Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Werkzeugmaschinen & Fertigung

Automatic extraction of POMDPs from Simulink models

Oliver Stollmann

Bachelor Thesis
2011

Bastian Migge
PD Dr. habil. Andreas Kunz

Prof. Dr. Konrad Wegener

Abstract

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum.

Your task description pdf file here!

Acknowledgment

This is the acknowledgment

Contents

List of Figures

List of Tables

Introduction

This is the introduction

Motivation

This is the "Motivation" chapter!

Background

The following chapter provides a quick introduction to the theoretical foundation of this work. The extraction of *Partially Observable Markov Decision Processes* from *Simulink* model requires an understanding of *dynamic systems*, *stochastic processes*, specifically *Markov Chains*, *Markov Decision Processes* and *Partially Observable Markov Decision Processes*, and finally *Simulink*. This chapter introduces each of these concepts or tools, providing examples where helpful. An understanding of these key concepts will facilitate the understanding of the next two chapters, *Methodology* and *Implementation*.

3.1 Dynamic Systems

This section covers deterministic and randomized dynamic systems in theory and gives a few intuitive examples of dynamic systems.

Dynamic systems are systems whose development over time can be described by a single or a set of mathematical equations. Although these equations may not always be symbolically solvable they describe the system's dynamics perfectly. Examples include the time-varying temperature of an object as it is placed in the oven or the voltage across an inductor.

A common example of dynamic systems is the ideal one-dimensional mathematical pendulum seen in Figure 3.1. By considering the forces on the pendulum or it's energy it is quite simple to deduce the following differential equation to describe the system:

$$0 = \frac{g}{l} \cdot sin(\phi(t)) + \ddot{\phi}(t)$$

With the small-angles assumption

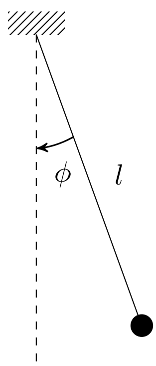

Figure 3.1: *1-dimensional mathematical pendulum*

$$sin(\phi(t)) \approx \phi(t)$$

the solution of the differential equation is

$$\phi(t) = \hat{\phi} \cdot sin(\sqrt{\frac{g}{l}} \cdot t + \phi_0)$$

where $\hat{\phi}$ is the semi-amplitude and ϕ_0 the phase at time $t = 0$. This equation describes the dynamic system perfectly for all times.

An interesting property of the pendulum system is that given the same initial state and excitation (eg. initial angle at 20° and speed 0), the system will behave the same way at as time progresses. This property is called determinism and guarantees that given the same excitation and initial state the system will always develop identically over time. Systems that posess this property are called *deterministic dynamic systems* and differ greatly from the opposed *randomized dynamic systems*.

Randomized dynamic systems are dynamic systems with an element of *randomness*. The consequence of this is that the same initial conditions and excitation do not guarantee an identicial system response. A trivial example of a discrete randomized dynamic system is the following:

$$\begin{aligned} q[n+1] &= q[n] + x[n] + e[n] \\ y[n] &= q[n] + x[n] \end{aligned}$$

where $x[n]$ is the input, $e[n]$ is white noise and $y[n]$ the output. If this system is provided with the same input signal twice, the resulting output signal is likely to be slightly different. This is caused by the inherit randomness of a white noise input.

The above example touches on an important point in the field of signal theory and system dynamics. The pendulum example also differs from the white noise example because the former is defined in *continuous time* whilst the latter is defined in *discrete time*. A continuous time system is defined for any time value $t \in T$, where T is the system's time space. A discrete time system is, on the other hand, only defined on a subset of the time line T, at discrete times $n \in N \subset T$.

A simple example to illustrate this point is the comparison of the continuous and the discrete sine functions:

$$\begin{aligned} y(t) &= sin(t), \\ y[n] &= sin(n) = sin(n * T_s). \end{aligned}$$

Here $y[n]$ is the discrete-time version of the continuous time system $y(t)$. As one can see, $y[n]$ is produced by *sampling* $y(n)$ with the sampling frequency T_s. This means that whilst $y(t)$ is defined for all times t, $y[n]$ is only defined for the times

$$
\begin{aligned}
&n \in N \\
where& \\
&N = n \cdot T_s \,\forall\, n \,\in\, \mathbb{Z} \\
&\quad = [-\infty \cdot T_s, -(\infty - 1) \cdot T_s, \cdots, -1 \cdot T_s, 0, -1 \cdot T_s, \cdots, (\infty - 1) \cdot T_s, \infty \cdot T_s].
\end{aligned}
$$

It is easy to see here that a discrete time system contains less information than a continuous time signal. Nonetheless discrete time signals are prevalent because computers can only deal with digital (ie. discrete) signals.

