Optimization Method & Optimal Guidance

Haichao Hong

AE8120

Contents

- J. T. Betts.

Practical methods for optimal control using nonlinear programming, volume 3 of Advances in Design and Control.
SIAM, Philadelphia, 2001.

- A. E. Bryson and Y.-C. Ho.

Applied Optimal Control.
Hemisphere Publishing Corporation, Washington, 1975.

- M. Gerdts.

Optimal Control of ODEs and DAEs.
DeGruyter, Berlin, 2011

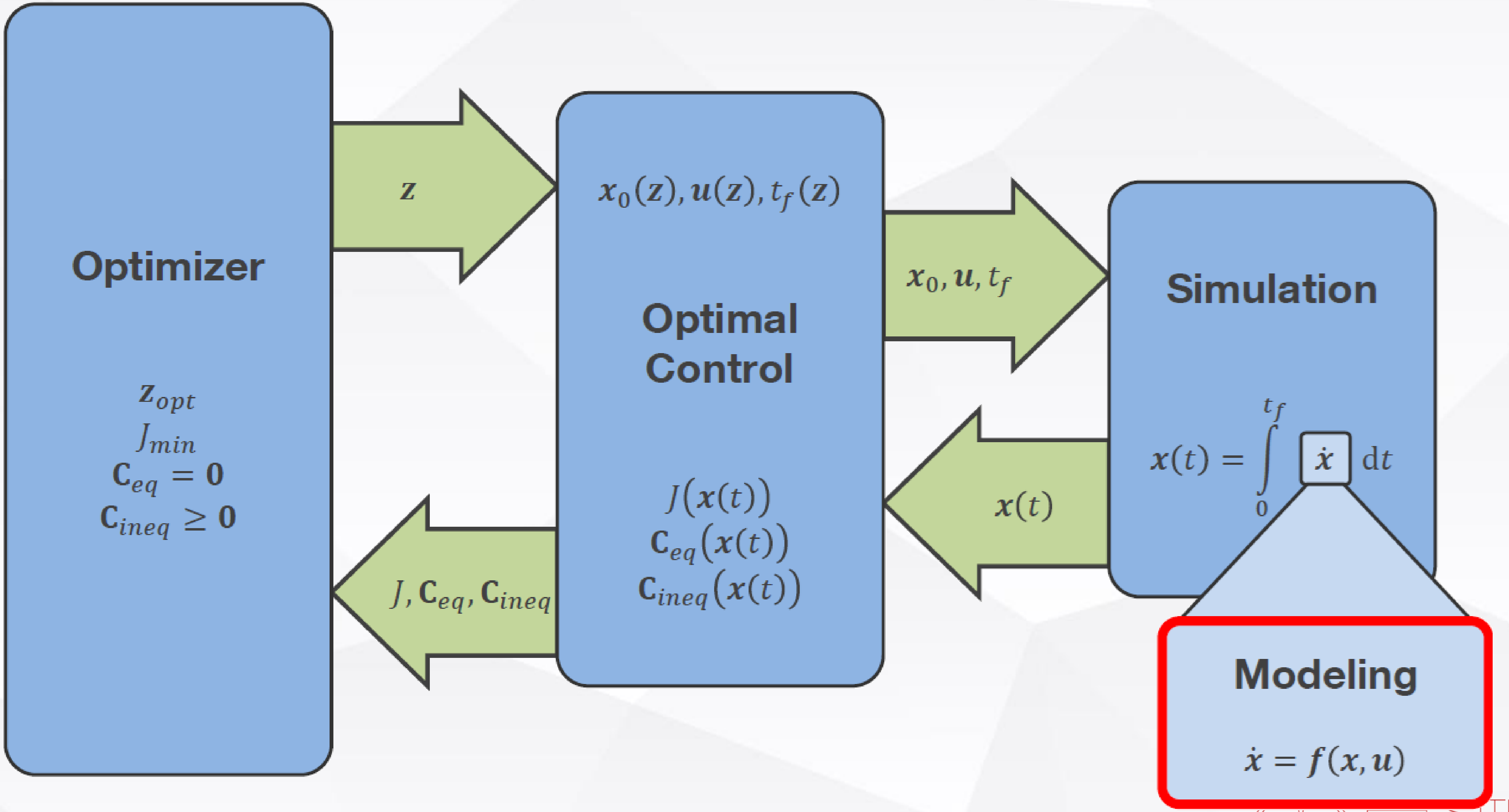
- M. R. Hestenes.

Calculus of variations and optimal control theory.
John Wiley & Sons, New York, 1966.

