WHITE PAPER

# Three Ways to Speed Up Model Predictive Controllers

MathWorks®

## Introduction

Model predictive control (MPC) is an advanced controls technique that has been used for process control since the 1980s. With the increasing computing power of microprocessors, the use of MPC has spread to real-time embedded applications, often used in the automotive and aerospace industries. MPC can handle multi-input multi-output (MIMO) systems with coupled input-output channels, which simplifies the architecture of the control loop. Designing a controller for such a system with PID controllers, for example, would be challenging since the control loops and associated responses would be intertwined. Additionally, MPC can handle constraints on the inputs, outputs, and states, which is important since real-world systems have physical limits that need to be respected. Other MIMO techniques such as linear quadratic regulators (LQRs) cannot handle constraints explicitly. Another advantage of MPC is that controls calculated with MPC take into account what will (likely) happen several steps into the future to improve performance. This is possible because the MPC uses an internal prediction model of the system that is controlled. Finally, MPC can optimize multiple objectives, including economics, controls, and safety.

If MPC has all these advantages, do we still need traditional methods like PIDs? Unlike PIDs, MPC is more challenging to apply in high-bandwidth, feedback control applications— that is, applications where the response time of the control loop must be short. This is because, for feedback control, MPC solves an optimization problem online, and that requires significant computing power and memory. Specifically, MPC works as follows (Figure 1):

1. At each time step, the solver solves a constrained optimization problem. The objective is to minimize a cost function that encodes the desired control objective (e.g., trajectory tracking), subject to state/output constraints, manipulated variable (MV) constraints, and the internal plant dynamics. Depending on the nature of the optimization problem (panel 1), MPC can be divided into linear and nonlinear. In linear MPC, the cost function is quadratic with respect to the outputs/states and control, while the internal prediction model and constraints are linear with respect to the same. In nonlinear MPC, the cost, internal prediction model, and constraints can be nonlinear.
2. The optimization outputs a sequence of MV moves for a user-specified time horizon; this is often referred to as open-loop control (panel 2).
3. The controller applies only the first MV move of the solution to the system and discards the rest (panel). MPC is considered feedback control because current prediction model states are required to calculate optimal control actions. The states can be measured or estimated by appropriate state estimators such as Kalman filters at run time.
4. The time horizon is then shifted by one step, current state information is acquired (panel 4), and this process is repeated indefinitely.

Because the time horizon is constantly moving forward after each time step, MPC is also referred to as receding horizon control.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

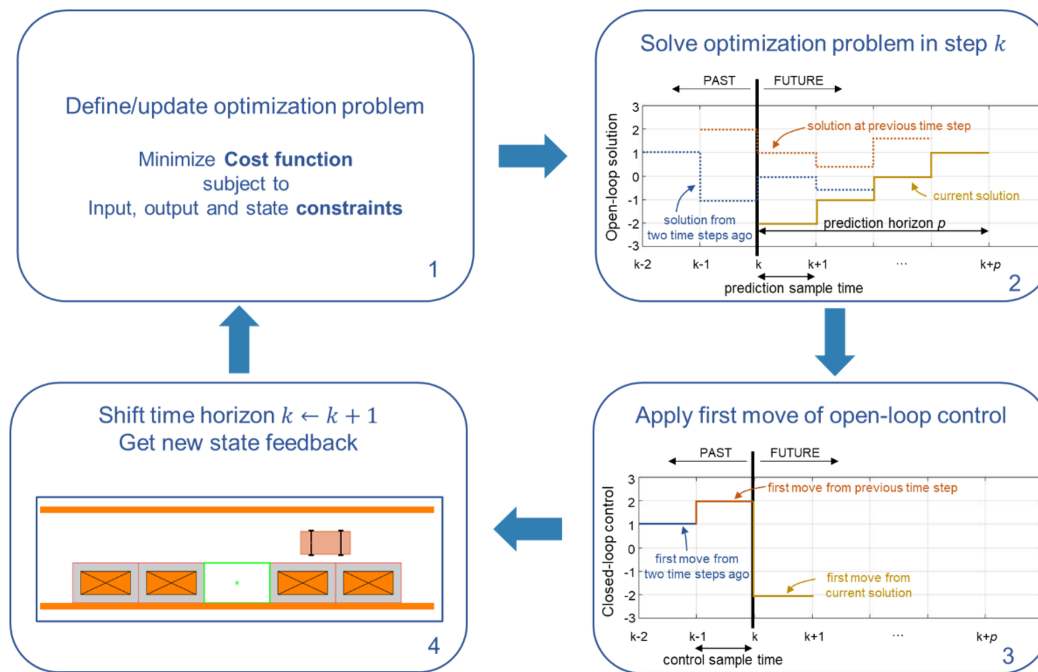


Figure 1. The MPC process.

MPC (linear or nonlinear) requires many design choices that affect the complexity of the underlying optimization problem, and thus, how fast the controller can run. Common challenges include, for example, choosing the appropriate MPC type, structure, and model complexity based on plant dynamics and control objectives, or selecting the appropriate MPC settings based on hardware/software limitations. The goal of this white paper is to summarize the design choices that affect execution speed of linear and nonlinear MPC controllers and provide tips and tricks that will let you run MPC controllers faster with Model Predictive Control Toolbox™. Note that, unless mentioned otherwise, the following sections will focus on using MPC for feedback (closed-loop) control, as these applications can greatly benefit from execution speedup. Model Predictive Control Toolbox can also be used in open-loop applications (e.g., as a trajectory optimization technique). Many of the topics covered are still applicable in this scenario, but, because trajectory optimization is typically performed offline or at a much slower rate, speed is often not as a big concern as in feedback control problems.

The sections that follow are organized as shown in Figure 2 and cover the following three ways to speed up MPC controllers:

- Pick appropriate parameter values for the MPC problem.
- Choose the appropriate solver and solver options.
- Change your approach.

For completeness, some guidelines on minimizing memory requirements of MPC controllers are also provided.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