The last important property of dynamic systems is the notion of *state*. A system's state is the smallest set of internal and/or external values that represent the entire state of the system. This means that a system's state completely describes that systems condition at a certain time. Coming back to the pendulum example it is clear that the entire system's condition can be described by two variables, the current angle $\phi(t)$ and the current angular velocity $\dot{\phi}(t)$. A definition of these two values at time t completely define the system's condition in past and future times $t \pm \tau \in T$. The set of all possible states that a system may find itself in is defined as the *state space* of that system.

Deterministic dynamic systems and *randomized dynamic systems* are two extremely useful mathematical constructs and serve as the basis of the more advanced theory of the next few sections.

3.2 Stochastic Processes

A stochastic process is a set of random variables indexed by a parameter. If the indexing parameter is time and the random variables represent possible states then a random process describes a randomized dynamic system that reaches certain states at certain times. The formal definition of a time-indexed stochastic process is a collection $(X_t : t \in T)$ on a probability space where t is the time-index and X_t a random variable on the state space X.

A number of properties allow a more detailed classification of random processes: if the index set T is countable the process is *discrete* and if it is not countable the process is *continuous*, if the state space X is finite the process has a *finite state space* and if the random variable X_t represents values from a countable set the process values are *discrete* and otherwise *continuous*.

A common example of stochastic processes is Fractional Brownian motion as seen in Figure 3.2.

3.2.1 Markov Chains

A Markov Chain is a stochastic process that describes the dynamics of a probabilistic system by defining the conditional transition probabilities $Pr(X_{n+1} = x|X_n = x_n)$ between members of a finite or infinite state-space. Markov chains posess the Markov Property,

$$Pr(X_{n+1} = x|X_1 = x_1, X_2 = x_2, ..., X_n = x_n) = Pr(X_{n+1} = x|X_n = x_n),$$

which states that the current conditional transition probabilities are dependent only on the system's current state.

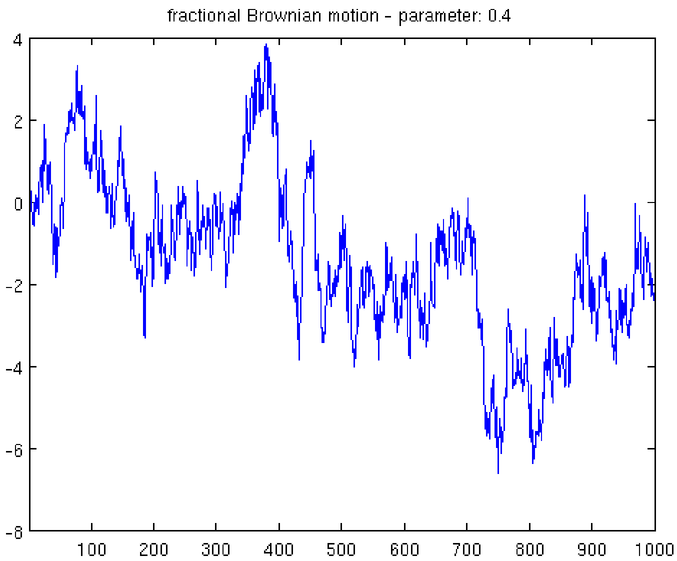

Figure 3.2: *Fractional Brownian motion with Hurst parameter of* 0.4

If a Markov Chain's state-space is *finite* the transition probabilities between states can be represented in a *transition probability matrix* where the probability of going from state i to state j, $Pr(X_{n+1} = j|X_n = i)$, is equal to the (i, j) element of the matrix:

$$Pr = \begin{pmatrix} Pr(X_{n+1} = 1|X_n = 1) & Pr(X_{n+1} = 2|X_n = 1) & \cdots & Pr(X_{n+1} = N|X_n = 1) \\ Pr(X_{n+1} = 1|X_n = 2) & Pr(X_{n+1} = 2|X_n = 2) & \cdots & Pr(X_{n+1} = N|X_n = 2) \\ \vdots & \vdots & \ddots & \vdots \\ Pr(X_{n+1} = 1|X_n = N) & Pr(X_{n+1} = 2|X_n = N) & \cdots & Pr(X_{n+1} = N|X_n = N) \end{pmatrix}$$

