Advanced Optimization-based Robot Control

Marco Peressutti
230403
marco.peressutti@studenti.unitn.it

November 19, 2021

Contents

Chapter 1

Optimal Control

1.1 Introduction

Optimal control is quite different type of control theory w.r.t. reactive control, but the advantage of the control methods to be discussed is that they can be utilized even to do planning (what trajectory you are going to execute before you actually execute it).

1.1.1 History

Optimal control was born in the XVII century and was created by Bernoulli (1696). The first control problem that was solved can be described as follows:

Imagine that you have a ball in a certain position A and you would like that ball to reach a target position B.

Under this condition Bernoulli asked himself, what is the shape of the ground so that the ball in A reaches B in minimum time?

Several possibilities can be considered:

1. a straight line
2. something that goes down sharply and then reaches the target point with a straight line.

Turned out that was neither of those: they are both slower than minimum time, which materializes with a brachistochrone curve.

So Optimal control is a framework that helps us answer questions such as: what is the trajectory that allows me to go from this place to another

- in minimum time
- consuming the minimum amount of energy
- reaching the maximum velocity possible at the end of the trajectory.

These kind of question where we want to find the best motion for doing something, and by best we mean that it maximizes a certain critiria.

Before we dive deep into what optimal control is all about let us clarify what is the difference between Optimal control and Reactive control:

1. One of the key feature of reactive control which we have seen so far is that it always assume that you have a reference trajectory that you want to track. But we have never answer the question how do we actually compute these reference trajectories?

There are some cases where those reference trajectories are basically given by the problem (e.g. robot that cuts a material with a laser, knowing the profile of the cut you automatically know the trajectory that the end-effector should follow), but there are many cases where it is not obvious what trajectory the robot manipulator should follow.

In all of those circumstances where there is no clear trajectory to follow Reactive control is not enough because it does not allow to choose a specific solution for this problem, on this aspect Optimal control is the superior tool to leverage.

2. In reactive control the performance of the system are measured using quantities such as rise time, overshoot and settling time. In Optimal control, instead, we can use many more criteria in order to optimize the quality of the motion (minimize the energy, maximize the speed, minimize the time taken to do the motion)

3. In optimal control since it is based in optimized it is really easy to handle constraints. Inequality constraints are commonly used to represent the limits of the system (current limits, friction cones, joint limits, torque limits).

Reactive control	**Optimal control**
need reference trajectory	no need for reference trajectory
rise time	objective clearly specified throguh cost function
overshoot	
settling time	
hard to handle constraints (e.g. joint position limits)	can handle constraints

1.1.2 Optimal control problem (O.C.P.)

An optimal control problem takes the form of a minimization problem, but it is not really a minimization problem:

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \int_0^T l(x(t),u(t),t)\,dt + l_f(x(T))$$

subject to:

- $\dot{x}(t) = f(x(t),u(t),t) \quad \forall t \in [0,T]$ (dynamics) — `dynamics`
- $x(0) = x_0$ (initial conditions) — `initial conditions`
- $g(x(t),u(t),t) \leq 0 \quad \forall t \in [0,T]$ (path constraints), e.g. actuator limits — `path constraints`

where:

- $\int_0^T l(x(t),u(t),t)\,dt$ is the running cost — `running cost`
- $l_f(x(T))$ is the terminal cost — `terminal cost`

As said before this is not a standard minimization/optimization problem, but an optimal control problem because

- $x(\cdot), u(\cdot)$ are trajectories, i.e. infinite-dimensional objects.
- the constraints of the problem are infinitely many, given that they must be valid for all instances in the range $[0,T]$.

Example of OCP: Pendulum

We can imagine a pendulum robot and we want him to reach the vertical position (i.e. $q = 0$), where q is the angle between the pose of the robot and the desired pose.

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}} \int_0^T q(t)^2\,dt + q(T)^2$$

subject to:

- $I\,\ddot{q} = \tau + m\,g\,sin(q) \Rightarrow \begin{bmatrix}\dot{q}\\ \ddot{q}\end{bmatrix} = \begin{bmatrix}\dot{q}\\ I^{-1}(\tau + m\,g\,sin(q))\end{bmatrix}$
- $q(0) = q_0$
- $\dot{q}(0) = \dot{q}_0$
- $q^{min} \leq q(t) \leq q^{max} \qquad \forall t \in [0, T]$
- $|\tau(t)| \leq \tau^{max} \qquad \forall t \in [0, T]$

where

- $x \triangleq (q, \dot{q})$
- $u \triangleq \tau$

However the formulation of the running cost will make the robot manipulator reach $q = 0°$ as fast as possible, and maybe it is not something that you want. For this reason, additional terms are also included in the running cost. e.g,

$$\int_0^T (q(t)^2 + u(t)^2 + \dot{q}(t)^2)\,dt$$

1.1.3 Optimal control method families

There are many different approaches to solve a control problem. Some of them works in Continous Time, other in Discrete time, some of them are global and other are local.

	optimize ⇒ discretize Continuous time	discretize⇒optimize Discrete time
Global	Hamilton-Jacobi-Bellman (HJB)	Dynamic Programming (DP)
Local	Pontryagin Maximum Principle (PMP) Calculus of variations Indirect methods	Direct methods

1.2 Dynamic Programming (DP)

1.2.1 Principle of optimality (Bellman)

Dynamics Programming is based on the principle of optimality, aka the Bellman principle.

principle of optimality

Bellman principle

Let us suppose we have an optimal trajectory to go from one point to another.

48.26

What the principle of optimality says is that if you have an optimal trajectory then each subarc is in itself an optimal arc.

In other words, if the trajectory is the optimal trajectory in order to go from point A to point B, then no matter which initial point of the curve you start from the arc connecting such initial condition to the final point is the optimal trajectory.

This is rather trivial because if there is another arc that is the optimal trajectory to reach the final point than I could replace such arc to the original optimal trajectory, but since the original one is optimal by assumption that cannot be the case.

The principle of optimality can be used to simplify the solution of a discrete time optimal control problem.

1.2.2 Dynamic Programming approach

In dynamic programming the optimal control problem is assumed to be already in discrete time; that might be bacause we already have a discrete time problem (e.g. games, chess) or might be a continuous time problem that you discretized in time.

Therefore, the trajectories $x(\cdot)$ and $u(\cdot)$ are not in continuous time, but they are trajectories in discrete time (basically matrices).

The integral is replaced by a summation (equivalent of the integral).

$$\underset{X,U}{\text{minimize}} \sum_{i=0}^{N-1} l(x_i, u_i)$$

The dynamics that we have as a constraints are a discrete time dynamics. subject to:

- $x_{i+1} = f(x_i, u_i) \qquad i = 0 \dots N-1$
- No terminal cost for sake of simplicity

To apply the principle of optimality to dynamic programming we need to split this problem into 2 parts.

Take M s.t. $0 \leq M < N$. Then we split the problem cost:

$$\underset{X,U}{\text{minimize}} \left[\sum_{i=0}^{M-1} (l(x_i, u_i) + I(x_{i+1} - f(x_i, u_i))) + \sum_{i=M}^{N-1} (l(x_i, u_i) + I(x_{i+1} - f(x_i, u_i))) \right] \tag{1.1}$$

$$\underset{X,U}{\text{minimize}} \left[c_0(X_{1:M}, U_{0:M-1}) + c_M(x_M, X_{M+1:N}, U_{M:N-1}) \right] \tag{1.2}$$

In this formulation that constraint has been integrated in the cost function through the Indicator function. Such operator is zero when the input is zero and it is infinity otherwise.

Indicator function

Our constraints can now be seen as a high cost term if it is violated.

Now instead of doing the minimization of the sum of the two terms, since they depend on the different part of the trajectory (even though they both depend on x_M) I can treat them as decoupled and find the minimum of c_M, find the value of x_M and lastly minimize c_0.

Formally:

$$\underset{X_{1:M}, U_{0:M-1}}{\text{minimize}} \left[c_0(X_{1:M}, U_{0:-1}) + \underset{X_{M+1:N}, U_{M:N-1}}{\text{minimize}} \; c_M(x_M, X_{M+1:N}, U_{M:N-1}) \right]$$

$$V_0(x_0) = \underset{X_{1:M}, U_{0:M-1}}{\text{minimize}} \; c_0(X_{1:M}, U_{0:M-1}) + V_M(x_M)$$

We have still an optimal control problem, which parameters are still hidden inside c_0, but the big difference is that now we are optimizing over a smaller trajectory thanks to a separation of the problem formulation into two smaller minimization problems.

And since M can be chosen to be every value we want, we can repeat the same process as many times as we want and make the smaller problems to be optimizing over one value of the trajectory (i.e. one time step).

We therefore solve a sequence of problems: one for each time step, which simplifies drastically the original problem.

By iterating over we can get the optimal solution of the form:

$$V_i(x_i) = \underset{u_i}{\text{minimize}} \left[l(x_i, u_i) + V_{i+1}(f(x_i, u_i)) \right]$$

$V_M(x_M)$ can be interpreted as the optimal cost that we have to pay if we start from state x_M at time M onward and behaving optimally. This function is called Value function or Optimal cost-to-go.

Value function

Optimal cost-to-go

In conclusion:

Given a discrete-time finite-horizon OCP, we can initialize the dynamic programming algorithm writing the value function for the last time step

$$V_N(x_N) = l_f(x_N)$$

And then you use the recursive optimality principle (backward in time):

$$V_i(z) = \underset{u}{\text{minimize}}\, l(z, u) + V_{i+1}(f(z, u)) \qquad i = N-1, ..., 0$$

Once you have V_i $\forall i \in [0, N]$ you can compute the optimal control as:

$$\boxed{u_i^*(x) = \underset{u}{\text{argmin}}\, l(x, u) + V_{i+1}(f(x, u))}$$

Observations:

- u_i^* is not only an optimal control trajectory, but an optimal feedback control policy, because u can be computed as function of x so depending where we are in the state we get a different value of u.
- The function to minimize is often called Q function.
- the formulation of $V_i(z)$ is a parametric optimization, which is different from numerical optimization, because the result is not a value, but a function.

optimal feedback control policy

Q function

parametric optimization

numerical optimization

In summary:

1. Dynamic Programming is a global method, so it tries to get the global solution for a discrete time OCP.

2. The solution is not an open loop trajectory but it is a feedback control policy. So at every state and every time it tells you what the optimal solution is.

3. The main problem with dynamic programming is that it is applicable only in specific cases.

   (a) discrete state and control space. So the number of states and control inputs is bounded. (tipically not the case in robotics) In this scenario parametric optimization is replaced by numerical optimization applied to all possible states.

   The results are stored in a table called lookup table.

   lookup table

   The main disadvantage is the so called Curse of dimensionality. In fact, one may think, if we have a continuos robotic problem, can we discretize the state and the control space with a sufficiently fine grid. Yes, but the only problem is that if you want to discretize with a fine grid than you will get a very large state and control space (even if it is discrete).

   Curse of dimensionality

   So suppose that you have a robot manipulator with 6 joints and you want to discretize the range of each joint both in position and velocity in 10 values. In this case you are going to have 10^{12} possible states which is a big number.

   The table that you are going to store in memory is of the same size, it has 10^{12} columns.

   So Dynamic programing is only applicable up to 2 to 4 states-control.

   However, in Dynamic programming it is easy to handle integer variables.

(b) Linear dynamics and quadratic cost function. In this case the problem that you have to solve is a Linear Quadratic Regulator (LQR).
The problem that we try to solve becomes a quadratic program that has a closed solution.

