Finite-Horizon Optimal Control by Differential Dynamic Programming with Application to Robotics

Internship report

Mengda Li
M1 student in Computer Science at ENS Paris-Saclay in 2018-2019
Intern at INRIA Paris from 04/2019 to 08/2019

supervised by

Justin Carpentier, Nicolas Mansard

Contents

Introduction - Summary

My internship is co-supervised by Justin Carpentier and Nicolas Mansard in the Willow research group at INRIA Paris in France.

Justin Carpentier is a postdoctoral researcher at the interface between Robotics, Machine Learning and Control in Willow. His research is devoted to the embedding of Optimal Control theory inside the formalism of Machine Learning, with Humanoid Robotics as a main target application.

Nicolas Mansard is a permanent researcher in the Gepetto research group at LAAS/CNRS, Toulouse. His research activities are concerned with sensor-based control, and more specifically the integration of sensor-based schemes into humanoid robot applications.

WILLOW is based in the Laboratoire d'Informatique de l'École Normale Superiéure and is a joint research team between INRIA Rocquencourt, École Normale Supérieure de Paris and Centre National de la Recherche Scientifique. Their research is concerned with representational issues in visual object recognition and scene understanding.

GEPETTO is a robotics research team in the Laboratoire d'analyse et d'architecture des systèmes of Centre National de la Recherche Scientifique. They aim at analyzing and generating the motion of anthropomorphic systems.

General Context

The goal of my internship is to embed the *augmented Lagrangian method* to *Differential Dynamic Programming (DDP)* to solve robotics *optimal control* problem with constraints.

DDP is an optimal control algorithm of the trajectory optimization class. The algorithm was introduced in 1966 by Mayne and subsequently analysed in Jacobson and Mayne's eponymous book [1]. The [2] could be served as a brief introduction to the algorithm.

Augmented Lagrangian methods are a certain class of algorithms for solving *constrained optimization problems*. The chapter 17 of the book [3] could be a good introduction.

Studied Problems

The DDP is used in robotics for finding an optimal trajectory from some point to some other point. In practice we often need to add constraint(s) to the desired trajectory. The [4] discussed how to add *box-constraint* [1] on the *control variable*. But adding box constraints on the *state variable* remains a open problem. The motivation is to implement a framework on DDP to include more general constraints which are useful in robotics.

[1] The simplest inequality constraint over the control sequence u: $\underline{b} \leq u \leq \bar{b}$ where the $\underline{b}$ is the lower bound and the $\bar{b}$ is the upper bound.

Proposed Contributions

I proposed an algorithm which combines the DDP and augmented Lagrangian to solve equality constrained optimal control problem. The DDP is used to solve sequentially the sub-problem generated by augmented Lagrangian method. Frankly there is not a lot of contributions in the theoretical part. The internship focuses on the implementation of algorithm and test on robotics examples.

Arguments in term of validity

I implemented the algorithm on unicycle problem and the convergence is satisfactory. There are some problems in my implementation on manipulator arm: the norm of the gradient of Lagrangian doesn't converge to 0. I tried to debug the problem for a long time without success.

I cannot show this solution is a valid good solution. Personally I think the *penalty method* which is more simple and easier to implement would work sufficiently good in practice after a long sequence for solving subproblems. The augmented Lagrangian method also requires a long series of iterations for solving sub-problems. Even in theory it requires less iterations than penalty method, but in case of manipulator it has the same level of slowness and could fail in some case like reaching 4 different points while the penalty method succeed in this case and didn't take a long time to satisfy the constraints.

Review and Prospects

The idea of the algorithm is general: combining a pencil with a rubber, and it is very common in numerical optimization algorithms: solving a unconstrained optimization sub-problem sequentially to converge to a solution of a constrained optimization problem. The *Sequential Quadratic Programming* and *Interior point method* share the same idea.

If this solution is a good solution, we can extend the method to solve problems with inequality constraints by using the bound-constrained formulation[3]. During the internship, I implement the algorithm of augmented Lagrangian manually and maybe that is the source of problem. Using existing augmented Lagrangian package like `LANCELOT` and binding with DDP package may be a good solution. The only problem is the difference of programming languages.

Chapter 1

Definitions of the problem

We want to find a path from some point x_0 to some point p which doesn't cost too much.

1.1 Mathematical formulations

Let $n \in \mathbb{N}$, $m \in \mathbb{N}$, $x_0 \in \mathbb{R}^n$, $p \in \mathbb{R}^n$, $x \in \mathscr{C}^1([0,T] \to \mathbb{R}^n)$, $u \in \mathscr{L}^\infty([0,T) \to \mathbb{R}^m)$, and $f \in \mathscr{C}^2(\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n)$.

n is the dimension of the state variable and m is the dimension of the control variable. We use x as the state variable, u as the control variable.

x_0 is the initial position (or state) and p is the desired final position.

x is an unknown function from $[0,T]$ to $\mathbb{R}^n$. It represents the trajectory.

u is the control function which represents the variation of control over time.

We can control the trajectory x by u:

$$\dot{x}(t) = f(x(t), u(t)) \tag{1.1}$$

the derivative of x depends on x, u and a function f.

Goal 1: Controllability

$$\begin{array}{ll} \text{find} & u \\ \text{subject to} & x(0) = x_0,\ x(T) = p, \\ & \dot{x}(t) = f(x(t), u(t)). \end{array} \tag{1.2}$$

Goal 2: Optimal Control

In reality, everything has a cost.

