COMPUTATIONAL MODEL
OF A ROTARY SYSTEM
WITH DISCONTINUOUS ELEMENTS

A dissertation submitted to The University of Manchester
for the degree of Master of Science
in the Faculty of Engineering and Physical Sciences

2009

Rebekah Carter
School of Mathematics
# Contents

Abstract 11

Declaration 13

Copyright 15

Preface and Acknowledgements 17

1 Introduction 19

1.1 Motivation 19

1.2 Discontinuous Dynamical Systems 20

1.2.1 Filippov and Utkin Methods 21

1.3 Hybrid Dynamical Systems 22

1.3.1 Hybrid Automata 24

2 The Discontinuous System 29

2.1 The System 29

2.2 Long-Term Behaviour: The Theory 32

2.3 Long-Term Behaviour: Simulations 34

2.3.1 Convergence to Equilibrium 36

2.3.2 Permanent Stuck Bit 36

2.3.3 Stick-slip Motion 38
2.3.4 Overall Trend of Long-Term Behaviour

3 Hybrid Automaton Models
3.1 Three Hybrid Automata
3.2 Refining the Definitions of the Automata
3.3 Hybrid Automaton for the System with a Sliding-Mode Based Controller

4 Simulation of the Hybrid Automata
4.1 Simulation of Hybrid Systems
4.1.1 Problems Encountered
4.1.2 Possible Solutions
4.1.3 Software Used in this Project
4.2 Simulations of the Hybrid Automata for the Original Problem
4.2.1 Simulation using MATLAB’s Stateflow Toolbox
4.2.2 Simulation using Modelica’s Stategraph library
4.2.3 Comparison: Stateflow vs Modelica
4.3 Simulation of the Hybrid Automaton for the System with the Sliding-Mode Based Controller

5 Some Analytical Aspects of Hybrid Automata
5.1 An Example
5.2 Defining Languages and their Properties
5.3 Languages of our Hybrid Automata

6 Conclusions

Bibliography

A Appendix
A.1 MATLAB Script File for Simulation of the Discontinuous Problem .................................................. 97
A.2 Stateflow Model of 2–Location Hybrid Automaton .......................................................... 100
A.3 Modelica Models for the System .................................................................................. 101
A.4 Model for Sliding-Mode Based Controller Automaton ........................................... 109
List of Figures

1.1 A finite automaton. ........................................... 26
1.2 A hybrid automaton. .......................................... 26

2.1 Mechanical model describing the torsional behaviour of the conventional drillstring. Extracted from [15]. ......................... 30
2.2 A typical discontinuous friction torque graph. .................... 33
2.3 Examples of the three long-term behaviour patterns. .......... 37
2.4 Long-term behaviour for various values of the motor torque, $u$, and the weight on the bit, $W_{ob}$. ......................... 39

3.1 Hybrid automata models for the drillstring, with two and three locations. ........................................... 43
3.2 Directed graph associated with the five location hybrid automaton model for the drillstring. ................................. 45
3.3 Domains for each location for the new 5-location automaton 48
3.4 The new five location hybrid automaton. ...................... 49
3.5 Directed graph of the hybrid automaton model for the system with the sliding-mode based controller. .................. 51

4.1 Stateflow representations of the two, three, and five location automata. To unclutter the picture for 5-locations, long guard conditions have been replaced by [cond_{q4q2}] or similar. .... 62
List of Tables

2.1 Long-term behaviour of the bit for different parameter values. 40
# List of Symbols

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>SI Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_r$</td>
<td>Top-rotary inertia value</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$J_b$</td>
<td>Bit-rotary inertia value</td>
<td>kg m$^2$</td>
</tr>
<tr>
<td>$R_b$</td>
<td>Radius of the bit</td>
<td>m</td>
</tr>
<tr>
<td>$k_t$</td>
<td>Torsional stiffness</td>
<td>N m/rad</td>
</tr>
<tr>
<td>$c_t$</td>
<td>Torsional damping</td>
<td>N m s/rad</td>
</tr>
<tr>
<td>$c_r$</td>
<td>Damping on rotary inertia</td>
<td>N m s/rad</td>
</tr>
<tr>
<td>$c_b$</td>
<td>Damping on bit inertia</td>
<td>N m s/rad</td>
</tr>
<tr>
<td>$\mu_{cb}$</td>
<td>Coulomb friction coefficient</td>
<td>None</td>
</tr>
<tr>
<td>$\mu_{sb}$</td>
<td>Static friction coefficient</td>
<td>None</td>
</tr>
<tr>
<td>$D_v$</td>
<td>Size of transition region</td>
<td>None</td>
</tr>
<tr>
<td>$\gamma_b$</td>
<td>Speed of exponential decay</td>
<td>None</td>
</tr>
<tr>
<td>$\nu_f$</td>
<td>Constant velocity</td>
<td>m/s</td>
</tr>
<tr>
<td>$u$</td>
<td>Motor torque</td>
<td>N m</td>
</tr>
<tr>
<td>$W_{ob}$</td>
<td>Weight on the bit</td>
<td>N</td>
</tr>
</tbody>
</table>
Abstract

Discontinuous dynamical systems have complex behaviour patterns, due to the discontinuous or switching elements in the system. These dynamical behaviours have generally been well investigated within different frameworks. However, there are other ways of looking at these systems which can give valuable insights for less well understood problems. Hybrid dynamical systems is a framework which is useful for many engineering systems, and is investigated in this project. The aim is to model a particular system computationally by means of hybrid automata-based models. Different computer programs and simulation packages are used in order to determine how the hybrid dynamical system interpretation of the problem relates to the original discontinuous system, and to understand the limitations of computational models.
Declaration

No portion of the work referred to in this dissertation has been submitted in support of an application for another degree or qualification of this or any other university or institute of learning.
Copyright

1. Copyright in text of this dissertation rests with the author. Copies (by any process) either in full, or of extracts, may be made only in accordance with instructions given by the author. Details may be obtained from the appropriate Graduate Office. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the author.

2. The ownership of any intellectual property rights which may be described in this dissertation is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

3. Further information on the conditions under which disclosures and exploitation may take place is available from the Head of School of the School of Mathematics.
Rebekah Carter has been studying for a MSc in Mathematics and Computational Science in this academic year (2008–2009). She has developed an interest in aspects of numerical simulation of dynamical systems over her university career, and has taken this further by undertaking this computational project as part of her MSc. She previously attended the University of Oxford, where she studied Mathematics to masters level, obtaining a first class degree. As part of the masters year at Oxford, she wrote a dissertation entitled “Quantum Privacy Amplification”, in which she brought together the latest work on how security can be guaranteed when using a quantum mechanics based cipher for sending and receiving messages.

She would like to thank her supervisor, Eva Navarro-López, for all the support and encouragement given throughout the course of this project, and for the enthusiasm she has imparted for this fascinating area of research.
Chapter 1

Introduction

1.1 Motivation

A large part of any modern day investigation into a dynamical system is computational simulation. This gives information about the behaviour of the system, showing how the system works and what factors affect it. Having this sort of information can be very useful in predicting when unwanted behaviour will happen within a system, especially if catastrophic results may occur.

In this project, a discontinuous dynamical system will be studied, which is known to have catastrophic behaviour under certain conditions. We will firstly understand how and why this behaviour occurs, through the use of computational simulations of the discontinuous system, and will, secondly, study different ways to model this system, and compare their behaviour with the original discontinuous model. Lastly, we will use Computer Science theory to try and explain the behaviour we see.

However, before we move on to looking at the specific problem in hand, we will briefly consider some theory which is needed to understand the contents
1.2 Discontinuous Dynamical Systems

Discontinuous dynamical systems have been well studied, and hence are understood in most cases. These systems can also be called switched systems or non-smooth systems, and are such that the system is generally continuous (or smooth), but there are points where there is a discontinuity (or jump) in some part of the system. For example, there could be a jump in a forcing function of the differential equation, say $\dot{y}(x) = \text{sign}(x)$ in $(a, b)$ with some boundary conditions. Another possible source of discontinuity is in the initial or boundary conditions, as at the corners of the region in

$$\Delta U(x, y) = f(x, y), \quad \text{in} \quad (0, 1) \times (0, 1),$$

with $U = 0$ on $x = 0, 1, y \in [0, 1],$

$U = 1$ on $y = 0, 1, x \in (0, 1).$

The discontinuous system being studied in this project has a discontinuity in the friction torque, which will be discussed in Section 2.2. This is a common source of discontinuity in mechanical systems, since the friction a body experiences is related to its current velocity, such that every time the velocity changes sign, the friction experienced changes sign. However, this change of sign would not, in itself, make the friction discontinuous — the discontinuity arises from the property that the friction acting on a body trying to start moving is greater in magnitude than the friction acting on the body when it is already moving. Indeed, the friction torque at zero velocity is unknown and varies within a closed set.
One specific consequence of this discontinuous friction torque is that we need to have a model for the friction when we are at the discontinuous point. There are various methods that can be used to find a representation at a discontinuity; the two that can be used to obtain the equations for the system studied in this project will be introduced next.

1.2.1 Filippov and Utkin Methods

Formally known as Filippov’s continuation method and Utkin’s equivalent control method [8, 21], these are both ways of producing a function for the vector field of a system of equations when at a discontinuity. We will first give the details for the simpler case of the Filippov method, before discussing the approach of the Utkin method.

If we have a surface of discontinuity, which has the form \( \sigma(x) = 0 \), and the functions which describe the dynamics outside of the discontinuity are

\[
\dot{x} = \begin{cases} 
  f_1(x) & \text{for } \sigma(x) < 0, \\
  f_2(x) & \text{for } \sigma(x) > 0,
\end{cases}
\]

then Filippov’s continuation method says that the vector field \( f_s \) on the surface of discontinuity is a convex combination of the two vector fields in the separated sections of the state-space [8]. More precisely, the vector field is given by

\[
\dot{x}_s = \alpha f_2(x) + (1 - \alpha) f_1(x),
\]

with

\[
\alpha = \frac{\nabla(\sigma) \cdot f_1}{\nabla(\sigma) \cdot (f_1 - f_2)},
\]

\[
1 - \alpha = \frac{\nabla(\sigma) \cdot f_2}{\nabla(\sigma) \cdot (f_2 - f_1)},
\]

with \( \nabla(\sigma) \) denoting the gradient of \( \sigma \). \( \alpha \) is selected in such a way so that \( f_s \) lies on the tangential manifold of \( \sigma = 0 \).
Utkin’s equivalent control method is also called the regularisation method, and may be considered as a physical interpretation of the Filippov method [21]. The general idea is as follows. Consider a discontinuous system of the form

\[ \dot{x} = f(x, u), \quad x \in \mathbb{R}^n, \ u \in \mathbb{R}^m, \]

with

\[ u(x) = \begin{cases} 
    u^+(x) & \text{if } \sigma(x) > 0, \\
    u^-(x) & \text{if } \sigma(x) < 0.
\end{cases} \]

The control \( u \) such that \( \frac{\partial}{\partial x} f(x, u) = 0 \) is the equivalent control \( u_{eq} = u \). Then the dynamics on \( \sigma(x) = 0 \) are governed by \( \dot{x} = f(x, u_{eq}) \). This evolution of the system on the discontinuity surface is referred to as the sliding mode or sliding motion.

The right-hand side \( f(x, u_{eq}) \) resulting from the equivalent control method does not necessarily coincide with that given by the Filippov method. In particular cases, these two methods give a different form for the vector field on this discontinuity surface, which leads to problems with non-uniqueness of the solutions. Although these problems must be addressed if they occur, the issue does not arise for the specific system in this project and will therefore not be considered in any further detail.

### 1.3 Hybrid Dynamical Systems

The study of hybrid dynamical systems is a fairly new area, but the concept has the power to revolutionise the way we look at certain dynamical problems. The basic idea of these systems is that they contain both discrete and continuous elements [22]. Many systems can be reinterpreted in the hybrid systems framework, including those we have just looked at: discon-
tinuous dynamical systems. These are obviously systems which have discrete
and continuous elements, since they are governed by (mainly) continuous
dynamics, but there are also discrete jumps in the forcing functions, or the
initial or boundary conditions.

Some other examples that can be reinterpreted in the hybrid format are given
below, and are taken from [13]:

1. Automated Highway System (AHS). Here, the basic idea is that we
want to control all the cars on a highway automatically, so that we can
maximise the number of cars getting through the system and minimise
the number of accidents that occur. This is a hybrid system since there
is the combination of continuous movement of the cars and continuous
monitoring of the surroundings, plus discrete communication between
the controller and the vehicles.

2. Electrical circuits. These are made up of components which have a
continuous voltage and current flowing through them, but there are
some components, like switches and diodes, which can turn on and off.
These components add discrete changes into the flow in the circuit, and
so these circuits can be thought of as hybrid systems.

3. Computer controlled systems. These systems have a continuous (often
physical) process, which is monitored at discrete points in time by the
computer. The computer then takes actions based on the control proto-
col it is following. This is clearly a hybrid system, since the continuous
system is being monitored and changed at discrete points in time.

An interesting thing to note about the concept of hybrid dynamical systems
is that we can use it to model many different systems which may have been
already studied via a different modelling route. This means that we can
gain a greater understanding of the hybrid modelling process by using this framework to model some systems which have already been studied.

In order to model these hybrid systems mathematically we need to have a way of representing them. As has already been said, the field of hybrid dynamical systems is fairly new, which means that the representation of these systems has not yet been standardised. The representation used often depends on the background of the researcher, with the main research groups involved being computer science, mathematics, and control engineering. Certain problems are more suited to particular representations, so it is not necessarily to be expected that the representations will ever merge completely.

The representation of hybrid systems typically used by control engineers and theorists invokes the usual modelling frameworks for continuous systems. Some differential and algebraic equations are used for this representation, with added reset or transition functions to give changes to the discrete dynamics [4, 5]. We can also approach hybrid dynamical systems from a behavioural point of view [22, 23]; the system’s behaviour is specified, for instance by defining a set of trajectories it may follow. In this project, however, we are not using either of these approaches — we are using the hybrid automaton framework [3, 9].

1.3.1 Hybrid Automata

The hybrid automaton representation models a hybrid dynamical system in terms of a finite automaton combined with a state-space model. A finite automaton is a mathematical model for a finite state machine, and it comes from classical theory of computer science. This approach is, therefore, the
one most often used by computer scientists.

Before we go on to consider the formal definition of a hybrid automaton, we should consider the definition of the finite automaton.

**Definition 1** (Finite Automaton, [10]). A finite automaton is a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where:

- $Q$ is a finite set of discrete states or *locations*.
- $\Sigma$ is a finite set of input symbols or *events*.
- $\delta$ is a *transition function* that takes as arguments a state and an input symbol, and returns a state.
- $q_0$ is an *initial state*.
- $F$ is a set of final or *accepting* states; $F \subset Q$.

An automaton is usually depicted by a graph, with the discrete locations represented by vertices, and the transitions represented by edges. The elements of the set $\Sigma$ can then be seen to be labels for the edges on the graph. An example diagram for a finite automaton is shown in Figure 1.1.

