Personal Lecture Notes
MITx6.832x: Underactuated Robotics
(Spring 2019)

Algorithms for Walking, Running, Swimming, Flying, and Manipulation

Julian Eßer

March 10, 2020

Contents

Chapter 1

Introduction

The purpose of this document is to provide a brief overview of the essential knowledge of the underactuated robotics class [**?**] from MIT taught in Spring 2019. Special emphasis is on the following topics:

- Understanding Control as Optimization
- Dynamics of Biped Locomotion
- Optimization of Biped Locomotion

Chapter 2

Lecture 1: Why Study Robot Dynamics?

2.1 Background / Motivation

The **motivation** for this course is to

- Build great robots that can do amazing things
- Exploit natural dynamics of robots, not just doing dump control
- Achieve extraordinary performance in terms of speed, efficiency, or robustness (Honda's ASIMO vs. passive dynamic walkers)
- Controlling nonlinear systems without complete control authority
- View computation of challenging tasks in robotics (manipulation, autonomous driving) through the lense of dynamics.

This course is all about nonlinear dynamics and control of underactuated mechanical systems, with an emphasis on computational methods. Especially it covers the **topics**

- Nonlinear dynamics
- Applied optimal and robust control
- Motion planning
- Examples from biology and applications to legged locomotion, compliant manipulation, underwater robots, and flying machines

2.2 Definitions

Nonlinear differential equations typically take the form

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

where f is a vector valued function, x is the state vector and u is the vector of control input and $\dot{x} = \frac{dx}{dt}$ is the time derivative. Mechanical Systems are described by second order differential equations. When the state vector is defined as

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix},$$

where the system dynamics can be described as

$$\ddot{q} = f(q, \dot{q}, u).$$

Since mechanical systems are *control affine*, this specializes to

$$\ddot{q} = f_1(q, \dot{q}) + f_2(q, \dot{q})u.$$

A system of this form is called *underactuated* if $rank[f_2] <= n$. Other causes of underactuated include

- Input saturation (e.g. torque limits)
- State constraints (e.g. joint limits)
- Model uncertainty / state estimation

2.3 Manipulator Equations

The equations of motion for simple systems, e.g. a double pendulum, are quite simple to derive. Results, e.g. obtained from an Lagrangian calculation approach, can be expressed in the form of the standard "manipulator equations":

$$M(q)\ddot{q} + C(q, \dot{q})\ddot{q} = \tau_g(q) + Bu$$

where M is the Inertia matrix, C is the matrix of Coriolis terms τ_g covers gravitational torques, B maps inputs to generalized force and u is the control input (either force or torque).
The acceleration then is expressed as

$$\ddot{q} = M^{-1}(q)[\tau_g(q) + Bu - C(q, \dot{q})\dot{q}]. \tag{2.1}$$

With equation 2.1, the dynamics of the systems and accordingly the functions f_1 and f_2 are fully defined.
For simulating the dynamics of a robot, it is sufficient to provide the kinematics in form of a *URDF file*, pass it to an forward Dynamics solver and you get the resulting acceleration and its integrations.

