Hochschule Kaiserslautern
University of Applied Sciences

Co-funded by the Erasmus+ Programme of the European Union

E-Portfolios in Engineering Education

The 21st Century Engineer

FH Technikum Wien

M.A. Marina Zingraf

XP2P - Peer-to-Peer-Learning in Mechatronics

E-Portfolio Definition

- Digital collection
- Assessment of learning progress
- Multimedia evidence of students' efforts & competences
- Creator of e-portfolio is owner
- Intrinsic motivation of the student is emphasized

Reflections (Theory)

- Enable formative assessment
- Distinguish personal character of e-portfolios
- Provide links between artefacts
- Promote exchange with viewers
- Student-centered format: Critical examination of what has been learned & own learning progress, placement of learned content in overall context

Hochschule Kaiserslautern University of Applied Sciences

Co-funded by the Erasmus+ Programme of the European Union

XP29

Competencies (Example)

Skill	Essential Components	Main Expectations
Self-Assessment	Critical examination of own learning	• Critical examination of own learning and own approaches to solutions • Consideration of alternative solutions

Hochschule Kaiserslautern
University of Applied Sciences

Co-funded by the Erasmus+ Programme of the European Union

XP29

Reflections (Practice)

+Add comment Details

Milestone 1 - Reflection

Posted by Maximilian Pfleger on 05 November 2020, 16:35
Last updated 18 November 2020, 15:00
Tags: Control Design, Control System Tuner, Milesone 1, Multibody Simulation, Simscape

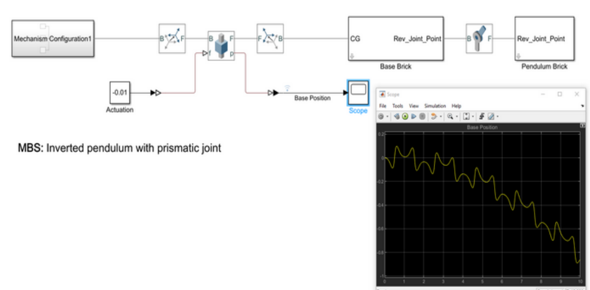
The content of this reflection is about the multibody simulation and control design according to the collateral. It explains the technical details of the model and simulation roughly and gives an overview about what has been worked out so far. To give an idea about the workflow, the aspects that were known before as well as those that had to be learnt are pointed out too.

There was no division of work between group members, as the tasks set are the acquisition of basic skills so far. Each group member worked on their own. Where necessary, members supported each other. The results were then compared and discussed together in team.

Multibody Simulation

The final *Simscape multibody* system (file: *my_inverted_pendulum_prismatic_joint.slx*) according to the instructions is shown in the figure below. The prismatic joint only allows movement in direction of its z-axis. It is configured through *"Rigid transform"*-blocks to allow movement of the base brick in the x-direction of the world coordinate system. Actuation force can be applied, to deflect the joint and move the base brick. The two brick subsystems are configured, that they are connected by their points of interest (centre of gravity, revolution joint). The masks of the two systems can be used to set the edge length, mass of the two bricks as well as the start angle of the pendulum. The plots shown below were generated with the following parameters:

- *m_base = 0,2 kg*
- *l_base = 0,05 m*
- *m_pend = 0,4 kg*
- *l_pend = 0,3 m*
- *phi_init = 5°*

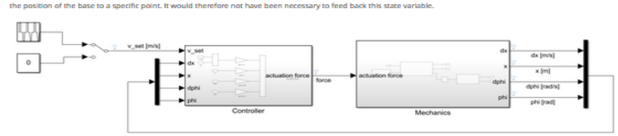


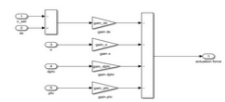
MBS: Inverted pendulum with prismatic joint

Even I did not have any experience in working with *Simscape multibody* before, working with it was straight forward. The robotics lessons in bachelor's programme were quite helpful, as the model setup works like the theory discussed in the course. The *MATLAB* documentation "Get started with Simscape Multibody" [1] also helped by showing how to build a proper model.