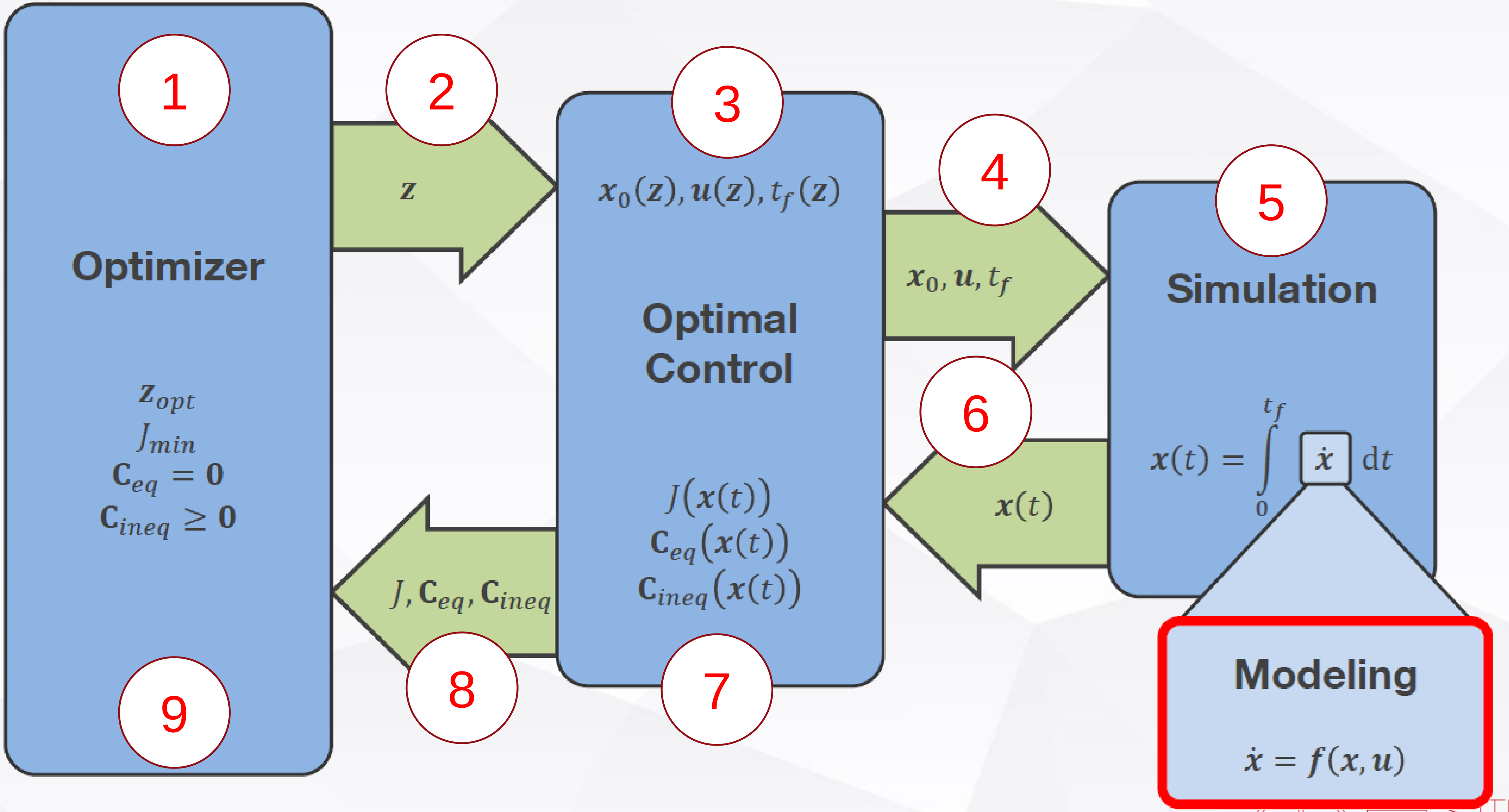
- A. D. Ioffe and V. M. Tihomirov.

Theory of extremal problems.
volume 6 of Studies in Mathematics and its Applications, Amsterdam, New York, Oxford, 1979. North-Holland Publishing Company.