2.4 Plan for the Course

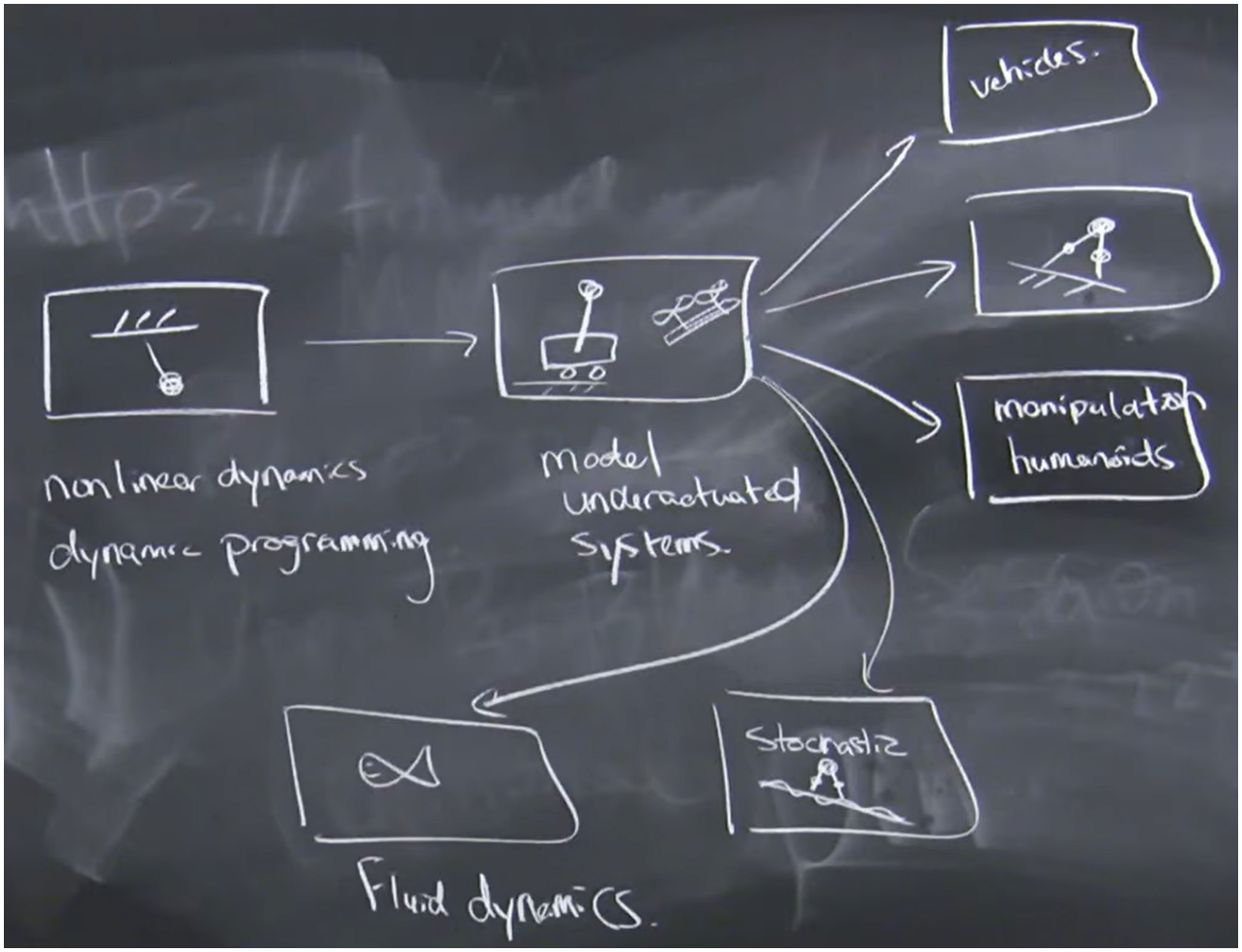

Figure 2.1: Course overview: Basics first, then simple systems and then various advanced systems.

Chapter 3

Lecture 2: Nonlinear Dynamics

3.1 The Simple Pendulum

Even the most simple dynamical systems, e.g. the simple pendulum, can not be solved in a closed form. This is due to the nonlinear characteristics of the underlaying differential equations. But actually you don't have to in order to describe and analyse fundamental dynamical characteristics.
But what we really care about is the long-term behaviour of the sytem. For low dimensional systems, two central tools are available in order to analyse the systems behaviour:

- Linearization
- Graphical Analysis

The equations of motion of the simple pendulum can be derived with the Langrangian as:

$$ml^2\ddot{\theta}(t) + mgl\sin\theta(t) = Q$$

Considering the generalized force Q as combination of damping and a control torque input

$$Q = -b\dot{\theta}(t) + u(t).$$

For the case of a constant torque this yields

$$ml^2\ddot{\theta} + b\dot{\theta} + mgl\sin\theta = u_0.$$

3.2 Graphical Analysis

3.2.1 Fixed Points

The central goal of control is to meaninfully shift the vector field via sophisticated control input of a system in order to change its dynamics. So called *phase plots* are useful for visualizing this vector field of two-dimensional systems. In case of the state vector $x = \begin{bmatrix}\theta\\ \dot{\theta}\end{bmatrix}$ this means $\dot{\theta}$ over θ.

Definition 1 *A point the system will remain forever without applying external forces is called a fixed point or a steady state respectively.*

The position of fixed points, e.g. stable positions of the pendulum, strongly depend on the parameter of the system (damping, input torque etc.).

3.2.2 Definitions of Stability

There are existing different types of stability in order to describe the behaviour next a fixed point $x^\star$. The fixed point can be

- *Stable* in the sense of Lyapunov (i.e. will remain within certain radius)
- *Asymptotically stable* (i.e. for $t- > \infty$ reaches certain point)
- *Exponentially stable* (reaches certain point at defined rate).

