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

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

Julian Eßer

February 18, 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

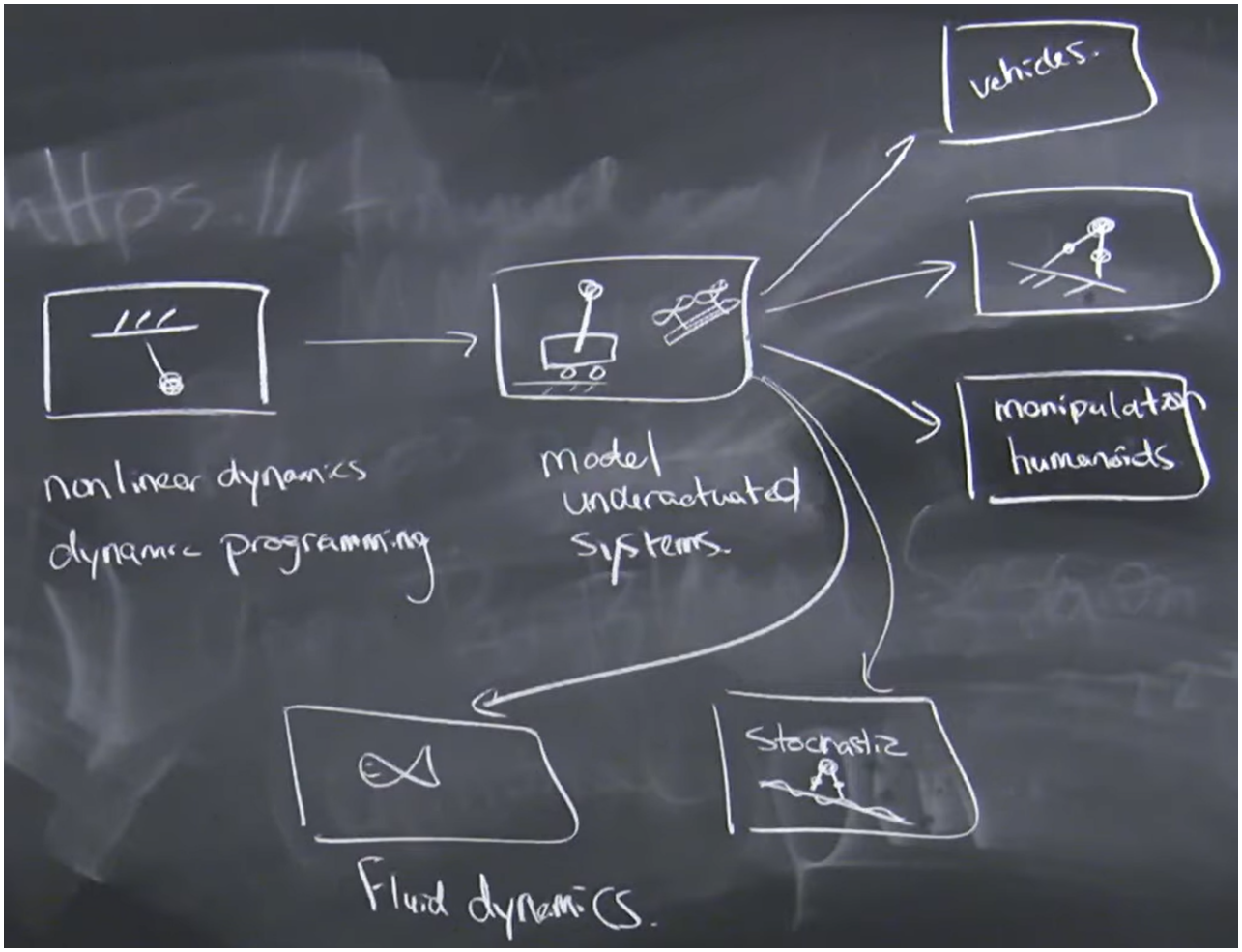

Figure 2.1: Course overview: Basics first, then simple systems and then various advanced systems.

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: Dynamic Programming I

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 Dynamic Programming Algorithm

4.3.1 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

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

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

4.3.2 Continous Dynamic Programming

4.4 Numerical Optimal Control of the pendulum