EN.530.678: Nonlinear Control and Planning in Robotics

Direct Methods for Trajectory Optimization

April 14, 2014

Outline

Intro

Optimization

- Unconstrained
- Equality Constraints
- Inequality Constraints

Direct Methods

- Direct Shooting
- Direct Multiple-Shooting
- Direct Transcription/Collocation

Special cases

- Differentially flat systems
- Constraint Homotopy

Outline

Intro

Optimization
- Unconstrained
- Equality Constraints
- Inequality Constraints

Direct Methods
- Direct Shooting
- Direct Multiple-Shooting
- Direct Transcription/Collocation

Special cases
- Differentially flat systems
- Constraint Homotopy

The trajectory planning problem

- design a reference trajectory $x(t) \in \mathbb{R}^n$ and control inputs $u(t) \in \mathbb{R}^m$ by solving the constrained optimal control problem:

$$\min_{x(\cdot),u(\cdot),t_f} J = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt, \tag{1}$$

subject to:

$$x(t_0) = x_0, \quad x(t_f) \text{ and } t_f \text{ free} \tag{2}$$

$$\dot{x}(t) = f(x(t), u(t), t) \tag{3}$$

$$c(x(t), u(t), t) \leq 0, \text{ for all } t \in [t_0, t_f] \tag{4}$$

$$\psi(x(t_f), t_f) \leq 0, \tag{5}$$

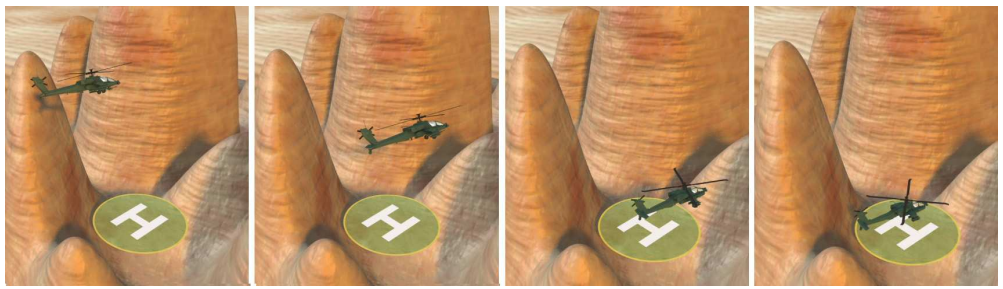
where

- $[t_0, t_f]$ - time horizon, x_0-initial state
- L - trajectory cost: e.g. control effort, energy, time, distance
- ϕ - terminal cost: e.g. reaching a desired region
- c - trajectory constraints: e.g. control bounds, forbidden regions in X
- ψ - terminal constraint defines algebraically a goal region

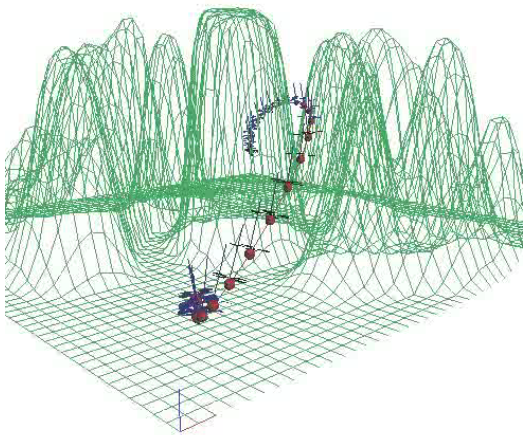
Methods

- Local gradient-based methods
  - direct collocation/shooting/multiple-shooting/sweep methods
  - indirect collocation/shooting/multiple-shooting/sweep methods
- Global methods
  - dynamic programming (DP)
    - approximate dynamic programming (ADP)
      value function approximation, simulation / roll-outs
    - graph-based methods
      1.) state+control space discretization (value/policy iteration)
      2.) state space approximate exploration: sampling-based planning
  - evolutionary methods/stochastic optimization
    - stochastic gradients (SPSA); simulated annealing (SA)
    - genetic algorithms (GA); differential evolution strategies (DES)
    - particle-swarm optimization (PSO); ant-colony optimization (ACO)
    - covariance matrix adaptation (CMA); cross-entropy method (CEM)
- Special structure-exploiting features (applies to both local and global)
  - differential flatness (removes differential constraints)
  - symmetries (allows sequencing of precomputed trajectories)
  - hybrid dynamics; constraint relaxation
  - linearity; convexity; least-squares; mixed integer, etc...

