Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Using an embedded QP solver for automotive applications

Tomáš Rutrle

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.
May 2022

Acknowledgements

I want to express my gratitude towards my family for enabling and supporting me throughout the course of my studies, for which I will always be grateful.

Furthermore, another thank you belongs to my supervisor, friends, and colleagues for the guidance regarding this thesis.

Declaration

I, Tomá² Rutrle, hereby declare that this thesis is a product of my own work and that to the best of my knowledge all information sources have been listed in accordance with the methodical instructions for observing the ethical principles in the preparation of a university thesis.

Prague, May 2022 ....

Abstract

The main focus of this thesis is on the development of a framework for an external quadratic program solver utilizing the CAN and CAN FD communication buses. To begin with, the deployment of two quadratic solvers onto a controller is presented. Further, an embedded application is developed to handle bus communication, memory management, and solver calling. To utilize the external solver, Matlab & Simulink based CAN and CAN FD interfaces are implemented to allow for real-time simulation in the hardware-in-the-loop configuration. Both the embedded application and the interfaces are tested for functionality and performance with a focus on the time it takes the framework to solve quadratic problems of various sizes. Finally, two automotive-related model predictive control demos are presented, employing the external solver interface and showcasing its capabilities.

Keywords: Quadratic programming, QP, Embedded solver, CAN, CAN FD, Optimal control, MPC, External solver, Vehicle control

Supervisor: doc. Ing. Tomá² Hani², Ph.D.
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Abstrakt

Hlavním cílem této práce je vývoj rámce pro externí solver kvadratických problém· vyuºívající CAN a CAN FD sb¥rnice. V první °ad¥ je p°edstaven mikrokontroler, na který jsou nasazeny dva r·zné solvery. Dále je pro kontroler vyvinuta vestav¥ná aplikace, která obstarává komunikaci p°es sb¥rnici, správu pam¥ti a obsluhu solver·. Na druhé stran¥ jsou navrhnuta a implementována dv¥ prost°edí v programu Matlab & Simulink, jedno pro CAN a druhé pro CAN FD sb¥rnici. Tato prost°edí pak slouºí k p°ístup¥ní externího solveru a umoº¬ují simulaci v reálném £ase v konfiguraci "hardware-ve-smy£ce". Ob¥ prost°edí jsou d·kladn¥ otestována spolu s aplikací v kontroleru a je provedena °ada m¥°ení testující dobu, kterou celý rámec pot°ebuje k vy°e²ení kvadratického problému v závislosti na velikosti daného problému. Na záv¥r jsou p°edvedena dv¥ dema vyuºívající modelového prediktivního °ízení a vyvinutého externího solveru k optimalizaci formulovaných problém·.

Klí£ová slova: Kvadratické programování, QP, Vestav¥ný solver, CAN, CAN FD, Optimální °ízení, Modelové prediktivní °ízení, Externí °e²i£, •ízení vozidla

P°eklad názvu: Vyuºití vestav¥ného QP solveru pro automotive aplikace

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

Historically, vehicles were considered to be mechanical systems. To improve performance or efficiency, car manufacturers would add cylinders to the combustion engine or make the car's body more aerodynamic. Only in recent decades, with the rise of digitalization, have vehicles transformed into exhaustively complex digital systems, which rely on software no less than on properly tuned suspension. With the current trends of the automotive industry to push for emission reduction and safety improvements, the development cycle of new vehicle models is not only about deploying new hardware components but also about optimizing the software controlling these components. But with the increasing complexity of the system, the need for sophisticated control strategies, which ensure reliable operation of the whole plant, increases as well. For that reason is model based predictive control (MPC) popularized in the automotive industry [1].

It is not uncommon for a modern vehicle to be equipped with a number of systems that would benefit from model based predictive control. These systems can be anything from battery management to cruise control. However, equipping all of them with the hardware and software necessary to run the MPC controller could prove time and money inefficient. Since both linear and non-linear MPCs are eventually reformulated as quadratic optimization problems, centralizing the task of solving these problems may be an elegant solution to this issue. Furthermore, developing a dedicated external quadratic program (QP) solver presents several other advantages. Firstly, it would decrease the computational requirements on the system controllers, as the demanding task of QP optimization would be performed on external hardware. Moreover, deploying and maintaining the embedded QP solver would be simplified as well, since any updates and improvements to the software would directly affect only one controller in the vehicle.

1.2 Thesis goals

The goal of the thesis is to develop and implement a Simulink based framework, which will allow the user to solve quadratic problems on embedded hardware using automotive grade bus like CAN and CAN FD. The thesis will address following points:

- Familiarization with the topic of quadratic programming, automotive grade communication and embedded hardware
- Analysis of representative automotive grade communication solutions
- Adaptation of an existing QP solver and its deployment
- Realization of baseline and improved communication logic and verification of the workings of the embedded QP solver
- Utilization of the embedded QP solver in the MPC domain, analysis and presentation of acquired results

1.3 Thesis outline

This thesis will be divided into five main chapters, excluding the introduction and conclusion. Chapter number two will consist of research on thesis-related topics, such as quadratic programming, linear model based predictive control, and automotive-grade field buses. How does receding horizon control reformulate into a QP, and what communication bus to use to transmit the QP problem to an external controller? This will be followed by a chapter that deals with the presentation of the selected microcontroller and the deployment of two QP solvers onto the said controller. The next chapter will focus on the development of a CAN bus based interface, both on the side of the embedded application and the side of personal computer running Matlab & Simulink. First tests and measurements will be performed, showcasing the workings of the deployed solvers and the newly developed interface. Chapter five will then be similar in its content as the making of a CAN FD based interface will be showcased. This time more focus will be on the interface performance testing and subsequent presentation of acquired results. Last but not least, the final chapter will present MPC demos running in a hardware-in-the-loop (HIL) configuration to display the functionality of the developed external QP solver framework.

Chapter 2

Theoretical analysis

2.1 Quadratic programming

Mathematical optimization has always been a subject of great interest for mathematicians. Newton's iterative method for locating roots of real continuous functions, the Lagrange multiplier method used for finding optima of constrained functions, or the simplex method used for solving linear programs, all play an essential role in the current state of mathematical optimization. Quadratic programming as we know it has its root in the 1950' and was popularized f.e. by the Economist and Nobel prize laureate Harry Markowitz who formulated the portfolio optimization problem as a quadratic program [3]. As of today, QP is the most prominent method for solving non-linear optimization problems and is applied in many areas such as computer graphics, signal and image processing, investment optimization, and optimal control theory.