Controlling something has a cost, being somewhere has a cost. So we define a cost (or loss) function l on x and u, supposing that l is in $\mathscr{C}^2(\mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R})$.

$$\begin{array}{ll} \underset{u}{\text{minimize}} & \displaystyle\int_0^T l(x(t), u(t))dt \\ \text{subject to} & x(0) = x_0,\ x(T) = p, \\ & \dot{x}(t) = f(x(t), u(t)). \end{array} \tag{1.3}$$

1.1.1 Transformation of the problem

Satisfying $x(0) = x_0$ is easy while satisfying $x(T) = p$ is not trivial considering $\dot{x}(t) = f(x(t), u(t))$. To ensure that $x(T) = p$, we can define a final position loss function l_f on x, $l_f \in \mathscr{C}^2(\mathbb{R}^n \to \mathbb{R})$ and transform the problem to:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \int_{[0,T[} l(x(t), u(t))dt + l_f(x(T)) \\ \text{subject to} \quad & x(0) = x_0, \\ & \dot{x}(t) = f(x(t), u(t)). \end{aligned} \tag{1.4}$$

For example, $l_f(x) = \frac{w}{2}\|x(T) - p^*\|$. w is a weight parameter (like 10^6).

1.1.2 Discretization of a continuous problem

To be computable, we discretize the problem with an horizon $N \in \mathbb{N}^*$, let $h = \frac{T}{N}$, $i \in \{0, ..., N-1\}$:

$$\begin{aligned} x_{i+1} &= x_i + h\dot{x}_i \\ &= x_i + hf(x_i, u_i) \\ &= F(x_i, u_i) \end{aligned}$$

$$J(U) = \sum_{i=0}^{N-1} L(x_i, u_i) + L_T(x_N) \approx \int_{[0,T[} l(x(t), u(t))dt + l_f(x(T)) \tag{1.5}$$

where $F(x_i, u_i) = x + hf(x_i, u_i)$, $L(x_i, u_i) = h\, l(x_i, u_i)$, $L_T = l_f$.

Our goal is then to minimize the J in $\ell_N^\infty \subset \mathbb{R}^N$ - a finite-dimensional vector space where the control sequence $U = (u_0, ..., u_{N-1})$ lies. Here, we transform a problem of infinite-dimensional control to a finite one.

1.2 Dynamic Programming

Notice that:

$$\begin{aligned} \min_U J(U) &= \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \\ &= \min_{u_0} L(x_0, u_0) \\ &+ (\min_{u_1} L(x_1, u_1) + ... \min_{u_{N-1}} L(x_{N-1}, u_{N-1}) + L_T(F(x_{N-1}, u_{N-1}))) \end{aligned} \tag{1.6}$$

Optimizing directly J with respect to the whole U may be difficult. Following the principle of dynamic programming, we can optimize only with respect to u_i one by one if all future choices $u_{i+1}, .., u_{N-1}$ are optimal.

We define the optimal value function V and a function Q such that $V(x) = \min_u Q(x, u)$.

$$\begin{aligned} V_0(x_0) &= \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \\ &= \min_{u_0} L(x_0, u_0) + \underbrace{\min_{u_1} ... \min_{u_{N-1}} L(x_1, u_1) + ... + L_T(x_N)}_{V_1(x_i)} \end{aligned} \tag{1.7}$$

$$Q_0(x_0, u_0) = L(x_0, u_0) + \underbrace{\min_{u_1} ... \min_{u_{N-1}} L(x_1, u_1) + ... + L_T(x_N)}_{V_1(x_i)} \tag{1.8}$$

For all $i \in \{0, 1, ..., N-1\}$ iteratively:

$$\begin{aligned} V_i(x_i, u_i) &= \min_{u_i} L(x_i, u_i) + V_{i+1}(x_{i+1}) \\ V_N(x_N) &= L_T(x_N) \end{aligned} \tag{1.9}$$

$$\begin{aligned} Q_i(x_i, u_i) &= L(x_i, u_i) + V_{i+1}(x_{i+1}) \\ &= L(x_i, u_i) + V_{i+1}(f(x_i, u_i)) \\ &= L(x_i, u_i) + \min_{u_{i+1}} Q_{i+1}(f(x_i, u_i), u_{i+1}),\ i \leq N-2 \end{aligned} \tag{1.10}$$

where

$$V_i(x_i) = \min_{u_i} Q_i(x_i, u_i) \tag{1.11}$$

• V_i is the optimal value of the cost in the sub-trajectory in the time interval $[t_i, T]$ which depends only on the initial position of the sub-trajectory x_i.
• Q_i is the "raw form" of V_i before minimizing with respect to u_i considering that the future controls $u_{i+1}, .., u_{N-1}$ are always optimal.

Remark. *In practice, what we use in the DDP is the quadratic approximation of the function Q, and that is why we name it Q. If f is linear and all cost functions are quadratic, then Q and V will all be quadratic and we can exactly find the $\min_{u_i} Q_i(x_i, u_i)$ for all i.*

Duality

From 1.9, we can deduce that V_x is the optimal Lagrangian multiplier of the problem 1.4. If the sequence u is optimal, then

$$V_x^i = L_x^i + V_x^{i+1} F_x^{i\,1} \tag{1.12}$$

So $\nabla J(x, u) + A(x, u)^\top \lambda = 0$ holds for $\lambda := V_x$ where $A(x, u)$ is the jabobian matrix for the dynamic constraints. Here I consider J as a function of x, u instead of u.

The dual Lagrangian multiplier is particularly useful for verifying the first-order necessary conditions of optimality.

[1] the notations will be introduced in the next chapter.

Chapter 2

Differential Dynamic Programming

The computation of differentials (partial derivatives, gradient, and hessian...) is the core of optimization once the function we optimize is differentiable.

