Der Simulink-Knigge

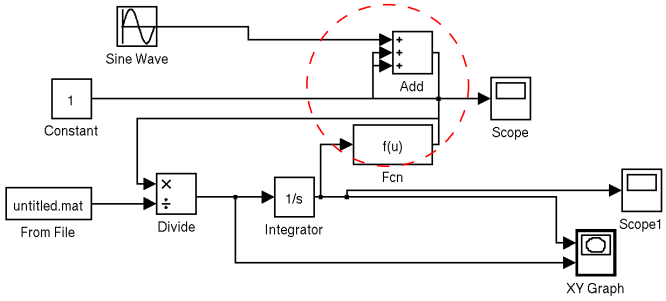
Florian Brunner, Gregor Goebel und Jan Maximilian Montenbruck

14. November 2012

Zusammenfassung

In vielen praktischen und akademischen Anwendungen wird MATLAB & Simulink zur Modellbildung, Simulation und Regelung technischer Systeme verwendet. Häufig weisen die resultierenden Programme hierbei eine hohe Komplexität auf, sodass selbige später oft nicht ohne größeren zusätzlichen Aufwand richtig nachvollzogen oder geändert werden können. Dies gilt nicht nur für Fremdanwender, sondern auch für den Urheber des Programmes selbst. Um diesen Komplikationen vorzubeugen, ist es ratsam, sich an einige einfache Regeln zu halten, die zur Übersichtlichkeit von `*.mdl` Dateien beitragen. In diesem Dokument sollen einige wichtige dieser Regeln kurz aufgezeigt und erläutert werden.

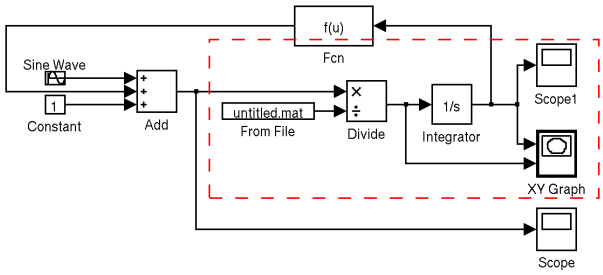
Innerhalb dieser Anleitung werden wir ein einfaches Simulink-Modell schrittweise verbessern. Hierbei gehen wir von einem zunächst sehr unübersichtlichen Modell aus. Insbesondere sollte hier bemerkt werden, dass die Signalflüsse in-



nerhalb des markierten Bereiches der Abbildung nicht eindeutig unterschieden werden können. Es ist mitunter nicht möglich zu erkennen a) welche Signale der Eingang von *Scope*, der Eingang 3 von *Add* und der Eingang 1 von *Divide* sind, sowie b) wie die Ausgangssignale von *Constant*, *Fcn* und *Add* genutzt werden. Zunächst sollen also diese Uneindeutigkeiten im Signalfluss korrigiert werden. Dieses geschieht durch Umordnung von Blöcken und Signalen.

Schritt 1. Organisiere die Signale deines Modelles so, dass sie sich nicht schneiden oder übereinander liegen!

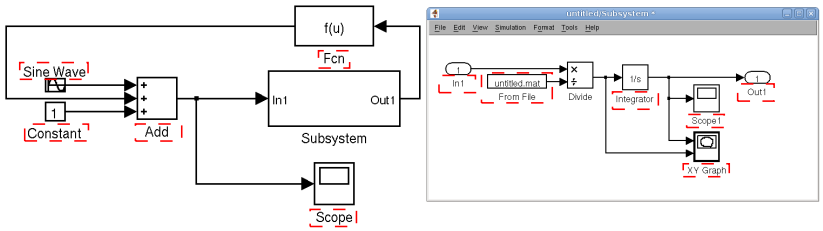
Wir erhalten ein deutlich übersichtlicheres Modell, in dem nun eine eindeutige Zuordnung von Ein- zu Ausängen möglich ist. Noch immer ist jedoch keine Abstraktion der Modellstruktur möglich; Schliesslich sind die Elementarblöcke



nicht gemäss ihrer Aufgaben oder Systemgrenzen geordnet. Solch eine mangelnde Ordnung kann durch die Einteilung von Elementarblöcken in Subsysteme erfolgen. Werden zusätzlich sinnvolle Ein- und Ausgänge des *Subsystem*-Blocks definiert, kann im vorliegenden Fall beispielhaft eine Strecke bestehend aus einer Störgrösse und einem Integrator separiert werden.