2.1.1 Quadratic program definition

A general linear quadratic program, i.e. an optimization problem with quadratic cost function and linear constraints, can be written as follows.

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} x^\top H x + f^\top x \\ \text{s.t.} \quad & Ex = e \\ & Ax \leq a \end{aligned} \tag{2.1}$$

Where $x \in \mathbb{R}^n$ is the optimization vector and $H \in \mathbb{R}^{n \times n}$ with $f \in \mathbb{R}^n$ denote the quadratic and linear parts of the cost function. The constraints are then defined in two categories, firstly the equality constraints, $E \in \mathbb{R}^{m_1 \times n}$ and $e \in \mathbb{R}^{m_1}$, secondly the inequality constraints, $A \in \mathbb{R}^{m_2 \times n}$ and $a \in \mathbb{R}^{m_2}$.

Furthermore, without a loss of generality it will be assumed that H is a symmetric matrix [5].

Proof. To prove this it can be shown, that replacing any matrix H with the symmetric matrix $\frac{1}{2}(H + H^\top)$ will not affect the value of the cost function.

Since $x^\intercal H x \in \mathbb{R}$:

$$
\begin{aligned}
x^\intercal H x &= x^\intercal \left(\frac{2H}{2} + \frac{H^\intercal - H^\intercal}{2} \right) x \\
&= x^\intercal \left(\frac{H + H^\intercal}{2} + \frac{H - H^\intercal}{2} \right) x \\
&= x^\intercal \left(\frac{H + H^\intercal}{2} \right) x + \frac{1}{2} x^\intercal H x - \frac{1}{2} x^\intercal H^\intercal x \\
&= x^\intercal \left(\frac{H + H^\intercal}{2} \right) x + \frac{1}{2} x^\intercal H x - \left(\frac{1}{2} x^\intercal H^\intercal x \right)^\intercal \\
&= x^\intercal \left(\frac{H + H^\intercal}{2} \right) x
\end{aligned}
\tag{2.2}
$$

□

2.1.2 Convex quadratic programming

General set $S \in \mathbb{R}^n$ is said to be *convex* if for any two points $x; y \in S$ the straight line connecting these points lies entirely within S. Formally, $\lambda x + (1 - \lambda) y \in S \ \forall \lambda \in [0; 1]$. Similarly, the function $f : A \rightarrow B$ is convex if the domain of the function A is a convex set and for any two points $x; y \in A$ the following is true:

$$
f(\lambda x + (1 - \lambda) y) \leq \lambda f(x) + (1 - \lambda) f(y) \quad \forall \lambda \in [0; 1] \tag{2.3}
$$

For numerical optimization the concept of convexity is extremely useful as a local minimum of a convex function on a convex subspace is also its global minimum. For reference see the topic of convexity (page 7) in [2].

Proof. Let us define $f : A \rightarrow B$ as a convex function with a convex domain A. Furthermore, $x^* \in A$ is the local minimum of f. By definition of local minimum, an open neighborhood N exists s such that $f(x^*) \leq f(n) \ \forall n \in N$. For any arbitrary point $y \in A$ we can take the linear combination $z = (1 - \lambda) x^* + \lambda y$ and say that as λ approaches 0, z approaches x^* which implies $z \in N$. Therefore, for $\lambda \rightarrow 0$:

$$
\begin{aligned}
f(x^*) &\leq f(z) \\
f(x^*) &\leq f((1 - \lambda) x^* + \lambda y) \\
f(x^*) &\leq (1 - \lambda) f(x^*) + \lambda f(y) \\
\lambda f(x^*) &\leq \lambda f(y) \\
f(x^*) &\leq f(y)
\end{aligned}
\tag{2.4}
$$

This shows that for a convex problem the local minimum is also the global minimum as $f(x^*) \leq f(y) \ \forall y \in A$. □

Whether the general QP problem as defined in 2.1 is a convex problem or not depends on both the cost function and the linear constraints. The cost function itself generally consists of a sum of two terms - quadratic and linear. As for the linear term, $f^\intercal x$ is convex for any $f \in \mathbb{R}^n$, whereas the quadratic form $x^T Hx$ is convex only for a positive semidefinite matrix H. For such (symmetric) positive semidefinite matrix the following is true:

$$x^T Hx \geq 0 \quad \forall x \in R^n \iff \forall \quad eig(H) \geq 0 \tag{2.5}$$

In short, the cost function of a general quadratic program is convex on R^n as long as H is positive semidefinite, page 8 [2].

As for the QP constraints described by the linear equations and inequations, this is where the topic of convexity analysis becomes more complicated. Firstly, the set of feasible solutions described by these constraints must be non-empty. As an example, if the problem presents two equality constraints $x = 1$ and $x = 2$, the problem cannot be solved. Secondly, no constraints should be redundant. Again, out of these two inequality constraints $x \leq 1$ and $x \leq 2$ is the second one redundant. These two requests imply certain limitations towards the matrices A and E. For that reason, an expanded, more limiting definition of a QP program is often presented in the literature. The equality constraint matrix E ought to be of dimension $m_1 \times n$ where $m_1 \leq n$ and $rank(E) = m_1$, i.e. the constraints are linearly independent. Similarly, for the inequality constraint matrix A. This ensures that the set of feasible solutions is an n-dimensional polyhedron which is always convex, see page 451 in [2]. Note that some methods exist to deal with problems with redundant constraints, but the feasibility set must be non-empty in any case, as per page 491 in [2].

To summarize a QP is said to be a convex problem as long as the quadratic cost function matrix H is positive semidefinite and the problem constraints limit the set of feasible solution to a polyhedron.

2.2 Model predictive control

The roots of model based predictive control can be tracked to the early 1960' when the renowned mathematician Rudolf E. Kalman presented the Linear Quadratic Regulator and subsequently the Linear Quadratic Gaussian (LQG) controller for stochastic systems. Although a powerful tool in theory, LQG presented several drawbacks which prevented it from mass adoption in the control industry. For example, the lack of input/state constraints handling, no way to incorporate process non-linearities, or no robustness guarantees [4]. Developed from the same theory of optimal control, model predictive control is a universal control methodology used to optimize the performance of an arbitrary system under constraints. This section will describe the basic concepts of MPC as well its ties to the concept of quadratic programming discussed in the previous section "Quadratic programming".