Example: direct collocation



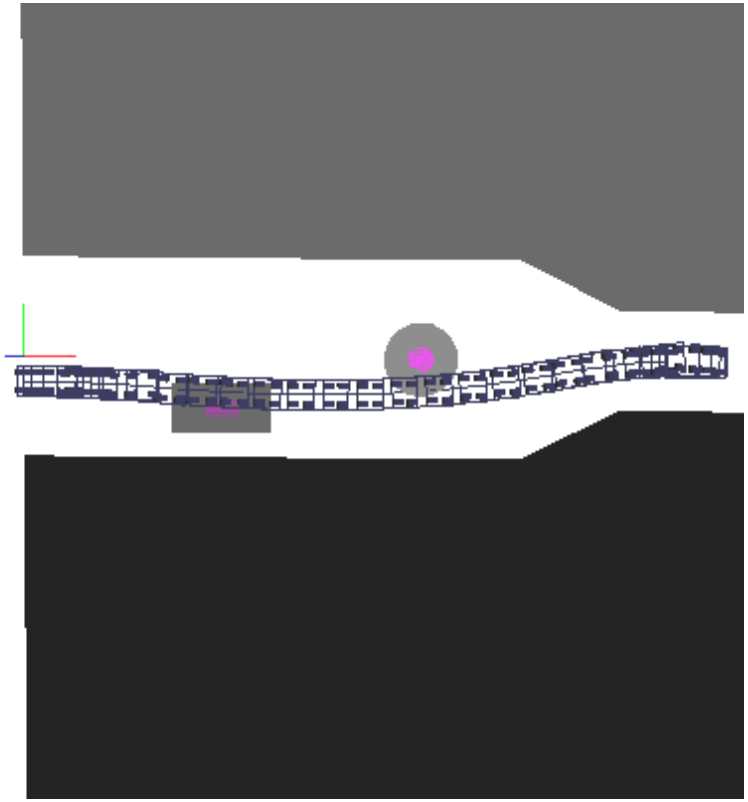
Helicopter in a canyon



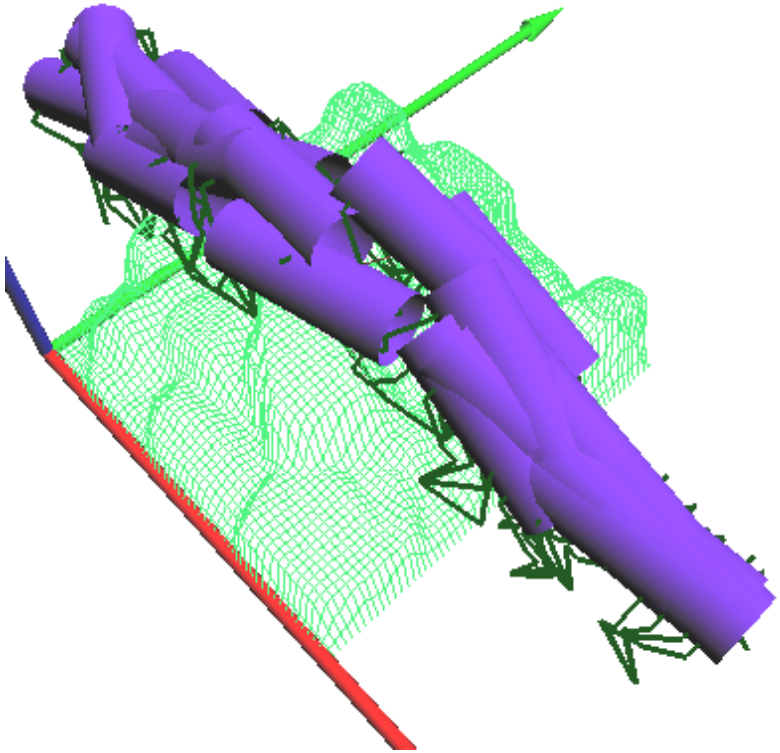
Numerical representation

- an underactuated helicopter model in a 3D terrain
- problem: time-optimal navigation and landing with zero-velocity
- approach: sequential quadratic programming (SQP), exploiting Jacobian sparsity
- terrain non-penetration enforce using the Proximity-query package (PQP)
- trajectory is discretized, and optimization is over each discrete state, control, and time
- solution is only locally optimal, convergence could be slow if many constraints