Schritt 2. Unterteile dein Modell in sinnvolle Subsysteme!

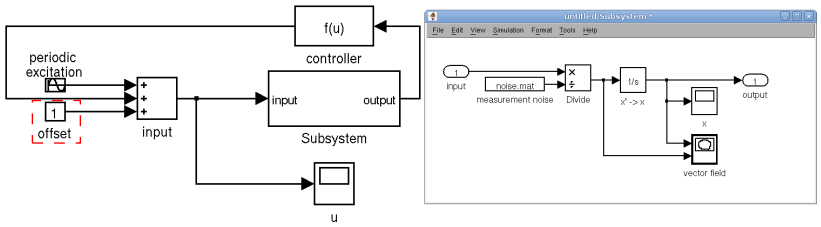
Der vorangegangene Schritt trägt bereits wesentlich zur Verständlichkeit des Modelles bei, wie aus der resultierenden Abbildung hervor geht. Nun soll es



aber zusätzlich möglich sein, die physikalische/technische/anschauliche Bedeutung der Blöcke ohne Vorkenntnisse zu erkennen. Hierzu ist es ratsam, die Blöcke mit für den Anwender intuitiv erklärenden Namen zu benennen. Hierzu genügt exemplarisch keineswegs die Bezeichnung *Integrator*, sondern vielmehr die Bezeichnung `x'->x`.

Schritt 3. Benenne die Elemente deines Blockschaltbildes!

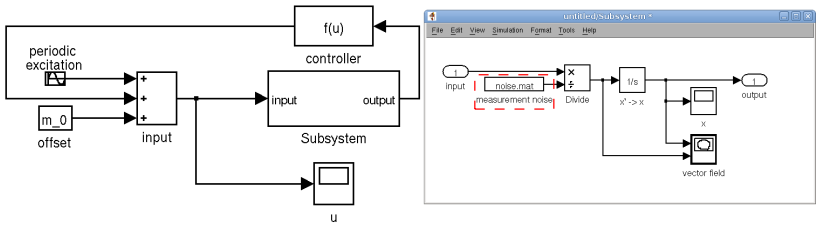
Nach erfolgreicher Durchführung dieses Schrittes hat das Modell bereits einige Eigenschaften, die unseren Anforderungen an intuitive Verständlichkeit genügen. Ferner können jedoch auch funktionale Anforderungen gestellt werden.



Beispielsweise ist es ratsam, niemals numerische Werte für Eigenschaften von Blöcken vorzugeben, sondern stets Variablen, die Simulink automatisch im aktuellen Workspace abgreift. Dies impliziert, dass es ein zum Modell gehöriges `*.m`-file erstellt wird (meist `init.m`), in dem alle erforderlichen Variablen definiert werden. Dies kann nicht nur zum Ermöglichen einer späteren automatischen Abfolge von Simulationsreihen mit verschiedenen Parametern beitragen, sondern vereinfacht auch das mitunter langwierige Suchen nach dem für den Einfluss des Parameters entscheidenden Block. Im vorliegenden Beispiel ist es sinnvoll, den enthaltenen *offset* statt des numerischen Wertes `1` durch `m_0` zu konfigurieren, und diesen Wert in einem seperaten `*.m`-file `init.m` durch `m_0=1` zu definieren. Konsequent muss `init.m` fortan zu Beginn einer Simulation ausgeführt werden, oder via `Model Properties`, `Callbacks`, `StartFcn` eingebunden werden.

Schritt 4. Benutze durch `*.m`-files initialisierte Variablen statt numerischer Werte!