Remind that our goal is to optimize $J \in \mathscr{C}^2(\mathbb{R}^N \to \mathbb{R})$.

$$\begin{aligned} J(u_0, ...u_{N-1}) =& L(x_0, u_0) + L(F(x_0, u_0), u_1) + \\ & L(F(F(x_0, u_0), u_1), u_2) + \\ & L_T[F(F(...F(F(F(x_0, u_0), u_1), u_2), ..., u_{N-2}), u_{N-1})] \end{aligned} \tag{2.1}$$

J is differentiable. However, the computation of differentials is not always easy because of the compositions of the "dynamic system" of F. Differential dynamic programming can avoid this hardcore work.

2.1 Decomposition to sub-problems

Remind our final goal is, for all x_0,

Goal in the general form

$$\begin{aligned} \text{find} \quad & U^* = (u_0^*, u_1^*, ..., u_{N-1}^*) \\ \text{subject to} \quad & x(0) = x_0, \\ & J(U^*) = \min_{u_0} \min_{u_1} ... \min_{u_{N-1}} J(U) \end{aligned} \tag{2.2}$$

which is equivalent to, for all $i \in \{0, 1, ..., N-1\}$,

Goals in the DDP form

$$\begin{aligned} \text{find} \quad & u_i^* \\ \text{subject to} \quad & x(t_i) = x_i, \\ & Q_i(x_i, u_i^*) = \min_{u_i} Q(x_i, u_i) = V_i(x_i) \end{aligned} \tag{2.3}$$

where $t_0 = 0,\ t_{i+1} = t_i + h$

We will first pre-compute what we need to find each "minimum" of Q_i (their gradients and hessians) from $N-1$ to 0 which is called a *Backward Pass*. Once we have these information from differentials, we will compute the "minimum" from 0 to N which is called a *Forward Pass*.

The raison why we first go from future to present is the convenience of computing of differentials to find the minimum. However, we cannot find them immediately in the future: we need to wait the time goes by to compute the minimum from present one by one.

Remark. *We don't search the exact global minimum and we don't even get a local minimum by doing backward and forward pass if the functions Q_i are not quadratic or linear in one iteration. However, we will use the principle of* Sequential Quadratic Programming *to make an approximation of the problem as an LQR to compute a descent direction and converge* [1] *to a local minimum.*

2.2 Linear Quadratic Regulator (LQR)

If L, L_T are quadratic and F is linear, then the DDP is a *Linear Quadratic Regulator*. All Q and V are quadratic, so all $\approx$ will be $=$ in this section and we can resolve the problem in one iteration.

The LQR is only one method to solve a particular *Quadratic Programming Problem* with *Linear Equality Constraints* which exploits the structure of the problem to solve the problem efficiently:

$$\begin{aligned} \underset{x\in\ell^\infty_{N+1}, u\in\ell^\infty_N}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} L(x_i,u_i) + L_T(x_N) \\ \text{subject to} \quad & x(0) = x_0, \\ & x_{i+1} = F(x_i,u_i)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.4}$$

In this section, we assume that the ongoing loss L with respect to each state is the same. However, LQR works also in the case we have different quadratic L_i with respect to (x_i, u_i).

2.2.1 Backward pass: Optimal step, Q and its derivatives

For some fixed x_i, u_i

$$V_i(x_i) = \min_u Q_i(x_i,u) = Q_i(x_i,u_i) + \min_{\delta u}[Q_i(x_i,u_i+\delta u) - Q_i(x_i,u_i)] \tag{2.5}$$

We approximate Q_i quadratically in practice, because *doing global optimization of a function without special properties is hopeless*[2], and approximating the function in higher orders Taylor series is computationally expensive:

$$Q_i(x_i+\delta_x, u_i+\delta_u) - Q_i(x_i,u_i) \approx DQ_i(x_i,u_i)(\delta_x,\delta_u) + \frac{1}{2}D^2Q_i(x_i,u_i)(\delta_x,\delta_u)(\delta_x,\delta_u) \tag{2.6}$$

What we need is then to find the optimal δ_u^* minimizing $Q_i(x_i+\delta_x, u_i+\delta_u)$. **For all** x_i, u_i, δ_x

$$\begin{aligned} \delta_u^*(i) &:= \underset{\delta_u(i)}{\arg\min}\, Q_i(x_i+\delta_x(i), u_i+\delta_u(i)) \\ \delta_u^*(i) &= -(D^2_{uu}Q_i(x_i,u_i))^{-1}(\nabla_u Q_i(x_i,u_i) + D^2_{ux}Q_i(x_i,u_i)\delta_x) \end{aligned} \tag{2.7}$$

[1] there is some theoretical subtleties in the roll-out (DDP line search) phase. But there are theoretical convergence guarantees on classical line search. DDP line search strategies work in practice better and respect the equality constraint of dynamics

[2] by Francis Bach, 2019

By abuse of notation, we note

- $$\delta_u^* = -Q_{uu}^{-1}(Q_u + Q_{ux}\delta_x) \tag{2.8}$$

with the open-loop term:

$$k = -Q_{uu}^{-1}Q_u \tag{2.9}$$

and the feedback gain term:

$$K = -Q_{uu}^{-1}Q_{ux} \tag{2.10}$$

So

$$\delta_u^* = k + K\delta_x \tag{2.11}$$

To compute the best variation of control variable δ_u^*, we need to know the differential of Q:

$$DQ_i(x_i, u_i) = DL(x_i, u_i) + \underline{DV_{i+1}}(x_{i+1}) \circ DF(x, u) \tag{2.12}$$

where the unknown is underlined. And the hessian which can be ignored in iLQR but not in DDP [2]

$$\begin{aligned} D^2Q_i(x_i, u_i)(e_k)(e_l) = {} & D^2L(x_i, u_i)(e_k)(e_l) + \\ & \underline{DV_{i+1}}(x_{i+1})\big(D^2F(x_i, u_i)(e_k)(e_l)\big) + \\ & \underline{D^2V_{i+1}}(x_{i+1})\big(DF(x_i, u_i)(e_k)\big)\big(DF(x_i, u_i)(e_l)\big) \end{aligned} \tag{2.13}$$