Linear Quadratic Regulator (LQR)

4. There are variants of Dynamic programming that go under the name of Approximate Dynamic Programming or Neural Dynamic Programming and those are basically Reinforcement Learning algorithms.

Approximate Dynamic Programming

Neural Dynamic Programming

1.3 Direct methods

Direct methods is a family of methods where there are different classes of methods, and the main difference w.r.t. Dynamic Programming is that they are local: i.e. they do not try to solve the problem for every possible state at every possible time and find the global optimum, but they just try to find a solution that is locally optimal.

Direct methods

It means that if you try to perturbe the solution you can only do worst.

1.3.1 Introduction

The key idea of Direct methods is really simple and intuitive.

Considering the following optimal control problem, which looks a lot like an optimization problem, you proceed to discretize the parameters of the problem. So you take the trajectory and you parametrize such trajectory (e.g. with a polynomial) and obtain a finite number of parameters to represent this trajectory (e.g. coefficients of the polynomial).

In this way the number of decision variables is now finite.

The second problem related to the constraints (which are infinitly many) and you discretize em in time: i.e. you check the constraint at each time step (e.g. every 10ms).

The problem now becomes an optimization problem, so you can apply Newton Method to try to find the local solution.

However, the discretization process is not as simple as it sounds.

In summary:

- Discretize the Optimal control Problem and use a non linear programming solver (NLP) to find a local optimum.

non linear programming solver (NLP)

- The Discretization takes two steps:
  1. Parametrize state and control trajectories (e.g. with polynomials).
  2. Enforce constraints (both dynamics and path inequalities) on a grid.

$$t_0 < t_1 < t_2 < ... < t_N$$

Tons of theory/methods to choose trajectory parametrization and time grid so that NLP is a good approximation of OCP and well conditioned (a matrix is ill-conditioned if it has both very large and very small elements, in this conditioned it may occur loss of precision).

well conditioned

ill-conditioned

- There are three main families of Direct methods:
  1. Single Shooting aka Sequential approach.

Single Shooting

Sequential approach

     - Discretize only the control trajectory ($u(t)$).
     - Comput the state $x(t)$ integrating the dynamics.
     - No need to have dynamic constraints (you can omit the state as a control variable of the problem, because you can indirectly compute it from $u(t)$ and from the knowledge of the dynamics).

  2. Collocation aka Simultaneous approach.

Collocation

Simultaneous approach

     - Discretize both $x(t)$ and $u(t)$
     - Enforce the dynamics on a time grid. In this case you have to use the dynamics as a constraint to make sure since you have both $x(t)$ and $u(t)$ as decision variables, the solver chooses the state in a coherent way.

Multiple Shooting

- Discretize $u(t)$
- Discretize $x(t)$ only at few points in time
- Compute intermediate values of $x(t)$ by matter of integration

1.3.2 Single Shooting - Sequential Approach

In single shooting you discretize the control $u(t)$ on a fixed grid $0 = t_0 < t_1 < ... < t_N = t_f$

A very classical option for the discretization of the control is to do a piece-wise constant trajectory. So the control can only change at certain time and it remains constant.

$$u(t) = y_i \qquad \forall t \in [t_i, t_{i+1}]$$

You then proceed to compute $x(t)$ from $u(t)$ by integrating the dynamics

$$\begin{aligned} \dot{x} &= f(x, u, t) \\ x(0) &= x_0 \end{aligned}$$

Of course the control can be discretize in other ways, but in practice this is what happends most of the time (e.g. piece-wise polynomial).

The non linear program that you get by using single shooting is:

$$\underset{y}{\text{minimize}} \int_0^{t_f} l(x(t;y), u(t;y))\, dt \; + \; l_f(x(t_f;y))$$

subject to:

- $g(x(t_i;y), u(t_i;y), t_i) \leq 0 \qquad i = 0...N$ are the Discretized Path constraints

Discretized Path constraints

The Running cost integral tipically is computed when integrating dynamics, and you do it by using an augmented state

augmented state

$$c(t) = \int_0^t l(\cdot,\cdot)\, dt$$

$$\begin{cases} \dot{c}(t) = l(\cdot,\cdot) \\ c(0) = 0 \end{cases}$$

Given that we can define the augmented state $\bar{x}$ as:

$$\begin{aligned} \bar{x} &= (x, c) \\ \dot{\bar{x}} &= \begin{bmatrix} f(x,u,t) \\ l(x,u) \end{bmatrix} \end{aligned}$$

What we still need to do in order to implement a single shooting optimal control method are the computation of the sensitivities.

Computing the sensitivities

The sensitivities are the derivatives of the integration scheme that we are using to derive the state from the control.

sensitivities

Why do we need to differentiate between integration schemes? Because we are using an optimal optimization solver that requires the gradient of the cost function to work. So our cost function depends both on the state and the control, but actually the state depends itself on the control and this dependency is decided by the integration scheme that we are using.

So when we are computing the gradient of the cost function we need to take into account the gradient of the integration scheme, because to compute the cost function we need to evaluate the integration scheme.

This is not very difficult, it boils down to apply the chain rule in order to compute the derivatives of a function that it is itself a function of a variable, which is not the variable we want to differentiate, but an indirect dependency. (in broad terms: we want to differentiate the cost function which depends on the state which once again depends on u)

There is an efficient way to conduct this computation which is tipically used to implement single shooting methods.

Let us assume to have:

- Cost function and constraints of a single shooting NLP depend on x and u
- $x(t)$ us computed from $u(t)$ by matter of integration
- $u(t)$ is discretized in a piecewise matter or alternative way
  This means that given the generic dynamic equation:

$$\dot{x} = f(x(t), u(t), t)$$

  Assuming a piece wise constant discretization of the control $u(t)$ of the form:

$$u(t) = y_i \qquad \forall t \in [t_i, t_{i+1}]$$

  we can reformulate the differential equation with the notion of the control variable discretization:

$$\dot{x} = f(x(t), y_i, t) \qquad \forall t \in [t_i, t_{i+1}]$$

Then we have a cost function (which is a function of y)

$$c(y) = \int_0^T l(x(t), u(t))\, dt + l_f(x(T)) \approx \sum_{i=0}^{N-1} l(x_i, y_i) \cdot h + l_f(x_N)$$

where the running cost integral is approximated with the Euler Method instead of using the augmented state. Tipically we are not required to be accurate in the computation of the running cost, but we have to be accurate during the computation of the dynamics.

We obtained our cost that we would like to differentiate, i.e. we want to compute:

$$\frac{dc}{dy} = \sum_{i=0}^{N-1} \frac{dl(x_i, y_i)}{dy} \cdot h + \frac{dl_f(x_n)}{dy}$$

In order to do so we need to compute the two unique differentiation in the above formulation

- l depends on y directly (because is a function of y) but it also depends on y indirectly (because is a function of x that you obtain by integrating the dynamics using y as a control input).
  Therefore we need to take care of both dependencies

$$\frac{dl}{dy} = \frac{\partial l}{\partial x_i}\frac{dx_i}{dy} + \frac{\partial l}{\partial y}$$

- For the second term we only have the indirect dependency because the final cost has no direct dependency from the control.

$$\frac{dl_f}{dy} = \frac{\partial l_f}{\partial x_N}\frac{dx_N}{dy}$$

From an analysis of the two expressions we can conclude that the partial derivatives of l and l_f are usually easy to compute, because they are just the derivatives of these functions (which is the running cost, that we decide as a designer of the optimal control problem) and it is tipically a simple

function (e.g. quadratic function that penalizes the deviations of the state or the control from a given reference).

The total derivatives of the two expressions are less trivial to compute and they depend on the integration scheme. Their meaning is: how much a certain state is gonna change if I modify my control input (considering the whole trajectory of control inputs, not a single control input).

From this observations we can conclude that the partial derivatives are vectors whereas the total derivative are matrices.

We will now look into how to compute the $\frac{dx_i}{dy}$ terms.

Let us introduce the integrator function Φ, which is the discretized dynamics function, such that:

integrator function

discretized dynamics function

$$x_{i+1} = \Phi(x_i, y_i) \qquad \Rightarrow \qquad \frac{dx_{i+1}}{dy} = \frac{\partial \Phi_i}{\partial x_i}\frac{dx_i}{dy} + \frac{\partial \Phi_i}{\partial y}$$

This is basically what our integration scheme computes. We have derived a recoursive relationship that allows us to compute the term $\frac{dx_i}{dy}$ for time $i+1$ if we know the value of that term at the previous time step.

However, we need to know the other two derivatives of the integrator function that will depend on our choice of integration scheme.

Note that:

$$\frac{\partial \Phi}{\partial y} = \left(\frac{\partial \Phi_i}{\partial y_0}, ..., \frac{\partial \Phi_i}{\partial t_{N-1}} \right)$$

where

$$\frac{\partial \Phi_i}{\partial y_j} = 0 \ \forall i \neq j$$

Two questions remain unanswered:

1. How do we initialize the computation? (i.e. what is the value of $\frac{dx_0}{dy}$)

   We can initialize the computation with

   $$\frac{dx_0}{dy} = 0$$

   Because the input only affect the future not the present, the initial condition does not depend on the choice of control input.

2. How do we compute the other two terms in the above expression?

   This depends on the integration scheme that we are using. Let us review all the method that we have reviewed in Appendix**??**.

   - **Explicit Euler** Suppose Φ is explicit Euler (RK1), than:

     $$\Phi(x, u) = x + h\, f(x, u)$$

     where f is the continuous time dynamic of the system.

     $$\frac{\partial \Phi}{\partial x} = I + h\, \frac{\partial f}{\partial x}$$
     $$\frac{\partial \Phi}{\partial u} = h\, \frac{\partial f}{\partial u}$$

     Of course we need the derivatives of the continuous time dynamics.

