wunsch in einer *.mat-Datei (und ggf. zusätzlichen Hilfsdateien) abgespeichert werden und stehen damit einer späteren Auswertung durch entsprechende Matlab-Skripte zur Verfügung. Diese Auswertung ist jedoch nicht mehr Bestandteil der hier beschriebenen Simulationsumgebung. Abgespeicherte Simulationsdaten können darüber hinaus anstelle einer Experimentspezifikation zum Start des Simulators verwendet werden. In diesem Fall ist in den oben angegebenen Funktionsaufrufen der Pfad zur *.mat-Datei anzugeben. Auf diese Weise ist es möglich, eine zuvor unterbrochene Simulation fortzusetzen oder bereits berechnete Ergebnisse (möglicherweise aus einer anderen Perspektive) erneut zu visualisieren. Hierbei überspringt der Simulator-Kern bei bereits vorhandenen Daten alle Modellberechnungen und führt nur den Visualisierungs-Code aus. Dies ist insbesondere bei sehr langsamen Modellberechnungen für eine Demonstration der Ergebnisse sinnvoll. Bei sehr aufwendigen grafischen Darstellungen kann jedoch auch der Visualisierungscode eine signifikante Verzögerung bewirken, sodass eine Darstellung in Realzeit nicht mehr möglich ist. Für diese Fälle besteht die Möglichkeit, vom Hauptfenster der Visualisierung eine Videoaufzeichnung zu erstellen.

1.2 Systemstruktur

Der Hauptbestandteil der Experiment-Spezifikation aus Abbildung 1 ist das zu simulierende System. Dieses besteht aus mehreren Blöcken. Diese besitzen jeweils einen Ausgang und beliebig viele Eingänge, an welche die Ausgänge anderer Blöcke angeschlossen sind. Ein Beispielsystem aus fünf Blöcken ist in Abbildung 2 skizziert. Die an einen Ausgang angeschlossenen Blöcke werden als „abhängige Blöcke“ bezeichnet, da sie zur eigenen Berechnung die vom vorangehenden Block bereitgestellten (Ausgangs-)Daten als (Eingangs-)Daten benötigen. Die Menge dieser Abhängigkeiten bezüglich eines Blocks i wird als $Succ(i)$ bezeichnet. Die umgekehrte Relation, d. h. die Menge der Blöcke, die für einen Block j Eingangsdaten bereitstellen, sei $Pred(j)$. Im Bild sind z. B. die Blöcke 2 und 3 von Block 1 abhängig, d. h. $Succ(1) = \{2, 3\}$ bzw. $Pred(2) = Pred(3) = \{1\}$. Block 1 ist seinerseits von den Blöcken 4 und 5 abhängig. Alle Verbindungen sind stets als rückwirkungsfrei zu betrachten. Darüber hinaus bestehen keine weiteren Anforderungen an das Datenformat. Es können beliebige von Matlab unterstützte Formulierungen verwendet werden (skalare Werte, Matrizen, Strukturen, Cell-Arrays, verschachtelte Strukturen…). Wenn nötig, kann das Format in jedem Berechnungsschritt wechseln („nonuniform output“). Dieser Fall bewirkt jedoch einen größeren Overhead beim Speichern der Simulationsergebnisse.