To extend this definition to the hybrid automaton, we consider each of the discrete locations to have an associated continuous dynamics and domain. We also define initial states, and continuous input and output spaces. The outputs are obtained via the discrete and continuous output maps, which are defined at each location.

The transitions from one location to another can either be externally induced (controlled transitions), or internally induced (autonomous transitions). Autonomous transitions occur according to conditions on the *continuous* state of the system. That is, the system starts off in one location and evolves
for a time under the continuous dynamics associated with that location, until the continuous state vector satisfies some guard condition. This enables the system to change to the location which that condition is guarding. On entry to the new location, a reset condition is applied, which re-initialises the continuous state to a specified value — this reset condition could be the identity, so that no changes are required on entry to the new location. After the reset, the system carries on evolving under the dynamics specified by the new location.

**Definition 2** (Hybrid Automaton, \([16, 17]\)). A hybrid automaton with inputs and outputs is a collection

\[ H = (Q, E, X, \Sigma, \mathcal{U}, \mathcal{Y}, Dom, \mathcal{F}, Init, G, R, h, r), \]

where:

- \( Q \) is a finite set of locations.
- \( E \) is a finite set of edges called transitions or events.
- \( X \subseteq \mathbb{R}^n \) is the continuous state-space.
• $\Sigma$ is a finite set of symbols labelling the edges and representing discrete input events.
• $\mathcal{U} \subseteq \mathbb{R}^m$ is the continuous input space.
• $O$ is a finite set of symbols representing the discrete output events.
• $\mathcal{Y} \subseteq \mathbb{R}^m$ is the continuous output space.
• $\text{Dom} : Q \rightarrow 2^X \times \mathcal{U}$ is the location domain. It assigns a set of continuous states and inputs to each discrete location $q_i \in Q$, thus, $\text{Dom}(q_i) \subset X \times \mathcal{U}$.
• $\mathcal{F} = \{ f_{q_i}(x, u) : q_i \in Q \}$ is the collection of vector fields describing the continuous dynamics, such that $f_{q_i} : X \times \mathcal{U} \rightarrow X$. Each $f_{q_i}(x, \cdot)$ is assumed to be Lipschitz continuous on the location domain for $q_i$ in order to ensure that the solution exists and is unique.
• $\text{Init} \subseteq Q \times X$ is a set of initial states.
• $G : E \rightarrow 2^X$ is a guard condition. $G$ assigns to each edge a set of continuous states; this set contains the states which enable transition along that edge.
• $R : E \times X \times \mathcal{U} \rightarrow 2^X$ is a reset map for the continuous states for each edge. It is assumed to be non-empty, so that the dynamics cannot be destroyed, only changed.
• $h : Q \times X \times \mathcal{U} \rightarrow \mathcal{Y}$ is the continuous output mapping; there is one for each location.
• $r : Q \times X \times \Sigma \times \mathcal{U} \rightarrow O$ is the discrete output map; there is one for each location.

These hybrid automata can again be represented by graphs, with the discrete locations represented by vertices, and the continuous dynamical equations written at each location. For an example graph of a hybrid automaton see Figure 1.2. The guard and reset conditions are quite often used to label
the edges, instead of elements from $\Sigma$, since these conditions are the key information needed for transitions. The guard condition to enable us to change from location $q_i$ to location $q_j$ has the form $G(q_i, q_j)$, and the reset condition has the form $R(q_i, q_j, x, u)$, which resets the continuous state vector $x$, using the input $u$, on transition from location $q_i$ to location $q_j$.

We have now looked at the basic background material needed to understand this project, so we will move on to the introduction of the specific system that will be simulated.
Chapter 2

The Discontinuous System

In this project, one of the first aims is to understand the complex dynamical behaviour of a particular discontinuous system. To do this we need to introduce the system being studied, and take a look at the analytical work that has already been undertaken for this system. We will then present the simulations that have been made and their relationship to the theory.

2.1 The System

In this project, the problem for comparison of models will be the torsional behaviour of a conventional vertical oilwell drillstring with two degrees of freedom. This problem has been well studied by Navarro-López and others [15, 16, 17, 18, 19]. These papers propose a set of piecewise-smooth differential equations to model how friction on the bit affects the motion of the drillstring, and present the theory behind the behaviour that occurs.

The drillstring behaviour is modelled as a simple torsional pendulum driven by an electrical motor, and a discontinuous friction model is used for the
Figure 2.1: Mechanical model describing the torsional behaviour of the conventional drillstring. Extracted from [15].

contact between the bit and the rock. The mechanical model describing the system is shown in Figure 2.1, and most of the key problem parameters are indicated in this figure. $T_m$ is the torque supplied by the motor to turn the drillstring, which is a control input (considered as a constant value, $u$, in this study). The other control input used is the weight on the bit, also considered to be a constant, and denoted by $W_{ob}$ (not shown in the figure).

$J_r$ and $J_b$ are the top and bit inertias respectively. These inertias are connected by a spring which represents the drill in between the two inertias; the spring has torsional stiffness $k_t$ and torsional damping $c_t$. The top-rotary system has $\varphi_r$ as the angle of rotation from a fixed axis, and consequently the rotary velocity is denoted $\dot{\varphi}_r$. At the top-rotary system there is also a viscous damping torque due to drilling mud, given by $c_r \dot{\varphi}_r$. Similarly, the
angle of rotation of the bit-rotary system is $\varphi_b$, with a corresponding rotary
velocity $\dot{\varphi}_b$, but here, torque on the bit is given by $T_b(\dot{\varphi}_b) = c_b \dot{\varphi}_b + T_{f_b}(\dot{\varphi}_b)$. Again, $c_b \dot{\varphi}_b$ is the damping torque due
to drilling mud, but the new term, $T_{f_b}$, is the friction torque modelling the bit-rock contact.

The equation for the friction torque is

$$T_{f_b}(\dot{\varphi}_b) = W_{ob} R_b [\mu_{sb} + (\mu_{sb} - \mu_{cb}) \exp \left(-\frac{2\dot{\varphi}_b}{\gamma_b}\right)] \text{sgn}(\dot{\varphi}_b), \quad (2.1)$$

where $R_b > 0$ is the radius of the bit; $\mu_{sb}, \mu_{cb} \in (0, 1)$ are, respectively, the
static and Coulomb friction coefficients for the bit inertia; and $0 < \gamma_b < 1, \nu_f > 0$ are parameters to adjust the rate of exponential decay in the
model. It should be noted here that the static friction torque is defined as $T_{sb} = W_{ob} R_b \mu_{sb}$, and similarly, the Coulomb friction torque is defined as $T_{cb} = W_{ob} R_b \mu_{cb}$. The definition of the sign function in Equation 2.1 is

$$\text{sgn}(\dot{\varphi}_b) = \begin{cases} \dot{\varphi}_b/|\dot{\varphi}_b| & \text{if } \dot{\varphi}_b \neq 0, \\ \in [-1, 1] & \text{if } \dot{\varphi}_b = 0. \end{cases} \quad (2.2)$$

It is obvious from this definition that the sign function is not uniquely defined
when $\dot{\varphi}_b = 0$. We must therefore use the Filippov or Utkin method to derive
a suitable form for the friction torque when we have zero bit velocity. This
was introduced in Section 1.2.1.

From the above representation of the system, the piecewise-smooth equations
follow, but first it is necessary to define the state vector of the drillstring,
which is $\mathbf{x} = (\dot{\varphi}_r, \varphi_r - \varphi_b, \dot{\varphi}_b)^T = (x_1, x_2, x_3)^T$. In this notation, the equations
that describe the drillstring behaviour are:

$$\begin{align*}
\dot{x}_1 &= \frac{1}{J_r}[-(c_t + c_r)x_1 - k_t x_2 + c_t x_3 + u], \\
\dot{x}_2 &= x_1 - x_3, \\
\dot{x}_3 &= \frac{1}{J_b}[c_t x_1 + k_t x_2 - (c_t + c_b)x_3 - T_{f_b}(x_3)].
\end{align*} \quad (2.3)$$
We must also choose an initial condition to apply. In this project, we take it as \( x_0 = 0 \), so that we start from a motionless state in each simulation.

### 2.2 Long-Term Behaviour: The Theory

The system being studied in this project exhibits interesting dynamical behaviour under certain conditions. This behaviour stems from the discontinuity in the friction torque when the rotary velocity of the bit is zero. In order for the bit to start moving there must be enough torque to overcome the static friction, \( T_{sb} \), and then the friction drops exponentially to the value of the Coulomb friction, \( T_{cb} \), when the bit is moving. This gives us a friction graph looking like the one in Figure 2.2. Here, the static friction torque value is about 6599 N m, and the Coulomb friction torque is about 4124 N m. These values have been obtained with \( R_b = 0.155575 \) m, \( W_{ob} = 53018 \) N, \( \mu_{cb} = 0.5 \), and \( \mu_{sb} = 0.8 \).

When the bit has zero velocity, the friction could be anything between negative \( T_{sb} \) and positive \( T_{sb} \). The uncertainty of this value when approaching zero velocity can cause strange behaviour, since we might not be able to immediately overcome the static friction, becoming stuck with zero velocity for a while or permanently. It turns out that it is possible to see three long-term behaviour patterns.

1. Convergence to an equilibrium point, with positive angular velocity.
2. Permanent stuck bit.

When we converge to an equilibrium point the bit-rotary inertia keeps turning at the same rate as the top-rotary inertia, although the drill is permanently
twisted. That is, when the drillstring is in an equilibrium state, $x_1 = x_3 \neq 0$, and $x_2 \neq 0$. In fact, the equations for the equilibrium point have been found analytically, and are given in paper [15]. The equations for a positive velocity equilibrium are: $x_1 = x_3 > 0$ such that $u - (c_r + c_b)x_3 - T^+_b(x_3, W_{ob}) = 0$, and $x_2 = h(x_3, W_{ob}, u)/k_1$ where $h(x_3, W_{ob}, u) = [c_r T^+_b(x_3, W_{ob}) + c_b u]/(c_r + c_b)$. Here, $T^+_b$ is the friction torque defined for $x_3$ positive, so that $\text{sign}(x_3) = 1$.

We obviously need the bit of the drill to be rotating if we are going to drill into the ground, so this is the desired behaviour for the system.

Permanent stuck bit, the second behaviour, is an unwanted outcome. The system trajectories are such that there can be a few (or many) positive velocity oscillations, but eventually we get to a point where $x_3 = 0$ for the rest of time. Obviously no drilling will be done once the bit is permanently stuck, but worse than that, this situation is very harmful in the drillstring operation;
oilwells usually have to be abandoned when the bit is stuck permanently.

The last behaviour, stick-slip motion, is the most harmful of the three [18]. Such trajectories have the velocity of the bit increasing, then decreasing, then zero, and then these three sections repeat over again (although not necessarily for the same lengths of time). This is why the behaviour is referred to as stick-slip — the bit sticks for a while, then it slips (moves) for a while, and then goes back to sticking again, and repeats. This behaviour can go on indefinitely, and is a type of oscillation, which brings about harmful torsional mechanical vibrations. From the dynamical viewpoint, this is a periodic orbit exhibiting *sliding motion*, so called because it is sliding within the discontinuity surface. The discontinuity surface in this example is $x_3 = 0$, where the torque values change without the velocity of the bit changing.

### 2.3 Long-Term Behaviour: Simulations

Having looked at the theoretical long-term behaviour we should see, we will now move on to consider this behaviour via simulation. In order to give accurate results when simulating the discontinuity, the model of the friction torque has a slightly different form from that given originally in Equation 2.1 (p. 31), and this simulation torque is given by [16]:

\[
T_{fb}(x) = \begin{cases} 
T_{eb}(x) & \text{if } |x_3| \leq \delta, \ |T_{eb}| \leq T_{sb} \text{ (stuck)}, \\
T_{sb}\text{sign}(T_{eb}(x)) & \text{if } |x_3| \leq \delta, \ |T_{eb}| > T_{sb} \text{ (stick-to-slip transition)}, \\
f_b(x_3)\text{sign}(x_3) & \text{if } |x_3| > \delta \text{ (sliding)},
\end{cases}
\]

with $\delta > 0$, and $f_b(x_3) = W_{obo}R_b[\mu_{eb} + (\mu_{sb} - \mu_{eb})\exp^{-\gamma_b|\dot{x}_3|/\nu_f}]$. The reaction torque is $T_{eb} = c_t(x_1 - x_3) + k_t x_2 - c_b x_3$, which plays the role of the equivalent control in order to obtain the dynamics on the discontinuity surface, $x_3 = 0$. 

34
This friction torque is a combination of a switch model [12], and the Karnopp model [11]. This combined model introduces a region of “zero velocity”, whereby if the magnitude of the bit velocity is less than a small number, $\delta$, we can either be stuck or in transition. Defining this region makes it easier for simulation software to detect when the velocity is close to zero.

Given this working definition of the friction torque, we can now start making simulations of the discontinuous differential equations to study the system’s behaviour. As mentioned in Section 2.1, there are only two parameters used in this project which are changed between simulations (control parameters), with all other parameters being kept the same for every simulation. These two control parameters are $W_{ob}$, the weight on the bit, and $u$, the motor torque, and are the only ones changed because they have most effect on the long-term behaviour of the system. Throughout this report the fixed parameters are given the values below.

\[
c_t = 172.3067 \text{ N m s/rad}, \quad J_b = 471.9698 \text{ kg m}^2, \quad \mu_{cb} = 0.5, \\
c_r = 425 \text{ N m s/rad}, \quad J_r = 2122 \text{ kg m}^2, \quad \mu_{sb} = 0.8, \\
c_b = 50 \text{ N m s/rad}, \quad R_b = 0.155575 \text{ m}, \quad \gamma_b = 0.9, \\
k_t = 861.5336 \text{ N m/rad}, \quad \nu_f = 1 \text{ rad/s}, \quad \delta = 10^{-6}.
\]

With these parameters fixed, we can simulate for different values of $W_{ob}$ and $u$ to find the long-term behaviour of the velocity of the bit. The simulations we have made use MATLAB script and function files, which are attached for reference in Appendix A.1. In Section 2.2 we discussed the three possible long-term behaviour patterns, which are convergence to an equilibrium point (with positive velocity), permanent stuck bit, and stick-slip motion. Before we go on to looking at the overall pattern of long-term behaviour, it is wise
to take a closer look at one simulation from each category to understand exactly what each behaviour entails.

2.3.1 Convergence to Equilibrium

An example which converges to the positive velocity equilibrium occurs when we set $u = 6000$ N m and $W_{ob} = 50000$ N. The simulation then gives us the top diagram in Figure 2.3. The left hand plot shows the velocities of the top and bit inertias as time progresses. We can see that both of the velocities tend to an equilibrium value, which is about $4.34$ rad/s in this case. This means that the whole drill is rotating at this velocity, and therefore it is actually drilling into the ground. This is the desired behaviour, as mentioned in Section 2.2, since we want to drill through the rock.

