# SC42155 - MINI-PROJECT
# DYNAMICAL MODELLING OF A CART WITH A DIAMOND-SHAPED PENDULUM

February 1, 2021

Some images are small but in PDF format.
Please zoom in for these, I had great difficulties to adhere to the 12 page limit.
Apologies for the inconvenience.

Joris Verhagen
5189020

# Contents

# 1 PHYSICAL MODELLING

In this section I will address the dynamical modelling of the cart-pendulum system in Figure 13. The cart is propelled by a DC motor and rolls on wheels over an ideal and flat surface. To the cart a diamond-shaped pendulum-like structure is connected consisting of 4 joints $j_i$ and 4 beams $b_i$. Joints $j_1$ and $j_2$ are connected via a linear non-ideal spring (a spring that contains damping). Both the cart as well as the joints and beams have a mass, defined as $m_{cart}$, $m_{j_i}$ and $m_{b_i}$ respectively. Let us first state some assumptions

- The cart rolls over an ideal and flat surface
- Beam $b_1$ and $b_4$ are of equal length $l_{b1}$
- Beam $b_2$ and $b_3$ are of equal length $l_{b2}$
- The spring connecting $j_1$ and $j_2$ is linear but contains damping
- The beams $b_i$ have a uniformly distributed mass where $m_{b_1} = m_{b_4}$ and $m_{b_2} = m_{b_3}$
- All joints $j_i$ and the wheels of the cart contain damping terms that quadratically depend on the generalized velocity of the respective joint angles and cart displacement
- The cart is fully constrained in the $y$ direction and is also not allowed to rotate

A figure of the cart and its generalized coordinates is shown in Figure 13.

## 1.1 DC motor equations

First, let us obtain the physical equations connecting the supplied voltage at the motor to the state of the mechanical device. I assume the following electrical scheme representative of the motor itself shown in Figure 2. We know that based on this figure the torque exerted by the motor can be described by $\tau = k_{DC} I$ where $\tau$ is the

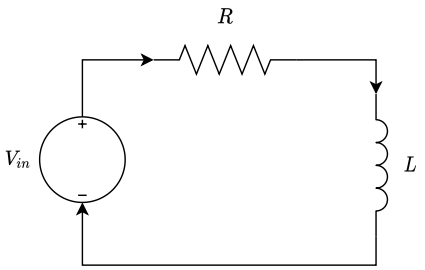



**Figure 1:** Electrical equivalent to a simple DC motor

torque in $Nm$, $k_{DC}$ is the DC motor constant, and $I$ is the current in $A$. I can transcribe the torque to propulsive force using the wheel radius

$$F = \frac{k_{DC}}{r_{wheels}} I \tag{1}$$

where $F$ is the propulsive force in $N$ and $r$ is the wheel radius in $m$. Using the scheme in Figure 2 I can obtain the following equation relating current to voltage

$$V = RI + \frac{dI}{dt} L \tag{2}$$

where $V$ is the voltage in Volt and $L$ is the inductance in Henry. I can now straightforwardly describe a state space model relating the input voltage to the current (the state), time derivative of the current, and the propulsive

force (the output)

$$\begin{cases} \dot{I} = -\frac{R}{L}I + \frac{1}{L}V \\ F = \frac{k_{DC}}{r}I \end{cases} \longrightarrow \begin{cases} \dot{x} = Ax + Bu \\ y = Cx \end{cases} \tag{3}$$

The state-space model resulting in the propulsive force $F$ will later be used when constructing the equations of motion of the form $\dot{x} = f(x,u)$ for numerical integration.

## 1.2 Lagrange equations of motion

Next, I will describe how the equations of motion can be obtained for the system based on the aforementioned assumptions. The Euler-Lagrange method and its required terms will only be briefly addressed. For a full derivation of the equations of motion, please refer to the appendix. The main advantage of Euler-Lagrange is that equations of motion are obtained on the energy level of the system which allows for a straightforward computation although mechanical intuition is certainly lost to methods such as Newton-Euler.

First let us define $q$, the vector of generalized coordinates. $q$ is by definition a minimal set of coordinates which can uniquely describe the configuration of the system at hand. As such, and based on the assumption that $l_{b_1} = l_{b_4}$ and $l_{b_2} = l_{b_3}$ I can find three generalized coordinates $q_1$, $q_2$, and $q_3$ as shown in Figure 13 (the x-displacement of the cart, the absolute rotation of the diamond-shaped pendulum, and the angle in between beam 1 and beam 2). Based on these generalized coordinates and the generalized coordinate time derivatives I can describe the kinetic and potential energy of the system and solve for the accelerations in Equation 20. A detailed derivation is shown in the Appendix.

The dissipation term $D$ can be computed by considering the friction on the wheels, the friction in the joints, and the damping of the spring.

$$D = \frac{1}{2}\dot{q}^T \begin{bmatrix} b_{wheels} & 0 & 0 \\ 0 & b_{j_4} & 0 \\ 0 & 0 & b_{j_1} + b_{j_2} + b_{j_3} \end{bmatrix} \dot{q} + b_{spring}(l_{b_1}\cos(\frac{q_3}{2})\dot{q}_3)^2 \tag{4}$$

where $b_x$ is the damping coefficient of component $x$. The external force in $Q$ can be described by a momentary expansion of the generalized coordinates with the current $I$ (this will become more clear later on). Assuming the current $I$ is known to the system, $Q$ is given as

$$Q = \begin{bmatrix} \frac{k_{DC}}{r_{wheels}}I \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

applying the derivatives according to Equation 20 allows us to find an equation which linearly depends on the acceleration of the generalized coordinates $\ddot{q}$. Using the MATLAB command `equationsToMatrix` I can obtain the linear combination of $A\ddot{q} = b$. The equations of motion can then be obtained by

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dot{I} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ A\backslash b \\ -\frac{R}{L}I + \frac{1}{L}V \end{bmatrix} \tag{6}$$

which is of the form $\dot{x} = Ax + Bu$ with the state x momentarily being equal to $x = \begin{bmatrix} q & \dot{q} & I \end{bmatrix}^T$ and input voltage $V$. I assume the reader is somewhat familiar with these conversions and methodologies. For more information on the derivation of the equations of motions see the Appendix.

Lastly, I add a hydraulic brake on the wheels by adding to the $D$ term a pressure dependent damping according to $D = D + \rho \dot{q}_1^2 P$ where $\rho$ is the pressure to dissipation coefficient (dependent among others on the friction coefficient between brake disk and brake claw) and $P$ is the supplied brake pressure. Contrary to supplying the DC motor with a negative voltage, which will slow down the cart and make it move in the other direction, supplying pressure to the hydraulic brake will bring the system to a total standstill.

## 1.3 Simulation

In this section I will simulate the derived dynamical model in MATLAB and analyze its behavior under different interesting initial conditions and voltage inputs. Reasonable parameters are chosen for simulation purposes. A table of all parameters is shown in Appendix A.3.

### 1.3.1 DC motor simulation

First I will look at the dynamics of the DC motor. The inductor represents the inductance generated by the coil and magnets of the DC motor and the resistor represents the energy loss from voltage to current (if there is no resistor then the current would be the direct integral of the applied voltage). A comparison for different inductor and resistance values is shown in Figure **??** Most importantly, we notice the energy loss for positive values of

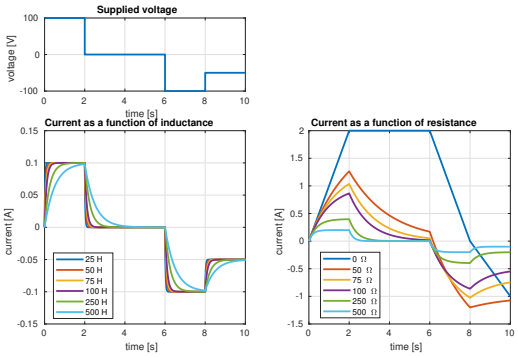



**Figure 2:** Simulation of aforementioned DC motor for different inductance and resistance values

the resistor in the schematic. Larger values of the resistance lead to a smaller current and a faster drop-off in current after the supplied voltage has dropped to zero. Subsequently, the DC motor model is able to accurately represent both the DC motor characteristics with the $L$ and $R$ values as well as model its losses using the $R$ value.

### 1.3.2 Cart-pendulum simulation

Next I will look at the dynamics of the cart-pendulum system. First I will not supply a voltage to the cart's DC motor and instead give initial positive angles to $q_2$ and $q_3$ as shown in Figure 13. In Figure 3 we can see

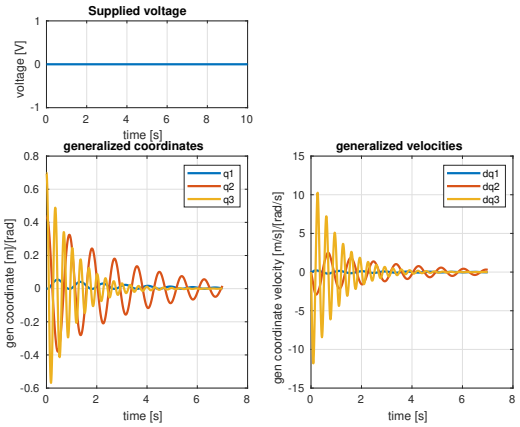



**Figure 3:** Simulation of cart-pendulum model without input and with initial state $x_0 = [q_{1_0}\, q_{2_0}\, q_{3_0}\, \dot{q}_{1_0}\, \dot{q}_{2_0}\, \dot{q}_{3_0}\, i] = [0\ 25\,\text{deg}\ 40\,\text{deg}\ 0\ 0\ 0\ 0]$

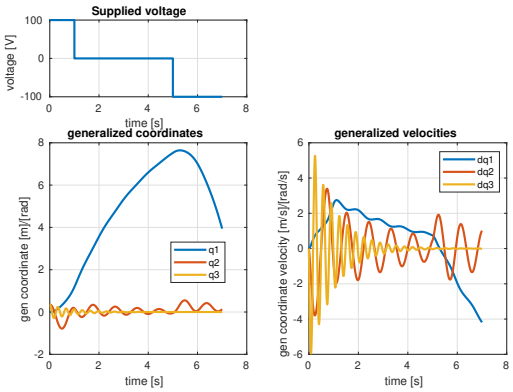
results that we would expect from a cart-pendulum system without input. From positive initial state values, we see oscillation around zero (due to a zero rest length of the spring, see Appendix for a plot with positive rest length and gravity) and convergence to zero of both $q_2$ and $q_3$ due to the damping in the joints and of the spring. Additionally it is of note that the cart moves slightly due to the positive initial angle $q_2$. This is entirely realistic and more importantly we see an initial positive increase of $q_1$ due to the initial positive angle of $q_2$.

Next I provide step inputs of the input voltage and simulate the cart-pendulum model as shown in Figure 4a. Notice that due to the growing torque on the wheels in the initial second of the simulation, the value of $q_2$ (the absolute angle of the diamond-shaped pendulum) flies towards the negative side and, while oscillating, tends to want to stay there due to the initial acceleration. When the voltage is back to zero, the position of the cart $q_1$ does not monotonically increase due to the friction of the wheels with the ground surface. Lastly, the sudden breaking (or negative torque) on the wheels results in a positive jerk on $q_2$ which is a logical conclusion from the negative acceleration. Due to the zero rest-length spring (which was deliberately chosen to be 0 as we will see later), $q_3$ has already damped out and when it is sufficiently zero, no input can make it increase again.

