Lab 3: Fast gradient algorithm-based predictive control

Bulat Khusainov and Eric Kerrigan

Learning outcomes

By the end of this lab the student should be able to implement a predictive controller based on the fast gradient algorithm on a Zedboard.

Files and folders provided

main_script.m	Main script of the lab
model_generator.m	Function for mass-spring-damper system model generation
qp_generator.m	Function for QP matrices generation
generate_header.m	Script for *.h files generation
SIL_simulation.m	Software-in-the-loop simulation script
hls.m	High-level synthesis script (C → VHDL)
synthesis.m	Synthesis script (VHDL → Bitstream)
HIL_simulation.m	Processor-in-the-loop simulation script
src	folder with C source files
protoip_project	Protoip project folder

Optimal control problem

We consider the following optimal control problem:

$$\underset{u_0 \dots u_{N-1}, x_0 \dots x_N}{\text{minimize}} \quad \sum_{k=0}^{N-1} \left(\frac{1}{2} x_k^T Q_d x_k + \frac{1}{2} u_k^T R_d u_k \right) + \frac{1}{2} x_N^T P_d x_N \tag{1a}$$

$$\text{subject to} \quad x_0 = \hat{x} \tag{1b}$$

$$x_{k+1} = A_d x_k + B_d u_k, \quad \text{for } k = 0, 1, \dots, N-1 \tag{1c}$$

$$u_{min} \leq u_k \leq u_{max}, \quad \text{for } k = 0, 1, \dots, N-1 \tag{1d}$$

where $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, $A_d \in \mathbb{R}^{n \times n}$, $B_d \in \mathbb{R}^{n \times m}$. Q_d, R_d and P_d are the weight matrices of appropriate dimensions that have to be chosen to ensure convexity of the objective function. $\hat{x}$ is the initial state and N is the prediction horizon.

Quadratic Programming (QP) formulation

By eliminating the states, the optimal control problem (1) can be transformed into a quadratic programming problem of the form

$$\text{minimize} \quad \frac{1}{2} \theta^T H \theta + \theta^T h \tag{2a}$$

$$\text{subject to} \quad \theta_{min} \leq \theta \leq \theta_{max} \tag{2b}$$

Solving the QP

Since (2b) has so-called box constraints, calculating the projection on the feasible set is computationally cheap. This facilitates using Nesterov's projected gradient algorithm, also known as the Fast Gradient Method (FGM), as detailed in Algorithm 0.1. If we use a constant step scheme, then $\beta = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$, where L is the Lipschitz constant of the Hessian H and μ is the convexity parameter, which is equal to the minimum eigenvalue of the Hessian.

Algorithm 0.1: Projected fast gradient algorithm for constrained optimization with a constant step size.

1: Initial guess: θ_0
2: $v_0 = \theta_0$
3: **for** $i = 0$ to N_{FGM} **do**
4: $\quad \theta_{i+1} = (I - (1/L)H)\nu_i - (1/L)h$ {anti-gradient step}
5: $\quad z_{i+1} = P(\theta_{i+1})$ {projection on the feasible set}
6: $\quad v_{i+1} = (1 + \beta)z_{i+1} - \beta z_i$ {extra-momentum step}
7: **end for**

Plant model

The system that we will control represents a chain of ten masses that are connected via springs and dampers (Figure 1). The first mass is also connected to a fixed wall. Each mass can be actuated with an input force that has input and output limits. It is assumed there is no gravitational force.

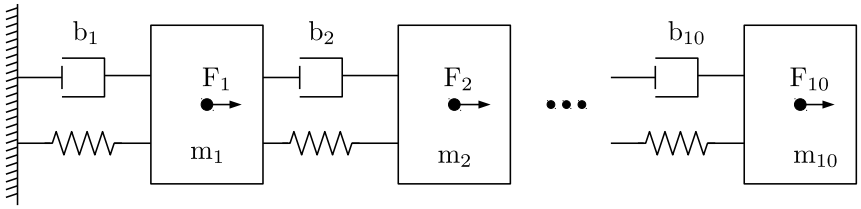

Figure 1: Mass-spring-damper system.