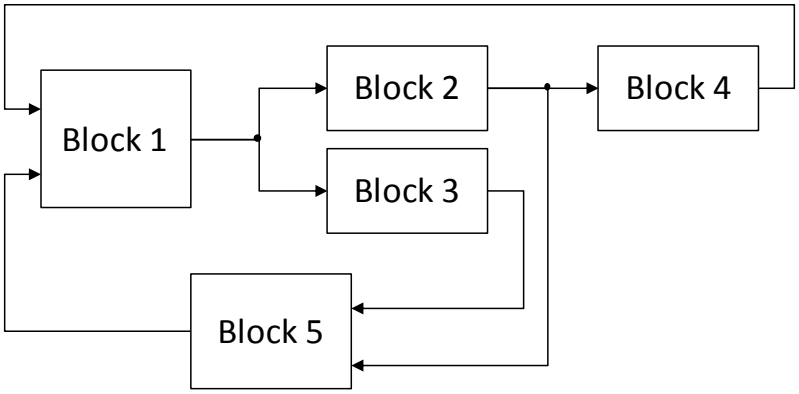
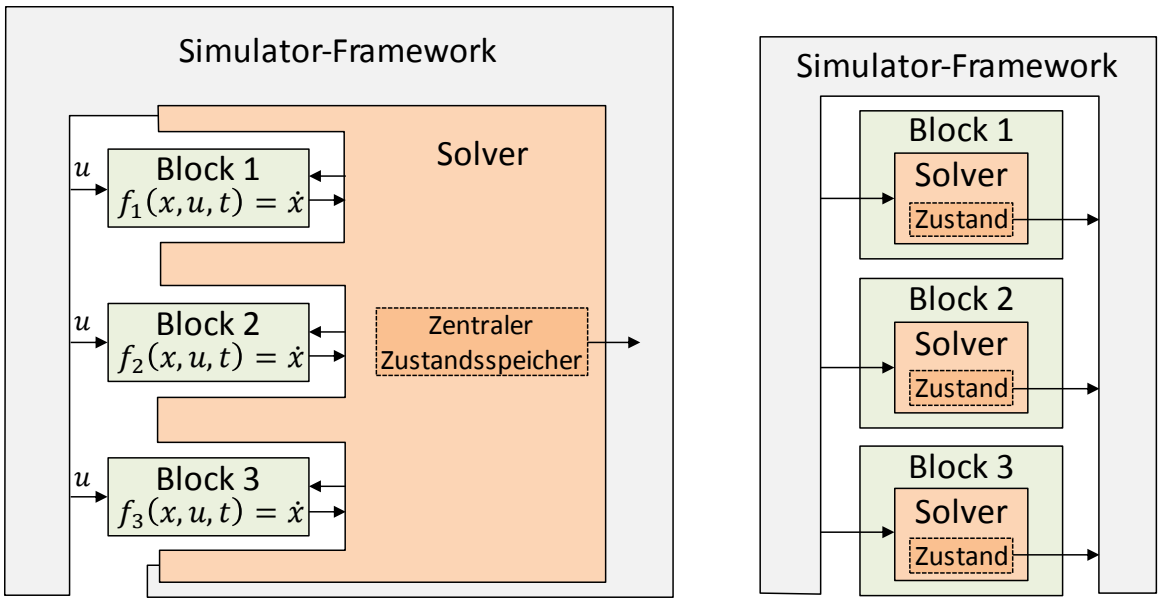


Abbildung 2: Ein Beispiel-System bestehend aus fünf Blöcken

Strukturell besteht eine starke Ähnlichkeit zu Simulink. Ein Vorteil der hier beschriebenen Simulationsumgebung gegenüber Simulink ist die vereinfachte Visualisierung und Datenaufzeichnung. Der wesentliche Unterschied ist jedoch die Verlagerung des Solvers aus dem Simulator-Kern in die Modellblöcke. Während der numerische Solver ein zentraler Bestandteil von Simulink ist und jeder (zeitkontinuierliche) Block in einer kompatiblen Beschreibung formuliert sein muss, überlässt die hier beschriebene Simulationsumgebung die Lösung von Modellgleichungen vollständig den jeweiligen Blöcken (vgl. Abbildung 3). Die Schnittstelle zwischen Block und Simulator-Framework reduziert sich auf die Rückgabe des internen Zustands sowie daraus abgeleiteter Ausgangsdaten zu bestimmten, streng monoton wachsenden Zeitpunkten. Dies ist bei statischen oder zeitdiskret arbeitenden Blöcken ohne Solver möglich. In kontinuierlichen Blöcken kann die gewünschte Zustandsinformation je nach Modellkomplexität z. B. auf analytischem Weg oder durch eine geeignete numerische Technik generiert werden. Der genaue zeitliche Ablauf einer Simulation wird im Abschnitt ??? beschrieben.