2.2.1 Dynamic models

The core idea behind MPC is to use a dynamic model of a system to predict its behavior and, based on that prediction, propose the best control decision. Unlike input/output based regulators, models and the ability to simulate them are key for the performance of model based controllers. A general non-linear system can be described by the following equations:

$$\begin{aligned} \frac{dx}{dt} &= f(x;u;t) \\ y &= g(x;u;t) \\ x(t_0) &= x_0 \end{aligned} \tag{2.6}$$

Where $x\ 2\ R^n$ is the state of the system, $u\ 2\ R^m$ the input and $y\ 2\ R^p$ the output [6]. Note that it is generally assumed that the input, state and output are functions of time unless specified otherwise.

A specific class of models especially useful in control engineering is the linear state space model of a system:

$$\begin{aligned} \frac{dx}{dt} &= A(t)x + B(t)u \\ y &= C(t)x + D(t)u \\ x(t_0) &= x_0 \end{aligned} \tag{2.7}$$

Where $A(t)\ 2\ R^{n\ \ n}$ is the state transition matrix, $B(t)\ 2\ R^{m\ \ n}$ describes the effect of input on the state dynamics, $C(t)\ 2\ R^{p\ \ n}$ is the output matrix and finally $D(t)\ 2\ R^{p\ \ m}$ is the direct link between input and output of the system omitting the state dynamics. For state matrices independent on time is such linear model called the *linear time invariant* state space model.

As MPC is used with discrete time models a discrete time LTI model can be defined similarly to its continuous time counterpart:

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \\ x_0 &= \text{given} \end{aligned} \tag{2.8}$$

2.2.2 Discrete model optimal control

To begin with, the problem of optimal control in discrete time system will be defined. Lets assume the goal is to optimally regulate a model described by some discrete time function - i.e., find such sequence of inputs that will guide the state of the system to the origin optimally according to a criterion function $J(x;u)$.

$$
\begin{aligned}
\min_{x_1 \dots x_N, u_0 \dots u_{N-1}} \quad & J(x, u) = \phi(x_N) + \sum_{k=0}^{N-1} L(x_k, u_k) \\
\text{s.t.} \quad & x_{k+1} = f(x_k, u_k) \\
& x_0 = \text{given} \\
& x_{lb} \leq x_k \leq x_{ub} \\
& u_{lb} \leq u_k \leq u_{ub}
\end{aligned}
\tag{2.9}
$$

When looking at the criterion function $J(x, u)$ one could propose to incorporate the terminal cost function $\phi(x_N)$ into the sum but it is generally a useful practice to to define the terminal cost independently. Another thing to consider is whether the final state x_N is free and penalized within $J(x, u)$ (as shown in 2.9) or whether x_N is fixed and therefore one of the optimization problem constraints [7].

2.2.3 Unconstrained linear MPC

Inspired by 2.9 the optimal regulation of a linear system with quadratic cost function can be defined. Note that this section, together with the following section "Constrained linear MPC" is based on the following materials [5, 7]. For now, the constraints on states and inputs will be omitted and addressed later.

$$
\begin{aligned}
\min_{x_1 \dots x_N, u_0 \dots u_{N-1}} \quad & J(x, u) = x_N^\top P x_N + \frac{1}{2} \sum_{k=0}^{N-1} x_k^\top Q x_k + u_k^\top R u_k \\
\text{s.t.} \quad & x_{k+1} = A x_k + B u_k \\
& x_0 = \text{given}
\end{aligned}
\tag{2.10}
$$

In this formulation, the matrices P, Q, R are weights that tune the final controller. Do I allow for slower regulation by increasing the input cost and thus limiting the action amplitude, or do I prefer to regulate the system as fast as possible by penalizing the state error more than the input cost? And how much do I care about the final state? For reasons which will emerge later, it is required that P, Q are symmetric real positive semi-definite matrices and R is symmetric real positive definite.

It still holds that the goal of this optimization problem is to find u^*, x^* that would minimize the cost function and thus optimize the state trajectory of the system. One approach to solve this problem is to reformulate it in a way such that the cost function $J(x_0, u)$ is only a function of inputs and the initial state. In literature this is referred to as *batch* or *sequential* approach. First, let us rewrite the cost function in matrix form.

$$J(x;u) = x_0^T Q x_0 + \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}^T \underbrace{\begin{bmatrix} Q & 0 & & & 0 \\ 0 & Q & 0 & & 0 \\ \vdots & 0 & \ddots & & \vdots \\ \vdots & \vdots & & Q & 0 \\ 0 & 0 & & 0 & P \end{bmatrix}}_{\mathcal{Q}} \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}^T \underbrace{\begin{bmatrix} R & & 0 \\ & \ddots & \\ 0 & & R \end{bmatrix}}_{\mathcal{R}} \underbrace{\begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{u} \tag{2.11}$$

And similarly the discrete state space equation.

$$\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} = \underbrace{\begin{bmatrix} B & 0 & & 0 \\ AB & B & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{N-1}B & A^{N-2}B & & B \end{bmatrix}}_{S} \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} + \underbrace{\begin{bmatrix} A \\ \vdots \\ A^N \end{bmatrix}}_{T} x_0 \tag{2.12}$$

By plugging 2.12 into 2.11 the final form of the cost function is obtained.

$$\begin{aligned} J(x_0;u) &= \frac{1}{2}u^T \underbrace{2(\mathcal{R} + S^T \mathcal{Q} S)}_{H} u + x_0^T \underbrace{2T^T \mathcal{Q} S}_{F^T} u + \underbrace{x_0^T(Q + T^T \mathcal{Q} T)x_0}_{c = \text{ const.}} \\ &= \frac{1}{2}u^T H u + x_0 F^T u + c \end{aligned} \tag{2.13}$$

And finally to obtain the optimal control sequence u^* the gradient of the cost function with respect to u will be calculated and solved for setting it equal to zero.

$$\nabla_u J(x_0;u) = Hu + Fx_0 \tag{2.14}$$

$$Hu + Fx_0 = 0 \implies u^* = -H^{-1}Fx_0 \tag{2.15}$$

This leaves us with the conclusion that the unconstrained linear MPC with a fixed finite horizon is simply a linear state feedback.