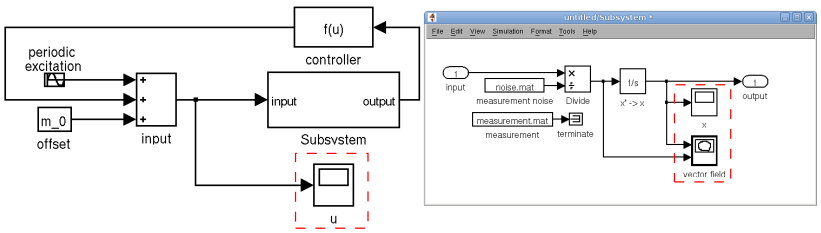
Diese Vorgehensweise ergibt in Summe mit den vorhergegangenen Schritten bereits einen sehr professionellen Umgang mit dem Simulationsmodell. Eine wei-



tere Quelle für Unübersichtlichkeit sind Modellstrukturen und Blöcke, die den Benutzer zu einem *copy & paste* Verhalten verführen. Häufig iterieren wir bei der Erstellung von Reglern und Modellen zwischen verschiedenen Varianten des gleichen Systems und kopieren der Einfachheit halber die Varianten schlicht nebeneinander. Selbiges gilt häufig auch für *read*- und *write*-Blöcke. Obwohl es generell zu vermeiden ist, verschiedene Systemvarianten im gleichen Modell abzubilden (d.h. bei strukturellen Änderungen *kein* copy & paste, sondern richtiges *Versionieren* und *Löschen*), erleichtert es in einigen seltenen Fällen tatsächlich des Handling verschiedener Simulationsreihen. In solchen und nur in solchen Fällen können verschiedene strukturelle Varianten des Systems im selbgen Modell koexistieren, insofern die ungenutzten Signale *stets* terminiert werden.

Schritt 5. Wenn du dir verschiedene Optionen für deine Modellstruktur offen hälst, lösche obsolete Blöcke oder terminiere sie, falls notwendig!

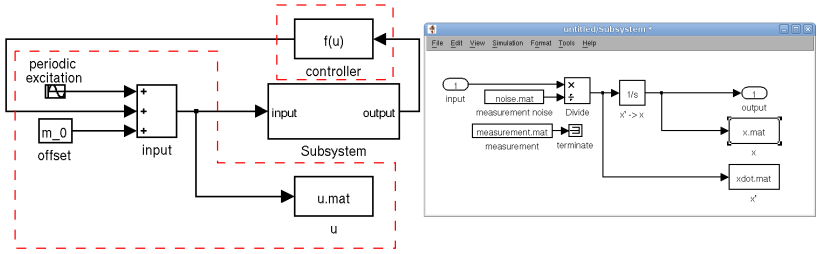
Ein Gros von Zweideutigkeiten und Unübersichtlichkeiten wird durch die oben genannten Massnahmen bereits rigoros unterbunden. Gehen wir von unserem hier beispielhaft behandelten Modell aus, so ergibt sich zunächst die hier abgebildete Struktur, in der zur einfachen Auswertung der Simulationsergebnisse einige *scopes* genutzt werden. Die Nutzung von scopes ist für die schnelle und einfache Auswertung von ersten Simulationsergebnissen sicher wichtig und richtig. Aus zweierlei Gründen reicht diese Art der Auswertung jedoch langfristig häufig nicht aus; a) Nimmt die Anzahl der für die Auswertung interessanten Größen mit der Komplexität des Modelles zu, sodass die Lokalisierung und Identifizierung von scopes schwierig werden kann und b) sollen die Ergeb-



nisse der Simulation häufig für Präsentationen, Berechnungen oder Dokumentationen weiterverwendet werden. Scopes genügen weder den Anforderungen an die Strukturierung der Simulationsergebnisse noch gewährleisten sie ein für oben genannte Aufgaben hinreichendes *postprocessing*. Aus diesem Grunde ist es Möglich, Signalverläufe auch in `*.mat`-files (oder Variablen im workspace) zu sichern, die nach abgeschlossener Simulation in gleicher Weise durch `plot(·,·)` ausgewertet werden können. Darüber hinaus bieten diese Formate jedoch auch die Möglichkeit zur Anwendung von MATLAB-Befehlen jeglicher Art, die auch via `Model Properties`, `Callbacks`, `StopFcn` automatisiert werden können.

Schritt 6. Benutze *Scopes* nur für Testläufe und speichere Signale generell in `*.mat`-files!