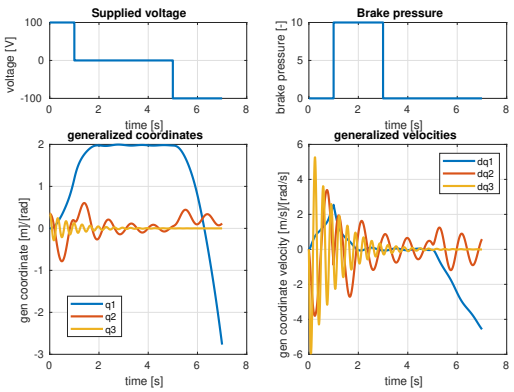
Next I provide additional brake action of the pneumatic brake as shown in Figure 4b Notice that while the brake action has already resulted in zero velocity of the cart, the cart does not drive backwards. This is because the pneumatic brake lowers the energy of the system. It does not supply opposite directive energy as the negative voltage does towards the end of the simulation. Also again we see excitation of $q_2$ that are in accordance with my expectations.

## 2 COMPUTING MODELLING

In this section I will consider that angular velocity sensors are placed at the joint of each of the first two links connecting them to the cart. With my aforementioned generalized coordinates $q_2$ and $q_3$ representing the rotation of the diamond shaped pendulum and the inner angle between beam 1 and 4 respectively, the sensors $s_1$ and $s_2$ subsequently measure the angular velocity $s_1$ *measures* $q_2 - q_3$ and $s_2$ *measures* $q_2 + q_3$. Both $s_1$




**(a)** Simulation of cart-pendulum model with input. $x_0 = [q_{1_0}\, q_{2_0}\, q_{3_0}\, \dot{q}_{1_0}\, \dot{q}_{2_0}\, \dot{q}_{3_0}\, i] = [0\ 20\text{deg}\ 20\text{deg}\ 0\ 0\ 0\ 0]$




**(b)** Simulation of cart-pendulum model with input, break. $x_0 = [q_{1_0}\, q_{2_0}\, q_{3_0}\, \dot{q}_{1_0}\, \dot{q}_{2_0}\, \dot{q}_{3_0}\, i] = [0\ 20\text{deg}\ 20\text{deg}\ 0\ 0\ 0\ 0]$

and $s_2$ produce an output of 0 when the angular velocity is within some safe bound $|\omega| < W$ and a 1 when the bound is exceeded. Importantly however, the sensor is low quality and has an error rate of 1 bit every 8 bits. It is my assumption then that in every 8 bits incoming, at most 1 bit is flipped and that the incoming 8 bits are ordered (i.e. 8 measurements are saved inside the sensor until they are all send to a post-processing device. In these 8 bits, at most 1 bit is flipped).

## 2.1 Certainty of having exceeded safety speed

Due to the added uncertainty, I can not simply state when that the bound has been exceeded after receiving one 1. Instead I will first create an NFA and a DFA that capture the language of all sequences at which I can be certain that the speed bound has been exceeded. First realize that for to be sure that the alarm has been flipped, two 1's should be observed in a series of 8 bits. When a new bit is considered, it can either be flipped or not and it can only be flipped if no other flipped bit has been considered thus far. To save on unnecessary complexion, I will limit the theoretical analysis of the NFA and DFA to an incoming series of 4 bits.

### 2.1.1 NFA

The beginning of our NFA will subsequently look according to Figure 15. It is important to address some important notation. For each state we observe a top row of $(\Sigma\Sigma\Sigma\Sigma)$ and a bottom row of $(\Sigma\Sigma\Sigma\Sigma)$ where $\Sigma \in \{0,1\}$. The top row indicates the incoming bits and the bottom row indicates whether the above bit is flipped or not (1 or 0 respectively). Next, a transition is made by a pair $[i,j]$ where $i$ indicates the value of the bit and $j$ on whether this bit is actually flipped or not. Subsequently, from the initial state I can take four possible transitions; $[1,1]$, $[0,1]$, $[1,0]$, and $[0,0]$. It is then logical that after either $[1,1]$ or $[0,1]$ I can only take two transitions for all subsequent steps; $[1,0]$ and $[0,0]$ because a flipped bit has already occurred. If I choose a transition in which no bit flip has occurred, I are left with four possible transitions for the next bit. This pattern keeps on repeating as shown in Figure 16.

It is important to notice that after transition $[0,0]$ I am left with 4 possible transitions instead of 2. If I then choose $[0,0]$ again I am again left with 4 possible transitions. This repeats itself another time as I only consider 4 incoming bits in this simplification. The diagram is clearly an NFA as I do not actually know whether a bit

is flipped or not (I have no information on $j$ in $[i, j]$) and subsequently I have two possible transitions for the same value of $i$ in $[i, j]$.

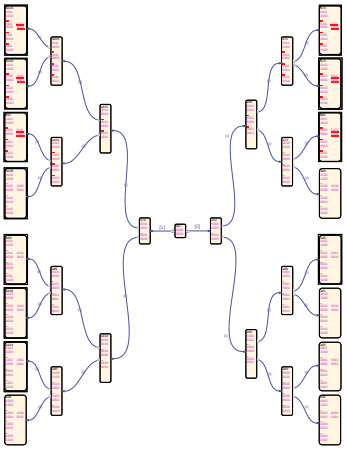
### 2.1.2 DFA

In order to transition the NFA into a DFA I straightforwardly incorporate all possible states after an arbitrary transition into the same state (i.e. I combine the states after $[1,1]$ and $[1,0]$ into a new DFA state. This limits our number of states. The DFA is shown in Figure 5a. In this DFA it is clear that after the first transition (either 0 or 1) the system can be in two different states (one where the bit is flipped and one where the bit is not flipped). At the next transition (either 0 or 1) I can be in three states, the first state with the bit flipped gets a non-flipped 0 or 1 and the first state with the bit not flipped gets either a flipped or a non-flipped 0 or 1. This repeats another two times as I only consider 4 incoming bits in this simplification. In the final state, I can be in 5 different states; the upper row is identical as the path to the final state is identical but the lower row is different with a flipped bit at either one of the four positions or no flipped bit at all. Final states are marked with a double box around the state which indicates that at these positions, one is certain that the safety limit has been exceeded (at every state inside the final DFA state, there exists a 1 bit without an accompanying flipped bit). The accepted language is

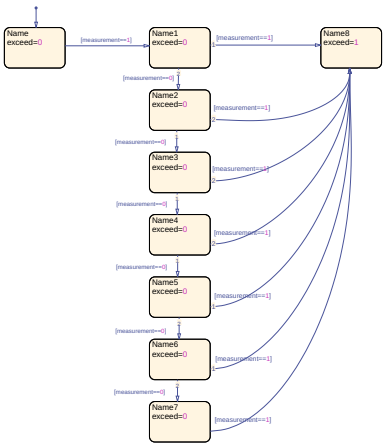
$$(11\Sigma\Sigma) \vee (1\Sigma1\Sigma) \vee (1\Sigma\Sigma1) \vee (\Sigma11\Sigma) \vee (\Sigma1\Sigma1) \vee (\Sigma\Sigma11) \quad \textit{where} \quad \Sigma \in \{0,1\} \tag{7}$$

### 2.1.3 Simulation

Detecting this double one pattern inside 8 considered bits is however quite easy and I subsequently drop the 4 bit simplification. The Simulink model is shown in Figure 5b. There is no reset present for simpler presentation but this can easily be incorporated by including a count (as will be shown in the next subsection).



**(a)** DFA of having exceeded the safety speed




**(b)** 8 bit string DFA simulation in Simulink as Moore machine

Simulation of this DFA is shown in Figure 6 in which an input sequence of 8 bits is given and an output signal is given below. Notice that once the second 1 bit has been processed, the output signal stays 1 since until the end of the 8 bit pattern, we are sure that the threshold has been exceeded.

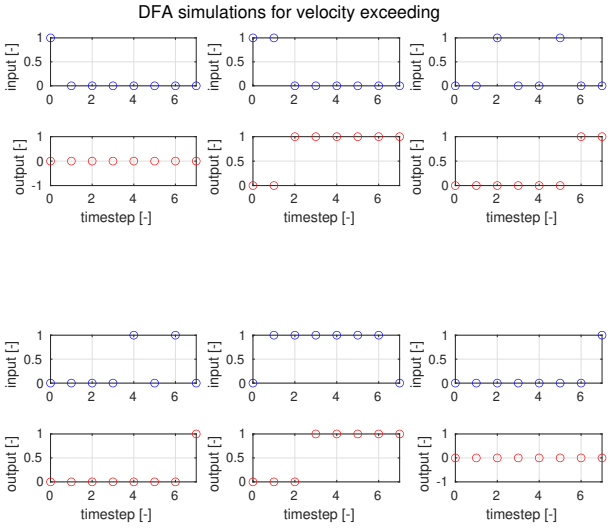



**Figure 6:** 8 bit DFA simulation in Simulink for different input bit sequences

## 2.2 Certainty of sounding an alarm

Next I will consider a post-processing system which will ring an alarm when three consecutive ones are measured. It is important to note that the alarm system is in between the sensor and the bit flipping stage. Therefore it is interesting to develop a machine which filters the signal from the sensor after the flipping stage which is able to tell us whether an alarm has been gone off surely (T), probably (D), and not (O).

First let us take a closer look at what we are interested in detecting. As 1 in 8 bits might be flipped, I can only say for sure that an alarm has gone of when 6 consecutive ones have been detected or two pairs of three have been detected;

$$11111100 \vee 01111110 \vee 00111111 \vee 11101110 \vee 01110111 \tag{8}$$

In these bit sequences it is impossible to place a zero at a one and not have a situation where we do not have three consecutive ones. For example, the following sequence does not necessarily lead to an alarm although it does have 6 ones;

$$110110110 \vee 011011011 \tag{9}$$

Lastly, a three-bit pattern consisting of at least two ones definitely has the probability of sounding an alarm (but not being sure about it);

$$011 \vee 110 \vee 101 \vee 111 \tag{10}$$

I subsequently divide our Mealy machine into three distinct phases

- Trying to detect a three bit pattern with at least two ones. This will bring one to a state in which the alarm might have sounded.
- Not being in the position to have 6 ones (i.e. the incoming bit string has started with more than two zeros).
- Detecting 6 ones in the bit string with the right order (comparing Equation 8 and Equation 9). This will bring one to a state in which the alarm definitely has sounded.

Furthermore, I bring some additional state variables into the problem to easier keep track of the occurred pattern and reduce the complexity of the Mealy machine;

- I keep track of the number of bits that have been processed in order to not have to write out the entire trace of possibilities and can instead just write a return transition with a count condition (every state has a condition in which one can return to the origin and assess a new string of bits).
- I keep track of the number of ones that have occurred in order to know when we can or cannot arrive to a situation in which the alarm has definitely rang.
- I can go backwards in the Mealy machine if two consecutive ones have been detected when I am still in the pattern detecting phase as this essentially resets the pattern detection.

### 2.2.1 Mealy machine

I arrive at a Mealy machine according to Figure 7 which, although unclear, indicates the three aforementioned phases of detecting the three bit pattern, never going to be sure about the alarm and being sure about the alarm. The top and bottom lines go back to the original state and are of the form

$$(s == 1 \vee s == 0) \ \& \ cnt == 7 \longrightarrow cnt = 0 \ \& \ ones = 0 \ \& \ T = i \tag{11}$$

where $s$ is the considered bit (signal), $cnt$ is the count keeping track of how many bits have been processed, $ones$ is the count keeping track of how many ones have been seen and $T$ is the output signal where instead of an output being of the form $T, D, O$ it is of the form $T = 0, 1, 2$ where 0 is no alarm, 1 is unsure, and 2 is sure. Zooming in on the begin phase of the Mealy machine in which we are to detect the three bit patterns we see on the left in Figure 7. The end phase is according to the right. Let us quickly zoom in on a final state in which we are sure that an alarm has sounded in Figure 18. In here we can see that we can either keep on receiving ones or zeros. This increases the count and the ones and the count respectively. The return condition requires a count of 7 which blocks unwanted returns to the origin before 8 bits have been considered (it is a Mealy machine so the 8th output occurs during the 8th input).