2.2.4 Constrained linear MPC

The main benefit of MPC its capability to handle input and state/output constraints. Given the optimal control problem as defined earlier in 2.10 more constraints will be added in form of lower and upper bounds of states and inputs.

$$\begin{aligned} u_{lb} \le u_j \le u_{ub}; \quad j = 0 \dots N-1 \\ x_{lb} \le x_i \le x_{ub}; \quad i = 1 \dots N \end{aligned} \tag{2.16}$$

These can be rewritten in matrix form as follow.

$$\begin{aligned} \underbrace{\begin{bmatrix} u_{lb} \\ \vdots \\ u_{lb} \end{bmatrix}}_{U_{lb}} \le \begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix} \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \le \underbrace{\begin{bmatrix} u_{ub} \\ \vdots \\ u_{ub} \end{bmatrix}}_{U_{ub}} \\ \underbrace{\begin{bmatrix} x_{lb} \\ \vdots \\ x_{lb} \end{bmatrix}}_{X_{lb}} \le \underbrace{\begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix}}_{I} \underbrace{\begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}}_{x} \le \underbrace{\begin{bmatrix} x_{ub} \\ \vdots \\ x_{ub} \end{bmatrix}}_{X_{ub}} \end{aligned} \tag{2.17}$$

Substituting for x from 2.12 and rewriting.

$$\begin{aligned} U_{lb} \le Iu \le U_{ub} \\ X_{lb} \le I \quad Su + Tx_0 \le X_{ub} \end{aligned} \tag{2.18}$$

We can now formulate the final constrained optimization problem using the results obtained in 2.13 and 2.18.

$$\begin{aligned} \min_u \quad & \frac{1}{2} u^T H u + x_0 F^T u \\ \text{s.t.} \quad & \begin{bmatrix} U_{lb} \\ X_{lb} - Tx_0 \end{bmatrix} \le \begin{bmatrix} I \\ S \end{bmatrix} u \le \begin{bmatrix} U_{ub} \\ X_{ub} - Tx_0 \end{bmatrix} \\ & x_0 = \text{given} \end{aligned} \tag{2.19}$$

This quadratic problem finally formulates the task of finding an optimal control sequence u for regulating some linear discrete-time system with constrained inputs and states. It is clearly an instance of a generic QP formulation defined in the previous section (see 2.1). The first row of constraints represents the box constraints of inputs defined directly, and the second row represents the affine constraints of inputs derived from constraints on system states.

Output constraints

It is often beneficial to enforce constraints onto the outputs instead of the states of the system. This can be achieved via small modification to the previous derivation. As in 2.18 we begin by defining the constraints in the following form.

$$y_{lb} \le y_i \le y_{ub}; \quad i = 1 \dots N \tag{2.20}$$

Rewriting as a matrix equation.

$$\underbrace{\begin{bmatrix} y_{lb} \\ \vdots \\ y_{lb} \end{bmatrix}}_{Y_{lb}} \le \begin{bmatrix} I & & \\ & \ddots & \\ & & I \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \le \underbrace{\begin{bmatrix} y_{ub} \\ \vdots \\ y_{ub} \end{bmatrix}}_{Y_{ub}} \tag{2.21}$$

Since $y_k = Cx_k$, by using the state space equation 2.12 the vector of outputs can be rewritten using the system matrices and inputs.

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \underbrace{\begin{bmatrix} CB & 0 & & 0 \\ CAB & CB & & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{N-1}B & CA^{N-2}B & & CB \end{bmatrix}}_{M} \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} + \underbrace{\begin{bmatrix} CA \\ \vdots \\ CA^{N} \end{bmatrix}}_{N} x_0 \tag{2.22}$$

Combining the last two equation 2.21 and 2.22 the output constraint inequality can be formulated as follows.

$$Y_{lb} \le Mu + Nx_0 \le Y_{ub} \tag{2.23}$$

Finalizing the formulation, the quadratic program representing input and output constrained optimal control problem is presented.

$$\begin{aligned} \min_{u} \quad & \frac{1}{2} u^\top H u + x_0 F^\top u \\ \text{s.t.} \quad & \begin{bmatrix} U_{lb} \\ Y_{lb} - Nx_0 \end{bmatrix} \le \begin{bmatrix} I \\ M \end{bmatrix} u \le \begin{bmatrix} U_{ub} \\ Y_{ub} - Nx_0 \end{bmatrix} \\ & x_0 = \text{given} \end{aligned} \tag{2.24}$$

Output tracking

Until this point, the obtained results were derived for the system regulation problem, which is guiding the system's states to the origin. More often than not is the actual use case the so-called output tracking - the task of controlling the system in such a way that the outputs follow a set trajectory. Excluding a few necessary modifications, the approach to reformulating the tracking problem as a quadratic program is almost identical to the regulation problem. Only the key idea and results will therefore be presented and the reader will be referred to several sources for more detailed derivation.

To begin with, the quadratic cost function of the optimization problem must be reformulated. Given the output y_k and the reference trajectory r_k the tracking error e_k can be defined.

$$e_k = r_k - y_k = r_k - Cx_k \tag{2.25}$$

The goal is then to minimize the tracking error as well as the inputs. But in order to formulate the new cost function correctly, the control increment $\Delta u_k = u_k - u_{k-1}$ must be introduced. This is due to the reason that generally, to drive e_k to zero, the system inputs are not zero. But a cost function minimizing both the error e_k and the inputs u_k does not take this into account, and it would therefore fail to provide an input trajectory leading to zero tracking error. By introducing Δu_k as the new input signal, the previous input $u_{k-1} = x^u_k$ will be introduced as a new state variable of the system. This leads to the following augmented state-space equations.

$$\begin{aligned} \begin{bmatrix} x_{k+1} \\ x^u_{k+1} \end{bmatrix} &= \underbrace{\begin{bmatrix} A & B \\ 0 & I \end{bmatrix}}_{\hat{A}} \underbrace{\begin{bmatrix} x_k \\ x^u_k \end{bmatrix}}_{\hat{x}_k} + \underbrace{\begin{bmatrix} B \\ I \end{bmatrix}}_{\hat{B}} \Delta u_k \\ y_k &= \underbrace{\begin{bmatrix} C & 0 \end{bmatrix}}_{\hat{C}} \begin{bmatrix} x_k \\ x^u_k \end{bmatrix} \end{aligned} \tag{2.26}$$

Using this augmented system the unconstrained optimization problem of linear system tracking can be defined.

