Control Models Documentation

Kunal Singla

August 5, 2020

Introduction

Control systems are very important in software engineering. Any machine/robot/vehicle/etc. that needs some degree of precise control will have a certain control model implemented to accomplish this. The purpose of this document is to summarize some popular choices for control models. Control models have varying range of precision and purposes. We will go over the following control models:

I. PID

II. MPC

III. Auto-tuning PID

PID Controller

PID controller stands for proportional–integral–derivative controller which is a feedback-based control algorithm at the center of all basic robotic motion control. The loop uses data (feedback) from a sensor or multiple sensors and determine the precise output of motion. This is done through three different output controls: **Proportional** Control, **Integral** Control, and **Derivative** Control. The algorithm is the following:

$$\textbf{output} = (\textbf{kP} \times error) + (\textbf{kI} \times \sum error) + (\textbf{kD} \times \frac{\Delta error}{\Delta time})$$

Proportional Control involves the usage of a proportional constant (referred to as **kP**) that controls the motion of a system proportional to the error that we receive from a sensor. This factor minimizes position error. For example, lets say we have an elevator that wants to reach a certain floor. As we approach the desired floor level we want to slow the elevator down to a stop. The elevator has an inversely proportional relation to the error between its current floor and the target floor. Many mechanisms don't always work with just proportional control. Sometimes an oscillation occurs where the mechanism overshoots its target by some margin of error and oscillates back and forth between a high and low error. This can be solved by implementing one or both of the following two controls.

Integral Control involves the usage of a proportional constant (referred to as **kI**) that controls the motion based on the accumulated error received from a sensor over time; basically the sum of all past error. The algorithm keeps a running total of all error values it has received and multiplies it by a constant (kI). This helps reduce oscillation from the proportional gain. However, while it may reduce oscillation, it may take time for the system to settle to its target (increase of settling time). This is where derivative control can be of use.

Derivative Control involves the usage of a proportional constant (referred to as **kD**) that controls the motion based on the change in error received from a sensor; essentially reducing the velocity error. Derivative control helps predict future error and decreases the settling time. The derivative constant is also referred to as the dampening constant, because of this characteristic.

MPC Controller

Model Predictive Control is a feedback-based control algorithm that uses a model of a system to make predictions about the future outputs for controlling the system. MPC is especially useful for MIMO (multi-input multi-output) systems because, unlike a PID controller, MPC can factor in multiple inputs and produce multiple outputs for a system. Another feature of MPC is it can factor in soft and hard constraints for the system, like speed-limits for an autonomous vehicle. Furthermore, another benefit of MPC control is that it has the ability to factor in future control decisions to allow for better control performance.

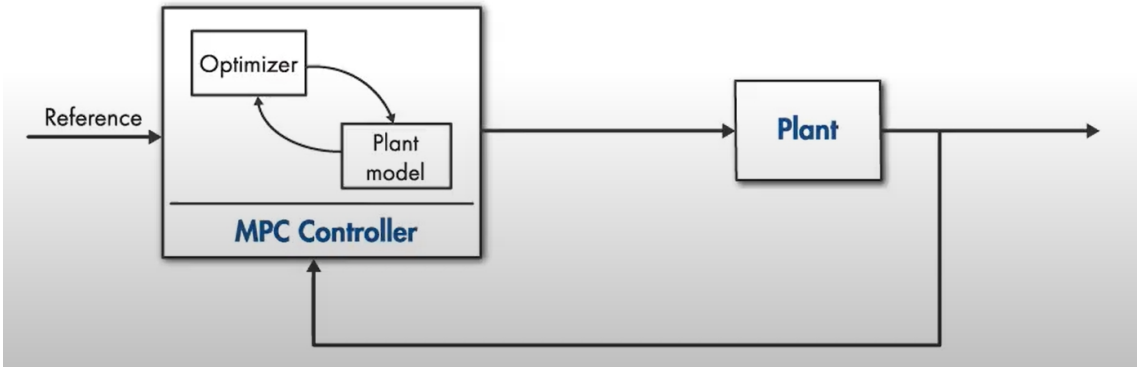

Figure 1. Basic diagram of an MPC controller. Credit: MATLAB

How an MPC Controller Works: An MPC controller controller works by first having a reference. A reference can be thought of as a target/goal you want the system to approach. MPC uses a model of the system along with an optimizer function in order to predict the optimal control output to reach its reference. It then sends the output to the "plant" (the system) and feeds the output of the system back into the MPC controller.