### 2.2.2 Simulation

I now provide the Mealy machine with strings of 8 bits. Some of which will definitely not sound the alarm (output = 0), sound the alarm with uncertainty (output = 1), and sound the alarm for sure (output = 2). Figure 8 shows us the simulation results. It can be seen that both a pattern from Equation 8 and Equation 9 is considered with an output as we would have expected. There are also multiple three-bit patterns detected inside the Mealy machine. We can subsequently conclude that our Mealy machine performs as desired given our simplifications.

## 3 HYBRID SYSTEM MODELLING

Next I will consider the situation where the joint masses of the pendulum can collide with each other and with the ground in Figure 13. Afterwards I will introduce a safety system which disables voltage input if bounds on angular velocity has been exceeded using the system developed in the previous section.

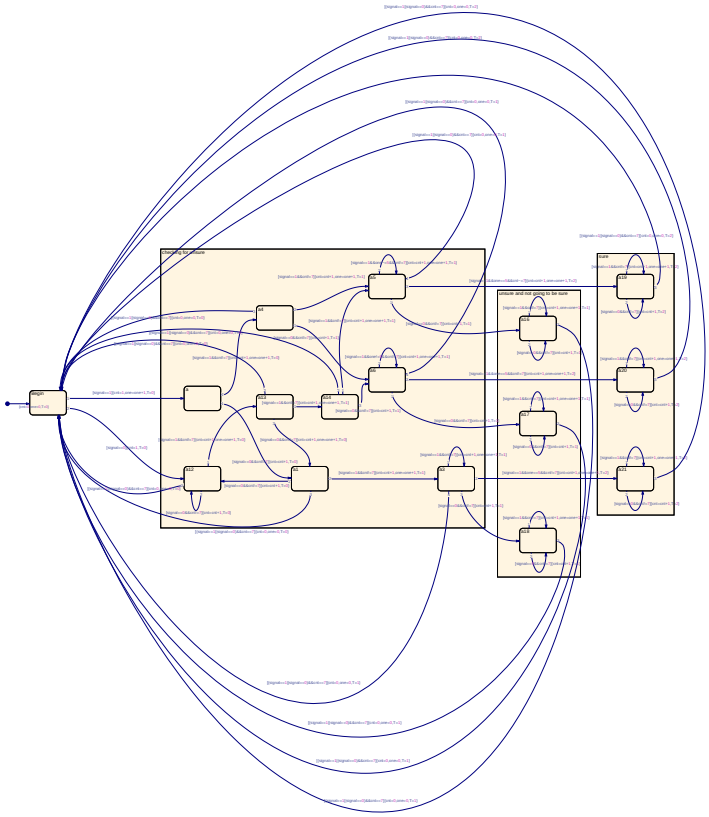


**Figure 7:** Whole Mealy machine for detecting whether an alarm has sounded with guarantee or not

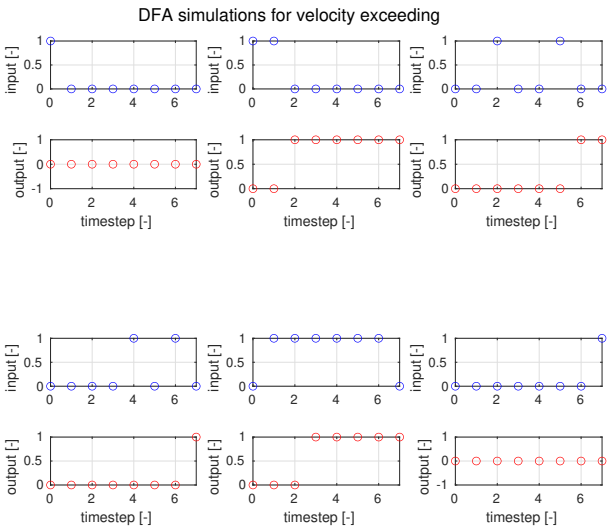



**Figure 8:** 8 bit Mealy machine simulation in Simulink for different input bit sequences

## 3.1 Impacts

### 3.1.1 Joint impacts

First consider the situation where joint 1 and joint 3 are able to collide inelastically (introducing an energy loss). Clearly when the angle $q_3$ crosses the zero line, it should not continue with its current velocity but instead return with a reduced velocity. This can be represented as

$$\textbf{IF:} \quad q_3 <= 0 \wedge \dot{q}_3 < 0 \quad \textbf{THEN:} \quad \dot{q}_3^+ := -\lambda \dot{q}_3^- \tag{12}$$

In which $\lambda$ is a dimensionless restitution constant indicating the energy loss between the joint surfaces after impact. I have also added the negative velocity requirement in order to ensure that after an impact, no immediate impact can occur again as the velocity switches sign. The impact is realized using *ode events* in MATLAB which halt an ode-solver when a certain event occurs (in this instance a positive to negative crossing of zero for $q_3$). Then the initial state gets reset and the ode-solver is allowed to continue until another impact occurs.

A straightforward simulation without control input is shown in Figure 9 which only shows the impact and nothing more There is now two energy dissipation sinks for the diamond-shaped pendulum; the joint damping

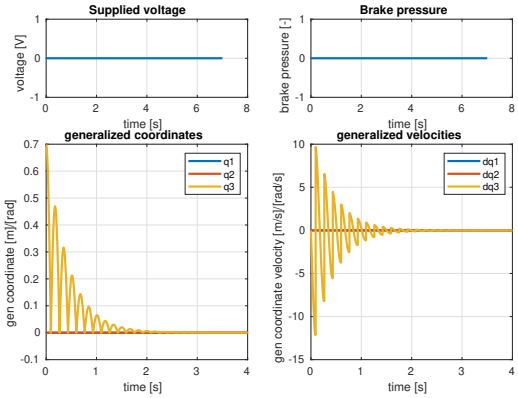



**Figure 9:** Simulation of cart-pendulum model with an impact on generalized coordinate $q_3$. Without input or break. $x_0 = [q_{1_0}\, q_{2_0}\, q_{3_0}\, \dot{q}_{1_0}\, \dot{q}_{2_0}\, \dot{q}_{3_0}\, i] = [0\ 0\ 40\text{deg}\ 0\ 0\ 0\ 0]$

and the impact energy loss.

### 3.1.2 Floor impacts

Similarly to the joint impact, the ground impacts can be modeled as *ode events* in MATLAB which detect when the crossing has occurred between either joint 1 and the left part of the ground (with respect to the cart) or joint 3 and the right part of the ground. Joint 1 and the right part of the ground and other combinations do not have to be checked since the joint impact between joint 1 and 3 already prohibits this from occurring. The reset for joint 1 can be represented, after some simple geometry rules, as

$$\textbf{IF:} \quad -cos(q_2 - \frac{q_3}{2}) l_{b_1} - h_{j_4} <= 0 \wedge -\dot{q}_2 + \dot{q}_3 < 0 \quad \textbf{THEN:} \quad \dot{q}_2^+ := -\lambda \dot{q}_2^- \wedge \dot{q}_3^+ := f(\dot{q}_3^-) \tag{13}$$

where $h_{j_4}$ is the height of joint 4 with respect to the ground and where $f()$ is a nonlinear function that further decreases $\dot{q}_3$ if $\dot{q}_3$ is negative and increases with some loss factor $\lambda$ if $\dot{q}_3$ is positive.

$$\dot{q}_3^+ = f(\dot{q}_3^-): \tag{14}$$

$$\dot{q}_3^+ = 1.2\dot{q}_3^- \quad \textbf{IF} \quad \dot{q}_3^- < 0 \tag{15}$$

$$\dot{q}_3^+ = -0.8\dot{q}_3^- \quad \textbf{IF} \quad \dot{q}_3^- >= 0 \tag{16}$$

$$\tag{17}$$

For joint 3 the reset is similar

$$\textbf{IF:} \quad -cos(q_2 + \frac{q_3}{2})l_{b_4} - h_{j_4} <= 0 \wedge \dot{q}_2 - \dot{q}_3 > 0 \quad \textbf{THEN:} \quad \dot{q}_2^+ := -\lambda\dot{q}_2^- \wedge \dot{q}_3^+ := f(\dot{q}_3^-) \tag{18}$$

A straightforward simulation without control input is shown in Figure 19. We observe that indeed that indeed the diamond-shaped pendulum is able to return after impact with the ground and does not fly through the ground.

## 3.2 Safety System

Next I add another extension in which the sensor readings from the angular velocity sensors determine whether the control voltage to the cart can remain on or not. If an angular velocity threshold for either beam 1 or beam 4 has been exceeded, the input voltage will be disabled for two seconds.

While I have already developed the Mealy machine which, based on incoming ones and zeros, establish when an alarm has sounded or not, I will have to sample the angular velocities of the beams at a sampling frequency of 1/100 Hz. Next I will add or subtract the angular velocity $q_3$ from $q_2$ (depending on which beam we are looking at), take the maximum value and compare it to the threshold. If the threshold is exceeded, the processing system will result in a 1, otherwise a 0. For the string of ones and zeros I introduce a 1 in 8 bit flip change per 8 bits via a MATLAB function using persistent variables (keeping track how many bits have passed and if a flip has been applied) shown in Figure 20.

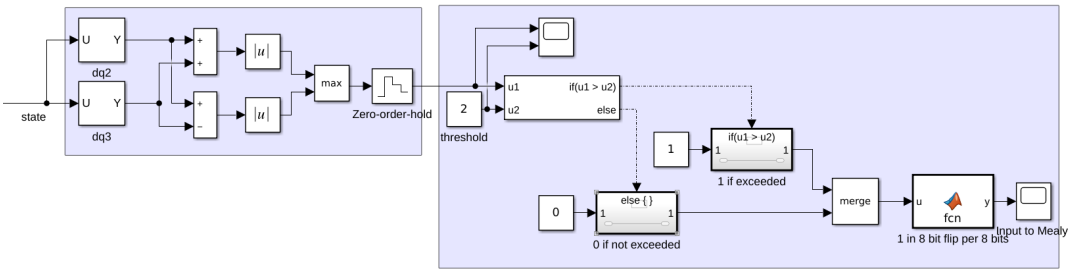



**Figure 10:** Discretization using Zero-order-hold and comparison. Input is the state, output is a 1 if an angular velocity bound has been exceeded and 0 otherwise

The string of bits will be sent to the previously developed Mealy machine where it will be assumed that the information arrives in 8 bits at a time. If the alarm was definitely enabled (i.e. a pattern in Equation 8

has occurred) and the output of the Mealy machine is thus 2, the voltage is disabled which is modeled as a subtraction of the same signal. The discretization with zero-order-hold and comparison is shown in Figure 20. A Simulink simulation result is shown in Figure 11. In this figure the disabling is longer than 2 seconds since the

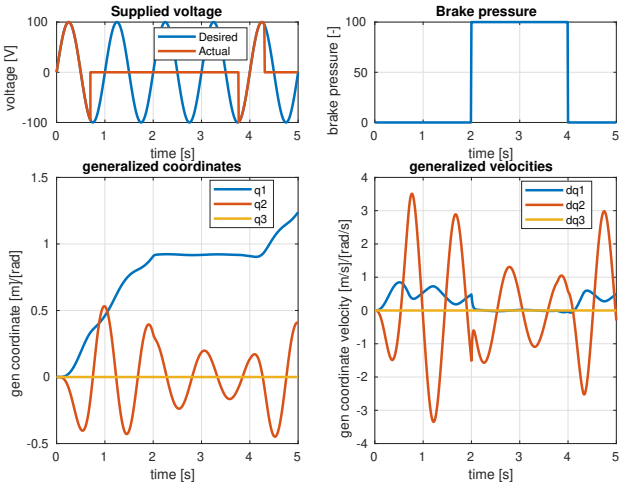



**Figure 11:** The safety system disabling the voltage input for multiple seconds after the angular velocity threshold has been exceeded

angular velocity threshold is exceeded some moments after the voltage input has already been disabled. As $q_3$ is deliberately set to zero, the threshold can be read from the generalized velocities plot when a angular velocity of 2 or -2 has been exceeded by $q_2$. The safety system works exactly as intended.