The plot on the right-hand side is the state-space trajectory for this particular case. The trajectory starts off at the origin (since this is the initial condition), and spirals round towards the equilibrium point, where it then remains for the rest of time. An interesting thing to note about this trajectory is that the equilibrium point has a positive value for $x_2 = \varphi_r - \varphi_b$, which means that the drillstring is permanently twisted whilst drilling. This is an inevitable side effect of the system, and we must therefore design the drillstring to cope with this twisting effect.

2.3.2 Permanent Stuck Bit

The second long-term behaviour is to have a permanently stuck bit. The middle diagram in Figure 2.3 is an example of this behaviour, and occurs when we have $u = 6000$ N m and $W_{ob} = 59208$ N. In this case the bit
Figure 2.3: Examples of the three long-term behaviours, for $u = 6000$ N.m. Top: Convergence to equilibrium behaviour for $W_{ob} = 50000$ N. Middle: Permanent stuck behaviour for $W_{ob} = 59208$ N. Bottom: Stick-slip behaviour for $W_{ob} = 53018$ N. In each case, the left plot shows the rotary velocities of the top and bit inertias, and the right plot shows the state-space trajectory.
goes through four oscillations from zero to positive velocity, but then it stops permanently at about $t = 22$ s. The top inertia ends up oscillating between positive and negative velocity, due to the drillstring being twisted and the motor torque being applied. The top does eventually stop moving, to all intents and purposes, and from that time onwards the drillstring is stationary. However, the state-space trajectory shows that it is still twisted when it stops moving, since there is a (non-standard) equilibrium point with $x_1 = x_3 = 0$ rad/s and $x_2 \approx 7$ rad. This non-standard equilibrium belongs to the surface of discontinuity.

This behaviour is not very useful, since no drilling is performed once the drill has stopped, which is an undesirable long-term outcome.

### 2.3.3 Stick-slip Motion

Stick-slip motion is dangerous for the drillstring, since it is an oscillation and can set up torsional mechanical vibrations. An example of stick-slip motion is given when $u = 6000$ N m and $W_{ob} = 53018$ N; the plots for these values are shown in Figure 2.3, at the bottom. The left-hand plot shows the oscillatory motion very clearly — the velocity of the bit oscillates between zero and positive velocity, and does not ever settle down to an equilibrium value. Instead this motion tends to a periodic orbit, as demonstrated by the state-space plot.

### 2.3.4 Overall Trend of Long-Term Behaviour

We have now seen what the three different behaviour patterns look like, and so we will consider which parameter values cause each type of behaviour.
Figure 2.4: Long-term behaviour for various values of the motor torque, $u$, and the weight on the bit, $W_{ob}$.

The parameter values have been varied, simulations have been run, and the long-term behaviour outcome has been noted for each. Table 2.1 shows the results that have been obtained for the respective parameters used, and a plot of these results is in Figure 2.4.

This figure shows a clear trend, where ‘high’ values of $W_{ob}$ cause stuck behaviour, ‘medium’ values cause the stick-slip behaviour, and ‘low’ values cause the positive velocity behaviour. However, the bifurcation points, at which we change between the types of long-term behaviour, vary with the value of $u$. The exact equation of the curve of bifurcation points is not an obvious one from looking at the graph, but we can say that it looks more complex than a line. The analysis of the bifurcations encountered in the system is outside the scope of this project.
### Table 2.1: Long-term behaviour of the bit for different parameter values.

<table>
<thead>
<tr>
<th>$u$ (N m)</th>
<th>$W_{ob}$ (N)</th>
<th>Behaviour</th>
<th>$u$ (N m)</th>
<th>$W_{ob}$ (N)</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>3500</td>
<td>29000</td>
<td>Positive vel.</td>
<td>5000</td>
<td>38000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3500</td>
<td>30000</td>
<td>Positive vel.</td>
<td>5000</td>
<td>40000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3500</td>
<td>30500</td>
<td>Positive vel.</td>
<td>5000</td>
<td>42500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>3500</td>
<td>30100</td>
<td>Stick-slip</td>
<td>5000</td>
<td>45000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>3500</td>
<td>30250</td>
<td>Stick-slip</td>
<td>5000</td>
<td>46000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>3500</td>
<td>30500</td>
<td>Stick-slip</td>
<td>5000</td>
<td>47500</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>3500</td>
<td>31000</td>
<td>Stick-slip</td>
<td>5000</td>
<td>50000</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>3500</td>
<td>32500</td>
<td>Perm. stuck</td>
<td>5500</td>
<td>42500</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3500</td>
<td>35000</td>
<td>Perm. stuck</td>
<td>5500</td>
<td>45000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3900</td>
<td>30000</td>
<td>Positive vel.</td>
<td>5500</td>
<td>47500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>3900</td>
<td>30500</td>
<td>Positive vel.</td>
<td>5500</td>
<td>50000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>3900</td>
<td>32500</td>
<td>Positive vel.</td>
<td>5500</td>
<td>52500</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>3900</td>
<td>34500</td>
<td>Stick-slip</td>
<td>5500</td>
<td>55000</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>3900</td>
<td>35000</td>
<td>Stick-slip</td>
<td>6000</td>
<td>48000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3900</td>
<td>36000</td>
<td>Perm. stuck</td>
<td>6000</td>
<td>50000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>3900</td>
<td>37500</td>
<td>Perm. stuck</td>
<td>6000</td>
<td>52000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>30000</td>
<td>Positive vel.</td>
<td>6000</td>
<td>53018</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>35000</td>
<td>Positive vel.</td>
<td>6000</td>
<td>55000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>35500</td>
<td>Positive vel.</td>
<td>6000</td>
<td>57577</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4300</td>
<td>35625</td>
<td>Positive vel.</td>
<td>6000</td>
<td>59208</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4300</td>
<td>35700</td>
<td>Positive vel.</td>
<td>6500</td>
<td>52500</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>4300</td>
<td>35750</td>
<td>Stick-slip</td>
<td>6500</td>
<td>55000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>4300</td>
<td>36000</td>
<td>Stick-slip</td>
<td>6500</td>
<td>57500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>36500</td>
<td>Stick-slip</td>
<td>6500</td>
<td>60000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>37000</td>
<td>Stick-slip</td>
<td>6500</td>
<td>62500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4300</td>
<td>37500</td>
<td>Stick-slip</td>
<td>6500</td>
<td>65000</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4300</td>
<td>40000</td>
<td>Perm. stuck</td>
<td>6500</td>
<td>67500</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4300</td>
<td>41000</td>
<td>Perm. stuck</td>
<td>7000</td>
<td>60000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>4600</td>
<td>36000</td>
<td>Positive vel.</td>
<td>7000</td>
<td>61000</td>
<td>Positive vel.</td>
</tr>
<tr>
<td>4600</td>
<td>37500</td>
<td>Positive vel.</td>
<td>7000</td>
<td>62500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4600</td>
<td>39000</td>
<td>Stick-slip</td>
<td>7000</td>
<td>65000</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4600</td>
<td>40000</td>
<td>Stick-slip</td>
<td>7000</td>
<td>67500</td>
<td>Stick-slip</td>
</tr>
<tr>
<td>4600</td>
<td>42500</td>
<td>Stick-slip</td>
<td>7000</td>
<td>68500</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4600</td>
<td>45000</td>
<td>Perm. stuck</td>
<td>7000</td>
<td>70000</td>
<td>Perm. stuck</td>
</tr>
<tr>
<td>4600</td>
<td>46000</td>
<td>Perm. stuck</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Chapter 3

Hybrid Automaton Models

We will now start using the hybrid automaton approach to model the discontinuous system. In this chapter we will present three hybrid automata models for the basic problem as given in Section 2.1, which are taken from the paper [16]. Then, we will look at the suitability of these automata, and rectify any problems that exist in their definitions. Lastly, we will give a controlled automaton for the drillstring problem, taken from [17], which is designed to stop unwanted behaviour from occurring in the majority of cases.

3.1 Three Hybrid Automata

The first automaton we will be using is based on the original model for the friction torque, Equation 2.1 (p. 31), with the Filippov model when at zero bit velocity. This Filippov method gives us dynamics on $x_3 = 0$ defined by

\[
\dot{x} = f_s(x, u) = \begin{pmatrix}
\frac{1}{J_r}[-(c_t + c_r)x_1 - k_t x_2 + u] \\
x_1 \\
0
\end{pmatrix}.
\]

41
This automaton has two locations, one of which is the stuck location, which is active when we have zero bit velocity, and the other is the slip location, active when the bit velocity is non-zero. The directed graph associated with this automaton is shown on the left in Figure 3.1. We can see that the vertices of the graph contain the name of the location, the dynamics that are used when in that location, and the domain in which it is valid. The edges show the guard and reset conditions between the two locations. In order to take a closer look at these guards and resets, we should note here that the equivalent control, \( u_{eq} \), is equal to the reaction torque, \( T_{eb} \), in this model — it was established in the last chapter that
\[
T_{eb}(x) = u_{eq}(x) = c_t(x_1 - x_3) + k_t x_2 - c_b x_3.
\]

Now, the condition for us to enter the stuck location is for the bit velocity to be zero, and also for the magnitude of the equivalent control, \( u_{eq} \), to be less than or equal to the static friction torque, \( T_{sb} \). This makes sense if we think about the discontinuous version of the system, since we said that if the velocity is zero then the friction we must overcome is the static friction. Hence, if we have zero velocity and a reaction torque greater than the static friction, we will not stick, but carry on slipping. So we only want to enter the stuck location if we have zero velocity on the bit, and magnitude of the reaction torque less than or equal to the static friction torque.

The guard condition to leave the stuck location is to still have zero velocity on the bit, since the velocity will not change whilst we are in the stuck location, but we also want the magnitude of the reaction torque to be greater than the static friction torque, so that we have overcome the friction keeping the bit from moving. This is again fairly self explanatory, and we can see that if the friction has not been overcome, that is, if the magnitude of the reaction torque is less than or equal to the static friction, then we will remain in the
stuck location. We can therefore see the justification behind this automaton model.

The second hybrid automaton that will be studied in this project is the three location automaton for the drillstring, shown on the right of Figure 3.1. This model has three locations, which are the stuck location, the positive velocity slip location, and the negative velocity slip location. So the only difference between this automaton and the last one is that we have split up the slip location into two separate locations. The vector fields which define the dynamics within these two slip locations are found by simply replacing the occurrence of $|x_3|$ in the friction torque by either $x_3$ for the positive slip location, or $-x_3$ for the negative slip location.

We can again see that the guard conditions are fairly self explanatory. To enter the stick location we need zero velocity on the bit and magnitude of the reaction torque less than the static friction torque. However, when we try to leave this location there are two places we can go to. This simply means that we split up the guard condition so that a reaction torque greater
than the static friction sends us to the positive slip location, and a negative reaction torque with magnitude greater than the static friction sends us to the negative velocity location.

In this three location automaton there are also the transitions between the two slip locations to be considered. If we are in the positive velocity location, and get to zero velocity with a negative reaction torque which has magnitude greater than the static friction, we will transfer to the negative velocity location. The transition from the negative velocity location to the positive one is similar. These transitions make sense given the original transitions out of the stick location to these slip locations.

The final automaton that we will study in this project is the five location automaton, which has the same model for the friction torque as in the discontinuous simulation model in Section 2.3, making it (in theory) easier for simulation software. As a reminder, the friction torque used is,

\[
T_f(x) = \begin{cases} 
T_e(x) & \text{if } |x_3| \leq \delta, \ |T_e| \leq T_s \text{ (stuck)}, \\
T_s\text{sign}(T_e(x)) & \text{if } |x_3| \leq \delta, \ |T_e| > T_s \text{ (stick-to-slip transition)}, \\
f_b(x_3)\text{sign}(x_3) & \text{if } |x_3| > \delta \text{ (sliding)}, 
\end{cases}
\]

with \( \delta > 0 \), where \( f_b \) and \( T_e \) are as given in Section 2.3.

The directed graph representing the automaton is given in Figure 3.2. The different friction torque means that we require some intermediate locations between stick and slip. These locations are called transition locations, and we must go through them to get to the slip locations in this model. Since we have five locations, we must have more exacting guard conditions, and this results in conditions on the rotary acceleration of the bit (that is, \( \dot{x}_3 \)).

The guard conditions for entering and leaving the stick location are just as
they were in the three location model, only when we leave stick we go to trans locations instead of slip locations. The guard conditions for leaving a transition location only depend on the velocity and acceleration of the bit, in the sense that if we enter the trans\(^+\) location, for example, we only leave to go to the slip\(^+\) location if we still have either positive acceleration or positive velocity when we get to the edge of the transition region. These guard conditions are in order to avoid non-determinism problems that have been highlighted in the literature (see [16] for details). However, it is not clear that they follow from the definition of the friction torque used here.

### 3.2 Refining the Definitions of the Automata

Having looked at all three of these proposed automata, we should check them for consistency before moving on to simulate them. We will first consider the
two and three location automata together since they correspond to each other very well.

The first thing we notice about these two automata is that all the guard conditions have the proviso that \( x_3 = 0 \). This is a perfectly valid guard condition, but we notice that when we enter any of the slip locations, our domain is assumed to be a region where \( x_3 \) is not zero. This gives us a contradiction whenever we enter a slip location which, theoretically, should \( kill \) the automaton; that is, the automaton should stop without any output. However, this is not what the physical system does, and we do not want this to happen, so we need to refine the domain for the slip locations. To do this, we will simply add the parts which are needed to the current domains.

Considering the 2-location automaton, we see that the domain for the slip location needs to be changed to: \( [x_3 \neq 0] \) or \( [(x_3 = 0) \text{ and } (|u_{eq}| > T_{sb})] \). It is briefly worth pointing out that, with this new slip domain, if we leave the domain for the stick location as it is, we will have intersecting location domains. There is some argument as to whether this is allowed or not, but our view is that as long as the guard conditions are defined well enough, no trajectories will remain in the stick location unless they are supposed to be there. This makes it a fairly irrelevant matter either way.

Doing the same for the 3-location automaton, we decide that the domains should become:

\[
\begin{align*}
\text{Dom}(\text{slip}^+) &= [x_3 > 0] \text{ or } [(x_3 = 0) \text{ and } (u_{eq} > T_{sb})], \\
\text{Dom}(\text{slip}^-) &= [x_3 < 0] \text{ or } [(x_3 = 0) \text{ and } (u_{eq} < -T_{sb})].
\end{align*}
\]

Having done this, we conclude that these two automata are now theoretically consistent, and we will use these new domains in the models from now on.
We now need to look at the 5-location hybrid automaton. As mentioned at the end of the last section, there is some doubt about whether the domains and guard conditions for the trans locations actually follow from the definition of the friction torque. Because of this, we decided to start from the beginning and derive an automaton from the friction torque as it is given on page 44.

