UNIVERSITY OF SALERNO

DEPARTMENT OF INFORMATION AND ELECTRICAL ENGINEERING AND APPLIED MATHEMATICS

Master's Degree Course in Automation and Control Systems

DIGITAL CONTROL SYSTEM DESIGN

Direct coding MPC with constraints

Authors:

Nicola ROMANO 0622701549
Antonio ROTONDO 0622701489
Catello SORRENTINO 0622701490

ANNO ACCADEMICO 2021/2022

Contents

1 Introduction

The aim of this project is to implement a Model Predictive Control scheme (**MPC**) with constraints to control the speed of a motor. The control scheme has been implemented in direct coding on the **STM32F401RE** board using OSQP as quadratic solver to calculate the solutions. The challenging part of this project will be, as we'll see further, the computational time. Let's see in detail the main parts of the project.

2 Model Description

The motor used in the project is the **Pololu 25D Gearmotor**, a 12V DC brushed motor equipped with a Hall effect quadrature encoder for speed/position measurement. It has been interfaced to the board through the **STM X-Nucleo-IHM04A1** driver which implements the H-bridge useful to set the rotation direction of the motor.
The transfer function of the motor is the following:

$$G(s) = \frac{w(s)}{v(s)} = \frac{167773.98}{s^2 + 392.96s + 15992.15} \tag{1}$$

that sampled with $T_s = 0.05$ s gives us the following discrete transfer function:

$$G_d(z) = \frac{9.284z + 0.1604}{z^2 - 0.09973z + 2.931e - 09} \tag{2}$$

which will be used in the following project.

3 Model Predictive Control

The Model Predictive Control has been designed to give good performances without taking too much computational time. To achieve so, after a phase of tuning of the parameters, we chose to use the following ones:

- $n_y = 10$,
- $n_u = 2$.

n_y and n_u have been chosen low to ensure that the computational time is relatively small since they affect the size of the problem to solve. This, combined with the choice of $T_s = 0.05$ s, gives enough time to the STM32 board to solve the problem and it ensures at the same time good results and reactivity of the control.
Before introducing the results, let's take a closer look to the quadratic problems solver that we are going to use to implement the MPC control law, defined by the following QP optimisation problem:

$$\min_{\underset{\rightarrow}{\Delta u_k}} \quad \underset{\rightarrow}{\Delta u_k}^T S \underset{\rightarrow}{\Delta u_k} - \underset{\rightarrow}{\Delta u_k}^T a \tag{3}$$

$$\text{s.t.} \quad CC \underset{\rightarrow}{\Delta u} \leq dd + d_u u_{k-1} + dd_u \underset{\leftarrow}{\Delta u_{k-1}} + d_y \underset{\leftarrow}{y_k} \tag{4}$$

4 Open Source Quadratic Problem

OSQP solves convex quadratic problems (QPs) of the form:

$$\min \quad \frac{1}{2}x^T P x + q^T x \tag{5}$$

$$\text{s.t } l \leq Ax \leq u \tag{6}$$

where $x \in R^n$ is the optimization variable. The objective function is defined by a positive semi-definite matrix $\mathbf{P} \in S^n_+$ and vector $\mathbf{q} \in R^n$. The linear constraints are defined by matrix $\mathbf{A} \in R^{m*n}$ and vectors $\mathbf{l}$ and $\mathbf{u}$ so that $l_i \in R \cup \{+\infty\}$ and $u_i \in R \cup \{+\infty\}$ for all $i \in \{1, ..., m\}$.
It is then feasible for our purposes with MPC since the constrained optimization that we want to solve is in the form:

$$\underset{x}{min} J = \frac{1}{2}x^T S x + b^T x + c \tag{7}$$

$$\text{s.t } Mx \leq d \tag{8}$$

5 MATLAB implementation

Before getting on the STM32 board we decided to simulate the behavior of the MPC controller on MATLAB. Usually the quadratic problems solver used in MPC is **quadprog** so, to have a simulation with OSQP, we rewrote the mpc simulation function providing the new solver. We can then proceed to run some simulations which yield the following behaviors:

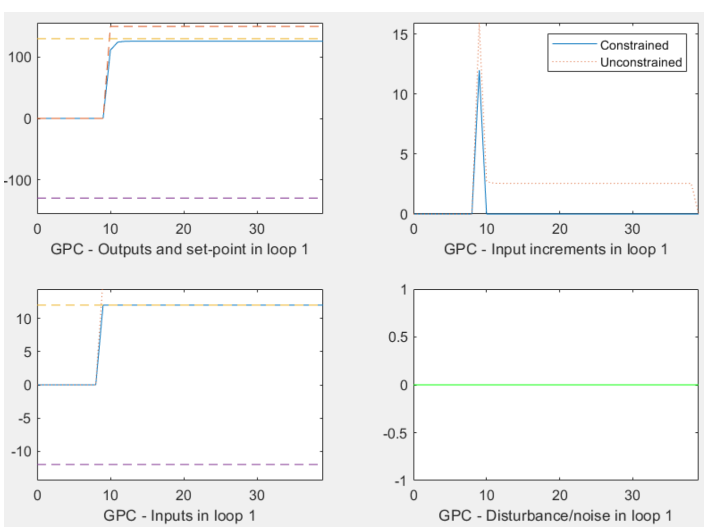

Figure 1: Simulation of MPC with OSQP, violation of constraints.

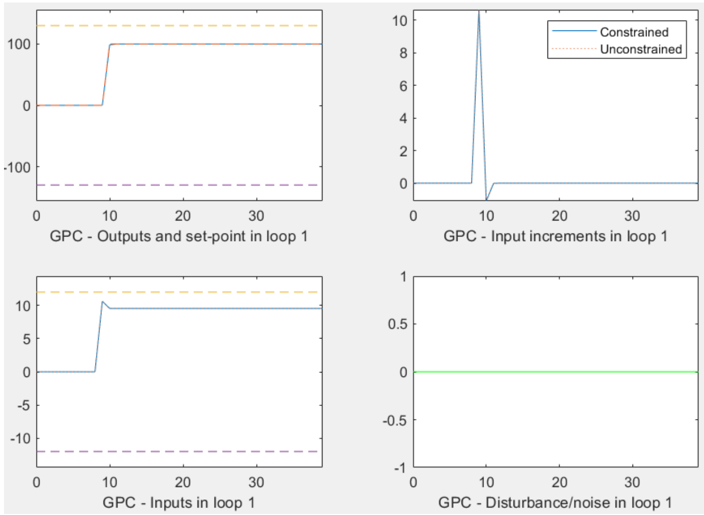

Figure 2: Simulation of MPC with OSQP, reach of the target behavior.

We can see how, when a constraint is violated, the MPC keeps the input and output inside the bounds and, when constraints aren't violated, the MPC reaches the target behavior.

Once we ensured that the implementation of the MPC controller with OSQP is feasible we can implement the controller on the STM32 board and test it on the motor itself.

6 Pinout & configuration

The following paragraph explains in detail how the board got configured.
Pins **PB4** (D5) and **PB5** (D4), physically connect to **IN1A** and **IN2A** respectively, are configured as output pins for PWM generation. For this purpose, the hardware timer **TIM3** is allocated and its channels **CH1** and **CH2** are used. Pin **PA10** (D2), physically connected to ENA, is set as digital output pin. Pins **PA8** (D7) and **PA9** (D8), physically connect to encoder A and encoder B respectively, are configured as input pins in **Encoder mode**. To this aim, the hardware timer/counter **TIM1** is allocated and its channels **CH1** and **CH2** are used respectively. Finally, by allocating pins **PA2** and **PA3** as $USART2\_TX$ and $USART2\_RX$ respectively, we allow the **STM32F401RE** to communicate with the USART protocol over USB with the PC.