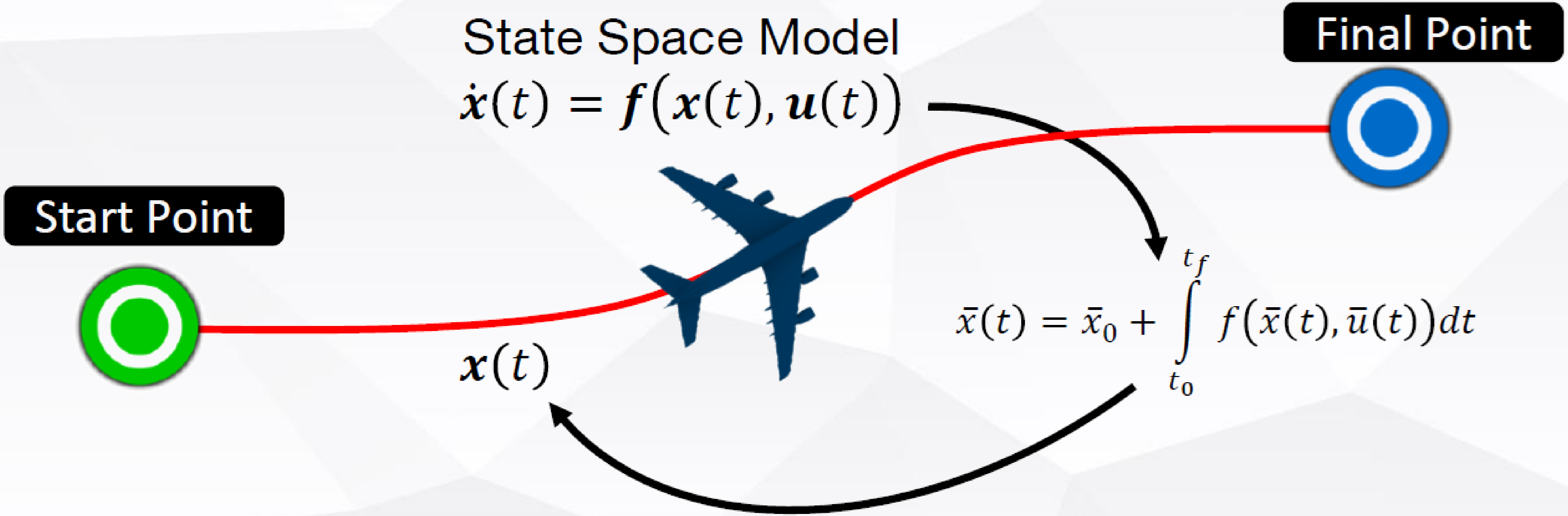
Trajectory Optimization Practice



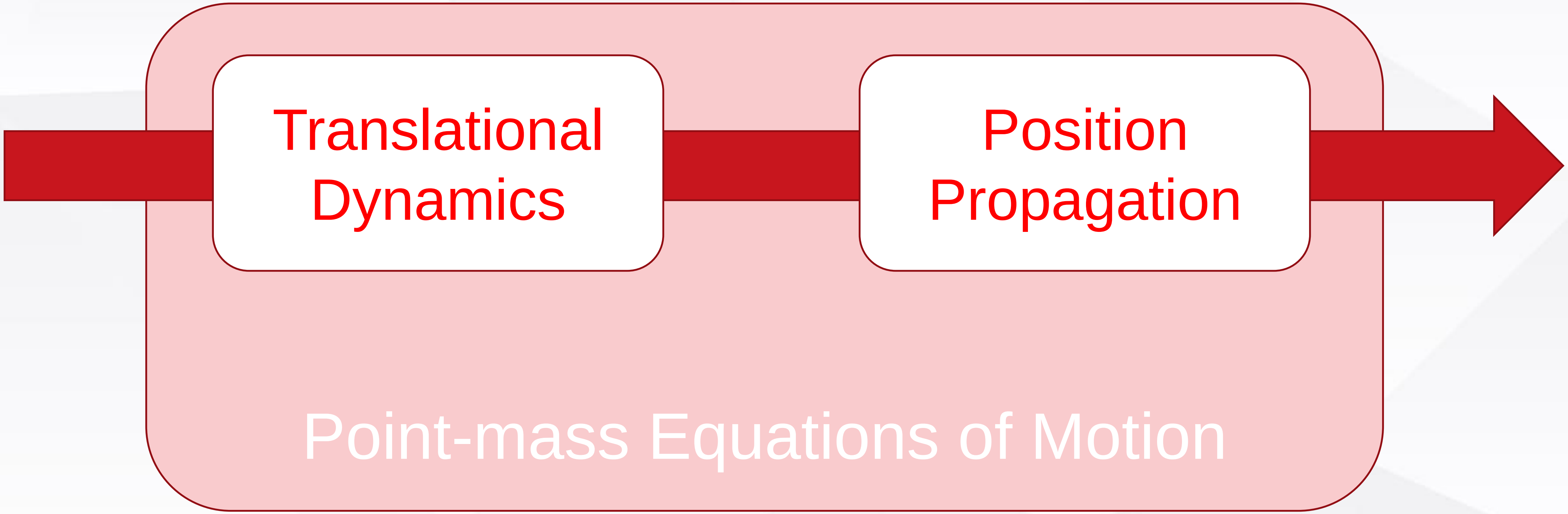
Trajectory Optimization Practice



Trajectory Optimization Practice



Trajectory Optimization Practice



Trajectory Optimization Practice

Position	λ geodetic longitude ϕ geodetic latitude h altitude	x northward position y eastward position z downward position
Velocity	u northward velocity v eastward velocity w downward velocity	V absolute velocity χ course angle γ climb angle

Control surfaces cannot perform discrete changes due to

- inertia of the control surfaces
- inertia of mechanical/hydraulic systems, actuators, ...
- aerodynamic forces counteracting on the control surfaces

Inherent dynamics can be taken into account by introducing 2nd order transfer functions for the control surface deflections (e.g. elevator deflection):

$$\eta = \frac{\omega_0^2}{s^2 + 2 \cdot \zeta \cdot \omega_0 \cdot s + \omega_0^2} \cdot \eta_{CMD} \qquad \begin{pmatrix} \dot{\eta} \\ \ddot{\eta} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2 \cdot \zeta \cdot \omega_0 \end{bmatrix} \cdot \begin{pmatrix} \eta \\ \dot{\eta} \end{pmatrix} + \begin{pmatrix} 0 \\ \omega_0^2 \end{pmatrix} \cdot \eta_{CMD}$$

Note: Three 2nd order transfer functions (one for each control surface) augment the simulation model by a total of SIX states.

A second-order system in the time-domain can be written as

$$u_{cmd}(t) = \frac{1}{\left(\omega_n(\boldsymbol{p})\right)^2}\ddot{u}(t) + \frac{2\zeta(\boldsymbol{p})}{\omega_n(\boldsymbol{p})}\dot{u}(t) + u(t)$$

where ζ and ω_n are the damping ratio and the natural frequency, respectively. $\boldsymbol{p}$ is the model-dependent parameters, e.g., the aerodynamic hinge moment. One may therefore be able to reversely compute the ideal actuator command, $u_{cmd}(t)$ based on the desired control variables and their derivatives.

Engine dynamics

- Thrust force cannot perform discrete changes due to engine dynamics
- Introducing 1st order transfer function