Examples: nonholonomic / multi-body constraints



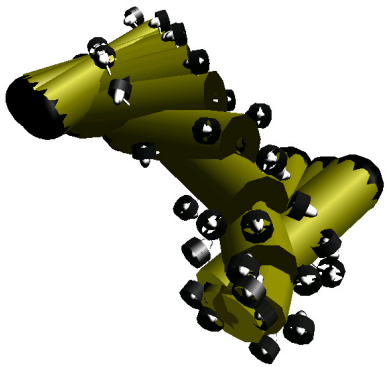
Car in a tunnel



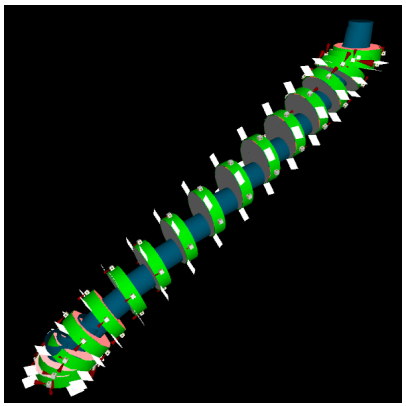
LittleDog on rough terrain

- complex constraints require special handling: homotopy continuation
- start with a relaxed problem (obstacles smoothed/removed) and slowly deform it back to origin while reoptimizing
- solution is only locally optimal, convergence could be slow

