Modern Control Paradigms:

Lecture 6: Trajectory Optimization and Nonlinear Model Predictive Control

Up to today we have discussed mainly the techniques that allow us to stabilize points or given trajectories. However the natural question is how to find these trajectories, which may be non trivial especially for complex under-actuated robots.

And there is the answer for that...

But before we go in to details I want you to go over these short videos to understand what you may do once you master some of the trajectory optimization/stabilization tecchniques from this class:

- Aggressive Quadrotors: part 1 and part 2
- Push recovery and NMPC for A1 legged robot
- Back-flips of mini cheetah
- Fancy ATLAS parkour part 1 and part 2
- Autonomous drift over Toyota
- Vertical landing rockets

General Problem Definition

A trajectory optimization problem seeks to find the a trajectory for some dynamical system that satisfies
some set of constraints while minimizing some cost functional. Given an initial condition, $\mathbf{x}_0$, and an input trajectory $\mathbf{u}(t)$ defined over a finite interval, $t \in [t_0, t_f]$,
we can compute the long-term (finite-horizon) cost of executing that trajectory. Then the trajectory design is formulated as finite time optimal control problem:

$$\begin{aligned}
\min_{\mathbf{x}(t),\mathbf{u}(t)} \quad & \ell_f(\mathbf{x}(t_f)) + \int_{t_0}^{t_f} \ell(\mathbf{x}(t), \mathbf{u}(t)) dt \\
\text{subject to} \quad & \dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)), \quad \forall t \in [t_0, t_f] \\
& \mathbf{x}(t_0) = \mathbf{x}_0
\end{aligned}$$

Note how the problem above is very similar to LQR and MPC formulations we already had in before.

In fact this is definition of optimal control problem in general, while the LQR is just a special case dedicated to linear systems. However, there is substantial difference: for now we are looking not for the control function (policy) but just a trajectories $\mathbf{x}(t)$, $\mathbf{u}(t)$ that are just a one finite and specific solution of problem above that work for predefined initial condition $\mathbf{x}_0$

PICTURE OF PLAN VERSUS TRAJECTORY

The problem above is continues in state, control and time, thus making these variables infinite dimensional. In order to find the numerically tractable solution one should **transcript** the problem that we will have finite amount of variables to optimize.

There is a several approaches dedicated to facilitate this transcription. We roughly classify them as:

- **Direct methods:** take attempt to propose aproximation of the OCP above by transforming it to the non-convex program in variables $\mathbf{x}$ or/and $\mathbf{u}$ that is then solved with help of some nonlinear solver.
- **Indirect methods:** Use so called the adjoint equations and their gradients. (e.x. Pontryagin maximum principle). There usually much more accurate but less intuitive to implement and solve, especially when there are a lot of inequality constraints.

In this class we will briefly describe direct methods, however I highly encourage you to read this overview paper with several interesting references in where there are more information on the indirect methods as well.

Numerical Approximation, Direct Method

The direct transcription is effectively transform the infinite dimensional problem above, to the following **nonlinear program**:

$$\begin{aligned}\min_{\mathbf{x}_k,\mathbf{u}_k} \quad & \ell_f(\mathbf{x}_N) + \sum_{k_0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x}_k, \mathbf{u}_k), \quad \forall k \in [0, N-1] \\ & \mathbf{x}_0 = \mathbf{x}(0) \\ & + \text{other constraints}\end{aligned}$$

If the dynamics of your system is already given in the discrete time settings - it is naturally fall in to the formulation above. Let us for example consider the linear systems

However in practice most of our systems are **nonlinear and the continues** domain, so we need the way how to approximate the continues dynamics:

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

There is two techniques that facilitate this approximation:

- **Shooting:** The full trajectory is approximated via one **numerical integrator** applied at each sample t_k
- **Collocation:** The trajectory is represented by **piecewise polynomials** which are enforced to **satisfy dynamics in finite amount of points**.

To distinguish between them let us recall what we have done for the linear systems, at first we have solved the discrete dynamics and represent it in terms of $\mathbf{u}_k$:

$$\begin{gathered}
\mathbf{x}_1 = \mathbf{A}\mathbf{x}_0 + \mathbf{B}\mathbf{u}_0 \\
\mathbf{x}_2 = \mathbf{A}^2\mathbf{x}_0 + \mathbf{A}\mathbf{B}\mathbf{u}_0 + \mathbf{B}\mathbf{u}_1 \\
\dots \\
\mathbf{x}_N = \mathbf{A}^N\mathbf{x}_0 + \mathbf{A}^{N-1}\mathbf{B}\mathbf{u}_0 + \dots + \mathbf{B}\mathbf{u}_{N-1}
\end{gathered}$$

In the end we have used optimization to solve the following problem:

$$\begin{aligned}
\underset{\mathbf{U}_k}{\text{minimize}} \quad & \frac{1}{2}\mathbf{U}_k^T\mathcal{H}\mathbf{U}_k + \mathbf{x}_k^T\mathbf{G}^T\mathbf{U}_k + d \\
\text{subject to} \quad & \mathbf{M}\mathbf{U}_k \leq \mathbf{m} \\
& \mathbf{D}\mathbf{U}_k = \mathbf{c}
\end{aligned}$$

which turn out to be convex.