$$\delta_T = \frac{1}{T_\delta \cdot s + 1} \cdot \delta_{T,CMD} \qquad \dot{\delta}_T = \frac{1}{T_\delta} \cdot \left(\delta_{T,CMD} - \delta_T\right)$$

Note: One 1st order transfer functions augments the model by ONE additional state.

For flight over longer time durations, fuel consumptions and thus mass change of the aircraft has to be taken into account:

$$\dot{m}_{fuel} = \dot{m}_{fuel,idle} + \delta_T \cdot \left(\dot{m}_{fuel,\max} - \dot{m}_{fuel,idle}\right)$$

$$\dot{m} = -\dot{m}_{fuel}$$

Note: Modeling of the mass flow augments the model by ONE additional state

FALCON.m

FALCON.m is the *FSD optimAL CONtrol tool for MATLAB* that has been developed at the Institute of *Flight System Dynamics* of *Technische Universität München*.

FALCON.m is able to solve optimal control problems of the following form: Minimize the cost function

$$\min \; J(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p})$$

subject to a set of constraints, formed by the differential algebraic equation

$$\begin{bmatrix} \dot{\boldsymbol{x}}(t) \\ \boldsymbol{y}(t) \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) \\ \boldsymbol{h}(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{p}) \end{bmatrix}$$

where $\boldsymbol{x}(t)$ specifies the states, $\dot{\boldsymbol{x}}(t)$ the state derivatives and $\boldsymbol{y}(t)$ additional model outputs.

Remark: A maximization of the cost function $\bar{J}$ can be achieved by simply choosing

$$J = -\bar{J}.$$

The states $\boldsymbol{x}(t)$, the controls $\boldsymbol{u}(t)$ and the parameters $\boldsymbol{p}$ are limited by a lower and an upper bound:

$$
\begin{aligned}
\boldsymbol{x}_{lb} &\leq \boldsymbol{x}(t) \leq \boldsymbol{x}_{ub} \\
\boldsymbol{u}_{lb} &\leq \boldsymbol{u}(t) \leq \boldsymbol{u}_{ub} \\
\boldsymbol{p}_{lb} &\leq \boldsymbol{p} \leq \boldsymbol{p}_{ub}
\end{aligned}
$$

The problem is considered on the time interval $[t_0, t_f]$ with each of the two either being fixed or free. In the formulation presented here, t_0 and t_f are seen to be part of the parameter vector $\boldsymbol{p}$.

