Embedded Model Predictive Control Using Robust Penalty Method

Abhijith Sharma\textsuperscript{1}, Chaitanya Jugade\textsuperscript{1}, Shreyawalkar\textsuperscript{1}, Vaishali Patne\textsuperscript{1}, Deepak Ingole\textsuperscript{2}, and Dayaram Sonawane\textsuperscript{1}

Abstract— Model predictive control (MPC) has become a hot cake technology for various applications due to its ability to handle multi-input multi-output systems with physical constraints. The optimization solvers require considerable time, limiting their embedded implementation for real-time control. To overcome the bottleneck of traditional quadratic programming (QP) solvers, this paper proposes a robust penalty method (RPM) to solve an optimization problem in a linear MPC. The main idea of RPM is to solve an unconstrained QP problem using Broyden–Fletcher–Goldfarb–Shannon (BFGS) algorithm. The beauty of this method is that it can find optimal solutions even if initial conditions are in an infeasible region, which makes it robust. Moreover, the RPM is computationally inexpensive as compared to the traditional QP solvers. The proposed RPM is implemented on resource-limited embedded hardware (STM32 microcontroller), and its performance is validated with a case study of a citation aircraft control problem. We show the hardware-in-the-loop co-simulation results of the proposed RPM and compared them with the active set method (ASM) and interior point method (IPM) QP solvers. The performance of MPC with the aforementioned solvers is compared by considering the optimality, time complexity, and ease of hardware implementation. Presented results show that the proposed RPM gives the same optimality as ASM and IPM, and outperforms them in terms of speed.

Index Terms— Model predictive control, online optimization, penalty method, hardware implementation, linear systems.

I. INTRODUCTION

The model predictive control (MPC) has played a comprehensive role in process industries. \cite{1}. This algorithm has been widely accepted in process industries. The underlying reason for this popularity is the less human intervention requirement and ease in tuning for operators. It has been implemented under restricted conditions a myriad of times \cite{2}, \cite{3} in petrochemical, oil, and gas, and food processing industries \cite{2}. This is possible because of its capability of handling multi-input and multi-output (MIMO) systems with constraints on inputs, states, and outputs \cite{4}. It has been successfully imbibed by aerospace, autonomous vehicles, automobile sectors in the present scenario \cite{5}.

MPC is an optimal control algorithm wherein the control action is obtained by solving a constrained convex optimization problem (generally convex quadratic programming (QP) problem) over the finite-time interval for the system’s current state at each sampling time. In general, MPC formulation leads to three components of the system’s response: heading of process dynamics (state estimation), final destination (steady-state target optimization), and the best set of control (input) that would drive the states to target. The initial control (input) of the sequence obtained is implemented, and then the entire calculation is repeated for the subsequent control cycles \cite{4}, Chapter 12. This control technique of repeatedly solving a constrained problem over moving time to choose a control horizon is called receding horizon control (RHC). This introduces inherent negative feedback to the system, allowing automatic compensation of disturbances in the systems \cite{4}, Chapter 12.

There are several state-of-the-art solvers developed to solve the underlying QP problem in an MPC. Examples of such solvers are CVXGEN, \(\mu\text{AO-MPC}\), ECOS, qPOASES, etc. CVXGEN generates custom c code for embedded applications, but cannot explicitly handle feasibility and constraints. \cite{5}. The \(\mu\text{AO-MPC}\) offers low memory utilization for real-time linear MPC implementation, but the execution time is slow. ECOS is based on a primal-dual IPM. The majority of the solvers employ algorithms that are based on the active set method (ASM) and the interior point method (IPM) \cite{5}–\cite{9}. The active set method considers the set of active constraints for each iteration to solve the QP problem. As a result, computation time is directly proportional to the number of active constraints. In contrast, the computation time of IPM is relatively constant, regardless of the number of active constraints. This time can be large enough compared to ASM in cases of a small QP problem with fewer constraints and variables \cite{10}. It has been shown that embedded implementation of the IPM and ASM is limited to the small problems and needs more resources, i.e., memory, clock cycles, etc. Authors in \cite{11} presented the approaches of solving QP problems using ASM and IPM methods and compared the performances for an FPGA implementation for MPC applications. They have mentioned that ASM can make the system unstable due to numerical errors for the control system applications compared to IPM.