$$\begin{aligned} \min_{\hat{x}_1 \dots \hat{x}_N;\, \Delta u_0 \dots \Delta u_{N-1}} \quad & J(\hat{x}; \Delta u) = \left(r_N - \hat{C}\hat{x}_N\right)^\top P \left(r_N - \hat{C}\hat{x}_N\right) \\ & + \frac{1}{2}\sum_{k=0}^{N-1} \left[\left(r_k - \hat{C}\hat{x}_k\right)^\top Q \left(r_k - \hat{C}\hat{x}_k\right)\right] \\ & + \frac{1}{2}\sum_{k=0}^{N-1} \Delta u_k^\top R \Delta u_k \\ \text{s.t.} \quad & \hat{x}_{k+1} = \hat{A}\hat{x}_k + \hat{B}\Delta u_k \\ & \hat{x}_0 = \text{given} \end{aligned} \tag{2.27}$$

Utilizing the same sequential approach as in section "Constrained linear MPC" the problem 2.27 can be reformulated into the following form [5, 8, 9].

$$J(\hat{x}_0; \Delta u) = \frac{1}{2} \Delta u^\top \hat{H} \Delta u + \begin{bmatrix} \hat{x}_0^\top & r_0^\top \end{bmatrix} \hat{F}^\top \Delta u \tag{2.28}$$

Where $\Delta u = [\Delta u_0; \dots; \Delta u_{N-1}]^\top$ and $\hat{F}; \hat{H}$ are matrices obtained during the derivation. Adding the constraints on inputs, states and outputs is again similar to the approach presented on the mpc regulation problem and results in inequality affine constrained QP.

Transforming affine constraints into box constraints using slack variables and softening

Given an MPC problem formulation with constrained states or outputs, the final quadratic program generally has affine constraints as per the results presented in 2.24 and 2.19. Such a problem is already solvable using today's solvers, but it could be beneficial to transform the affine constraints to obtain a purely box constrained QP as such a problem is generally easier to solve and supported by a greater number of solvers. Furthermore, relaxing the hard state/output constraints increases the odds of finding feasible solutions for the cost of some constraint violations [10].

The approach to this transformation is following. Given a generic box and affine constrained QP problem

$$\begin{aligned} \min_{x} \quad & \frac{1}{2}x^\top Hx + f^\top x \\ \text{s.t.} \quad & x_{lb} \leq x \leq x_{ub} \\ & a_{lb} \leq Ax \leq a_{ub} \end{aligned} \tag{2.29}$$

where $x \in \mathbb{R}^{n \times 1}$ and $A \in \mathbb{R}^{m \times n}$, introduce a new vector of slack variables $\epsilon \in \mathbb{R}^{m \times 1}$. These variables will then be constrained by the lower and upper bounds $a_{lb}; a_{ub}$ and the cost function will be extended by a new element as follows.

$$J(x;\epsilon) = \frac{1}{2}x^\top Hx + f^\top x + \frac{1}{2}(Ax - \epsilon)^\top W(Ax - \epsilon) \tag{2.30}$$

The matrix $W \in \mathbb{R}^{m \times m}$ is a new weight pushing the new problem to respect the original affine constraints. Note that it makes sense to select W diagonal either with constant diagonal elements or with varying elements depending on the "importance" of individual constraints. Rewriting 2.30 results in.

$$J(x;\epsilon) = \frac{1}{2}x^\top Hx + f^\top x + \frac{1}{2}(x^\top A^\top WAx - \epsilon^\top WAx - x^\top A^\top W\epsilon + \epsilon^\top W\epsilon) \tag{2.31}$$

And after rewriting the cost function in vector form the transformation of the QP problem can be finalized.

$$\begin{aligned} \min_{x;\epsilon} \quad & \begin{bmatrix} x^\top & \epsilon^\top \end{bmatrix} \begin{bmatrix} H + A^\top WA & -WA \\ -A^\top W & W \end{bmatrix} \begin{bmatrix} x \\ \epsilon \end{bmatrix} + \begin{bmatrix} F^\top & 0^\top \end{bmatrix} \begin{bmatrix} x \\ \epsilon \end{bmatrix} \\ \text{s.t.} \quad & \begin{bmatrix} x_{lb} \\ a_{lb} \end{bmatrix} \leq \begin{bmatrix} x \\ \epsilon \end{bmatrix} \leq \begin{bmatrix} x_{ub} \\ a_{ub} \end{bmatrix} \end{aligned} \tag{2.32}$$

This QP problem finally represents the transformation of the original affine constrained problem 2.29 into a box and soft constrained problem. Getting rid of these affine restrictions is at the cost of larger number of optimization variables ($n + m$ instead of n) and potential violation of the original constraints.

2.2.5 Receding horizon control

With the obtained result, the general approach of MPC regulation can now be described. At each time step, the current state of the system is obtained. This current state can be denoted as an initial state at time $t_0 = 0$ as an LTI system is assumed. Given x_0, the QP problem is formulated, solved, and the optimal input trajectory u is obtained. As a next step, the first input u_0 is applied to the system, and in the next time step, the whole process is repeated. This strategy is often referred to as *receding horizon control*. See figure 2.1 for an illustration of a single MPC regulation step.

Figure 2.1: Basic MPC horizon preview

Figure 2.2: MPC with control horizon and input blocking

Control horizon and input blocking

The most computationally and memory demanding step of the receding horizon control is solving a new quadratic problem at each time step. Given the prediction horizon N and the number of inputs m, the QP optimizes over

$N \quad m$ variables, and it is up to the designer to select N large enough for acceptable controller performance but small enough to keep the dimensionality of the QP within reasonable bounds.

In order to reduce said dimensionality, one strategy is to define so called prediction horizon N_p and control horizon N_u such that $N_u \quad N_p$. The problem is then reformulated in such way that the input is assumed constant after N_u steps which reduces the number of optimization variables. Another approach is to introduce input blocking. Not only is a constant input assumed after the control horizon but further blocks are defined within the control horizon in which is the input again deemed constant. Best explained with a simple example (see 2.2), let's assume $N_u = 7$, $N_p = 9$ and input is blocked via the following strategy $u_{blocks} = [1; 1; 2; 3]$, where u_{blocks} denotes the length of blocks (sample vise) over which $u = 0$. Note that $\sum u_{blocks} = N_u$.