Chapter 4

Lecture 3/4: Dynamic Programming

4.1 Control as Optimization

- The big idea is to formulate control design as an optimization problem.
- Given a trajectory $x(.), u(.)$ we want to assign a score (scalar) to decribe the performance.
- Additionally we can set constraints in order to exclude trajectories that exceed certain limits (e.g. control limit $|u(t)| <= 1$).
- The goal is to find a control policy $u = \Pi(t, x)$ that optimizes that score!
- When solving optimal control problems, one most often needs numerical approximation.

The strengths of Optimal Control are that it

- is a very general approach; it can be applied to fully/underactuated, linear/nonlinear systems,
- contains an very intuitive approach by describing just the goal and some constraints
- works very well with numerical approximation.

4.2 Example: Double Integrator

- Very simple example that can be solved without numerical approximation.
- Consists of "brick of ice" on a flat floor
- Goal: Go to origin as fast as possible

Easiest case: Formulate optimal control as "Bang-Bang". Accelerate (full throttle) then slam on the brakes.

4.3 DDP: Discrete Time Space - Optimal Control as Graph Search

For systems with a finite, discrete set of states and a finite, discrete set of actions, dynamic programming also represents a set of very efficient numerical algorithms which can compute optimal feedback controllers.
Cost function

$$one - stepcost : g(s, a)$$

$$totalcost : \sum_{n=0}^{\infty}$$

Key idea: Additive cost

$$\int_0^T \ell(x(t), u(t))dt,$$

There are existing numerous possibilities on how to design the cost function:

- Min-time: $g(s, a) = 1"if"s \ = s_{goal}; 0 if otherwise$
- Quadratic cost: $g(x, u) = x^T x + u^T u$

There are many algorithms for finding (or approximating) the optimal path from a start to a goal on directed graphs. In dynamic programming, the key insight is that we can find the shortest path from every node by solving recursively for the optimal cost-to-go (the cost that will be accumulated when running the optimal controller) from every node to the goal. Recursive form of the optimal control problem:

$$\hat{J}^*(s_i) \Leftarrow \min_{a \in A} \left[\ell(s_i, a) + \hat{J}^* \left(f(s_i, a) \right) \right] \tag{4.1}$$

If we know the optimal cost-to-go, then it's easy to extract the optimal policy:

$$\pi^*(s_i) = argmin_a \left[\ell(s_i, a) + J^* \left(f(s_i, a) \right) \right]. \tag{4.2}$$

Limitations:

- Accuracy for continuous systems (discretisation error)
- Scaling (curse of dimensionality)
- Assumes full state information: Absolutely not necessary to know everything

4.4 DDP: Continuous Time Space

4.4.1 The Hamilton-Jacobi-Bellman Equation

An analogous set of conditions can be found in the continuous time space. For a system

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

and an infinite-horizon additive cost

$$\int_0^{\infty} l(x, u)dt$$

we have

$$0 = min_u \left[l(x, u) + \frac{\delta J^*}{\delta x} f(x, u) \right]$$

$$\Pi^\star = argmin_u \left[l(x, u) + \frac{\delta J^*}{\delta x} f(x, u) \right]$$

Chapter 5

Lecture 5-7: Acrobots, Cart-poles, and Quadrotors

5.1 Introduction

So far we have covered the following topics:

- Manipulator Equations
- Feedback Linearization
- Optimal Control
- Value Iteration (Algorithm for DDP in discrete time)

After introducing basics of "classic" non-linear control, we started thinking about control as optimization.

In this chapter the most simple standard models for underactuated robots are introduced. These low-dimensional systems are supposed to capture the essence of the problem without all the real-world complexity of advanced systems.

5.2 System Dynamics: Manipulator Equations

The Acrobot is a simple underactuated system since it has two DoF. But, in comparison to the double pendulum, it only has one actuator at the elbow so that $\boldsymbol{B} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$. Manipulator Equations:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \tau_g(\boldsymbol{q}) + \boldsymbol{B}\boldsymbol{u}.$$

The goal is to swing-up and balance while satisfying some torque limits. One possible approach to solve this problem is using *value iteration*. But the grids would have to be very fine, in order to get a good solution. There are better tools to solve this problem: LQR!