Additionally, an arbitrary number of nonlinear constraints of the form

$$\boldsymbol{g}_{lb} \leq \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{u}, \boldsymbol{p}) \leq \boldsymbol{g}_{ub}$$

may be imposed. A special type of constraints appearing in many problems are initial and final boundary conditions specifying a start and an end state condition of the form

$$\begin{aligned} \boldsymbol{x}_{0,lb} \leq& \boldsymbol{x}(t_0) \leq \boldsymbol{x}_{0,ub} \\ \boldsymbol{x}_{f,lb} \leq& \boldsymbol{x}(t_f) \leq \boldsymbol{x}_{f,ub} \end{aligned}$$

For all constraints, equality conditions can be achieved by simply setting the upper and the lower limits to the same values.

$$\square_{lb} = \square_{ub}$$

Trajectory Optimization Practice

- FALCON.m uses direct discretization methods in order to solve optimal control problems. The free variable is considered to be *time* throughout the implementation but may be chosen however suitable.
- For each value appearing in an optimal control problem, FALCON.m uses value definition objects, specifying the names, bounds and scaling of the values as appropriate. If required, these values are extended to grids over time inside FALCON.m. Examples for value definition objects are `falcon.State`, `falcon.Control`, `falcon.Constraint`, `falcon.Cost`, `falcon.Parameter` and some more.

Trajectory Optimization Practice

- FALCON.m allows the solution of multi phase optimal control problems, where each problem has to hold at least on `falcon.core.Phase`. Each phase holds a stategrid, one or more controlgrids, and a model. Phases may or may not be linked together.
- FALCON.m performs the optimization on a normalized time grid $\tau \in [0, 1]$ for every phase that is mapped to the real time grid by a linear transformation. Problems with variable final and/or initial time can be solved by choosing the initial or final time to be a free parameter.

- FALCON.m performs the optimization on a normalized time grid $\tau \in [0, 1]$ for every phase that is mapped to the real time grid by a linear transformation. Problems with variable final and/or initial time can be solved by choosing the initial or final time to be a free parameter.

- FALCON.m uses autonomous dynamics ("time-invariant") as default (the dynamic equations may not directly depend on the free variable, time). Anyway, non-autonomous dynamics can be tackled by introducing a new state t with the dynamics

$$\dot{t} = 1$$

that is added to the state vector of the problem. All other steps in creating the model, e.g., adding a final time parameter to the phase, remain the same. The collocation integrator should achieve that the final time in the state and the parameter are the same. If this is not the case an additional constraint may be added to the final point to assure that the times match.

Trajectory Optimization Practice

- In order to achieve better numerical properties of a problem, FALCON.m internally scales all appearing values by a fixed scaling factor. The following relationship is used for scaling

$$\square_{\text{scaled}} = \square_{\text{original}} \cdot M_{\text{scaling}}$$

  It is recommended to scale all values to an order of magnitude of one, meaning that e.g. the scaling factor of a value expected to be about 10^5 in the problem should be scaled by a scaling factor $M_{\text{scaling}} = 10^{-5}$.

