

Model Predictive Control

S. Keshav
University of Cambridge

December 29, 2020

Abstract

This short note presents a conceptual overview of Model Predictive Control.

1 Introduction

The discussion in the text so far has been of classical (PID) and state-space control. These techniques work well when the system (or *plant*) is linear (or can be easily linearized in the neighbourhood of the operating point) and there is little coupling between the outputs, that is, the outputs are relatively independent of each other, depending only on the system state and plant inputs. However, many practical systems are non-linear and with coupled outputs. These systems are better controlled using *Model Predictive Control* (MPC).

At its heart, an MPC controller uses a *model* of the system to *predict* its expected evolution in response to its controlled and uncontrolled inputs. Specifically, the system is assumed to be fully described by its *state variables*. The model mathematically describes the relationship between the system's inputs and its outputs and the predictions are of the system state for the immediate future. Given these predictions, unlike PID controllers, whose controllers are relatively simple functions of the measured error, an MPC controller uses a potentially complex *optimizer* to choose control inputs such that:

- system *inputs* obey operating constraints
- system *outputs* are as close as possible to their reference values and obey operating constraints
- the *cost* of the control actions is minimized

Example 1: An AV Speed Controller We use a running example of a controller for an autonomous vehicle (AV). For simplicity, focus on a MPC-based speed controller, which controls how fast the AV drives using an accelerator and a braking system. The AV's state is its current speed. The model mathematically relates the accelerator setting and brake level (the controlled inputs) as well as environmental factors such as wind direction and speed and road slope (the uncontrolled inputs) to the expected speed of the car (the output). The controller uses this model to predict the future speed of the vehicle in response to both controlled and uncontrolled inputs and uses the prediction to choose a feasible accelerator and braking level such that (a) the vehicle speed matches the desired value and (b) changes in speed do not subject the vehicle to too high a G force. Ideally, this set of control decisions uses the least amount of fuel.

Importantly, to deal with unforeseen disturbances, the optimal *trajectory* of control inputs (i.e., the set of control inputs for the immediate future) is computed but not entirely carried out. Instead, it is periodically recomputed, and after each re-computation only the first few control actions are carried out; the remainder of the trajectory is ignored. Thus, if the anticipated system state diverges from the true measured state, previously computed control actions are ignored.

Example 2: Consider the actions of the AV speed controller when the AV turns a corner and spots a pedestrian on the road, an unanticipated situation. Prior to spotting the pedestrian, the controller would have chosen control inputs to ensure that the AV's speed matched the reference value. However, once the pedestrian is spotted, the prior control trajectory is set aside and a new control trajectory is computed to bring the AV to rest well before a potential collision.

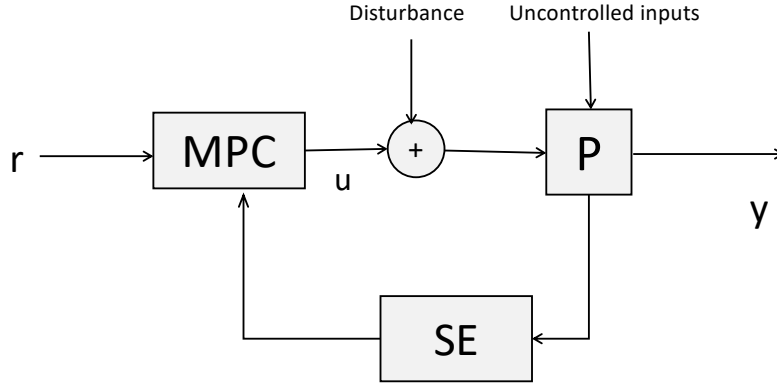


Figure 1: Conceptual model of a Model Predictive Controller (MPC)

Of course, this is wasteful in terms of computation overhead. Hence, an MPC designer must carefully decide how often to control the system, how much of the future trajectory to predict, and how best to model the system so that the optimal trajectory is easy to compute. We will discuss these issues later in this note.