- **RK4** Suppose that Φ is RK4 integrator and $f(\cdot)$ is time independent.

$$
\begin{aligned}
k_1 &= f(x_i, y_i) \\
k_2 &= f(x_i + \frac{1}{2}hk_1, y_i) \\
k_3 &= f(x_i + \frac{1}{2}hk_2, y_i) \\
k_4 &= f(x_i + hk_3, y_i) \\
x_{i+1} &= x_i + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \triangleq \Phi_i(x_i, y_i)
\end{aligned}
$$

The sensitivity w.r.t. x is:

$$
\frac{\partial \Phi_i}{\partial x_i} = I + \frac{h}{6}\left(\frac{\partial k_1}{\partial x_i} + 2\frac{\partial k_2}{\partial x_i} + 2\frac{\partial k_3}{\partial x_i} + \frac{\partial k_4}{\partial x_i}\right)
$$

where:

$$
\begin{aligned}
\frac{\partial k_1}{\partial x_i} &= \left.\frac{\partial f}{\partial x}\right|_{x_i} \\
\frac{\partial k_2}{\partial x_i} &= \left.\frac{\partial f}{\partial x}\right|_{x_{i2}} \left(I + \frac{1}{2}h\frac{\partial k_1}{\partial x_i}\right) && x_{i2} = x_i + \frac{1}{2}hk_1 \\
\frac{\partial k_3}{\partial x_i} &= \left.\frac{\partial f}{\partial x}\right|_{x_{i3}} \left(I + \frac{1}{2}h\frac{\partial k_2}{\partial x_i}\right) && x_{i3} = x_i + \frac{1}{2}hk_2 \\
\frac{\partial k_4}{\partial x_i} &= \left.\frac{\partial f}{\partial x}\right|_{x_{i4}} \left(I + \frac{1}{2}h\frac{\partial k_3}{\partial x_i}\right) && x_{i4} = x_i + \frac{1}{2}hk_3
\end{aligned}
$$

whereas the sensitivity w.r.t y is:

$$
\frac{\partial \Phi_i}{\partial y_i} = \frac{h}{6}\left(\frac{\partial k_1}{\partial y_i} + 2\frac{\partial k_2}{\partial y_i} + 2\frac{\partial k_3}{\partial y_i} + \frac{\partial k_4}{\partial y_i}\right)
$$

$$
\begin{aligned}
\frac{\partial k_1}{\partial y_i} &= \left.\frac{\partial f}{\partial y}\right|_{xi} \\
\frac{\partial k_2}{\partial y_i} &= \left.\frac{\partial f}{\partial y}\right|_{x_{i2}} + \left.\frac{\partial f}{\partial x}\right|_{x_{i2}} \left(\frac{1}{2}h\frac{\partial k_1}{\partial y_i}\right) \\
\frac{\partial k_3}{\partial y_i} &= \left.\frac{\partial f}{\partial y}\right|_{x_{i3}} + \left.\frac{\partial f}{\partial x}\right|_{x_{i3}} \left(\frac{1}{2}h\frac{\partial k_2}{\partial y_i}\right) \\
\frac{\partial k_4}{\partial y_i} &= \left.\frac{\partial f}{\partial y}\right|_{x_{i4}} + \left.\frac{\partial f}{\partial x}\right|_{x_{i4}} h\frac{\partial k_3}{\partial y_i}
\end{aligned}
$$

In summary:

- Remove x from problem variables
- Discretize $u(t) = y_i \qquad \forall t \in [t_i, t_{i+1}]$
- Compute x by integrating dynamics $\dot{x} = f(x, u)$
- Since x is computed by integrating the dynamics we do not need to have the dynamics as a constraint in the problem. i.e. remove dynamics from constraints of the problem
- Tipically in single shooting you use a High-order integration scheme because it is more efficient
- In any case, when you use an integration scheme you need to differentiate the integration scheme itself in order to compute the gradient of the cost and of the constraints. (Hardest part in the implementation)

1.3.3 Collocation - Simultaneous Approach

In collocation, we not only discretize the control input $u(t)$, but we also discretize the state $x(t)$ trajectory.

We are going to have many more decision variables in our problem, we are also gonna have many more constraints given that in this case we need to represent the dynamics of the system as a constraint in the problem. However, this is not necessarily slower to solve even though we have so many variables and constraints.

Formally:

- We discretize $x(t)$, tipically with polynomials (e.g. order k=0, piece-wise constant) on fine grid (i.e. the time between two successive steps is going to be small) which defines how accurate the representation of the dynamics is.

  $$x(t) = s_i \qquad \forall t \in [t_i, t_{i+1}] \qquad \text{One polynomial for each time interval}$$

  This was not the case for single shooting where you could take a large step size and use an high-order integration scheme to retrieve an accurate integration of the dynamics.
- Replace the dynamics $\dot{x} = f(x, u)$ with:

  $$\frac{s_{i+1} - s_i}{t_{i+1} - t_i} - f\left(\frac{s_{i+1} + s_i}{2}, y_i\right) = 0 \qquad\qquad i = 0, ..., N-1$$

  The first fraction is an approximation of $\dot{x}$, whereas the $\frac{s_{i+1} + s_i}{2}$ is an approximation of x. This formulation can be interpreted as a constraint that relates s_{i+1} with s_i and y_i, i.e.:

  $$c(s_{i+1}, s_i, y_i) = 0$$

- Approximate the cost integral:

  $$\int_{t_i}^{t_{i+1}} l(x(t), u(t))\, dt \approx l\left(\frac{s_i + s_{i+1}}{2}, y_i\right)(t_{i+1} - t_i) \triangleq l_i(s_i, s_{i+1}, y_i)$$

  It is not exactly an Euler integration scheme, but it is quite similar.

We can now write down the complete problem to see how it looks like.

$$\underset{s,y}{\text{minimize}} \sum_{i=0}^{N-1} l_i(s_i, s_{i+1}, y_i) + l_f(s_N)$$

subject to:

- $s_0 - x_0 = 0$
- $c_i(s_i, s_{i+1}, y_i) = 0 \qquad i = 0, ..., N-1$
- $g_i(s_i, y_i, t_i) \leq 0 \qquad i = 0, ..., N$

This problem has a very important feature which is sparsity, (e.g. a matrix is sparse if it has a lot of zeros inside), i.e. the gradient, the jacobian, the hessian of the problem (so the derivatives of cost function and constraints) are sparse matrices and vectors. sparsity

It is important because if you have a sparse problem and you use a solver that it is able to exploit the sparcity then it can be solved much more efficiently than if you just neglet the sparcity.

Why can we say that this problem formulation is sparse? What is the feature that gives away the sparsity? For instance from the constraint: $c_i(s_i, s_{i+1}, y_i) = 0$, we can deduct that the Jacobian will be sparse

The sparsity is given to this term by the dependency of some limited variables, in other terms the constraint does not depend on all the variables. In this case when we are going to compute the derivative of the constraint w.r.t. all the variables, many of them will be zero.

The same concept applies to the cost function (dependency on a subset of all the variables). For this term the hessian will be sparse (however the gradient will not be sparse).

1.3.4 Multiple shooting

Multiple shooting it is a middle ground between collocation and single shooting.

- Discretize the control $u(t)$ on a coarse grid $t_0 < t_1 < ... < t_n$ — coarse grid

$$u(t) = y_i \qquad \forall t \in [t_i, t_{i+1}]$$

- Integrate numerically ODE in each interval $[t_i, t_{i+1}]$, starting from a state variable s_i:

$$\begin{aligned} \dot{x}(t) &= f(x_i(t), y_i) \qquad\qquad t \in [t_i, t_{i+1}] \\ x_i(t_i) &= s_i \end{aligned}$$

- Obtain $x_i(t)$ and Numerically compute the integrals:

$$l_i(s_i, y_i) = \int_{t_i}^{t_{i+1}} l(x_i(t), y_i)\, dt$$

So in multiple shooting you have the control input discretized as in single shooting, but you also have some state variables and this is the main difference between the two approaches.

Contrary to Collocation where you have the state variable on a fine grid, here you have the state variables on a coarse grid, that guarantees very few state variables.

The remaining values of the states inbetween this state variables will be computed by matter of numerical integration.

What can happen in multiple shooting is that the result of the integration does not match with the value of the state variable that you have at the end, so extra contraints needs to be formulated to guarantee continuity of the state trajectory. The problem formulation then becomes:

$$\underset{s,y}{\text{minimize}} \sum_{i=0}^{N-1} l_i(s_i, y_i) \; + \; l_f(x_N)$$

subject to:

- $s_0 - x_0 = 0$
- $s_{i+1} - x_i(t_{i+1}, s_i, y_i) = 0$ where $x_i(\cdot)$ is the result of numerical integration starting from s_i. And these type of constraints are called continuity contraints. — continuity contraints
- $g(s_i, y_i) \leq 0$

During the optimization problem the solver is allowed to have partial solution that have discontinuities and then the gaps will be collapsed to zero little by little. At the end of the solution problem we will obtain a continuous state trajectory.

This may seem like a bad property of the problem, but actually is what allows the solver to find better solutions most of the time.

In practical cases, Collocation and Multiple Shooting can find better solutions than Single Shooting.

1.3.5 Comparison of Direct methods

1.4 Linear Quadratic Regulator (LQR)

Linear Quadratic Regulator LQR is a method to solve a very specific kind of optimal control — Linear Quadratic Regulator LQR

Single shooting	Multiple shooting	Collocation
⊕ Small problem	⊜ Medium problem	⊖ Large problem
⊕ Need only initial guess for $u(t)$	⊕ Sparser than single shooting	⊖ Cannot adapt time grid
⊕ Can use off-the shelp ODE solver	⊖ Less sparse than collocation	⊕ Sparse problem
⊖ Cannot exploit knowledge of x in initialization	⊕ Unstable Ok	⊕ Work well with unstable systems
⊖ ODE solution can depend very nonlinearly on y	⊕ Initialize $x(t)$	⊕ Can initialize x
⊖ Numerical issues for unstable systems		

problem. What is special about this optimal control problem is that is particularly simple because the dynamics is represented by a discrete-time linear system:

discrete-time linear system

$$x_{t+1} = A\,x_t + B\,u_t$$

And the objetive is to choose the control inputs $u_0, u_1, ...$ so that:

- x is "small" $\Rightarrow$ good regulation/performance
- u is "small" $\Rightarrow$ small effort/energy consumption

These are usually competing objectives (you cannot have both at the same time) because if you keep u very small than x does whatever it wants, and tipically if you want to keep x small you need some control effort.

So we need to find a trade-off between these two objectives, and that is what LQR does.

More precisely the LQR control problem takes the following form:

$$\underset{U}{\text{minimize}}\, J(U) = \sum_{i=0}^{N-1} = (x_i^T\,Q\,x_i + u_i^T\,R\,u_i) + x_N^T\,Q_f\,x_N$$

subject to:

- $x_{i+1} = A\,x_i + B\,u_i \qquad i = 0, ..., N-1$

With a quadratic cost function, where$Q$ defines the quadratic form of the state, R is the matrix that defines the quadratic form of the control and Q_f defines the quadratic form of the final cost.

All the quadratics form are strictly positive definites (strinctly positive eigenvalues) or semi-positive definites (eigenvalues can be zero), i.e.

$$Q = Q^T \geq 0 \qquad ; \qquad Q_f = Q_f^T \geq 0 \qquad ; \qquad R = R^T > 0$$

As a consequence, any time that you have a state or a control that is not zero than you are increasing the cost. So idealy you would like to have all the variables at zero to achieve zero cost which is the minimum of the cost function (however that is not going to be possible).

The two knobs to tune the problem to achieve the behaviour of the system that you want are Q and R, in particular:

- Q large $\Rightarrow$ x small is more important
- R large $\Rightarrow$ u small is more important

The LQR optimal control problem can be rewritten in the following form

$$\underset{X,U}{\text{minimize}}\, J(X,U) = \|\bar{Q}\,X\|^2 + \|\bar{R}\,U\|^2$$

subject to:

- $X = G\,U + H\,x_0$

- with block diagonal matrices:

$$\bar{Q} \triangleq diag(Q^{1/2}, ..., Q^{1/2}, Q_f^{1/2}) \qquad \text{and} \qquad \bar{R} \triangleq diag(R^{1/2}, ..., R^{1/2})$$

where we introduced the variables X and U containing the whole trajectory of the state and the control (discretized).