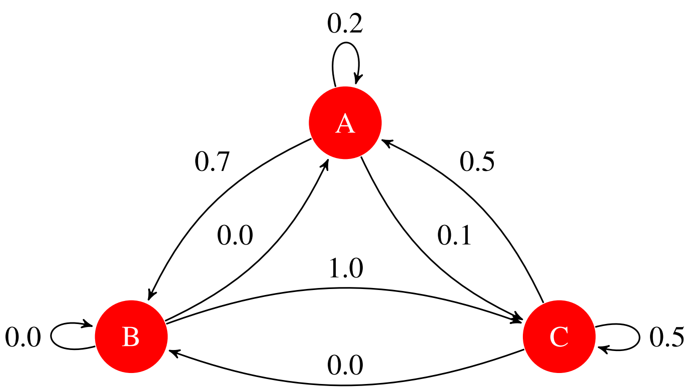

Figure 3.3: *Three-State Markov Chain*

Figure 3.3 shows an example of a three state Markov Chain. The transition probabilities matrix is as follows:

$$\begin{pmatrix} 0.2 & 0.7 & 0.1 \\ 0 & 0 & 1 \\ 0.5 & 0 & 0.5 \end{pmatrix}$$

Looking at this transition probability matrix or Figure 3.3, we can, for example, deduce that if the system is currently in state A the probability of transitioning to state B in the next step is 0.7, the probability of transitioning to state C is 0.1 and the probability of remaining in state A is 0.2. Interestingly, because the second row only contains a single non-zero value, which must thus be equal to 1, the transition from state B is entirely deterministic, meaning that it will always be the same.

3.2.2 Markov Decision Processes

Markov Decision Processes (MDPs) are an extension of Markov Chains and describe *controllable* probabilistic dynamic systems. They are defined as a four touple (S, A, P, R) with:

- S: set of states,
- A: set of actions,
- P: conditional transition probabilities,
- R: rewards.

MDPs are used to model probabilistic systems that can be influenced through decision-taking. These decisions are represented as actions and have a direct influence on the transition probabilities of the system. This idea of actions influencing transition probabilities can also be interpreted as *systems with uncertain actions*. Transition probabilities are now three-dimensional and depend not only on the current state, but also on the action being taken; $Pr(X_{n+1} = x_n | X_n = x, a_n = a)$ is the probability of going to state x_n in the next step given that the system is currently in state x and action a has been chosen.

MDPs also define rewards. Although rewards are not necessary to describe *controllable* dynamic systems, they are necessary when using MDPs for optimization. Every transition probability is paired with a reward, or cost (negative reward), value; $R(s, s', a)$ is the reward (scalar) that the system receives when it transitions from state s' to state s given that action a was chosen. *Note that rewards are not inherently probabilistic*.

MDPs can be used to optimize decision making. The combination of a system description and an associated reward model allows the computation on an optimal decision to take in a given situation. The aim of this optimization is to produce a *policy*, $\pi(s)$, that defines exactly which action should be taken if the system finds itself in a certain state.

The computation of an optimal policy requires the definition of *optimality*. In most cases optimization simply aims for the maximization of rewards (or minimization of costs) over a certain decision span. This optimization goal is defined in a so called reward function, the most common of which is the *expected discounted total reward* (infinite horizon),

$$\sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1}),$$

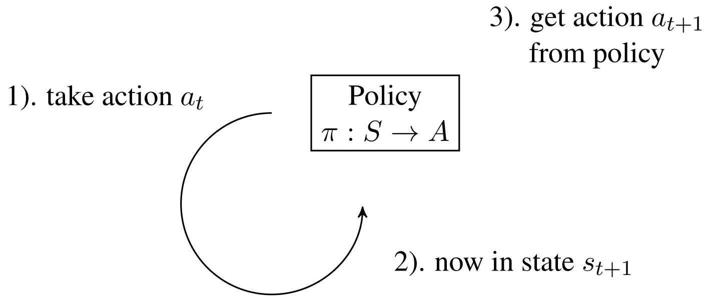

Figure 3.4: *MDP control loop*

with:

- γ: discount factor, where $\gamma \in (0, 1]$,
- $R_{a_t}(s_t, s_{t+1})$: reward received in $t+1$ for taking action a from state s_t at time t.

