Tools Used in the ORB Research Group

A Short Introduction

Anna Lena Emonds

November 2018

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

Useful tools for our research

Puppeteer
model creation & generation of motions from motion capture data

MeshUp
Visualization of models and motions (including video export)

RBDL (Rigid Body Dynamics Library)
generation of equations of motion & contact dynamics

MUSCOD (MUltiple Shooting COde for Direct Optimal Control)
solution of optimal control problems

Optimal control problem

$$\min_{x(\cdot),x(\cdot),p,T} \underbrace{\int_0^T \phi(x(t)\,,u(t)\,,p)\,\mathrm{d}t}_{\text{Lagrange type}} + \underbrace{\Phi(T, x(T)\,,p)}_{\text{Mayer type}}$$

$$\begin{aligned} \text{s.t.} \quad & \dot{x}(t) = f(t, x(t)\,,u(t)\,,p) \\ & g(t, x(t)\,,u(t)\,,p) \geq 0 \\ & r_{eq}(x(0)\,,\ldots, x(T)\,,p) = 0 \\ & r_{ineq}(x(0)\,,\ldots, x(T)\,,p) \geq 0 \end{aligned}$$

- System dynamics can be manipulated by controls & parameters
- State and control variables are functions in time

How to handle infinite dimensionality of states & controls?

Three different approaches:

1. Dynamic programming / Hamilton-Jacobi-Bellman equation
2. Indirect methods / calculus of variations / Pontryagin Maximum Principle
3. Direct methosd (control discretization, state parametrization)

Transformation from optimal control problem to nonlinear optimization problem

$\downarrow$

Solution with nonlinear optimization methods

Control discretization ("First-discretize-then-optimize")

Definition of a grid:

$$t_a = t_0 < t_1 < \cdots < t_{m-1} < t_m = t_b$$

Control discretization by base functions $\phi_j(t, q_j)$:

$$u(t) = \phi_j(t, q_j)\,, \quad q_j \in \mathbb{R}^{k_j}, t \in [t_j, t_{j+1}] \quad \text{for} \quad 0, 1, \ldots, m-1$$

Possible base functions:

- piecewise constant
- piecewise linear
- splines
- ...

State discretization

Three different methods:

- direct collocation
- direct single shooting
- direct multiple shooting

Basic idea of shooting methods:

Trace the boundary value problem back to an initial value problem!

State discretization

Direct single shooting:

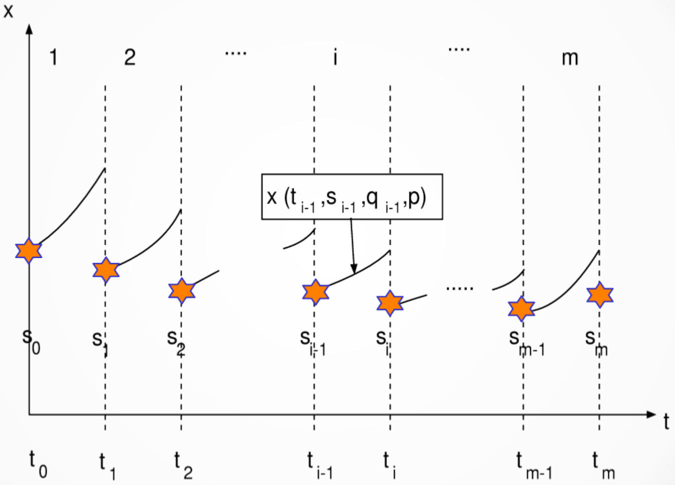
- Choose an initial value s_0.
- Solve the initial value problem:

$$\begin{aligned} \dot{x}(t) &= f(t, x, \phi(t, q_j)\,, p) \\ x(t_0) &= s_0 \qquad\qquad \Rightarrow \text{ solution: } \bar{x}(t; t_0, s_0) \end{aligned}$$

- Is the boundary condition satisfied?
  - YES! $\rightarrow$ stop here.
  - NO! $\rightarrow$ solve iteratively by Newton's method

State discretization

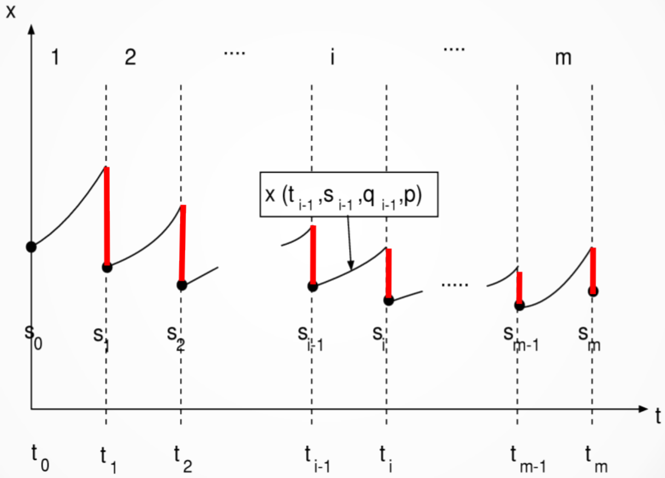
Direct multiple shooting:



Picture from Katja's MORMS 2016/17 lecture

State discretization

Direct multiple shooting:



Picture from Katja's MORMS 2016/17 lecture

State discretization

Direct multiple shooting:

- Idea: split long integration interval into many shorter ones
- Define a grid: $I_j = [t_j, t_{j+1}]$ for $j = 0, \ldots, m-1$
- Choose initial values s_j for $j = 0, \ldots, m$
- On each of the m intervals, solve initial value problems of form

$$\begin{aligned} \dot{x}(t) &= f(t, x, \phi_j(t, q_j)\,, p) \\ x(t_j) &= s_j \qquad\qquad \Rightarrow \text{ solution: } \bar{x}(t; t_j, s_j) \end{aligned}$$

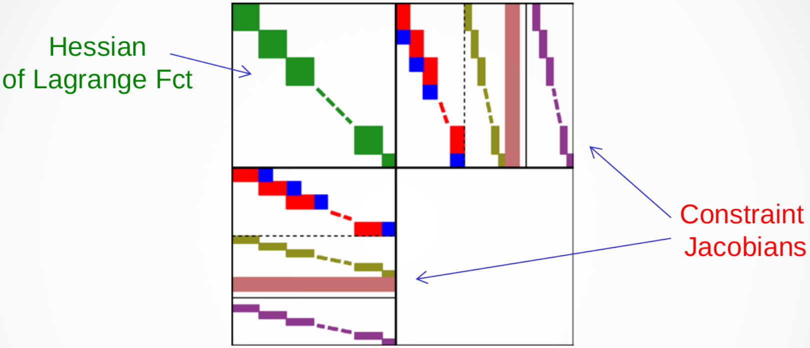
- Introduce continuity conditions to close the gaps:

$$\bar{x}(t_{j+1}; s_j, q_j, p) - s_{j+1} = 0$$

- Are the boundary and continuity conditions satisfied?
  - YES! $\rightarrow$ stop here.
  - NO! $\rightarrow$ solve iteratively by Newton's method

Resulting non-linear programming problem

Special structure in KKT-matrix due to discretization:



Picture from Katja's MORMS 2016/17 lecture

Solution with structure-exploiting, tailored sequential quadratic programming (SQP) method, special condensing techniques

A simple MUSCOD example

MUSCOD-II: Example 1: Rocket Car

Daniel Leineweber et al.

This is a simple example that should get you started with MUSCOD. It is devoted to the introduction to one-phase optimal control problems. To this end we consider a rocket car, see Figure 1.

- The rocket car should drive for 300m.
- The initial velocity at $t_0 = 0$ is 0m/s.
- The car can accelerate and brake between -2m/s^2 and 1m/s^2.
- The maximum velocity is limited to 30m/s.
- At final time t_f the car should come to a complete stop.

Figure 1: Rocket Car

Tasks

1. Optimization 1: The car should drive for 32s and then come to a complete stop. Minimize energy consumption.
2. Optimization 2: Minimize final time.

A simple MUSCOD example

Mathematical formulation:

Objective function (Lagrange type):	$\min_{u,q} \int_0^T u^2(t)\,\mathrm{d}t$	**lfcn**
	subject to the constraints:	
Dynamic process model:	$\dot{q}(t) = v(t)$ $\dot{v}(t) = u(t)$	**ffcn**
Initial & final constraints:	$q(0) = 0, \quad q(T) = 300$ $v(0) = 0, \quad v(T) = 0$	**rdfcn**
Bounds	$0 \leq q(t) \leq 300$ $0 \leq v(t) \leq 30$ $-2 \leq u(t) \leq 1$	**data** **file**

Configuring, Compiling & Running the MUSCOD example

Switch to your *projectname/* directory:

- create a *build* folder: `mkdir build`
- change into the *build* folder: `cd build`
- configure the problem with CMake: `cmake ..` or `ccmake ..`
- compile the problem: `make`
- run the problem: `muscod projectname` (sometimes it is installed as `muscod_release` or `muscod_debug`)

Rigid Body Dynamics Library (RBDL)

- Models: Lua-Files
- forward and inverse kinematics
- forward and inverse dynamics
- Jacobians
- constraints for contact & collision handling

Second Example: Cart Pendulum

RBDL and MUSCOD-II: Example 2: Cart Pendulum

Debora Clever, Manuel Kudruss (debora.clever@iwr.uni-heidelberg.de)

This is a simple example that should get you started with MUSCOD and RBDL. It is devoted to the introduction to forward dynamics and one-phase optimal control problems. To this end we consider a cart pendulum, see Figure 1.