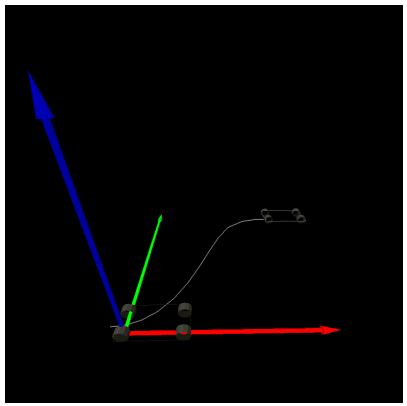
Examples: sweep methods



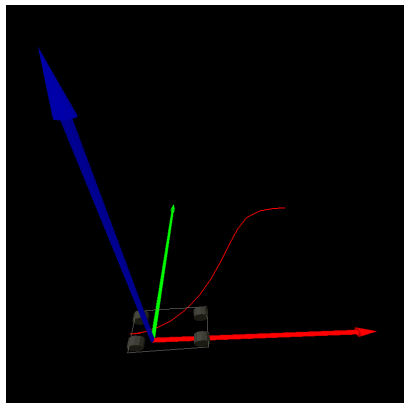
UUV



Satellite



Car iterations

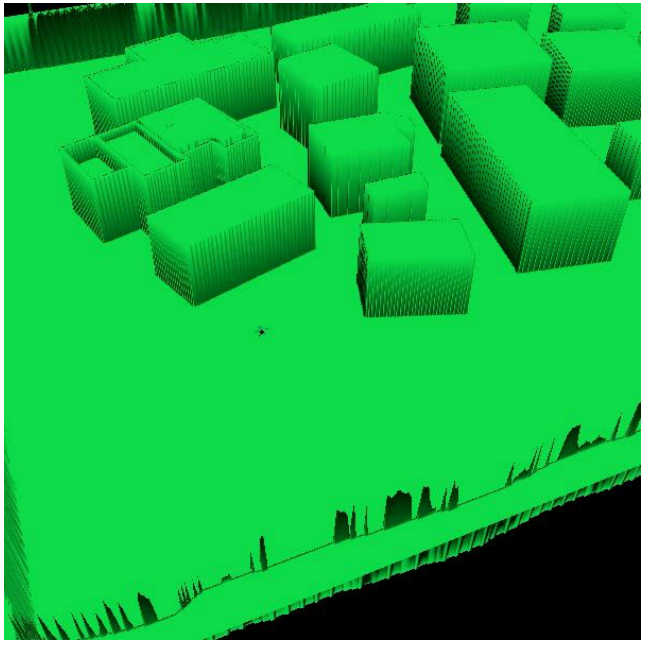


Car motion

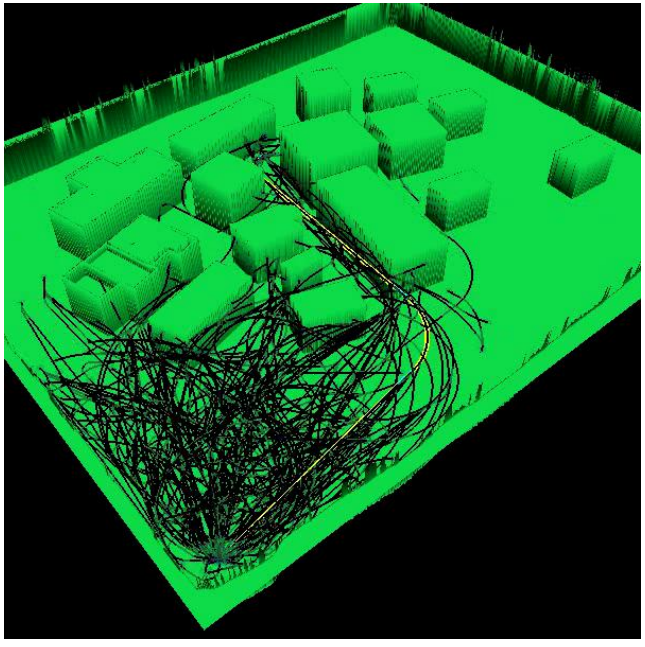
- fast methods when no path constraints are present
- optimization requires up to a few milliseconds
- only applicable when dynamics, cost are smooth; and control constraints are convex
- solution is only locally optimal, convergence is very fast

Example: sampling-based planning

- Probabilistic roadmaps (PRM) or trees (RRT*)