## 3.3 Hybrid automaton

Next for completeness I will describe the complete hybrid model and present it in the form of a hybrid automaton. I will consider the impacts, the safety system, and the control inputs (voltage and brake pressure).

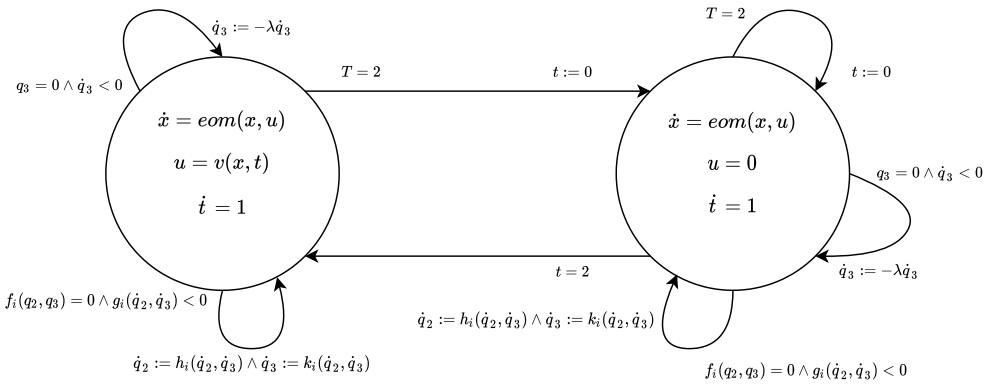



**Figure 12:** Hybrid automaton representation of the system incorporating impacts (left- and right ground impact are given identically with subscript $i$ in the figure), control input (both voltage and brake pressure are assumed to be incorporated into $u$), and the safety system

# A Appendix

## A.1 Cart-pendulum figure

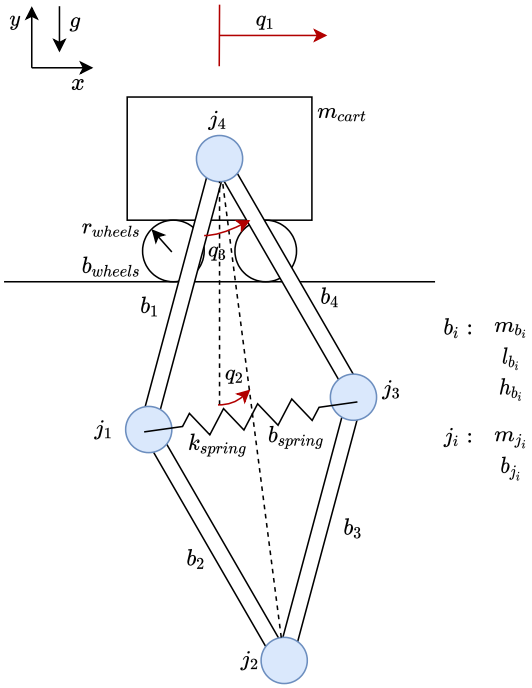



**Figure 13:** Cart with diamond-shaped pendulum

## A.2 Lagrange Equations of motion derivations

The Lagrangian function $L$ contains all physical information of a system on an energy level and is given by

$$L = K - P \tag{19}$$

where $K$ is the kinetic energy and $P$ is the potential energy. The equations of motion can then be obtained by applying the Euler-Lagrange equations according to

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = -\frac{\partial D}{\partial \dot{q}} + Q \tag{20}$$

where $q$ is the vector of generalized coordinates, $\dot{q}$ is its time derivative, $D$ is the dissipation term, and $Q$ is the external forces mapped to generalized coordinates is a mapping from control input to generalized coordinates.

First let us define $q$, the vector of generalized coordinates. $q$ is by definition a minimal set of coordinates which can uniquely describe the configuration of the system at hand. As such, and based on the assumption that $l_{b_1} = l_{b_4}$ and $l_{b_2} = l_{b_3}$ we can find three generalized coordinates $q_1$, $q_2$, and $q_3$ as shown in Figure 13. Based

on these generalized coordinates we can describe the center of mass coordinates of the cart, joints, and beams

$$p_{cart} = f_{cart}(q) \qquad p_{j_i} = f_{j_i}(q) \qquad p_{b_i} = f_{b_i}(q) \tag{21}$$

and the velocities as summation of partial derivatives with respect to the generalized coordinates

$$\dot{p}_{cart} = \sum_{k=1}^{3} \frac{\partial p_{cart}}{\partial q_k} \dot{q}_k \qquad \dot{p}_{j_i} = \sum_{k=1}^{3} \frac{\partial p_{j_i}}{\partial q_k} \dot{q}_k \qquad \dot{p}_{b_i} = \sum_{k=1}^{3} \frac{\partial p_{b_i}}{\partial q_k} \dot{q}_k \tag{22}$$

We can now define the kinetic energy $K$ as the summation of individual terms of rotational and translational kinetic energy of the individual terms

$$K = \frac{1}{2} m_{cart} \dot{p}_{cart,x}^2 + \sum_{i=1}^{4} (\frac{1}{2} m_{j_i} \dot{p}_{j_i,x}^2 + \frac{1}{2} m_{j_i} \dot{p}_{j_i,y}^2) + \sum_{i=1}^{4} (\frac{1}{2} m_{b_i} \dot{p}_{b_i,x}^2 + \frac{1}{2} m_{b_i} \dot{p}_{b_i,y}^2 + \frac{1}{2} I_{b_i} \dot{p}_{b_i,z}^2) \tag{23}$$

where $I_{b_i}$ is the moment of inertia of $b_i$ around the center of mass of the beam around the z-axis (coming out of the paper) which can straightforwardly be computed by the aforementioned uniformly distributed mass as

$$I_{b_i} = \frac{1}{12} m_{b_i} (l_{b_i}^2 + h_{b_i}^2) \tag{24}$$

for which we can conclude that, due to the equal lengths $I_{b_1} = I_{b_4}$ and $I_{b_2} = I_{b_3}$.

The potential energy $V$ can be defined as the summation of potential energy of the joints and beams and the spring connecting $j_1$ and $j_3$

$$V = \sum_{i=1}^{3} m_{j_i} g p_{j_i,y} + \sum_{i=1}^{4} m_{b_i} g p_{b_i,y} + \frac{1}{2} k_{spring} (2 l_{b_1} \sin(\frac{q_3}{2}) - l_0)^2 \tag{25}$$

where $g$ is the acceleration due to gravity in $m/s^2$, $k_{spring}$ is the spring constant in $N/m$ and $l_0$ is the spring slack length in $m$.

The dissipation term $D$ can be computed by considering the friction on the wheels, the friction in the joints, and the damping of the spring.

$$D = \frac{1}{2} \dot{q}^T \begin{bmatrix} b_{wheels} & 0 & 0 \\ 0 & b_{j_4} & 0 \\ 0 & 0 & b_{j_1} + b_{j_2} + b_{j_3} \end{bmatrix} \dot{q} + b_{spring} (l_{b_1} \cos(\frac{q_3}{2}) \dot{q}_3)^2 \tag{26}$$

where $b_x$ is the damping coefficient of component $x$.

The external force in $Q$ can be described by a momentary expansion of the generalized coordinates with the current $I$ (this will become more clear later on). Assuming the current $I$ is known to the system, $Q$ is given as

$$Q = \begin{bmatrix} \frac{k_{DC}}{r_{wheels}} I \\ 0 \\ 0 \end{bmatrix} \tag{27}$$

applying the derivatives according to Equation 20 allows us to find an equation which linearly depends on the acceleration of the generalized coordinates $\ddot{q}$. Using the MATLAB command `equationsToMatrix` we can obtain the linear combination of $A\ddot{q} = b$. The equations of motion can then be obtained by

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dot{I} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ A \backslash b \\ -\frac{R}{L} I + \frac{1}{L} V \end{bmatrix} \tag{28}$$

which is of the form $\dot{x} = Ax + Bu$ with the state x momentarily being equal to

$$x = \begin{bmatrix} q \\ \dot{q} \\ I \end{bmatrix} \tag{29}$$

and input voltage $V$. I assume the reader is somewhat familiar with these conversions and methodologies. For more information on the derivation of the equations of motions see the Appendix.

Lastly, we can add a hydraulic brake on the wheels by adding to the $D$ term a pressure dependent damping according to

$$D = D + \rho \dot{q}_1^2 P \tag{30}$$

where $\rho$ is the pressure to dissipation coefficient (dependent among others on the friction coefficient between brake disk and brake claw) and $P$ is the supplied brake pressure. Contrary to supplying the DC motor with a negative voltage, which will slow down the cart and make it move in the other direction, supplying pressure to the hydraulic brake will bring the system to a total standstill.

## A.3 Parameters

| **Parameter** | **Shorthand** | **Value** | **Unit** |
|---|---|---|---|
| Beam mass | $m_{b_i}$ | 0.5 | $kg$ |
| Beam length | $l_{b_i}$ | 0.2 | $m$ |
| Beam height | $h_{b_i}$ | 0.02 | $m$ |
| Cart mass | $m_{cart}$ | 4 | $kg$ |
| Cart wheel radius | $r_{wheels}$ | 0.05 | $m$ |
| Cart wheel friction | $b_{wheels}$ | 0.1 | – |
| Joint mass | $m_{j_i}$ | 0.1 | $k$ |
| Joint damping | $b_{j_i}$ | 0.05 | – |
| Spring stiffness | $k_{spring}$ | 100 | $N/m$ |
| Spring damping | $b_{spring}$ | 0.1 | – |
| Acceleration gravity | $g$ | 9.81 | $m/s^2$ |
| DC motor resistance | $R$ | 1000 | $\Omega$ |
| DC motor inductance | $L$ | 100 | $H$ |
| DC motor constant | $k_{DC}$ | 10 | $Nm/A$ |

## A.4 Results with non-zero spring length

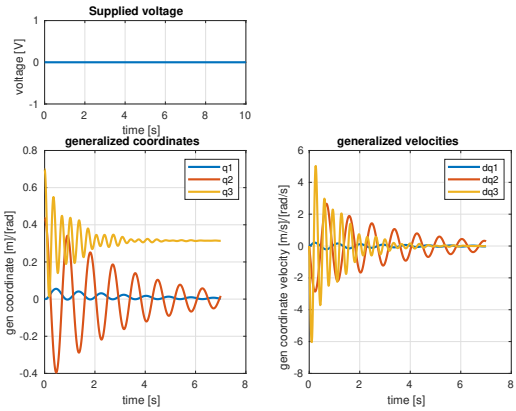



**Figure 14:** Simulation of cart-pendulum model without input and with initial state $x_0 = [q_{10} q_{20} q_{30} \dot{q}_{10} \dot{q}_{20} \dot{q}_{30} i] = [025\,\text{deg}\,40\,\text{deg}\,0000]$ and positive spring rest length $l_0$