To mitigate this problem many authors worked on different approaches. Authors in \cite{12} presented the novel idea to use the posit number system to reduce numerical error in ASM and improving the performance. In \cite{13}, the authors proposed an interesting method to identify the worst-case number of iterations required for the primal and dual active-set algorithms to reach optimality. Authors in \cite{14} introduced the convergence depth control method into the interior-point method to accelerate the QP solving process for embedded MPC implementation.

For large scale problems, alternating method of multipliers
(ADMM) is preferred. However, due to large data, it can become complex for implementation [15]. Another method that is used for optimization of QP problem is gradient descent method. These methods use sensitivity information to evaluate the search directions [16]. The real-time dynamic systems introduce the principle challenge for the implementation of MPC. These systems demand a high sampling rate with embedded implementation on hardware with limited resources. To overcome the bottleneck of traditional QP solvers, we propose a robust penalty method (RPM) based on linear MPC and its microcontroller implementation. Because of its simplicity and intuitive appeal, this approach is often used in practice. The idea of the penalty method is to convert the original constrained QP problem to an unconstrained optimization problem. This reduces the solving time for the problem. We have developed a robust penalty method by incorporating Broyden–Fletcher–Goldfarb–Shannon (BFGS) algorithm to solve the unconstrained QP problem. The concept of the penalty method [17, Chapter 17] is not new but has some limitations. The original penalty method stops when it reaches the boundary of constraints. This happens because no constraints will be violated when it is inside a feasible region; hence no penalty function is used. But, there can be a scenario when the optimal value is completely inside the feasible region and not at the boundary. To handle this problem, we propose to solve the unconstrained QP problem without penalty. This will give us an optimal solution even if the initial guess is in the infeasible region. The proposed penalty method is implemented on an STM32 microcontroller. The hardware-in-the-loop (HIL) co-simulation results of the proposed penalty method are presented for a benchmark QP problem and a case study of citation aircraft control problem using MPC. The main contribution of this paper is to develop a robust penalty method and its embedded implementation to highlight the performance of the penalty approach in solving the MPC problem as well as in standalone QP problems.

The paper is organized as follows: Section II describes the MPC problem formulation. In Section III, a detailed description of the robust penalty method is presented. In Section IV, embedded implementation of RPM, ASM, and IPM for linear MPC is discussed. Section V presents the hardware-in-the-loop (HIL) results of all QP methods for the benchmark QP problem and citation aircraft control problem. At the end, conclusion with a summary of the work and possible future scope is stated in Section VI.

II. LINEAR MODEL PREDICTIVE CONTROL

This section describes the linear model predictive controller and its QP problem formulation for reference tracking.

A. Prediction Model

Consider a discrete-time version of the linear-time invariant (LTI) system (1).

\[ x(t + T_s) = Ax(t) + Bu(t), \quad (1a) \]
\[ y(t) = Cx(t) + Du(t), \quad (1b) \]

where \( x(t) \in \mathbb{R}^n \) is the system state vector, \( u(t) \in \mathbb{R}^l \) is the system input vector, \( y(t) \in \mathbb{R}^m \) is the system output vector, and \( T_s \) is the sampling time. Moreover, \( A \in \mathbb{R}^{n \times n}, B \in \mathbb{R}^{n \times l}, C \in \mathbb{R}^{m \times n}, \) and \( D \in \mathbb{R}^{m \times l} \) are system matrices. We assume that the pair \((A, B)\) is stabilizable, and \((C, A)\) is detectable.

