AA 274: Principles of Robotic Autonomy | Winter 2019

Lecture 3: Open-Loop and Closed-Loop Motion Control

3.1 Introduction

This lecture discusses the main techniques in optimal control and trajectory optimization.

First, we will review constrained optimization by formulating the problem using the Lagrangian. Then we will explore two different methods in solving an optimal control problem: we will thoroughly explore some equations behind Indirect Methods, enabling its computation along with examples, and give an overview on another approach: Direct Methods. Then we will explore differentially flat problems, and how these enable the use of algebraic equations in non-linear problem solving. Next we will explore closed-loop control, and some implementations. Finally, we will combine open-loop and closed-loop approaches by introducing trajectory tracking.

3.2 Constrained Optimization

3.2.1 Formalization

Before presenting the optimal control problem it is first useful to briefly review a standard constrained optimization problem given by

$$\begin{aligned} &\min_u \ f(x) \\ &\text{subject to } h_i(x) = 0, \quad i = 1, \dots, m \end{aligned} \tag{3.1}$$

Problem 3.1 is solved by introducing the ***Lagrangian*** and then generating the necessary conditions for optimality (NOC). The Lagrangian is given by

$$L(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i h_i(x)$$

and the necessary conditions for optimality are

$$\begin{aligned} \nabla_x L(x^*, \lambda^*) &= 0 \\ \nabla_\lambda L(x^*, \lambda^*) &= 0 \end{aligned} \tag{3.2}$$

One way to think about the necessary conditions (Eq. 3.2) for optimality is as a filter that takes in all points x and outputs a subset of potential optimal points x^*. If several potential points exist that satisfy the NOC, these could be local optima.

3.2.2 Example

Let's present a constrained optimization problem as an example.
Let $\mathbf{x}$ be $\mathbf{x} = (x_1, x_2)$. The constrained optimization problem is:

$$\begin{array}{l} \min_{\mathbf{x}} = x_1 + x_2 \\ \text{subject to } x_1^2 + x_2^2 = 2 \end{array} \tag{3.3}$$

In other words, we want to find x_1 and x_2 such that $x_1 + x_2$ is minimal and the point $\mathbf{x} = (x_1, x_2)$ belongs to the circle of radius $\sqrt{(2)}$ centered on O, as shown by the figure.

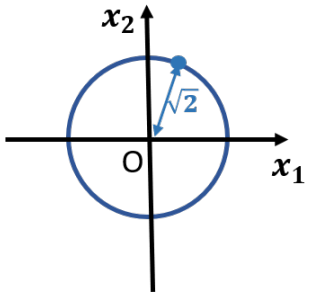

Figure 3.1: A constrained Optimization problem

In this problem,

$$f(\mathbf{x}) = x_1 + x_2$$
$$h(\mathbf{x}) = x_1^2 + x_2^2 - 2$$

The *Lagrangian* function is

$$\begin{array}{rcccc} L(\mathbf{x}, \lambda) & = & f(\mathbf{x}) & + & \lambda h(\mathbf{x}) \\ & = & x_1 + x_2 & + & \lambda(x_1^2 + x_2^2 - 2) \end{array} \tag{3.4}$$

The necessary conditions for optimality are:

$$\begin{array}{rcl} \nabla_{\mathbf{x}} L(\mathbf{x}^*, \lambda^*) & = & 0 \\ \nabla_{\lambda} L(\mathbf{x}^*, \lambda^*) & = & 0 \end{array} \tag{3.5}$$

Here,

$$\nabla_{\mathbf{x}} L(\mathbf{x}^*, \lambda^*) = \begin{bmatrix} \frac{\partial L}{\partial x_1}(\mathbf{x}^*, \lambda^*) \\ \frac{\partial L}{\partial x_2}(\mathbf{x}^*, \lambda^*) \end{bmatrix} = \begin{bmatrix} 1 + 2\lambda^* x_1^* \\ 1 + 2\lambda^* x_2^* \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3.6}$$

and

$$\nabla_{\lambda} L(\mathbf{x}^*, \lambda^*) = x_1^{*2} + x_2^{*2} - 2 = 0 \tag{3.7}$$

Equation (3.6) gives

$$\begin{array}{l} x_1^* = -\frac{1}{2\lambda^*} \\ x_2^* = -\frac{1}{2\lambda^*} \end{array} \tag{3.8}$$

Injecting (3.8) into (3.7) gives

$$\begin{aligned} &\tfrac{1}{4\lambda^{*2}} + \tfrac{1}{4\lambda^{*2}} - 2 = 0 \\ \iff\ &\lambda^* = \pm\tfrac{1}{2} \end{aligned} \tag{3.9}$$

Injecting (3.9) in (3.8):

$$x_1 + x_2 = -2 \text{ if } \lambda^* = \frac{1}{2}$$

and

$$x_1 + x_2 = 2 \text{ if } \lambda^* = -\frac{1}{2}$$

As a result, we choose $\lambda^* = \frac{1}{2}$ for which $x_1 + x_2$ is minimal.

Conclusion: The optimal solution of the constrained problem (3.3) is:

$$\mathbf{x}^* = (-1, -1) \tag{3.10}$$

The solution (3.10) is presented in figure 3.2.