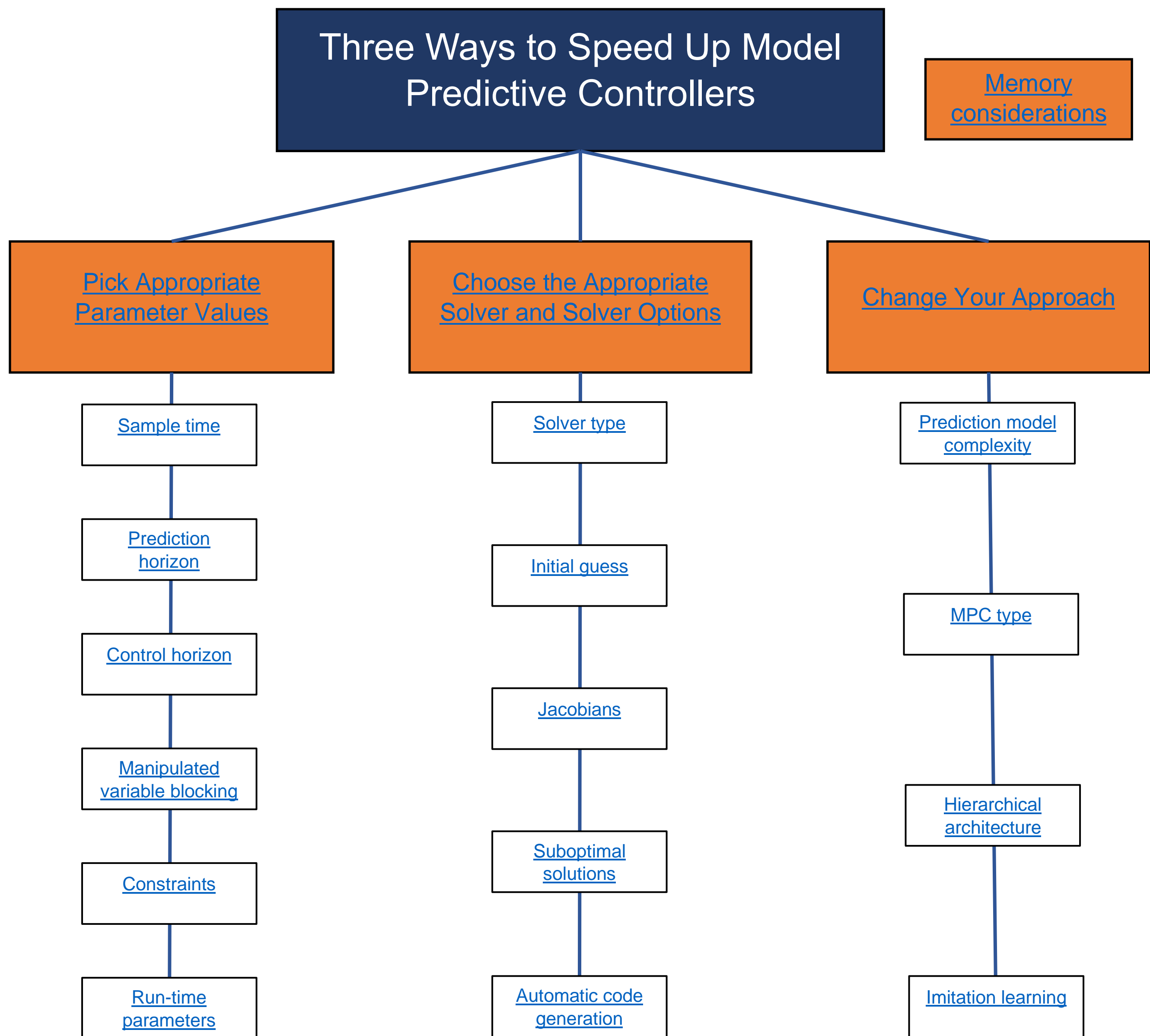


Figure 2. Methods for speeding up model predictive controllers.
Select a topic to jump to that section of the white paper.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

## Pick Appropriate Parameter Values for the MPC Problem

Regardless of whether you are using linear or nonlinear MPC, the design parameters that need tuning remain effectively the same. This section covers the key parameters of MPC problems, with explanations on how they affect the computational complexity of the MPC optimization problem.

### Sample Time

The sample time is a key concept in model predictive control and can be separated into two parts: the prediction sample time and the control sample time. When designing an MPC problem, the prediction sample time and control sample time are often set to be equal or even treated as one parameter, but it is important to distinguish between the two and how they affect performance.

#### Prediction *Ts*

The prediction *Ts* is the sample time of the internal prediction model. It defines how long each prediction step lasts, and the value of all MVs remains constant during each prediction step (panel 2 in Figure 1). Intuitively, the prediction sample time determines the upper bound of achievable control bandwidth. The product of the prediction horizon (discussed in the next section) and the prediction sample time is the prediction time of the controller—that is,  how far into the future the controller is planning for. Since the prediction time is often provided as part of the control objective, the value of the prediction sample time is often selected together with the prediction horizon.

The upper bound of prediction *Ts* is determined by plant dynamics and control response time. For example, if prediction *Ts* is slow, you may not have enough control bandwidth to stabilize an open-loop unstable plant. On the other hand, faster prediction *Ts* will require a longer prediction horizon to keep the prediction time constant. However, as explained in the Prediction Horizon section, longer prediction horizons lead to more decision variables and more constraints, which make the optimization problem larger (higher memory footprint) and more complex to solve. A rule of thumb is to try and fit 10–20 MV moves within the rise time of the open-loop step response.

#### Control *Ts*

The control *Ts* determines the sample time of the MPC controller. It defines how often the MPC optimization problem is solved at run time (panel 3 in Figure 1). The control sample time is typically equal to the prediction sample time, but it can also be set to be faster (but not slower). Faster control *Ts* generally improves performance (i.e., bandwidth) and robustness (i.e., gain and phase margins) to some extent. Also, as control *Ts* gets smaller, rejection of unknown disturbances, including discrepancies between internal MPC model and actual plant, usually improves and then plateaus. Qualitatively, this makes sense as the controller is able to respond faster to changes in the environment. The control sample time value at which performance plateaus typically depends on the plant dynamic characteristics. For example, processes with slow dynamics will not benefit much from small control sample times, unlike real-time control applications such as motor control.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

> "We were searching for a prototyping solution that was fast for development and robust for production. We decided to go with Simulink for controller development and code generation, while using MATLAB to automate development tasks."
>
> *— Alan Mond, Voyage*

However, as the control sample time becomes small, the computational effort increases dramatically as the MPC optimization problem is solved more frequently. Thus, the optimal choice is a balance of performance and computational effort. To determine the control sample time, first test a less aggressive controller (large control *Ts*, possibly equal to prediction *Ts*) that produces acceptable performance. Next, decrease control *Ts* and monitor the execution time of the controller. If you are working on a real-time application, you can further decrease control *Ts* as long as the optimization safely completes within each sampling period under normal plant operating conditions. As mentioned in the following sections in more detail, there is no way to predict how many solver iterations are required to find an optimal solution, so using a suboptimal solution of the optimization problem makes it easier to tune control *Ts*. Keep in mind that the smallest control *Ts* that an MPC controller can achieve is system-dependent, and will differ across different (real-time) hardware and simulation platforms.

**Tip**

To minimize the number of computations, choose prediction Ts and control Ts that are fast enough to satisfy control requirements, but not any faster.

To run MPC at a fast rate, consider choosing control Ts < prediction Ts to retain a reasonably sized optimization problem that can be solved faster than the control sample time.