We can attempt to generalize this ides for the nonlinear systems, yielding:

$$\begin{gathered}
\mathbf{x}_1 = \mathbf{f}_d(\mathbf{x_0}, \mathbf{u_0}) \\
\mathbf{x}_2 = \mathbf{f}_d(\mathbf{x_1}, \mathbf{u_1}) = \mathbf{f}_d(\mathbf{f}_d(\mathbf{x_0}, \mathbf{u_0}), \mathbf{u_1}) \\
\dots
\end{gathered}$$

Now we can define and solve the optimization problem in terms of $\mathbf{U} = \mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{N-1}$. We call familly of such methods - **shooting**

However, there was some problems with **numerical issues** and **conditioning**. (For linear systems we had to do high powers of $\mathbf{A}$). Also the resulting problem now become more dense and nonlinear. However, for the cases when the control is particularry simple (i.e couple of constant variables) - shooting indeed produce the accurate results and may be solved much faster (thanks to the excluding $\mathbf{x}$ from the decision variables). However in robotics both of this assumptions are usually not true. However recently there is some interesting approaches that use Dynamic Programming and its approximated LQR solution. We will comeback to them at the end of this lecture.

Let us recall how we tackle numerical conditioning in linear case. We have to receipts, first - add pre-stabilizing feedback and second include the state trajectory as auxiliary decision variables, to arrive to the following optimization problem:

$$\begin{aligned} \underset{\mathbf{u}_k, \mathbf{x}_k}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \left(\mathbf{x}_k^T \mathbf{Q}_k \mathbf{x}_k + \mathbf{u}_k^T \mathbf{R}_k \mathbf{u}_k\right) + \mathbf{x}_N^T \mathbf{P} \mathbf{x}_N \\ \text{subject to} \quad & \mathbf{x}_k = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k \\ & \mathbf{x}_0 = \mathbf{x}(0) \\ & + \text{other linear constraints on } \mathbf{x}_k, \mathbf{u}_k \end{aligned}$$

In here instead of solving our difference (or differential) equations directly, we impose them as constraints in the optimization problem. We call such method **collocation**.

However usually the dynamics of our robots is given in the continues time, so one need to perform numerical integration:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \int_{t_k}^{t_{k+1}} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau, \quad \mathbf{x}(t_k) = \mathbf{x}_k$$

Then we need a tools to compute this integral, in collocation framework we distinguish this methods by two classes, in first we will increase the number of steps N in order to build the accurate solution of integral above via imposing more constraints on the state and control trajectory which are represented as low order polynomials (usually up to 3-rd order) we call such **h-methods**. Another possibility is to treat the whole trajectory as one or several **higher order polynomial** that should satisfy the dynamics in several points (collocation points), these are **p-methods**

Here we will discuss one of the simplest techniques that control as piece-wise linear functions while all integrals are represented using following trapezoidal rule:

$$\int_{t_0}^{t_f} \mathbf{g}(\tau, \mathbf{x}(\tau), \mathbf{u}(\tau)) d\tau \approx \sum_{k=0}^{N-1} \frac{1}{2} h_k (\mathbf{g}_k + \mathbf{g}_{k+1})$$

Now we can revrite the OCP as follows:

$$\begin{aligned} \min_{\mathbf{x}_k,\mathbf{u}_k} \quad & \ell_f(\mathbf{x}_N) + \sum_{k_0}^{N-1} \frac{h_k}{2}\Big(\ell(\mathbf{x}_k,\mathbf{u}_k) + \ell(\mathbf{x}_{k+1},\mathbf{u}_{k+1})\Big) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = \mathbf{x}_k + \frac{1}{2}h_k\Big(\mathbf{f}(\mathbf{x}_k,\mathbf{u}_k) + \mathbf{f}(\mathbf{x}_{k+1},\mathbf{u}_{k+1})\Big), \quad \forall k \in [0, N-1] \\ & \mathbf{x}_0 = \mathbf{x}(0) \\ & + \text{other constraints} \end{aligned}$$