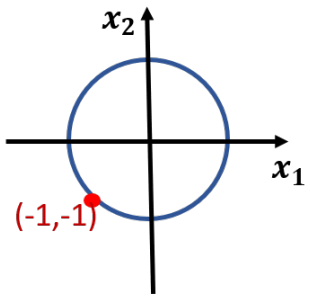

Figure 3.2: Optimal solution of the constrained problem

3.3 Optimal Control Problem

$$\begin{aligned} \min_u\ & h(x(t_f), t_f) + \int_{t_0}^{t_f} g(x(t), u(t), t)dt \\ \text{subject to } & \dot{x}(t) = a(x(t), u(t), t) \\ & x(t) \in \mathcal{X}, \quad u(t) \in \mathcal{U} \end{aligned} \tag{3.11}$$

In general, $x(t) \in R^n$, $u(t) \in \mathbb{R}^m$, and the initial condition $x(0) = x_0$ is given. In the context of this course the constraint $x(t) \in \mathcal{X}$ is typically relaxed to just be $x(t) \in \mathbb{R}^n$. In other words there are no state constraints.

The problem given by Eq. 3.11 contains many details. In the objective function, the term $h(x(t_f), t_f)$ represents a cost associated with the end condition. For example if you wanted to minimize time you could have $h = t_f$. Also in the objective function, the integral produces a path cost, or in other words produces penalties along the way from t_0 to t_f. This would be useful if you wanted to minimize fuel use: for example you could set $g = u^2$.

The first constraint in the problem defines the differential constraints that represent the system dynamics. These in general are nonlinear which is why they are represented as $a(x(t), u(t), t)$.

Problem 3.11 is typically solved to yield an open-loop trajectory and control, which are denoted as $x^*(t)$ and $u^*(t)$. Note that these are entire trajectories through time, not just individual points as we saw in Eq. 3.1. Since the control is open loop we also typically write

$$u^*(t) = f(x(t_0), t)$$

to indicate that the control is a function of time only, and does not utilize any state feedback.

It is important to realize the difference between solving these optimal control problems and solving general optimization problems. As mentioned, in optimal control we are solving for *trajectories*, and not just points. For any given set of boundary conditions there are an infinite number of possible trajectories, and with the dynamics constraints also involved this becomes a formidable task.

The tools used to solve this Problem 3.11 are generally classified as ***Indirect Methods*** and ***Direct Methods***.

3.3.1 Indirect Methods

Given the control inputs and thus the control function (either an open loop or closed loop scenario) we look to optimize over, one can proceed with either a direct or indirect method for solving. A good thought process to follow for an indirect method is to ***"first optimize, then discretize"***. The indirect method works by imposing all NCOs, looking at the optimal functions of time we generated, and then selecting the one that is the best. Take the unconstrained finite dimensional case of a simple function f(x) that we look to optimize over $x \in \mathbb{R}^n$:

$$\min_x \ \nabla_f(x) = 0$$

Taking the gradient of my function and looking at those candidates that survive, one can determine which values would optimize their function (looking to minimize for our case). Our parallel infinite dimensional problem takes the solutions from NCOs as differential equations which should then be discretized to solve accordingly.

After a series of complicated derivations to establish an augmented cost function and referring to section **3.2** on how to generally handle constrained optimization, one can arrive at what is known as the Hamiltonian to solve our functions:

$$H := g(x(t), u(t), t) + p^{\mathbf{T}}(t)[a(x(t), u(t), t] \tag{3.12}$$

where the first paranthetical term denotes the running cost, $p^{\mathbf{T}}(t)$ denotes the vector of Lagrange Mutlipliers (one for each individual constraint),and the final term denoting the RHS of the differential equation. From here we can declare the NCO's that our system requires for optimality:

$$\begin{aligned} \dot{x}^*(t) &= \frac{\partial H}{\partial p} = (x^*(t), u^*(t), p^*(t), t) \\ \dot{p}^*(t) &= -\frac{\partial H}{\partial x} = (x^*(t), u^*(t), p^*(t), t) \\ 0 &= \frac{\partial H}{\partial u} = (x^*(t), u^*(t), p^*(t), t) \end{aligned} \tag{3.13}$$

The first equation represents the derivative of the Hamiltonian with respect to your individual Lagrange Multipliers which interestingly enough gives back the kinematic state equations of the robot. The second equation is called the costate of your function - looking to optimize the function over continuous infinite time, the lagrange multipliers become functions of time. The final equation is the optimality condition which is simply the Hamiltonian differiniated with respect to the constraint producing an algebraic equation. This is **under the assumption that we have no state constraints** which is sufficient for our focus of the class. The tedious part of optimal control is now solving these equations: we have produced 2N optimization variables in $x^*(t)$ and $p^*(t)$ with one M varibale $u^*(t)$ totaling 2N + M variables alongside our 2N differinitable equations and M algebraic equation.