B. Optimal Control Problem

A constrained finite-time optimal control (CFTOC) problem considering model in (1) for reference tracking can be represented as follows:

\[
\min_U \sum_{k=0}^{N-1} (y_k - y_{r,k})^T Q(y_k - y_{r,k}) + \Delta u_k^T R \Delta u_k, \quad (2a)
\]

subject to

\[
\begin{align*}
x_{k+T_s} &= Ax_k + Bu_k, & k = 0, \ldots, N - 1, \quad (2b) \\
y_k &= Cx_k + Du_k, & k = 0, \ldots, N - 1, \quad (2c) \\
\Delta u_k &= u_k - u_{k-1}, & k = 0, \ldots, N - 1, \quad (2d) \\
x_0 &\in X, & k = 0, \ldots, N - 1, \quad (2e) \\
u_k &\in U, & k = 0, \ldots, N - 1, \quad (2f) \\
y_k &\in Y, & k = 0, \ldots, N - 1, \quad (2g) \\
u_{-1} &= u(t - T_s), \quad (2h) \\
x_0 &= x(t), \quad (2i)
\end{align*}
\]

where \( Q \in \mathbb{R}^{n \times n} \) and \( R \in \mathbb{R}^{l \times l} \) are the weighting matrices, with \( Q \succeq 0 \) and \( R > 0 \) to be positive definite. We denote by \( N \) the prediction horizon, \( x_{k+1} \) as the vector of predicted states at time instant \( k, U = \{u_0, \ldots, u_{N-1}\} \) as the sequence of control actions, \( y_{r,k} \) is the output reference trajectory to be tracked, and \( x_0 \) and \( u_{-1} \) are the given initial conditions. States, inputs, and outputs belong to polytopic constraints \( x \in X \subseteq \mathbb{R}^n, u \in U \subseteq \mathbb{R}^l, \) and \( y \in Y \subseteq \mathbb{R}^m \) where \( X, U, \) and \( Y \) are polyhedral sets. The optimization is performed with respect to \( U = \{u_0^*, \ldots, u_{N-1}^*\}. \) As per the concept of a receding horizon control (RHC), only the first optimized input, i.e., \( u_0^* \) is applied to the system in (1), and the whole procedure is repeated at a subsequent time instant for a new value of the initial condition obtained using (2h) and (2i).

C. QP Problem Formulation

The open-loop CFTOC problem in (2) can be formulated as a general convex QP problem as given below. For the detailed formulation of the QP problem see [18].

\[
\min_U \frac{1}{2} U^T H U + f^T U, \quad (3a)
\]

subject to \( GU \leq w, \quad (3b) \)

where \( H \in \mathbb{R}^{lN \times lN} \) is the Hessian matrix, \( f \in \mathbb{R}^{n \times lN}, G \in \mathbb{R}^{q \times lN}, \) and \( w \in \mathbb{R}^{q} \) are the matrices/vectors with \( q \) as number of inequalities. The most general approach to solve QP problems is to use the active-set methods [19], interior-point methods [17], and gradient-based [20] methods, which have shown good convergence and stability properties. However,
we propose a penalty method which, in addition to having good convergence and stability will also perform well for the problem with a large number of constraints, unlike ASM. Moreover, the dependency of initial guess does not influence the convergence, which is a common issue with IPM.

III. PENALTY METHOD

In this section, we describe the proposed robust penalty method-based QP solver aiming to develop fast embedded MPC. Consider the following QP problem:

$$\min_{U} F = J + K^T P,$$

(4a)

$$P = (\max[z, g])^2,$$

(4b)

$$\text{where } g = GU - w,$$

(4c)

$$J = \frac{1}{2} U^T H U + f^T U,$$

(4d)

where $K \in \mathbb{R}^q$ is penalty gain vector with each element being the gain for elements in $P$. $P \in \mathbb{R}^q$ is the penalty function satisfying the following conditions: (i) $P$ is continuous, (ii) $P \geq 0$ for all $x \in E$, where $E$ is feasible set. The $z \in \mathbb{R}^q$ such that $z = [0, \ldots, 0]$. The frequently used penalty function is as shown in (4b). Hence, the objective function $F(U)$ will be an augmentation of original function $J$ and penalty term $P$.