The first thing to do is to define the domains for the locations. These follow directly from the definition of the torque, and we will give them the same names as in the original 5-location automaton, since they do serve the same purposes. These domains (along with their friction torque values) will be:

\[
\begin{align*}
\text{Dom}(\text{stick}) &= \{ |x_3| \leq \delta \} \text{ and } \{ |u_{eq}| \leq T_{sb} \} \quad \text{with } T_f(x) = T_e(x), \\
\text{Dom}(\text{trans}^+) &= \{ |x_3| \leq \delta \} \text{ and } \{ u_{eq} > T_{sb} \} \quad \text{with } T_f(x) = T_{sb}, \\
\text{Dom}(\text{trans}^-) &= \{ |x_3| \leq \delta \} \text{ and } \{ u_{eq} < -T_{sb} \} \quad \text{with } T_f(x) = -T_{sb}, \quad (3.1) \\
\text{Dom}(\text{slip}^+) &= \{ x_3 > \delta \} \quad \text{with } T_f(x) = f_b(x_3), \\
\text{Dom}(\text{slip}^-) &= \{ x_3 < -\delta \} \quad \text{with } T_f(x) = -f_b(x_3).
\end{align*}
\]

We now need to consider the transitions that can take place, and what the guards should be. Figure 3.3 shows us how these domains look on a graph of \( x_3 \) against \( u_{eq} \), and gives us an idea of the transitions that can be made, since we want transitions to cross boundaries in a continuous way. Looking at which regions are next to each other, we see that there are 16 possible transitions. We will firstly consider the transitions that can happen to go out of the stick location.

When we are in the stick location, the equation of motion for \( x_3 \) is \( \dot{x}_3 = 0 \). This means that once we enter the stick region, the value of \( x_3 \) does not change, so we cannot move into either of the slip locations. Hence we see that
the possible transitions are to trans$^+$ and trans$^-$, with the guard conditions for these locations given by $u_{eq} > T_{sb}$ and $u_{eq} < -T_{sb}$ respectively.

The next location we will consider is trans$^+$, where the possible transitions are to stick, slip$^+$, and slip$^-$. The equation defining $x_3$ in trans$^+$ is $\dot{x}_3 = \frac{1}{J_b}(u_{eq} - T_{sb})$. However, the domain of the location is $u_{eq} > T_{sb}$, and so $\dot{x}_3 > 0$. Hence, $x_3$ only increases in trans$^+$, which means we can definitely not reach slip$^-$ from here, as this needs $x_3 < -\delta$. So, the valid transitions are to slip$^+$ with guard $x_3 > \delta$, and stick with guard $|x_3| \leq \delta$ and $|u_{eq}| \leq T_{sb}$.

The transitions for trans$^-$ are obtained in a similar way.

We will lastly consider transitions out of slip$^+$ (slip$^-$ is similar). When we are in slip$^+$ we have $x_3 > \delta$, and if we are to reach any other region we need to get to a point where $x_3 \leq \delta$. This means that $x_3$ must be decreasing leading up to the transition, and so $\dot{x}_3 < 0$. But, when we are in slip$^+$ we
Figure 3.4: Directed graph associated with the new model for the five location hybrid automaton model for the drillstring.

have $\dot{x}_3 = \frac{1}{J_b} (u_{eq} - f_b)$. This implies that (when about to transition out of slip$^+$):

$$\frac{1}{J_b} (u_{eq} - f_b) < 0,$$

$$\Rightarrow u_{eq} < f_b = T_{cb} + (T_{sb} - T_{cb}) \exp \frac{-\gamma_b}{\nu_f} x_3,$$

$$< T_{cb} + (T_{sb} - T_{cb}) \cdot 1,$$  \hspace{1cm} (since $x_3 > \delta > 0$)

$$< T_{sb}.$$

Therefore, if we change out of slip$^+$ we must go to the stick location, because $|u_{eq}| \leq T_{sb}$ when $|x_3| \leq \delta$.

Putting all these transitions and guards together, we see that the hybrid automaton now looks like that in Figure 3.4. The interesting thing to note about this is that it has the same layout as the original automaton — the only differences are the domains of the trans locations, and the guards which leave these locations. We have added reset conditions as in the original 5-
location automaton, due to the fact that when we enter the “zero velocity” location (stick) we will not necessarily be at $x_3 = 0$, so we reset $x_3$ to this value on entry to stick. Also, the vector fields at each location are as before, with the friction torque as given in Equation 3.1.

### 3.3 Hybrid Automaton for the System with a Sliding-Mode Based Controller

Having looked at the original drillstring system, we note that although all of the hybrid automaton models we have been looking at are very interesting, they do not tell us anything about how to stop the unwanted behaviour from occurring. There is a hybrid automaton model in existence which takes this issue into account, and is a model of a control system for the drillstring. This means that under certain conditions, it will force the trajectory of the system to go to the desired long-term behaviour, by controlling the motor torque. We will not justify the hybrid automaton here, but we will simply present it and explain its features. See [15, 17, 19] for more details of this automaton.

The basic idea behind this hybrid automaton is that we create a *sliding surface* which will, under most conditions, force the solution to go to the positive velocity long-term behaviour. This surface is attractive, so all trajectories will eventually get to it, and once on it they will not leave it again. The switching surface is defined as [19] $S^r := \{ x \in \mathbb{R}^5 : s^r(x, t) = 0 \}$, where,

$$s^r(x, t) = (x_1 - \Omega) + \lambda \int_0^t [x_1(\tau) - \Omega] d\tau + \lambda \int_0^t [x_1(\tau) - x_3(\tau)] d\tau.$$  

Here, $\lambda > 0$ is a parameter which affects the velocities for which stability is achieved, and $\Omega$ is the desired rotary velocity of the drilling system. The
control input that makes the surface $s^r(x, t) = 0$ attractive is [19]:

$$u = c_t(x_1 - x_3) + k_t x_2 + c_r x_1 - J_r [\lambda(x_1 - \Omega) + \lambda(x_1 - x_3) + \eta \text{sign}(s^r)],$$

where $\eta > 0$ is a parameter which affects the speed of the controlled convergence to the equilibrium. Higher values of $\eta$ mean a faster convergence, but also imply more effort in implementing the control.

We should note that two new continuous states have been introduced into the problem for modelling purposes, hence the $\mathbb{R}^5$ above. These two states are defined by $\dot{x}_4 = x_1 - \Omega$, and $\dot{x}_5 = x_1 - x_3$, with initial conditions usually taken to be zero.
Now, all of the above leads us onto the definition of the hybrid automaton. In the 3-location hybrid automaton the state-space was (originally) split up into positive bit velocity states, negative bit velocity states, and zero bit velocity states, and this is the idea behind how we split up the space in this automaton. We still have the \( s^b = x_3 \) state-space split up like this, but we also split up the \( s^r \) space in this way. Putting these together we get nine distinct regions of the state-space, and so we assign a single automaton location to each region. The location definitions along with their domains are:

\[
\begin{align*}
q_1 &= \{ \text{slip}^+_b, \text{slip}^+_r \}, & \text{Dom}(q_1) &= \{ x \in \mathbb{R}^5 : s^b(x) > 0, s^r(x, t) > 0 \}; \\
q_2 &= \{ \text{slip}^+_b, \text{slip}^-_r \}, & \text{Dom}(q_2) &= \{ x \in \mathbb{R}^5 : s^b(x) > 0, s^r(x, t) < 0 \}; \\
q_3 &= \{ \text{slip}^-_b, \text{slip}^+_r \}, & \text{Dom}(q_3) &= \{ x \in \mathbb{R}^5 : s^b(x) < 0, s^r(x, t) > 0 \}; \\
q_4 &= \{ \text{slip}^-_b, \text{slip}^-_r \}, & \text{Dom}(q_4) &= \{ x \in \mathbb{R}^5 : s^b(x) < 0, s^r(x, t) < 0 \}; \\
q_5 &= \{ \text{slip}^+_b, \text{stick}_r \}, & \text{Dom}(q_5) &= \{ x \in \mathbb{R}^5 : s^b(x) > 0, s^r(x, t) = 0 \}; \\
q_6 &= \{ \text{stick}_b, \text{stick}_r \}, & \text{Dom}(q_6) &= \{ x \in \mathbb{R}^5 : s^b(x) = 0, s^r(x, t) = 0 \}; \\
q_7 &= \{ \text{slip}^-_b, \text{stick}_r \}, & \text{Dom}(q_7) &= \{ x \in \mathbb{R}^5 : s^b(x) < 0, s^r(x, t) = 0 \}; \\
q_8 &= \{ \text{stick}_b, \text{slip}^+_r \}, & \text{Dom}(q_8) &= \{ x \in \mathbb{R}^5 : s^b(x) = 0, s^r(x, t) > 0 \}; \\
q_9 &= \{ \text{stick}_b, \text{slip}^-_r \}, & \text{Dom}(q_9) &= \{ x \in \mathbb{R}^5 : s^b(x) = 0, s^r(x, t) < 0 \}.
\]

The directed graph associated with this automaton is given in Figure 3.5. Some physical constraints of the system are considered to reduce the number of feasible transitions (see [17] for more details). The guard conditions are either given explicitly, or are in terms of three symbols \( G^+, G^-, \) and \( G^0 \). These are defined by \( G^+ := \{ x_3 = 0 \text{ and } u_{eq}(x) > T_{sb} \} \), \( G^- := \{ x_3 = 0 \text{ and } u_{eq}(x) < -T_{sb} \} \), \( G^0 := \{ x_3 = 0 \text{ and } |u_{eq}(x)| \leq T_{sb} \} \), with \( u_{eq}(x) = T_{eb}(x) \) as defined previously.
There is a different vector field describing the dynamics at each location, and the equations for these are given below. Note that \( \varphi_1(x) = -\lambda(x_1 - \Omega) - \lambda(x_1 - x_3) \), and \( \varphi_2(x) = \frac{1}{J_b}[c_1x_1 + k_1x_2 - (c_1 + c_b)x_3] \).

\[
\begin{align*}
\mathbf{f}_{q_1} &= \begin{pmatrix} \varphi_1(x) - \eta \\ x_1 - x_3 \end{pmatrix}, & \mathbf{f}_{q_2} &= \begin{pmatrix} \varphi_1(x) + \eta \\ x_1 - x_3 \end{pmatrix}, \\
\mathbf{f}_{q_3} &= \begin{pmatrix} \varphi_1(x) - \eta \\ x_1 - x_3 \end{pmatrix}, & \mathbf{f}_{q_4} &= \begin{pmatrix} \varphi_1(x) + \eta \\ x_1 - x_3 \end{pmatrix}, \\
\mathbf{f}_{q_5} &= \begin{pmatrix} \varphi_1(x) \\ x_1 - x_3 \end{pmatrix}, & \mathbf{f}_{q_6} &= \begin{pmatrix} -2\lambda x_1 + \lambda \Omega \\ x_1 \end{pmatrix}, \\
\mathbf{f}_{q_7} &= \begin{pmatrix} \varphi_1(x) \\ x_1 - x_3 \end{pmatrix}, & \mathbf{f}_{q_8} &= \begin{pmatrix} -2\lambda x_1 + \lambda \Omega - \eta \\ x_1 \end{pmatrix}.
\end{align*}
\]
\[ f_{q_0} = \begin{pmatrix} -2\lambda x_1 + \lambda \Omega + \eta \\ x_1 \\ 0 \\ x_1 - \Omega \\ x_1 \end{pmatrix}, \]

with \( T^+_f(x_3) \) and \( T^-_f(x_3) \) defined by taking the friction torque for \( x_3 > 0 \) and \( x_3 < 0 \) respectively.

The theory says that we will end up with the desired behaviour, positive velocity, as long as the desired velocity, \( \Omega \), is large enough (see [19] for more information). Otherwise, we will have stick-slip motion between \( q_5 \) and \( q_6 \), because the permanent stuck bit behaviour is eliminated by the control.
Chapter 4

Simulation of the Hybrid Automata

One of the main aims for this project is to look at the different models for the drillstring that have been obtained by Navarro-López, and simulate their behaviour, in order to compare them against each other. In the first section we discuss the numerical problems and software solutions that arise when simulating hybrid systems. Subsequently, the behaviour of the three hybrid automata which attempt to model the system will be simulated, and the results obtained will be compared with the results of the original discontinuous system model. We will then simulate the hybrid automaton for the system with the sliding-mode controller to get an idea of its behaviour.

4.1 Simulation of Hybrid Systems

Before we go on to simulate the hybrid automata that have been proposed, we should consider how the simulations will take place. There are many different computational packages and programs which have been designed to
make the simulation of systems easier, and these programs, used correctly, are also likely to be efficient. In this section the problems that can be met when simulating hybrid dynamical systems will be discussed, and possible software solutions proposed. There will then be an explanation of the choice of software used in this project, with discussion of the particular problems involved with each of the simulation solutions.

4.1.1 Problems Encountered

Due to the need for discretisation, there can be problems with numerical simulation of any type of continuous dynamical system, since we must choose appropriate step sizes and tolerances for our algorithm. For slowly-varying dynamics, this does not usually make too much difference to the overall behaviour of the problem. However, for problems whose dynamics vary rapidly, we could end up completely missing a big variation in the solution if the step size is taken to be too large. It is important, therefore, that the user has some idea of the expected solution in order to be able to spot when a simulation is not accurate.

When making simulations of dynamical systems which are governed by differential equations, an important decision for the user to make is the choice of the algorithm used to calculate the solution. If the solution can vary quickly we need to use a variable step solver, rather than a fixed step one, so that the solver can decrease the length of the time steps when at points of rapid change. These solvers compare the error at the current time step to the given tolerance. If the current error is larger than the tolerance, the solver must go back to the previous time step and use a smaller step size. This is repeated until the error is small enough, at which point the solver can start at this
new time step and take another step. The process of finding and using small enough time steps can be costly, so it is essential to balance the required accuracy of the solution with the time the solver takes to find the solution.

The last problem that will be discussed here is specific to the simulation of discontinuous systems, and is the issue of needing to be able to precisely detect where discontinuities occur. This problem is generally called zero-crossing detection, and is a major research area for numerical software manufacturers, among others (see, for example, [7, 20, 24]). To illustrate this issue, let us look at the simple discontinuous problem

\[
\dot{x} = \begin{cases} 
1 & \text{for } 0 \leq x < 1, \\
-1 & \text{for } x \geq 1,
\end{cases}
\]  
(4.1)

with \( x(0) = 0 \).

This problem has the gradient equal to 1 at first, but then it changes to \(-1\) when we get to \( x = 1 \) (at time \( t = 1 \)). If we try to simulate this with no specific algorithm in place to detect when we cross the boundary \( x = 1 \), we will tend to overshoot, and have a permanent error after the discontinuity, unless we use very small tolerances.

There are, in fact, two issues associated with zero-crossing detection. The first is event detection, which is the problem of detecting that an discontinuity occurs. The second is event location, the process of accurately working out at what time instant this discontinuity should occur.

### 4.1.2 Possible Solutions

Having looked at some of the issues surrounding the simulation of hybrid dynamical systems, it is clear that it will not be easy for us to sort out
all of these problems on our own. Fortunately for us, there has been a lot of research into these problems, and others have found some reasonable solutions which we can make use of. The range of software solutions in which these problems have been at least partially addressed is reasonably large, but here we will discuss only a few of the most suitable options.