The *expected discounted total reward* seen above is one of the four traditional reward functions, defined in [Put94] as: TODO. The *expected discounted total reward* represents the idea that the decision maker values the total sum of rewards, but prefers rewards received earlier to rewards received later, ie. *rewards are discounted over time*. This reward function has a valuable property of being non-myopic. It takes into account not only the reward received in the next step, but looks far into the future taking into account that actions that are highly rewarded in the short-term may in fact be the wrong decision because of their effect in the long-term.

Given an MDP and a reward function, an optimal policy can be computed. The computation of such a policy can be undertaken using either *linear programming* or, more commonly, *dynamic programming*, namely value iteration. An in-depth analysis of the different policy computation algorithms is out of the scope of this text, but is covered in detail by the field's literature [Put94]. The result of the policy computation is, as described above, a policy $\pi(s)$ that maps states to actions and thus provides a decision maker with an *optimal decision* to take when the controlled system finds itself in a certain state.

Although control and optimization are not the motivation of this work it is interesting to look at the use of MDPs as control and optimization tools. Figure 3.4 shows the MDP's computed policy's position in the decision-taking control loop of an abstract controller. The controller has taken some decision at time t. He then received an information as to what state s_{t+1} the system is in now, at time $t+1$, he then uses the policy $\pi : S \rightarrow A$ to determine the optimal action a_{t+1} to take at this time $t+1$. This will ensure that the system is *controlled in an optimal way*.

3.2.3 Partially Observable Markov Decision Processes

A Partially Observable Markov Decision Process (POMDP) is a further extension of a Markov Decision Processe, the difference being that the decision maker can no longer observe the entire system state, but must instead deal with partial observations when making decisions. It is formally defined as a six-tuple (S, A, O, T, Ω, R) with:

- S: set of states,
- A: set of actions,

- O: set of observations,
- T: conditional transition probabilities,
- Ω: conditional observation probabilities,
- R: rewards.

An observation is any system output that the decision maker can *observe*. MDPs assume that the decision maker has the ability to *see* what state the system is currently in, whereas POMDPs make no such assumption, relying instead on partial observations.

Although the reward function for MDPs and POMDPs remains the same, optimization differs in that the output is no longer a policy that maps *states* to *actions*, but rather a policy that maps *observations* to *actions*. This more realistically models reality where decision makers often do not have the ability to observe the complete state of a system, but rather receive only partial information (example: sensor data). The result of this added observation dimension is that the decision agent must now maintain a constantly changing belief state, that is initially defined and then continuously updated as actions are taken and observations received.

The computation of optimal policies for POMDPs is computationally extremely costly. The traditional policy computation approaches used for MDPs are often intractable for POMDPs and it is thus often necessary to solve approximately. A number of interesting approaches have been studied [Han98].

3.3 Simulink

Simulink is a commercial modelling and simulation tool developed by MathWorks. It is an industry standard in the field of control engineering. Models are created graphically in a block-based user interface and simulation results can be easily be analysis with plots or mode advanced tools. Additionally a large number of internal and third-party libraries further simplify modelling, especially in specialized fields such a music or aerospace.

Simulink is also tightly integrated in MATLAB, MathWorks' commercial numerical computation tool, which is widely used in academia and industry in many different fields. Simulink runs inside the MATLAB environment and can thus easily be controlled, tested or extended using the MATLAB programming language.

Simulink is especially useful for engineers because no programming knowledge is required even for modelling complex dynamic systems (eg. power plants). Default configuration options and an entirely graphical interface hide most implementation details (simulation details, solver types, et). A graphical modelling interface is also useful because it allows a more intuitive understanding of the model, unlike the large mathematical matrices used in Markov Chains, Markov Decision Processes or Partially Observable Markov Decision Processes. Figure 3.5 shows an example of the MATLAB environment, a Simulink model and plots of simulation results.