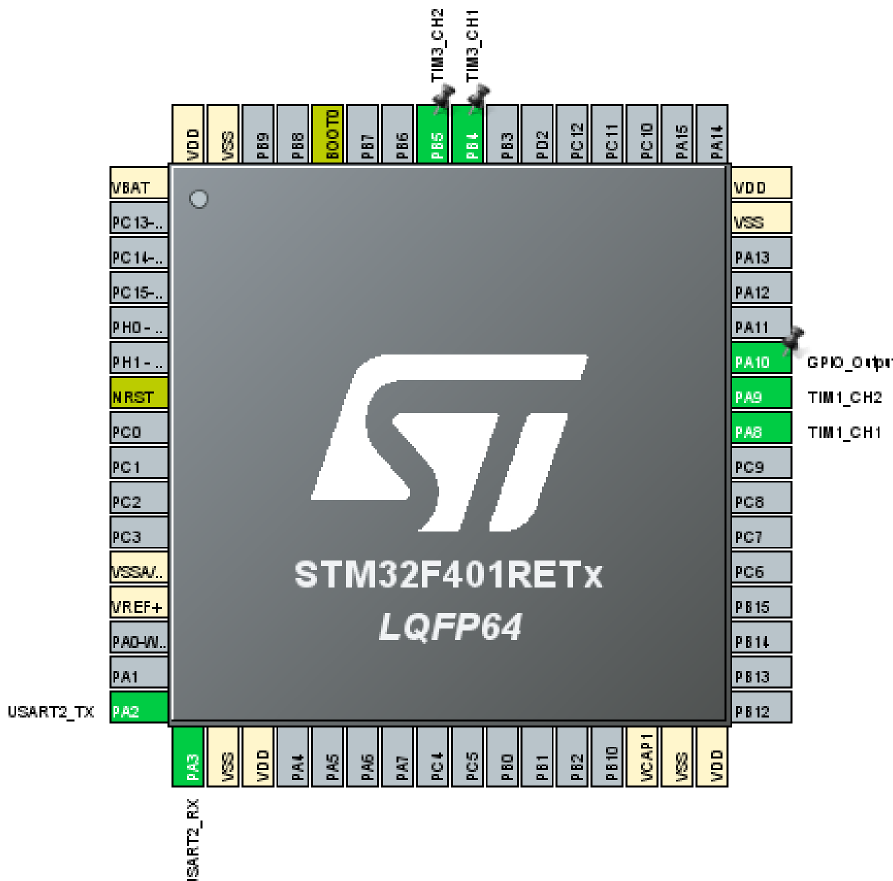

Figure 3: Pinout & configuration.

7 Direct coding implementation

The OSQP library provides a Python interface that gives the possibility to generate C code for the resolution of a quadratic problem. It is possible to set various parameters of the solver to grant better performances or to speed up the algorithm and obtain the solution in less time. Due to the poor performances of the board it has been mandatory to modify some settings in order to get a solution that is still valid in a much smaller time. After a tuning phase it has been assessed that the optimal performances are given by the following parameters:

- relaxation parameter **alpha = 1.0**, which is a parameter used to speed up the convergence of OSQP. The value has to be between 0 and 2. We chose **1.0** to get a trade off between execution time and precision of the solution;
- absolute tolerance **eps_abs = 0.05**, which is the tolerance of the error we want on our results. The chosen value speeds up the algorithm and gives reasonable results;
- maximum number of iterations of the OSQP algorithm**,** max_iter = 5;
- **warm_start = False** to avoid performing the warm start of the algorithm, since the algorithm's running time depends largely on the distance between the algorithm's initial iterate and the problem's set of optimizers, one can set a solution of the previous problem instance as the initial iterate in the next instance. However this initialization of the solver takes a lot of time and it's not that useful for small problems, so we decided to deactivate this function.

The generated code of the OSQP solver has then been exported on the board to use it in the controller.
To implement the control on the STM32 board we decided to write the control law in the main loop and use some ballast code to synchronize the program with the designed sampling time. The motor is controlled through Pulse Width Modulation, implemented using Timer3 in PWM mode, and the speed is calculated reading the ticks of the encoder mounted on the motor using Timer1 in encoder mode on both channels to have the possibility to read also the sign of the speed. For each iteration the program will calculate the speed, update the problem to the current state, solve the problem using MPC with OSQP and apply the optimal solution. Once done, the ballast code will wait the sample time to finish then proceed to execute the next iteration.
To observe the behavior of the controlled system we use USART, monitoring the following parameters: required speed, actual speed of the motor, u computed from OSQP, u applied on the system, execution time of OSQP, total execution time.