2.3 Automotive grade communication buses

With the first two sections of this chapter describing the concept of quadratic programming and its use in model predictive control, this section will focus on another key topic of this thesis, automotive grade bus communication. As the number of electronic control units in a vehicle grew, the need to replace their point-to-point connections with more space and cost-efficient solutions increased. This is where the concept of a fieldbus comes in. A fieldbus is a distributed computer network that allows multiple nodes (ECUs) to be connected to said bus and exchange messages with each other, thus removing the need for designated connections between individual pairs of nodes [11]. Typically, multiple communication buses operate in a vehicle simultaneously, interconnecting vehicle subsystems based on their needs and importance. The following figure 2.3 shows an example distribution of vehicle subsystems with regards to their communication speed and safety requirements.

Figure 2.3: Distribution of vehicle subsystems based on their requirements, adopted from [12]

Throughout the years, a large number of automotive field buses have been developed. For example, the Media Oriented System Transport (MOST) network is used for infotainment data transmission due to its high bandwidth. On the other hand, it only supports the ring network topology and is therefore not suitable for more safety critical data [12]. Another popular solution is the FlexRay communication bus. Suitable for sensitive data due to its inherent redundant and deterministic properties, its main disadvantage is the need for all nodes to be made completely aware of the whole network topology. This makes designing a network using FlexRey more complicated than using some alternatives [12]. For the usecase of this project, three other automotive buses will be described in more detail.

2.3.1 Local interconnected network

Local interconnected network (LIN) typically operates over the Low priority data shown in figure 2.3. Some other usecases include temperature, light and rain sensors reading, headlights control and a number of other functions. An individual LIN bus message (frame) is shown in the following picture 2.4. It consists of the following fields:

- Break: Start of frame notification for all bus nodes
- Sync: Predefined bit sequence used for bit rate synchronization
- ID: Identifier, some messages are meant only for some nodes
- Payload: Actual data content of the frame
- Checksum: Used to validate the payload contents (sometimes including the ID as well)

Figure 2.4: Local interconnected network frame, adopted from [13]

The main advantages of the LIN bus are [13]:

- Low-cost and simple solution
- Single wire with vehicle chassis acting as ground
- Master slave configuration - option to add and remove slave nodes without affecting other slave nodes, up to sixteen slave nodes
- Checksum error detection
- Collision avoidance usingtime divided media access

On the other hand, LIN bus also has several disadvantages. Mainly its low bit-rate of maximum 20 kbit/s and small payload size in a single frame which is at most 8 bytes [12]. Remember that the scope of this project includes using a communication bus to send a full box constrained QP problem as defined in 2.32 and subsequently receive its solution. The slow communication speed of the LIN bus thus poses a serious problem. To illustrate this, let's do some simple calculations.

QP sending time using the LIN bus

Given the number of optimization variables n_{var}, the total number of numeric values n_{tot} within a box constrained QP can be determined as follows.

$$n_{tot} = n_H + n_F + n_{lb} + n_{ub} + n_{x_0} = n_{var}^2 + 4n_{var} \tag{2.33}$$

Where $n_H; n_F; n_{lb}; n_{ub}; n_{x_0}$ denote the number of numeric values corresponding to individual parts of the box constrained QP problem. To reduce n_{tot}, two assumptions will be made. Firstly, H is symmetric and only its upper triangular part needs to be sent, therefore $n_H = n_{var}(n_{var}+1)/2$. Secondly, the initial condition x_0 is not known and therefore not transmitted as well. These assumptions result in:

$$n_{tot} = \frac{n_{var}(n_{var}+1)}{2} + 3n_{var} \tag{2.34}$$

Figure 2.5: QP size and send time dependency using the LIN bus with bit-rate of 20 kbit/s

Additionally, these numeric values will be represented using the 32 bit long single precision floating point format, and only these values will need to be transmitted via the bus. Given the payload size of 64 bits and the total size of the frame, which is 108 bits (as per 2.4), it can be reasoned that to transmit two elements of the QP problem takes precisely 108 bits of information. Finally, taking into account the maximum bit rate of the LIN bus, the dependency between the QP size and the sending time is plotted in the following figure 2.5. Note that this figure represents an upper estimate as an ideal scenario is assumed where the bus works at 100% all the time and no overhead information is transmitted.

By looking at this dependency it is obvious that using LIN bus is basically infeasible for the purposes of this project as the communication speed would simply be too slow for any reasonable controller.

2.3.2 Controller Area Network

Controller Area Network (CAN) is another example of a frequently used automotive grade communication bus. Since its initial release by Robert Bosch GmbH.— in the late 1980s, several versions of the CAN specification were developed, and it went on to become the most widely used communication network in the automotive industry [14]. To refer to the in-vehicle systems distribution discussed previously in 2.3, CAN is typically used for the control and safety critical data applications due to its robustness and higher bit rate of up to 1 Mbit/s. It has not been popularized only in the automotive industry but in other industries as well, such as aviation, ship industry and/or robotics.

In this thesis, the CAN 2.0 specification will be discussed, sometimes referred to as high speed CAN. Referring to the seven-layer Open Systems Interconnection (OSI) model, the CAN 2.0 specification is defined for the two "lowest layers" - the physical layer and data link layer, both defined in their respective ISO standards [16]. In a typical embedded application, the implementation of CAN bus bypasses the connection between application and data link layers to maximize the performance and minimize the resource overhead. Alternatively, including the rest of the OSI layers within the application requires a higher layer protocol such as CANopen or SAE J1939 [17]. An overview is shown in the following figure 2.6.

Figure 2.6: Controller Area Network arbitration phase state diagram, adopted from [16, 17]

CAN bus arbitration

Unlike the LIN bus, CAN is not a master-slave standard and is instead a multi-master bus, meaning that any node can initiate data transfer on its own initiative. Typically, networks operating in the multi-master configuration have to implement some form of data collision detection and recovery. The CAN standard solves this issue of data collision by defining a so-called arbitration phase at the beginning of each message, see 2.9. In case two or more nodes start transmitting simultaneously, the network automatically gives exclusive access to the node transmitting the highest priority message, which is detected by the remaining nodes.

The main idea behind the arbitration process could be described as follows. The bus finds itself either in a dominant or recessive state at all times. During the frame transmission, a zero bit sets the bus to the dominant state a one bit sets it to the recessive state. Furthermore dominant state always overrides the recessive one in case of simultaneous transmitting. Practically, the state of the bus is determined by the logical AND operation where all nodes' inputs give the final state of the bus, note that nodes in listening mode can be viewed as outputting recessive bits [15].

Given this property of the CAN bus, the logic behind giving access to the most important node/frame is straightforward. If multiple nodes begin transmitting a frame at the same time, the frame transmitting the message with the highest priority (lowest ID) gets access. This is best explained with the following state diagram 2.7.