Because this work deals extensively with Simulink a short introductory example may be helpful. Figure 3.6 shows the pendulum model from section 3.1 (see Figure 3.1) implemented in Simulink through a series of integrators, a sin function and two contant parameters, the gravitational acceleration g and the length of the pendulum l. The initial phase is set to zero and an initial condition can be defined in either of the two integrators. For this simulation the angular velocity was used as an initial condition and set to $1\frac{rad}{s}$. Figure **??** shows two plots of the system response once with a gravitational constant of $9.8\frac{m}{s^2}$ and

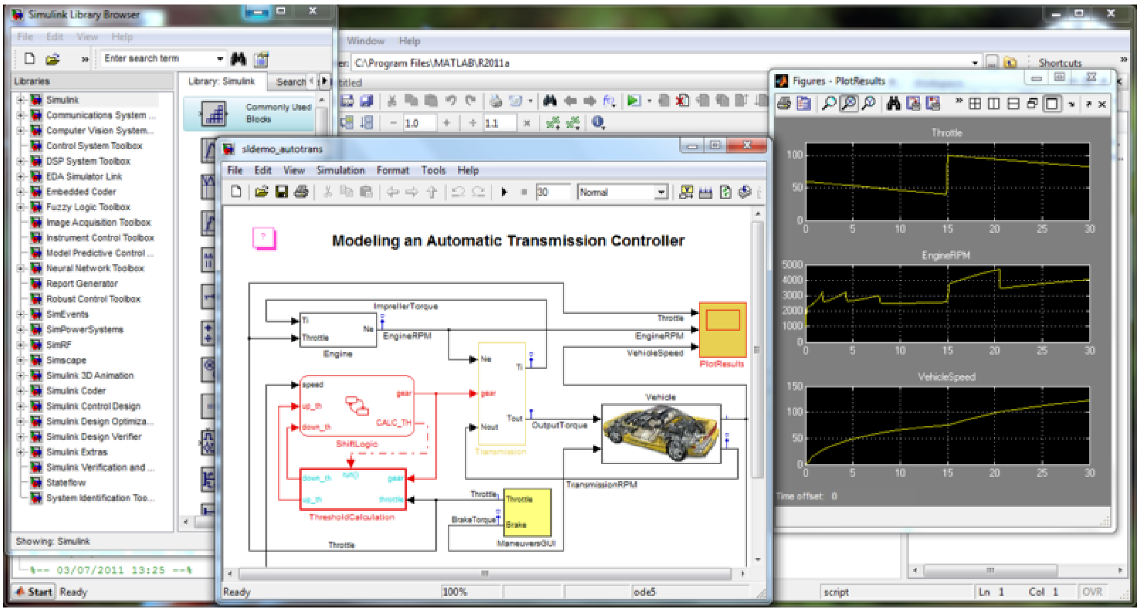

Figure 3.5: *Screenshot of Simulink*

once with a gravitational constant of $19.6\frac{m}{s^2}$.

As can be seen in this example Simulink provides a very intuitive platform for modelling and simulating dynamic systems. Although the above example model is entirely deterministic, it is trivially easy to add randomness to deterministic models in Simulink.