[3] where e_k, e_l are vectors of canonical basis. We need use them as canonical variations to compute the Hessian matrix. In a less rigorous form:

$$\begin{aligned} Q_u &= L_u + \underline{V_x'}\, F_u \quad {}^{[4]} \\ Q_{uu} &= L_{uu} + \underline{V_x'} \cdot F_{uu} + F_u^T \underline{V_{xx}'} F_u \\ Q_{ux} &= L_{ux} + \underline{V_x'} \cdot F_{ux} + F_u^T \underline{V_{xx}'} F_x \end{aligned} \tag{2.14}$$

where the $\cdot$ denote the tensor dot product. [5]

As the underline information is from the future, we need the backward pass.

2.2.2 Forward pass: V and its derivatives

In each backward pass, we compute the first and second order differentials (≈gradient and hessian) of each V_i and we use this information to find the quadratically approximate minimum of each V_i (thus J).

If $i = N$, $V_N = L_T$. If $i \in [0\,..\,N-1]$, for all $\underline{\text{fixed } u_i}$, by 2.5,

$$V_i(x_i) = \min_u Q_i(x_i, u) \approx Q_i(x_i, u_i) - \frac{1}{2} D_u Q_i(x_i, u_i)(D_{uu}^2 Q_i(x_i, u_i))^{-1} \nabla_u Q_i(x_i, u_i) \tag{2.15}$$

In fact:

$$V_i(x_i) = \min_u Q_i(x_i, u) = Q_i(x_i, u_i + \delta_u^*(i)) \tag{2.16}$$

[3] To prove this, use the theorem in the reminder of Differential calculus

[4] In the community, we use the prime notation to mean the "next"

[5] It is not a real tensor dot product and F_{uu}, F_{ux} are neither real tensors defined in Wikipedia. It is a 3-dimensional array, but we don't need this term because F is supposed to be linear: $D^2F = 0$.

Notice that here $\delta_u^*(i) = k(i) = -Q_{uu}^{-1}Q_u$. So

$$\begin{aligned} V &\approx Q + Q_u^T\delta_u^* + \frac{1}{2}{\delta_u^*}^T Q_{uu}\delta_u^* \\ &= Q - \frac{1}{2}Q_u^T Q_{uu}^{-1}Q_u \end{aligned} \tag{2.17}$$

Remark that $V = Q + \frac{1}{2}Q_u^T\delta_u^*$ which is like a gradient descent.

$$DV_i(x_i) = D_xQ_i(x_i,u_i) - D_uQ_i(x_i,u_i) \circ [D_{uu}^2Q_i(x_i,u_i)]^{-1} \circ D_{ux}^2Q_i(x_i,u_i) \tag{2.18}$$

$$D^2V_i(x_i) = D_{xx}^2Q_i(x_i,u_i) - D_{xu}^2Q_i(x_i,u_i) \circ [D_{uu}^2Q_i(x_i,u_i)]^{-1} \circ D_{ux}^2Q_i(x_i,u_i) \tag{2.19}$$

which are

$$\begin{aligned} V_x &= Q_x - Q_uQ_{uu}^{-1}Q_{ux} &&= Q_x + Q_uK \\ V_{xx} &= Q_{xx} - Q_{xu}Q_{uu}^{-1}Q_{ux} &&= Q_{xx} + Q_{xu}K \end{aligned} \tag{2.20}$$

Also

$$V_x = Q_x + kQ_{ux} \tag{2.21}$$

Remark. *As we do a quadratic approximation, the higher order (> 2) differentials are supposed to be null. So Q_{uu}^{-1} is constant.*

2.2.3 Roll-out

Once we have the $\delta_u^* = (\delta_u(i)^*)_{i\in[0..N-1]}$, we "integrate" δ_u^* to obtain $u^* := u + \delta_u^*$. Then we compute x^* recursively by doing the equation in Euler Schema:

$$\begin{aligned} x_0^* &= x_0, \\ x_{i+1}^* &= F(x_i^*, u_i^*)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.22}$$

2.3 Feasible and non-feasible trajectories

It can happen that there are some "gaps" between the next predicted state node $x_{i+1}^- = F(x_i,u_i)$ and the real x_{i+1} such that $x_{i+1} = F(x_i,u_i) + \epsilon_{i+1}\ \forall i \in [0\,..\,N-1]$. The sequence of states will be

$$\begin{array}{ccc} x_0 & \xrightarrow{u_0} & x_1^- \\ & & \downarrow \epsilon_1 \\ & & x_1 \xrightarrow{u_1} x_2^- ... \end{array}$$

When all gaps ϵ are 0, the trajectory is *feasible*.