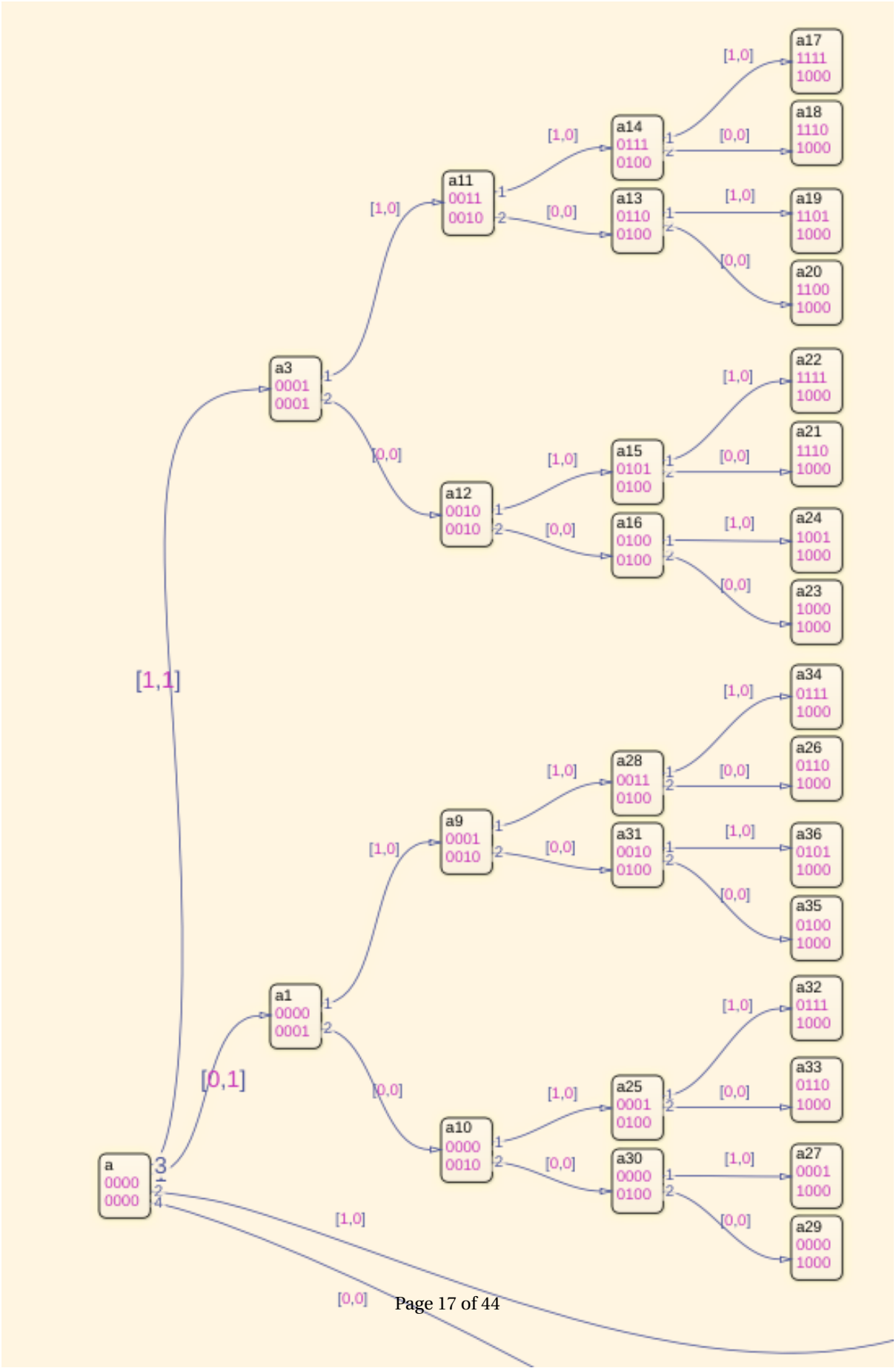





**Figure 15:** beginning of the NFA having exceeded the safety speed

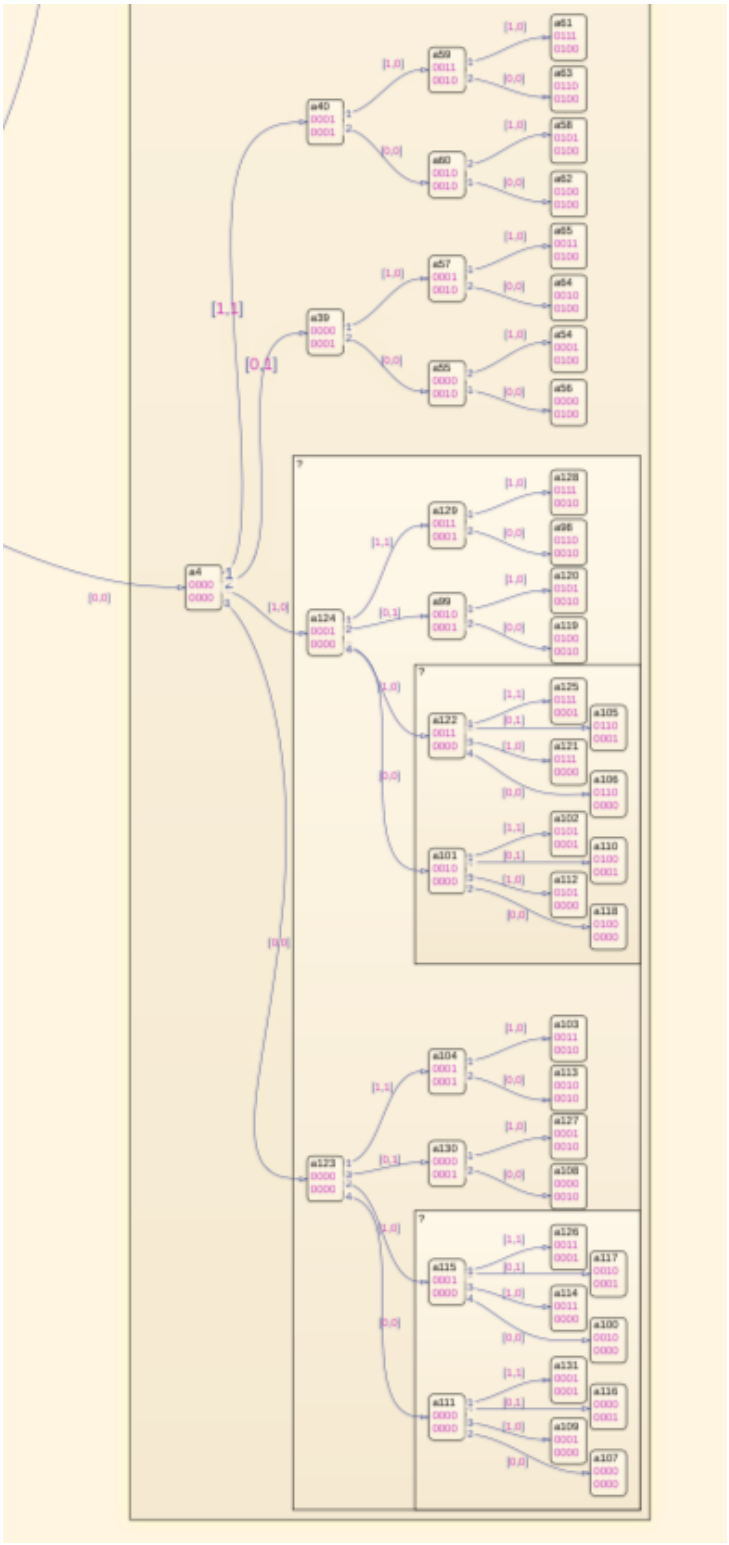


**Figure 16:** end of the NFA having exceeded the safety speed

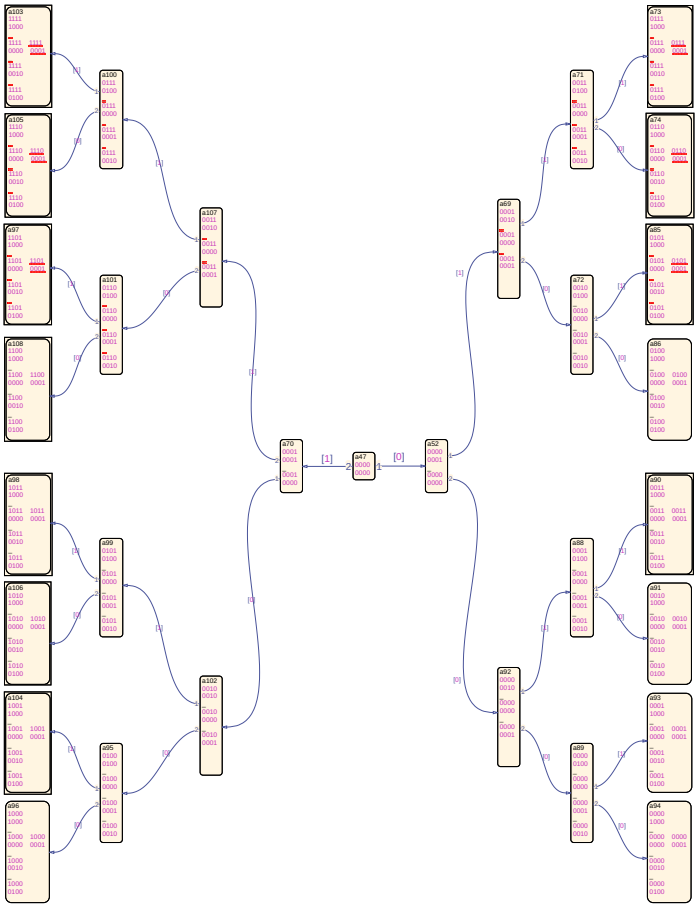


**Figure 17:** DFA of having exceeded the safety speed

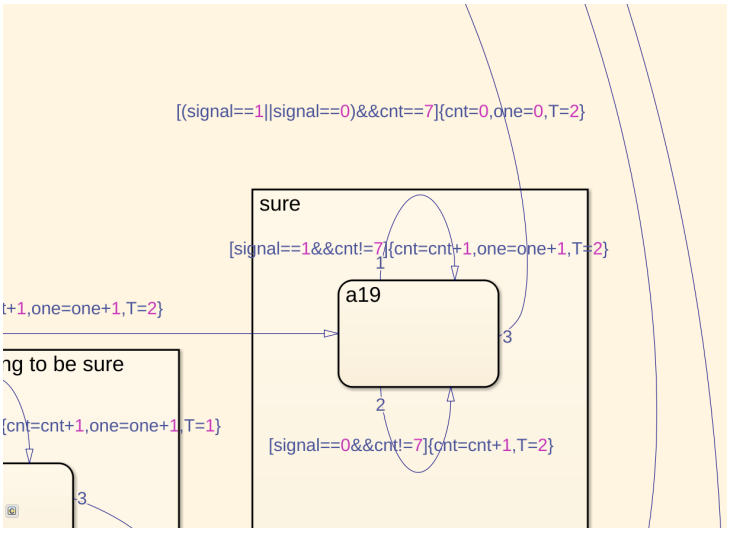



**Figure 18:** Closeup of a final state in the Mealy machine

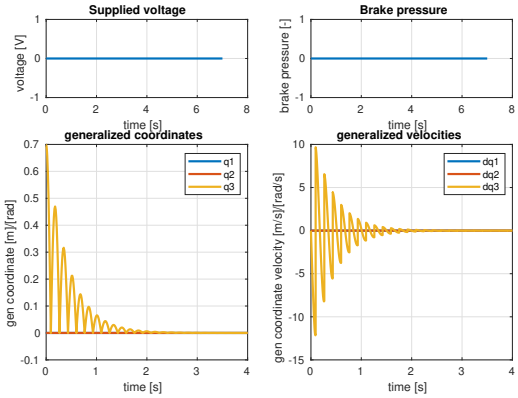



**Figure 19:** Simulation of cart-pendulum model with an impact on generalized coordinate $q_2$ and $q_3$ via impact with the ground of joint 3. Without input and break, and with initial state $x_0 = [q_{1_0}\, q_{2_0}\, q_{3_0}\, \dot{q}_{1_0}\, \dot{q}_{2_0}\, \dot{q}_{3_0}\, i] = [00\,\mathrm{deg}\,0\,\mathrm{deg}\,020\,rad/s00]$