To solve we must impose boundary conditions on our function with respect to the robot's final position and time such that:

$$[\frac{\partial h}{\partial x}(x^*(t_f), t_f - p^*(t_f))]^T \delta x_f \ + [H(x^*(t_f), u^*(t_f), p^*(t_f), t_f + \frac{\partial h}{\partial t}(x^*(t_f), t_f)]\delta x_f = 0 \tag{3.14}$$

which produces four possible solutions to satisfy the equation:

t_f = fixed and $x(t_f)$ = fixed ... $\delta t_f = 0$ and $\delta x(t_f) = 0$

$$\begin{aligned} \textbf{BC} : x^*(t_0) &= x_0 \\ x^*(t_f) &= x_f \end{aligned}$$

t_f = fixed and $x(t_f)$ = free ... $\delta t_f = 0$ and $\delta x(t_f)$ = arbitrary

$$\begin{aligned} \textbf{BC} : x^*(t_0) &= x_0 \\ [\frac{\partial h}{\partial x}(x^*(t_f), -p^*(t_f))] &= 0 \end{aligned}$$

t_f = free and $x(t_f)$ = fixed ... δt_f = arbitrary and $\delta x(t_f) = 0$

$$\begin{aligned} \textbf{BC} : x^*(t_0) &= x_0 \\ x^*(t_f) &= x_f \\ [H(x^*(t_f), u^*(t_f), p^*(t_f), t_f + \frac{\partial h}{\partial t}(x^*(t_f), t_f)]\delta x_f &= 0 \end{aligned}$$

t_f = free and $x(t_f)$ = free ... δt_f = arbitrary and $\delta x(t_f)$ = arbitrary

$$\begin{aligned} \textbf{BC} : x^*(t_0) &= x_0 \\ [\frac{\partial h}{\partial x}(x^*(t_f), -p^*(t_f))] &= 0 \\ [H(x^*(t_f), u^*(t_f), p^*(t_f), t_f + \frac{\partial h}{\partial t}(x^*(t_f), t_f)]\delta x_f &= 0 \end{aligned}$$

The last scenario is what we call a free final time state and introduces an unkown variable to our system of equations making it a set of 2N + M + 1. Think of this as an optimal control problem where we look to minimize the cost of our function at any given final time. We look at a thorough example for clarification.

Let's consider an example of free final time. Consider a point that moves in one dimension. Control input of the system is acceleration of the point. Boundary conditions for positions and velocities at initial and final times are given, but the final time is not fixed.

$$\ddot{x} = u, x(0) = 10, \dot{x} = 0, x(t_f) = 0, \dot{x}(t_f) = 0 \tag{3.15}$$

$$J = \frac{1}{2}\alpha t_f^2 + \frac{1}{2}\int_{t_0}^{t_f} bu^2(t)\mathrm{d}t \tag{3.16}$$

Cost function shown in (3.16) is typical in control problems. The final term in cost function tries to minimize total time to get back to origin, while the stagewise cost attempts to keep acceleration from being too high, which requires a lot of control effort. Analytical solution (3.17) of the final time is pretty reasonable. When b increases, penalty on control effort will be more serious, which causes the acceleration to be relatively low and t_f to be longer. When α increases, the final cost term will become dominant, so t_f will become shorter.

$$t_f = (1800b/\alpha)^{1/5} \tag{3.17}$$

In order to solve the control problem, firstly we need to define state $\mathbf{z}$ and Hamiltonian. From the first equation of (3.15) we have two state constraints (in which $x_1 = x$). Thus we have two Lagrange multipliers p_1 and p_2 correspondingly.

$$\begin{aligned} \dot{x_1} &= x_2 \\ \dot{x_2} &= u \end{aligned} \tag{3.18}$$

According to the trick for free final time, we have a dummy state r that is equal to t_f with dynamics $\dot{r} = 0$. So state $\mathbf{z}$ has five elements.

$$\mathbf{z} = [z_1, z_2, z_3, z_4, z_5]^T = [x_1^*, x_2^*, p_1^*, p_2^*, r^*]^T \tag{3.19}$$

Then we form Hamiltonian H based on (3.16) and (3.18).

$$H = \frac{1}{2}bu^2 + p_1x_2 + p_2u \tag{3.20}$$

We can get Hamiltonian equations from H.

$$\begin{aligned} \dot{x_1^*} &= \frac{\partial H}{\partial p_1}(\mathbf{x}^*(t), u^*(t), \mathbf{p}^*(t), t) = x_2^* \\ \dot{x_2^*} &= \frac{\partial H}{\partial p_2}(\mathbf{x}^*(t), u^*(t), \mathbf{p}^*(t), t) = u^* \\ \dot{p_1^*} &= -\frac{\partial H}{\partial x_1}(\mathbf{x}^*(t), u^*(t), \mathbf{p}^*(t), t) = 0 \\ \dot{p_2^*} &= -\frac{\partial H}{\partial x_2}(\mathbf{x}^*(t), u^*(t), \mathbf{p}^*(t), t) = -p_1^* \\ 0 &= \frac{\partial H}{\partial u}(\mathbf{x}^*(t), u^*(t), \mathbf{p}^*(t), t) = bu^* + p_2^* \end{aligned} \tag{3.21}$$

Boundary conditions include the original ones and substitution for free t_f and fixed $\mathbf{x}(t_f)$ problem.