8 Results

To check the results, an array of references has been implemented. The values used will be $\{-130, 80, -60, 30, 0, -30, 60, -80, 200\}$. Notice the swap of sign between the values, that's to give a huge gap between the speeds and to change the rotation of the motor in order to test the controller in the worst situations and evaluate the transient behavior. Furthermore the last value exceeds the y_{max} constraint so we expect the motor to rotate at its max speed, which is around 134 RPM.
The first thing we can notice is that the time of resolution of the problem is $13 - 14ms$ so the T_s is satisfied. We can then check the transients:

```
Required speed: -130 RPM
Actual speed: 38.75 RPM
u from OSQP: -15.63 V
u applied: -12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: -130 RPM
Actual speed: -100.56 RPM
u from OSQP: -12.42 V
u applied: -12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: -130 RPM
Actual speed: -126.62 RPM
u from OSQP: -12.02 V
u applied: -12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: -130 RPM
Actual speed: -132.63 RPM
u from OSQP: -11.79 V
u applied: -11.79 V
OSQP exec time:  14 ms
Total exec time:  50 ms
```

Figure 4: Transient to reach $-130RPM$ from $0RPM$.

```
Required speed: 80 RPM
Actual speed: -129.29 RPM
u from OSQP: -3.17 V
u applied: -3.17 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 80 RPM
Actual speed: -93.55 RPM
u from OSQP: 3.61 V
u applied: 3.61 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 80 RPM
Actual speed: -13.70 RPM
u from OSQP: 7.00 V
u applied: 7.00 V
OSQP exec time:  13 ms
Total exec time:  50 ms
```

```
Required speed: 80 RPM
Actual speed: 46.77 RPM
u from OSQP: 8.07 V
u applied: 8.07 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 80 RPM
Actual speed: 75.50 RPM
u from OSQP: 8.12 V
u applied: 8.12 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 80 RPM
Actual speed: 85.86 RPM
u from OSQP: 7.84 V
u applied: 7.84 V
OSQP exec time:  13 ms
Total exec time:  50 ms
```

Figure 5: Transient to reach $80RPM$ from $-130RPM$.

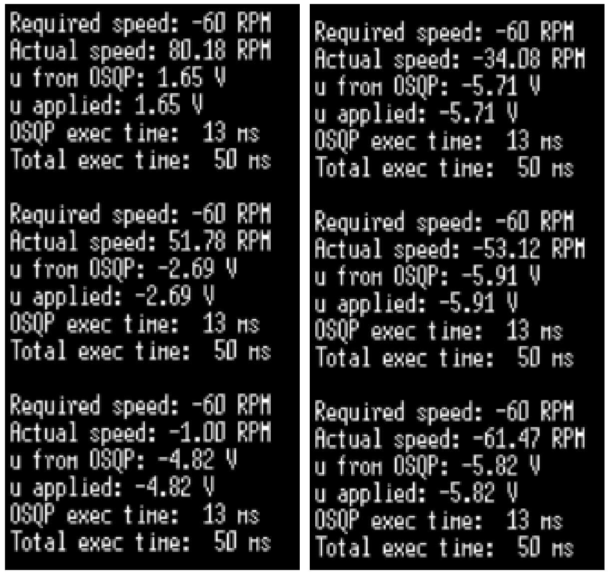

Figure 6: Transient to reach $-60RPM$ from $80RPM$.