It is evident that as $K$ is large, the corresponding $P$ has to be small when the objective function is minimized [17, chapter 17]. A subsequent increase in $K$ at each iteration would bring a corresponding point in the feasible region minimizing objective function $F$. The idea is to penalize the minimization function every time a constraint is violated. The general practice is to start $K$ with 0.1 and increase it 10 times for the next iteration. Soft handling of constraints makes the computations fast. For safety-critical applications, tolerance can be adjusted to get the desired performance. If the solution lies on the boundary for some problems, the tolerance can be defined to set the acceptable limit of the error in the optimal solution. Thus, defining this tolerance will prevent the solution from oscillating between feasible and infeasible regions and never settling. The algorithm used for the RPM to solve the QP problem [21] in a linear MPC is presented in 1.

To solve the unconstrained QP problem in (4), a Quasi-Newton BFGS algorithm similar to [21, chapter 10] is used with a few changes. Using this algorithm, we can maintain the positive definiteness of the Hessian matrix. This is done by defining $\tilde{H}_k$, which is a positive definite identity matrix. As the point moves to the optimal solution, the approximation converges to the original Hessian. The approximate Hessian is revised based on the gradient information of previous and current iteration. This method helps to avoid unfavorable results caused due to ill-conditioning of Hessian matrix when penalty $K$ increases.

Let objective function $F(U)$ have a quadratic model, as shown in (5a):

$$M = F(U_k) + (\nabla F(U_k))^T p_k + p_k^T \tilde{H}_k p_k,$$

(5a)

$$p_k = -\tilde{H}_k^{-1} \nabla F(U_k).$$

(5b)

**Algorithm 1**  Robust penalty method using BFGS unconstrained QP solver.

**Input:** $J, g, K, \delta$.  
**Output:** $U_N, F_{val}$.  
Choose arbitrary $U$ and slack tolerance $= \xi$.  
iteration = 0  
repeat  
if $(g > \xi)$ then  
$K = \{0.1, \ldots, 0.1\}$  
end if  
Update $K = K \times 10^{(\text{iteration})}$  
$[U_N, F_{val}] = \text{BFGSSolve}(J, g, K, U, \delta)\{\text{See Algorithm 2}\}$  
if $(g \leq \xi)$ then  
if (Convergence tolerance $< \delta$) then  
$U_N$ is the optimum point in feasible region  
else  
$U_N$ is in feasible region but not the optimum point  
end if  
else  
$U_N$ not in feasible region  
end if  
iteration = iteration + 1  
until $U_N$ is not optimum

The $p_k$ that would guarantee the minimization of the convex quadratic model is given in (5b). Algorithm 2 presents the BFGS method used to solve unconstrained QP problems using Algorithm 1. The Hessian ($H$) matrix derived in MPC formulation (3a) turns out to be symmetric positive definite. Due to the nature of $H$, the inverse of the $H$ matrix can be obtained through Cholesky factorization [22] and forward/backward substitution. As the final goal is to have a fast solver, the inexact line search is implemented here.

Fig. 1 shows the flowchart of the proposed robust penalty method. The traditional penalty method is followed till the solution enters the feasible region, thereafter using the proposed extension of the method till convergence is achieved.

In MPC, optimization is performed at each sample time. For the first iteration of the closed-loop simulation, we take the initial guess for input $U_N = [0, 0, 0, \ldots, 0]$ where $N$ represents the prediction horizon, and the size of $U_N$ depends on the pre-defined prediction horizon ($N$). In subsequent iterations, the solver’s initial guess will be the solution of the previous iteration. This will ensure a warm startup for the solver. If the initial point is in a feasible region, then the penalty is zero, and an unconstrained optimization problem is solved. Since the process is not under violation, the optimization problem is solved until the convergence criteria are fulfilled or any constraint is violated. If the former criteria terminates the problem, then the solution is in the feasible region. Else it lies on the boundary of the feasible region. In contrast to the above situation, if the initial guess is in
Algorithm 2 BFGS QP solver (BFGSSolve) used in RPM to solve unconstrained QP