MATLAB is probably the most commonly used program for simulation of dynamical systems, and it has many different simulation environments, most of which integrate well with each other. There are two main options within the MATLAB program that can be used for simulating a hybrid dynamical system; the Simulink package, or the Stateflow toolbox. Simulink is an environment which provides a graphical user interface to build models of dynamical systems, with numerical ODE solvers and zero-crossing detection behind the scenes. The discrete elements of the system must be built in by using extra blocks, for example by using integrator blocks with resets. This can be time consuming when making the Simulink model, especially if there are a large number of discrete elements to build in. However, because these discrete elements are built in using standard Simulink blocks, the actual simulation is very robust, since Simulink has been well tested for a number of years.

The second option within MATLAB is to use the Stateflow toolbox [14]. This works within the usual Simulink package, with special Stateflow blocks to deal with the discrete parts of the system. This means that Simulink now only directly controls the continuous dynamics of the system, and communicates with the Stateflow blocks to send continuous state information, and receive discrete location information. A Stateflow diagram is basically a finite automaton with inputs and outputs, so is very useful for simulating
hybrid dynamical systems modelled using the hybrid automaton representation. Creating the discrete elements in the Stateflow simulation of a system is relatively simple, and so it is easy to simulate a system with a large number of discrete parts. There are, however, problems with having two separate controllers for the discrete and continuous part of the problem, since points where discrete locations change can be detected inaccurately.

Another way we can simulate hybrid dynamical systems is by using a programming language called SHIFT, which was originally created for modelling automated highway systems [6]. The syntax of this language matches the hybrid automata formalism very closely, and so this is an easy language to use for users familiar with hybrid automata. This language is object-oriented and scalable, with the unique feature that the system being modelled can be changed whilst the simulation is running, by adding or removing copies of an automaton, for example. The main problem with this language is that the underlying numerical algorithms are very poor, and do not give accurate results in most cases.

The last software package that we will discuss is called Modelica [2]. It is an object-oriented and scalable language which has been designed to model physical systems, and so has a large library of components which can be connected together to make a model of the system. The main advantage of Modelica is the fact that it has a numerically stable simulation engine, which can cope with differential-algebraic equations, and which does zero-crossing detection automatically. There is also a specific Stategraph library which is part of the language, so it is fairly straightforward to simulate systems that are modelled as hybrid automata, by using the Stategraph components for the discrete location changing, and then textual commands to control the
4.1.3 Software Used in this Project

Having considered some of the different options for simulation of hybrid dynamical systems, we should make a decision about which of these software packages we are going to use in this project. The main aim of the simulation part of this project is to accurately compare hybrid automata models of the system (shown in Chapter 3), with the discontinuous equation model of the system. Bearing this in mind, we will discount SHIFT as an option, on account of its poor numerical algorithms.

Since we are modelling hybrid automata, it seems sensible to use software that makes the process of simulating automata fairly simple, and so we consider MATLAB’s Stateflow package and Modelica as our two simulation options — the process of creating discrete elements in Simulink seems to be unnecessarily arduous for the purposes of this project. Since we still have two options that will both make simulation of hybrid automata fairly simple, we have considered that it will be an interesting exercise to compare the simulations that result from the two. These conclusions will hopefully be useful for any future simulations of hybrid automata that we undertake.

4.2 Simulations of the Hybrid Automata for the Original Problem

The rest of this Chapter is devoted to simulating the different hybrid automata introduced in Chapter 3. This section looks at the simulations of the three automata for the original problem using MATLAB’s Stateflow toolbox,
and compares them against the discontinuous differential equation representation of the system studied in Section 2.3. We then use Modelica to simulate these automata, and compare its performance to that of MATLAB.

4.2.1 Simulation using MATLAB’s Stateflow Toolbox

Simulation using the Stateflow toolbox involves graphically representing the automaton system using standard Simulink blocks, with the discrete location changes modelled by a Stateflow chart. The Stateflow chart itself is effectively a finite automaton with inputs and outputs; the inputs are values from the continuous part that are required in the conditions in the automaton, and the outputs are the resulting discrete location and possibly the reset continuous state. It is worth noting that the dynamics are always outside of the Stateflow chart; the location number is outputted from the Stateflow chart to the Simulink model, and then it is used as an input to a switch block to choose between the right-hand-sides for the differential equations.

The three hybrid automata presented in Section 3.1 have been modelled in Simulink with Stateflow, which gives us a graphical model. These models look quite confusing, so we only include the Stateflow part of the models in Figure 4.1. However, the whole of the 2-location hybrid automaton Simulink model is shown in Appendix A.2 to give an idea of the feel of these kind of models. Note that the 5-location automaton which has been simulated is the original one presented in paper [16], since the theoretical refinement of the definition of this automaton was obtained after the simulations had been made.

We should make a comment here before we move on to simulating the models.
Figure 4.1: Stateflow representations of the two, three, and five location automata. To unclutter the picture for 5-locations, long guard conditions have been replaced by \([\text{cond}_q \text{q}4 \text{q}2]\) or similar.
we have created. When using Simulink with Stateflow to model differential
equations, we must use numerical integration routines which are built into
MATLAB. These integrators are generally very efficient and integrate ac-
curately given suitable values for step sizes and tolerances. However, the
default setting in Simulink models is to take the maximum step size as one-
fiftieth of the length of the time interval, which is very large when we want
accurate solutions to the problems. There are also other parameters whose
default values are fairly large and which need to be adjusted, specifically the
minimum step size, and the relative or absolute tolerance.

With the configuration parameters set to values which give a high level of
accuracy (for most simulations we use a maximum step size of $10^{-3}$, and
a relative tolerance of $10^{-6}$), we can now simulate these models to see how
they compare with each other, and how they compare with the discontinuous
dynamical system simulations. We will firstly look at the three cases that
we considered in Section 2.3. These all had the motor torque set to $u = 6000$ N m, with the weight on the bit given by:

- $W_{ob} = 50000$ N for convergence to an equilibrium with positive velocity;
- $W_{ob} = 59208$ N for permanent stuck bit;
- $W_{ob} = 53018$ N for stick-slip motion.

The Stateflow models were all simulated for the above values, and the results
for all four models at each parameter value were compared (Figure 4.2).
There is hardly any noticeable difference between the solutions obtained, so
we conclude that these hybrid automata are good models for the system at
these parameter values. We have also run these Stateflow model simulations
at other parameter values, and have obtained the same kind of results when
the parameters lie well inside one specific region of long-term behaviour.
Figure 4.2: Comparison of the hybrid automata and the discontinuous differential equation for the three behaviour patterns. $u = 6000$ Nm. Top: Convergence to equilibrium behaviour for $W_{ob} = 50000$ N. Middle: Permanent stuck behaviour for $W_{ob} = 59208$ N. Bottom: Stick-slip behaviour for $W_{ob} = 53018$ N. In each case, the left plot shows the rotary velocity of the bit for the four cases, and the discrete location changes are on the right.
Now, since we are comparing the behaviour of the automata with the discontinuous system, we would like to see if we can find a noticeable difference between the models when they are simulated. To do this we consider a set of parameters that are basically on the boundary between two long-term behaviour types; that is we simulate the system for parameter values very near to a bifurcation point. After investigating parameter values in between the stick-slip and permanently stuck states we settled on a parameter set to try: \([u, W_{ob}] = [6000, 57577]\). The plot of the bit velocity obtained for these parameter values from each different model is shown in Figure 4.3.

This plot shows us that there are some differences between the simulations of the four models for these parameter values. All of the models still converge to the same long-term behaviour, which is to have a permanently stuck bit,
but they stop oscillating at slightly different points in time. However, the main issue is that the 5-location hybrid automaton solution is noticeably out of sync with the other three simulations, from about \( t = 13 \) s onwards, which is shown by the right-hand plot. These differences in behaviour could be caused by one of three things.

1. **Is there a problem with the simulation?**
   This is the first option we consider, since there are many problems that could occur when making a computational simulation. We have, however, run simulations of these models with smaller and larger time-step sizes and tolerances to see if the correct behaviour is shown if these are changed, and have not obtained any better results than these.

2. **Is there a problem with the hybrid definition of the original system?**
   The hybrid automata models are perhaps flawed, and should be thought out more carefully, since they may not actually be modelling the same system as the original. We may need to revise the guard conditions, and perhaps the reset conditions also. Then we should run the simulations again to see if this has corrected the problems.

3. **Is the hybrid system a new dynamical system?**
   If we cannot make simulations of the original or revised models that give the same results as each other, then we must start to consider whether this hybrid representation of the system is actually a new system. It can never be proven that this is the case by doing computer simulations, but it can be made likely by the lack of any evidence of error in the model. However, if this is the solution to the problem, we will have to consider whether we can actually model this particular system via the hybrid framework, since we want the new models to be *equivalent*.
forms of the system, rather than new systems in their own right.

Given these discussion points, it is desirable that we start narrowing down the options for what is causing the problem. We have said that the first consideration is whether this is a simulation problem, so this will be the first investigation made. We will do this by simulating the discontinuous problem and hybrid automata using a different piece of software, namely, Modelica.

### 4.2.2 Simulation using Modelica’s Stategraph library

As mentioned before, Modelica is an object-oriented modelling language [2], which has a library of packages of predefined components to make the study of complex physical systems easier. One of these packages is the Stategraph package, which is a set of components that can be used to define automata. The main components are a “step” and a “transition”. The step component is equivalent to a location in the automaton, and the transition component is a way of implementing a guard condition. Any reset conditions can be included using the text-based command “reinit(x,y)”, where x is the variable that is being reinitialised with the value y.

Given these basic building blocks of a Modelica model, we can produce models for the three hybrid automata presented earlier in this project. The textual representation of these models is given in Appendix A.3, along with the Modelica model for the discontinuous system. It should be noted here that Modelica models can be created using either a graphical model editor, or just by using text. The models in this project were made by using a combination of these two methods, but since the graphical editing software simply writes the equivalent Modelica code in the underlying text file, along
with some annotation details, it is possible to obtain an entirely text version of the model.

The three hybrid models were simulated under the Dymola environment [1] (designed to use Modelica); the algorithm used was LSODAR, which stands for “Livermore Solver of Ordinary Differential Equations with Automatic Method Switching for stiff and nonstiff problems, and with Root-finding”. As suggested in the long version of the name, this algorithm has an inbuilt root finder, meaning it tries to find the exact places where discrete events happen and locations should be changed.

In Dymola, the only parameter that can be changed to affect the workings of the LSODAR solver is the tolerance, which makes it very user friendly. If we compare this with Stateflow under MATLAB, all the different parameters are given default values, but it is left possible for the user to change them. The down side of this (as mentioned earlier) is that it is not always obvious which parameter should be changed if the results are not satisfactory, which is often the case given the large step sizes and tolerances that are used by default. However, this does have the upside of leaving the user entirely in control.

Given the above discussion, we thought it would be useful to make a short digression into the effect of changing the tolerance within Modelica, to get an idea of the value it should take to solve this problem accurately. We will only look at the stick-slip case, since this is changing most rapidly, and will therefore be most affected if we have not chosen a small enough tolerance. Figure 4.4 shows the result of using different tolerances in the case of the 3-location automaton. The results are analogous for the other automata.
The left plot is a general overview of how out-of-sync the solutions can get by the time we reach $t = 100$ s, and the right plot is a closer view of the end of the time interval, from 90 s to 100 s. This closer view clearly shows that the largest tolerance, $10^{-2}$, is not suitable for the problem, since we can obviously see the improvement when we make the tolerance smaller. The other tolerance sizes give much similar results, but we can still see, without zooming in any more, the difference between having a tolerance of $10^{-3}$, and having a smaller one. Lastly, the difference between the two smallest tolerances here is negligible, and so we will use a tolerance of $10^{-4}$ as a starting point for any other Modelica experiments. Choosing this tolerance gives us results which are very close to the MATLAB results, as we will discuss in the next section.

Figure 4.4: The effect of simulation tolerance in Dymola. $u = 6000$ N m, and $W_{ob} = 53018$ N. Left: overall effect of changing tolerance. Right: zoomed view of the top end of the interval.
4.2.3 Comparison: Stateflow vs Modelica

In Section 4.2.1 we saw that when the automata are simulated using the Stateflow package we get reasonable results, provided the parameters do not lie near the boundary of the behaviour regions. In a moment we will look at how accurate the Modelica simulations are for parameters near this boundary, but firstly we will check that the results we get when inside the three behaviour regions are as we expect.

Figure 4.5 shows examples of comparisons made for the MATLAB and Modelica simulations. The top row shows an example of a comparison made for the discontinuous system simulations, and the three lower rows show examples of the comparisons for the 2-location, 3-location, and 5-location hybrid automata respectively. It is clear to see that the Modelica simulations show near exact correlation with the MATLAB ones, which is good, especially given the different methods used to perform the integration in the two programs. However, for the MATLAB version of the 5-location hybrid automaton to achieve this level of accuracy, the solver (ODE45) needed a tolerance of $10^{-6}$, whereas the Modelica solver (LSODAR) only needed a tolerance of $10^{-4}$. This difference in the tolerance required meant that the number of steps the Modelica solver was required to take was far fewer than the number of steps for the MATLAB solver, which impacted directly on the time to solution.

Setting the simulation issues aside, we can attempt to form some conclusions about the models we have created in Modelica. We can safely assume that the Modelica models are modelling the same systems as the MATLAB models, since they give very close simulation results for parameter values away from
Figure 4.5: Examples of comparisons between MATLAB and Modelica simulations. The left column shows the whole of the time simulation interval, \([0, 100]\), and the right shows a closer view of a fast changing part of the interval. Top: discontinuous system simulations for the stick-slip case. Next-to-top: 2-location automaton simulations for the convergence to equilibrium case. Next-to-bottom: 3-location automaton simulations for permanently stuck bit case. Bottom: 5-location automaton simulations for the stick-slip case.
the boundaries between domains. However, we must look at the boundaries between domains, since we have already found that the Stateflow models do not work very well around these areas. We will only use the discontinuous system in MATLAB for comparison with the Modelica simulations, since the other three MATLAB models did not seem to give good solutions for these boundary cases.

We will look at the boundary case which has been studied for the MATLAB models, that is, we will take the motor torque to be \( u = 6000 \text{ Nm} \), and the weight on the bit to be \( W_{ob} = 57577 \text{ N} \). This case is near the boundary between the stick-slip long-term behaviour and the permanently stuck long-term behaviour.

The first comparison between MATLAB and Modelica focused on the two discontinuous system simulations. Both of these discontinuous simulations used tolerances of \( 10^{-6} \), and the plots obtained for comparison of these simulations are at the top of Figure 4.6. The left plot is the overall view of the simulation interval, and the right plot is a closer view of the bottom of the last two ‘lumps’. You can see that the MATLAB oscillation is occurring slightly ahead of the Modelica one, and examination of the data shows this difference is about 0.1 s. This, although noticeable, is not a major issue given the length of time the simulation has been running and the fact that two different algorithms have been used for the integration. Hence we see that the MATLAB and Modelica discontinuous models are still doing the same thing for this bounding case.