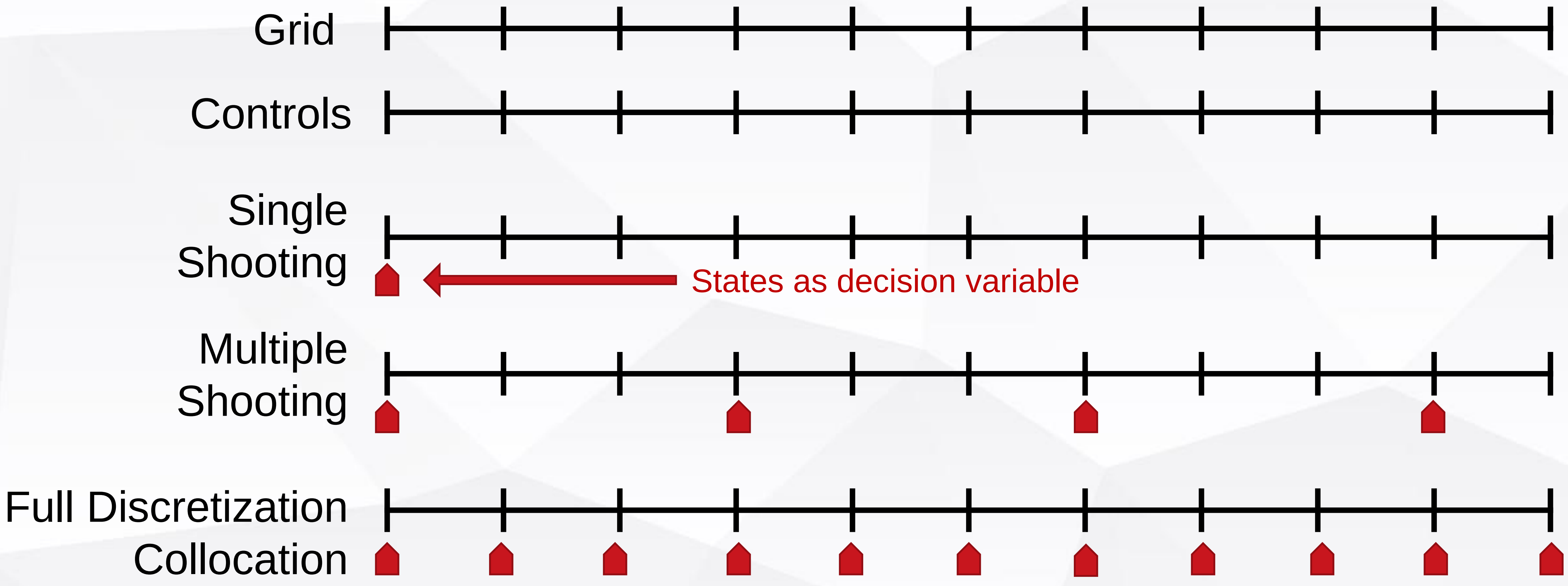
- As FALCON.m uses gradient based optimization algorithms, initial guess values for everything to be optimized need to be available. In case no initial guess values are specified by the user, FALCON.m tries to create them itself.

- In order to solve an optimal control problem in FALCON.m four main steps are required:

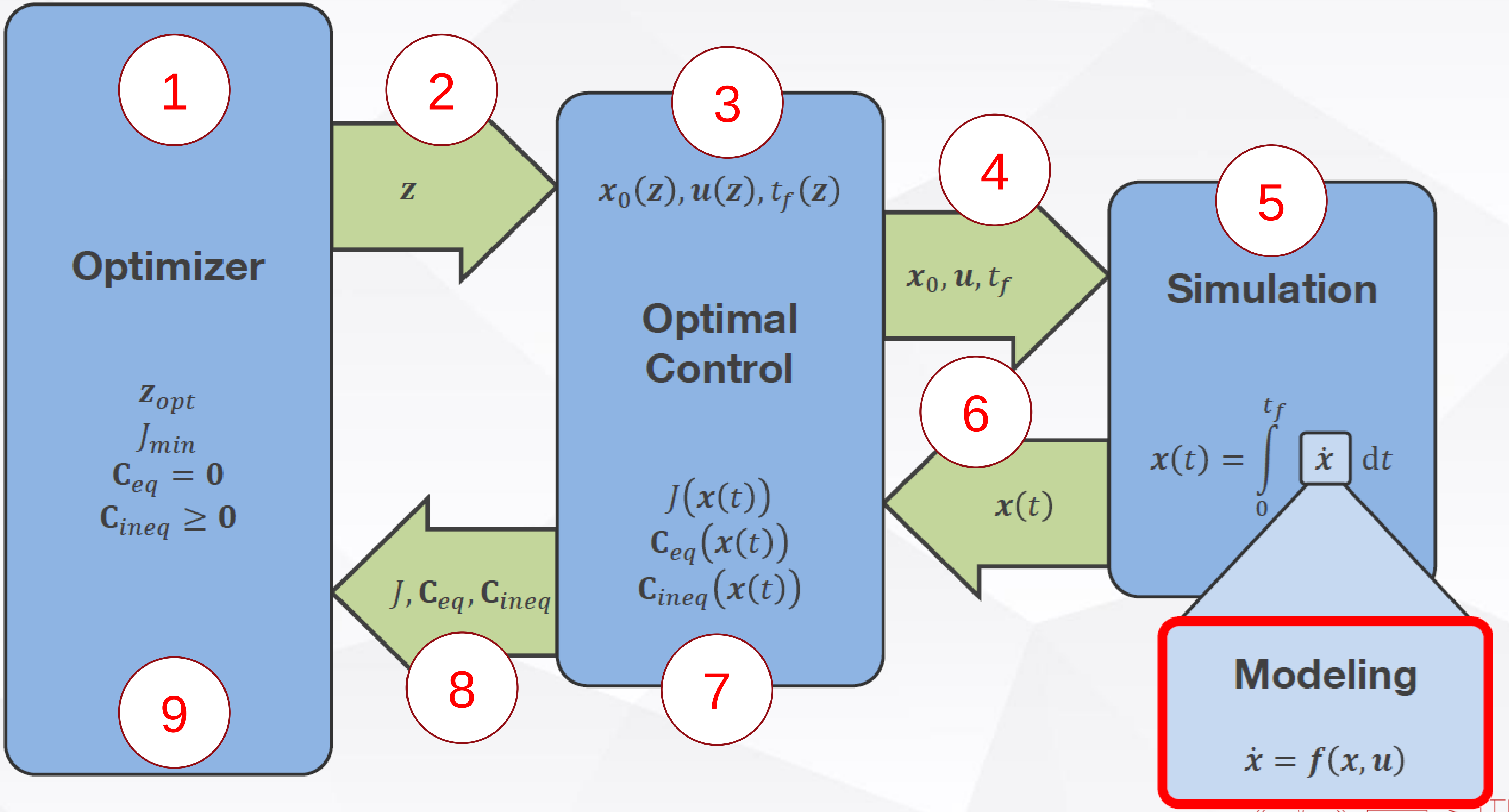
  1. Define the model, constraint equations and problem structure in FALCON.m.
  2. Create the analytic derivatives of all appearing functions and create MATLAB executables (.mex files) from these functions.
  3. Prepare the problem itself for solution.
  4. Solve the problem using third party numerical optimization algorithms.