**Input:** $J, g, K$.
**Output:** $U_N, F_{val}$.
Initialize $\tilde{H}_k$ to identity matrix $\in \mathbb{R}^{n \times n}$, convergence tolerance $\delta$, and solution tolerance $\epsilon$.

while ($\epsilon > \delta$) do
Define $\mathbf{b} = -\nabla (F(U))$
$g = \tilde{H}_k \mathbf{b}$
Obtain $t_k$ by backtracking algorithm
Set $U_{k+1} = U_k + t_k \mathbf{p}_k$
if (in infeasible region) then
check $\epsilon = \frac{|(U_k) - (U_{k+1})|}{U_k}$
if ($\epsilon < \alpha$) then
break
else
Update $\tilde{H}_k$
$k = k + 1$
Update $K$
$U_k = U_{k+1}$
end if
else
check $\epsilon = \frac{|(U_k) - (U_{k+1})|}{U_k}$
if ($\epsilon < \alpha$) then
$U_N = U_{k+1}$
else
$U_k = U_{k+1}$
value of $K$ does not change
$k = k + 1$
end if
end if
end while

the infeasible region, a penalty is added on the violated constraints, and Algorithm 1 is followed until the point moves into the feasible space where the penalty becomes zero. Solving the problem even when the penalty is zero is an extension of the penalty method. This is independent of the initial guess, and the point will converge to a solution irrespective of whether the initial point was in a feasible or infeasible region.

IV. EMBEDDED IMPLEMENTATION

This section describes the embedded implementation of linear MPC for citation aircraft model using ASM, IPM, and proposed robust penalty method as QP solvers. The hardware used for implementation is STM Nucleo-144 development board with STM32F746ZGT6 MCU. It is an ARM 32-bit Cortex M7 microcontroller running at 216 MHz. It has 1024 Kb of program memory and 320 Kb of SRAM. Fig. 2 shows the HIL co-simulation set-up. Fig. 3 shows the design flow of the embedded implementation. The top left block shows the design steps carried in the MATLAB environment. The bottom left block shows the HIL co-simulation by interfacing with the Simulink environment. The blocks at right depict the function call of the respective QP solvers into the MATLAB environment for performance comparison.

- System modeling: We consider a linear discrete-time state-space model (as in (1)) of the citation aircraft system/plant under consideration to design MPC. Here, we consider a case study of the citation aircraft system. The citation aircraft system is inherently nonlinear. The model is linearized about equilibrium points and is transformed into a discrete-time model. This linearized model is used in the formulation of a linear MPC.
- MPC problem: the MPC problem is constructed (as in (1)) for the aircraft system by considering the reference tracking formulation (as in (2)). Subsequently, the QP matrices/vectors, i.e., $H, f, G, w$ as in (3) are generated in double-precision floating-point numbers.
- QP solver: the algorithms are C++ programs, in single-precision floating-point format i.e, float and synthesized using STM32CubeIDE, with GNU ARM compiler, which is used in MATLAB by creating its MEX file.
- Software-in-the-loop (SIL) testing/verification: from this step, we get an idea about the computational burden and memory demand. After getting the desired performance, we deployed the code on the hardware.
- HIL co-simulation: we designed the simulator model in the MATLAB/Simulink. For performing HIL co-simulation, Hardware development board STM32 Nucleo F746ZG is used for the implementation of the QP solver. Processor-in-the-loop (PIL) communication interface is serial communication with the universal syn-
chronous/asynchronous receiver/transmitter (USART2). We used the Simulink embedded coder for deploying RPM solver and for ASM/IPM, we use the S function block in the Simulink model. We deployed the QP solver on the microcontroller. We built a subsystem of the closed-loop model on the microcontroller, which generates the PIL block. This replaces the subsystem block. The closed-loop results obtained are discussed in section V.

V. Results

This section presents the performance comparison of proposed method against ASM and IPM for benchmark QP problem and citation aircraft model.