(a) Architektur von Simulink

(b) Architektur der Matlab-Simulationnsumgebung

Abbildung 3: Architekturvergleich der Simulationsumgebungen

Die gewählte Architektur mag für Systeme, die primär durch zeitkontinuierliche Modelle beschrieben sind, weniger geeignet sein. Für derartige Systeme sollte weiterhin Simulink verwendet werden. Der typische Anwendungsfall für die hier vorgestellte Lösung ist die Erprobung von ggf. komplexen Signalverarbeitungsalgorithmen. Diese werden in zeitdiskret arbeitende Blöcke integriert und ggf. mit

wenigen kontinuierlichen Blöcken, die physikalische Systemkomponenten darstellen, gekoppelt. Ein realistisches Beispiel eines geeigneten Systems ist die in Abbildung 4 dargestellte Flugzeugsteuerung mit optisch unterstützter Navigation. Die Flugdynamik ist hier der einzige zeitkontinuierlich arbeitende Block ($\Delta t = 0$). Die Implementierung kann beispielsweise mit Hilfe der Funktion `ode45` erfolgen. Die „Abtastzeitpunkte“ werden durch die angeschlossenen zeitdiskret arbeitenden Sensorblöcke bestimmt (vgl. Abschnitt ???), die entsprechend der definierten Abtastintervalle T_{IMU} und T_{CAM} periodisch berechnet werden. Die Auswertung der Sensordaten erfolgt in komplexen Algorithmen, die jeweils in zeitdiskret arbeitende Blöcke eingebettet sind. Im Falle der Bildverarbeitung steuert ein Triggersignal des Kameramodells die Berechnungszeitpunkte, wodurch dieser Block automatisch mit dem Kameramodell synchronisiert wird.

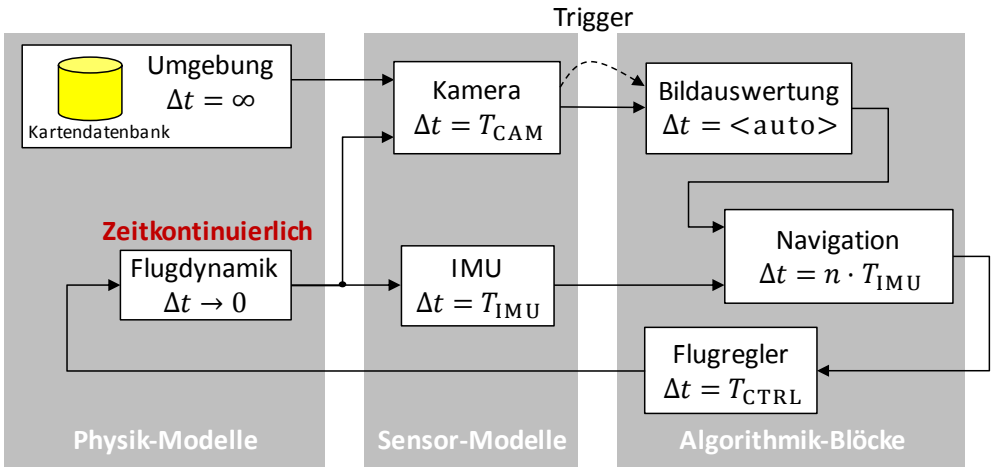


Abbildung 4: Flugzeugsteuerung mit optisch unterstützter Navigation als Beispielsystem