Trajectory Optimization Practice

Discretization Re-visit



Trajectory Optimization Practice



Optimization parameter vector z is built up from

Single shooting

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{u}_0 & \ldots & \boldsymbol{u}_N & \boldsymbol{p} \end{pmatrix}^{\mathsf{T}}$$

Multiple shooting

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_K & \boldsymbol{u}_0 & \ldots & \boldsymbol{u}_N & \boldsymbol{p} \end{pmatrix}^{\mathsf{T}}$$

Collocation:

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_N & \boldsymbol{u}_0 & \ldots & \boldsymbol{u}_N & \boldsymbol{p} \end{pmatrix}^{\mathsf{T}}$$

Optimization parameter vector $\boldsymbol{z}$ is built up from

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_N & \boldsymbol{u}_0 & \dots & \boldsymbol{u}_N & \boldsymbol{p} \end{pmatrix}^\mathsf{T}$$

Integration defect

$$\mathbf{c}_k(\boldsymbol{z}) = \boldsymbol{x}_k(\boldsymbol{z}) - \boldsymbol{x}_{k+1}(\boldsymbol{z}) + h_k \cdot \Phi(\boldsymbol{x}_k(\boldsymbol{z}), \boldsymbol{x}_{k+1}(\boldsymbol{z}), \boldsymbol{u}_k(\boldsymbol{z}), \boldsymbol{u}_{k+1}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z})) \overset{!}{=} 0$$

for general integration scheme $\Phi(\boldsymbol{x}_k(\boldsymbol{z}), \boldsymbol{x}_{k+1}(\boldsymbol{z}), \boldsymbol{u}_k(\boldsymbol{z}), \boldsymbol{u}_{k+1}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z}))$.

Examples:

Euler backward collocation

$$\boldsymbol{x}_k(\boldsymbol{z}) - \boldsymbol{x}_{k+1}(\boldsymbol{z}) + h_k \cdot \boldsymbol{f}(\boldsymbol{x}_{k+1}(\boldsymbol{z}), \boldsymbol{u}_{k+1}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z})) \stackrel{!}{=} 0.$$

Trapezoidal collocation

$$\boldsymbol{x}_k(\boldsymbol{z}) - \boldsymbol{x}_{k+1}(\boldsymbol{z}) + \frac{h_k}{2} \cdot (\boldsymbol{f}(\boldsymbol{x}_k(\boldsymbol{z}), \boldsymbol{u}_k(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z})) + \boldsymbol{f}(\boldsymbol{x}_{k+1}(\boldsymbol{z}), \boldsymbol{u}_{k+1}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z}))) \stackrel{!}{=} 0.$$