Also the dynamics can be written in linear form:

$$x_1 = A\,x_0 + B\,u_0$$
$$x_2 = A\,x_1 + B\,u_1$$

substituting the first expression into the second

$$= A^2\,x_0 + AB\,u_0 + B\,ux_3 \qquad\qquad = A^3\,x_0 + A^2Bu_0 + ABu_1 + B\,u_2$$

Iterating through each state we can represent the linear dynamics system in the matrix form:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ \vdots \\ \vdots \\ A^N \end{bmatrix} x_0 + \begin{bmatrix} B & & & \\ AB & \ddots & & \\ \vdots & \ddots & \ddots & \\ \vdots & & \ddots & B \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ \vdots \\ u_{N-1} \end{bmatrix}$$

So we can see that the state trajectory takes the following compact form:

$$X = H\,x_0 + G\,U$$

If we eliminate X from the problem variables given the linear dynamic constraints, obtaining the new problem without constraints:

$$\begin{aligned} \underset{U}{\text{minimize}}\ J(U) &= \|\bar{Q}(G\,U + H\,x_0)\|^2 + \|\bar{R}\,U\|^2 \\ &= \left\| \begin{bmatrix} \bar{Q}\,G \\ \bar{R} \end{bmatrix} U + \begin{bmatrix} \bar{Q}\,H\,x_0 \\ 0 \end{bmatrix} \right\|^2 \end{aligned}$$

Now we can clearly see that the LQR optimal control problem is nothing more that a big least-squares problem, with dimension $N(n+m) \times Nm$ and cost $O(N^3nm^2)$ using Quadratic regulator. `least-squares problem`

where:

- N is the length of the horizon (tipically quite large) `horizon`
- n is the size of the state
- m is the size of the control

As a consequence, LQR gives us a more efficient way of computing the solution of the least square program by exploiting the structure of the program.

1.4.1 LQR via Dynamic Programming

LQR can be derived from the principle of Dynamic Programming.

Dynamic Programming is in general very difficult to apply to a real problem because you need to solve a parametric optimization problem. And LQR is one of those rare cases in which you can actually use Dynamic Programming, because the parametric optimization problem is a quadratic program (can be solved analytically).

We therefore proceed to write down the LQR problem definining the value function (optimal cost starting at a certain state, at a certain time):

$$V_t(z) = \underset{u_t=(u_t,...u_{N-1})}{\text{minimize}} \sum_{k=1}^{N-1} (x_k^T\,Q\,x_k + u_k^T\,R\,u_k) + x_N^T\,Q_f\,x_N$$

subject to:

- $x_t = z$

- $x_{k+1} = A\,x_k \;+\; B\,u_k \qquad \forall k = t, ..., N-1$

Therefore the value function $V_t(z)$ represent the optimal cost starting in z at time t.

Of course the value function at time 0 starting at state x_0 (i.e. $V_0(x_0)$) is nothing more than the optimal cost of the original problem.

The objective of Dynamic Programming is recursively minimize the value function starting at the last time step N backwards. The reason why we do so is because for the last time step we already know the value function, i.e. the terminal cost of the LQR problem:

$$V_N(z) = z^T\,Q_f\,z$$

From this point we can leverage the Bellman optimality equation that defines the value function at time i as a function of the value function at time $i+1$ and apply this recursively going backward in time.

$$V_t(z) = \underset{w, U_{t+1}}{\text{minimize}}\; z^T Q z + w^T R w \;+\; \sum_{k=t+1}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T Q_f x_N$$

subject to:

- $x_{k+1} = Ax_k + Bu_k$
- $u_t = w$

Note that the value function above was split in the cost at time t (left) and the cost starting from the t going onwards until N.

The second part of the problem is nothing more than the value function at time $t+1$, i.e. $V_{t+1}(Az + Bw)$. So I can replace that part obtaining:

$$v_t(z) = \underset{w}{\text{minimize}}\; z^T Q z + w^T R w \;+\; V_{t+1}(Az + Bw)$$

NOTE: $Az + Bw$ is the next state i.e. x_{t+1}.

Now that we obtained the recursive optimality equation we can initialize it with the quadratic form for time N and compute the value function recursively backward in time to see if it mantains this quadratic form or if it takes a more complex form.

Assuming the v_{t+1} is quadratic (that is true for $t+1 = N$), we can define the following quantities:

$$V_{t+1}(z) = z^T P_{t+1} z \qquad\qquad P_{t+1} = P_{t+1}^T \geq 0 \qquad\qquad P_n = Q_f$$

And expand the minimization problem formulation with this new quantities

$$\begin{aligned}
V_t(z) &= \min_w (z^T Q z + w^T R w + V_{t+1}(Az + Bw)) \\
&= z^T Q z \;+\; \min_w (w^T R w \;+ (Az + Bw)^T P_{t+1} (Az + Bw)) \\
&= z^T Q z \;+\; \min_w (w^T (R + B^T P_{t+1} B) w + 2 z^T A^T P_{t+1} B w) + z^T A^T P_{t+1} A z \\
&= z^T (Q + A^T P_{t+1} A) z + \min_w (\,(w^T H w + 2 g^T w) \triangleq f(w)\,)
\end{aligned}$$

where H is the Hessian of the quadratic form and g is the gradient of the linear form.

Hessian of the quadratic form

gradient of the linear form

The minimization process is achieved by computing the gradient of the objective function to minimize (i.e. $f(w)$) and set it to zero:

$$\nabla f = 2Hw + 2g = 0$$

which has solution:

$$\begin{aligned}
w^* &= -H^{-1} g = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A z \\
&= K_t z
\end{aligned}$$

Note that the matrix H is always invertible by assumption: in fact, R and P_{t+1} are positive definite and their sum will always be a positive definite matrix, which is always invertible.

At this point we can compute $V_t(z)$ with the optimal value of the decision variable $w = w^*$.

$$\begin{aligned}
V_t(z) &= z^T (Q + A^T P_{t+1} A) z + g^T H^{-1} \cancel{H H^{-1}} g - 2 g^T H^{-1} g \\
&= z^T (Q + A^T P_{t+1} A) z - g^T H^{-1} g \\
&= z^T (Q + A^T P_{t+1} A - A^T P_{t+1} B H^{-1} B^T P_{t+1} A) z
\end{aligned}$$

The result is a purely quadratic function in z and the matrix inside is what we called P_t. By doing so we demonstrated that if the value function

$$V_{t+1}(z) = z^T P_{t+1} z$$

is purely quadratic at time $t+1$ than also the value function at time t is purely quadratic, i.e.

$$V_t(z) = z^T P_t z$$

Therefore the value function is quadratic for all t and we have a formulation of the value function at time t starting from the value function starting at time $t+1$.

We can also prove that if the value function is positive semi-definite at time $t+1$ then also the value function at time t will be positive semi-definite.

Proof. Given $P_{t+1} \geq 0, Q \geq 0, R > 0$ then $P_t \geq 0$, which translates into proving that :

$$P_t = Q + A\grave{i}TPA - A^T PB(R + B^T PB)^{-1} B^T PA \geq 0$$

$$\begin{aligned}
P - PB(R + B^T PB)^{-1} B^T P &\geq 0 \\
S(I - S^T B(R + B^T SS^T B)^{-1} B^T S)S^T &\geq 0 \\
I - S^T B \left(\begin{bmatrix} L & B^T S \end{bmatrix} \begin{bmatrix} L^T \\ S^T B \end{bmatrix} \right)^{-1} B^T S &\geq 0 \\
I - \begin{bmatrix} 0 & 1 \end{bmatrix} V^T (VV^T)^{-1} V \begin{bmatrix} 0 \\ 1 \end{bmatrix} &\geq 0 \\
I - \begin{bmatrix} 0 & 1 \end{bmatrix} V^\dagger V \begin{bmatrix} 0 \\ 1 \end{bmatrix} &\geq 0 \\
\begin{bmatrix} 0 & 1 \end{bmatrix} (I - V^\dagger V) \begin{bmatrix} 0 \\ 1 \end{bmatrix} &\geq 0 \\
I - V^\dagger V &\geq 0
\end{aligned}$$

$$\begin{aligned}
S &\triangleq P^{1/2} \Rightarrow P = SS^T \\
L &\triangleq R^{1/2} \Rightarrow R = LL^T \\
B^T S = \begin{bmatrix} L & B^T S \end{bmatrix} \begin{bmatrix} O_m \\ I_n \end{bmatrix} &= V \begin{bmatrix} O_m \\ I_n \end{bmatrix} \\
I &= \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\end{aligned}$$

Where the left hand side of the inequality is the Projection matrix (into nullspace of V) which eigenvalues are either 0 or 1 which implies that the inequality is always satisfied. □

Projection matrix

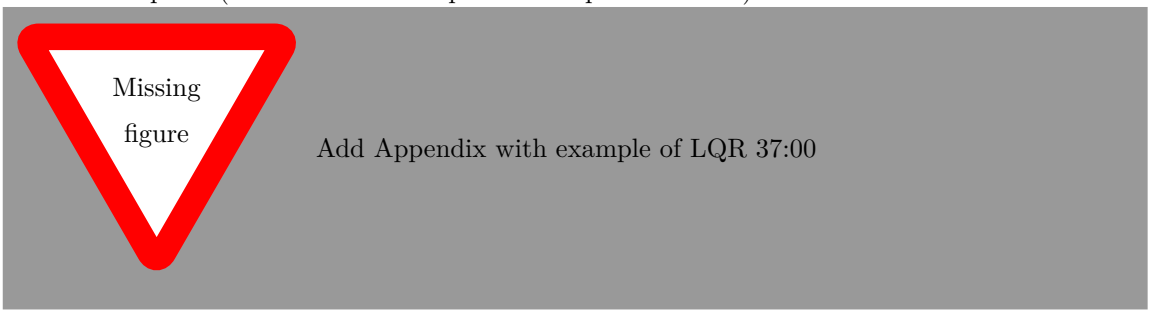
Summary of the DP approach to solve the LQR problem:

1. Set $P_N = Q_f$ initialization of value function at time N with the terminal cost.
2. For $t = N...1$, compute
$$P_{t-1} = Q + A^T P_t A - A^T P_t B(R + B^T P_t B)^{-1} B^T P_t A$$
3. For $t = 0...N-1$ compute the gain matrix
$$K_t = -(R + B^T P_{t+1} B)^{-1} B^T P_{t+1} A$$
4. For $t = 0...N-1$, compute the control input
$$u_t = K_t \, x_t$$

You may notice that for this specific problem we actually not only recover the trajectory of optimal control inputs but we also have an optimal control policy given that u_t is a linear function of the state. This gives us something more than an open-loop trajectory in contrast with the Direct methods (especially in Single Shooting).

The obtained optimal linear feedback policy can be used online on the real system: we can compute u based on the current state, which might be different from what we would expect from the nominal plant (more useful in the presence of perturbations).

optimal linear feedback policy



1.4.2 Steady State Regulator

Usually P_t converges as t decreases below N.

Steady-state value satisfies:

$$P_s = Q + A^T P_s A - A^T P_s B(R + B^T P_s B)^{-1} B^T P_s A$$

Discrete-time algebraic Riccati Equation (DT-ARE) Can be solved by direct method or iterating RIccati recursion.

Discrete-time algebraic Riccati Equation (DT-ARE)

Therefore, for t not close to N we have constant feedback gain:

$$u = -(R + B^T P_s B)^{-1} B^T P_s A$$

1.4.3 Time Varying System

LQE is easily extended to Time Varying systems:

$$x_{t+1} = A_t x_t + B_t u_t$$

and Time Varying cost matrices:

$$\sum_{t=0}^{N-1} (x_t^T Q_t x_t + u_t^T R_t u_t) + x_N^T Q_f x_N$$

In this case there need not to be a steady-state solution though.