5.3 Balancing for Acrobot and Cart-Pole

For both the Acrobot and the Cart-Pole systems, we will begin by designing a linear controller which can balance the system when it begins in the vicinity of the unstable fixed point. To accomplish this, we will linearize the nonlinear equations about the fixed point, examine the controllability of this linear system, then using linear quadratic regulator (LQR) theory to design our feedback controller.

What we'll do to accomplish the balancing:

1. Linearizing the manipulator equations
2. Check controllability of linear systems
3. Use LQR to design a feedback controller

5.3.1 Recap: LQR

We have a linear time-invariant system in state-space form

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u},$$

the cost function is

$$J = \int_0^\infty [\boldsymbol{x}^T \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T \boldsymbol{R}\boldsymbol{u}]dt,$$

and the goal is to find the optimal cost-to-go function $J^\star(\boldsymbol{x})$ which satisfies the Hamilton-Jacobi-Bellman equation. This yields

$$J^\star(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{S}\boldsymbol{x}$$

and the optimal control policy

$$\boldsymbol{u}^\star = \boldsymbol{K}\boldsymbol{x}.$$

In the end, this means you get the control policy and the cost-to-go function by

$$\boldsymbol{K}, \boldsymbol{S} = \mathit{LinearQuadraticRegulator}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{Q}, \boldsymbol{R}).$$

So you set $\boldsymbol{A}, \boldsymbol{B}$ from linearization and choose $\boldsymbol{Q}, \boldsymbol{R}$ and receive an optimal controller.

5.3.2 Linearization of Nonlinear Systems

Problem: Our systems are non-linear! How shall we apply Linear-Quadratic Control?!

Solution: We linearize our system around a specific fixed point.

But we need to be aware that our linearization only is valid within a certain area around this point. If you go to far away from it, the non-linearity overwhelms your solution.

One Optimal Control Algorithm therefore is to combine multiple LQR Controllers that treat all relevant fixed points in order to handle the relevant workspace.

5.4 Throwback: Hand-Designed Control

- Optimal control is a powerful framework for solving control problems via optimization.
- Solving OC problems for non-linear systems is hard!
- Sometimes we would be happy to just have any controller (And sometimes they turn out to be even better).
- But how can we proof this "hand-designed" are any good?

5.5 Partial Feedback Linearization (Acrobot,Cart-pole)

Although we cannot always simplify the full dynamics of the system, it is still possible to linearize a portion of the system dynamics. The technique is called partial feedback linearization.

- *Collocated* PFL: A controller which linearizes the dynamics of the *actuated* joints
- *Non-Collocated* PFL: A controller which linearizes the dynamics of the *unactuated* joints

One of the most important lessons from partial feedback linearization, is the idea that if you have m actuators, then you basically get to control exactly m quantities of your system.
-> We proof that they are stable.

5.6 Swing-Up Control: Energy Shaping (Acrobot,Cart-pole)

If we seek to design a nonlinear feedback control policy which drives the simple pendulum from any initial condition to the unstable fixed point, a very reasonable strategy would be to use actuation to regulate the energy of the pendulum to place it on this homoclinic orbit, then allow the system dynamics to carry us to the unstable fixed point.
This idea turns out to be a bit more general than just for the simple pendulum. As we will see, we can use similar concepts of 'energy shaping' to produce swing-up controllers for the acrobot and cart-pole systems. It's important to note that it only takes one actuator to change the total energy of a system.
The basic Idea for the swing-up control is to

1. Use collocated PFL to simplify the dynamics
2. Use energy shaping to regulate the pendulum to its homoclinic orbit
3. Add a few terms to make sure that the cart stays near the origin

5.7 Differential Flatness (Quadrotors)

The task we'll consider for quadrotors is trajectory optimization:
How can you find a feasible trajectory through state space for the quadrotor, even if there are obstacles to avoid that are only known at runtime?
Trajectory design, and especially trajectory optimization, is a big idea that we will explore more thoroughly later in the text. But there is one idea that I would like to present here, because in addition to being a very satisfying solution for quadrotors, it is philosophically quite close to the idea of partial feedback linearization. That idea is called differential flatness.

- Similar idea as PFL: Given a trajectory of m (number of actuators) coordinates.
- Then, the control input and all left states can be guessed
- Condition: Trajectory needs to be four-times differentiable

2D-Quadrotor Example (m=2): Given x,y of trajectory, guess resulting jaw and control input.
3D-Quadrotor Example (m=4): Given x,y,z,jaw guess roll, pitch and control for all motors.