The cart pendulum consists of two rigid bodies, the *Cart* and the *Pendulum*. The pendulum itself consists of two elements, a spherical mass and a massless link. The model has two degrees of freedom:

- q_0: the x-translation of the body *Cart*.
- q_1: the rotation around the y axis of the body *Pendulum*.

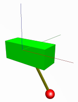


Figure 1: Cart Pendulum

The movement of the pendulum can be controlled by a force u_0 acting in horizontal direction on the cart.

Cart:

- Cuboid
- x-length= 0.5m, y-length = 0.2m, height = 0.2m
- mass = 10.0kg

Pendulum:

- Massless link: length = 0.5m
- Sphere: radius = 0.1m, mass = 1.0kg

Tasks

At initial time $t_0 = 0$ the pendulum is hanging down. Determine an optimal control, such that at final time t_f, the pendulum is standing up. To this end:

1. Set up a feasible lua model, describing the cart pendulum model.
2. Complete source and data file.
3. Optimization 1: Minimize energy consumption.
4. Optimization 2: Minimize final time.

Multiphase OCP

Each phase can have

- its own objective function (both Mayer and Lagrange terms)
- own dynamics
- individual duration
- own number of states
- own number of controls
- own number of free parameters
- type of differential equation (ODE/DAE)
- its own constraints

Discontinuities between phases

Discontinuity:

$$x\left(\tau_j^+\right) \neq x\left(\tau_j^-\right)$$

$\Rightarrow$ Additional phases in Muscod (transition phase/stage):

- phase time fixed to zero
- instead of an integrator: `libind ind_strans`
- no right hand side of differential equation, instead: jump function $x\left(\tau_j^+\right) = J\left(x\left(\tau_j^-\right), p\right)$ as `ffcn`

Third Example: Hopping Robot

RBDL and MUSCOD-II: A One-legged Hopping Robot

Martin Felis (martin.felis@iwr.uni-heidelberg.de)

This is a simple example that should get you started with MUSCOD and RBDL. It is devoted to the introduction to contact forces and multi-phase optimal control problems. To this end we consider a simple one-legged hopping robot, see Figure 1.

The hopping robot consists of two rigid bodies, the *Body* and the *Leg*. The robot has two degrees of freedom:

- q_0: the height (i.e. Z–coordinate) of the body *Body*.
- q_1: the retraction of the body *Leg*.

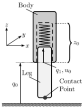


Figure 1: One-legged Hopping Robot

The two elements bodies are connected by a prismatic joint and a spring. In the case of maximal extension of the leg, i.e. $q_1 = 0$, the spring is fully extended to length z_0. The translation of the robot's leg can be controlled by the linear force in the control u_0.

In addition to gravity and interior forces of the actuated joints, it is important to also model external ground reaction forces during the contact phase. Furthermore, we want to be able to model contact gains due to collisions.

Here, the collision is modeled as an instantaneous event that results in discontinuities in the velocities. In MUSCOD-II this can be modeled using "Transition Phases" that have zero duration and are specified by using the `def_strans` pseudo-integrator in the DAT-file.

This leads to three different contact phases for the robot: the *flight* phase, the *collision* transition phase, and the *contact* phase (see Figure 2).

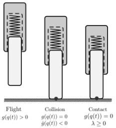


Figure 2: Overview of the three phases

The three different phases a characterized by the following properties:

Flight Phase: There is no contact with the ground ($g(q(t)) > 0$) and there are no external forces.

Collision Transition Phase: The contact condition $g(q(t)) = 0$ is fulfilled. In addition, the contact point is moving along the negative contact normal. Note, that collisions in general result in discontinuities in the velocity variables.

Contact Phase: The contact condition $g(q(t)) = 0$ is fulfilled and there are positive ground reaction forces λ in the direction of the contact normal.

All phases can be modeled using RBDL. For details we refer to the official documentation (Section "'External Contacts"').

Tasks

Determine an optimal control, such that the robot performs a periodic motion with a (vertical) velocity of $v > 5$ at take of.

1. Define the model name and the correct integrators for the individual phases in the DAT-file (`libind`).
2. Complete the function `update_generalized_variables` that copy the values from the `double` arrays to the correct Eigen vectors `Q, QDot, Tau` that are then used by RBDL methods.
3. Complete the right-hand side function stubs `ffcn_flight`, `ffcn_touchdown`, `ffcn_contact`.
4. Define the point constraints `rdfcn_*` and `def_mpc()`.
5. Optimization 1: Minimize the applied force u_0 during the whole process.
6. Optimization 2: Minimize the collision impact.