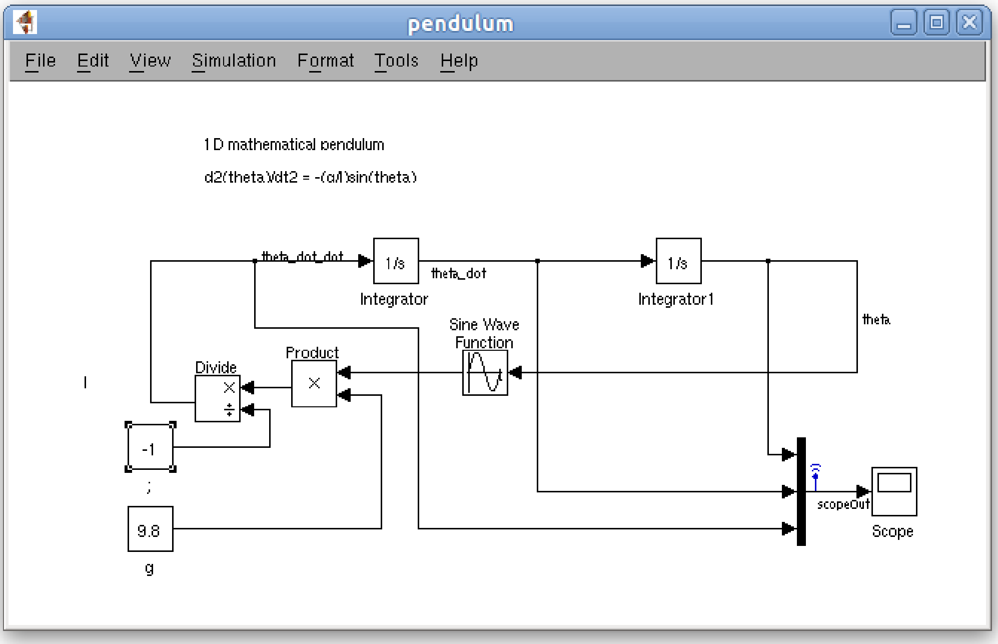

Figure 3.6: *1-dimensional mathematical pendulum model*

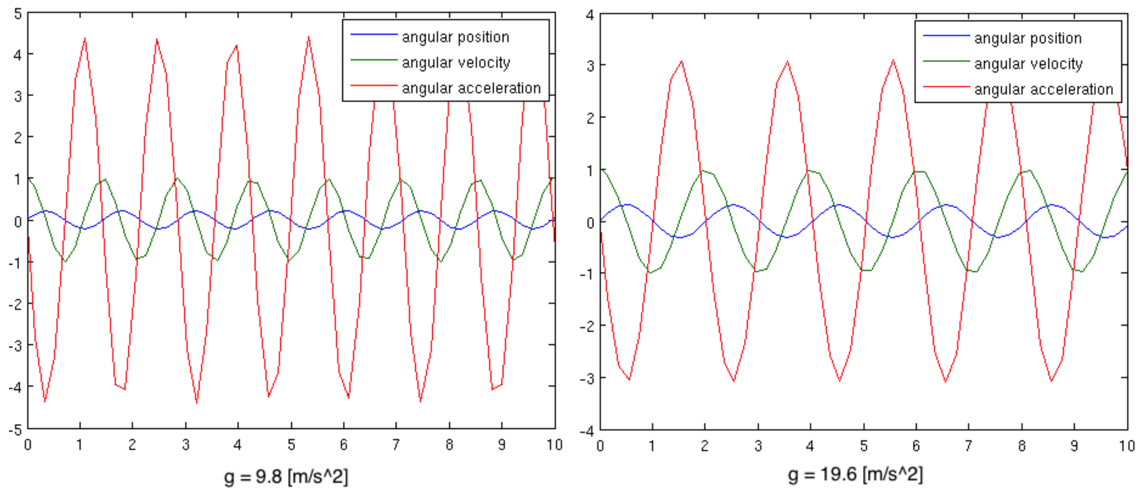

Figure 3.7: *1D mathematical pendulum response*

Methodology

This chapter documents the ideas and approaches behind the implementation of the *extraction algorithm* and the *validation*. It provides a high-level overview of the ideas and the problem/solution tuples that defined the final implementation.

4.1 Extraction

Simulink models represent dynamic systems in a number of different ways. Systems can be described through a graphical representation of differential equations (see example in section 3.3). Systems can also be described by transfer functions or state machines. Simulink offers many different representational forms. Almost all of these representational forms have in common that they represent *rules* of some sort. When asked to simulate these systems Simulink solves these *rules* in real time and produces the system response. MDPs and POMDPs are much simpler construct that do not require real-time solvers because the system dynamics is represented as simple state transitions probabilities. Given an MDP or a POMDP representation of a dynamic system, the simulation thereof is merely a question of random sampling.

In order to build these transition probability matrices the *extractor* must simply simulate and observe the given Simulink model enough times and with enough different inputs to build up this transition probability matrix. In parallel the *extractor* observes the Simulink model's reward and observation outputs and incrementally builds the reward matrix and the conditional observation probability matrix. The following sections go through each of the steps required during this extraction and gives a short high-level overview of the main functions.

4.1.1 Inputs and actions

MDPs and POMDPs model *controllable* dynamic systems, where a decision taker can influence the system's development over time. Simulink models usually have a single or multiple inputs that are sampled in every time-step and used as input for the given system. In order to extract transition probabilities that depend on the action chosen by the decision maker, simulations must be observed for each of the possible actions. This means that for every state the system may find itself in, simulations must be run with every possible action a decision maker may choose to take. In order to guarantee that the MDP or POMDP will be able to represent the dynamics of the system correctly this means that every possible permutation of permitted input values must be used during the extraction.