1.4.4 Inhomogeneous system and costs

$$\text{minimize} \sum_{t=0}^{N-1} \begin{bmatrix} x_t^T & u_t^T & 1 \end{bmatrix} \begin{bmatrix} Q_t & S_t & q_t \\ S_t^T & R_t & s_t \\ q_t^T & s_t^T & 0 \end{bmatrix} \begin{bmatrix} x_t \\ u_t \\ 1 \end{bmatrix} + \begin{bmatrix} x_N^T & 1 \end{bmatrix} \begin{bmatrix} Q_f & q_N \\ q_N^T & 0 \end{bmatrix} \begin{bmatrix} x_N \\ 1 \end{bmatrix}$$

subject to:

- $x_{t+1} = A_t x_t + B_t u_t + c_t$
- $x_0 = x^{init}$

Define augmented state $\bar{x} = \begin{bmatrix} x & 1 \end{bmatrix}$

$$\begin{bmatrix} x_{t+1} \\ 1 \end{bmatrix} = \begin{bmatrix} A_t & c_t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ 1 \end{bmatrix} + \begin{bmatrix} B_t \\ 0 \end{bmatrix} u_t$$

$$\bar{x}_{t+1} = \bar{A}_t \bar{x}_t + \bar{B}_t u_t \quad \Leftarrow \text{Homogeneous}$$

$$J = \sum \begin{bmatrix} \bar{x}_t^T & u_t^T \end{bmatrix} \begin{bmatrix} \bar{Q}_t & \bar{S}_t \\ \bar{S}_t^T & R_t \end{bmatrix} \begin{bmatrix} \bar{x}_t \\ u_t \end{bmatrix} + \bar{x}_N^T \bar{Q}_f \bar{x}_N$$

with:

$$\bar{Q}_t \triangleq \begin{bmatrix} Q_t & q_t \\ q_t^T & 0 \end{bmatrix} \qquad \bar{S}_t \triangleq \begin{bmatrix} S_t \\ s_t^t \end{bmatrix}$$

The only difference is cross term $x^T S u$ in cost function. DP solution easily extended.

Fin doptimal fedback policy for augmented system:

$$u = \bar{K}\bar{x} = \begin{bmatrix} K & k \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} = Kx + k$$

Why inhomogeneous systems and costs:

- Tracking problems
- Linearization of nonlinear systems
- Filtering problems (e.g. Kalman)

1.4.5 Tracking problems

$$J = \sum_{t=0}^{N-1} (x_t - \bar{x}_t)^T Q (x_t - \bar{x}_t) + (u_t - \bar{u}_t)^T R (u_t - \bar{u}_t)$$

$\bar{x}, \bar{u}$ = reference trajectories to track.

Can be rewriteen as inhomogeneous cost:

inhomogeneous cost

$$\begin{aligned} J &= \sum x^T Q x + u^T R u - 2\bar{x}^T Q x - 2\bar{u}^T R u \\ &= \sum \begin{bmatrix} x^T & u^T & 1 \end{bmatrix} \begin{bmatrix} Q & 0 & Q\bar{x} \\ 0 & R & R\bar{u} \\ \bar{x}^T Q & \bar{u}^T R & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ 1 \end{bmatrix} \end{aligned}$$

1.4.6 Differential Dynamic Programming (DDP)

Differential Dynamic Programming (DDP) is in some sense an extension of LQR, that handles non-linear dynamical systems and also non-linear cost functions. This is a method that can be applied to a much more generic kind of optimal control problems, which takes the form:

Differential Dynamic Programming (DDP)

non-linear dynamical system

non-linear cost function

$$\text{minimize} \sum_{t=0}^{N-1} l_t(x_t, u_t) + l_N(x_N)$$

subject to:

- $x_{t+1} = f(x_t, u_t)$
- $x_0 = x^{init}$

It is not completely a generic control problem because we do not have inequality constraints or any other type of constraints beside the dynamics. But surely more generic than Direct Methods, given that we can handle non linearities.

The idea of DDP vaguely resemble the approach of Newton Method and Numerical Optimization methods.

Start with a guess for the decision variable of the problem U and alternate between the following three steps:

1. Linearize around current trajectory
2. Solve LQR to get variation of U
3. Line search to ensure convergence

This is just an euristic, there is no guarantee of convergence, sometimes it will converge to the global optimum and sometimes it will converge to a local optimum.

The idea it remains the same, i.e. apply dynamic programming, but since you cannot really apply DP to the original system, you apply it to a linearization (aka local approximation) of the system

$$V(z, i) = \underset{U_i}{\text{minimize}}\, J_i(z, U_i) \qquad \text{where:} \quad U_i = (u_i, ..., u_{N-1})$$

We can subsequently apply Bellman optimality principle in order to split the cost in two parts:

$$V(z, i) = \underset{w}{\text{minimize}} \left[l_i(z, w) + V(f(z, w), i + 1) \right] = \min_w Q(z, w)$$

$$Q(\bar{x} + z, \bar{u} + w) \approx Q(\bar{x}, \bar{u}) + \begin{bmatrix} Q_x^T & Q_u^T \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix} + \frac{1}{2} \begin{bmatrix} z^T & w^T \end{bmatrix} \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{xu}^T & Q_{uu} \end{bmatrix} \begin{bmatrix} z \\ w \end{bmatrix}$$

Since we do not really know how to minimize a non linear function we can take a quadratic approximation (via Taylor expansion) and minimize it.

where:

$$Q_x \triangleq \frac{\partial Q}{\partial x} \qquad Q_u \triangleq \frac{\partial Q}{\partial u} \qquad Q_{xx} \triangleq \frac{\partial^2 Q}{\partial x^2} \qquad Q_{uu} \triangleq \frac{\partial^2 Q}{\partial u^2} \qquad Q_{xu} \triangleq \frac{\partial Q_u}{\partial x}$$

Once defined the derivative of the value function as follows:

$$V' \triangleq V(\cdot, i+1)$$

We can expand the partial derivatives, using the chain rule:

$$\begin{aligned}
Q_x &= l_x + f_x^T V'_x \\
Q_u &= l_u + f_u^T V'_x \\
Q_{xx} &= l_{xx} + f_x^T V'_{xx} f_x + V'_x f_{xx} \\
Q_{uu} &= l_{uu} + f_u^T V'_{xx} f_u + V'_x f_{uu} \\
Q_{xu} &= l_{xu} + f_x^T V'_{xx} f_u + V'_x f_{xu}
\end{aligned}$$

Moreover the quadratic cost function takes the following compact form:

$$\begin{aligned}
Q(\bar{x}, \bar{u}) &= l(\bar{x}, \bar{u}) + V'(f(\bar{x}, \bar{u})) \\
\bar{Q} &= \bar{l} + \bar{V}'
\end{aligned}$$

The DDP problem is reduce to minimizing the local quadratic model of Q. In doing so we get:

$$w^* = \underset{w}{\operatorname{argmin}}\, Q(z, w) = -Q_{uu}^{-}1(Q_u + Q_{ux}\, z)$$

This expression is a bit more complicated than what we had before with LQR because it is not a purely linear function of the state, but it is an affine function of the state (Bias/constant term Q_u and linear term $Q_{ux}z$). The reason why it is not purely linear is because the cost is not purely quadratic but you have also the affine term (i.e. Linear terms).

Substituting w^* inside the definition of the local approximation of the Q function Q to recover the value function at the previous time step V, i.e.:

$$\begin{aligned}
V &= \bar{Q} + Q_x^T z + Q_u^T w^* + \frac{1}{2} z^T Q_{xx} z + \frac{1}{2} {w^*}^T Q_{uu} w^* + z^T Q_{xu} w^* \\
&= \bar{Q} + (Q_x^T - Q_u^T Q_{uu}^{-1} Q_{ux}) z + \frac{1}{2} z^T (Q_{xx} + \cancel{Q_{xu} Q_{uu}^{-1} Q_{ux}} - 2 Q_{xu} Q_{uu}^{-1} Q_{ux}) z + \\
&\quad \cancel{Q_u^T Q_{uu}^{-1} Q_{ux} z} - \cancel{z^T Q_{xu} Q_{uu}^{-1} Q_u} + \dots \\
&= \Delta V + V_x z + \frac{1}{2} z^T V_{xx} z
\end{aligned}$$

where:

- $V_x \triangleq Q_x - Q_{xu} Q_{uu}^{-1} Q_u$
- $V_{xx} \triangleq Q_{xx} - Q_{xu} Q_{uu}^{-1} Q_{ux}$
- $\Delta V = \bar{Q} - \frac{1}{2} Q_u^T Q_{uu}^{-1} Q_u$

Once again what we just observed is that the value function starting from a quadratic approximation of the value function, if we propagate it backward in time using Bellman optimality equation, we still get a quadratic approximation of the value function for time $t-1$.

By applying this principle iteratively we always maintain a quadratic form of the value function.

There is one small detail: when we are working with a non linear system (that you do not need to worry about in the LQR problem given the initial assumption). The matrix Q_{uu}, which we need to invert in order to compute the optimal control inputs, might not be invertible.

The way this is tipically handled is to introduce regularization: regularization

$$\bar{Q}_{uu} = Q_{uu} + \mu\, I$$

where μ is the regularization value. If this value is sufficiently large that you can be sure that the new regularized matrix will always be invertible. regularization value

You can then replace the inverse of Q_{uu} with the regularized version:

$$w^* = -\bar{Q}_{uu}^{-1}(Q_u + Q_{ux} z) = \bar{w} + Kz \qquad \bar{w} \triangleq -\bar{Q}_{uu}^{-1} Q_u \qquad K \triangleq -\bar{Q}_{uu}^{-1} Q_{ux}$$

Substituting w^* inside Q to get the value function, you get:

$$\begin{aligned}
V &= \bar{Q} + Q_x^T z + Q_u^T w^* + \frac{1}{2} z^T Q_{xx} z + \frac{1}{2} w^{*T} Q_{uu} w^* + z^T Q_{xu} w^* \\
&= \bar{Q} + Q_u^T \bar{w} + (Q_x^T + Q_u^T K) z + \frac{1}{2} z^T (Q_{xx} + K^T Q_{uu} k + 2 Q_{xu} K) z + \bar{w}^T Q_{uu} K z \\
&\quad + \frac{1}{2} \bar{w}^T Q_{uu} \bar{w} + z^T Q_{xu} \bar{w} \\
&= \quad \bar{Q} + Q_u^T \bar{w} + \frac{1}{2} \bar{w}^T Q_{uu} \bar{w} && \Leftarrow \Delta V \\
&\quad + (Q_x^T + Q_u^T K + \bar{w}^T Q_{uu} K + \bar{w}^T Q_{xu}^T) z && \Leftarrow V_x \\
&\quad + \frac{1}{2} z^T (Q_{xx} + K^T Q_{uu} K + 2 Q_{xu} K) z && \Leftarrow V_{xx}
\end{aligned}$$

Summary of the DDP algorithm:

1. Given an initial guess for the optimal control trajectory $\bar{U}$, compute the state trajectory $\bar{X}$ by matter of forward simulation in time $\dot{x} = f(x, u)$
2. Backward Pass (it is called this way because we have a loop backward in time): `Backward Pass`
   (a) Initialize the derivatives of the value function with the derivatives of the terminal cost, ie.e
   $$V_x(N) = \nabla_x l_N \qquad V_{xx}(N) = \nabla_{xx} l_N$$
   (b) Loop backward in time $(i = N-1, ..., 0)$
   - Compute the derivatives of the Q function, i.e. $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{xu}$
   - Compute the optimal control input
   $$w^* = -\bar{Q}_{uu}^{-1}(Q_u + Q_{ux} z) = \bar{w} + K z$$
   - Compute the value function at time i, i.e. $V_x(i), V_{xx}(i)$
3. Forward Pass, since you do not the state yet, we cannot compute the control input. Basically we need to do a line search. `Forward Pass`
   (a) set $\alpha = 1$. Which translate into taking a full step.
   (b) Simulate with
   $$\begin{aligned} u &= \bar{u} + w \\ &= \bar{u} + \alpha\, \bar{w} + K(z) \\ &= \bar{u} + \alpha\, \bar{w} + K(x - \bar{x}) \end{aligned}$$
   (c) If cost has not decreased enough, you decrease α and repeat the previous step
   (d) Assign $\bar{x} = x$ and $\bar{u} = u$
   (e) If the forward pass has not converged repeat the Backward Pass, around the new trajectory $\bar{U}$.