A. Benchmark QP problem: qptest

QP solver requires a standard QP problem for the performance evaluation. We used the qptest problem, which is a standard QP problem adopted from the 1999 Maros and Meszaros repository [23].

\[
\begin{align*}
\min f(x,y) &= 4 + 1.5x - 2y + \\
&\quad \frac{1}{2}(8x^2 + 2xy + 2yx + 10y^2), \\
\text{s.t.} \\
2x + y &\geq 2, \\
-x + 2y &\leq 6, \\
0 &\leq x \leq 20, \ y \geq 0,
\end{align*}
\]

(6a)

The performance comparison of the solvers for the qptest is shown in Table I. We see that the execution time for IPM and RPM is comparable. However, the ASM takes a longer time to reach the same optimal value.

| Solver                  | Execution Time [s] | Optimal Value       |
|-------------------------|--------------------|---------------------|
| Active Set Method       | 0.05               | 4.3718750           |
| Interior Point Method   | 0.004              | 4.3718750           |
| Robust Penalty Method   | 0.004              | 4.3718750           |

B. Citation Aircraft Control using MPC

To show the efficiency of the developed RPM QP solver, we present a case study of citation aircraft control problem [18, Chapter 3]. The longitudinal dynamics of a citation aircraft model are considered here using its linearized model with constant speed approximation. An aircraft is considered to be moving at an altitude of 5000 m and a constant speed of 128.2 m/s. Model has 4 states as \( x = [\text{angle of attack} (\degree), \text{pitch angle} (\degree), \text{altitude} (\text{m}), \text{altitude rate} (\text{m/s})] \). The control input is the elevator angle (\degree). The outputs of interest are: \( y = [\text{pitch angle} (\degree), \text{altitude} (\text{m}), \text{altitude rate} (\text{m/s})] \).

We consider a discrete-time linear-time invariant state-space model as presented and parameter values in [18, Chapter 3].

\[
\begin{align*}
x(t+T_s) &= Ax(t) + Bu(t), \\
y(t) &= Cx(t) + Du(t),
\end{align*}
\]

(7a) (7b)
where,

\[
A = \begin{bmatrix}
0.240 & 0 & 0.1787 & 0 \\
-0.372 & 1.000 & 0.270 & 0 \\
-0.990 & 0 & 0.138 & 0 \\
-48.935 & 64.100 & 2.399 & 1.000
\end{bmatrix}, \quad B = \begin{bmatrix}
-1.234 \\
-1.438 \\
-4.482 \\
-1.799
\end{bmatrix},
\]

\[
C = \begin{bmatrix}
0 & 1.000 & 0 & 0 \\
0 & 0 & 0 & 1.000 \\
-128.200 & 128.200 & 0 & 0
\end{bmatrix}, \quad D = \begin{bmatrix}
0 \\
0 \\
0
\end{bmatrix}.
\]

The elevator angle has the constraints of \(\pm 15^\circ\) and the elevator slew rate is limited to \(\pm 30^\circ\) s. These limitations are introduced due to the inherent design aspect of the aircraft. In addition to this, the passenger’s comfort level is maintained by limiting the pitch angle between \(\pm 20^\circ\).

To compare the performance of different methods, we plot the relative cumulative suboptimality (RCSO), relative to ASM. The RCSO is obtained as:

\[
\text{RCSO}_{T_{\text{sim}}} = \frac{\text{DR}_{T_{\text{sim}}} - \text{DR}_{\text{ASM}, T_{\text{sim}}}}{\text{DR}_{\text{ASM}, T_{\text{sim}}}},
\]

where \(T_{\text{sim}}\) is the number of time steps in the HIL-simulation. Table II shows the RCSO comparison. A prediction horizon of 5 and 10 is set for this application. Moreover, the RPM is also tested for prediction horizon of 20 and 30, which resulted in a minimal increase of computational time and with no significant improvement of response. The result is generated for the convergence tolerance of 1E-6 for all solvers. However, it can be changed as per the application and respective requirements.