$$x_1^*(0) = 10, x_2^*(0) = 0, x_1^*(t_f) = 0, x_2^*(t_f) = 0 \tag{3.22}$$

$$\begin{aligned} H(\mathbf{x}^*(t_f), u^*(t_f), \mathbf{p}^*(t_f), t_f) + \frac{\partial h}{\partial t}(\mathbf{x}^*(t_f), t_f) = 0 \\ \frac{1}{2} b u^*(t_f)^2 + p_1^*(t_f) x_2^*(t_f) + p_2^*(t_f) u^*(t_f) + \alpha t_f = 0 \end{aligned} \tag{3.23}$$

When you put $u^* = -\frac{1}{b} p_2^*$ and $x_2^*(t_f) = 0$ into (3.23), you can get a more clear version of the last boundary equation.

$$-\frac{1}{2b} p_2^*(t_f)^2 + \alpha t_f = 0 \tag{3.24}$$

After substitution of $u^* = -\frac{1}{b} p_2^*$ into other Hamiltonian equations in (3.21), we unify the notation of states to $\mathbf{z}$ and rescale time with $\tau = t/t_f$. BVP becomes

$$\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\tau} = t_f \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}t} = z_5 \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\frac{1}{b} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{z} \tag{3.25}$$

After the same operation, BC become

$$\begin{aligned} z_1(0) = 10, z_2(0) = 0, z_1(1) = 0, z_2(1) = 0, \\ -\frac{1}{2b} z_4(1)^2 + \alpha z_5(1) = 0 \end{aligned} \tag{3.26}$$

Then we can input these BVP and BC into Python to solve the problem.

3.3.1.1 Python Example for Indirect Methods - Problem Setup

Let's take a look at the unicycle model from a different perspective, and see how to solve this model in Python. We start with a side-view of a coupled rod-and-wheel system - ie a unicycle with a seat. This problem is derived from the paper by Tum et. al [7].

The dynamics of the system [7] are given as follows:

$$y = \begin{bmatrix} x \\ \dot{x} \\ \theta \\ \dot{\theta} \end{bmatrix}$$

$$\dot{y} = \begin{bmatrix} \dot{x} \\ u \\ \dot{\theta} \\ \frac{1}{I+ml^2}(mgl\sin(\theta) - c\dot{\theta} - ml\cos(\theta)u) \end{bmatrix}$$