In the end you obtain:

1. Optimal open-loop control $\bar{u}$
2. Locally optimal linear feedback gains K

Which can be summarized in the feedback control policy (locally optimal)

$$u = \bar{u} + K(x - \bar{x})$$

How much should the cost decrease?

It depends on α, because if α is very small we cannot expect a big improvement in the cost because we are changing the control input by a small amount, whereas if α is large we can expect a larger improvement. The idea is the same that we presented with Newton Method in line search, i.e. you compute how much you expect the cost to improve if your approximation of the function that you are minimzing is exact. And this will give you a baseline to compare against how much the cost is actually decreasing.

To compute how much the cost should decrease you need to take a look at the variation of the value function starting from the value function at time 0 for $z = 0$:

$$\begin{aligned} V_0(0) = \Delta V_0 &= \alpha \sum_{i=0}^{N-1} \bar{w}_i^T Q_{u,i} + \frac{\alpha^2}{2} \sum_{i=0}^{N-1} \bar{w}_i^T Q_{uu,i} \bar{w}_i + \sum_{i=0}^{N} \bar{l}_i \\ &= \alpha \sum_{i=0}^{N-1} \bar{w}_i^T Q_{u,i} + \frac{\alpha^2}{2} \sum_{i=0}^{N-1} \bar{w}_i^T Q_{uu,i} \bar{w}_i + J(\bar{U}) \\ \Delta V_i &= \alpha \bar{w}_i^T Q_{u,i} + \frac{\alpha^2}{2} \bar{w}_i^T Q_{uu,i} \bar{w}_i + \Delta V_{i+1} + \bar{l}_i \end{aligned}$$

Here you can see that α appears both linearly and quadratically.

Hence the expected cost improvement takes the form:

$$\Delta J(\alpha) = \Delta V_0 - J(\bar{U}) = \alpha d_1 + \frac{\alpha^2}{2} d_2$$

That means that every time I change α in the forward pass I do not need to recompute d_1 and d_2, but I only need to recompute $\Delta J(\alpha)$.

What is tipically done is:

$$\frac{J(U) - J(\bar{U})}{\Delta J(\alpha)} > c_1$$

You compute the real improvement (numerator) and the expected improvement (denominator) and finally accept this step if the ratio between the two is greater than a certain value (smaller than 1).

The parameter c_1 is an hyperparameter of the problem that you have to tune by hand. `hyperparameter`

Regularization

How do we tune the regularization value μ.

In fact if this parameter is too small it will happen that:

- $\bar{Q}_{uu}$ may be singular
- $\bar{w}$ may be too large, which results in very large control inputs. In this situation the local quadratic model may no longer be accurate with resulting small cost improvements

On the other hand if this parameter is too large then $\bar{w}$ may be too small and the convergence will be too slow.

Therefore the Regularization Scheme that we are going to follow is: `Regularization Scheme`

- If $\bar{Q}_{uu}$ is not invertible or $\dfrac{J(\bar{U}) - J(U)}{\Delta J(\alpha)} < c_2$, then we will increse the regularization factor
- otherwise we will decrease the regularization factor, in order to speed up the convergence.

In Summary:

- DP gives us an efficient algorithm to solve LQR
- The solution of LQR yields a globally optimal feedback policy `globally optimal feedback policy`
- Extension to non linear systems (DDP)
  - efficient recursive algorithm
  - no guarantees of convergence
  - can only find local optimum
  - feedforward + feedback policy
  - many hyper-parameters for regularization/line search c_1, c_2, μ, k_μ, K

1.5 Model Predictive Control (MPC)

Model Predictive Control (MPC) is a very popular technique used in many robotic systems and other fields outside robotics. Model Predictive Control (MPC)

Moreover, this particular control method connects what we have seen in reactive control and optimal control, because what we have seen so far in optimal control is just a mean of computing a trajectory and then using reactive control we can track the reference trajectory.

With MPC we are unifying these 2 problems and solutions, because the idea of MPC is to use optimal control not only for computing the reference trajectory, but also to design the real time control loop (aka stabilizing the system).

We took a glimps of this type of approach in DDP, because DDP does not only yield the control trajectory but also the feedback gains: this allows us to the result of DDP as a feedback controller. However this is really specific to DDP since it is the only method that gives us the feedback gains (i.e. something to stabilize the systems).

All the other methods, such as Direct methods, only yield a control trajectory not the feedback gains, so in order for them to work they need to be coupled with a reactive controller.

Why do we need something else to do the stabilization? Mainly because we will always have uncertainty in the system, so even if you plan for an optimal trajectory, you cannot hope that you apply this control and the system would behave exactly as planned, you always need a feedback to stabilize the system.

In MPC we are trying to get rid completely of the reactive controller and use the optimal controller also to stabilize the system. The principle behind this method goes as follow:

Solve a finite-horizon OCP using the current state (that you measure or estimate) as initial state. finite-horizon

Under this condition the optimal control problem takes the form:

$$X^*, U^* = \underset{X,U}{\operatorname{argmin}} \sum_{k=0}^{N-1} l(x_k, u_k)$$

subject to

- $x_{k+1} = f(x_k, u_k, k) \qquad k = 0, ..., N-1$
- $x_{k+1} \in \mathfrak{X}, \quad u_k \in \mathfrak{U} \qquad k = 0, ..., N-1$
- $x_0 = x^{meas}$

The process to follow is:

1. Solve the optimal control problem
2. Obtain the optimal state trajectory and optimal control trajectory
3. Take the first value of the optimal control trajectory u_0, and apply it
4. In the next iteration of the control loop, repeat the same process we the value of the optimal control input obtained in the previous step.

   Of course you consider the initial state as the state measured or estimated in the current iteration.

By solving the optimal control problem at each iteration you are always behaving in a optimal way because you are re-optimizing the trajectory based on the real state of the system not based on where you thought your system is according to your model.

The main problem of MPC comes from the difference between the trajectory that you predict whe you solve the OCP and the trajectory that you actually going to follow with your real system (this point does not take into account the presence of disturbances). Even if you do not have any disturbance, the trajecory that you are going to predict with the optimal control problem and the one that you are going to get with the MPC are going to be different.

The reason is because we are re-optimizing the trajectory at each step with an unseen point w.r.t. the previous optimization.

In other terms

- the OCP at time k and $k+1$ optimize over different horizons so it can result in different trajectories.
- Even without disturbances, predicted and actual trajectories can be different
- What if at time $k+1$ I shorten the horizon to $N-1$?
  In this case the trajectory would be the same due to Bellman optimality principle.

The three main challenges when dealing with MPC are:

1. feasibility. — `feasibility`

   Can I ensure that the OCP is always feasible?

   What can I do if it is not feasible?

   This is a problem given that if the problem is not feasible than the solver will not be able to yield any solution and so you lack a control input to apply to the system.

2. stability. — `stability`

   Can I ensure that MPC stabilizes the system?

   Because even if you are computing the trajectory that all converge to zero, maybe the real trajectory that system is performing is diverging.

3. computation time — `computation time`

   Can I solve the OCP sufficiently fast?

1.5.1 Infinite-Horizon MPC

What would happen if instead of having a finite horizon MPC, we had an horizon that is infinite?

If the horizon is infinite than the horizon that is seen by each OCP that you solve is the same since they all go to infinity. So you no longer have the problem that we have discussed before (i.e. the trajectory that you get when optimizing is different from the trajectory that you actually get on the real system), because at each iteration the OCP uses the same horizon and the trajectory will be the same at each optimization.

In short: in this case predicted and actual trajectories are the same, assuming no distrurbances. This follows from Bellman's principle of optimality.

Having $N = \infty$, if cost $l(x, u) \geq \alpha \|x\| \ \forall x, u$ for some $\alpha > 0$ then

1. Having a finite cost implies stability. From the condition of the cost we can infer that the the cost can only be finite only if the state (or its norm) goes to zero.

2. Since predicted and actual trajectories are equal, I have recursive feasibility along the closed-loop trajectory. — `recursive feasibility`

Even though this scenario is purely theoretical (in practice we cannot impose an infinite horizon), it helps us get the key idea of stability and recursive feasibility of MPC which is: try to mimic an infinite horizon problem by including a terminal cost and terminal constraint that mimic the effect that I would get with the infinite horizon (much like what we did with DP with the value function). — `terminal cost`

In other terms the terminal cost should be an approximation of the tail of my infinite horizon running cost and the terminal constraints should be an approximation of all the extra constraints I would get with an infinte horizon. — `terminal constraint` `infinite horizon running cost`

If we take the standard finite horizon OCP:

$$V_N^*(\bar{x}) = \underset{U}{\text{minimize}} \sum_{k=0}^{N-1} l(x_k, u_k)$$

subject to:

- $x_{k+1} = f(x_k, u_k) \qquad k = 0, ..., N-1$
- $x_k \in \mathfrak{X}, \quad u_k \in \mathfrak{U} \qquad k = 0, ..., N-1$
- $x_0 = \bar{x}$

In order to ensure fisibility and stability we can extend the formulation using a terminal cost and terminal constraints:

$$V_N^*(\bar{x}) = \underset{U}{\text{minimize}} \sum_{k=0}^{N-1} l(x_k, u_k) + l_f(x_N)$$

subject to:

- $x_{k+1} = f(x_k, u_k) \qquad k = 0, ..., N-1$
- $x_k \in \mathfrak{X}, \quad u_k \in \mathfrak{U} \qquad k = 0, ..., N-1$
- $x_0 = \bar{x}$
- $x_N \in \mathfrak{X}_\mathfrak{f}$

The question remains: how can we choose $l_f(\cdot)$ and $\mathfrak{X}$?

1.5.2 Feasibility

When talking about feasibility we need to distinguish between two cases, because they are very different from each other:

1. Input constraints only

   This case is always feasible, because u is our decision variable so if your control set is not empty, you can always choose a point inside this set and always obtain a solution.

2. Hard state constraints

   If $N < \infty$ there is no guarantee that OCP remains feasible, even in nominal case.

   $N = \infty$ ensures feasibility, but OCP has infinitely many constraints.

Input constraints only

Hard state constraints

The Maximum Output Admissible Set theory states that $N < \infty$ is enough to enforce recursive feasibility, but it does not specify how long the horizon should be in order that this is guaranteed.

A better way to deal with this problem is to use the Maximum Control Invariant Set which is a set that can be used as a terminal constraint and this ensures closed-loop convergence, but it can reduce the domain of feasibility, i.e. starting from a state for your original problem would be feasible, by using the Maximum Control Invariant Set as a constraint, your problem might become not feasible (especially if the system is perturbed).

In practice, this is almost the best thing that you could do (if you can do it).