*Learn More*

How to choose the sample time

Sampling rate in real-time environment

## Prediction Horizon

The *prediction horizon*, *p*, is the number of future control intervals the MPC controller must "plan" for (using the internal plant model for prediction) when optimizing its MVs. The duration of each control interval is determined by the prediction sample time. Similar to the sample time discussion above, the choice of prediction horizon depends on the characteristics of the plant dynamics. The main guideline on how to select *p* is in fact to satisfy the prediction time (*p* * prediction *Ts*) requirements for the system of interest. Typically, systems with slower dynamics require longer prediction times such that the MPC controller can sufficiently predict how the manipulated variables may affect the cost/outputs of interest. Thus, the values of prediction horizon *p* and prediction *Ts* are, in a sense, intertwined.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

However, larger *p* values lead to more decision variables and constraints (optimization constraints should be satisfied for each control interval), which lead to a larger optimization problem; dimensions of many matrices in the MPC optimization problem are proportional to *p* (see table for details) with longer execution times and higher memory footprint. Using linear MPC as an example, when only MVs are being optimized (as opposed to also optimizing with respect to states and outputs, for example), the total number of decision variables is *p * # of MVs*, which leads to a *(p * # of MVs)* x *(p * # of MVs)* Hessian matrix. Similarly, the number of MV constraints is *2p * # of MVs.* Typically, the prediction horizon should not be adjusted to tune the controller. The value of *p * Ts* should be such that the controller is internally stable and anticipates constraint violations and environment changes early enough to allow corrective action. This may be accomplished by making sure that *p * Ts* covers the rise time or transient time of the open-loop step response. If the plant is open-loop unstable, *p* should not be greater than the maximum number of control intervals required for the open-loop step response of the plant to become infinite.

**Prediction horizon vs. control Ts vs. prediction Ts through an example**

Assume that the desired prediction time (*p* * prediction *Ts*) is 1 second and that we know (from experiments) the average execution time for solving a single MPC optimization problem on specific hardware.

**Case 1:** For *p*=10, assume we know that execution time is 1 ms. For 1 second prediction time, this corresponds to a prediction *Ts* of 100 ms. Since the prediction *Ts* is larger than the total execution time, the controller can run on the hardware without overrun. To improve performance, we could choose the control *Ts* to be faster than the prediction *Ts* as long as it is < 1 kHz.

**Case 2:** For *p*=25, assume we know that execution time is 40 ms. For 1 second prediction time, this corresponds to a prediction *Ts* of 40 ms. Since the prediction *Ts* is equal to the total execution time, this is the smallest prediction *Ts* (or largest prediction horizon *p*) we can use to run the controller on the hardware without overrun. The control *Ts* cannot go any lower than 40 ms either.

If both cases above lead to good performance, which one should you choose? This varies by case, but since larger *p* values lead to larger memory footprint, the first case may be preferable for hardware with limited memory capacity.

**Tip**

As the prediction horizon increases, the controller can better anticipate and plan for future events, but the solution time and memory requirements for the controller increase.

*Learn More*

How to Design Model Predictive Controllers (3:00)

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## Control Horizon

The control horizon, *m*, is the number of MV moves to be optimized at control interval *k* and takes values between 1 and the prediction horizon *p* (Model Predictive Control Toolbox uses a default control horizon value of $m = 2$). MV moves determine the values of the MPC solution (open-loop control) at each step of the control horizon, for each manipulated variable specified in the problem (Figure 3). As a result, the number of variables that need to be optimized by the solver grow with the number of control inputs and the control horizon value. For example, when only MVs are decision variables (see [Optimization Solvers]() section), the number of optimization variables is *m * # of MVs*. If states are also included as decision variables (e.g., in sparse problem formulations), the number of optimization variables is *m * # of MVs + p * # of states.* In MPC, regardless of your choice for *m*, when the controller operates, only the first optimized MV move of the MPC solution is used (at the beginning of the horizon) and any others are discarded. Practically, this means that the longer the control horizon, the more time it takes to solve the optimization problem and the more information is discarded when the controller moves to the next interval.

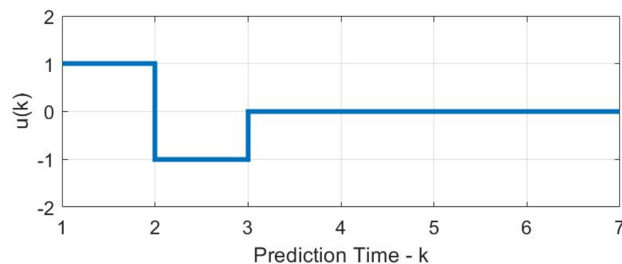


Figure 3. Example of MPC solution (open-loop control) at interval k with m=2 and p=7.

**Tip**

*m<<p* means fewer variables to optimize at each control interval, which promotes faster computations.

*Learn More*

[How to Choose the Prediction Horizon]()

[How to Choose the Control Horizon]()

## Manipulated Variable Blocking

*Manipulated variable blocking* is an alternative to the simpler control horizon concept, and it has many of the same benefits. Instead of using a scalar value to specify the control horizon, manipulated variable blocking divides the prediction horizon into a series of blocking intervals by specifying the control horizon as a vector of block sizes. The sum of the block sizes must match the prediction horizon *p*. This allows you to specify not only the desired number of MV moves, but also the duration of each move (as a multiple of the prediction sample time). Figure 4 shows

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

an example of manipulated variable blocking for a control horizon of *m*=[2 3 2] and prediction horizon of *p*=7.

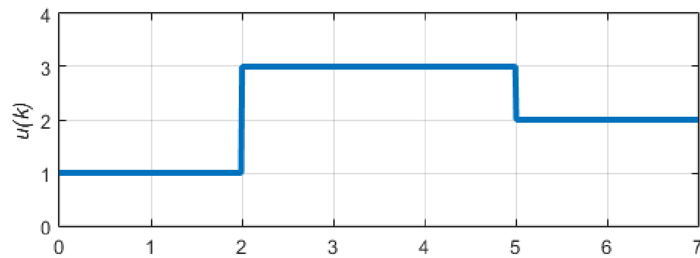


Figure 4. MPC solution at interval k with m=[2 3 2] and p=7.

**Tip**

Similar to the simpler control horizon concept, manipulated variable blocking allows you to control the number of *decision variables at each control interval, which promotes faster computations.*

*Learn More*

[Manipulated Variable Blocking]

## Constraints

As mentioned previously, MPC solves a constrained optimization problem at each time step. Typically, the larger the number of constraints, the longer it takes to solve the problem as complexity grows. Constraints limit the space of admissible controls, increasing the risk of running into infeasible problems—problems that that are impossible to satisfy. Keeping the number of constraints to the absolutely necessary makes the optimization problem easier to solve.

Model Predictive Control Toolbox lets you define both hard and soft constraints to help you set up a well-defined optimization problem. Hard constraints must be satisfied by the solution, while soft constraints can be violated when necessary. Hard constraints, unlike soft constraints, typically limit the space of admissible solutions, so the optimization process takes longer to converge. If the only constraints in your application are bounds on MVs, the MV bounds can be hard constraints, as they are by default. MV bounds alone cannot cause infeasibility. The same is true when the only constraints are on MV increments. However, a hard MV bound with a hard MV increment constraint can lead to an over-constrained problem and infeasibility. The same is true if the plant is subject to disturbances and there are either hard output constraints or hard mixed input-output constraints. As rule of thumb, you should opt for soft constraints when possible.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Tip**

Keep the number of constraints small and opt for soft as opposed to hard constraints when possible.