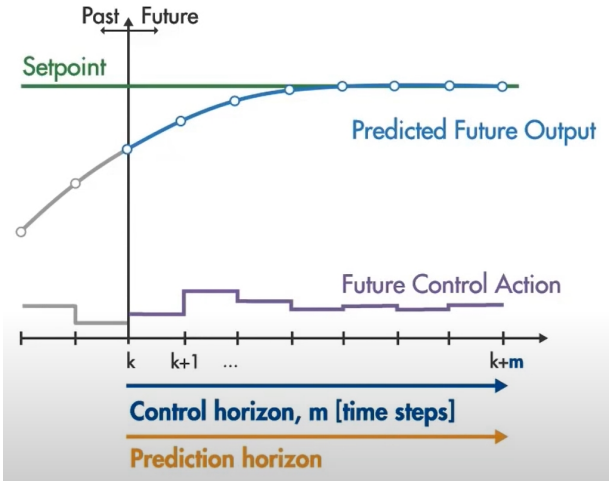

Figure 2. Diagram of MPC Design Parameters. Credit: MATLAB

MPC Design Parameters: There are a few different design parameters to take into account when designing an MPC controller for a particular system.

1. **Sample Time:** This is the rate at which the MPC control agorithm is triggered. If the sample time is too large, the control algorithm can have a delayed response to disturbances in the system. A sample time too small can definitely resolve any disturbances much quicker, but requires much more computational power.

2. **Prediction Horizon:** This is how far in time the control algorithm predicts the output of the system. A short prediction horizon can result in late control actions for the system. A

large prediction horizon may result in unnecessary calculation and you may end up wasting computational power for unused/unimportant data.

3. **Control Horizon:** The control horizon is how far in time the control algorithm plans future control actions. The shorter the control horizon, the fewer the number of computations. However. keeping a short control horizon may not give us optimal maneuvers or control outputs for the system. And keep a long control horizon results in much more computation requirements.

4. **Constraints:** These are hard or soft bounds set on control actions for a system. For example, an autonomous vehicle may have to adhere to speed limits, which can be taken into account using constraints.

5. **Weights:** For MPC controllers, weights are used to achieve a balance between reference tracking and smooth control moves. They are also used to assign weights to multiple outputs. In a MIMO system, if one output is more important than another, weights can be assigned to model that importance.

Different Types of MPC Algorithms: Depending on the system, the constraints, and the cost function there can be a number of ways to approach the different MPC control algorithms out there. In a completely linear system, with linear constraints, and a quadratic cost function a **linear MPC** control algorithm can be used. This results in a convex optimization problem. In a nonlinear system, with linear constraints, and a quadratic cost function you can still use a **linear MPC** control algorithm. This results in two different types of linear MPC algorithms: **Adaptive MPC** and **Gain-Scheduled MPC**. In a non-linear system, with non-linear constraints, and a non-linear cost function a **non-linear MPC** control algorithm can be used. This results in a non-convex optimization problem. This can be very computationally complex and may require more powerful hardware.

Auto–Tuning PID

Ziegler–Nichols method

Control Type	K_p	K_i	K_d
P	$0.50K_u$	—	—
PI	$0.45K_u$	$0.54K_u/T_u$	—
PID	$0.60K_u$	$1.2K_u/T_u$	$3K_uT_u/40$

Figure 3. Table showing the Ziegler-Nichols method. Credit: Wikipedia

Tuning a PID controller can be time consuming and imperfect through trial and error. For that reason, there are many tuning methods used to perfect a PID controller. In many cases, the fastest way to tune a PID algorithm is by using software to automatically tune it. There are 2 popular ways to tune a PID controller.

1. **Ziegler-Nichols Method:** There are methods to implement an automated version of the Ziegler-Nichols method (Figure 3) to tune your PID system directly. You can read more about tuning using the Ziegler-Nichols methods here.

2. **Simulation Software:** Simulation software – such as MATLAB – can be used to automatically tune a PID controller given you have a mathematical model of the system. With a mathematical model, not only can you tune a PID controller to compute the kP, kI, and kD gains, but you can also visualize and tweak the generated controller. This allows for increased flexibility and customization of controller response.