Maximum Output Admissible Set theory

Maximum Control Invariant Set

closed-loop convergence

domain of feasibility

Control Invariant Set

A set S is Control Invariant if once you are inside the set you can stay inside the set.

In mathematical term: A set S is control invariant if $\forall x \in S, \exists u \in \mathfrak{U}$ s.t. the next state $f(x, u) \in S$.

Control Invariant

Theorem 1. *If $\mathfrak{X}_f$ is control invariant then MPC will be recursively feasible.*

Visual proof:

Example: For a manipulator, set of zero velocity is control invariant (with zero velocity the manipulator remains in the same position with zero velocity i.e. you remain in the same state) if

$$|g(q)| \leq \tau^{max} \qquad \forall q \in [q^{min}, q^{max}]$$
$$S = \{(q, 0) | q^{min} \leq q \leq q^{max}, q \in \mathbb{R}^n\}$$

Which means that if the motor are sufficiently strong to compensate for gravity, then the set of possible states where the velocity is zero S is control invariant.

How to compute the control invariant sets in general?

- Hard problem for nonlinear systems (e.g. robot manipulator)
- "Maximal Output Admissible Set" theory for linear systems. The same theory gives you some numerical algorithm to compute control invariant sets, but they only apply to linear systems. Let us suppose that we have a system with linear dynamics:

  $$x^+ = Ax \qquad y = Cx$$

  And we have some constraints on the output (which are allowed to be nonlinear, and they can include state constraints and input constraints)

  $$y \in \mathfrak{Y} = \{y \in \mathbb{R}^p : h_i(y) \leq 0, i = 1, ..., s\}$$

  The objective of the algorithm is to find a specific control invariant set, that is the maximum output admissible set:

  $$O_\infty = \{x \in \mathbb{R}^n : h_i(CA^t x) \leq 0, i = 1, ..., s, t = 0, ..., \infty\}$$

  which is the set of all initial states x, s.t. if you start in x the constraints on the output are always satisfied.

Maximal Output Admissible Sets

Theorem 2. *If A is Lyapunov stable, h_i are continuous and they contain the origin (i.e. $h_i(0) \leq 0$) then:* `origin`

$$O_\infty = O_{t^*} = \{x \in \mathbb{R}^n : h_i(CA^t x) \leq 0, i = 1, ..., s, t \in \{0, ..., t^*\}\}$$

With the only difference that we can compute the set with a finite number of computation instead of infinity.

One possible way to compute t^*, even if it is not the best way, is to start with $t = 0$ and compute O_t until $O_{t+1} = O_t$.

This is theoretically fine but actually comparing O_{t+1} to O_t could be quite hard, so there is another way to do it:

1. Sove this problem for $i = 1, ..., s$:

   $$\max_x J_i(x) = h_i(CA^{t+1}x)$$
   $$\text{s.t. } h_j(CA^k x) \leq 0 \qquad j = 1, ..., s, k = 0, ..., t$$

2. If $J_i^* < 0 \quad \forall i = 1, ..., s$ then assign $t^* = t$.

   Otherwise increment t and repeat.

What you are trying to do is similar to what we were trying to do with the first method, but the computation that you need to at each iteration is just s-optimization problems, where s is the number of inequalities that define your constraints set, and what you are trying to do is to see if it is possible to violate one of the constraints at the time step $t+1$ assuming that you satisfied the constraints at all the previous time steps.

If it is possible to do so, then you need to increase t and keep going, whenever it is not possible to do so, it means that all constraints are satisfied and at that point you obtain t^* and the algorithm stops.

What about control inputs?

Fix $u = Kx$, which implies $x^+ = (A + BK)x$. Ultimately obtaining :

$$y = \begin{bmatrix} I \\ K \end{bmatrix} x \qquad \mathfrak{Y} = \mathfrak{X} \times \mathfrak{U}$$

Theorem 3. *If $\mathfrak{X}_f$ is control invariant then the MPC is persistently feasible*

persistently feasible

Proof plan:

1. Given X_f compute backward reachable sets X_i, i.e. set from which X_f can be reached
2. Show that if X_f is control invariant, then X_i is control invariant $\forall i < N$
3. Show that if X_1 is control invariant, then the MPC is recursively feasible

Proof. 1. Given $x_n \in \mathfrak{X}_f = \mathfrak{X}_n$ define backward reachable sets recursively:

backward reachable sets

$$\mathfrak{X}_i = \{x \in \mathfrak{X} | \exists u \in \mathfrak{U}, \text{s.t. } f(x,u) \in \mathfrak{X}_{i+1}\}$$

By definition, if $x_N \in \mathfrak{X}_N$ then $x_i \in \mathfrak{X}_i \qquad \forall i \in [0, N-1]$

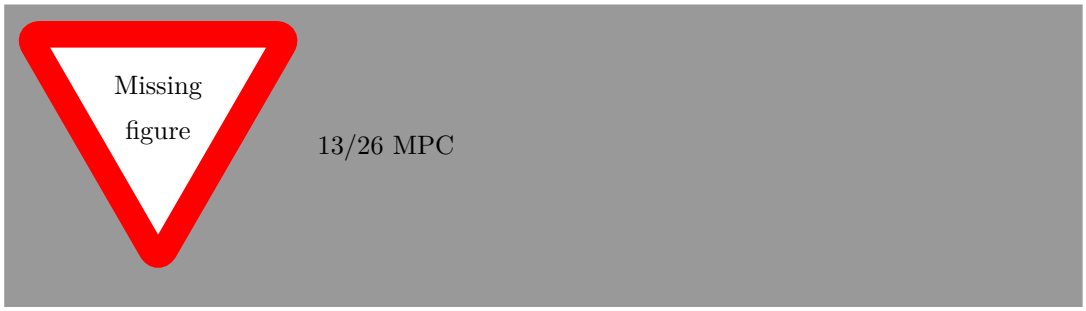
2. If $X_f = \mathfrak{X}_N$ is control invariant, then $\mathfrak{X}_i$ is control invariant $\forall i$.

   Assume $\mathfrak{X}_{i+1}$ is control invariant. This is true if and only if $\forall x \in \mathfrak{X}_{i+1}$, which implies that $\exists u \text{s.t. } f(x,u) \in \mathfrak{X}_{i+1}$.

   So from $\mathfrak{X}_{i+1}$ you can stay in $\mathfrak{X}_{i+1}$.

   We know that $\mathfrak{X}_i$ contains all states from which you can reach $\mathfrak{X}_{i+1}$ so we infer:

$$\mathfrak{X}_{i+1} \subseteq \mathfrak{X}_i$$



   Since from $\mathfrak{X}_i$ you can reach $\mathfrak{X}_{i+1}$, which is contained in $\mathfrak{X}_i$, we infer that from $\mathfrak{X}_i$ you can stay in $\mathfrak{X}_i$ which implies that $\mathfrak{X}_i$ is control invariant.

   By induction, starting from $\mathfrak{X}_N$ we can show that $\mathfrak{X}_i$ is control invariant $\forall i < N$.

3. We can show that

$$\mathfrak{X}_i \subseteq \mathfrak{X}_0$$

   Since the next state $x_1 \in \mathfrak{X}_1$ then it is true that $x_1 \in \mathfrak{X}_0$. However this is true if and only if the next MPC problem is feasible because $\mathfrak{X}_0$ is the set of all states from which MPC is feasible (by definition of $\mathfrak{X}_0$)

□

1.5.3 Stability

We can utilize Lyapunov Stability theorems to verify stability conditions of the MPC. Under this conditions, the key idea is to use the Value function (aka Optimal cost-to-go) as the exponential Lyapunov function.

Value function

The reason why it makes sense to use the Value function as a Lyapunov function is displayed in the following figure:

exponential Lyapunov function

Along the time horizon over which we are optimizing and associated to each time step we have a cost which depends on the state and control at that time step.

So when we are at $t = 0$ the horizon that we see, goes until time step N, whereas at $t = 1$ the horizon that we see goes until time step $N + 1$.

Imagine the cost that we will get in each of these two control problems that we solve:

- When we solve the first optimal control problem the cost will be:

$$V_0^*(x_0) = \sum_{i=0}^{N-1} l(x_i, u_i) + l_f(x_N)$$

  with a summation of the running costs and the terminal cost
- The same applies to the second optimal control problem starting at $t = 1$:

$$V_1(x_1) = \sum_{i=1}^{N} l(x_i, u_i) + l_f(x_{N+1})$$

  With the only difference being on the limit of the summation and the argument of the terminal cost.

We can notice that the intermediate running costs in the range $t \in [1, N-1]$ are equal: the only difference is in the terms:

$$l(x_0, u_0) \qquad l_f(x_N) \qquad l(x_N, u_N) \qquad l_f(x_{N+1})$$

Hence, we can write down the difference between the cost V_1 and the cost V_0^* (the first two conditions of the Lyapunov function are easy to fullfill):

$$\begin{aligned} V_1(x_1) - V_0^*(x_0) &= \sum_{i=1}^{N} l(x_i, u_i) + l_f(x_{N+1}) - \sum_{i=0}^{N-1} l(x_i, u_i) - l_f(x_N) \\ &= \cancel{\sum_{i=1}^{N-1} l(x_i, u_i)} + l(x_N, u_N) + l_f(x_{N+1}) - l(x_0, u_0) - \cancel{\sum_{i=1}^{N-1} l(x_i, u_i)} - l_f(x_N) \\ &= l(x_N, u_N) + l_f(x_{N+1}) - l_f(x_N) - l(x_0, u_0) \end{aligned}$$

if i am able to prove that:

$$V_1(x_1) - V_0^*(x_0) = l(x_N, u_N) + l_f(x_{N+1}) - l_f(x_N) - l(x_0, u_0) \leq -\alpha_3 \|x_0\| \qquad \forall x_N \in \mathfrak{X}_f$$

then the value function is an exponential Lyapunov function and I can use it to prove the stability of the origin for my system when it is controlled by the MPC.

One thing to notice is that V_0^* is the optimal value function (i.e. the states and controls are optimal) , whereas forV_1 we assumed that we have used the values of the optimal state and control until x_N and u_N computed at the previous time step.

optimal value function

However, they may not be the optimal ones given that we are optimizing on a different horizon.

So by considering a non optimal value function we are taking a conservative approach: we are assuming that the trajectory will stay the same until the second to last time step.

This means that the condition that I want is a sufficient condition, but it is not necessary. So if it is satisfied I am sure that the system controlled by the MPC is stable.

sufficient condition

Stability Assumption

Ass. #1: The origin is an equilibrium point, the runnning cost at the origin is 0, the terminal cost at the origin is 0.

$$f(0,0) = 0 \qquad l(0,0) = 0 \qquad l_f(0) = 0$$

Note that assuming $f(0,0)$ is not restictive.

Proof. If we want to stabilize $x^* \neq 0$ s.t. $f(x^*, u^*) = x^*$, i.e. x^* must be an equilibrium.

To achieve so we can define a new state $\bar{x}$, a new control $\bar{u}$ and new dynamics as follows:

$$\begin{aligned} \bar{x} &= x - x^* \\ \bar{u} &= u - u^* \\ \begin{cases} x^+ = f(x,u) \\ x^* = f(x^*,u^*) \end{cases} &\Rightarrow \qquad x^+ - x^* = f(\bar{x} + x^*, \bar{u} + u^*) - x^* \\ \bar{x}^+ &= \bar{f}(\bar{x}, \bar{u}) \end{aligned}$$

So that $\bar{f}(0,0) = f(0 + x^*, 0 + u^*) - x^* = 0$ □

Ass. #2: The union of the state space and control space is a closed set (set that contains the boundaries):

closed set

$$\mathfrak{X} \times \mathfrak{U} \text{ is closed}$$

The terminal set is a compact set (≈ closed + bounded set)

compact set

$$\mathfrak{X}_f \text{ is compact}$$

Ass. #3: For every state in the terminal constraint set there must exists a value of the control a collection of conditions must be respected.