2 Model

Let's make this intuitive description more precise. Consider the controller shown in Figure 1. Here, the reference vector \mathbf{r} is one input to the **MPC** controller, the other input being the output of the state estimator **SE** that observes the plant **P** and estimates its current state. The output of the controller, \mathbf{u} , is subject to disturbances before being given to the plant. The output of the plant is \mathbf{y} , and our principal control goal is to ensure that $\mathbf{y} = \mathbf{r}$.

Example 3: For the case of the AV speed controller, the reference value is the scalar desired speed (this is assumed to be set by a higher-level controller). The plant is the AV, and its state corresponds to its speed. The state value is observed by the state estimator and made available to the controller. The output of the system is also its speed. Uncontrolled inputs to the plant include wind, road slope, pedestrians, barriers, and traffic signals. The accelerator and brake positions selected by the controller are the controlled inputs, and may themselves be subject to disturbances; for example, the braking system may apply a braking force that slightly differs from the one intended.

We will assume that the MPC controller operates at discrete time steps, indexed by k , as shown in Figure 2. Each step is called a *sample time*. At the time marked *Now*, the MPC controller uses a model of the system to predict its anticipated future evolution. This requires a prediction of the future uncontrolled inputs over the *prediction horizon*. This prediction can use an arbitrarily complex algorithm; common techniques include Auto-Regressive Moving Average (ARMA) and its many variants, such as Seasonal ARMA, ARIMA, and ARMAX or predictors based on Deep Neural Networks. The prediction horizon should be long enough to take into account significant future events. In the example of the AV speed controller, this would include stop signs and traffic lights anticipated along its path.

Predictions are used to compute the trajectory of future control actions. We would like these actions to be *feasible* and *optimal*. That is, the actions must obey operating constraints, and the anticipated outputs must obey operating constraints, minimize control effort, and allow the output to closely track the reference value. To ensure optimality of the control actions, the MPC uses an *optimizer* to compute the optimal control trajectory over the course of a *control horizon*. The control horizon is typically shorter than the prediction horizon, since it is more difficult to compute, and, in any case, only the first step of the control trajectory is actually implemented.

Note that after the control actions are taken at the present time, time advances by one step, and the predictions and control actions are re-computed over the new prediction and control horizons. For this reason, MPC is also called *receding-horizon* control.

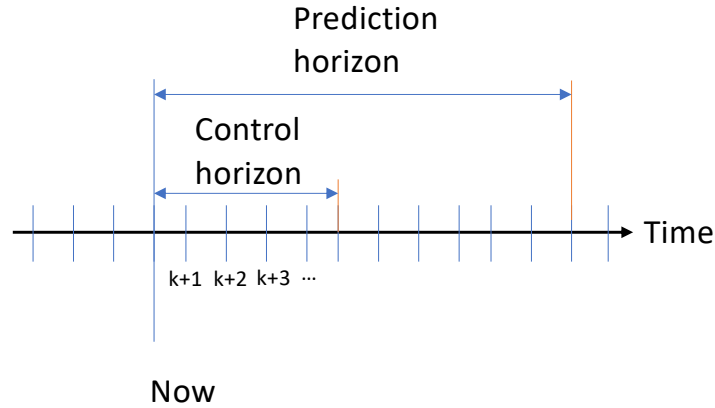


Figure 2: Time diagram

Example 4: For the AV speed controller, the MPC would use an optimizer to compute the accelerator and brake positions for the control horizon, taking into account the predictions of significant future events. However, at each time step, the controller would only actuate the first action in the trajectory.

In computing the control actions, the optimizer could be asked to ensure that the controlled input and outputs obey both *hard* and *soft* constraints. Hard constraints must always be met, whereas a soft constraint, when violated, only results in a penalty, with goal being to minimize this penalty.

Example 5: For the AV speed controller, a hard constraint is the maximum acceleration and the maximum braking level. These constraints must always be met. A soft constraint might be how often the brake level is changed: if this constraint were violated, the AV's passengers would feel the ride was jerky.