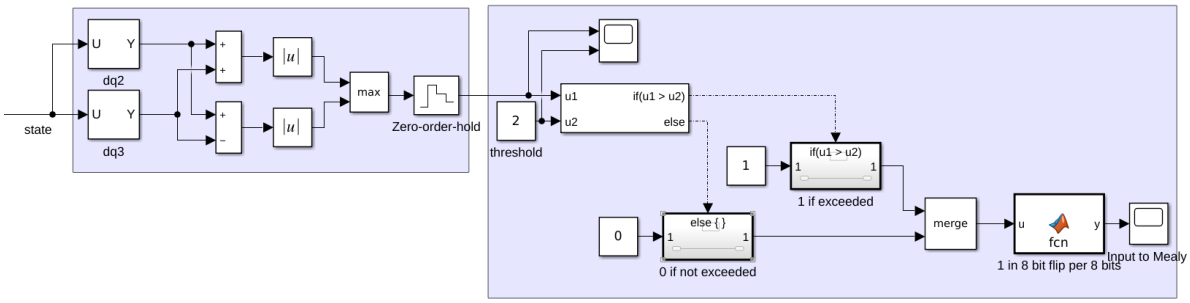



**Figure 20:** Discretization using Zero-order-hold and comparison. Input is the state, output is a 1 if an angular velocity bound has been exceeded and 0 otherwise

# B MATLAB CODE

## B.1 main

```
% close all;
clear all;

%% symbolic variables
syms g t real
syms mCart bWheels rWheels real

syms bJoint1 bJoint2 bJoint3 bJoint4 real

syms mJoint1 mJoint2 mJoint3 real
mJoint = [mJoint1 mJoint2 mJoint3]';

syms mBeam1 mBeam2 mBeam3 mBeam4 real
mBeam = [mBeam1 mBeam2 mBeam3 mBeam4]';

syms lBeam1 lBeam2 lBeam3 lBeam4 real
lBeam = [lBeam1 lBeam2 lBeam3 lBeam4]';

syms hBeam bBeam real
syms kSpring bSpring real

syms R L F real

syms icur real % integral of the current, artificial state
syms q1 q2 q3 cur vol pres real
syms dq1 dq2 dq3 dcur real
syms ddq1 ddq2 ddq3 ddcur real

%% symbolic derivations
IBeam1 = 1/12*mBeam1*(lBeam1^2 + hBeam^2);
IBeam2 = 1/12*mBeam2*(lBeam2^2 + hBeam^2);
IBeam3 = 1/12*mBeam3*(lBeam3^2 + hBeam^2);
IBeam4 = 1/12*mBeam4*(lBeam4^2 + hBeam^2);
IBeam = [IBeam1 IBeam2 IBeam3 IBeam4]';

%% define generalized coordinates
q = [q1 q2 q3]';
dq = [dq1 dq2 dq3]';
ddq = [ddq1 ddq2 ddq3]';
```

```
40
41 %% position of c.o.m. and their derivatives
42 comPositions;
43
44 %% kinetic energy
45 T = 0;
46
47 % kinetic energy cart
48 T = T + 1/2*mCart*dcom.cart.x^2;
49 T = T + 1/2*mCart*dcom.cart.y^2;
50
51 % kinetic energy beams
52 for i = 1:4
53     T = T + 1/2*mBeam(i)*dcom.beam.x(i)^2;
54     T = T + 1/2*mBeam(i)*dcom.beam.y(i)^2;
55     T = T + 1/2*IBeam(i)*dcom.beam.p(i)^2;
56 end
57
58 for i = 1:3
59     T = T + 1/2*mJoint(i)*dcom.joint.x(i)^2;
60     T = T + 1/2*mJoint(i)*dcom.joint.y(i)^2;
61 end
62
63 %% potential energy
64 V = 0;
65
66 % potential energy beams
67 for i = 1:4
68     V = V + mBeam(i)*g*com.beam.y(i);
69 end
70
71 % potential energy joints
72 for i = 1:3
73     V = V + mJoint(i)*g*com.joint.y(i);
74 end
75
76 % potential energy spring
77 l0 = 0.0;
78 V = V + 1/2*kSpring*(2*lBeam1*sin(q3/2)-l0)^2;
79
80 %% damping
81 B = diag([2.0,0.05,0.01]);
```

```
D = 1/2*dq'*B*dq + ...
    0.1*(lBeam1*cos(q3/2)*dq3)^2 + ...
    1*dq1^2*pres; % + ...
%     1/2*cur*R*cur;

% W = Wdamping + Wfriction;
% Q = jacobian(Wdamping, dq)';
%% external forces
Q = sym(zeros(length(q),1));

R = 1000;
L = 100;
kMotor = 10;
rWheels = 0.05;
Q(1) = Q(1) + (kMotor/rWheels)*cur;

% Q(3) = Q(3) + Wdamping;

%% motor equations
dT_ddq = jacobian(T,dq).';
ddT_ddqdt = jacobian(dT_ddq,t) + ...
            jacobian(dT_ddq,q)*dq + ...
            jacobian(dT_ddq,dq)*ddq;

dT_dq = jacobian(T,q).';

dV_dq = jacobian(V,q).';

dD_ddq = jacobian(D,dq).';

EoM = simplify(ddT_ddqdt - dT_dq + dV_dq - Q + dD_ddq);

[A,b] = equationsToMatrix(EoM,ddq);
A = simplify(A);
b = simplify(b);

%% values for system properties
mBeam1 = 0.5; mBeam2 = 0.5; mBeam3 = 0.5; mBeam4 = 0.5;
mCart = 4;
mJoint1 = 0.1; mJoint2 = 0.1; mJoint3 = 0.1;
lBeam1 = 0.2; lBeam2 = 0.2; lBeam3 = 0.2; lBeam4 = 0.2;
rJoint1 = 0.05; rJoint2 = 0.05; rJoint3 = 0.05;
```

```
hBeam = 0.02; bBeam = 0.02;
kSpring = 50; bSpring = 100;
bJoint1 = 0.6; bJoint2 = 0.6; bJoint3 = 0.6; bJoint4 = 0.6;
g = 9.81;

A = subs(A);
b = subs(b);

%% create function to evaluate eom
matlabFunction([dq; simplify(A\b); -R*cur/L + vol/L], ...
    'Vars',{t,[q; dq; cur],vol,pres}, ...
    'File', 'evaluateEoM');

matlabFunction([-R*cur/L + vol/L; 0], ...
    'Vars',{t,[cur;dcur],vol}, ...
    'File', 'evaluateEoDC');

% simulate DC motor
y0 = [0  0]';
tspan = [0  5];
vInput = [100 0 0 -100];
tInput = [1 2 2 2];

tDC = []; yDC = [];
for i = 1:length(vInput)
    t1 = 0;
    tspan = [t1(end) tInput(i)+t1(end)];
    [t1,y1] = ode45(@(t,y)evaluateEoDC(t,y,vInput(i)),tspan,y0);
    y0 = y1(end,:);
    % add time and output to total simulation results
    try
        tDC = [tDC; t1+tDC(end)];
    catch
        tDC = t1;
    end
    yDC = [yDC; y1];
end

figure
plot(tDC,yDC(:,1))
grid on
xlabel('time [s]')
```

```
166 ylabel('current [A]')
167
168 %% initial conditions
169 q10 = 0;                    % position cart
170 q20 = deg2rad(0);
171 q30 = deg2rad(0);
172 %      q1   q2   q3   dq1 dq2 dq3 i
173 y0 = [q10 q20 q30 0    20    0    0]';
174
175 %% simulate ODE
176 t_end = 7;
177 impact = 1;
178 input = 1;
179
180 %%%%%%% 1
181 %      tspan = [0 t_end];
182 %      [t,y] = ode45(@(t,y)evaluateEoM(t,y,0,0),tspan,y0);
183
184
185 %%%%%%% 2
186 % %      vInput = [100 0 0    -100];
187 % %      pInput = [0    10 0 0];
188 % %      tInput = [1 2 2 2];
189 % %      t = []; y = [];
190 % %      for i = 1:length(vInput)
191 % %          t1 = 0;
192 % %          tspan = [t1(end) tInput(i)+t1(end)];
193 % %          [t1,y1] = ode45(@(t,y)evaluateEoM(t,y,vInput(i),pInput(i)),tspan,y0);
194 % %          y0 = y1(end,:);
195 % %          % add time and output to total simulation results
196 % %          try
197 % %              t = [t; t1+t(end)];
198 % %          catch
199 % %              t = t1;
200 % %          end
201 % %          y = [y; y1];
202 % %      end
203
204 %%%%%%% 3
205      opt = odeset('Events',@impacts);
206      t = []; y = [];
207      t1 = 0;
```

```
208     while t1(end) ~= 4
209         tspan = [t1(end) 4];
210         [t1,y1,te,ye,ie] = ode45(@(t,y)evaluateEoM(t,y,0,0),tspan,y0,opt);
211 
212         % reset initial condition
213         y0 = y1(end,:);
214         if all(ie == 1) || all(ie == 2)
215             lambdaRoof = 0.9;
216             y0(5) = -lambdaRoof*y0(5);
217             if y0(6) < 0
218                 y0(6) = 1.2*y0(6);
219             else
220                 y0(6) = -0.8*y0(6);
221             end
222         elseif all(ie == 3)
223             lambdaJoints = 0.8;
224             y0(6) = -lambdaJoints*y0(6);
225         end
226 
227         % add time and output to total simulation results
228         try
229             t = [t; t1];
230         catch
231             t = t1;
232         end
233         y = [y; y1];
234     end
235 
236 %%%%%%% 4
237 % opt = odeset('Events',@impacts_and_discrete);
238 % t = []; y = []; in = [];
239 % alarm = false;
240 % t1 = 0;
241 % while t1(end) ~= 4
242 %     tspan = [t1(end) floor(t1(end)*100)/100+0.01];
243 %     [t1,y1,te,ye,ie] = ode45(@(t,y)evaluateEoM(t,y,0,0),tspan,y0,opt);
244 %
245 %     % reset initial condition
246 %     y0 = y1(end,:);
247 %     if all(ie == 1) || all(ie == 2)
248 %         lambdaRoof = 0.9;
249 %         y0(5) = -lambdaRoof*y0(5);
```

```
%     elseif all(ie == 3)
%         lambdaJoints = 0.8;
%         y0(6) = -lambdaJoints*y0(6);
%     end
%
%     if abs(y0(5)+y0(6)) > 1 || abs(y0(5)-y0(6)) > 1
%         in = [in; 1];
%     else
%         in = [in; 0];
%     end
%     inTS = [linspace(0,length(in)/100,length(in))' in];
%     out = sim('mealy_moore',t_end);
%     if out.two_2.Data(end) == 2
%         alarm = true;
%     else
%         alarm = false;
%     end
%
%    % add time and output to total simulation results
%     try
%         t = [t; tl];
%     catch
%         t = tl;
%     end
%     y = [y; yl];
% end

%% visualization

figure
subplot(3,2,1)
% plot([0 1 1.0001 3 3.0001 5 5.0001 7],[100 100 0 0 0 0 -100 -100],'LineWidth',2)
plot([0 7],[0 0],'LineWidth',2)
grid on
xlabel('time [s]')
ylabel('voltage [V]')
title('Supplied voltage')
set(gca,'FontSize',10)

subplot(3,2,2)
% plot([0 1 1.0001 3 3.0001 7],[0 0 10 10 0 0],'LineWidth',2)
plot([0 7],[0 0],'LineWidth',2)
```

```
grid on
xlabel('time [s]')
ylabel('brake pressure [-]')
title('Brake pressure')
set(gca,'FontSize',10)


subplot(3,2,[3 5])
plot(t,y(:,1),'LineWidth',2)
hold on
plot(t,y(:,2),'LineWidth',2)
plot(t,y(:,3),'LineWidth',2)
grid on
legend('q1','q2','q3')
xlabel('time [s]')
ylabel('gen coordinate [m]/[rad]')
title('generalized coordinates')
set(gca,'FontSize',10)

subplot(3,2,[4 6])
plot(t,y(:,4),'LineWidth',2)
hold on
plot(t,y(:,5),'LineWidth',2)
plot(t,y(:,6),'LineWidth',2)
grid on
legend('dq1','dq2','dq3')
xlabel('time [s]')
ylabel('gen coordinate velocity [m/s]/[rad/s]')
title('generalized velocities')
set(gca,'FontSize',10)


if false
    figure
    plot(t,-cos(y(:,2) - y(:,3)/2)*lBeam1)
    hold on
    plot(t,-cos(y(:,2) + y(:,3)/2)*lBeam4)
    plot(t,y(:,3) - (atan(rJoint1/lBeam1) + atan(rJoint3/lBeam4)))
    legend('joint1','joint3','joint1joint3')

    figure(3)
    for i = 1:length(y)
```

```
334             if mod(i,round(length(y)/100)) == 0
335                 q1 = y(i,1);
336                 q2 = y(i,2);
337                 q3 = y(i,3);
338 
339                 cartX = q1;
340                 cartY = 0;
341                 alpha = pi/2 - (2*pi - 2*q3)/4;
342                 joint1X = cartX + sin(q2 - q3/2)*lBeam1;
343                 joint1Y = cartY - cos(q2 - q3/2)*lBeam1;
344                 joint2X = joint1X + sin(q2 + alpha)*lBeam2;
345                 joint2Y = joint1Y - cos(q2 + alpha)*lBeam2;
346 
347                 joint3X = cartX + sin(q2 + q3/2)*lBeam4;
348                 joint3Y = cartY - cos(q2 + q3/2)*lBeam4;
349 
350                 figure(3)
351                 clf
352                 plot([cartX;joint1X],[cartY;joint1Y],'b')
353                 hold on
354                 plot([joint1X;joint2X],[joint1Y;joint2Y],'b')
355                 plot([cartX;joint3X],[cartY;joint3Y],'b')
356                 plot([joint3X;joint2X],[joint3Y;joint2Y],'b')
357                 xlim([-5 5])
358                 ylim([-0.5 0.5])
359                 drawnow
360             end
361         end
362 end
363 
364 
365 %%% get part 2 info
366 % figure(10)
367 % subplot(2,1,1)
368 % plot(out.in.Time,out.in.Data,'LineWidth',2)
369 % subplot(2,1,2)
370 % plot(out.two_1.Time,out.two_1.Data,'LineWidth',2)
371 % xlabel('time [s]')
372 % ylabel('output signal [-]')
373 %
374 % figure(11)
375 % subplot(2,1,1)
```

```
% plot(out.in.Time,out.in.Data,'LineWidth',2)
% subplot(2,1,2)
% plot(out.two_2.Time,out.two_2.Data,'LineWidth',2)
% xlabel('time [s]')
% ylabel('output signal [-]')
```