The bottom set of graphs in Figure 4.6 compare the four Modelica model simulations for this bounding case. We can see that these four models show far greater correlation than the previous comparison between the MATLAB
Figure 4.6: Modelica simulations for the boundary case $u = 6000$ N m, $W_{ob} = 57577$ N. The left column shows the whole of the time simulation interval, [0, 400], and the right shows a closer view of the end of the fast changing part of the interval. Top: MATLAB and Modelica discontinuous system simulations. Bottom: comparison of the different Modelica model simulations.

discontinuous simulation and the Modelica one. Indeed, these differences are too small to be discernible on the right-hand plot, which shows the last two ‘lumps’. The data shows that the four models are all within 0.02 s of each other, which is a negligible amount over this interval.

We will conclude this section with a few remarks about the suitability of Modelica’s Stategraph against MATLAB’s Stateflow for modelling hybrid automaton systems.

The first point to be made about Stateflow is that because it works under Simulink, the model must be set up graphically, by inserting blocks and con-
necting them together. Although this seems user friendly, it becomes cumbersome very quickly as the model size is increased. However, this graphical set-up gives us the ability to quickly identify the location of errors in the model, since the error handler brings up a dialog saying exactly which block(s) the error occurred in. In Modelica, the model can be set up graphically, as we mentioned above, but the real power in using it for simulations is the fact that models can be created entirely in text format, which is good for large problems.

Another feature of Modelica which makes it suitable for large problems is scalability, the ability to easily produce larger systems from smaller ones. This is done through having a library system, to which the user can add new packages of models, for use in future modelling efforts.

4.3 Simulation of the Hybrid Automaton for the System with the Sliding-Mode Based Controller

We will now move on to look at simulations of the 9-location hybrid automaton introduced in Section 3.3. This automaton controls the velocity of the drillstring, as previously discussed, and will force the drill to be in the positive velocity long-term behaviour, provided some conditions are met. We will not look at the theoretical considerations here, but will consider, through simulation, the values which produce convergence to positive velocity, and the values which do not.

The first step in the process of simulating this hybrid automaton is to make a model of it in one of our pieces of software. We have chosen to use Modelica
for this model, since we can do it purely in text format, which is good when considering the size of the model. The text file for this model is given in Appendix A.4, and shows the complexity of the model. There are 9 steps and 28 transitions, along with some extra library blocks to ensure zero-crossing detection, and code to implement the differential equations. We have executed this model, and although it contains a lot of components, it still only takes a short time to run (about 1 or 2 seconds on the machine used), and so appears to be an efficient simulation model for the system.

We will now look at the results obtained from the simulations of this system. The first set of plots we have obtained are shown in Figure 4.7. These give an idea of the two long-term behaviour types: convergence to equilibrium, and stick-slip motion (remember that the permanent stuck bit behaviour is eliminated by the control). These plots show the bit velocity on the left, and the discrete location evolution on the right.

Having noted the appearance of the time evolution, we will now look at the overall trend of long-term behaviour, as we did in Section 2.3. To do this we have kept the parameters $\eta$ and $\lambda$ constant at 1.0 and 0.3 respectively, and we have varied the parameters $W_{ob}$ (weight on the bit) and $\Omega$ (desired rotary velocity). The plot of the long-term behaviour pattern for each pair of parameters is given in Figure 4.8.

From this plot it is clear that the region of convergence to positive velocity is any value of $\Omega$ above a curve dependent on $W_{ob}$. It is interesting to note that the equation for the boundary line can be calculated [19]; it is dependent on the value of $\lambda$, and the other normal parameters of the drillstring. However, the equation of the boundary does not change with $\eta$, since this only affects the speed of convergence to the long-term behaviour.
Figure 4.7: Examples of the long-term behaviour for the 9-location automaton. $W_{ob} = 53018$ N. Top: Stick-slip behaviour for $\Omega = 2$ rad/s. Bottom: Convergence to equilibrium behaviour for $\Omega = 3$ rad/s. In both cases, the left plot shows the velocity of the bit, and the right plot shows the discrete location evolution.

Figure 4.8: Long-term behaviour for the 9-location automaton. Circles represent stick-slip behaviour, and plus signs represent positive velocity.
Remember that in the drillstring system without control, we had pairs of $u$ and $W_{ob}$ which determined the pattern of long-term behaviour. For any such pair, if we saw undesirable behaviour we needed to manually either increase the motor torque or decrease the weight on the bit to obtain positive velocity behaviour. This is similar to what we have now — the motor torque is taken care of by the control, but we have added in a parameter ($\Omega$) to specify the desired velocity. Provided we choose $\Omega$ high enough, the motor torque is able to overcome the friction torque, and the drillstring drills into the ground.

Now, it may seem like we have not actually improved the situation for the drillstring, in that we still must specify a pair of parameters correctly in order for it to drill into the ground. However, if we know the operational range of weight on the bit for the drillstring, then we can specify the operational range for the desired velocity, or vice-versa. The simulations we have performed above show that, for weight on the bit in the range 30000 N to 70000 N, the drillstring is safe provided $\Omega > 3.5 \text{ rad/s}$. This is a good situation to be in, because the usual desired rotary velocity for a drillstring is between 9 rad/s and 12 rad/s, so we are always going to be drilling into the ground.
Chapter 5

Some Analytical Aspects of Hybrid Automata

In this chapter, we will look at some theoretical points surrounding the hybrid automata we have examined. We will only consider the discrete behaviour of the automata, focusing on the discrete locations and the transitions — we will not consider the time and trajectory evolution. We aim to use some tools of finite state automata to describe partially the dynamical behaviour patterns encountered in the system under study. In particular, we want to look at the languages that our hybrid automata speak. To do this, we need to introduce some theory concerning the definition of a language, and the ways in which we can describe what kind of language we have. We will then apply this theory to the automata we have been studying, and prove some results about them.
5.1 An Example

We will firstly try to understand the idea of a language that an automaton can “speak”, that is, the language that an automaton defines. We will use an example taken from [13, p. 7–8], of a finite state automaton describing a manufacturing machine. The directed graph for this automaton is given in Figure 5.1. This graph shows us the three possible discrete locations, denoted by $I$ for “idle”, $W$ for “working”, and $D$ for “down”. The transitions are then labelled with the events that can happen, these events being $p$ for “part arrives”, $c$ for “complete processing”, $f$ for “failure” and $r$ for “repair”. We will assume that we start at the idle location, and want to end at the down location.

The only event that can happen when we are in the idle location is for a part to arrive. When a part arrives we go to the working location, and then
we either can complete processing to go back to the idle location, or we can have a failure which will take us to the down location. If we have gone to the down location, we can only go back to doing anything else if the machine is repaired, when it goes back to idling.

The question is: “What sequence of events are accepted by the automaton?” That is, what sequence of events will traverse around the automaton and end up at the final location? We write the answer to this using the symbols labelling the events, which we group together in an alphabet. The alphabet for this automaton is \{p, c, f, r\}. We form a word from letters in the alphabet for each possible trajectory. For example, in the manufacturing machine, pf is an accepted word, indicating that a part arrives and then we experience a failure. However, \((pc)^3pf\) is also an accepted word, indicating that we have received and processed parts three times before we receive a part and fail.

The language of the automaton is the set of all possible words that the automaton accepts. In this case, the language accepted by the automaton can be described by the regular expression \((pc + pfr)^*pf\), which looks complex, but is not difficult to explain. First think of basic words that will take you from the idle location back to the idle location: these are pc and pfr. Then do either of these as many times as you want (including none), so we still end up at the idle location: this gives us \((pc + pfr)^*\), where + represents ‘or’, and the * means ‘as many times as you like’. Then we want to add the specific terms which can take us to the final location from where we are: this is pf (which takes us to location D from location I).
5.2 Defining Languages and their Properties

The following definitions are extracted from [10].

Definition 3 (Language). Let $\Sigma$ be a finite alphabet, and $\Sigma^*$ be the set of all strings that can be made from the symbols in $\Sigma$. Then, a language on $\Sigma$ is any subset of $\Sigma^*$.

There is one note to be made about this definition, which is that there is no specification that the language should be made by an automaton. However, any language as defined above can be accepted by an automaton, but the automaton may have an infinite number of locations. Secondly, the only restriction that is placed on the words (or strings) in a language is that the alphabet, $\Sigma$, only contains a finite number of symbols.

The next definition characterises the notion of a regular language, which is closely related to automata. Recall that the finite automaton was defined in Section 1.3.1.

Definition 4 (Regular Language). Let $\Sigma$ be a finite alphabet, and $L$ be a language. If there exists a finite automaton which accepts this language then it is regular.

5.3 Languages of our Hybrid Automata

Now we have looked at some definitions relating to languages, we should apply them to our hybrid automata. We will not take into account the continuous dynamics, and we will only concern ourselves with the discrete locations and the guards (equivalent to transition functions in a finite au-
figure 5.2: Symbolic 2 and 3 location hybrid automata models for the drill-string.

Notice that in the hybrid automata no final locations are defined as we had in finite automata; instead, different long-term behaviour patterns define different final states.

2-Location Hybrid Automaton

We will firstly consider the 2-location automaton, which is pictured on the left of Figure 5.2. We have replaced the guard conditions with letters which represent them: \( a \) represents the condition \( x_3 = 0 \) and \( |u_{eq}| \leq T_s \), and \( b \) represents \( x_3 = 0 \) and \( |u_{eq}| > T_s \). We would like to consider the languages that represent the different long-term behaviour patterns.

Firstly, we will consider the permanently stuck bit behaviour. This is characterised by the fact that we leave the stick location and return to it a finite number of times, which means that our final location is \( q_2 \). The associated language is denoted by \((ba)^*\). This is a regular language, since we can just denote \( q_2 \) to be the final location, and then the 2-location automaton in the figure is a finite automaton which accepts this language.
Secondly, the convergence to equilibrium behaviour should be considered. This is characterised by ending up in the slip location after a finite number of transitions, which means our final location is $q_1$. This is equivalent to going to slip and back to stick as many times as you like (as long as it is finite), and then taking the transition from slip to stick as the last one. Hence, the language is denoted by $(ba)^*b$, which is regular, since we can just denote the slip location to be the final location, and then the automaton accepts this language.

The final language to consider is the one associated with stick-slip motion. This motion is characterised by going back and forth between stick and slip an infinite number of times, so the final state consists of a group of locations, alternating between $q_1$ and $q_2$. The stick-slip language is therefore denoted by $(ba)^\infty(\epsilon + b)$, where $\epsilon$ is the empty symbol, which returns us to our current location. However, since it will be an infinite language, it does not matter what the last symbol is, so we can just say the language is $(ba)^\infty$.

**Lemma 5.** The language associated with the stick-slip long-term behaviour for the 2-location automaton is not regular.

*Proof.* (By contradiction). Assume this stick-slip language is regular. Then there is a finite automaton which accepts this language. Then each string in the language must have a final symbol which takes it to the accepting location. However, the language for stick-slip motion is $(ba)^\infty$, which consists of one infinite string, and so it has no final symbol. Hence, we have a contradiction, and the language is not regular. □

The fact that the stick-slip language is non-regular means we cannot design a finite automaton to accept it. This makes sense when we think about the
infinite motion involved with this behaviour; if we are still oscillating, we
can never be sure that we will not settle down to one of the other long-term
behaviour patterns at some future point.

3-Location Hybrid Automaton

We will now consider the language of the 3-location automaton. The symbolic
representation of the automaton is given on the right in Figure 5.2. The
symbol $a$ represents the condition $[x_3 = 0 \text{ and } |u_{eq}| \leq T_s]$, $b$ represents
$[x_3 = 0 \text{ and } u_{eq} > T_s]$, and $c$ represents $[x_3 = 0 \text{ and } u_{eq} < -T_s]$. The
languages for the three long-term behaviour patterns are denoted by:

- Permanent stuck bit $-$ $[c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^*$;
- Positive velocity $-$ $[c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^*(b + cb)$;
- Stick-slip $-$ $[c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^\infty$.

As with the 2-location automaton, we see that the permanent stuck bit and
positive velocity behaviours have regular language representations, whereas
the stick-slip behaviour has a non-regular language, due to its infinite nature.

5-Location Hybrid Automaton

The 5-location automaton is the next we should consider. We will use the
refined 5-location automaton described in Section 3.2. The symbolic version
of it is shown in Figure 5.3. The symbols represent the guard conditions,
with $a$ for $[|x_3| \leq \delta \text{ and } |u_{eq}| \leq T_s]$, $b$ for $[|x_3| \leq \delta \text{ and } |u_{eq}| > T_s]$, $c$ for
$[|x_3| \leq \delta \text{ and } |u_{eq}| < -T_s]$, and $d$ for $[|x_3| > \delta]$. The languages obtained are
denoted by:
Figure 5.3: Symbolic representation of the 5-location hybrid automaton.

- Permanent stuck bit — \([b+c](\epsilon + d)a]^*;\)
- Positive velocity — \([(b+c)(\epsilon + d)a]^*bd;\)
- Stick-slip — \([(b+c)(\epsilon + d)a]^{\infty}.\)

Again, we see that the stuck and positive velocity languages are regular, but the stick-slip language is not. It can be put forward as a general proposition that the stick-slip motion will always have a non-regular language, whatever automaton is used to model the system.

**9-Location Hybrid Automaton**

We will now consider the 9-location automaton, whose symbolic representation is given in Figure 5.4. The language for this is complicated, as could be expected, but it is made simpler by the fact that the sliding surface is attractive. In terms of Figure 5.4, this means that trajectories which reach any of the three locations in the middle column can never escape from this column. The basic structure of each column of 3 locations is comparable
to the 3-location automaton, which will also help in our definition of the language of this hybrid automaton.

We will first find the language which is associated with the transition from $q_9$, the initial location, to $q_6$, the stick location on the attractive surface. This is not a transition that is associated with either of the long-term behaviour patterns for this system, but it will help us find the languages associated with the behaviours we see. We split up the definition of the language into three sections. Firstly we consider which words take $q_9$ to itself, then we consider which words take $q_9$ directly to $q_6$ (without any loops), and lastly we consider the words which take $q_6$ to itself. We will then be able to combine these words.

Firstly then, $q_9$ to itself. This is the same as the definition of the stuck
language for the 3-location automaton, with $a$ replaced by $f$. Hence, the language associated with going from $q_9$ to itself is denoted by $[c(bc)^* (\epsilon + b)f + b(cb)^*(\epsilon + c)f]^*$.

We now consider the language which is associated with the transition from $q_9$ to $q_6$ without any loops. This consists of going to a location in the right hand column, jumping across to the middle column, and getting to $q_6$. This language is denoted by $[d + (\epsilon + b)c(i + g(\epsilon + b)a) + (\epsilon + c)b(i + h(\epsilon + c)a)]$. This is explained by thinking about each location of the right column in turn. If we jump directly from $q_9$ to $q_6$ we do this by word $d$. If we jump from $q_2$ to the middle automaton, we must first go from $q_9$ to $q_2$ via $[c + bc]$, then we must either do $i$ to go directly to $q_6$, or $ga$ or $gba$ to go through $q_5$. In a similar manner, we can obtain the last term when we go through $q_4$.