Note that the MPC controller does not require the plant or operating constraints to be linear. Instead the complexity of non-linear operation is dealt with by the optimizer and the predictor. The actions of the optimizer can be arbitrarily complex. They could, for instance, require the simulation of many possible future evolution paths of the system, with the best of these chosen as the one to realize. This can place a heavy computational burden on the controller.

In case of a highly non-linear system, predictions may be inaccurate and the system behaviour may be far from optimal. Moreover, the cost of computing the optimal trajectory may be quite high. Nevertheless, the overall control paradigm would still be applicable.

3 MPC parameters

This section discusses how one might choose MPC control parameters such as the sample time, prediction horizon, and control horizon.]<https://www.youtube.com/watch?v=dAPRamI6k7Q>

Sample time: The sample time determine how often the MPC controller takes control actions. The shorter this period, the more responsive the control. However, this also increases the cost of computation. A rule of thumb is to first measure the *rise time* of the uncontrolled system, that is, the time it takes to go from 10% to 90% of its final value, when given a step input. It is best to have the sample time be about 5-10% of the rise time, so that there is ample time to make corrections during the rise time.

Example 6: Consider the AV when it is given a step input, that is, the accelerator setting goes from zero to one unit (arbitrarily defined). We would expect that the speed of the AV would rise as shown in Figure 3. Then, we would like the sample time to be about 10-20 times smaller than this rise time.



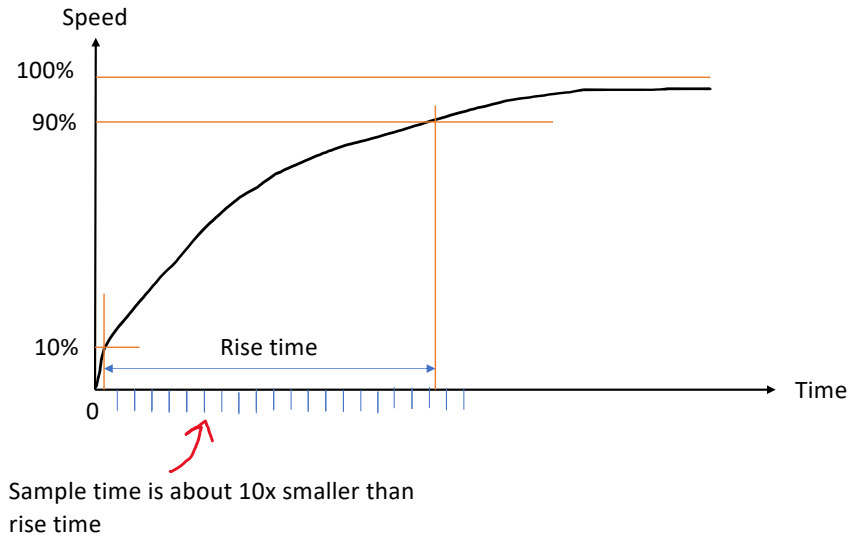


Figure 3: Guidance on choosing sample time.

Prediction horizon: How far should the controller predict into the future? Again, this balances computation time and accuracy of control. In general, the prediction horizon needs to be long enough to cover significant future events, such as an upcoming traffic light for an AV. From a control theoretic perspective, recall that when subjected to a unit step input a system exhibits both a *steady state* and *transient* response. The rule of thumb is that the prediction horizon should include about 20-30 sample times during the transient system response.

Control horizon: The control horizon is the time duration for which optimal control actions computed. One might think that the longer this horizon the better, but this makes the computation more expensive to carry out. Indeed, although the control horizon can be as long as prediction horizon, only the first few steps matter anyway, so there is no need to make the horizon too long. As a rule of thumb, the control horizon should be about 10%-20% of the prediction horizon.