Figure 2.8 shows an example of the arbitration. Two nodes begin transmitting simultaneously, meaning their start of frame bits overlap. Following that, both begin transmitting the IDs of their messages, beginning with the most significant bit. The first two bits match, so no collision is detected. On the other hand, during the third bit window, the first node notices that it transmitted a recessive bit, but the bus is in a dominant state, thus meaning another node is transmitting a higher priority message. The first node enters listening mode and tries to transmit its frame from the beginning once the second node is finished with its transmission. As long as each message is labeled with a unique ID, lower ID messages are always transmitted first.

Figure 2.7: Controller Area Network arbitration phase state diagram, adopted from [15]

Figure 2.8: An example of arbitration process between two nodes on the controller network area bus

CAN bus data frame

The previous subsection dealt with the issue of CAN bus arbitration which utilizes a dedicated part of the data frame. The CAN frame as a whole is shown in the figure 2.9 and it consists of the following slots:

Figure 2.9: Controller Area Network data frame, adopted from [15, 16, 18]

- SOF: Dominant bit indicating the start of frame
- ID: Message identifier used for arbitration and message filtering, two CAN 2.0 specifications are used, defining either 11 or 29 bit long ID
- RTR: Remote transmission request, set to dominant for data request frame and recessive for data frame
- Control: Control section contains three parts
    - Identifier extension bit: Dominant for 11 bit ID, recessive for 29 bit ID
    - Data length code (DLC), 4 bit code declaring the size of the payload (0-8 bytes)
    - 1 reserved bit, defined as a dominant
- Payload: Actual data contents
- CRC: 15 bit long cyclic redundancy check plus recessive bit delimiter
- ACK: 1 bit long acknowledgment slot plus recessive bit delimiter
    - The acknowledgment during the frame transmission works as follows. First the sender sets the ACK bit as recessive. Once a listening node detects this, it overwrites the bus to the dominant state. This is again detected by the sending node to which it indicates successful acknowledgment [18].
- EOF: 7 bit long recessive slot indicating the end of frame

CAN bus robustness

As aforementioned, in the automotive setting is the CAN bus heavily utilized in safety-critical applications. This is due to its robustness both on physical and protocol levels. Physically the bus is realized as a balanced differential medium using a twisted pair with 120 characteristic impedance, which offers high noise immunity [18]. The protocol itself includes several error detection mechanisms.

Using the cyclic redundancy check, the listening nodes simply do not acknowledge proper frame reception during the acknowledgment slot. Furthermore, as described in the previous subsection "CAN bus data frame", the frame has several predefined slots with recessive bits, which also aids with error detection. On a lower level, the transmitting nodes also check for the state of the bus after a bit has been transmitted. If the bus state does not correspond to said bit an error is raised. Note that this mechanic is disabled during the arbitration and acknowledgment slots for obvious reasons. Finally, as the CAN protocol does not use a return to zero bit representation, bit stuffing was introduced. In case five same level consecutive bits are transmitted, a bit of opposing logic level is "stuffed" into the frame. This ensures that if the transmitting node gets stuck transmitting a constant logic level the receiving nodes detect this and respond with an error frame. Bit stuffing is disabled during the seven bit long EOF phase [18, 14].

QP sending time using the CAN bus

Given the popularity, robustness, and typical use cases of the CAN bus, it seems like a good candidate for the purposes of this project. Similarly, as in section 2.3.1 let's create a simple plot demonstrating the relationship between the size of a box constrained QPn_{var} and the ideal time it would take the CAN bus to transmit such QP. Again, the number n_{tot} of single-type float values needed to be sent is given by this equation 2.34. As shown in the structure of the CAN frame, see 2.9, one frame can hold two of these values, each at 32 bits, and the total length of the frame is 108 bits using the standard 11 bit ID frame. In reality, the total number of frame bits can be higher due to the bit stuffing mechanic.

Excluding the end of the frame identifier, the length of the frame becomes 101 bits. Given this, the maximum number possible of stuffed bits is 25, which increases the total size of the frame to 133 bits. Assuming a 500 kbit/s bit rate, the following figure shows the ideal box QP sending times for both the 108 and 133 bit long CAN bus frames. Although these time values are highly optimistic due to a number of reasons, it seems that the higher bit rate of the CAN bus could allow for the usage of this protocol for the purposes of this project.

Figure 2.10: QP size and send time dependency using the CAN bus with bit-rate of 500 kbit/s

2.3.3 Controller Area Network Flexible Data-Rate

Despite its popularity, the classical CAN protocol began to be insufficient for the needs of the fast-evolving automotive industry. With the limited bit rate of 1 Mbit/s and the payload size of a single frame being eight bytes, car manufacturers would have to look elsewhere for a faster and more efficient solution. When increasing the communication speed of the CAN protocol, there are two issues. Firstly, the payload size of a single CAN frame could be increased and could potentially transfer more than two bytes of data. But with an unchanged bit rate, longer frames would block the bus for a longer time, and safety-critical data could be therefore delayed [19].

The second issue arises by trying to increase the bit rate itself. But even here is the classical CAN bus limited due to its arbitration and message acknowledgment specifications. The key problem is that the network must be set up in such a way that during a single bit time window is the "information wave" able to propagate between the two physically furthest nodes back and forth. Otherwise, the frame collision during the arbitration phase or message acknowledgment could be missed. Given the limits imposed by the finite speed of light, for 1 Mbit/s bit rate is the maximum length of the CAN bus wiring forty meters, and even less is used in practice [19]. Both of these issues are addressed in the newer CAN FD protocol specification, presented by Bosch in 2012 and ISO standardized in 2015 [21].

CAN FD bit rate switching

The main idea behind the CAN FD protocol was to introduce two bit rates and switch between them during a single frame transmission. As explained earlier, the bit rate is only really limited during the arbitration and acknowledgment phases of the CAN frame. So why not increase the communication speed, at least during the transmission of the payload? This is exactly what CAN FD specifies. The protocol divides its frame into three main sections, see 2.11. The header, payload, and tail where the header and tail are transmitted using the so-called *arbitration bit rate* - the same factors limit this bit rate as in the classical CAN protocol, maximum 1 Mbit/s. The payload is then transmitted using the faster *payload bit rate*, utilizing the fact that during the payload phase, the information flows purely in a single direction. The payload bit rate is at most 5 Mbit/s, although even higher values might be feasible [19].