### TABLE II
RELATIVE SUBOPTIMALITY OF THE DIFFERENT EXAMPLES. NOTE: SMALL VALUE IS BETTER

| QP Solver   | Prediction Horizon | RCSO    |
|-------------|--------------------|---------|
| Active Set Method | 5                  | 0E+00   |
|              | 10                 | 0E+00   |
| Interior Point Method | 5                  | 2.31e-02|
|              | 10                 | 5.12e-03|
| Robust Penalty Method  | 5                  | 4.73e-03|
|              | 10                 | 4.12e-05|

We observed that the RCSO of RPM is less compared to IPM. Table III demonstrates the performance of solver with a different predictive horizon. We see that RPM is highly competent and outperforms IPM and ASM. Since the execution of RPM will depend on the slack tolerance, smaller the slack value, the higher will be the error and vice versa.

The variation of computational time for different slack tolerance is shown in Table IV. We see that the increase in execution time is not very significant. Moreover, slack tolerance can be a user-defined parameter, depending on the application. In Table V, we show the comparison of convergence tolerance for the three solvers. RPM performs better than IPM and ASM for the lower convergence tolerance, but the execution time of RPM increases more significantly than the other two. Since the performance of MPC doesn't depend on the extent to which QP is solved [6], this does not have major contribution towards the final performance.

### TABLE III
AVERAGE EXECUTION TIME (IN SECONDS) ON STM NUCLEO-144 BOARD FOR THREE QP SOLVERS WITH DIFFERENT PREDICTION HORIZONS.

| Prediction Horizon | Active Set Method | Interior Point Method | Robust Penalty Method |
|--------------------|------------------|-----------------------|-----------------------|
| 5                  | 0.0071           | 0.0051                | 0.0046                |
| 10                 | 0.0075           | 0.0051                | 0.0046                |
| 20                 | 0.0075           | 0.0051                | 0.0054                |
| 30                 | 0.0085           | 0.0056                | 0.0054                |
| 50                 | 0.0116           | 0.0057                | 0.0055                |

### TABLE IV
AVERAGE EXECUTION TIME (IN SECONDS) ON STM NUCLEO-144 BOARD FOR ROBUST PENALTY METHOD SOLVER WITH DIFFERENT SLACK TOLERANCES WHEN \(N = 10\).

| Slack Tolerance | Execution Time [s] |
|-----------------|--------------------|
| 1E-4            | 0.0019             |
| 1E-6            | 0.0021             |
| 1E-9            | 0.0024             |
| 1E-12           | 0.0028             |
| 1E-15           | 0.0028             |
| 1E-18           | 0.0029             |

Fig. 4 displays the results for reference tracking using the three solvers. It is observed that all the three methods track the reference accurately while satisfying the constraints.

### VI. Conclusion

This paper has presented a robust penalty method approach for solving the linear MPC. Our extension makes the penalty method robust to any initial guess, unlike the traditional penalty method. The proposed QP solver is implemented on a low-cost STM32 microcontroller and its performance is compared with that of state-of-the-art QP solvers such as the ASM and IPM. The HIL co-simulation results of the three QP solvers are presented for control of aircraft problem. A detailed analysis of computational complexity in terms of hardware execution time and steady-state error is presented. The results show that the proposed RPM outperforms the
ASM and IPM solvers in execution time. We suggest some directions that can be explored for further research:

- The proposed QP solver needs to be tested for complex and large scale QP problem for further analysis.
- This solver can be extended to solve nonlinear MPC due to its capability to handle nonlinear constraints.

ACKNOWLEDGMENT

We gratefully acknowledge the support from the R&D center of Embedded lab in the Instrumentation and Control department at COEP. Vaishali Patne acknowledges the contribution of the Department of Science and Technology, Govt. of India, under Women Scientist Scheme-A (WOS-A/ET-120/2018). Deepak Ingole would like to thank the financial support of the Moonshot-FLEX project of the Flemish Government.