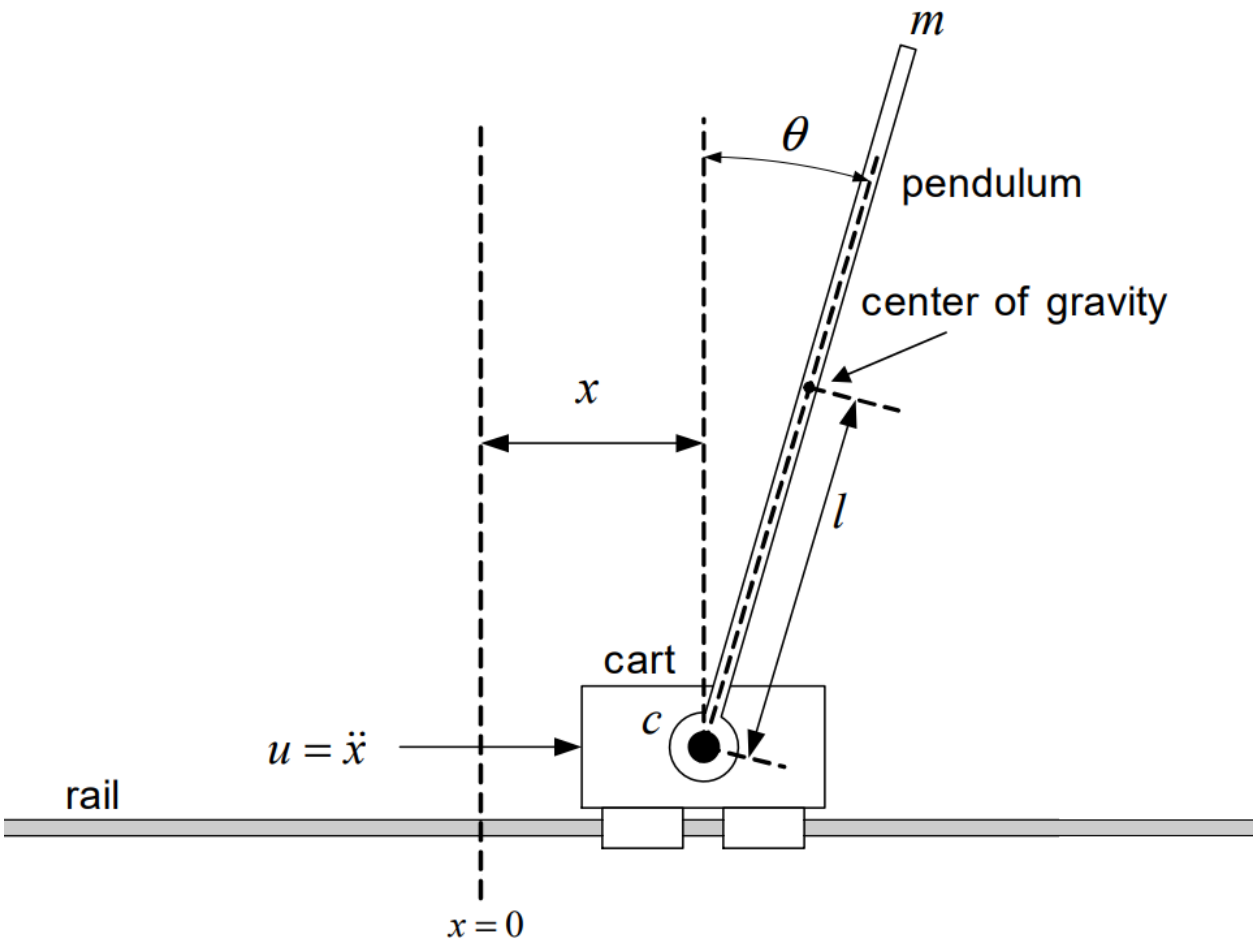

Figure 3.3: Inverse pendulum model [7]. Note that this can work for a unicycle as well, with some additional transformations on the input coordinates. We will ignore those for now and assume we have direct acceleration control of our wheel.

Where I, m, l, c are all known.

Note that since this is a python example, we will spend little time deriving the model. For more information, see [7]. Our functional to minimize will be

$$J = \int_0^{t_f} (600 + u^2) dt$$

Setting our state to be $z = [y, p, t_f]$ with time-rescaling, and using Hamiltonian techniques we get the following system:

$$\frac{dz}{d\tau} = z_9 \begin{bmatrix} z_2 \\ u \\ z_4 \\ \frac{1}{I+ml^2}(mgl\sin(z_3) - cz_4 - ml\cos(z_3)u) \\ 0 \\ -z_5 \\ \frac{-z_8}{I+ml^2}(mgl\cos(z_3) + ml\sin(z_3)u) \\ \frac{z_8 c}{I+ml^2} - z_7 \\ 0 \end{bmatrix}$$

$$u = \frac{1}{2}(\frac{z_8 ml\cos(z_3)}{I+ml^2} - p_2)$$

Finally, we have our boundary conditions: We want to go from a vertical position at the origin to a vertical position at 10 meters. This translates to

$$z_0(0) = z_1(0) = z_2(0) = z_3(0) = 0$$
$$z_0(1) = 10, z_1(1) = z_2(1) = z_3(1) = 0$$

We also get a bonus boundary equation from our free t_f:

$$u(1)^2 + p(1)^T\dot{y}(1) + 600 = 0$$

3.3.1.2 Python Example for Indirect Methods - Code

This code is available in the repo as "inverted_pendulum.py". The crux of the code will revolve around using the scipy bvp_solver package. There are many bvp solvers out there, but in general they all need the following things:

1. An ode function ($\frac{dz}{d\tau}$)
2. A boundary condition function
3. An initial guess

Please view the commented code for a thorough description of the process.

3.3.2 Direct Methods

The alternate way for solving these optimal control problems is through use of what is known as a direct method where instead we carry out the reverse process and rather textit**"first discretize, then optimize"**. We take our same infinite time dimensional problem, discretize it into itemized finite dimensional elements, and then optimize those parts known as a nonlinear programming problem (NLP).

As direct methods are not highlighted in great detail for the extent of this course a brief conceptual overview is provided. Interested students are referred to further courses and literature for further exploration. Two main ways for carrying out a direct method can be observed through what is called either the **Control Parameterization Method** which discretizes the control function over n steps treating the trajectory as continuous or through **State and Parameterization** which discretizes both the state and control functions.

Several software packages including DIDO, Propt, and GPOPS provide excellent tools for solving with these direct method types.

3.4 Differential Flatness

The concept of differential flatness is particularly useful in controlling certain nonlinear systems, especially when explicit trajectory generation is required.

For example, consider the problem of designing a trajectory for a quadrotor (which is a differentially flat nonlinear system). Without leveraging the flatness property, a trajectory designer would have to compute a trajectory for all 12 state variables, position (x,y,z), velocity ($\dot{x}$,$\dot{y}$,$\dot{z}$), angles (ϕ,θ,ψ), and angular rates ($\dot{\phi}$,$\dot{\theta}$,$\dot{\psi}$), the 4 control inputs ($u_{1,2,3,4}$), and also make sure these satisfy the differential equations $\bar{\dot{x}} = f(\bar{x}, u)$.

When utilizing the concept of differential flatness, it turns out that the flat outputs $\bar{z}$ are just the position (x,y,z) and the yaw angle (ψ). Therefore a trajectory designer could design trajectories only for these flat variables, and then algebraically compute the remaining states and controls. Simply put, when a nonlinear system is differentially flat the trajectory designer can plan in the flat output space and then directly map those to the appropriate control inputs without having to worry about violating the dynamics equations for the system!