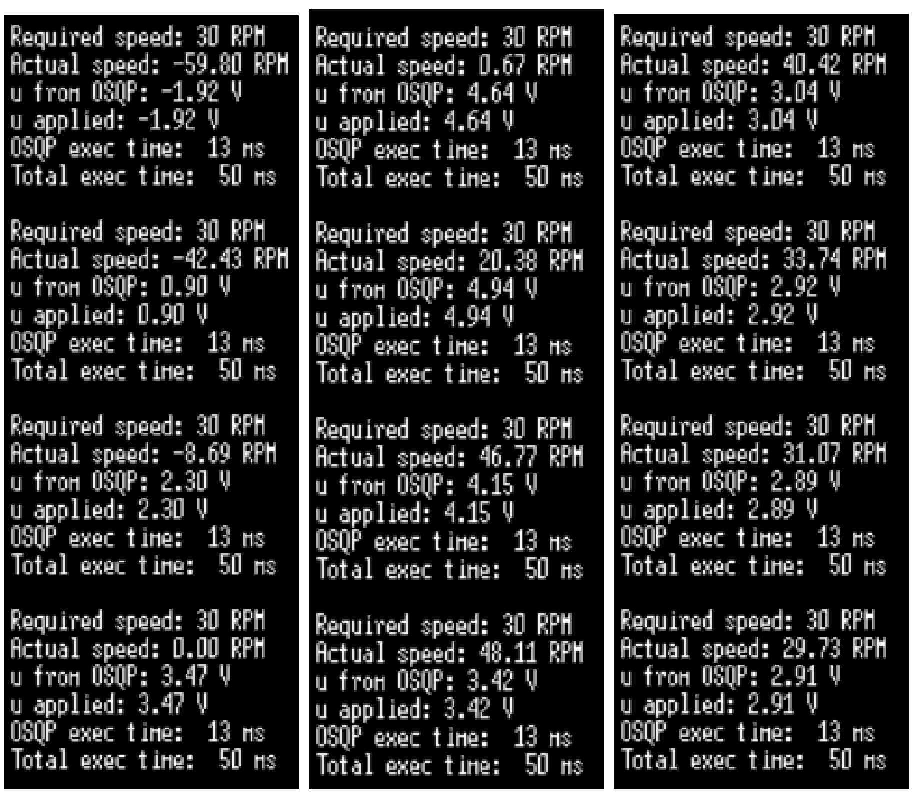

Figure 7: Transient to reach $30RPM$ from $-60RPM$.

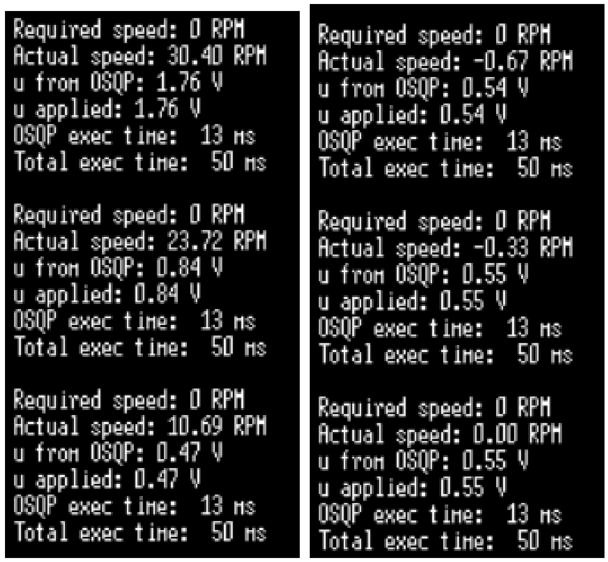

Figure 8: Transient to reach $0RPM$ from $30RPM$.