Lastly, we consider the language associated with the transition from $q_6$ to itself. This is exactly the language which we obtained for the stuck language for the 3-location automaton: $[c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^*$.

Putting these three sections together, we obtain the language defining the transitions to the stuck long-term behaviour. However, when this automaton was introduced, it was mentioned that the permanently stuck behaviour is eliminated by this controller. Hence, we only need to consider the languages for the positive velocity and stick-slip behaviours. The positive velocity language is denoted by

$$[c(bc)^*(\epsilon + b)f + b(cb)^*(\epsilon + c)f]^* \cdot [d + (\epsilon + b)c(i + g(\epsilon + b)a) + (\epsilon + c)b(i + h(\epsilon + c)a)] \cdot [c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^* \cdot [c + bc],$$

and the stick-slip language is denoted by
\[
[c(bc)^*(\epsilon+b)f+b(cb)^*(\epsilon+c)f]^* \cdot [d+(\epsilon+b)c(i+g(\epsilon+b)a)+(\epsilon+c)b(i+h(\epsilon+c)a)] \\
\cdot [c(bc)^*(\epsilon + b)a + b(cb)^*(\epsilon + c)a]^\infty.
\]

The positive velocity language represents the desired dynamical long-term behaviour, that is, ending in \(q_5\) and having \(x_3 = \Omega > 0\). The stick-slip language represents alternating between \(q_5\) and \(q_6\). The definition of the stick-slip language is interesting, due to the infinite nature of the expression. Since the sliding surface is attractive, the continuous time trajectory will definitely reach it, therefore the infinite length word enters when we are in the sliding surface. We have again obtained a regular language for the positive velocity behaviour, and a non-regular language for the stick-slip behaviour.
Chapter 6

Conclusions

In this project we have looked at one particular discontinuous system, namely a conventional torsional oilwell drillstring. We have simulated the original system, and have also simulated reformulations of it in the terms of three hybrid automata. The hybrid automaton models were found to be good representations of the original system, provided they were simulated appropriately. The simulations were made using MATLAB script files, MATLAB Stateflow models, and Modelica models through Dymola.

The different simulation formats were compared for this problem, and it was discovered that although MATLAB is much more well-known, it is not necessarily the best software to use when simulating systems with discontinuities. Specifically, Simulink’s system for zero-crossing detection seemed to be lacking somewhat, whereas Modelica found crossing points with ease. In addition, it is easy to unwittingly create a Simulink model with an algebraic loop in it, which slows down the simulation considerably.

We also introduced a hybrid automaton for the system with a sliding-mode based controller, and simulated its behaviour for varying values of the key
parameters. We found that, although the control does not remove all unwanted behaviour, it does give us a choice of desired velocity for the drill. Provided we choose a high enough velocity, we will converge to it, and we can provide safe ranges for the operational velocity of the drill if we know the operational range of weight on the bit.

Finally, we looked at the languages defined by the various automata for the different long-term behaviour patterns. We discovered that the stick-slip behaviour is described by a non-regular language. This means we can never use a finite automaton to tell us when we are in the stick-slip behaviour, because a non-regular language cannot be accepted by a finite automaton.

Additional theory, specifically designed for hybrid automata, and different to conventional automata theory, should be used to identify the different long-term dynamical behaviours in a hybrid automaton with a finite number of locations. This includes all the verification theory, model-checking paradigms, and viability or timed-automata theory, which is beyond of the scope of this project.
Bibliography


Appendix A

A.1 MATLAB Script File for Simulation of the Discontinuous Problem

Driver Script

% MATLAB script file to simulate a drillstring
% ODE45 is used to simulate the drillstring,
% and results are plotted.

% Initialise parameters
Jr = 2122;    % Rotary Inertia, kg m^2
Jb = 471.9698; % Bit inertia, kg m^2
Rb = 0.155575; % Radius of the bit, m
cr = 425;    % Damping on rotary inertia, N m s / rad
ct = 172.3067; % Torsional damping, N m s / rad
mucb = 0.5;  % Coulomb friction coefficient
musb = 0.8;  % Static friction coefficient
Dv = 0.000001; % Size of transition region
vf = 1;      % " " " "
gamb = 0.9;  % Speed of exponential decay
kt = 861.5336; % Torsional stiffness, N m / rad
u = 6000;    % Motor torque, N m
Wob = 53018; % Weight on bit, N
tspan = [0,100];
x0 = [0 0 0];
[t,x] = ode45(@ss_diff,tspan,x0,[],...
    ...u,ct,cr,cb,kt,Jr,Jb,Rb,mucb,musb,Dv,gamb,vf,Wob);

subplot(1,2,1)
plot(t,x(:,1),'-r',t,x(:,3),'-k')
title('Inertia velocities')
legend('Rotary inertia','Bit inertia','Location', 'Best')
axis auto
xlabel('Time')
ylabel('Rotary Velocity')

subplot(1,2,2)
plot3(x(:,2),x(:,1),x(:,3))
xlabel('phi_r - phi_b (rad)')
ylabel('phidot_r')
zlabel('phidot_b')
grid on

Function Handle for Integrator

function dx = ss_diff(t,x,u,ct,cr,cb,kt,Jr,Jb,Rb,mucb,musb,...
    ...Dv,gamb,vf,Wob)
    tor = Tfb;
    dx = zeros(3,1);
    dx(1) = (u-(ct+cr)*x(1)-kt*x(2)+ct*x(3))/Jr;
    dx(2) = x(1)-x(3);
    dx(3) = (ct*x(1)+kt*x(2)-(ct+cb)*x(3)-tor)/Jb;

function tor = Tfb
    Teb = ct*(x(1)-x(3))+kt*x(2)-cb*x(3);
    Tsb = Rb*Wob*musb;
    if (abs(x(3))>=Dv)
        tor = Rb*Wob*(mucb+(musb-mucb)*exp(-gamb/vf*...%
            ...abs(x(3))))*sign(x(3)); %sliding
    elseif (abs(Teb)>Tsb)
tor = Tsb*sign(Teb); \% stick to slip transition 
else 
    tor = Teb; \% stick 
end 
end
A.2 Stateflow Model of 2–Location Hybrid Automaton
A.3 Modelica Models for the System

Discontinuous Dynamical System

model drill_eqm "Model with discontinuous torque"

Real x1(start = 0.0);
Real x2(start = 0.0);
Real x3(start = 0.0);
Real Tfb;
Real Tsb;
Real Tab;
Real Teb;
Real mub;
Real fb;
parameter Real cb = 50.0;
parameter Real kt = 861.534;
parameter Real u = 6000.0;
parameter Real Wob = 50000.0;
constant Real Jr = 2122.0;
constant Real Jb = 471.97;
constant Real ct = 172.31;
constant Real cr = 425.0;
constant Real mucb = 0.5;
constant Real musb = 0.8;
constant Real gamb = 0.9;
constant Real vf = 1.0;
constant Real Rb = 0.155575;
constant Real Dv = 1e-06;

equation
Tsb = Wob * Rb * musb;
Tab = cb * x3;
Teb = ct * (x1 - x3) + kt * x2 - Tab;
mub = mucb + (musb - mucb) * exp((-gamb / vf) * abs(x3));
fb = Rb * Wob * mub;
Tfb = if abs(x3) >= Dv then fb * sign(x3)
    else if abs(Teb) > Tsb then Tsb * sign(Teb) else Teb;
der(x1) = 1 / Jr * (-(ct + cr) * x1 - kt * x2 + ct * x3 + u);
\[
\text{der}(x_2) = x_1 - x_3;
\]
\[
\text{der}(x_3) = \frac{1}{J_b} \left( ct \cdot x_1 + kt \cdot x_2 - (ct + cb) \cdot x_3 - T_{fb} \right);
\]
end drill_eqm;

2–Location Hybrid Automaton

model drill_eqm

Real x1(start = 0.0);
Real x2(start = 0.0);
Real x3(start = 0.0);
Real Tfb;
Real Tsb;
Real Tab;
Real Teb;
Real mub;
Real fb;
Integer q;
parameter Real cb = 50.0;
parameter Real kt = 861.534;
parameter Real u = 6000.0;
parameter Real Wob = 50000.0;
constant Real Jr = 2122.0;
constant Real Jb = 471.97;
constant Real ct = 172.31;
constant Real cr = 425.0;
constant Real mucb = 0.5;
constant Real musb = 0.8;
constant Real gamb = 0.9;
constant Real vf = 1.0;
constant Real Rb = 0.155575;
constant Real Dv = 1e-06;
Modelica.StateGraph.TransitionWithSignal transition
Modelica.StateGraph.TransitionWithSignal transition1
Modelica.StateGraph.InitialStepWithSignal Stick
Modelica.StateGraph.StepWithSignal Slip annotation
Modelica.Blocks.Sources.RealExpression x3value(y=x3)
Modelica.Blocks.Logical.And and1

102
Modelica.Blocks.Sources.RealExpression TebIn(y=Teb)
Modelica.Blocks.Math.Abs abs1
Modelica.Blocks.Logical.Not not1
Modelica.Blocks.Logical.Greater greater
Modelica.Blocks.Sources.RealExpression TsbIn(y=Tsb)

equation
  connect(Stick.outPort[1], transition.inPort)
  connect(transition.outPort, Slip.inPort[1])
  connect(Slip.outPort[1], transition1.inPort)
  connect(transition1.outPort, Stick.inPort[1])
  connect(x3value.y, novelocity.u)
  connect(TebIn.y, abs1.u)
  connect(novelocity.y, and1.u2)
  connect(not1.y, and1.u1)
  connect(and1.y, transition1.condition)
  connect(TsbIn.y, greater.u2)
  connect(abs1.y, greater.u1)
  connect(greater.y, not1.u)
  connect(greater.y, transition.condition)
  connect(Slip.active, novelocity.enable)

Tsb = Wob * Rb * musb;
Tab = cb * x3;
Teb = ct * (x1 - x3) + kt * x2 - Tab;
mub = mucb + (musb - mucb) * exp(( -gamb / vf) * abs(x3));
fb = Rb * Wob * mub;
Tfb = if Slip.active==true then fb * sign(x3) else Teb;
der(x1) = 1 / Jr * ( -(ct + cr) * x1 - kt * x2 + ct * x3 + u);
der(x2) = x1 - x3;
der(x3) = 1 / Jb * (ct * x1 + kt * x2 - (ct + cb) * x3 - Tfb);
q = if Stick.active then 2 else 1;

end drill_eqm;

3–Location Hybrid Automaton

model drill_eqm
Real x1(start=0.0);
Real x2(start=0.0);
Real x3(start=0.0);
Real Tfb;
Real Tsb;
Real Tab;
Real Teb;
Real mub;
Real fb;
Integer q;
parameter Real cb=50.0;
parameter Real kt=861.534;
parameter Real u=6000.0;
parameter Real Wob=50000.0;
constant Real Jr=2122.0;
constant Real Jb=471.97;
constant Real ct=172.31;
constant Real cr=425.0;
constant Real mucb=0.5;
constant Real musb=0.8;
constant Real gamb=0.9;
constant Real vf=1.0;
constant Real Rb=0.155575;
constant Real Dv=1e-06;

Modelica.StateGraph.TransitionWithSignal st_to_sl_plus
Modelica.StateGraph.TransitionWithSignal sl_plus_to_st
Modelica.StateGraph.InitialStepWithSignal Stick(nIn=2, nOut=2)
Modelica.StateGraph.StepWithSignal Slip_plus(nIn=2, nOut=2)
Modelica.Blocks.Sources.RealExpression x3value(y=x3)
Modelica.Blocks.Logical.And and1
Modelica.Blocks.Sources.RealExpression TebIn(y=Teb)
Modelica.Blocks.Logical.Greater greater
Modelica.Blocks.Sources.RealExpression TsbIn(y=Tsb)
Modelica.StateGraph.StepWithSignal Slip_minus(nIn=2, nOut=2)
Modelica.StateGraph.TransitionWithSignal sl_minus_to_st
Modelica.StateGraph.TransitionWithSignal sl_minus_to_sl_plus
Modelica.StateGraph.TransitionWithSignal st_to_sl_minus
Modelica.StateGraph.TransitionWithSignal sl_plus_to_sl_minus
Modelica.Blocks.Logical.Less less
Modelica.Blocks.Math.Gain gain(k=-1)
Modelica.Blocks.Math.Abs abs1
Modelica.Blocks.Logical.LessEqual lessEqual
Modelica.Blocks.Logical.And and2
Modelica.Blocks.Logical.And and3
Modelica.Blocks.Sources.BooleanExpression enableDetect(y=true)
Modelica.Blocks.Logical.Edge enter_st
Modelica.Blocks.Logical.Edge enter_sl_plus
Modelica.Blocks.Logical.Edge enter_sl_minus

equation
  connect(Stick.outPort[1], st_to_sl_plus.inPort)
  connect(st_to_sl_plus.outPort, Slip_plus.inPort[1])
  connect(Slip_plus.outPort[1], sl_plus_to_st.inPort)
  connect(sl_plus_to_st.outPort, Stick.inPort[1])
  connect(x3value.y, novelocity.u)
  connect(novelocity.y, and1.u2)
  connect(and1.y, sl_plus_to_st.condition)
  connect(TsbIn.y, greater.u2)
  connect(greater.y, st_to_sl_plus.condition)
  connect(Slip_minus.outPort[1], sl_minus_to_st.inPort)
  connect(sl_minus_to_st.outPort, Stick.inPort[2])
  connect(sl_minus_to_st.outPort, Stick.inPort[2])
  connect(sl_minus_to_st.outPort, Stick.inPort[2])
  connect(Stick.outPort[2], st_to_sl_minus.inPort)
  connect(st_to_sl_minus.outPort, Slip_minus.inPort[1])
  connect(Slip_plus.outPort[2], sl_plus_to_sl_minus.inPort)
  connect(sl_plus_to_sl_minus.outPort, Slip_minus.inPort[2])
  connect(Stick.outPort[2], st_to_sl_minus.inPort)
  connect(TebIn.y, greater.u1)
  connect(TebIn.y, less.u1)
  connect(TsbIn.y, gain.u)
  connect(gain.y, less.u2)
  connect(less.y, st_to_sl_minus.condition)
  connect(TebIn.y, abs1.u)
  connect(lessEqual.y, and1.u2)
  connect(abs1.y, lessEqual.u2)
  connect(TebIn.y, lessEqual.u2)
  connect(and1.y, sl_minus_to_st.condition)
  connect(greater.y, and2.u1)
connect(novelocity.y, and2.u2)
connect(and2.y, sl_minus_to_sl_plus.condition)
connect(less.y, and3.u1)
connect(novelocity.y, and3.u2)
connect(and3.y, sl_plus_to_sl_minus.condition)
connect(enableDetect.y, novelocity.enable)
connect(Stick.active, enter_st.u)
connect(Slip_plus.active, enter_sl_plus.u)
connect(Slip_minus.active, enter_sl_minus.u)

when enter_st.y then
  reinit(x3, 0.0);
end when;
Tsb = Wob*Rb*musb;
Tab = cb*x3;
Teb = ct*(x1 - x3) + kt*x2 - Tab;
mub = mucb + (musb - mucb)*exp((-gamb/vf)*abs(x3));
fb = Rb*Wob*mub;
Tfb = if Slip_plus.active then fb
     else if Slip_minus.active then -fb else Teb;
der(x1) = 1/Jr*(-(ct + cr)*x1 - kt*x2 + ct*x3 + u);
der(x2) = x1 - x3;
der(x3) = 1/Jb*(ct*x1 + kt*x2 - (ct + cb)*x3 - Tfb);
q = if Stick.active then 3
     else if Slip_minus.active then 2 else 1;

end drill_eqm;