## B.2 comPositions

```
com.cart.x = ql;
com.cart.y = 0;
com.cart.p = 0;

com.beam.x(1) = com.cart.x + sin(q2 - q3/2)*lBeam1/2;
com.beam.y(1) = com.cart.y - cos(q2 - q3/2)*lBeam1/2;
com.beam.p(1) = q2 - q3/2;

com.joint.x(1) = com.cart.x + sin(q2 - q3/2)*lBeam1;
com.joint.y(1) = com.cart.y - cos(q2 - q3/2)*lBeam1;
com.joint.p(1) = 0;

alpha = pi/2 - (2*pi - 2*q3)/4;

com.beam.x(2) = com.joint.x(1) + sin(q2 + alpha)*lBeam2/2;
com.beam.y(2) = com.joint.y(1) - cos(q2 + alpha)*lBeam2/2;
com.beam.p(2) = q2 + alpha;

com.joint.x(2) = com.joint.x(1) + sin(q2 + alpha)*lBeam2;
com.joint.y(2) = com.joint.y(1) - cos(q2 + alpha)*lBeam2;
com.joint.p(2) = 0;

com.beam.x(4) = com.cart.x + sin(q2 + q3/2)*lBeam4/2;
com.beam.y(4) = com.cart.y - cos(q2 + q3/2)*lBeam4/2;
com.beam.p(4) = q2 + q3/2;

com.joint.x(3) = com.cart.x + sin(q2 + q3/2)*lBeam4;
com.joint.y(3) = com.cart.y - cos(q2 + q3/2)*lBeam4;
com.joint.p(3) = 0;

com.beam.x(3) = com.joint.x(3) + sin(q2 - alpha)*lBeam3/2;
com.beam.y(3) = com.joint.y(3) - cos(q2 - alpha)*lBeam3/2;
com.beam.p(3) = q2 - alpha;

dcom.cart.x = sym(0); dcom.cart.y = sym(0); dcom.cart.p = sym(0);
```

```
dcom.beam.x = sym(zeros(4,1));
dcom.beam.y = sym(zeros(4,1));
dcom.beam.p = sym(zeros(4,1));
dcom.joint.x = sym(zeros(3,1));
dcom.joint.y = sym(zeros(3,1));
dcom.joint.p = sym(zeros(3,1));

for i = 1:length(q)
    dcom.cart.x = dcom.cart.x + diff(com.cart.x,q(i))*dq(i);

    for ii = 1:4
        dcom.beam.x(ii) = dcom.beam.x(ii) + diff(com.beam.x(ii),q(i))*dq(i);
        dcom.beam.y(ii) = dcom.beam.y(ii) + diff(com.beam.y(ii),q(i))*dq(i);
        dcom.beam.p(ii) = dcom.beam.p(ii) + diff(com.beam.p(ii),q(i))*dq(i);
    end
    for ii = 1:3
        dcom.joint.x(ii) = dcom.joint.x(ii) + diff(com.joint.x(ii),q(i))*dq(i);
        dcom.joint.y(ii) = dcom.joint.y(ii) + diff(com.joint.y(ii),q(i))*dq(i);
        dcom.joint.p(ii) = dcom.joint.p(ii) + diff(com.joint.p(ii),q(i))*dq(i);
    end
end
```

## B.3 impacts

```
function [value, isterminal,direction] = impacts(t,y)
lBeam1 = 0.2; lBeam2 = 0.2; lBeam3 = 0.2; lBeam4 = 0.2;
rJoint1 = 0.05; rJoint2 = 0.05; rJoint3 = 0.05;
hBeam = 0.02; bBeam = 0.02;

value = [-cos(y(2) - y(3)/2)*lBeam1;
         -cos(y(2) + y(3)/2)*lBeam4;
%          -cos(y(2) + alpha)*lBeam2/2 - cos(y(2) + alpha)*lBeam2;
%          -cos(y(2) + alpha)*lBeam2/2 - cos(y(2) + alpha)*lBeam2;
         y(3)];% - (atan(rJoint1/lBeam1) + atan(rJoint3/lBeam4))];

isterminal = [1;
              1;
              1];

direction = [1;
             -1;
             -1];
end
```

## B.4 prerun

```
close all; clear all;

%% symbolic variables
syms g t real
syms mCart bWheels rWheels real

syms bJoint1 bJoint2 bJoint3 bJoint4 real

syms mJoint1 mJoint2 mJoint3 real
mJoint = [mJoint1 mJoint2 mJoint3]';

syms mBeam1 mBeam2 mBeam3 mBeam4 real
mBeam = [mBeam1 mBeam2 mBeam3 mBeam4]';

syms lBeam1 lBeam2 lBeam3 lBeam4 real
lBeam = [lBeam1 lBeam2 lBeam3 lBeam4]';

syms hBeam bBeam real
syms kSpring bSpring real

syms R L F real

syms icur real % integral of the current, artificial state
syms q1 q2 q3 cur vol pres real
syms dq1 dq2 dq3 dcur real
syms ddq1 ddq2 ddq3 ddcur real

%% symbolic derivations
IBeam1 = 1/12*mBeam1*(lBeam1^2 + hBeam^2);
IBeam2 = 1/12*mBeam2*(lBeam2^2 + hBeam^2);
IBeam3 = 1/12*mBeam3*(lBeam3^2 + hBeam^2);
IBeam4 = 1/12*mBeam4*(lBeam4^2 + hBeam^2);
IBeam = [IBeam1 IBeam2 IBeam3 IBeam4]';

%% define generalized coordinates
q = [q1 q2 q3]';
dq = [dq1 dq2 dq3]';
ddq = [ddq1 ddq2 ddq3]';

%% position of c.o.m. and their derivatives
```

```
41 comPositions;
42
43 %% kinetic energy
44 T = 0;
45
46 % kinetic energy cart
47 T = T + 1/2*mCart*dcom.cart.x^2;
48 T = T + 1/2*mCart*dcom.cart.y^2;
49
50 % kinetic energy beams
51 for i = 1:4
52     T = T + 1/2*mBeam(i)*dcom.beam.x(i)^2;
53     T = T + 1/2*mBeam(i)*dcom.beam.y(i)^2;
54     T = T + 1/2*IBeam(i)*dcom.beam.p(i)^2;
55 end
56
57 for i = 1:3
58     T = T + 1/2*mJoint(i)*dcom.joint.x(i)^2;
59     T = T + 1/2*mJoint(i)*dcom.joint.y(i)^2;
60 end
61
62 %% potential energy
63 V = 0;
64
65 % potential energy beams
66 for i = 1:4
67     V = V + mBeam(i)*g*com.beam.y(i);
68 end
69
70 % potential energy joints
71 for i = 1:3
72     V = V + mJoint(i)*g*com.joint.y(i);
73 end
74
75 % potential energy spring
76 l0 = 0.0;
77 V = V + 1/2*kSpring*(2*lBeam1*sin(q3/2)-l0)^2;
78
79 %% damping
80 B = diag([2.0,0.05,0.01]);
81 D = 1/2*dq'*B*dq + ...
82     0.1*(lBeam1*cos(q3/2)*dq3)^2 + ...
```

```
83      1*dq1^2*pres; % + ...
84  %       1/2*cur*R*cur;
85
86  % W = Wdamping + Wfriction;
87  % Q = jacobian(Wdamping, dq)';
88  %% external forces
89  Q = sym(zeros(length(q),1));
90
91  R = 1000;
92  L = 100;
93  kMotor = 10;
94  rWheels = 0.05;
95  Q(1) = Q(1) + (kMotor/rWheels)*cur;
96
97  % Q(3) = Q(3) + Wdamping;
98
99  %% motor equations
100 dT_ddq = jacobian(T,dq).';
101 ddT_ddqdt = jacobian(dT_ddq,t) + ...
102             jacobian(dT_ddq,q)*dq + ...
103             jacobian(dT_ddq,dq)*ddq;
104
105 dT_dq = jacobian(T,q).';
106
107 dV_dq = jacobian(V,q).';
108
109 dD_ddq = jacobian(D,dq).';
110
111 EoM = simplify(ddT_ddqdt - dT_dq + dV_dq - Q + dD_ddq);
112
113 [A,b] = equationsToMatrix(EoM,ddq);
114 A = simplify(A);
115 b = simplify(b);
116
117 %% values for system properties
118 mBeam1 = 0.5; mBeam2 = 0.5; mBeam3 = 0.5; mBeam4 = 0.5;
119 mCart = 4;
120 mJoint1 = 0.1; mJoint2 = 0.1; mJoint3 = 0.1;
121 lBeam1 = 0.2; lBeam2 = 0.2; lBeam3 = 0.2; lBeam4 = 0.2;
122 rJoint1 = 0.05; rJoint2 = 0.05; rJoint3 = 0.05;
123 hBeam = 0.02; bBeam = 0.02;
124 kSpring = 50; bSpring = 100;
```

```
bJoint1 = 0.6; bJoint2 = 0.6; bJoint3 = 0.6; bJoint4 = 0.6;
g = 9.81;

A = subs(A);
b = subs(b);

%% create function to evaluate eom
matlabFunction([dq; simplify(A\b); -R*cur/L + vol/L], ...
    'Vars',{t,[q; dq; cur],vol,pres}, ...
    'File', 'evaluateEoM');

matlabFunction([-R*cur/L + vol/L; 0], ...
    'Vars',{t,[cur;dcur],vol}, ...
    'File', 'evaluateEoDC');

%% initial conditions
q10 = 0;                % position cart
q20 = deg2rad(0);
q30 = deg2rad(0);
%      q1   q2   q3   dq1 dq2 dq3 i
y0 = [q10 q20 q30 0   0   0   0]';
```