*Learn More*

Constraints for Linear MPC

Constraints for Nonlinear MPC

Terminal Constraints

Time-Varying Constraints

### Run-Time Parameters

Model Predictive Control Toolbox allows certain design parameters of the MPC problem to vary at run time. For example, to tune the cost function weights, you can either make modifications offline or online, as the controller operates, by specifying weights that vary over the prediction horizon. Similarly, you can tune the control and prediction horizon at run time during prototyping, or you may want to adjust these parameters online, for example in cases where the plant dynamics change during operation. Constraints may also vary over the prediction horizon, for example to compensate for changing operating conditions. While useful for certain situations, adjusting parameters at run time also increases complexity, as more calculations are necessary to set up the MPC optimization problem. Thus, the optimal choice here is also a balance of performance and computational effort.

**Tip**

To limit the number of calculations required at each time step, keep the number run-time parameter changes small.

*Learn More*

Tune Weights at Run Time

Update Constraints at Run Time

Adjust Horizons at Run Time

Specify Constraints for Nonlinear MPC

## Choose the Appropriate Solver and Solver Options

For techniques like MPC that rely heavily on numerical optimization, performance is heavily dependent on choosing the right solver and solver configuration. Even for the same set of MPC parameters discussed previously, a better solver choice could lead to much faster solutions than a poor one. In this section, you will learn about the different solver options available in Model Predictive Control Toolbox, as well as some tips and tricks to reach solutions faster, regardless of the solver choice.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## Optimization Solvers

In MPC applications, a high-quality solver is required to carry out real-time optimization. Common performance requirements involve execution time, memory footprint, accuracy, robustness, etc. Selecting the appropriate solver could lead to significant improvement in execution time, even without making changes to other parameters of the MPC problem.

Some solvers are specifically designed for real-time applications with fast control sample time and limited memory capacity. These are referred to as “embedded” solvers and can be roughly categorized in two ways:

- Based on algorithm used (e.g., active-set, interior-point, augmented-Lagrangian)
- Based on matrix sparsity (e.g., dense or sparse)

Model Predictive Control Toolbox provides active-set and interior-point methods with dense and/or sparse formulations through built-in solvers and integration with third-party tools (Table 1). While the solver algorithm and underlying representations will likely not matter for many cases, these choices could be the decisive factor for real-time performance in fast MPC applications:

**Active-set algorithm.** Active-set methods can provide fast and robust performance for small- and medium-scale optimization problems. Any active-set solver slows down significantly when the number of constraints grows (imagine an MPC problem with a long prediction horizon over which the constraints are enforced). Therefore, the worst-case execution time can easily lead to task overrun on hardware.

**Interior-point algorithm.** Interior-point methods can provide superior performance for large-scale optimization problems, such as MPC applications that enforce constraints over large prediction and control horizons. Also, the number of iterations required by interior-point methods to converge is more or less constant (i.e. independent from the number of constraints). Thus, it is easier to estimate worst-case execution time on embedded system.

**Dense problems.** From an MPC perspective, if the only decision variables are the manipulated variables, we have a dense optimization problem. The dense formulation has fewer decision variables than the sparse one, which makes it efficient, especially when used with manipulated variable blocking and small control horizons (e.g., with fewer decision variables). However, if the internal plant is open-loop unstable, the matrices in a dense problem might become near-singular as the prediction horizon grows.

**Sparse problems.** From an MPC perspective, if the manipulated variables, states, and outputs are decision variables, the optimization problem is sparse. Sparse problems have more decision variables than dense problems, but produce near-diagonal matrices with a large number of zero elements. The sparse structure of these matrices can be exploited to reduce computation times and memory footprint of the solver. Unlike dense formulations, sparse problems produce matrices with much better condition numbers, especially when the internal plant model is open-loop unstable.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

## Table 1. Solver options in Model Predictive Control Toolbox.

| | Built-In Solvers | Third-Party Solver Options |
|---|---|---|
| **Linear** | • Active-set solver – KWIK (dense)<br>• Interior-point solver (dense) | • Custom solver<br>  ○ Custom implementation<br>  ○ Third-party solver<br>• FORCES PRO Plugin (with FORCES PRO license)<br>  ○ Interior-point solver (sparse) |
| **Nonlinear** | • `fmincon` function with SQP algorithm from Optimization Toolbox (sparse, similar to active-set) | • Custom solver<br>  ○ Custom implementation<br>  ○ Third-party solver<br>• FORCES PRO Plugin (with FORCES PRO license)<br>  ○ Interior-point solver (dense, sparse)<br>  ○ SQP solver (dense, sparse) |

Table 1 summarizes the solver options available in Model Predictive Control Toolbox. For linear MPC problems, the toolbox supports two built-in, "dense" QP solvers: an active-set solver that uses the KWIK algorithm and an interior-point solver that uses a primal-dual algorithm with a Mehrotra predictor-corrector. To solve the optimization problem in nonlinear MPC, Model Predictive Control Toolbox uses Optimization Toolbox™, and specifically, the `fmincon` function with the SQP algorithm. You can also specify a custom solver for your (linear or nonlinear) MPC controller. This solver is called in place of the built-in solvers at each control interval. Lastly, you can now use FORCES PRO, a real-time embedded optimization software tool developed by Embotech AG, to simulate and generate code MPC controllers designed using Model Predictive Control Toolbox. For information on using the FORCES PRO solvers together with Model Predictive Control Toolbox, see the FORCES PRO documentation for linear and nonlinear MPC.

In summary, when selecting and configuring the solver for your application, consider the following:

- If the total number of manipulated variables and rates, outputs, and constraints is large (i.e., more than a few hundred because of a large prediction horizon and large number of free moves), consider using an interior-point solver.
- If the internal plant is open-loop unstable, consider using a sparse problem formulation.
- Otherwise, an active-set/dense solver would still be a viable option.

To determine which solver is best for your application, you can also consider simulating your controller across multiple simulation scenarios using different solvers.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

> “With our traditional approach it would have taken about a year to develop a controller as complex as the MPC; with Model-Based Design it took us about six months to develop a prototype.”
>
> “The generated code for the QP solver was extremely efficient, so there was no need for us to explore other solvers … With Embedded Coder, once we had confirmed the functionality of the controller it took almost no time to implement it on the embedded processor.”
>
> — *Taku Takahama, Hitachi Automotive Systems*

**Tip**

The size and configuration of the MPC problem affects the relative performance of the available solvers. To determine which solver is best for your application, consider simulating your controller across multiple simulation scenarios using different solvers.

*Learn More*

QP Built-In and Custom Solvers

Constrained Nonlinear Optimization Algorithms in Optimization Toolbox

Optimizing Tuberculosis Treatment Using Nonlinear MPC with a Custom Solver

FORCES PRO Plugin for Model Predictive Control Toolbox

## Initial Guess

In numerical optimization, the initial values used for the decision variables play a big role in deciding the course the optimizer will follow. For example, while a properly configured standard linear MPC optimization problem has a unique solution, nonlinear MPC optimization problems often allow multiple solutions (local minima). In such cases, it is important to provide a good starting point near the (global) optimum when possible.