Example 7: Imagine that an on-board map allows the AV to predict that a stop sign is expected, say, 100 time steps in the future. If the prediction horizon was greater than 100 time steps, this significant fact would be available to the MPC controller. Given this information, it should change the desired reference speed trajectory r to slow down the AV from its current speed to zero in 100 time steps. However, the control actions, i.e., the accelerator and brake settings, will need to be computed only for, say, the next 10 steps, which is the control horizon. The control horizon need not be longer since unexpected disturbances, such as wind gusts, may require a re-computation of control settings several times before the stop sign is reached.

Hard and soft constraints: Hard and soft constraints can be set on on both inputs and outputs. If a hard constraint cannot be met, the optimizer will be forced to give up, claiming a solution cannot be found. In this situation, there needs to be a fallback plan, otherwise the system will be uncontrolled! Thus, it is best to define most constraints to be soft, reserving hard constraints only to express safety or operational limits.

Example 8: The upper and lower limits of operation of the accelerator are hard constraints, as is the maximum braking force. On the other hand, the requirements that the speed not change too abruptly from one time step to another is a soft constraint. We can express the expected absolute change in speed as a *penalty variable* in the optimization objective, encouraging the optimizer to reduce this penalty to the smallest possible value.

If a pedestrian were to suddenly step in front of the AV, so that it could not stop in time even with the maximum braking force, then the optimizer would declare infeasibility and give up. A fallback plan, in this case, might be to maximally steer the car away from the pedestrian.

Optimization objective: The optimization objective is an expression or function that an optimizer tries to

minimize or maximize. As an MPC designer, a careful choice of the objective function allows the optimizer to balance between competing goals. Every objective function must include a term that reflects the error between the reference and actual output values, so that this objective is achieved. In addition, the objective function may have terms reflecting the amount of change in inputs or outputs between two control steps, as well as terms reflecting the cost of control (e.g., the amount of fuel used by the engine). We can balance between competing goals by assigning different weights to each goal, as the next example demonstrates.

Example 9: A simple objective function for the AV MPC controller that it would seek to minimize might be $\sum_k \alpha |r(k) - y(k)| + (1 - \alpha) |(y(k) - y(k - 1))|$, where α is a tuning parameter, $r(k)$ is the reference speed, and $y(k)$ is the speed at time step k . When α is close to 1, then the goal is to match the output speed to the reference as fast as possible (even if this results in a jerky ride), whereas when it is close to 0, the goal is to ensure a smooth ride. Intermediate values allow us to balance these goals. In practice, the objective function would need to far more complex to account, for example, for the cost of fuel.

Example 10: For completeness, here is an example of the optimization problem the MPC controller might solve:

$$\min_{acceleration(k), brake(k)} \sum_{k=now}^{k=k+controlhorizon} \alpha |r(k) - y(k)| + (1 - \alpha) |(y(k) - y(k - 1))| \quad (1)$$

$$y(k) = f(y(k - 1), acceleration(k), brake(k)) \quad (2)$$

$$y(k) \geq 0 \quad (3)$$

$$acceleration(k) \geq 0 \quad (4)$$


$$brake(k) \geq 0 \quad (5)$$

where α is a tuning parameter, $r(k)$ is the reference speed, and $y(k)$ is the speed at time step k . f is the mathematical model that relates the prior state and current inputs to current outputs. Note that the objective function includes future time steps, thus requiring a predictor.

4 Some advanced concepts

This section discusses some advanced techniques for model predictive control.

4.1 Dealing with non-linearity

A significant advantage of MPC-based control over other approaches is that it can deal with non-linear systems. With MPC, any or all of the plant model, the constraints, and the control costs can be non-linear. However, this comes at the expense of additional computation. Moreover, there is no guarantee that the outcomes are optimal. 


More specifically, with a linear plant model, linear constraints, and linear or quadratic costs, the optimizer has to solve a convex optimization problem, which is easy to solve, and the outcome is guaranteed to be optimal. However, if any of these are non-linear, then the optimization problem may be non-convex, which can lead to sub-optimal outcomes.