Formally:

$$\forall x \in \mathfrak{X}_f \ \exists u \in \mathfrak{U} \text{ s.t.}$$

(a) $\mathfrak{X}_f$ is control invariant

$$f(x,u) \in \mathfrak{X}_f$$

and the terminal cost decreases

$$l_f(f(x,u)) - l_f(x) \leq -l(x,u)$$

(b) There must exists 2 positive scalar, such that the running cost is lower bounded and the terminal cost is upper bounded, i.e. $\exists \alpha_1, \alpha_f > 0$ s.t.

$$\begin{aligned} l(x,u) &\geq \alpha_1 \|x\| & \forall x \in \mathfrak{X}_N, \quad \forall u \in \mathfrak{U} \\ l_f(x) &\leq \alpha_f \|x\| & \forall x \in \mathfrak{X} \end{aligned}$$

where $\mathfrak{X}_N$ is the set of states from which OCP has a solution

If these three assumptions are satisfied then $V_N^*(\cdot)$ is an exponetial Lyapunov function and therefore the origin is exponentially stable.

Proof.

$$V_1(x_1) - V_0^*(x_0) = l(x_N, u_N) + l_f(x_{N+1}) - l_f(x_N) - l(x_0, u_0) \leq -\alpha_3 \|x_0\| \qquad \forall x_N \in \mathfrak{X}_f$$

where:

- $l(x_N, u_N) + l_f(x_{N+1}) - l_f(x_N) \leq 0$
  by Ass #3(a)
- $-l(x_0, u_0) \leq -\alpha_1 \|x_0\|$
  by Ass #3(b)

So we can be sure that the summation of these two will always be less that some negative function of $\|x_0\|$ □

Example: Linear System

Let us consider a linear system with dynamics:

$$x^+ = Ax + Bu$$

with some control constraints and state constraints:

$$u \in \mathfrak{U} \qquad x \in \mathfrak{X}$$

Whenever we have control and state constraints it is hard to find a global Control Lyapunov Function (CLF). For this reason we aim to find a local CLF inside a given region, called invariant region (given that it has the control invariant property).

Control Lyapunov Function (CLF)

invariant region

Let us define the running cost the following quadratic function:

$$l(x, u) = \frac{1}{2}(x^T Q x + u^T R u) \qquad \text{with } Q > 0\,,\ R > 0$$

and then let us assume that the couple (A, B) is stabilizable (i.e. it can be stabilized). Under these conditions the Value function of unconstrained infinite horizon problem $\mathbb{P}_\infty^{un}$ is known and defined as:

stabilizable

$$V_\infty^{un}(x) = \frac{1}{2} x^T P x$$

where

$$P = A_k^T P A_k + Q_k$$

with

$$A_k \triangleq A + BK \qquad Q_k = Q + K^T R K \qquad u = Kx \qquad K = -(B^T P B + R)^{-1} B^T P A^T$$

In practice you can compute P with an infinite summation of matrices

$$P = \sum_{i=0}^{\infty} (A_k^T)^i Q_k A_k^i$$

Of course this is not really practical (because you are doing an infinite summation), but since you are taking the power of a matrix that is stable (i.e. A), this matrix is going to zero as you keep summing. That means that at a certain point you can sto the summation and the error will be substantially small.

Knowing that this is the solution to the problem without the constraints what is very often done is, we try to utilize it as a terminal cost in the problem with the constraint:

$$l_f(x) = V_\infty^{un}(x)$$

This makes sense given that we are looking for a local Control Lyapunov Function that is valid only inside a certain region, we can hope that if we are sufficiently close to the origin then the constraint won't play any role (in other words we expect that if we are sufficiently close to the origin the optimal choice won't be affected by the constraints) and that is what happens in practical application.

However, we still need to prove that using this particular choice of final cost, the stability assumption that we formulate for the Lyapunov Function are satisfied.

- **Terminal cost decrease**:

$$l_f(A_k x) + \frac{1}{2} x^T Q_k x - l_f(x) \leq 0 \qquad \forall x \in \mathbb{R}^n$$

where:

$$\begin{aligned} l_f(x^+) &= l_f(Ax + Bu) && u = Kx \\ &= l_f(Ax + BKx) \\ &= l_f((A + BK)x) \\ &= l_f(A_k x) \end{aligned}$$

$$\begin{aligned} l(x, u) &= \frac{1}{2}(x^T Q x + u^T R u) && u = Kx \\ &= \frac{1}{2} x^T (Q + K^T R K) x \\ &= \frac{1}{2} x^T Q_k x \end{aligned}$$

To verify that the assumption is verified, we substitute the uncontraint value function as the final cost, i.e.

$$\begin{aligned} l_f(A_k x) + \frac{1}{2} x^T Q_k x - l_f(x) &\leq 0 \\ x^T A_k^T P A_k x + x^T Q_k x - x^T P x &\leq 0 \end{aligned}$$

We obtained a summation of three pluriquadratic functions of the state, so this summation will always be negative if the matrix defining the quadratic form is negative semidefinitive

$$A_k^T P A_k + Q_k - P \leq 0$$

which reminds us of the implicit definition of P, which is equal to zero.
We obtained that the terminal cost defined as the unconstrained value function can decrease and we also know that it decreases for the specific choice of the control input:

$$u = Kx$$

Note: K could takes an arbitrary value, as long as the assumption are satisfied

- **Terminal constraints**
  Choose $\mathfrak{X}_f$ so that $u = Kx \in \mathfrak{U} \forall x \in \mathfrak{X}_f$ and $u = Kx$ makes $\mathfrak{X}_f$ will be positive invariant.
  We can choose $\mathfrak{X}_f$ to be maximal invariant constrait admissible set for $x^+ = (A + BK)x$.

In summary:

1. We can pick $\mathfrak{X}_f$ and $l_f(\cdot)$ such that:

   - $\mathfrak{X}_f$ is control invariant
   - $\forall x \in \mathfrak{X}_f \; \exists u \in \mathfrak{U}$ s.t. $l_f(f(x, u)) - l_f(x) \leq -l(x, u)$

2. We can pick $\mathfrak{X}_f = \{0\}$, i.e. picking the origin are your terminal constraint set.

   Both conditions of the previous point are satisfied: the first one by definition given that the origin is control invariant, the second one is satisfied given that the terminal cost and the running cost at the origin are zero.

   However this method results in a small basin of attraction, because you are constraining the terminal state to be in a set with just one point.

   small basin of attraction

3. We can pick N "large enough" and set $l_f(x) = 0, \mathfrak{X}_f = \mathfrak{X}$. With nonlinear systems such as robots, that is what many people do (sadly).

Stability-Extensions

1. **What if the system is time varying?**

   So far we talked about stability of a point, i.e. the origin.

   That is a very specific task for a robot to achieve, quite useless actually, because it means that you want the robot manipulator to stay still.

   Tipically you want to do trajectory tracking and with the theory that we have looked at so far, we cannot really ensure stability for trajectory tracking because the running cost is not always the same but it depends on time.

   In this case we can cast the trajectory tracking as regulation of time-varying system.

   So the good news is that the theory can be extended to a time varying system:

   Stability and recursive feasibility can still be ensured with time-varying $\mathfrak{X}_f$ and $l_f(\cdot)$.

2. **What if the cost function measures some sort of energy consumption?**

   Another situation arises when the cost function measures some sort of energy consumption (quite common). In that case one of the assumptions that we took is not satisfied, i.e.

   $$l(x, u) \geq \alpha \|x\|$$

   Because the running cost is not lower bounded anymore by the norm of the state, because the energy consumption does not depend on the norm of the state but tipically depends on both state and control.

   In this situation, people rely on Economic MPC theory to prove and ensure stability of the system. `Economic MPC theory`

1.5.4 Computation time

In robotics, a lot of attention has been given to this issue: how do you make compilation faster?

One of the key idea that you can use to speed up computation is warm start, that goes as follow: `warm start`

When you are solving the MPC optimal control problem for a given horizon from 0 to N, and then at the next iteration you would have to solve another optimal control problem that looks quite a lot like the one we just solved (but we are shifting the horizon one step forward).

As a consequence, it seems really reasonable that the optimal solution won't change very much, because the problem and the solution are very similar. So what we can do is:

Take the optimal solution that you compute it at time 0 and use it as an initial guess for the problem that you solve at the next time step. Since most of the time the optimal solution will be really similar then you have a good initial guess and the solver needs to take just a few iteration to converge to the optimal solution.

Formally:

- Shift trajectory back by 1 time step:

$$u_k^{guess} = u_{k+1}^*$$

- Use zero as initial guess for last time step:

$$u_{N-1}^{guess} = 0$$

Another simple trick is that instead of iterating until convergence to a very small convergence threshold, do not iterate until convergence: at each iteration of the control loop just do one optimization iteration. `convergence threshold`

In this way you keep computation time to a minimum and you are show that you are always doing computation on the most recent information on the state of your robot (just do 1 Newton iteration).

There are many other ideas used to improve computation time, but these are the most general ones.

1.5.5 Uncertainties in MPC

Until now we did not really took into account uncertainty. We assumed that when we are at state x we apply control u and the next state will be $f(x, u)$.

uncertainty

In practice, what happens is that when we are at state x, we apply control u and the new state will be $f(x, u) + noise$, because we do not have perfect knowledge of the dynamics, the actuators cannot really deliver the control input as you asked and you do not really know that your current state is x because you have some uncertainty in your estimation due to sensor noise.

sensor noise

There are two main approaches to deal with uncertainties.

Robust MPC

In the Robust MPC approach, basically you consider the next state x^+ as a function of the state, the control input and the noise (with the noise bounded in a certain set), i.e.

Robust MPC approach

$$x^+ = f(x, u, w) \qquad w \in \mathfrak{W}$$

As a consequence the generic constraint of the problem needs to be satisfied for every uncertainty that my happens

$$g(x, u, w) \leq 0 \qquad \forall w \in W$$

In this way, i will always maintain a safety margin w.r.t. the constraint boundary and this safety margin is what allows us to ensure that the constraints will be satisfied for any possible uncertainty realization.

Stochastic MPC

Another approach is to define the uncertainty not simply bounded but it is a stocastic uncertainty, very often model with a Gaussian distribution with zero mean:

stocastic uncertainty

Gaussian distribution

$$w \ \mathfrak{N}(0, \Sigma)$$

In this case, since the uncertainty is Gaussian you cannot hope that the constraints will be satisfied for every possible value, given that it has a small probability of being infinitely large. So tipically, you impose that the probability of the constraint to be satisfied is greater than some value:

$$Pr[g(x, u, w) \leq 0] \geq 0.95$$

In general the Robust approach is easier to implement (mathematically speaking), but even though the stochastic approach is harder to implement it is less conservative.

less conservative

Both these problem are well understood with Linear dynamics, i.e. there exists methods to find the solution for both approaches. with Nonlinear dynamics, some methods exist, but it is still ongoing research (as of now the methods either requires high computation effort or are extremely conservative).

Linear dynamics

Nonlinear dynamics

Bibliography