2.3.1 Non-feasible start in LQR

The chain of Q, V will be changed: $\forall i \in [0\,..\,N-1]$

$$Q_i(x_i,u_i) = L_i(x_i,u_i) + V_{i+1}(\underbrace{f(x_i,u_i)}_{x_{i+1}^-}) \tag{2.23}$$

The schema in the interior loop will be

$$\begin{array}{rcl} ... \uparrow [V_{N-1}(x_{N-1})] & \xleftarrow{2.20} & [Q_{N-1}(x_{N-1}, u_{N-1})] \xleftarrow{2.14} [V_N(x_N^-)] \\ & & \uparrow 2.25 \\ & & [V_N(x_N)] \end{array}$$

[...] represents the first and second order differentials of ..., not the function value. For all $i \in [1\,..\,N]$,

$$\begin{aligned} V^i(\overbrace{x_i^- + \epsilon_i}^{x_i}) &= V^i(x_i^-) + V_x^i(x_i^-) \cdot \epsilon_i + \frac{1}{2}\epsilon_i^T V_{xx}^i \epsilon_i \\ &= \text{constant}(x_i^-) + V_x^i(x_i^-) \cdot x_i + \frac{1}{2}x_i^T V_{xx}^i x_i - x_i^T V_{xx}^i x_i^- \end{aligned} \tag{2.24}$$

$$\begin{aligned} V_x^i(x_i) &= V_x^i(x_i^-) + x_i^T V_{xx}^i - {x_i^-}^T V_{xx}^i \\ &= V_x^i(x_i^-) + \epsilon_i^T V_{xx}^i \\ V_x^i(x_i^-) &= V_x^i(x_i) - \epsilon_i^T V_{xx}^i \end{aligned} \tag{2.25}$$

Once all Q_u, Q_{uu}, Q_{ux} are computed, we have δ_u i.e. k, K. Then either we can compute a new feasible trajectory with the classical full-step LQR roll-out, or we can perform a partial step to have another non feasible trajectory with a smaller gaps[6] *with respect to* the LQR linear dynamics model.

Remember that what we always obtain from a inner-loop the k, K. How to roll out the new trajectory depends on our roll-out policy which means that we can obtain a new non feasible trajectory or a feasible one.

2.4 Algorithms

In this section, we consider the DDP as a *Non-Linear Optimization Problem* with respect to the state sequence $x \in \ell_{N+1}^\infty, u \in \ell_N^\infty$:

$$\begin{aligned} \underset{x \in \ell_{N+1}^\infty, u \in \ell_N^\infty}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{N-1} L(x_i, u_i) + L_T(x_N) \\ \text{subject to} \quad & x(0) = x_0, \\ & x_{i+1} = F(x_i, u_i)\ \forall i \in [0\,..\,N-1] \end{aligned} \tag{2.26}$$

At the beginning of game, we may have an initial trajectory $x = (x_i)_{i \in [0..N]}, u = (u_i)_{i \in [0..N-1]}$ in hand. x, u could be non feasible, i.e. not respect the (dynamic) equality constraint. So there will be some gaps in respect of the *exact* numerical schema of the differential equation. Note that these gaps are different from the gaps in the following *LQR* numerical schema.

2.4.1 Sequential LQR Programming

Let $\bar{L}, \bar{L}_T$ be some quadratic approximation functions of L, L_T, and $\bar{F}$ be some linear approximation function of F. We repeat the following procedure until the criteria of convergence is satisfied.

[6] the gaps could be bigger in a non LQR model. For example, in the each sequential LQR phase of DDP, the gaps of trajectory could be bigger with respect to the real non linear dynamics constraint but reduced in the LQR approximation model.

Approximative LQR problem

$$\begin{aligned} \underset{x\in\mathbb{X}^{N+1},u\in\mathbb{U}^N}{\text{minimize}} \quad & J(x,u)=\sum_{i=0}^{N-1}\bar{L}_i(x_i,u_i)+\bar{L}_T(x_N) \\ \text{subject to} \quad & x(0)=x_0, \\ & x_{i+1}=\bar{F}_i(x_i,u_i)\ \forall i\in[0\,..\,N-1] \end{aligned} \tag{2.27}$$

Given any *fixed* initial guess $x=(x_i)_{i\in[0..N]}, u=(u_i)_{i\in[0..N-1]}$, we define: for any $i\in[0\,..\,N-1]$,

$$\bar{L}_i(x,u):=L(x_i,u_i)+DL(x_i,u_i)(x-x_i,u-u_i)+\frac{1}{2}D^2L(x_i,u_i)(x-x_i,u-u_i)(x-x_i,u-u_i)$$

$$\bar{L}_i(x_i+\delta_x^i,u_i+\delta_u^i)=L(x_i,u_i)+\nabla L(x_i,u_i)(\delta_x^i,\delta_u^i)+\frac{1}{2}(\delta_x^i,\delta_u^i)\nabla^2L(x_i,u_i)\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix} \tag{2.28}$$

I note $\delta_x(i),\delta_u(i)$ as δ_x^i,δ_u^i to save space. For terminal loss:

$$\bar{L}_T(x):=L_T(x_N)+DL_T(x_N)(x-x_N)+\frac{1}{2}D^2L_T(x_N)(x-x_N)(x-x_N)$$

$$\bar{L}_T(x_N+\delta_x^N)=L_T(x_N)+\nabla L_T(x_N)\cdot\delta_x^N+\frac{1}{2}(\delta_x^N)^T\nabla^2L_T(x_N)(\delta_x^N) \tag{2.29}$$

Note that $\sum_{i=0}^{N-1}L(x_i,u_i)+L_T(x_N)$ is constant, so what we optimize is

$$\sum_{i=0}^{N-1}[\nabla L(x_i,u_i)\cdot\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix}+\frac{1}{2}\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix}^T\nabla^2L(x_i,u_i)\begin{pmatrix}\delta_x^i\\ \delta_u^i\end{pmatrix}]+\nabla L_T(x_N)\cdot\delta_x^N+\frac{1}{2}(\delta_x^N)^T\nabla^2L_T(x_N)(\delta_x^N)$$

with respect to $\delta_x=(\delta_x^i)_{i\in[0..N]}$ and $\delta_u=(\delta_u^i)_{i\in[0..N-1]}$.