CAN FD bus data frame

The CAN FD frame is similar to the one of the standard CAN but several new fields were added due to the increased complexity of the protocol. A single frame is shown in the following figure 2.11. Note that this frame structure takes into account the standard 11 bit identifier length, for a 29 bit identifier is the header of the frame slightly modified.

Figure 2.11: Controller Area Network Flexible Data-Rate frame for standard 11 bit ID length, adopted from [19, 21]

The meaning of some of the individual fields within the data frame has already been explained in the "CAN bus data frame" subsection, only the new fields will therefore be further described [19, 20]:

- RRS: Remote request substitution, replaces the RRT bit from the CAN protocol, defined as always dominant
- IDE: Identifier extension bit, dominant for standard ID, recessive for extended ID
- FDF: Defined as recessive, in CAN frame this bit is dominant (called the *reserve* bit)

- res: Reserved bit, defined as dominant
- BRS: Bit rate switch, dominant denotes no bit rate switching, recessive signalizes the use of different payload bit rate
- ESI: Error state indicator, transmitted dominant by error active nodes, recessive by error passive nodes
    - Error state is a new robustness mechanic introduced by the CAN FD protocol. Each node within the CAN FD network is initialized as error active. Each node has two inner counters, the transmit and receive error counter. These counters are increased/decreased according to a set of rules but once a certain value is surpassed the node is set to error passive. When the counters increase even further, the node becomes silent [22].
- DLC: Data length code, encoded by four bits, therefore the data lengths corresponding to the code are $\{0, \ldots, 8, 12, 16, 20, 24, 32, 48, 64\}$
- SBC: Stuffed bit counter, counts the number of stuffed bits before the CRC field for improved error detection
- CRC: Cyclic redundancy check plus CRC delimiter bit, CRC length is either 17 bits for data of lengths up to 16 bytes, or 21 bits for longer data
- FSB: Fixed stuff bits, to further improve error detection the frame specifies several slots with stuffed bits, the value of these bits is the opposite of the previous bit

QP sending time using the CAN FD bus

As in the previous cases of CAN and LIN protocols, given the knowledge about the structure of the data frame and bit rates of the CAN FD bus, some upper estimation about the sending times of a box constrained QP can be made. The setup will be similar as previously in "QP sending time using the CAN bus. The 11 bit identifier and 64 byte data field will be assumed which leads to 577 bit long frames (for simplicity, only fixed stuffing will be assumed), each having the capacity to transmit up to 16 single type float values.

For the CAN FD protocol, it is needed to take the dual bit rates into account. Each frame has a certain number of bits to be transmitted using the arbitration speed and another number of bits to be transmitted using the payload speed. To calculate the total time, it would take the bus to transmit this frame, an "average" bit rate can be obtained as follows:

$$\text{Average bit rate} = \frac{\text{Total number of bits}}{\frac{\text{Payload bits}}{\text{Payload bit rate}} + \frac{\text{Arbitration bits}}{\text{Arbitration bit rate}}} \tag{2.35}$$

The following figure 2.12 then shows the ideal sending times of variously sized box constrained QP problems with 500 kbit/s arbitration bit rate and 2 Mbit/s payload bit rate. The figure was plotted for CAN FD frames with different data field sizes to visualize how the sending time decreases with the growing length of the data field.

As a side note, it is interesting to compare the sending times using the standard CAN bus (see figures 2.10 and 2.12) and using the CAN FD bus with 8 bytes long data field. Although the bit overhead of "non-data" bits is larger in the case of CAN FD, and one could argue that in this sense is the standard CAN more efficient, the higher payload bit rate of CAN FD still ensures that the sending times are approximately twice as fast. For larger data fields (12 bytes and more) is the CAN FD not only faster but also more efficient than the standard CAN, as shown in the table 2.1.

Data bytes [-]	8	12	16	20	24	32	48	64
CAN frame eff. [%]	0.59	-	-	-	-	-	-	-
CAN FD frame eff. [%]	0.52	0.61	0.68	0.71	0.75	0.80	0.85	0.88

Table 2.1: CAN and CAN FD frame efficiencies defined as a number of "useful" data bits divided by the number of total frame bits, note that only fixed bit stuffing is assumed

Figure 2.12: QP size and send time dependency using the CAN FD bus with various data filed sizes and bit rates of 500/2000 kbit/s

Chapter 3

Embedded solver ECU setup

With the previous chapter discussing the theoretical background of the three key topics of this thesis, this chapter will focus on the setup of the embedded QP solver. The main idea of the project is quite simple. Take some two ECUs, the first one will generate arbitrary QP problems and send them to the second ECU. The second board will then solve these problems and transmit their solution to the original sender. For the purposes of this thesis, the ECU sending the problems will be a desktop computer running Matlab & Simulink, whereas the system hosting the QP solver will be an automotive grade embedded controller described further in an upcoming section. Furthermore, the communication will be realized using the CAN and CAN FD bus. A basic overview of this setup is visualized in the following figure 3.1.

Figure 3.1: Basic overview of the embedded QP solver setup

3.1 ECU hardware

The controller used for this project is the TC387 32 bit single-chip microcontroller developed by Infineon Technologies–. It offers a high-performance architecture along with advanced features for connectivity, security, and functional safety. It is therefore suited for a wide range of automotive applications, including the control domain and data fusion applications. Some of the specific systems utilizing this controller are air-bag system, braking system, active suspension control, and others [23]. The main features of the controller are listed below [23]:

- Four TriCore — cores running at 300 MHz
    - Both fixed and floating point representation supported on all cores
- 10 MB Flash memory (Error-Correcting Code protected)
- 1.5 MB static RAM memory (Error-Correcting Code protected)
- Peripherals: Ethernet, CAN (FD), FlexRay, LIN, I^{2}C, ...
- Up to ASIL-D/SIL3 safety requirements supported
- AUTOSAR 4.2 supported

To use the microcontroller for development purposes, a custom printed circuit board was ordered by Garrett Motion Inc. and manufactured by ADWITECH systems s.r.o. The final PCB shown in picture 3.3 hosts not only the controller itself but also a power management unit, peripheral transceivers plus connectors, and other supporting circuitry. The basic overview of this PCB is shown in the presented block diagram 3.2.

Figure 3.2: Basic overview of the printed circuit board hosting the TC387 microcontroller, designed and developed by Garrett Motion Inc. and ADWITECH systems s.r.o.