If you are using MPC for feedback control (closed-loop control), it is best practice to warm-start your (nonlinear) solver. To do so, use the predicted state and manipulated variable trajectories from the previous control interval as the initial guesses for the current control interval. The rationale behind this approach is that the solution should not vary significantly across sequential control intervals. This will provide a good initial search direction and speed up the optimization as the solver will avoid wasting computational resources to reinvent the wheel. For the first control interval, where there is no previous solution to use as initial guess, selecting simple feasible trajectories, such as straight lines, will likely be more efficient than setting the initial guess randomly, especially for active-set solvers.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Tip**

For feedback control, it is best practice to warm-start your solver with a feasible guess, especially in nonlinear MPC.

*Learn More*

Initial Guess in Nonlinear MPC

Initial Guess in Linear MPC

## Analytical Jacobians (Nonlinear MPC)

Unlike linear MPC, in nonlinear MPC the plant dynamics and constraints can be nonlinear. Additionally, the cost function can be a nonquadratic (linear or nonlinear) function of the decision variables. As mentioned previously, to solve the optimization problem in nonlinear MPC, Model Predictive Control Toolbox uses Optimization Toolbox, and specifically, the `fmincon` function with the SQP algorithm as the built-in solver. At each MPC time step, the SQP method solves an optimization subproblem that optimizes a quadratic model of the objective subject to a linearization of the constraints (which include the plant dynamics).

If you select the built-in SQP solver and choose not to specify the Jacobian for the plant, (custom) constraints, and (custom) cost function analytically, the controller computes the Jacobians using numerical perturbations. Given that Jacobians are calculated multiple times at each time step, this process is time consuming. To improve computational efficiency, it is best practice to specify Jacobians analytically. You can use Symbolic Math Toolbox™ to automatically calculate Jacobians from mathematical expressions with symbolic variables, thus avoiding tedious manual calculations. The quadrotor example linked below shows how to do this. If calculating the exact Jacobian is not feasible, specifying a good-enough approximation may also be sufficient. See how approximate Jacobians can be used for parallel parking with nonlinear MPC. The Jacobians are used to specify the search direction of the optimization, so an approximate Jacobian may still guide the optimization in the right direction.

Another option is to use automatic differentiation to calculate the required Jacobians. Automatic differentiation (autodiff) refers to a set of techniques to numerically compute derivatives by following an algorithmic approach. While not as efficient as providing Jacobians analytically, autodiff is more efficient than numerical perturbations and can improve execution time. For example, you can use Model Predictive Control Toolbox with third-party autodiff tools, or you can use the FORCES PRO Plugin (Table 1), which makes use of automatic differentiation.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Tip**

Analytical Jacobians and automatic differentiation speed up the solution of the optimization problem in nonlinear MPC.

*Learn More*

Cost Function Jacobian

Custom Constraint Jacobians

Prediction Model Jacobian

Control of Quadrotor Using Nonlinear Model Predictive Control

## Suboptimal MPC Solution

For a given MPC application with constraints (linear or nonlinear), there is no way to predict how many solver iterations are required to find an optimal solution. Also, in real-time applications, the number of iterations can change dramatically from one control interval to the next, depending on the solver algorithm used (e.g., for active-set methods). In such cases, the worst-case execution time can exceed the limit that is allowed on the hardware platform and determined by control sample time.

You can guarantee the worst-case execution time for your MPC controller by applying a suboptimal solution after the number of optimization iterations exceeds a specified maximum value. To set the worst-case execution time, first determine the time needed for a single optimization iteration by experimenting with your controller under nominal conditions. Then, set an upper bound on the number of iterations per control interval. For example, if it takes around 1 ms to compute each iteration on the hardware and the control sample time is 10 ms, set the maximum number of iterations to be no greater than 10. As mentioned previously, for interior-point solvers, the number of iterations used to converge approximately constant, which makes it easier to estimate worst-case execution time.

While the solution reached after the final iteration is not optimal, when applied, it will satisfy all specified constraints.

**Tip**

You can guarantee the worst-case execution time for your MPC controller by applying a suboptimal solution after the number of optimization iterations exceeds a specified maximum value.

*Learn More*

Use Suboptimal Solution in Fast MPC Applications

Use Suboptimal Solution for Robotic Manipulator Planning with Nonlinear MPC

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Automatic Code Generation

Perhaps the least invasive way to run MPC faster in simulations, without adjusting any of the problem or solver parameters, is through automatic code generation. All built-in solvers in Model Predictive Control Toolbox support code generation; you can automatically generate C code for your MPC design in MATLAB® (with MATLAB Coder™) or Simulink® (with Simulink Coder™ or Embedded Coder™) and deploy it to an arbitrary number of targets.

By setting the code generation target to MATLAB executable, you can generate and call a mex function of your MPC controller directly in MATLAB to reproduce simulation results and evaluate performance. This approach will lead to much faster MPC calculations, and thus shorter simulation times, which is useful when running multiple simulations using the same MPC controller design. You can change the code generation target to C static library, dynamic library, executable, and so forth by using different settings.

**Tip**

You can automatically generate C/C++ code for your MPC controller and use it for faster simulation and deployment.

*Learn More*

Parallel Parking Using Nonlinear MPC with Automatic Code Generation

Simulation and Code Generation Using Simulink Coder

Generate Code to Compute Optimal MPC Moves in MATLAB

## Change Your Approach

If the previous suggestions do not speed up the optimization problem, it may be worth taking a step back to consider alternative approaches. For example, it would not make sense to use an expensive nonlinear MPC controller over a linear one that leads to comparable performance. This section discusses model reduction, the complexity of different MPC approaches, as well as imitation learning—a machine learning method that can be used to avoid solving the MPC problem in real time.

### Prediction Model Complexity

As mentioned previously, MPC operates based on an internal prediction model of the physical plant. This model can be used to plan for several time steps into the future, to ensure that solutions satisfy the system dynamics, to calculate linearizations in nonlinear MPC, for estimation, and so forth. Even one redundant state in the internal MPC model leads to more constraints, more derivative calculations, more integrations, more outputs/estimated states, and therefore a significant computational overhead. Adding a single additional manipulated variable leads to $m$ (control horizon) additional optimization variables. Thus, the complexity of this prediction model, and specifically the number of manipulated variables, number of states, and number of outputs, affects calculations and the overall optimization speed.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

Typically, the number of manipulated variables and outputs is fixed and dictated by the problem/control objective. However, if, for example, there are decoupled input-output channels in the dynamic equations, you may consider designing separate control loops for these, to reduce the number of outputs and optimization variables in the MPC problem. Unlike with manipulated variables and outputs, control engineers have more flexibility and options for reducing the number of states in a model. One option is to use reduced-order modeling techniques where the objective is to eliminate states that do not have a significant contribution to the dynamics. Find out more about model reduction techniques with Control System Toolbox™.