Chapter 6

Lecture 8/9: Lyapunov Analysis

Optimal control provides a powerful framework for formulating control problems using the language of optimization. But solving optimal control problems for nonlinear systems is hard! In many cases, we don't really care about finding the optimal controller, but would be satisfied with any controller that is guaranteed to accomplish the specified task. In many cases, we still formulate these problems using computational tools from optimization, and in this chapter we'll learn about tools that can provide guaranteed control solutions for systems that are beyond the complexity for which we can find the optimal feedback.

6.1 Lyapunov Functions

6.1.1 Optimization Crash Course

- Goal: Find x_{min} for an objective (cost) function considering a set of (in/equality-) constraints
- Popular objective functions:
  - Convex quadratic cost (least squares)
  - Linear cost
- Convex optimization builds on convex cost functions or a convex set

6.1.2 Introduction

Lyapunov functions generalize the notion of an energy function to more general systems, which might not be stable in the sense of some mechanical energy. If I can find any positive function, call it$V(\boldsymbol{x})$, that gets smaller over time as the system evolves, then I can potentially use V to make a statement about the long-term behavior of the system. $V$is called a Lyapunov function.

6.2 Lyapunov Analysis with Convex Optimization

In this section, we'll look at some computational approaches to verifying the Lyapunov conditions, and even to searching for (the coefficients of) the Lyapunov functions themselves.

6.2.1 Lyapunov Analysis for Linear Systems

Imagine you have a linear system $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, and can find a Lyapunov function

$$V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P}\boldsymbol{x}, \boldsymbol{P} = \boldsymbol{P}^T > 0,$$

which also satisfies

$$\dot{\boldsymbol{V}}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{P}\boldsymbol{A}\boldsymbol{x} + \boldsymbol{x}^T \boldsymbol{A}^T Px < 0.$$

Then the origin is globally asymptotically stable.

6.2.2 Lyapunov Analysis as a Semi-definite Program (SDP)

Lyapunov analysis for linear systems has an extremely important connection to convex optimization. In particular, we could have also formulated the Lyapunov conditions for linear systems above using semi-definite programming (SDP). Semidefinite programming is a subset of convex optimization – an extremely important class of problems for which we can produce efficient algorithms that are guaranteed find the global optima solution

6.2.3 Sums-of-squares Optimization

It turns out that in the same way that we can use SDP to search over the positive quadratic equations, we can generalize this to search over the positive polynomial equations.

6.3 Lyapunov Analysis for Estimating Regions of Attraction

There is another very important connection between Lyapunov functions and the concept of an invariant set: any sub-level set of a Lyapunov function is also an invariant set. This gives us the ability to use sub-level sets of a Lyapunov function as approximations of the region of attraction for nonlinear systems.

Now we have arrived at the tool that I believe can be a work-horse for many serious robotics applications. Most of our robots are not actually globally stable (that's not because they are robots – if you push me hard enough, I will fall down, too), which means that understanding the regions where a particular controller can be guaranteed to work can be of critical importance.

Sums-of-squares optimization effectively gives us an oracle which we can ask: is this polynomial positive for all x? To use this for regional analysis, we have to figure out how to modify our questions to the oracle so that the oracle will say "yes" or "no" when we ask if a function is positive over a certain region which is a subset of R. That trick is called the S-procedure. It is closely related to the Lagrange multipliers from constrained optimization, and has deep connections to "Positivstellensatz" from algebraic geometry.

Chapter 7

Lecture 10: Trajectory Optimization

7.1 Recap: What Did We Cover so Far?

The **overall goal** is to specify complex behaviors with simple objective functions, letting the dynamics and constraints on the system shape the resulting feedback controller.
But the computational tools that we've provided so far have been **limited** in some important ways:

- **Dynamic programming** involves putting a mesh over the state space

  -> Stuck in low-dimensional systems.

- **LQR** + **Linearization** around an operating point; applicable to high-dimensional systems

  -> Linearization only valid for a certain region of the state space

- **Lyapunov Analysis via SDP/SOS** softens the requirements by not searching for the optimal controller, but only searching for stability. It does so for non-linear systems and all $\boldsymbol{x}$ (whole state-space).

  -> No controller actually synthesised + somehow limited to simple Lyapunov functions (quadratic, polynomials etc) which might limit the control design

But we have not yet provided any real computational tools for approximate optimal control that work for high-dimensional systems beyond the linearization around a goal. That is precisely the goal for this chapter.
Q: How can we handle complex systems?
A: We ask not for all states (like in Lyapunov Analysis), instead only the relevant ones!