A simple example of this can be made with the *ideal gas law* system,

$$p \cdot V = n \cdot R \cdot T,$$

where p $[Pa]$ is the pressure, V $[m^3]$ is the volume, n $[mole]$ is the mole quantity, $R = 8.314$ $[J \cdot K^{-1} \cdot mol^{-1}]$ is the universal gas constant and T $[K]$ the temperature. Although this system is not dynamic — it does not change over time — it is sufficient in this context.

If a conversion of this system to an MDP or a POMDP were necessary with the following configuration,

- - inputs: pressure p, volume V, mole quantity n,
- - output: temperature T,

the action set of an MDP or a POMDP would be defined as the cartesian product of all input sets. If the maximum of each input x were defined as x_{max} and it's minimum as x_{min}, the action set would be:

$$
\begin{aligned}
A &= \Pi \times \Lambda \times \Gamma \\
&= \{(p, V, n) \mid p \ \in \Pi \, and \, V \in \Lambda \, and \, n \in \Gamma\}, \\
where & \\
\Pi &= [p_{min}, (p_{min} + \pi), (p_{min} + 2 \cdot \pi), \cdots, (p_{min} + (N-1) \cdot \pi), p_{max}]] \\
\Lambda &= [V_{min}, (V_{min} + \lambda), (V_{min} + 2 \cdot \lambda), \cdots, (V_{min} + (N-1) \cdot \lambda), V_{max}]] \\
\Gamma &= [n_{min}, (n_{min} + \gamma), (n_{min} + 2 \cdot \gamma), \cdots, (n_{min} + (N-1) \cdot \gamma), n_{max}]] \\
\pi &= \frac{p_{max} - p_{min}}{N_p - 1} \\
\lambda &= \frac{V_{max} - V_{min}}{N_V - 1} \\
\gamma &= \frac{n_{max} - n_{min}}{N_n - 1}
\end{aligned}
$$

with N_i being the number of different input values required between the maximum and minimum values of input x_i (see section XXXX discretization). The cardinality of A (ie. the number of actions $a \in A$) is

$$|A| = \prod_{i \in I} N_i.$$

An example action set with numeric values could be

$$A = \begin{pmatrix} (p=0.0, V=0.0, n=0.1) \\ (p=0.0, V=0.0, n=0.3) \\ (p=0.0, V=0.0, n=0.5) \\ \vdots \\ (p=4.3, V=2.0, n=0.9) \\ (p=4.3, V=3.0, n=0.1) \\ (p=4.3, V=3.0, n=0.3) \\ \vdots \\ (p=9.9, V=9.0, n=0.5) \\ (p=9.9, V=9.0, n=0.7) \\ (p=9.9, V=9.0, n=0.9) \end{pmatrix},$$

$$\begin{aligned} where &= \\ p_{min} &= 0.0,\ p_{max} = 9.9,\ \pi = 0.1,\ N_p = 100 \\ V_{min} &= 0.0,\ V_{max} = 9.0,\ \lambda = 1.0,\ N_V = 10 \\ n_{min} &= 0.1,\ n_{max} = 0.9,\ \gamma = 0.2,\ N_n = 5 \end{aligned}$$

In this case the number of actions comes to

$$|A| = \prod_{i \in (p,V,n)} N_i = N_p \cdot N_V \cdot N_n = 100 \cdot 10 \cdot 5 = 5000.$$

4.2 Validation

Talk about necessity to validate extracted MDP. Step response approach.

Implementation

Implementation

5.1 Extractor

This is about the implementation of the extractor.

5.2 Validator

This is about the implementation of the validator.

Results

This is the "Results" chapter!

6.1 Extractor

Talk about extractor output.

6.2 Validator

Talk about the output of the validator.

Conclusion

This is the "Conclusion" chapter!

Outlook

This is the "Outlook" chapter!

Appendix

A.1 Item 1

Previously, this and that has been done and so on.

Bibliography

[Han98] Eric A. Hansen. Solving POMDPs by Searching in Policy Space. In *UAI*, pages 211–219, 1998.

[Put94] Matrin L. Putterman. *Markov Decision Processes, Discrete Stochastic Dynamic Programming*. Wiley-Interscience, Hoboken, New Jersey, 1994.