The linearization of F is defined as:

$$\begin{aligned} \forall i\in[0\,..\,N-1],\ \bar{F}_i(x,u)&=F(x_i,u_i)+DF(x_i,u_i)(x-x_i,u-u_i) \\ \bar{F}_i(x_i+\delta_x^i,u_i+\delta_u^i)&=\underbrace{F(x_i,u_i)}_{x_{i+1}}+\nabla F(x_i,u_i)(\delta_x^i,\delta_u^i) \\ \bar{F}_i(\delta_x^i,\delta_u^i)&=\nabla F(x_i,u_i)(\delta_x^i,\delta_u^i) \end{aligned} \tag{2.30}$$

We expect $x_{i+1}+\delta_x^{i+1}=\bar{F}_i(x_i+\delta_x^i,u_i+\delta_u^i)+\epsilon_{i+1}$, so the equality constraint becomes

$$\begin{aligned} \delta_x^0&=0 \\ \delta_x^{i+1}&=\bar{F}_i(\delta_x^i,\delta_u^i)\ \forall i\in[0\,..\,N-1] \end{aligned} \tag{2.31}$$

If the trajectory (x,u) does not respect the linear dynamics constraint, there are gaps to be computed:

$$\forall i\in[1\,..\,N],\ \epsilon_i:=x_i-\bar{F}_{i-1}(x_{i-1},u_{i-1}) \tag{2.32}$$

The gaps will not be changed if the optimization variable is (δ_x,δ_u) instead of (x,u). By 2.3.1,

Chapter 3

Augmented Lagrangian on DDP

3.1 Augmented Lagrangian method

Augmented Lagrangian methods are a certain class of algorithms for solving constrained optimization problems. They have similarities to *penalty methods* in that they replace a constrained optimization problem by a **series of unconstrained problems** and add a penalty term to the objective; the difference is that the augmented Lagrangian method adds yet another term, designed to mimic a *Lagrange multiplier*. The chapter 17 of [3] gives a general framework of this algorithm from pages 514 to 522. Here I only give an outline.

3.1.1 Penalty method

The *Augmented Lagrangian methods* is very related to the *Penalty method*. The *Penalty method* transform a *constrained optimization problem*:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & c_i(x) = 0, \quad i \in \mathcal{E} \end{aligned} \tag{3.1}$$

to an *optimization problem without constraint*:

$$\underset{x}{\text{minimize}} \quad f(x) + \frac{\mu}{2} \sum_{i \in \mathcal{E}} c_i^2(x), \quad \mu > 0 \tag{3.2}$$

μ is the penalty parameter. We solve sequentially the 3.2 with an increasing sequence[1] of μ_k until the final convergence test [2]is satisfied.

3.1.2 Augmented Lagrangian function

Augmented Lagrangian function is modified version of the *penalty function*:

$$\mathcal{L}^A(x) := f(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{\mu}{2} \sum_{i \in \mathcal{E}} c_i^2(x) \tag{3.3}$$

[1]For example, $\mu_{k+1} = 2\mu_k$ if some residual tolerance is satisfied.
[2]For example, the residual $\|c(x)\| < 10^{-8}$

The λ is the dual variable called *Lagrangian multiplier* who has the same dimension as the constraints. The idea of this method is similar to the penalty method: updating λ, μ sequentially and solving the 3.3. Let $(\tau_k) \in {\mathbb{R}_+^*}^{\mathbb{N}}$ be the sequence of tolerance. At each iteration, if the tolerance is satisfied, we update $\lambda^{k+1} := \lambda^k + \mu^k c(x_k)$. If the algorithm finally finds a numerical optimum, the final λ would also be the dual optimum numerically.

3.2 Augmented DDP

By the method augmented Lagrangian, for solving the problem 3.4 with equality constraints, we solve sequentially 3.6 by updating Lagrangian multiplier λ and penalization parameter μ until the KKT conditions for the original problem are satisfied:

$$\begin{aligned}
\underset{x\in\mathbb{X}^{T+1},u\in\mathbb{U}^T}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{T-1} L_i(x_i,u_i) + L_T(x_T) \\
\text{subject to} \quad & x(0) = x_0, \\
& x_{i+1} = F(x_i,u_i) && \forall i \in [0\,..\,T-1] \\
& c_j(x,u) = 0 && \forall j \in [0\,..\,M-1]
\end{aligned} \tag{3.4}$$

Let $c : \mathbb{X}^{T+1} \times \mathbb{U}^T \to \mathbb{R}^M$ [3]be the function for the set of constraints. However, one constraint c_j usually does not apply on all the states x or controls u but one x_i, u_i (or a few) of them and they are *independent*. So, in practice, I define $c_j^i : \mathbb{X} \times \mathbb{U} \to \mathbb{R}$ for each x_i, u_i and $c_j^T : \mathbb{X} \to \mathbb{R}$ if the constraint c_j applies on them i.e. $j \sim i$. Then I "augment" the loss (and terminal loss) with related constraint(s):

$$\mathcal{L}_i^A(x_i,u_i,\lambda,\mu) := L_i(x_i,u_i) + \sum_{j\sim i} \lambda_j c_j(x_i,u_i) + \frac{\mu}{2}\sum_{j\sim i} c_j^2(x_i,u_i) \quad \forall i \in [0\,..\,T-1] \tag{3.5}$$

$$\begin{aligned}
\underset{x\in\mathbb{X}^{T+1},u\in\mathbb{U}^T}{\text{minimize}} \quad & J(x,u) = \sum_{i=0}^{T-1} \mathcal{L}_i^A(x_i,u_i,\lambda,\mu) + \mathcal{L}_T^A(x_T,\lambda,\mu) \\
\text{subject to} \quad & x(0) = x_0, \\
& x_{i+1} = F(x_i,u_i) && \forall i \in [0\,..\,T-1]
\end{aligned} \tag{3.6}$$