1.3 Funktionsweise der Simulation

Die Simulation eines Systems erfolgt durch die wiederholte Berechnung der konstituierenden Modellblöcke zu bestimmten diskreten Zeitpunkten entsprechend der in den Blöcken hinterlegten Abtastintervalle. Die dabei generierten Ausgabedaten werden den abhängigen Blöcken als Eingabedaten zur Verfügung gestellt. Dazu verwaltet der Simulator-Kern die aktuelle Systemzeit t und für jeden Block i den Zeitpunkt der letzten Berechnung $t_{i,\mathrm{last}} \leq t$ sowie den nächsten Berechnungszeitpunkt $t_{i,\mathrm{next}} \geq t > t_{i,\mathrm{last}}$. Der Simulationsablauf ist in der folgenden Pseudocode-Formulierung zusammengefasst:

```
simulate(system)
    initialize: t = 0; ∀i t_{i,last} = −∞, t_{i,next} = t_{i,1}
    while experiment not finished do
        i_next = argmin_i(t_{i,next})
        t = min(t_{i,next})
        handle_block(i_next, i_next)
    end
end

handle_block(i, i_stop)
    foreach j ∈ Pred(i) do
       if t_{i,last} < t and t_{i,next} ≥ t and j ≠ i_stop
            handle_block(j, i_stop)
       end
    end
    process(i) // compute block outputs for time t
    t_{i,last} := t
    t_{i,next} := t + Δt_i
end
```

Zu Beginn wird die Simulationszeit zu $t = 0$ und alle $t_{i,\text{next}}$ mit einem nutzerdefinierten Offset $t_{i,1} \geq 0$ (typ. $t_{i,1} = 0$) initialisiert. In der anschließenden Simulator-Hauptschleife wird zunächst jeweils der als nächstes auszuführende Block i_{next} anhand des kleinsten $t_{i,\text{next}}$-Wertes bestimmt. Vor der Ausführung dieses Blocks werden alle Blöcke $j \in Pred(i_{\text{next}})$ mit $t_{j,\text{last}} < t$ und $t_{j,\text{next}} \geq t$ berechnet, um die notwendigen Eingangsdaten für i_{next} bereitzustellen. Diese Prüfung erfolgt rekursiv auch für alle $k \in Pred(j)$ u. s. w. Falls dabei algebraische Schleifen auftreten, werden diese bei i_{next} aufgetrennt. Im Allgemeinen wird der Zeitpunkt $t_{i,\text{next}}$ für mehrere oder sogar alle Blöcke i übereinstimmen, sodass die Auswahl nicht eindeutig ist. Aufgrund der zuvor beschriebenen Abhängigkeiten-Analyse ist die konkrete Auswahl jedoch unerheblich für die Berechnungsreihenfolge (Ausnahmen bestehend bei algebraischen Schleifen!).

In der obigen Darstellung wurde auf spezifische Details bei der Behandlung zeitkontinuierlicher Blöcke sowie die Realisierung von Trigger-Ereignissen verzichtet. Für Details hierzu sei auf die ausführliche Beschreibung in Abschnitt ??? verwiesen.

1.4 Visualisierung und Simulationssteuerung

Die Visualisierung erfolgt größtenteils im Hauptfenster der Simulationsumgebung. Dieses beinhaltet auch die Bedienelemente zur Simulationssteuerung. Abbildung 5 zeigt einen Screenshot des Hauptfensters mit den wichtigsten Elementen.