In some cases, the MPC approach can be modified to deal with non-linearities. Two variants of MPC that are widely used in practice to deal with non-linear system models are *adaptive* MPC and *gain-scheduled* MPC. In both approaches, the system is linearized around its operating point, using a Taylor expansion, as discussed on page 327 of the text. If the system has multiple operating points, the output space is decomposed into a set of sub-spaces, and a linearized model is defined for each subspace. This results in a piecewise-linear approximation of the plant.

With *adaptive MPC* the piecewise-linear model is computed on the fly as the system changes. Note that with this approach, the structure of the optimization problem stays the same, only its parameters change to reflect the underlying subspace.

With *gain scheduled MPC*, the output space is linearized offline, and one controller is designed for each linearized model. As the operating point of the system evolves, a higher-level controller switches between the appropriate controllers. This uses more memory than the adaptive approach, but can be more responsive, since the linearization does not need to be computed on-the-fly.

4.2 Improving MPC performance

It should be clear by now that the main problem with the MPC approach is its high computational demand. However, there are some approaches to reducing this computational cost, as discussed next. 

- The computational cost can be reduced by reducing the length of the prediction and control horizon. Generally speaking, prediction is easier to carry out than optimization. So, reducing the control horizon has the greater impact on reducing computational cost. However, especially with non-linear optimization, a reduction in computation may come at the expense of significant sub-optimality.
- In some cases it is possible to optimize over a longer control horizon without significantly increasing costs by using increasing larger time steps over the control horizon. For example, each time step can have a duration that multiplicatively increases the duration of the prior step. This allows the controller to respond to significant events quite far in the future, but with only a logarithmic increase in cost.
- A complex plant model is harder to simulate and optimize. If possible, portions of the plant model that are not strictly necessary should be excised. This is the intuition behind *model order reduction techniques*.
- If floating point operations are computationally expensive, then costs can be reduced by using lower precision or better data representation (as integers, for example, instead of floats).
- With *explicit MPC*, the optimal output of the controller for a given set of inputs is pre-computed and stored in a table. Then, in operation, instead of running the optimizer, the table can be looked up, reducing computational time. Of course, this requires substantial prior computation, as well as the need to store a potentially large lookup table corresponding to all possible operation states. As a compromise, the table could store (or cache) frequently occurring parameters combinations, reverting to standard operation when the system strays beyond the set of known states.

5 Conclusion

Model predictive control harnesses the power of modern microprocessors to compute optimal control actions based on the measured state. Unlike feedback or PID control, which does not work well when the plant model or constraints are non-linear, the MPC paradigm can deal with complex, coupled outputs and non-linearities. However, this comes at the cost of additional computation. Many variants of the basic approach have been developed to reduce this computational cost.

6 Further reading

Here are some references to read further about MPC. Rawlings[5] provides a more mathematical tutorial that links MPC to Linear Quadratic Gaussian control. Prediction using ARMA and its variants is discussed in Reference [3] and more recent approaches are surveyed in [2]. Alessio *et al* [1] survey approaches used for explicit MPC.

7 Acknowledgements

This note is based substantially on a 5-part tutorial from MATLAB on YouTube [4]. Thanks also to Fiodar Kazhamiaka for his comments on an earlier draft.

References

- [1] ALESSIO, A., AND BEMPORAD, A. A survey on explicit model predictive control. In *Nonlinear model predictive control*. Springer, 2009, pp. 345–369.
- [2] HAN, Z., ZHAO, J., LEUNG, H., MA, K. F., AND WANG, W. A review of deep learning models for time series prediction. *IEEE Sensors Journal* (2019).
- [3] MAKHOUL, J. Linear prediction: A tutorial review. *Proceedings of the IEEE* 63, 4 (1975), 561–580.

- [4] MATLAB: Understanding Model Predictive Control. <https://www.youtube.com/watch?v=8U0xi0kDcmw>. Accessed: 2020-12-22.
- [5] RAWLINGS, J. B. Tutorial overview of model predictive control. *IEEE control systems magazine* 20, 3 (2000), 38–52.