- For the beginning: Consider only one single initial condition!
- Maybe some x_0 in the neighbourhood are also good
- Represent the solution as a trajectory, $\boldsymbol{x}(t), \boldsymbol{u}(t)$, typically defined over a finite interval (instaed of feedback control function)
- **This means:**
  1. We define only a single trajectory $\boldsymbol{u}(t)$,
  2. Search for optimal states $\boldsymbol{x}$ and control inputs $\boldsymbol{u}$,
  3. To follow this trajectory as good as possible!

7.2 Problem Formulation

- Min over a finite trajectory over time $\boldsymbol{u}(\cdot)$
- Initial conditions $x(0)$ are fixed and known

$$\begin{aligned} \min_{\boldsymbol{u}(\cdot)} \quad & \int_{t_0}^{t_f} \ell(\boldsymbol{b}(t), \boldsymbol{b}(t)) dt \\ \text{subject to} \quad & \forall t, \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)), \\ & \boldsymbol{x}(t_0) = x_0 \end{aligned}$$

7.3 Computational Tools for Nonlinear Optimization

Intro

7.4 Trajectory Optimization as a Nonlinear Program

As written above, the optimization above is an optimization over continuous trajectories. In order to formulate this as a numerical optimization, we must parameterize it with a finite set of numbers.

Different approaches on how to do this, are:

- Idea 1). **Direct Transcription**: Fix breakpoints at even intervals dt and use Euler integration. Decision variables are x, n
- Idea 2.) **Direct Shooting Methods** - Idea: Destrict decision variables to only u and and compute x ourselves by knowing x_0 and $u(\cdot)$ via the simple forward dynamics

$$x[t = 0] = x_0$$

$$x[1] = Ax_0 + Bu[0]$$

$$x[2] = A \cdot (x[1] = Ax_0 + Bu[0]) + Bu[0]$$

$$\ldots$$

- Idea 3.) **Direct Collocation:** Assume first-order polynomial for $\boldsymbol{u}(t)$ and cubic polynomial $\boldsymbol{x}(t)$.

7.5 Pontryagins's Minimum Principle

7.6 Trajectory Stabilization: Local Trajectory Feedback Design

What we want is to move robots in the real world. Therefore it is not useful to have only one exact trajectory, but we need to allow the robot to move in a **band around our target trajectory**.

Idea: Locally linearize around our points from the trajectory, so that we can apply tools from linear control again.

7.6.1 Linear Model-predictive Control

Locally stabilizes a **constrained** system.

7.6.2 Time-varying LQR

Locally stabilizes $\boldsymbol{x}_0(t), \boldsymbol{u}_0(t)$

7.7 Iterative LQR

Chapter 8

Lecture 12/13: Simple Models of Walking and Running

In this chapter we'll introduce some of the simple models of walking and robots, the control problems that result, and a very brief summary of some of the control solutions described in the literature. Compared to the robots that we have studied so far, our investigations of legged locomotion will require additional tools for thinking about limit cycle dynamics and dealing with impacts.

8.1 Limit Cycles

In many of the systems that we have studied so far, we have analyzed the stability of a fixed-point, or even an (infinite-horizon) trajectory. For walking systems the natural equivalent is to talk about the stability of periodic solutions – a fixed "gait" is a cycle that repeats footstep after footstep. So we begin our discussion with a discussion of the stability of a cycle. A limit cycle is an asymptotically stable or unstable periodic orbit. One of the simplest models of limit cycle behavior is the Van der Pol oscillator.

8.1.1 Poincare Maps

8.2 Simple Models of Walking

8.2.1 The Rimless Wheel

The most elementary model of passive dynamic walking, first used in the context of walking by, is the rimless wheel. This simplified system has rigid legs and only a point mass at the hip as illustrated in the figure above. To further simplify the analysis, we make the following modeling assumptions

- No slip
- Collisions are inelastic and impulsive (no bouncing)
- No double support

8.2.2 The Compass Gait

8.3 Simple Models of Running

There are existing various definitions of running:

- Existence of an Aerial Phase
- Exchange of Energy

Why do we study simple models?