The resulting control and state then intorpolated via points $\mathbf{x}_k$, $\mathbf{u}_k$ using:

$$\mathbf{x}(t) \approx \mathbf{x}_k + \mathbf{f}_k\tau + \frac{\tau^2}{2h_k}(\mathbf{f}_{k+1} - \mathbf{f}_k)$$

$$\mathbf{u}(t) \approx \mathbf{u}_k + \frac{\tau}{h_k}(\mathbf{u}_{k+1} - \mathbf{u}_k)$$

where $\tau = t - t_k$, and $\mathbf{f}_k = \mathbf{f}(\mathbf{x}_k,\mathbf{u}_k)$

There are also more accurate integral approximations such as **Hermite-Simpson**, which yields smaller errors in representing continues dynamics:

$$\int_{t_0}^{t_f} \mathbf{g}(\tau,\mathbf{x}(\tau),\mathbf{u}(\tau))d\tau \approx \sum_{k=0}^{N-1} \frac{1}{6}h_k(\mathbf{g}_k + 4\mathbf{g}_{k+\frac{1}{2}} + \mathbf{g}_{k+1})$$

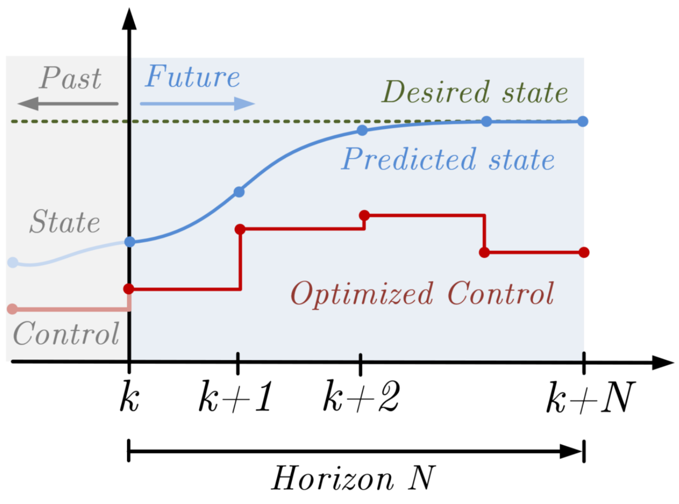
Using this type of collocation control and dynamics are represented as quadratic polynomial, while state is cubic one:

$$\begin{aligned} \min_{\mathbf{x}_k,\mathbf{u}_k,\mathbf{x}_{k+\frac{1}{2}},\mathbf{u}_{k+\frac{1}{2}}} \quad & \ell_f + \sum_{k_0}^{N-1} \frac{h_k}{6}\Big(\ell_k + 4\ell_{k+\frac{1}{2}} + \ell_{k+1}\Big) \\ \text{subject to} \quad & \mathbf{x}_{k+\frac{1}{2}} = \frac{1}{2}(\mathbf{x}_k + \mathbf{x}_{k+1}) + \frac{1}{8}h_k(\mathbf{f}_k - \mathbf{f}_{k+1}) \quad \text{interpolation} \\ & \mathbf{x}_{k+1} = \mathbf{x}_k + \frac{1}{6}h_k\Big(\mathbf{f}_k + 4\mathbf{f}_{k+\frac{1}{2}} + \mathbf{f}_{k+1}\Big) \quad \text{collocation} \\ & \mathbf{x}_0 = \mathbf{x}(0) \\ & + \text{other constraints} \end{aligned}$$

Then control is then interpolated as quadratic spline while state as a cubic one. For more information follow this tutorial

Planing as Control, Receding Horizon

As we have seen, for linear systems the trajectory planning can be effectively turned in to the control algorithm by running it online in the receeding horizon fashion.



So if we can optimize trajectories quickly enough, then we can use our trajectory optimization as a feedback policy:

- Measure the current state,
- Optimize a trajectory from the current state to obtain optimal state and control sequence
- Execute the first control element from the optimized trajectory
- Let the dynamics evolve for one step and repeat.

For the linear system the resulting control is MPC which is obtained via solution of convex problem while for nonlinear (not surprisingly) we call this **Nonlinear Model Predictive Control - NMPC**, and formulation inherently **non-convex**.

This approach is really powerful and may be applied to wide range of systems from legged robots to vertical landing rockets. However there are even more pitfalls and issues to consider relatively to convex MPC, most of them are basically computational and associated with general nonlinear optimization.

Optimal Trajectory Planning Software

The nonlinear trajectory planning, became essential tool in most of the robotic applications nowadays. So there are are a lot of mature frameworks coming from different research groups all around the world. A notable examples are ACADO, CasADi, OSC2 and many others.

Most of these frameworks are even capable to generate self contained C-code that fasten the solution of your specific problem, given predefined system dynamics and cost.

In order to facilitate yourself with OCP I suggest to use CasADi with build in `Opti` class.