Control Design

The system shown in the figure below (file: *controller_tuner*) is a slightly rearranged version of mechanic system from the multibody simulation exercise, extended by a state space controller. The controller subsystem inputs are the four state variables from the mechanic system (*x* - position of the base brick, *dx* - velocity of the base brick, *phi* - angle of the pendulum, *dphi* – angular velocity of the pendulum) and a set point for the velocity (*v_set*) of the base brick. Output of the controller is the actuation force to move the base of the inverted pendulum. Every state variable is multiplied by a gain- factor. In case of the position *x*, the gain-factor is always zero because it is not desirable to adjust the position of the base to a specific point. It would therefore not have been necessary to feed back this state variable.

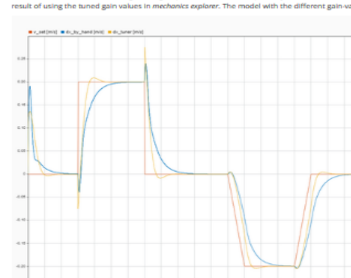




The plot below shows how the system reacts to a specific *v_set*-signal, when two different sets of gain factors are used. The parameters used are the same as mentioned in the section before. The target velocity at first is zero. Only the initial angle must be compensated. There is a step at *3 s* to *0.2 m/s* and back to *0 m/s* at *7 s*. At *12 s* the pendulum should move in reverse, but this time with a linear increasing velocity instead of a hard step to *0,2 m/s*. The set of gain-factors for the graph *dx_by_hand* is adjusted by hand:

- *gain_dx = -2*
- *gain_phi = 18*
- *gain_dphi = 1,5*

The figure shows that the controller configuration works. However, the system reacts rather slowly to a change of the target speed. The second set of gain-values is obtained by using the *Control System Tuner*. Second order characteristics, a time constant of *0.2 s* and an overshoot of *5%* are set as tuning goal. This controller reacts much faster to a change of the target speed. However, the resulting actuation force to compensate the initial angle is very high. The overshoot is due to the chosen second order characteristics of the controller. When increasing the target speed in form of a ramp, the system reacts much smoother in contrast to a step in the target signal. The video below shows the result of using the tuned gain values in *mechanics explorer*. The model with the different gain-values as well as the related *Control System Tuner*-session are attached.



As far as control design is concerned, I had no experience with it apart from the lecture in bachelor's programme. Thus, I knew some basics about control design and the idea behind it. Working with the Apps *Model Linearizer*, *Control System Designer* and *Control System Tuner* was not familiar. Building the controller, adjusting the gain factors by hand, and using the *Model Linearizer* did not cause any problems because of the detailed description in the collateral. However, using *Control System Designer* and *Control System Tuner* caused difficulties and it took some time until it was properly set up. The *MATLAB* documentation "Tune a Control System Using Control System Tuner" [2] was useful to complete the tasks.

Attached files (6)

Hochschule Kaiserslautern University of Applied Sciences

Co-funded by the Erasmus+ Programme of the European Union

Reflections (Practice)

SECOND REFLECTION

Tags: mechatronics, reflections, secondreflection, statemachines

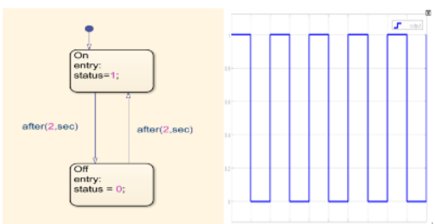
Second Reflection

Posted by Sri Vishnu Katreddi on 03 December 2020, 16:54
Last updated Monday, 07 December 2020, 14:43
Tags: statemachines, mechatronics, reflections, second reflection

LEARNING STATE MACHINES

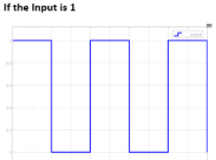
Stateflow charts are basically used as a functional tool for scheduling the tasks, decision logics and others, For understanding the concept of Stateflow in Simulink, I've started by watching a simple video from youtube. Later after doing the Matlab onramp course on Stateflow, I could incur basic information about different options that can be used.

Starting with the basics, I have built a simple chart that could toggle between two steps "on and off" with a time gap of 2 seconds, with the condition of showing output as "1" for on and "0" for off.



Besides, as an exercise, I was able to learn about Model Explorer and its use for altering different options for block parameters. Using it I've created an input port and also an error case so that if at all the input is 0, it should enter the error case and gives the output as -1, the outputs are as follows for different input.