- Tractable
- Mechanical Insights
- Comparative Biology: Fundamental Principles?
- As a 'Template' for Higher-DOF Robots

8.3.1 The Spring-Loaded Inverted Pendulum

Assumptions

- Masless leg -> command θ instantaneously
- Perfectly **elastic** collision -> Energy is always conserved (thread to stability)
- When the foot is on ground, we have a pin joint i.e. infinite friction (no sliding)

SLIP Modeling

The model is a point mass, m , on top of a massless, springy leg with rest length of l_0, and spring constant k. The state of the system is given by x, y the position of the center of mass, and the length,l , and angle θ of the leg. Like the rimless wheel, the dynamics are modeled piecewise - with one dynamics governing the flight phase, and another governing the stance phase.
Flight Phase. State variables: $\boldsymbol{x} = [x, y, \dot{x}, \dot{y}]^T$. Dynamics are

$$\dot{\boldsymbol{x}} \begin{bmatrix} \dot{x} \\ \dot{y} \\ 0 \\ -g \end{bmatrix}.$$

Stance Phase. State variables: $\boldsymbol{x} = [r, \theta, \dot{r}, \dot{\theta}]$. Kinematics are

$$x = \begin{bmatrix} -r\sin\theta \\ r\cos\theta \end{bmatrix}.$$

Energy is given by

$$T = \frac{m}{2}(\dot{r}^2 + r^2\dot{\theta}^2), \quad U = mgr\cos\theta + \frac{k}{2}(r_0 - r)^2.$$

Putting these into Lagrange yields:

$$m\ddot{r} - mr\dot{\theta}^2 + mg\cos\theta - k(r_0 - r) = 0 \tag{8.1}$$

$$mr^2\ddot{\theta} + 2mr\dot{r}\dot{\theta} - mgr\sin\theta = 0 \tag{8.2}$$

SLIP Control

Choose $\theta_{touchdown}$ during aerial phase.
Goal: Design controller $u[n] = \Pi(y[n])$ to stabilize y^d.

- Idea 1: find $u\star$ st. $y^\star = P(y^d, u^\star)$.
  Linearize P around (y^d, u^p) + do (discrete time) **LQR**.
  Results in an exponential convergence to the fixed point.
- Idea 2: Deadbeat Control.
  If P is invertible,
  $$u[n] = P^{-1}(y^d, y[n])$$
  If exists, results in an convergence to the fixed-point via one timestep.

8.3.2 Continuous Control: The Planar Monopod Hopper

- Hopping Height (push at toe-off)
- Foot Touchdown to regulate speed
- Stabilize attitude during stance

8.4 A Simple Model That Can Walk and Run

Chapter 9

Lecture 14/15: Computational Tools for Legged Robots - i.e.: Planning and Control through Contact

So far we've discovered the following tools:

- Fixed Points / Local Stability
- Local Stabilization (e.g. LQR)
- Lyapunov Analysis
- Trajectory Optimization

These tools work out for "smooth" systems where the equations of motion are described by a function $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$ which is smooth everywhere. But our discussion of the simple models of legged robots illustrated that the dynamics of making and breaking contact with the world are more complex – these are often modeled as **hybrid dynamics** with **impact** discontinuities at the collision event **and constrained dynamics** during contact (with either soft or hard constraints).

The goal of this chapter is to extend our computational tools into this richer class of models. Many of our core tools still work: trajectory optimization, Lyapunov analysis (e.g. with sums-of-squares), and LQR all have natural equivalents.

9.1 Modeling Contacts

We can model the robot in **floating-base coordinates** – we add a fictitious six degree-of-freedom "floating-base" joint connecting some part of the robot to the world (in planar models, we use just three degrees-of-freedom. We can derive the equations of motion for the floating-base robot once, without considering contact, then add the additional constraints that come from being in contact as contact forces which get applied to the bodies. The resulting manipulator equations take the form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} = \tau_g(\boldsymbol{q}) + \boldsymbol{B}\boldsymbol{u} + \sum_i \boldsymbol{J}_i^T(\boldsymbol{q})\lambda_i,$$

where λ_i are the contact forces and J_i are the contact Jacobians. Conveniently, if the guard function in our contact equations is the signed distance from contact, $\phi_i(\boldsymbol{q})$, then this Jacobian is simply $\boldsymbol{J}_i(\boldsymbol{q}) = \dfrac{\partial \phi_i}{\partial \boldsymbol{q}}$.

9.2 Trajectory Optimization

9.3 Randomized Motion Planning

9.4 Randomized Motion Planning

9.5 Stabilizing a Trajectory or Limit Cycle