Another option is to simply use a lower-fidelity model. MPC can rely on state feedback to compensate for model discrepancies, as long as the internal plant captures the dynamics essential to the control objective. Finally, consider whether you even need to capture the dynamics of the system you are working with. For example, Figure 5 shows an MPC controller planning collision-free joint trajectories for a robotic manipulator. In this scenario modeling every joint as a simple double integrator would suffice to solve this problem.

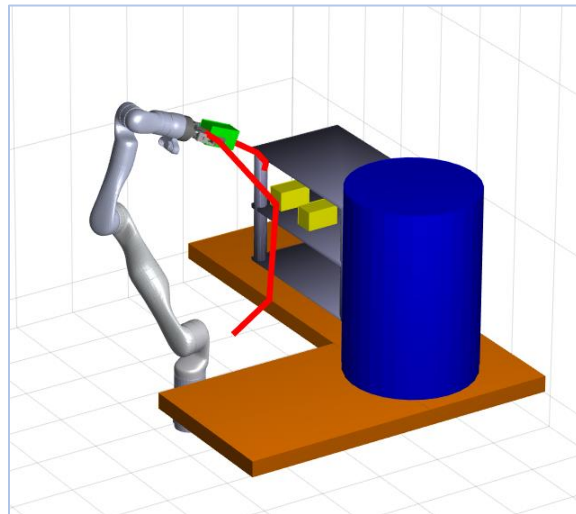

Figure 5. Planning a collision-free trajectory with MPC for a pick-and-place application.

**Tip**

Keep the number of inputs/manipulated variables, states, and outputs of the internal plant model small to simplify the MPC optimization problem.

*Learn More*

Model Reduction Techniques

Pick-and-Place Workflow for Robotic Manipulators with Nonlinear MPC

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

## MPC Type

Figure 6 shows the different types of MPC controllers supported by Model Predictive Control Toolbox. Depending on the problem, selecting the appropriate MPC approach can lead to significant improvements in execution speed:

- In linear time-invariant (LTI) MPC, the internal plant is linear and remains constant across time steps and across the prediction horizon.
- An adaptive MPC controller requires more run-time computations than LTI MPC, since the internal plant is updated at each time step (but does not vary across the prediction horizon). In addition, adaptive MPC requires you to implement a model-updating strategy, which might be computationally intensive.
- Linear time-varying (LTV) MPC is similar to adaptive MPC but updates the internal plant at each time step and across the prediction horizon.
- Explicit MPC, unlike the previous approaches where calculations are made online, uses offline computations to determine regions of the state space where control laws are linear in state. These precalculated controllers are stored and then used at run time instead of solving an optimization problem.
- Gain-scheduled MPC switches between a predefined set of LTI MPC and explicit MPC controllers, in a coordinated fashion, to control a nonlinear plant over a wide range of operating conditions. To implement gain-scheduled MPC, you first need to design an LTI MPC or explicit MPC controller for each operating point, and then design a scheduling signal that switches the controllers at run time.
- Nonlinear MPC deals with nonlinear constraints and plant and possibly nonquadratic cost functions.

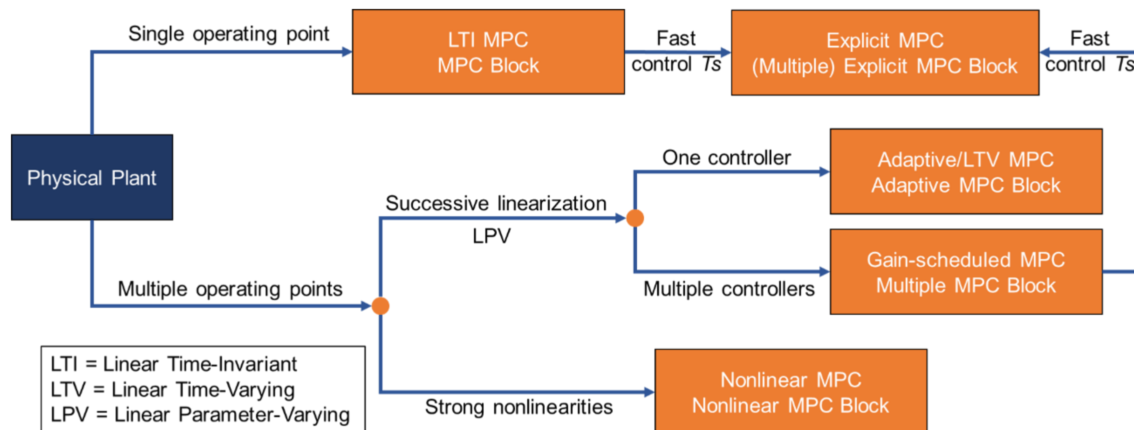


Figure 6. Types of MPC supported by Model Predictive Control Toolbox.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Figure 7 shows how different MPC approaches compare in terms of speed and memory requirements. Explicit MPC leads to fastest execution but also has the largest memory footprint, as well as a more limited set of features compared with other approaches. Nonlinear MPC is the most powerful, generic approach for MIMO systems. However, it is also the most computationally expensive approach, with the highest memory footprint after explicit MPC. Memory requirements for gain-scheduled (explicit) MPC depend on the number of controllers designed, which is also reflected in Figure 7.

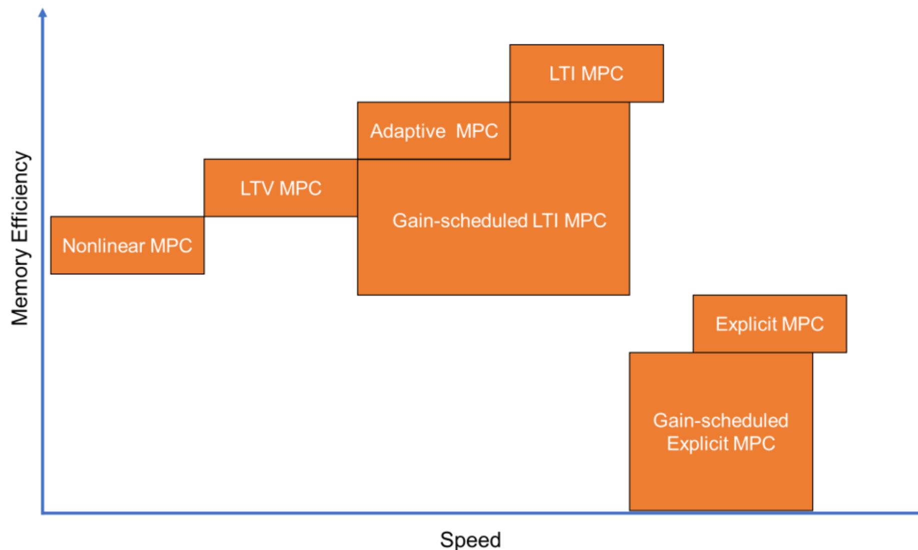


Figure 7. Speed and memory comparison of different MPC approaches in Model Predictive Control Toolbox.