The process really similar to the one you would go over with `cvxpy` :

```
# CODE FOR THE PROBLEM FORMULATION WITH CASADI AND OPTI

from casadi import *

# define the problem instance
problem = Opti()
# number of controls and states
m, n = 2, 5
N = 100 # number of control intervals
T = 2 # a time horizon
dt = T/N # length of a intervals


# control variables
u = problem.variable(1,N+1)
state = problem.variable(3,N+1) # state of the system


# form contraints
for k in range(N):
  state_prev = integrator(f, state[:,k], u[:, k], dt) # the dynamics constraints, dynami
  # + other constraints
  problem.subject_to(state[:,k+1]==state_prev)


control_bounds = problem.bounded(-100, u, 100)
problem.subject_to(control_bounds)
problem.subject_to(state[:,0]==x0)
#
problem.set_initial(state, 0)
problem.set_initial(u, 0)
# ---- solve NLP              ------
problem.solver("ipopt") # set numerical backend
sol = problem.solve()   # solve defined problem
```

Example: Please follow this colab where you can find the implementation of trapezoidal collocation for the cart pole swing-up.

Note that in some applications you may not have the predefined terminal time interval T of your trajectory, so the NLP give you the freedom to add T as additional decision variable.
Check this colab to find out the example of minimal fuel lunar landing trajectory where terminal time is not known in advance.

However, you should note how letting T to be variable making problem much more nonlinear and harder to solve.

Other Techniques

Differential Flatness

There is also a beautiful approach which has a really deep roots in the feedback linearization techniques that people use in the 'classical' nonlinear control theory and may really facilitate trajectory planning.

To go more in to this technique let us begin with the following example.

Consider the mobile robot described by following dynamics:

$$\begin{gathered}\dot{x} = v\cos\theta \\ \dot{y} = v\sin\theta \\ \dot{\theta} = \omega\end{gathered}$$

Assume that we have desired position in task space $x(t)$, $y(t)$, the question is can we calculate the reminding θ, v, ω as a function of x, y and it's derivatives.

First we take the squares of first and second terms yields:

$$v = \eta\sqrt{\dot{x}^2 + \dot{y}^2}$$

while division of the first and second:

$$\theta = \mathrm{atan2}(\eta\dot{y}, \eta\dot{x})$$

Taking the derivatives of first and second equation after a bit of algebra yields:

$$\omega = \eta\frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{v^2}$$

Not how the all signals are given as **algebraic functions** of $x, y, \dot{x}, \dot{y}, \ddot{x}, \ddot{y}$. Thus once the desired trajectory is given for the x, y, one can automatically recover the open loop controller ω, v and remaining state θ

General Differential Flatness

Given the system:

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

Is said to be differentially flat in outputs $\mathbf{y}$ if there is exists outputs (flat outputs):

$$\mathbf{y} = \mathbf{\Phi}(\mathbf{x}, \mathbf{u}, \dot{\mathbf{u}}, ..., \mathbf{u}^r).$$

Such that state and control can be expressed as **algebraic functions** of flat outputs and it's derivatives:

$$\mathbf{x} = \mathbf{x}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^r)$$
$$\mathbf{u} = \mathbf{u}(\mathbf{y}, \dot{\mathbf{y}}, \ldots, \mathbf{y}^r)$$

The differential flatness is interesting approach. However it is came with cost of nontrivial state transformation, also it is not clear how to tackle constraints that are exploited on the not flat outputs. Nevertheles if your system is differentially flat (and most trivial mobile robots actually are) you may use this property to greatly simplify planning by properly mixing nonlinear optimization with flatness.

Check for example:

- the recent methods in autonomous planning for vehicles
- celebrated works on planning for 3-d quad-rotors
- The numerical algorithms to exploit flatness property

There are numerous other different techniques are in the field of trajectory planning among many others:

- Iterative LQR and Differential Dynamic Programming
- Analytical methods based on virtual holonomic constraints

From Open to Closed Loop

Also it is really important to remember that result of the trajectory optimization techniques are just one specific local solution (open loop control) which is connected to given initial state $\mathbf{x}_0$.

While in most cases we are looking for the closed loop control as a function of state.

There are different techniques to do so, a most straight forward is to calculate the **LTV LQR along the optimizaed trajectory**, thus having feedforward/feedback pairs that are to be applied in real time. **Then with help of Lyapunov techniques can find the invariant region** (like region of attraction) of our controller. Thus eventually tranforming one open loop trajectory to the **closed loop solutiion that work well over the known bounded region of state space**. Going a bit further you may randomly pick the initial points and then build the tree of stable regions with associated trajectories and feedback controllers. The similar ideas in fact are used in the famous LQR Trees algorithm