Personal Lecture Notes
MITx6.832x: Underactuated Robotics
(Spring 2019)
Algorithms for Walking, Running, Swimming, Flying, and Manipulation

Julian Eßer

February 25, 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

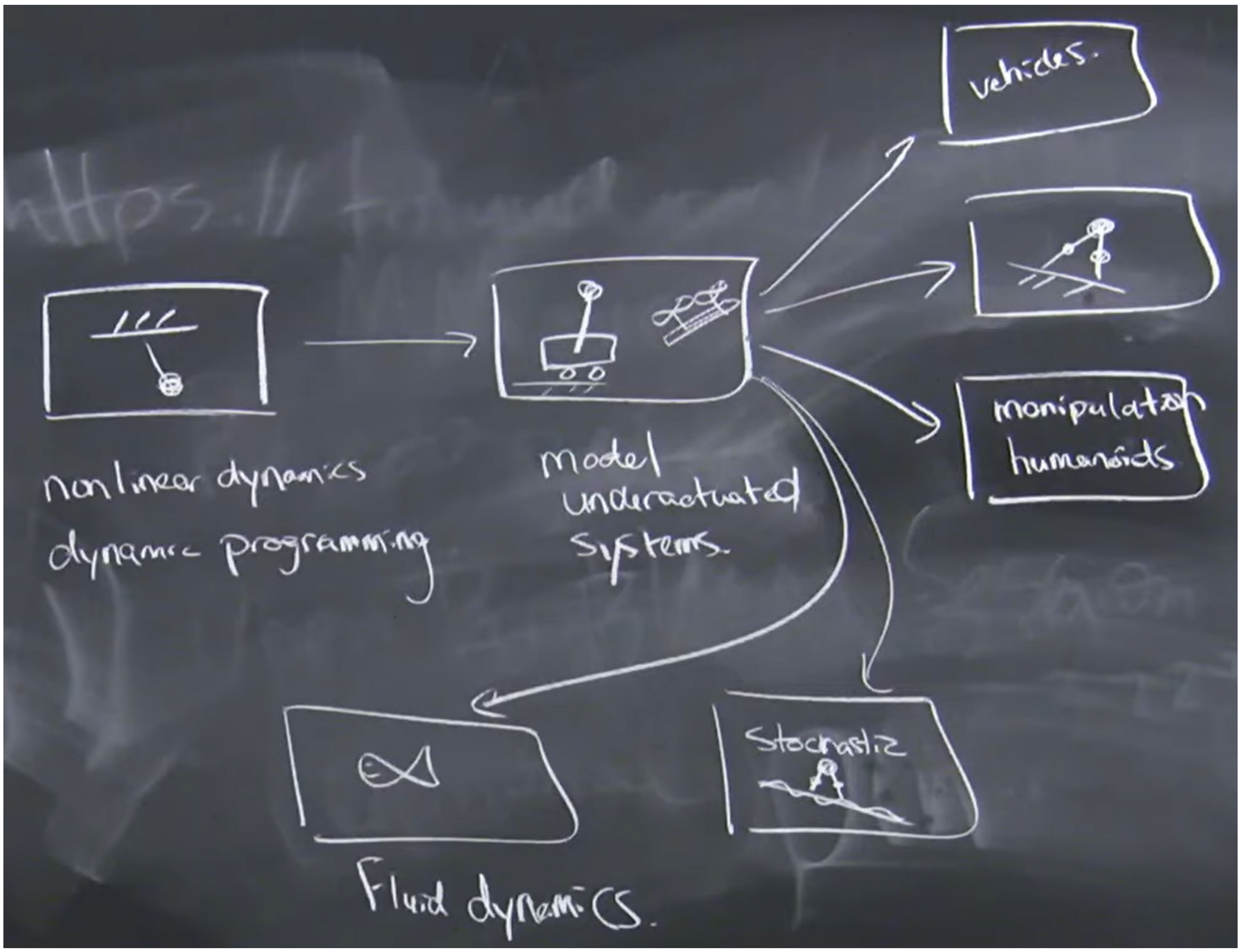

Figure 2.1: Course overview: Basics first, then simple systems and then various advanced systems.

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: $g(s, a) = 1"if"s \ = s_{goal}; 0ifotherwise$
- 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

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

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 operating 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.

5.4 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.

5.5 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

5.6 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

6.1 Lyapunov Functions

6.2 Lyapunov Analysis with Convex Optimization

6.3 Lyapunov Analysis for Estimating Regions of Attraction