To ensure fastest run-time execution, use the simplest MPC approach that leads to acceptable performance. For example:

- If you are working with a linear or linearized system, start with LTI MPC, and gradually move to gain-scheduled, adaptive, or LTV MPC as needed until performance meets expectations.
- Adaptive and LTV MPC should generally be preferred over gain-scheduled MPC when possible as the former two techniques require only one controller to be designed and are also less sensitive to switch scheduling.
- For fast MPC applications, consider switching to explicit MPC if possible.
- If the physical plant is highly nonlinear, you may start by designing a nonlinear MPC controller and then evaluate whether LTV or adaptive MPC can achieve the same performance. Model Predictive Control Toolbox lets you easily run a nonlinear MPC controller as linear (adaptive or time-varying) by setting the RunAsLinearMPC option in the `nlmpc` object.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

> **Tip**
>
> To ensure the fastest run-time execution, use the simplest MPC approach that leads to acceptable performance.

## Hierarchical Architecture

Due to the predictive nature of the optimization, as well as the internal plant model used in the process, MPC can also be used for planning problems. For example, instead of directly using the solution of the optimization problem for low-level control, MPC could apply this solution to the internal plant model to generate reference output trajectories for another controller to track. Figure 8 summarizes the two architectures. Architecture I uses MPC, in a sense, for both planning and control as there is no separate planning module. Architecture II decouples planning and control and has a dedicated planner (could be implemented with MPC or not) that generates reference trajectories and a dedicated controller (MPC or other) that is tracking these references. In this second architecture, if the environment is static, the planning part could also be performed offline. If there are moving obstacles that the system needs to avoid, dynamic planning with the planner in the loop is necessary. Typically, in architecture II, the outer loop (planning) does not need to run as fast as the inner loop (tracking control).

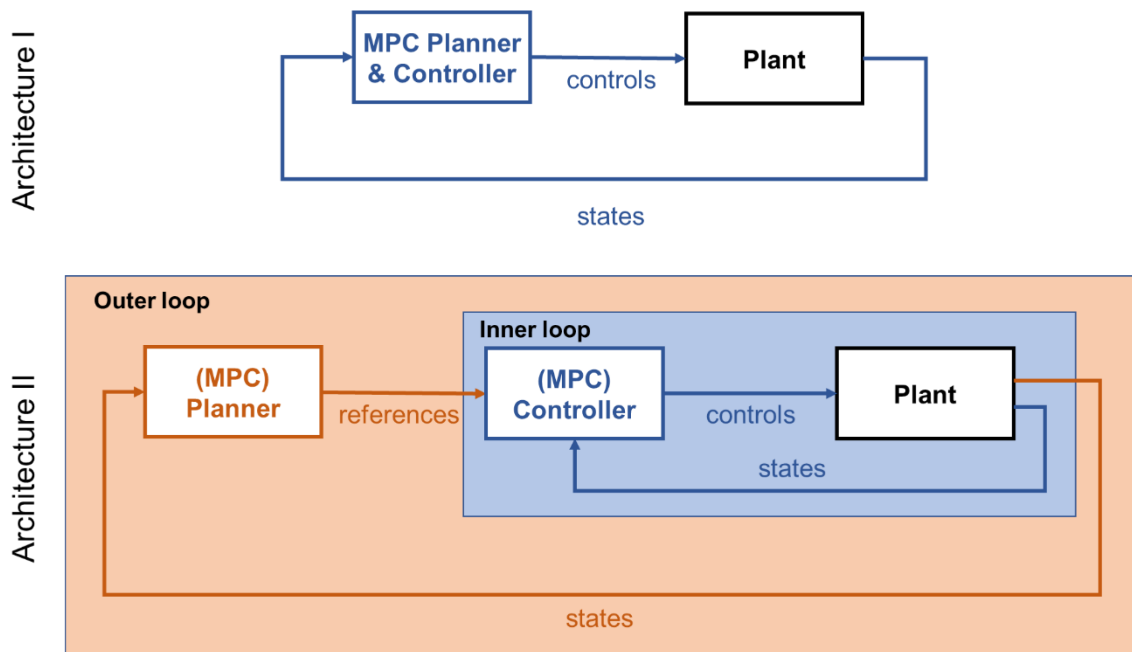


Figure 8. Different approaches in control system architecture.

By decoupling planning and low-level control, the controls engineer has more flexibility on how to set up the control system. For example, imagine a scenario where MPC is in architecture I, but the optimization is not fast enough for real-time application. If the problem was restructured using architecture II, the inner loop would still need to run at the same (fast) sample rate.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

However, with planning and control now decoupled, the MPC optimization problem in this case would be simpler, which could potentially allow faster control rates. If the improvement is not sufficient for using MPC in the inner loop, the outer loop would still be a valid option. As mentioned earlier, the outer loop typically runs at a slower rate than the inner loop, which makes it easier for the optimization to satisfy the real-time requirement. In this case, trajectory tracking in the inner loop could be accomplished with PID control, for example. Note that the outer loop planner does not need to be based on MPC either; popular planners like RRT can be used instead.

> "A small team of engineers pulled together an autonomous vehicle with off-the-shelf hardware and control algorithms developed and implemented with Model-Based Design. Though the system isn't production-ready, it does demonstrate important design concepts with a pragmatic design approach."
>
> — *Dr. Mark Tucker, TMETC*

As an example, architecture II was used by Tata Motors European Technical Centre (TMETC) to develop and deploy autonomous driving software in a Tata Hexa SUV. The team used Model Predictive Control Toolbox to develop lateral and longitudinal controllers that track reference set points.

**Tip**

Using a hierarchical, two-stage control system architecture decouples planning from (low-level) control and may simplify the optimization problem enough for MPC to be applicable in either or both stages.

### Imitation Learning

Recent advances in computing power have greatly reduced the training time of machine learning models. As an alternative to methods described earlier, you may want to rely on machine learning to speed up your MPC controller. At its foundation, an MPC controller is just a function that takes in state values and outputs control values; you just don't have the model that converts states to controls in closed form, since calculations are optimization-based. This is where a machine learning model can be used to help you parameterize and approximate, or imitate, the behavior of an MPC controller.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

Imitation learning is effectively a regression problem (supervised learning), so it requires training data (state values and associated MPC control values). Training time depends on the size of the dataset, the type of surrogate model used, and the complexity of the MPC problem solved. For imitation learning, MPC calculations are completed offline to create the training dataset. It is crucial to carefully construct the training dataset to be representative of the region or data that the surrogate model/controller will operate on; model performance will only be as good as the training data. Figure 9 shows performance of a neural network approximating an MPC controller designed with Model Predictive Control Toolbox for lane-keeping assist (LKA). The behavior of the neural network and the original MPC controller are almost identical.

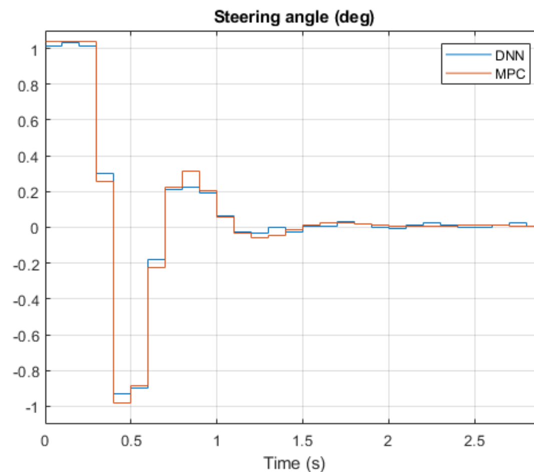


Figure 9. Comparison between original MPC controller and neural network approximation (DNN).