Mathematically, a non-linear system

$$\dot{x} = f(x, u), \tag{3.27}$$

is differentially flat with flat output z if there exists a function α such that

$$z = \alpha(x, u, \dot{u}, \ldots, u^{(p)}), \tag{3.28}$$

and such that the solutions to the system x and u can then be written as functions of the flat output z and a finite number of its derivatves

$$\begin{aligned} x &= \beta(z, \dot{z}, \ldots, z^{(q)}) \\ u &= \gamma(z, \dot{z}, \ldots, z^{(q)}) \end{aligned} \tag{3.29}$$

From these relations we can also write boundary conditions on the flat outputs that encode the initial and terminal conditions.

$$\begin{aligned} x(0) &= \beta(z(0), \dot{z(0)}, \ldots, z^{(q)}(0)) = x_0 \\ x(T) &= \beta(z(T), \dot{z}(T), \ldots, z^{(q)}(T)) = x_f \end{aligned} \tag{3.30}$$

Therefore any trajectory for z that satisfies these boundary conditions can be mapped to a feasible trajectory for the nonlinear system. This means that z can be defined however the designer chooses, but a common choice is by using a set of smooth basis functions $\psi(t)$

$$z(t) = \sum_{i=1}^{N} \alpha_i \psi_i(t), \quad \alpha_i \in \mathbb{R} \tag{3.31}$$

For example, one choice is to use polynomial basis functions $\psi_1(t) = 1$, $\psi_2(t) = t$, $\psi_3(t) = t^2$, and so on. Choosing a set of smooth basis functions leads to representation that is linear in the coefficients α_i, and thus can be easily identified algebraically.

Taking derivatives of the representation in Eq. 3.31 gives

$$\begin{aligned} \dot{z}(t) &= \sum_{i=1}^{N} \alpha_i \dot{\psi}_i(t) \\ &\vdots \\ z^{(q)}(t) &= \sum_{i=1}^{N} \alpha_i \psi_i^{(q)}(t). \end{aligned} \tag{3.32}$$

Given the initial and final conditions $z(0)$, $\dot{z}(0)$, $\ldots, z^{(q)}(0)$ and $z(T)$, $\dot{z}(T)$, $\ldots, z^{(q)}(T)$ that satisfy the boundary conditions in Eq. 3.30, the coefficients α_i can be found by solving (assuming full rank)

$$\begin{bmatrix} \psi_1(0) & \psi_2(0) & \ldots & \psi_N(0) \\ \dot{\psi}_1(0) & \dot{\psi}_2(0) & \ldots & \dot{\psi}_N(0) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(0) & \psi_2^{(q)}(0) & \ldots & \psi_N^{(q)}(0) \\ \psi_1(T) & \psi_2(T) & \ldots & \psi_N(T) \\ \dot{\psi}_1(T) & \dot{\psi}_2(T) & \ldots & \dot{\psi}_N(T) \\ \vdots & \vdots & & \vdots \\ \psi_1^{(q)}(T) & \psi_2^{(q)}(T) & \ldots & \psi_N^{(q)}(T) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_N \end{bmatrix} = \begin{bmatrix} z(0) \\ \dot{z}(0) \\ \vdots \\ z^{(q)}(0) \\ z(T) \\ \dot{z}(T) \\ \vdots \\ z^{(q)}(T) \end{bmatrix} \tag{3.33}$$

To summarize, for differentially flat nonlinear systems, nominal trajectories and control inputs can can be computed in a more computationally efficient way.

3.5 Closed-loop Control

In the case of closed-loop control, the goal is to find

$$u^*(t) = \pi(x(t), t) \tag{3.34}$$

often using two main techniques: Lyapunov analysis, or the Hamilton-Jacobi-Bellman equation and dynamic programming. The extent of these methods can be seen in classes such as AA203, however in this class we will primarily consider examples of these applications.

3.5.1 Lyapunov proof for stability

One type of stability for a system is known as Lyapunov stability. Lyapunov stability gives an indication of stability of a initial condition near a given equilibrium. Formally, this can be given as, for a autonomous nonlinear dynamical system, given by:

3.5.1.1 Theorem

$$\dot{x}(t) = f(x(t)), x(0) = x_0 \tag{3.35}$$

This system is Lyapunov stable if for any $\epsilon > 0$, there exists a $\delta = \delta(\epsilon)$ such that if $||x(0) - x_e|| < \delta$ then for every t ≥ 0, $||x(t) - x_e|| < \delta$ where x_e is an equilibrium point. [2]

The implication of this is that we are able to determine the stability of a system with respect to its equilibrium points. Systems that are Lyapunov stable will tend to stay "close to" the equilibrium point if a given initial condition satisfies the theorem. This closeness is defined by δ. The full proof for stability can be found in various sources, such as [6].

3.5.2 Example

Let's consider a differential drive example for closed-loop control. The differential drive mobile robot has unicycle dynamics, which can be described in $XY\theta$ coordinates as follows.

$$\begin{aligned} \dot{x}(t) &= V(t)cos(\theta(t)) \\ \dot{y}(t) &= V(t)sin(\theta(t)) \\ \dot{\theta}(t) &= \omega(t) \end{aligned} \tag{3.36}$$

The control inputs of this system are V, which is linear velocity of the wheel and ω, which is angular velocity around the vertical axis. It has been revealed that the system cannot be controlled in a closed loop with continuous time and variant feedback. In particular, if you set control inputs all to zero, the system will be stopped immediately. This feature has some advantages from a control standpoint, but clearly it is not realistic since no dynamical systems can be stopped immediately. However, many systems in real world can be adjusted model of this version.

3.5.3 Polar Coordinates

It is actually easier to design a closed-loop controller based on polar coordinates than $XY\theta$ coordinates. The polar coordinates are established according to desired state of the unicycle. Origin is at the desired position and X_N axis is along the main axis of the robot. The position is defined as center of the rear axis of the vehicle. The main axis is defined as the longitudinal axis of the robot.