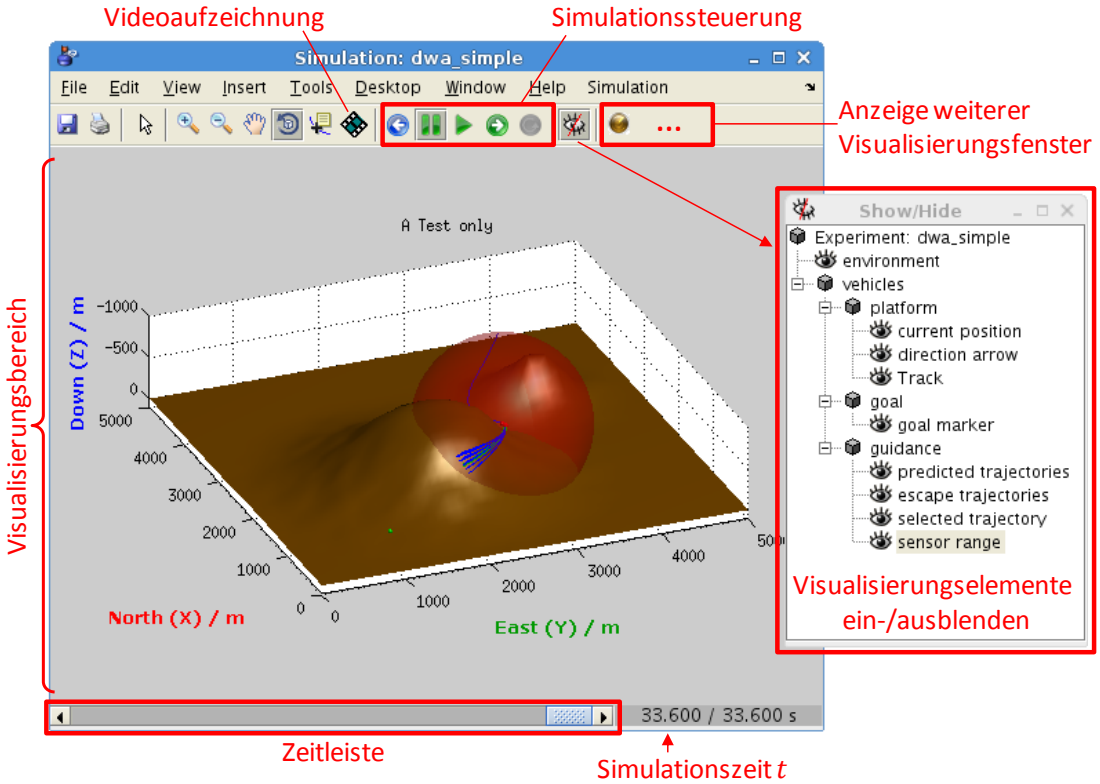


Abbildung 5: Screenshot der Simulationsumgebung mit Bezeichnung wichtiger Bedienelemente

Die Elemente zur Simulationssteuerung in der Toolbar ermöglichen das Starten/Anhalten/Fortsetzen der Simulation sowie den Einzelschrittbetrieb. Dieselben Elemente steuern auch die Replay-Funktion bereits berechneter Daten. Hierbei ändert sich die Farbe der Symbole von grün zu blau. Der Visualisierungsbereich besteht aus einer Matlab `axes`. Eine Vielzahl von Darstellungsparametern kann in der Experimentstruktur hinterlegt werden, sodass je nach Anwenderwunsch eine zweidimensionale oder dreidimensionale Darstellung entsteht. Innerhalb des Visualisierungsbereichs

kann jeder Block beliebige Grafikobjekte (Matlab Handle Graphics Objekte) platzieren. Das Simulations-Framework verwaltet die Handles und zeigt diese auf Wunsch in einer Baumstruktur an, sodass einzelne Objekte gezielt zur Laufzeit ein- und ausgeblendet werden können um ein besseres Verständnis des dargestellten Systems zu erlangen. Das Ausblenden von Grafikelementen verringert zudem den Rechenaufwand, da der Visualisierungscode nur für sichtbare Grafikelemente aufgerufen wird. Bei komplexen Modellblöcken kann es darüber hinaus sinnvoll sein, interne Daten außerhalb des Hauptfensters zu visualisieren. Daher besteht für jeden Block die Möglichkeit, eigene Visualisierungsfenster zu definieren. Diese lassen sich über die Symbole am rechten Rand der Toolbar ein- und ausblenden.
Unterhalb des Visualisierungsbereichs befindet sich die Zeitleiste, die wie in einem Videoplayer eine schnelle Navigation durch die bereits berechneten Simulationsdaten erlaubt. Nebenan sind die aktuelle Simulationszeit sowie der späteste Zeitpunkt, für den bereits Simulationsdaten vorliegen, eingeblendet. Ein Verschieben der Zeitleiste bei laufender Simulation pausiert diese automatisch und wechselt in den Replay-Modus (Symbole werden blau). Erreicht die Zeitleiste im Replay-Modus den rechten Rand, wird die Simulation nahtlos fortgesetzt (Symbole werden grün) und es werden weitere Datensätze berechnet.

2 Detaillierte Beschreibung

2.1 Experimentspezifikation

2.2 Blöcke

2.3 Logging

2.4 ...