Imitation learning leads to faster MPC executions, since the corresponding optimization problems are solved offline, and only a forward pass on the learned model is necessary to calculate controls for given state values. Compared with explicit MPC, where lookup tables are used to switch between explicit, linear-in-state models for different regions of the state space, imitation learning allows more flexibility as the surrogate models can be nonlinear with respect to the state. Theoretically, this means that a single highly nonlinear surrogate model could accurately approximate the response of an MPC controller, unlike explicit MPC, where different models map to different state-space regions. For example, (deep) neural networks are great function approximators that can learn complex nonlinear mappings by using the appropriate layer architecture. Finally, unlike explicit MPC, there is no need to identify polyhedral regions in imitation learning, which could lead to reduced memory footprint.

If machine learning models have so many advantages, you may be wondering, why is imitation learning not the standard approach for real-time MPC? Machine learning models like neural networks have a large number of parameters, which make it hard to intuitively understand the internal mechanics that generate the model output. This lack of explainability is the main reason why surrogate models like neural networks are often treated like black boxes. Additionally, unlike traditional control methods, performance of machine learning models is more challenging to verify and often involves exhaustive simulations. These limitations can be prohibitive,

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

especially for safety-critical applications. If your application is a good fit though, imitation learning can be a good way to speed up your MPC controller.

**Tip**

Imitation learning allows you to approximate the behavior of an MPC controller with machine learning models like neural networks. For best performance, make sure the dataset you use for training is representative of the state-space region you want the MPC approximation to operate in.

*Learn More*

Imitate MPC Controller for Lane Keeping Assist

Imitate Nonlinear MPC Controller for Flying Robot

## Memory Requirements

Another important performance aspect to consider when designing an MPC controller is memory footprint. This is particularly important for deployment, since embedded devices typically have limited memory capacity. While this white paper focuses on the execution time of MPC controllers, many of the design parameters discussed previously also affect the memory requirements of the controller.

**MPC problem parameters.** This table associates the dimensions of the matrices Model Predictive Control Toolbox generates for a linear MPC problem with several parameters discussed in the previous section. Parameters like the prediction horizon, control horizon, number of manipulated variables, number of optimization variables, number of states and outputs, and number of constraints all increase the memory requirements of a (linear) MPC controller. Online update features for constraints, weights, plant models, and horizons require more RAM, since some intermediate matrices used in the optimization are no longer constant.

**Solver.** The solver choice and configuration also play a role; for example, a sparse QP solver reduces the memory footprint of the Hessian matrix if the number of manipulated variables and free moves is large.

**MPC type.** As Figure 7 shows, memory requirements vary with type of MPC as well, with LTI MPC and (gain-scheduled) explicit MPC having the smallest and largest memory footprint, respectively.

**Data type.** Setting the controller to operate with single-precision data for both simulation and code generation leads to smaller memory footprint.

In summary, as you are tuning MPC parameters to optimize execution speed, make sure you consider the effect your choices may have on memory footprint of the controller.

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

**Tip**

Embedded applications require small memory footprint. As you are tuning MPC parameters to optimize execution speed, consider the effect your choices may have on memory footprint of the controller.

*Learn More*

[QP Problem Construction for Generated C Code]

[Simulate and Generate Code from MPC Controller in Single Precision]

## Conclusion and Next Steps

This white paper presented various ways to speed up MPC controllers. For a summary of the tips and tricks covered, see Table 2.

### Table 2. Tips and tricks to speed up model predictive controllers.

| | |
|---|---|
| **Pick Appropriate Parameter Values** | To minimize the number of computations, choose prediction *Ts* and control *Ts* that are fast enough to satisfy control requirements, but not any faster. |
| | To run MPC at a fast rate, consider choosing control *Ts* < prediction *Ts* to retain a reasonably sized optimization problem that can be solved faster than the control sample time. |
| | As the prediction horizon increases, the controller can better anticipate and plan for future events, but the solution time and memory requirements for the controller increase. |
| | $m<<p$ means fewer variables to optimize at each control interval, which promotes faster computations. |
| | Similar to the simpler control horizon concept, manipulated variable blocking allows you to control the number of decision variables at each control interval, which promotes faster computations. |
| | Keep the number of constraints small and opt for soft as opposed to hard constraints when possible. |
| | To limit the number of calculations required at each time step, keep the number of run-time parameter changes small. |

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

| | |
|---|---|
| **Choose the Appropriate Solver and Solver Options** | The size and configuration of the MPC problem affects the relative performance of the available solvers. To determine which solver is best for your application, consider simulating your controller across multiple simulation scenarios using different solvers. |
| | For feedback control, it is best practice to warm-start your solver with a feasible guess, especially in nonlinear MPC. |
| | Analytical Jacobians and automatic differentiation help speed up the solution of the optimization problem in nonlinear MPC. |
| | You can guarantee the worst-case execution time for your MPC controller by applying a suboptimal solution after the number of optimization iterations exceeds a specified maximum value. |
| | You can automatically generate C/C++ code for your MPC controller and use it for faster simulation and deployment. |
| **Change Your Approach** | Keep the number of inputs/manipulated variables, states, and outputs of the internal plant model small to simplify the MPC optimization problem. |
| | To ensure the fastest run-time execution, use the simplest MPC approach that leads to acceptable performance. |
| | Using a hierarchical, two-stage control system architecture decouples planning from (low-level) control and may simplify the optimization problem enough for MPC to be applicable in either or both stages. |
| | Imitation learning allows you to approximate the behavior of an MPC controller with machine learning models like neural networks. For best performance, make sure the dataset you use for training is representative of the state-space region you want the MPC approximation to operate in. |

Take the next step to speed up your MPC project with Model Predictive Control Toolbox.

**Explore**

Understanding Model Predictive Control (7 Videos) – Video Series

How to Design Model Predictive Controllers (3:00) – Video

How to Implement Model Predictive Controllers (3:09) – Video

**See Real-World Examples**

Tata Motors European Technical Centre Accelerates Development of Autonomous Vehicle Control Algorithms with Model-Based Design – User Story

Developing Longitudinal Controls for a Self-Driving Taxi – User Story

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.

Hitachi Automotive Systems Develops a Model Predictive Controller for Adaptive Cruise Control with Model-Based Design – User Story

Model Predictive Control Approach to Design Practical Adaptive Cruise Control for Traffic Jam – Technical Article

**Download**

Trial Software for Model Predictive Control Toolbox

© 2021 The MathWorks, Inc. MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See mathworks.com/trademarks for a list of additional trademarks.
Other product or brand names may be trademarks or registered trademarks of their respective holders.