We describe states of the robot by polar coordinates ρ, α and δ instead of X, Y and θ:

- ρ: distance of the reference point of the unicycle from the goal
- α: angle of the pointing vector to the goal with respect to the unicycle main axis

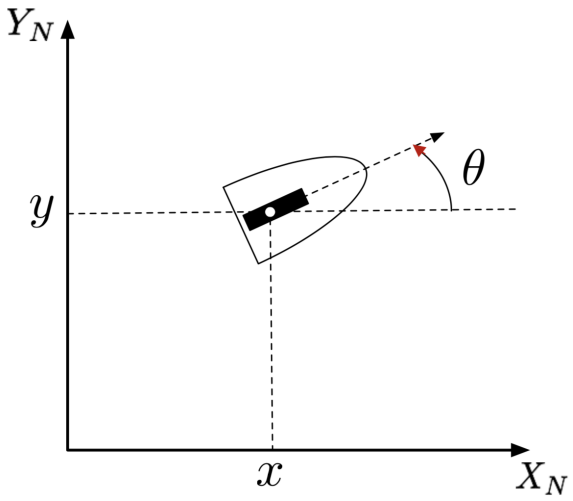

Figure 3.4: $XY\theta$ Coordinates

- δ: angle of the same pointing vector with respect to the X_N axis

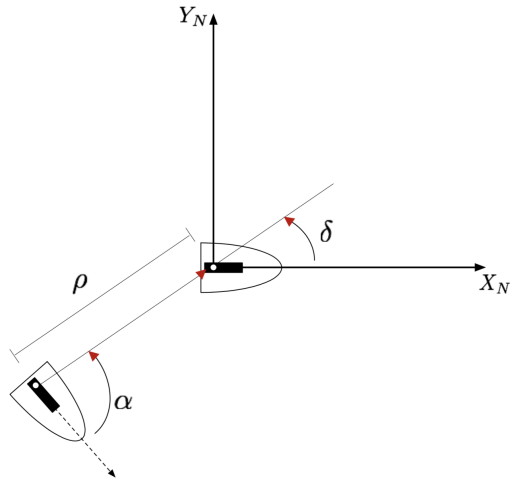

Figure 3.5: Polar Coordinates

Coordinate transformation between $\rho\alpha\delta$ and $XY\delta$ coordinates:

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2} \\ \alpha &= \text{atan2}(y, x) - \theta + \pi \\ \delta &= \alpha + \theta \end{aligned} \tag{3.37}$$

Let's derive dynamic equations of the unicycle in polar coordinates. Say current position of the unicycle is at point P and the origin is at point O. $\overrightarrow{OP}$ is position vector of the unicycle. The control input ω cannot

move the position, so $\dot{\rho}$ comes from the other input V.

$$\begin{aligned} \dot{\rho} &= \vec{V} \cdot \overrightarrow{OP} / ||\overrightarrow{OP}|| \\ \dot{\rho} &= V\cos(\pi - \alpha) = -V\cos\alpha \end{aligned} \tag{3.38}$$

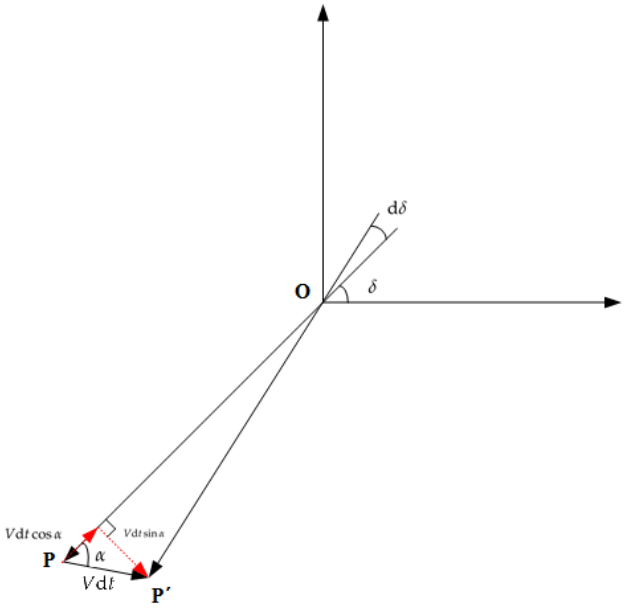

Figure 3.6: Polar Coordinates

$\dot{\delta}$ also comes from only the control input of V because δ is only related to the position of the vehicle. Consider that the vehicle moves for an infinitesimal time period $\mathrm{d}t$. Displacement of the vehicle is $V\mathrm{d}t$. We can know expression of $\mathrm{d}t$ from the figure 3.6 that

$$\begin{aligned} \rho \mathrm{d}\delta &= V\mathrm{d}t\sin\alpha \\ \dot{\delta} &= \frac{\mathrm{d}\delta}{\mathrm{d}t} = \frac{V\sin\alpha}{\rho} \end{aligned} \tag{3.39}$$

Expression of $\dot{\alpha}$ can be directly derived from (3.36), (3.37) and (3.39).