REFERENCES

[1] C. E. Garcia, D. M. Pret, and M. Morari, “Model Predictive Control: Theory and Practice—a Survey,” Automatica, vol. 25, no. 3, pp. 335–348, 1989.
[2] D. Q. Mayne, “Model predictive control: Recent developments and future promise,” Automatica, vol. 50, no. 12, pp. 2967–2986, 2014.
[3] T. A. Badgwell and S. J. Qin, Model-Predictive Control in Practice. Pergamon Press, 2013.
[4] F. Borrelli, A. Bemporad, and M. Morari, Predictive Control for Linear and Hybrid Systems. Cambridge University Press, 2017.
[5] J. Mattingley and S. Boyd, “CVXGEN: A Code Generator for Embedded Convex Optimization,” Optimization and Engineering, vol. 13, no. 1, pp. 1–27, 2012.
[6] Y. Wang and S. Boyd, “Fast Model Predictive Control Using Online Optimization,” IEEE Transactions on control systems technology, vol. 18, no. 2, pp. 267–278, 2009.
[7] A. Domahidi, E. Chu, and S. Boyd, “ECOS: An SOCP Solver for Embedded Systems,” in 2013 European Control Conference (ECC). IEEE, 2013, pp. 3071–3076.
[8] E. Chu, N. Parikh, A. Domahidi, and S. Boyd, “Code Generation for Embedded Second-Order Cone Programming,” in 2013 European Control Conference (ECC). IEEE, 2013, pp. 1547–1552.
[9] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, “qpoases: A parametric active-set algorithm for quadratic programming,” Mathematical Programming Computation, vol. 6, no. 4, pp. 327–363, 2014.
[10] R. Bartlett, A. Wachter, and L. Biegler, “Active Set vs. Interior Point Strategies for Model Predictive Control,” in Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No. 00CH36334), vol. 6. IEEE, 2000, pp. 4229–4233.
[11] M. S. Lau, S.-P. Yue, K. V. Ling, and J. M. Maciejowski, “A Comparison of Interior Point and Active Set Methods for Fpga Implementation of Model Predictive Control,” in 2009 European Control Conference (ECC). IEEE, 2009, pp. 156–161.
[12] C. Jugade, D. Ingole, D. Sonawane, M. Kvasnica, and J. Gustafson, “A framework for embedded model predictive control using posits,” in 59th IEEE Conference on Decision and Control, 2020, pp. 2509–2514.
[13] D. Armström and D. Axelhilm, “A unifying complexity certification framework for active-set methods for convex quadratic programming,” IEEE Transactions on Automatic Control, 2021.
[14] Y. Ding, Z. Xu, J. Zhao, K. Wang, and Z. Shao, “Embedded mpc controller based on interior-point method with convergence depth control,” Asian Journal of Control, vol. 18, pp. 2064–2077, 2016.
[15] P. Sutor and T. Goldstein, “The Alternating Direction Method of Multipliers An ADMM Software Library,” 2016. [Online]. Available: http://www.math.umd.edu
[16] M. Kögel and R. Findeisen, “A fast gradient method for embedded linear predictive control,” IFAC Proceedings Volumes (IFAC-PapersOnline), vol. 44, no. 1 PART 1, pp. 1362–1367, 2011.
[17] J. Nocedal and S. Wright, Numerical Optimization, 2nd ed. Springer-Verlag, USA, 2006.
[18] J. M. Maciejowski, Predictive Control: With Constraints. Pearson education, 2002.
[19] R. Fletcher, Practical Methods of Optimization, 2nd ed. John Wiley & Sons, 2013.
[20] J. Nocedal, Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms. Springer Science & Business Media, 2005, vol. 97.
[21] D. G. Luenberger, Y. Ye et al., Linear and Nonlinear Programming. Springer, 1984, vol. 2.
[22] N. J. Higham, “Cholesky Factorization,” Wiley Interdisciplinary Reviews: Computational Statistics, vol. 1, no. 2, pp. 251–254, 2009.
[23] I. Maros and C. Mészáros, “A Repository of Convex Quadratic Programming Problems,” Optimization Methods and Software, vol. 11, no. 1-4, pp. 671–681, 1999.