Design flow - basic implementation

Consider the design flow in Figure 2.
Every step of the flow corresponds to a section in the main_script.m file (sections are separated using %%). Go through every step of the flow by uncommenting a corresponding section and running main_script.m:

- *Define design parameters.* A structure with design parameters is created.
- *Generate state-space model.* State-space models (continuous-time and discrete) of the mass-spring-damper system are generated. The 'design' structure is also updated. Refer to the file model_generator.m to see how the model is created.

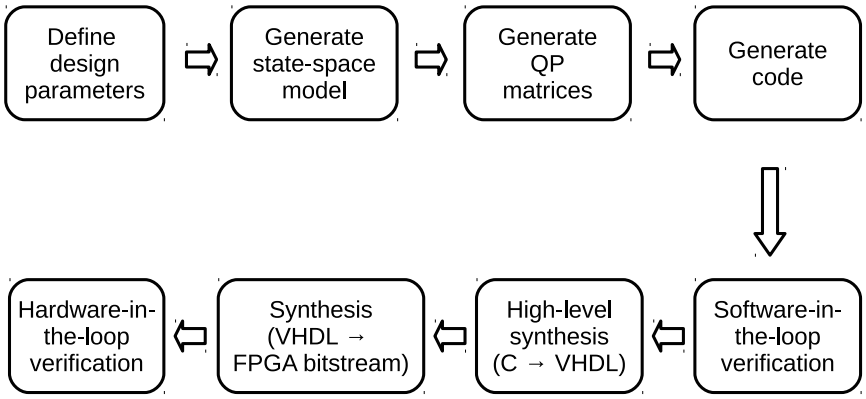

Figure 2: Design flow.

- *Generate QP matrices.* Matrices for the QP problem (e.g. Hessian, gradient) are generated. All required matrices can be found in the 'qp_problem' structure. See the file 'qp_generator.m' to understand the internal operation of the function.
- *Generate code.* Two header files are created at this stage: user_fgm_mpc.h (design parameters) and user_qp_matrices.h (precalculated matrices), which can be found in the src/ folder. The fast gradient algorithm implementation is in user_fgm_mpc.cpp. Note that the projection operation (line 5 of Algorithm 0.1) is not implemented.
- *Software-in-the-loop verification.* Compile C code from the previous step with a mex interface and test it in the Matlab environment.
- *High-level synthesis.* Convert C code to VHDL using Protoip. On the underlying level Protoip uses Vivado HLS. After this step you should be able to see FPGA resource usage *estimations* and algorithm execution time in protoip_project/doc/my_project0/ip_design.txt.
- *Synthesis.* Convert VHDL to FPGA bitstream using Protoip. On the underlying level Protoip uses Vivado. After this step you should be able to see FPGA resource usage (exact values) in protoip_project/doc/my_project0/ip_prototype.txt.
- *Processor-in-the-loop verification.* Load the bitstream to FPGA and test it in the loop with plant model. Protoip uses Xilinx SDK to perform this step. Processor-in-the-loop test results should be available in /protoip_project/ip_prototype/test/results/my_project0.

Possible extensions of the basic implementation

- Once you go through the steps above, try to add input constraint handling by modifying the user_fgm_mpc.cpp file and repeating the steps. Lower and upper bounds on the inputs are available in user_qp_matrices.h.
- Try commenting out pipelining directives in the code (user_fgm_mpc.cpp) in order to see the impact of pipelining on FPGA resource usage and algorithm execution time.
- Investigate the impact of horizon length on FPGA resource usage and algorithm execution time.

Further reading and references

- Please cite `https://doi.org/10.1109/ECC.2016.7810272` when using the code provided in this lab for your research.