$$\begin{aligned} \dot{\alpha} &= \dot{\delta} - \dot{\theta} \\ \dot{\alpha} &= \frac{\mathrm{d}\delta}{\mathrm{d}t} = \frac{V\sin\alpha}{\rho} - \omega \end{aligned} \tag{3.40}$$

Combination of (3.38), (3.39)) and (3.40) is the dynamic equations of the unicycle in polar coordinates.

$$\begin{aligned} \dot{\rho}(t) &= -V(t)\cos(\alpha(t)) \\ \dot{\alpha}(t) &= \frac{V(t)\sin(\alpha(t))}{\rho(t)} - \omega(t) \\ \dot{\delta}(t) &= \frac{V(t)\sin(\alpha(t))}{\rho(t)} \end{aligned} \tag{3.41}$$

In order to achieve the goal posture, variables (ρ, α, θ) should all converge to zero.

3.5.4 Control Law

For the unicycle problem proposed above, a suitable closed-loop control law is given by:

$$\begin{aligned} V &= k_1 cos(\alpha) \\ \omega &= k_2\alpha + k_1 \frac{sin(\alpha)cos(\alpha)}{\alpha}(\alpha + k_3\delta) \end{aligned} \tag{3.42}$$

These equations are chosen by selecting V and ω which result in the Lyapunov functions being negative semi-definite. It is possible to prove that when this is the case, the state variables will go to zero if the controller has been properly implemented.

A negative semi-definite form of the Lyapunov function is necessary because we want our Lyapunov function to always be decreasing in time, as this suggests that it is tending towards an equilibrium point rather than going further away. The key here is to consider the Lyapunov function as one form of "energy" of the system, which decreases to a minimum at equilibrium points.

A more detailed derivation for this result can be found in [1].

3.6 Trajectory Tracking

A method for trajectory tracking consists in compromising closed-loop and open-loop control. In fact, open-loop control methods are more computationally feasible to find, yet as we have seen they are not robust. On the other hand, closed-loop solutions can be robust but are more complicated to find since the necessary conditions for optimality are partial differential equations, instead of ordinary differential equations in the open-loop case.

We can formulate a solution as:

$$u^*(t) = u_d(t) + \pi(x(t), x(t) - u_d(t)) \tag{3.43}$$

where $u_d(t)$ is a nominal trajectory found using open-loop control techniques. We use closed-loop control techniques to find $\pi(x(t), x(t) - u_d(t))$ that keeps our trajectory as close as possible to the nominal one.

There exists a certain number of choices of techniques for trajectory tracking, among which:

- Geometric strategies:

  An example is pursuit control. An example of strategy for the robot is to constantly try to chase a point on the desired trajectory. This method is effective, but it is difficult to provide theoretical guarantees.

- Linearization techniques:

  They can be separated in two:

  1. Inexact linearization:

     The dynamics of a system are usually non-linear. Yet, in the case of tracking, if we are close

enough to the nominal trajectory then the relative dynamics can be linearized. The inexactness of this method comes from the fact that we only use the first terms of the relative dynamics' Taylor Series expansion to find a linear control law.

2. Exact linearization:

   This method consists in cancelling the non-linearities from the system's dynamics, and developing a tracking control law from the resulting linear system. More precisely:

   - Let's consider the system dynamics: $\dot{x} = f(x) + u$, with f non linear.
   - We can set a virtual control input v so that $u = v - f(x)$. Then v is linear, and we obtain $\dot{x} = v$.
   - We then design v so that x converges to the origin. Our actual control input is therefore $v - f(x)$.

   Not all systems admit this linearization. Even for those who do, one should be careful in using this method since it relies on the exactness of our non-linear model f. In fact, if f differs consequently from the real system dynamics, the non-linearities are not cancelled with this method and the consequences of using the resulting control law could be disastrous.

- Optimization based techniques:

  This consists in posing the problem of trajectory tracking as an optimal control problem. These techniques are thoroughly studied in AA203.

References

[1] Aicardi, Casalino, Bicchi, Balestrino. *Closed loop steering of unicycle like vehicles via Lyapunov techniques*. IEEE, Genoa, Italy, 1995.

[2] Wang, Zhanshan, Liu, Zhenwei, Zheng, Chengde *Qualitative Analysis and Control of Complex Neural Networks with Delays*. Springer-Verlag Berlin Heidelberg, 2016.

[3] Richard M Murray, *Optimization-based control*. California Institute of Technology CA, 2009.

[4] D. K. Kirk. *Optimal Control Theory: An introduction*. Dover Publications, 2004.

[5] Ascher, U., & Russell, R. D. *Reformulation of boundary value problems into "standard" form*. SIAM review, 23(2), 238-254, 1981.

[6] Notes from ee363 at Stanford University *https://stanford.edu/class/ee363/lectures/lyap.pdf*

[7] M. Tum, G. Gyeong, J. Park and Y. Lee, "Swing-up Control of a Single Inverted Pendulum on a Cart With Input and Output Constraints", *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics*, vol. 1, no. 1, pp. 475-482, 2014.

Contributors

Winter 2019: Michael Kossyrev
Winter 2018: Joseph Lorenzetti, Kenneth Hoffmann, Oriana Peltzer, Zhe Huang, William Mangram