For all $\lambda \in \mathbb{R}^M, \mu \in \mathbb{R}^{+*}$, the problem 3.6 can be solved by a DDP (or FDDP) solver ([5])

[3]Let $\mathbb{X}$ be the state space, for example $\mathbb{R}^n$ and $\mathbb{U}$ be the control space, for example $\mathbb{R}^m$. We can use $\mathcal{X} := \mathbb{X}^{T+1}, \mathcal{U} := \mathbb{X}^T$ to denote the state and control trajectory space.

Appendix

Examples with Augmented Lagrangian method

QP problems

Here I show some graphs of convergence of Augmented Lagrangian method. I use Augmented Lagrangian method to solve classical *Quadratic Programming* problems with equalities constraints. The dimension of problem is 100 and the number of constraints is 50 with random matrices and vectors. ν is the penalization term.

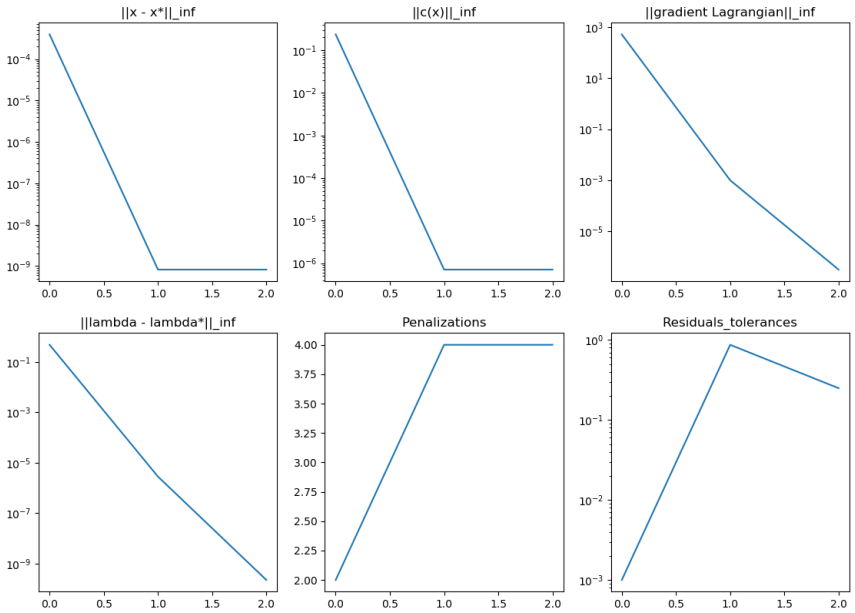

Figure 3.1: Example of resolution of a QP

If the parameters (here the residual tolerances) are not well chosen, there would be some numerical problems.

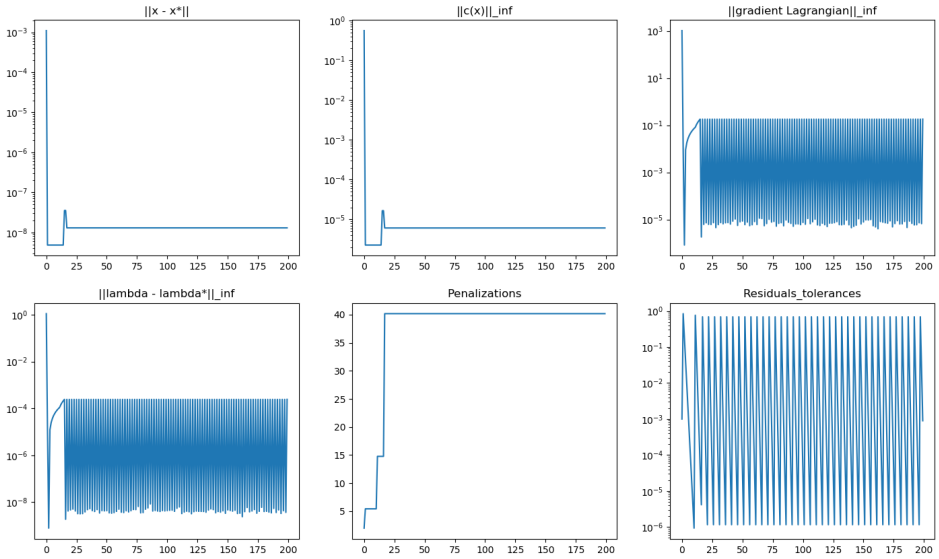

Figure 3.2: Example of resolution of a QP

Augmented Lagrangian on Unicycle problem

Unicycle problem is a simplified car driving problem: we want to find a sequence of control to have a vehicular trajectory satisfying certain conditions. Here, given a time interval T, I want the car to pass the yellow point at $\frac{T}{2}$ and the red point at T.