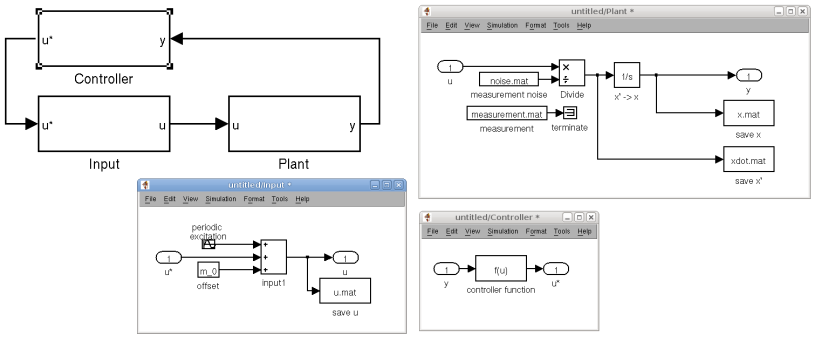
Das resultierende, insgesamt sehr ansehnliche Modell genügt nun zu grossen Teilen den eingangs definierten Anforderungen; Dem Modell mangelt es jedoch weiterhin an der Definition verschiedener Abstraktionsebenen. Genauer gesagt fehlt eine hierarchische Gliederung von Modellebenen. An unserem Beispiel wird dies gut anhand des bereits definierten Subsystems klar. Zwar separiert das Subsystem die logisch zueinander gehörigen Blöcke, es befindet sich jedoch auf der gleichen Ebene wie Elementarblöcke (*input*, *offset* oder *controller*), obwohl es sowohl zu einer anderen Abstraktions- als auch zu einer anderen Hierarchieebene gehört (der Begriff *Strecke ist dem darin befindlichen Integrator übergeordnet.* Die hierarchische Gliederung oder Aufteilung in Abstraktionsebenen kann also



einfach dadurch erfolgen, dass sich Elementarblöcke nur auf der letzten Ebene einer Struktur aus Subsystemen befinden dürfen. Diese Massnahme wird insbesondere bei sehr grossen und komplexen Modellen relevant.

Schritt 7. Versuche, dein Modell hierarchisch zu gliedern, so dass sich auf einer Ebene entweder nur Subsysteme oder nur Elementarblöcke befinden!

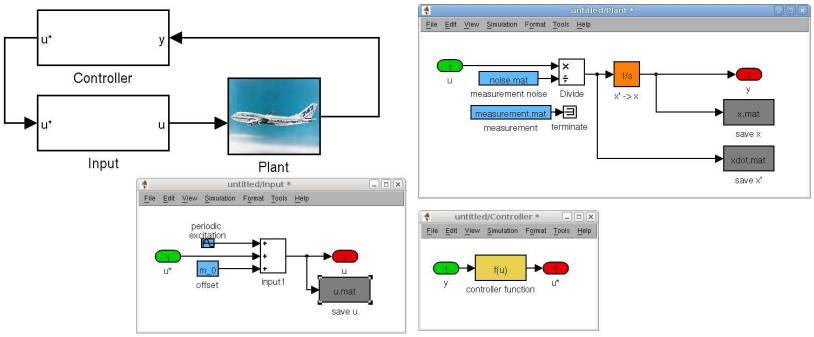
Das Modell ist damit frei von Mängeln und genügt allen *strukturellen* Anforderungen. Damit sind alle *notwendigen* Korrekturen erfolgt. Über strukturelle Ordnung hinaus lässt sich jedoch noch eine weitere - nicht unbedingt notwendige



- Form der Ordnung definieren, die ebenfalls zur intuitiveren Nutzbarkeit des Modells beiträgt. Als letzter Schritt kann dem Modell eine *rein* visuelle Ordnung gegeben werden, indem die Färbung von Blöcken konsistent durch eine vom Anwender definierte Farblegende erfolgt.

Schritt 8. Definiere dir eine sinnvolle Farblegende, um die Aufgabe und Bedeutung von Blöcken schnell identifizieren zu können!

Im vorliegenden Beispiel sind Eingangssignale stets grün, Ausgangssignale rot, externe Eingänge hellblau, externe Ausgänge grau, Integratoren orange und benutzerdefinierte Funktionen gelb. Zusätzlich ist die Strecke durch ein Bild maskiert worden. Ersteres erfolgt durch `Background Color` und zweiteres durch `Mask Subsystem` und `image(imread('.'))`. Insgesamt haben wir somit ein völlig



unordentliches und uneindeutiges Modell in acht einfachen Schritten zu einem leicht verständlichen und wiederverwendbaren Modell gemacht.