Resulting Constraint vector

$$\mathbf{C}(\boldsymbol{z}) = \begin{pmatrix} \boldsymbol{x}_0(\boldsymbol{z}) \\ \boldsymbol{x}_f(\boldsymbol{z}) \\ \boldsymbol{x}_k(\boldsymbol{z}) - \boldsymbol{x}_{k+1}(\boldsymbol{z}) + h_k \cdot \Phi(\boldsymbol{x}_k(\boldsymbol{z}), \boldsymbol{x}_{k+1}(\boldsymbol{z}), \boldsymbol{u}_k(\boldsymbol{z}), \boldsymbol{u}_{k+1}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z})) \\ \mathbf{C}(\boldsymbol{x}(\boldsymbol{z}), \boldsymbol{u}(\boldsymbol{z}), \boldsymbol{p}(\boldsymbol{z}), t(\boldsymbol{z})) \end{pmatrix}$$

Sequential Convex Optimization

1. Deal with the nonlinear dynamics
2. Deal with the constraints
3. Iteration

Trajectory Optimization Practice

$$(\dot{\boldsymbol{x}})^{(j+1)} = \boldsymbol{F}_{\boldsymbol{x}}^{(j)} \left(\boldsymbol{x}^{(j+1)} - \boldsymbol{x}^{(j)} \right) + \boldsymbol{F}_{\boldsymbol{u}}^{(j)} \left(\boldsymbol{u}^{(j+1)} - \boldsymbol{u}^{(j)} \right) + \boldsymbol{f}^{(j)}$$

$$d\boldsymbol{x}^{(j+1)} := \boldsymbol{x}^{(j+1)} - \boldsymbol{x}^{(j)} ,$$

$$d\boldsymbol{u}^{(j+1)} := \boldsymbol{u}^{(j+1)} - \boldsymbol{u}^{(j)} .$$

$$\boldsymbol{x}_{k+1}^{(j+1)} = \boldsymbol{x}_k^{(j+1)} + h \left(\dot{\boldsymbol{x}} \right)_k^{(j+1)}$$

$$\begin{aligned} d\boldsymbol{x}_{k+1}^{(j+1)} + \boldsymbol{x}_{k+1}^{(j)} &= h \left[(\boldsymbol{F}_{\boldsymbol{x}})_k^{(j)} d\boldsymbol{x}_k^{(j+1)} + (\boldsymbol{F}_{\boldsymbol{u}})_k^{(j)} d\boldsymbol{u}_k^{(j+1)} + \boldsymbol{f}_k^{(j)} \right] + \left[\boldsymbol{x}_k^{(j)} + d\boldsymbol{x}_k^{(j+1)} \right] \\ &= \left[h(\boldsymbol{F}_{\boldsymbol{x}})_k^{(j)} + \boldsymbol{I}_n \right] d\boldsymbol{x}_k^{(j+1)} + h(\boldsymbol{F}_{\boldsymbol{u}})_k^{(j)} d\boldsymbol{u}_k^{(j+1)} + h\boldsymbol{f}_k^{(j)} + \boldsymbol{x}_k^{(j)} \end{aligned}$$

$$\dot{x}(t) = V(t)\cos\chi(t)\cos\gamma(t)\ ,$$

$$\dot{y}(t) = V(t)\sin\chi(t)\cos\gamma(t)\ ,$$

$$\dot{h}(t) = V(t)\sin\gamma(t)\ ,$$

$$\dot{\chi}(t) = \frac{L(t)\sin\mu(t)}{m_v V(t)\cos\gamma(t)}\ ,$$

$$\dot{\gamma}(t) = \frac{L(t)\cos\mu(t) - m_v g\cos\gamma(t)}{m_v V(t)}\ ,$$

$$\dot{V}(t) = \frac{T(t) - D(t)}{m_v} - g\sin\gamma(t)$$

$$T(t) = \delta_T(t) T_{max}$$

$$L(t) = \frac{1}{2}\rho(V(t))^2 S C_L(t)$$

$$D(t) = \frac{1}{2}\rho(V(t))^2 S\left(C_{D_0} + k_i (C_L(t))^2\right)$$

$$\boldsymbol{x} = [x, y, h, \chi, \gamma, V]^{\mathrm{T}}\ ,$$

$$\boldsymbol{u} = [C_L, \mu, \delta_T]^{\mathrm{T}}\ .$$

Jacobian Matrix

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial \boldsymbol{f}}{\partial x_1} & \cdots & \dfrac{\partial \boldsymbol{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Note: MATLAB symbolic toolbox can help

Let's check some papers