The control variable $u := (v, w)$ is composed by the velocity and the angular velocity of car. The state variable (x, y, θ) is composed by the position in 2D plan and the angle of orientation. The dynamic equation is:

$$\begin{aligned} \dot{x} &= v \cos\theta \\ \dot{y} &= v \sin\theta \\ \dot{\theta} &= w \end{aligned} \tag{3.7}$$

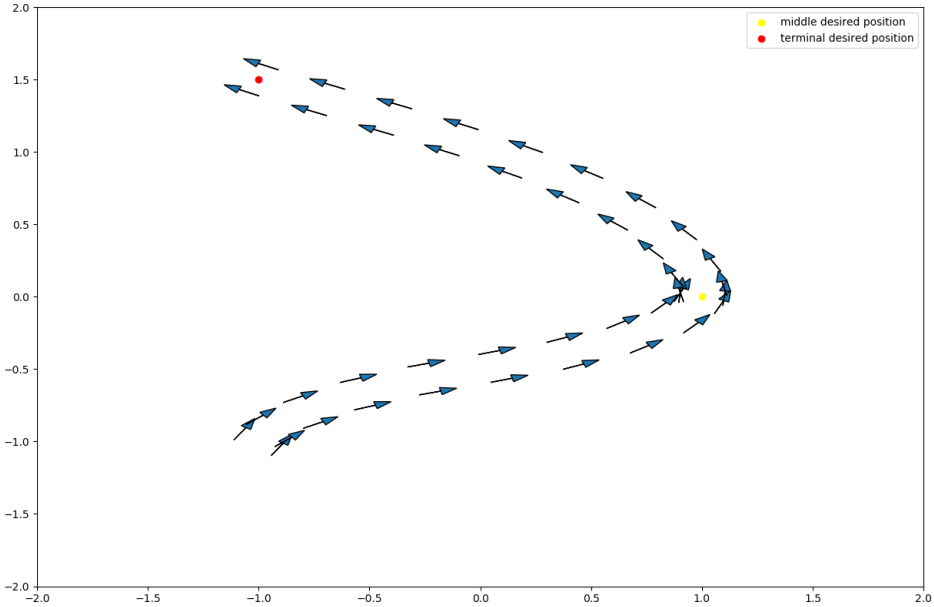

Figure 3.3: Unicycle problem, converged trajectory

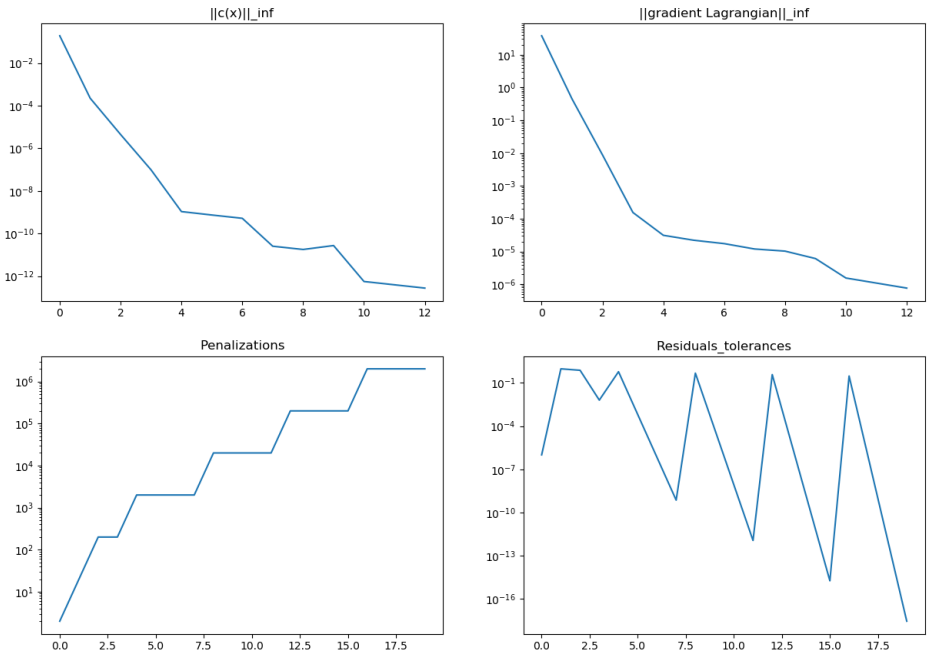

Figure 3.4: Unicycle problem, convergence of variables

Reminder of Differential calculus

Theorem 3.2.1 (Second order differential of composition of functions)**.**

$$D^2(f\circ g)(x)(h_1)(h_2) = \\ Df(g(x))\big(D^2g(x)(h_1)(h_2)\big) + D^2f(g(x))\big(Dg(x)(h_1)\big)\big(Dg(x)(h_2)\big)$$

Proof. By the lemma below. □

For $f : X \to L(V,W),\ g : X \to L(U,V)$

Definition 3.2.1 (generalized product of functions)**.**

$$(f\cdot g)(x) := f(x)\circ g(x)$$

Lemma 3.2.2.

$$D(f\cdot g)(x)(h) = f(x)\circ Dg(x)(h) + Df(x)(h)\circ g(x)$$

Remark. *Books remain silent about this theorem. Either not mentioned or in a false expression.*

Bibliography

[1] D.H. Jacobson and D.Q. Mayne. *Differential Dynamic Programming.* Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1970.

[2] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 10 2012.

[3] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, 2006.

[4] Yuval Tassa, Nicolas Mansard, and Emo Todorov. Control-limited differential dynamic programming. *Proceedings - IEEE International Conference on Robotics and Automation*, 05 2014.

[5] Carlos Mastalli, Rohan Budhiraja, Nicolas Mansard, and Justin Carpentier. *Contact RObot COntrol by Differential DYnamic programming Library.*