## B.5 postrun

```
y = out.y.Data;
t = out.y.Time;



figure
subplot(3,2,1)
% plot([0 1 1.0001 3 3.0001 5 5.0001 7],[100 100 0 0 0 0 -100 -100],'LineWidth',2)
plot(out.sine.Time,out.sine.Data,'LineWidth',2)
hold on
plot(out.u.Time,out.u.Data,'LineWidth',2)
grid on
xlabel('time [s]')
ylabel('voltage [V]')
title('Supplied voltage')
legend('Desired','Actual')
set(gca,'FontSize',10)

subplot(3,2,2)
```

```
% plot([0 1 1.0001 3 3.0001 7],[0 0 10 10 0 0],'LineWidth',2)
plot(out.u1.Time,out.u1.Data,'LineWidth',2)
grid on
xlabel('time [s]')
ylabel('brake pressure [-]')
title('Brake pressure')
set(gca,'FontSize',10)


subplot(3,2,[3 5])
plot(t,y(:,1),'LineWidth',2)
hold on
plot(t,y(:,2),'LineWidth',2)
plot(t,y(:,3),'LineWidth',2)
grid on
legend('q1','q2','q3')
xlabel('time [s]')
ylabel('gen coordinate [m]/[rad]')
title('generalized coordinates')
set(gca,'FontSize',10)

subplot(3,2,[4 6])
plot(t,y(:,4),'LineWidth',2)
hold on
plot(t,y(:,5),'LineWidth',2)
plot(t,y(:,6),'LineWidth',2)
grid on
legend('dq1','dq2','dq3')
xlabel('time [s]')
ylabel('gen coordinate velocity [m/s]/[rad/s]')
title('generalized velocities')
set(gca,'FontSize',10)

%% animation
% figure(3)
% for i = 1:length(y)
%     if mod(i,round(length(y)/100)) == 0
%         q1 = y(i,1);
%         q2 = y(i,2);
%         q3 = y(i,3);
%
%         cartX = q1;
```

```
%           cartY = 0;
%           alpha = pi/2 - (2*pi - 2*q3)/4;
%           joint1X = cartX + sin(q2 - q3/2)*lBeam1;
%           joint1Y = cartY - cos(q2 - q3/2)*lBeam1;
%           joint2X = joint1X + sin(q2 + alpha)*lBeam2;
%           joint2Y = joint1Y - cos(q2 + alpha)*lBeam2;
%
%           joint3X = cartX + sin(q2 + q3/2)*lBeam4;
%           joint3Y = cartY - cos(q2 + q3/2)*lBeam4;
%
%           figure(3)
%           clf
%           plot([cartX;joint1X],[cartY;joint1Y],'b')
%           hold on
%           plot([joint1X;joint2X],[joint1Y;joint2Y],'b')
%           plot([cartX;joint3X],[cartY;joint3Y],'b')
%           plot([joint3X;joint2X],[joint3Y;joint2Y],'b')
%           xlim([-5 5])
%           ylim([-0.5 0.5])
%           drawnow
%       end
% end
```

### B.6 dc motor

```
syms cur t vol dcur real

R = 1000;
Larray = [25, 50, 75, 100, 250, 500];

% simulate DC motor
figure
subplot(3,2,1)
vInput = [100 0 0 -100 -50];
tInput = [2 2 2 2 2];
plot([0 2 2.00001 4 4.00001 6 6.00001 8 8.00001 10], ...
    [100 100 0 0 0 0 -100 -100 -50 -50],'LineWidth',2)
grid on
xlabel('time [s]')
ylabel('voltage [V]')
title('Supplied voltage')
set(gca,'FontSize',10)

```

```
for ii = 1:length(Larray)
    L = Larray(ii);
    matlabFunction([-R*cur/L + vol/L; 0], ...
    'Vars',{t,[cur;dcur],vol}, ...
    'File', 'evaluateEoDC');

    y0 = [0 0]';
    tspan = [0 5];

    tDC = []; yDC = [];
    for i = 1:length(vInput)
        t1 = 0;
        tspan = [t1(end) tInput(i)+t1(end)];
        [t1,y1] = ode45(@(t,y)evaluateEoDC(t,y,vInput(i)),tspan,y0);
        y0 = y1(end,:);
        % add time and output to total simulation results
        try
            tDC = [tDC; t1+tDC(end)];
        catch
            tDC = t1;
        end
        yDC = [yDC; y1];
    end

    subplot(3,2,[3 5])
    plot(tDC,yDC(:,1),'LineWidth',2)
    hold on
    grid on
    xlabel('time [s]')
    ylabel('current [A]')
end
legend('25 H', '50 H', '75 H', '100 H', '250 H', '500 H')
title('Current as a function of inductance')
set(gca,'FontSize',10)




Rarray = [0, 50, 75, 100, 250, 500];
L = 100;
```

```
% simulate DC motor
for ii = 1:length(Rarray)
    R = Rarray(ii);
    matlabFunction([-R*cur/L + vol/L; 0], ...
    'Vars',{t,[cur;dcur],vol}, ...
    'File', 'evaluateEoDC');

    y0 = [0 0]';
    tspan = [0 5];

    tDC = []; yDC = [];
    for i = 1:length(vInput)
        t1 = 0;
        tspan = [t1(end) tInput(i)+t1(end)];
        [t1,y1] = ode45(@(t,y)evaluateEoDC(t,y,vInput(i)),tspan,y0);
        y0 = y1(end,:);
        % add time and output to total simulation results
        try
            tDC = [tDC; t1+tDC(end)];
        catch
            tDC = t1;
        end
        yDC = [yDC; y1];
    end

    subplot(3,2,[4 6])
    plot(tDC,yDC(:,1),'LineWidth',2)
    hold on
    grid on
    xlabel('time [s]')
    ylabel('current [A]')
end
legend('0 \Omega', '50 \Omega', '75 \Omega', '100 \Omega', '250 \Omega', '500 \Omega'
    )
set(gca,'FontSize',10)
title('Current as a function of resistance')
```

## B.7 DFA test

```
clear; close; clc;

in = [1 0 0 0 0 0 0 0;
      1 1 0 0 0 0 0 0;
```

```
        0 0 1 0 0 1 0 0;
        0 0 0 0 1 0 1 0;
        0 1 1 1 1 1 1 0;
        0 0 0 0 0 0 0 1];

out = [];
for i = 1:size(in,1)
    inTS = [linspace(0,7,8)' in(i,:)'];
    outTS = sim('DFA',7);
    out = [out;
           outTS.two_1.Data'];
end

%%
figure
subplot(5,3,1)
plot(linspace(0,7,8),in(1,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,4)
plot(outTS.two_1.Time,out(1,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)

subplot(5,3,2)
plot(linspace(0,7,8),in(2,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,5)
plot(outTS.two_1.Time,out(2,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)

subplot(5,3,3)
plot(linspace(0,7,8),in(3,:),'bo')
grid on
```

```
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,6)
plot(outTS.two_1.Time,out(3,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)



subplot(5,3,10)
plot(linspace(0,7,8),in(4,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,13)
plot(outTS.two_1.Time,out(4,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)

subplot(5,3,11)
plot(linspace(0,7,8),in(5,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,14)
plot(outTS.two_1.Time,out(5,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)

subplot(5,3,12)
plot(linspace(0,7,8),in(6,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,15)
```

```
plot(outTS.two_1.Time,out(6,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
set(gca,'FontSize',10)

sgtitle('DFA simulations for velocity exceeding')
```

### B.8 Mealy test

```
clear; close; clc;

in = [1 1 0 1 1 0 1 1;
      0 1 1 0 0 0 0 0;
      0 0 1 0 0 1 0 0;
      0 0 0 0 1 0 1 0;
      0 1 1 1 1 1 1 0;
      0 1 0 0 1 0 0 1];

out = [];
for i = 1:size(in,1)
    inTS = [linspace(0,7,8)' in(i,:)'];
    outTS = sim('Mealy',7);
    out = [out;
           outTS.two_2.Data'];
end

%%
figure
subplot(5,3,1)
plot(linspace(0,7,8),in(1,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,4)
plot(outTS.two_2.Time,out(1,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)

subplot(5,3,2)
```

```
plot(linspace(0,7,8),in(2,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,5)
plot(outTS.two_2.Time,out(2,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)

subplot(5,3,3)
plot(linspace(0,7,8),in(3,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,6)
plot(outTS.two_2.Time,out(3,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)



subplot(5,3,10)
plot(linspace(0,7,8),in(4,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,13)
plot(outTS.two_2.Time,out(4,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)

subplot(5,3,11)
```

```
plot(linspace(0,7,8),in(5,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,14)
plot(outTS.two_2.Time,out(5,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)

subplot(5,3,12)
plot(linspace(0,7,8),in(6,:),'bo')
grid on
ylabel('input [-]')
set(gca,'FontSize',10)
subplot(5,3,15)
plot(outTS.two_2.Time,out(6,:),'ro')
grid on
xlabel('timestep [-]')
ylabel('output [-]')
ylim([0 2])
set(gca,'FontSize',10)

sgtitle('Mealy simulations for sounding alarm')
```