If the Input is 1

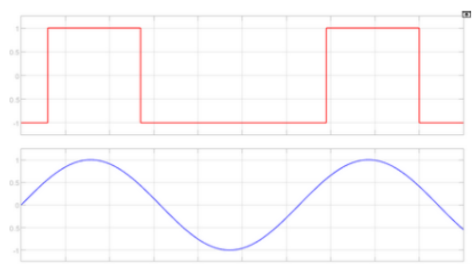


If the Input is 0

Truth Tables

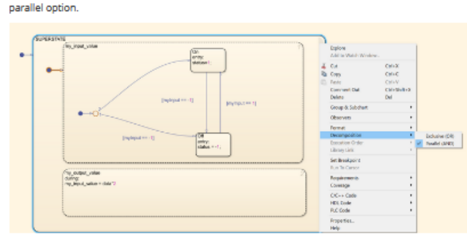
As mentioned truth tables are a very convenient way in setting up decision logics instead of using different if/else cases, using the truth tables, so I've built a new Stateflow chart that satisfies the following conditions

- If the input into the truth table is smaller/equal than 0.5 then the output achieved should be -1.
- If the input into the truth table is greater than 0.5 then the output achieved should be 1.
- Input to the truth table should be a sinus signal with amplitude 1.



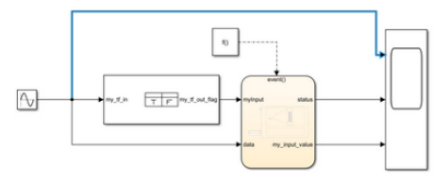
Creating a Superstate and adding a Simulink function

As part of next task, I have created a superstate with two different states, one state having the same conditions as above and whereas the second state shows the doubled sine value, it is also mentioned to both the states in a single superstate and have the same execution period, so we have the option of "decomposition", which we find on right-clicking on the superstate and select the parallel option.



Time based trigger for chart execution

For this exercise, I've initially added an input even from the Model Explorer, now for this port, in the Simulink window I've added a function call generator block with the time as 0.1 so that it gives time based compilation. The final State Machine Flow chart is observed to be as below.



Positively, In the figure Resulted in Scope on left, it can be observed that the first plot is of the sine wave with an amplitude of 1, moving on to the second plot it shows the graph of the truth table in which the graph is toggling between -1 to 1, from the conditions we have given for the truth tables if the value is greater than 0.5, it gives a continuous straight line at 1 and -1 if it is otherwise, In the third plot it can be seen that the sine wave is doubled by the Simulink function, so it shows the toggling between -2 to 2

Working with Code Exportation

As the prerequisites for working with the code exportation into the hardware, we were asked to install all the supported packages that are required, So as a precaution I've once again checked if all the support packages are installed in the MATLAB, Also the MecRoKa library which was specially created as per the hardware requirements is installed.

Since I'm in India the Robot was not delivered here due to some issues with the courier service centre, Nevertheless, I've gone through the information provided in the links, which show about how to install the drivers for the Arduino MKR1000 and MKRWifi1010 with the below links.

- MKR 1000: https://www.arduino.cc/en/Guide/MKR1000
- MKR 1010: https://www.arduino.cc/en/Guide/MKRWiFi1010

Later I've downloaded the Arduino IDE from the Microsoft store since I use windows 10, However, I couldn't setup and also check with basics like blinking an led etc., With the help of my teammates, I studied the task and the way the robot was working.

Using The MecRoKa

Since a part of the exercise doesn't require the hardware, I have checked with the IMU. The MecRoKa library has an inbuilt IMU block,the input to the simAcc port has to be provided and the output from the acc output port. The input is a sine wave along with constant block with the value 1 via a mux that is translated to the int32 data type as the simAcc port expects the data type input. The performance can be seen in a demux connected scope that is first connected to the IMU's access port. To slow down the simulation speed, a 'Set Path' block is used or the simulation runs faster than compared to real-time. The output for the above test can be seen as a sine wave.

Further Links & Contact information

Marina Zingraf (M.A.)
Project XP2P-Peer-to-Peer-Learning in Mechatronics
E | marina.zingraf@hs-kl.de
WWW | https://xp2p-project.eu/
Mahara | How to create an E-Portfolio?