5–Location Hybrid Automaton

model drill_eqm

  Real x1(start = 0.0);
  Real x2(start = 0.0);
  Real x3(start = 0.0);
  Real Tfb;
  Real Tsb;
  Real Tab;
  Real Teb;
Real mub;
Real fb;
Integer q;
parameter Real cb = 50.0;
parameter Real kt = 861.534;
parameter Real u = 6000.0;
parameter Real Wob = 50000.0;
constant Real Jr = 2122.0;
constant Real Jb = 471.97;
constant Real ct = 172.31;
constant Real cr = 425.0;
constant Real mucb = 0.5;
constant Real musb = 0.8;
constant Real gamb = 0.9;
constant Real vf = 1.0;
constant Real Rb = 0.155575;
constant Real Dv = 1e-06;

Modelica.StateGraph.Transition transition(condition=
    (abs(x3)<=Dv) and (Teb>Tsb))
Modelica.StateGraph.Transition transition1(condition=
    (abs(x3)<=Dv) and (abs(Teb)<=Tsb))
Modelica.StateGraph.InitialStepWithSignal Stick(nOut=2, nIn=4)
Modelica.StateGraph.Step Slip_plus(nIn=1, nOut=1)
Modelica.StateGraph.Step Slip_minus(nIn=1, nOut=1)
Modelica.StateGraph.Transition transition2(condition=
    (abs(x3)<=Dv) and (abs(Teb)<=Tsb))
Modelica.StateGraph.Transition transition4(condition=
    (abs(x3)<=Dv) and (Teb<-Tsb))
Modelica.StateGraph.Step Trans_minus(nOut=2)
Modelica.StateGraph.Step Trans_plus(nOut=2)
Modelica.StateGraph.Transition transitionWithSignal(condition=
    (x3>Dv) or (der(x3)>Dv))
Modelica.StateGraph.Transition transitionWithSignal1(condition=
    (x3<-Dv) or (der(x3)<-Dv))
Modelica.StateGraph.Transition transitionWithSignal2(condition=
    (x3<-Dv) or (der(x3)<-Dv))
Modelica.StateGraph.Transition transitionWithSignal3(condition=
    (x3>Dv) or (der(x3)>Dv))
Modelica.Blocks.Logical.Edge enter_st

equation
connect(Stick.outPort[1], transition.inPort)
connect(Slip_plus.outPort[1], transition1.inPort)
connect(transition1.outPort, Stick.inPort[1])
connect(Slip_minus.outPort[1], transition2.inPort)
connect(transition2.outPort, Stick.inPort[2])
connect(Stick.outPort[2], transition4.inPort)
connect(transition.outPort, Trans_plus.inPort[1])
connect(Trans_plus.outPort[1], transitionWithSignal.inPort)
connect(transitionWithSignal.outPort, Slip_plus.inPort[1])
connect(transition4.outPort, Trans_minus.inPort[1])
connect(Trans_minus.outPort[1], transitionWithSignal1.inPort)
connect(transitionWithSignal1.outPort, Slip_minus.inPort[1])
connect(Trans_plus.outPort[2], transitionWithSignal2.inPort)
connect(transitionWithSignal2.outPort, Stick.inPort[3])
connect(Trans_minus.outPort[2], transitionWithSignal3.inPort)
connect(transitionWithSignal3.outPort, Stick.inPort[4])
connect(Stick.active, enter_st.u)

when enter_st.y then
  reinit(x3, 0.0);
end when;
Tsb = Wob * Rb * musb;
Tab = cb * x3;
Teb = ct * (x1 - x3) + kt * x2 - Tab;
mub = mucb + (musb - mucb) * exp(( -gamb / vf) * abs(x3));
fb = Rb * Wob * mub;
Tfb = if Slip_plus.active then fb
  else if Slip_minus.active then -fb
  else if Trans_plus.active then Tsb
  else if Trans_minus.active then -Tsb else Teb;
der(x1) = 1 / Jr * ( -(ct + cr) * x1 - kt * x2 + ct * x3 + u);
der(x2) = x1 - x3;
der(x3) = 1 / Jb * (ct * x1 + kt * x2 - (ct + cb) * x3 - Tfb);
q = if Stick.active then 3 else if Slip_minus.active then 2
  else if Trans_plus.active then 5
  else if Trans_minus.active then 4 else 1;
A.4 Model for Sliding–Mode Based Controller Automaton

model drill

Real x1(start=0.0);
Real x2(start=0.0);
Real x3(start=0.0);
Real x4(start=0.0);
Real x5(start=0.0);
Real Tfbplus;
Real Tfbminus;
Real Tsb;
Real Tsb;
Real Tab;
Real Teb;
Real mub;
Real fb;
Real phi1;
Real phi2;
Integer q;
Boolean G0;
Boolean Gplus;
Boolean Gminus;
parameter Real cb=50.0;
parameter Real kt=861.534;
parameter Real Wob=53018.0;
parameter Real omega=12;
parameter Real lambda=0.3;
parameter Real eta=1.0;
constant Real Jr=2122.0;
constant Real Jb=471.97;
constant Real ct=172.31;
constant Real cr=425.0;
constant Real mucb=0.5;
constant Real musb=0.8;
constant Real gamb=0.9;
constant Real vf=1.0;
custom Real Rb=0.155575;
custom Real Dv=1e-06;
Real sr(start=-omega);
Boolean x3zero;

Modelica.StateGraph.Step q1(nOut=4,nIn=2);
Modelica.StateGraph.Step q2(nOut=4,nIn=2);
Modelica.StateGraph.Step q3(nOut=4,nIn=2);
Modelica.StateGraph.Step q4(nOut=4,nIn=2);
Modelica.StateGraph.Step q5(nOut=2,nIn=4);
Modelica.StateGraph.Step q6(nOut=2, nIn=8);
Modelica.StateGraph.Step q7(nOut=2,nIn=4);
Modelica.StateGraph.Step q8(nOut=3,nIn=2);
Modelica.StateGraph.InitialStep q9(nOut=3,nIn=2);

Modelica.StateGraph.Transition q1toq8(condition=G0 == true);
Modelica.StateGraph.Transition q1toq3(condition=Gminus == true);
Modelica.StateGraph.Transition q3toq1(condition=Gplus == true);
Modelica.StateGraph.Transition q3toq8(condition=G0 == true);
Modelica.StateGraph.Transition q8toq1(condition=Gplus == true);
Modelica.StateGraph.Transition q8toq3(condition=Gminus == true);
Modelica.StateGraph.Transition q5toq6(condition=G0 == true);
Modelica.StateGraph.Transition q5toq7(condition=Gminus == true);
Modelica.StateGraph.Transition q7toq5(condition=Gplus == true);
Modelica.StateGraph.Transition q7toq6(condition=G0 == true);
Modelica.StateGraph.Transition q6toq5(condition=Gplus == true);
Modelica.StateGraph.Transition q6toq7(condition=Gminus == true);
Modelica.StateGraph.Transition q2toq9(condition=G0 == true);
Modelica.StateGraph.Transition q2toq4(condition=Gminus == true);
Modelica.StateGraph.Transition q4toq2(condition=Gplus == true);
Modelica.StateGraph.Transition q4toq9(condition=G0 == true);
Modelica.StateGraph.Transition q9toq2(condition=Gplus == true);
Modelica.StateGraph.Transition q9toq4(condition=Gminus == true);

Modelica.StateGraph.Transition q1toq5(condition=
  (srzero.y == true and x3 > 0));
Modelica.StateGraph.Transition q8toq6(condition=
  srzero.y == true);
Modelica.StateGraph.Transition q3toq7(condition=110
(srzero.y == true and x3 < 0));
Modelica.StateGraph.Transition q2toq5(condition=
(srzero.y == true and x3 > 0));
Modelica.StateGraph.Transition q9toq6(condition= srzero.y == true);
Modelica.StateGraph.Transition q4toq7(condition= (srzero.y == true and x3 < 0));
Modelica.StateGraph.Transition q1toq6(condition= (G0 == true and srzero.y == true));
Modelica.StateGraph.Transition q2toq6(condition= (G0 == true and srzero.y == true));
Modelica.StateGraph.Transition q3toq6(condition= (G0 == true and srzero.y == true));
Modelica.StateGraph.Transition q4toq6(condition= (G0 == true and srzero.y == true));
Modelica.Blocks.Sources.RealExpression srvalue(y=sr);
Modelica.Blocks.Sources.BooleanExpression enableDetect(y=true);
Modelica.Blocks.Sources.RealExpression x3value(y=x3);
Modelica.Blocks.Logical.ZeroCrossing x3crosszero;
equation
connect(q1.outPort[1], q1toq8.inPort);
connect(q1toq8.outPort, q8.inPort[1]);
connect(q1.outPort[2], q1toq3.inPort);
connect(q1toq3.outPort, q3.inPort[1]);
connect(q3.outPort[1], q3toq1.inPort);
connect(q3toq1.outPort, q1.inPort[1]);
connect(q3.outPort[2], q3toq8.inPort);
connect(q3toq8.outPort, q8.inPort[2]);
connect(q8.outPort[1], q8toq1.inPort);
connect(q8toq1.outPort, q1.inPort[2]);
connect(q8.outPort[2], q8toq3.inPort);
connect(q8toq3.outPort, q3.inPort[2]);
connect(q5.outPort[1], q5toq6.inPort);
connect(q5toq6.outPort, q6.inPort[1]);
connect(q5.outPort[2], q5toq7.inPort);
connect(q5toq7.outPort, q7.inPort[1]);
connect(q7.outPort[1], q7toq5.inPort);
connect(q7toq5.outPort, q5.inPort[1]);
connect(q7.outPort[2], q7toq6.inPort);
connect(q7toq6.outPort, q6.inPort[2]);
connect(q6.outPort[1], q6toq5.inPort);
connect(q6toq5.outPort, q5.inPort[2]);
connect(q6.outPort[2], q6toq7.inPort);
connect(q6toq7.outPort, q7.inPort[2]);
connect(q2.outPort[1], q2toq9.inPort);
connect(q2toq9.outPort, q9.inPort[1]);
connect(q2.outPort[2], q2toq4.inPort);
connect(q2toq4.outPort, q4.inPort[1]);
connect(q4.outPort[1], q4toq2.inPort);
connect(q4toq2.outPort, q2.inPort[1]);
connect(q4.outPort[2], q4toq9.inPort);
connect(q4toq9.outPort, q9.inPort[2]);
connect(q9.outPort[1], q9toq2.inPort);
connect(q9toq2.outPort, q2.inPort[2]);
connect(q9.outPort[2], q9toq4.inPort);
connect(q9toq4.outPort, q4.inPort[2]);
connect(q1.outPort[3], q1toq5.inPort);
connect(q1toq5.outPort, q5.inPort[3]);
connect(q8.outPort[3], q8toq6.inPort);
connect(q8toq6.outPort, q6.inPort[3]);
connect(q3.outPort[3], q3toq7.inPort);
connect(q3toq7.outPort, q7.inPort[3]);
connect(q2.outPort[3], q2toq5.inPort);
connect(q2toq5.outPort, q5.inPort[4]);
connect(q9.outPort[3], q9toq6.inPort);
connect(q9toq6.outPort, q6.inPort[4]);
connect(q4.outPort[3], q4toq7.inPort);
connect(q4toq7.outPort, q7.inPort[4]);
connect(q1.outPort[4], q1toq6.inPort);
connect(q1toq6.outPort, q6.inPort[5]);
connect(q2.outPort[4], q2toq6.inPort);
connect(q2toq6.outPort, q6.inPort[6]);
connect(q3.outPort[4], q3toq6.inPort);
connect(q3toq6.outPort, q6.inPort[7]);
connect(q4.outPort[4], q4toq6.inPort);
connect(q4toq6.outPort, q6.inPort[8]);
connect(srvalue.y, srzero.u);
connect(enableDetect.y, srzero.enable);
connect(x3value.y, x3crosszero.u);
connect(enableDetect.y, x3crosszero.enable);

x3zero = x3crosszero.y or (x3 <= 0 and x3 >= 0);

when (edge(q6.active) or edge(q8.active) or edge(q9.active)) then
    reinit(x3,0.0);
end when;

Gplus = if ct*x1 + kt*x2 > Tsb and abs(x3)<=Dv then true
     else false;
Gminus = if ct*x1 + kt*x2 < -Tsb and abs(x3)<=Dv then true
     else false;
G0 = if abs(x3)<=Dv and abs(ct*x1 + kt*x2) <= Tsb then true
     else false;

Tsb = Wob*Rb*musb;
Tab = cb*x3;
Teb = ct*(x1 - x3) + kt*x2 - Tab;
mub = mucb + (musb - mucb)*exp((-gamb/vf)*abs(x3));
fb = Rb*Wob*mub;
Tfbplus = fb;
Tfbminus = -fb;
phi1 = -lambda*(x1 - omega) - lambda*(x1 - x3);
phi2 = (ct*x1 + kt*x2 - (ct + cb)*x3)/Jb;
der(x1) = if q1.active or q3.active then phi1 - eta
    else if q2.active or q4.active then phi1 + eta
    else if q5.active or q7.active then phi1
    else if q6.active then -2*lambda*x1 + lambda*omega
    else if q8.active then -2*lambda*x1 + lambda*omega - eta
    else -2*lambda*x1 + lambda*omega + eta;
der(x2) = if q6.active or q8.active or q9.active then x1
    else x1 - x3;
der(x3) = if q1.active or q2.active or q5.active then
phi2 - Tfbplus/Jb
else if q3.active or q4.active or q7.active then
    phi2 - Tfbminus/Jb
else 0;
der(x4) = x1 - omega;
der(x5) = if q6.active or q8.active or q9.active then x1
else x1 - x3;
der(sr) = der(x1) - phi1;

q = if q1.active then 1 else if q2.active then 2
else if q3.active then 3 else if q4.active then 4
else if q5.active then 5 else if q6.active then 6
else if q7.active then 7 else if q8.active then 8 else 9;
end drill;