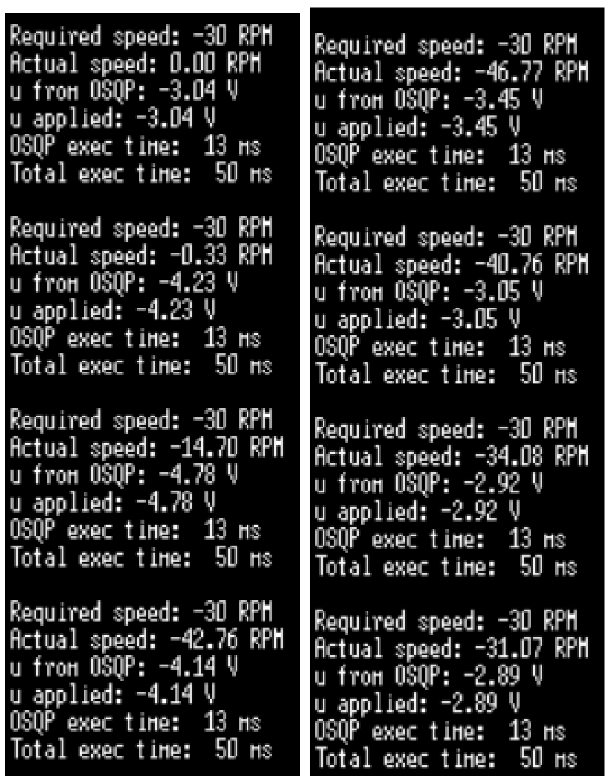

Figure 9: Transient to reach $-30RPM$ from $0RPM$.

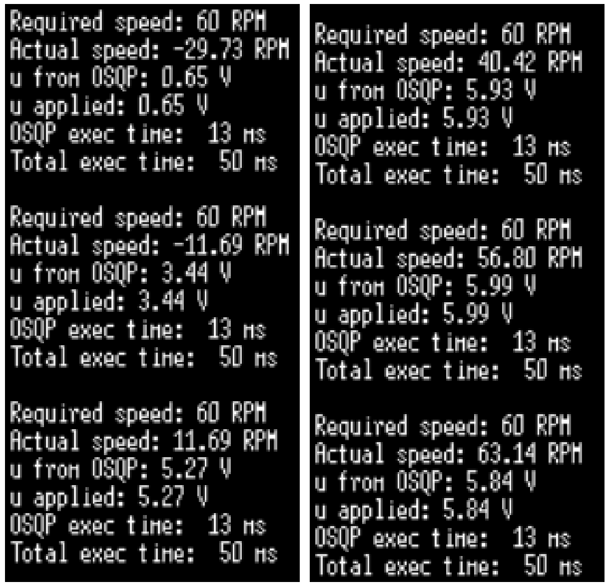

Figure 10: Transient to reach $60RPM$ from $-30RPM$.

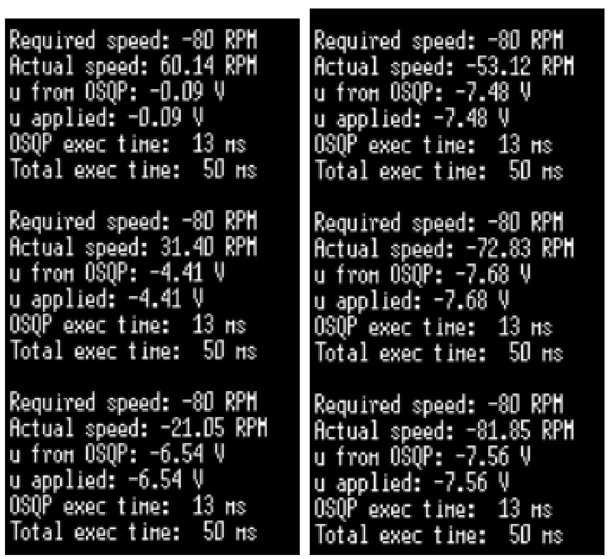

Figure 11: Transient to reach $-80RPM$ from $60RPM$.

```
Required speed: 200 RPM
Actual speed: -79.85 RPM
u from OSQP: 4.04 V
u applied: 4.04 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: -27.73 RPM
u from OSQP: 12.88 V
u applied: 12.00 V
OSQP exec time:  13 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: 65.15 RPM
u from OSQP: 15.34 V
u applied: 12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: 119.60 RPM
u from OSQP: 13.27 V
u applied: 12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: 131.30 RPM
u from OSQP: 12.71 V
u applied: 12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: 133.64 RPM
u from OSQP: 12.66 V
u applied: 12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms

Required speed: 200 RPM
Actual speed: 134.30 RPM
u from OSQP: 12.60 V
u applied: 12.00 V
OSQP exec time:  14 ms
Total exec time:  50 ms
```

Figure 12: Transient to reach $200RPM$ from $-80RPM$. Notice that the constraints are satisfied.