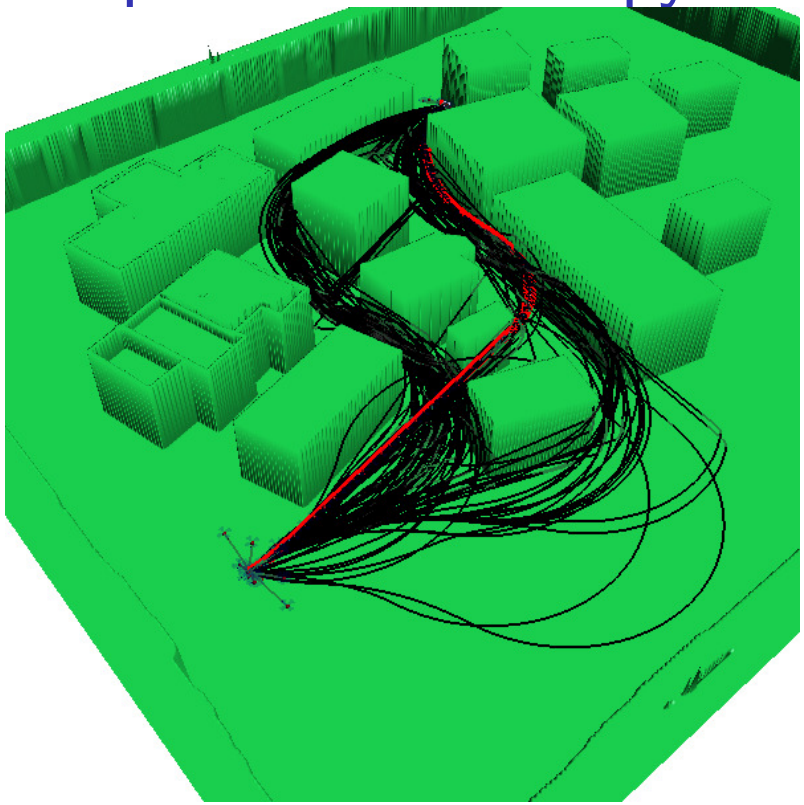
Time-optimal helicopter trajectory



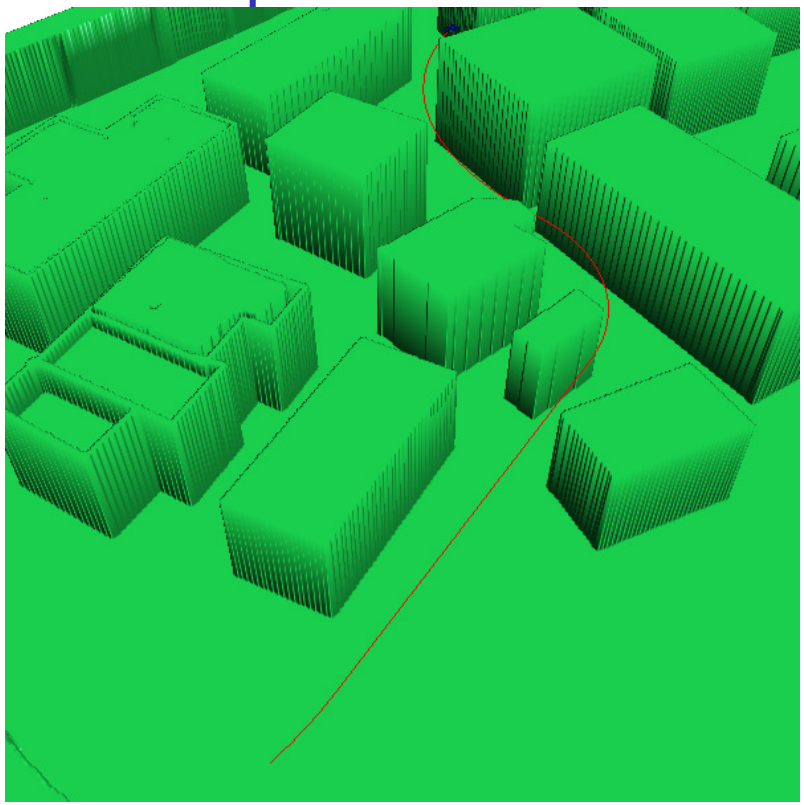
Probabilistic Roadmap (PRM)

- states sampled from environment
- connected with trajectories to form a tree/graph structure
- optimal path is then found through discrete graph search
- method can very quickly find *any* solution
- solution is only approximate; could be far from optimal

Example: cross-entropy stochastic optimization



adaptive sampling

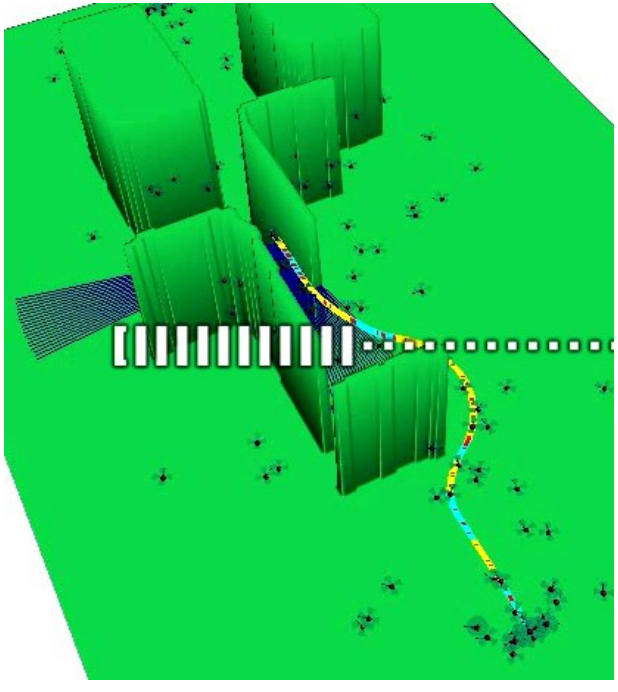


computed best path

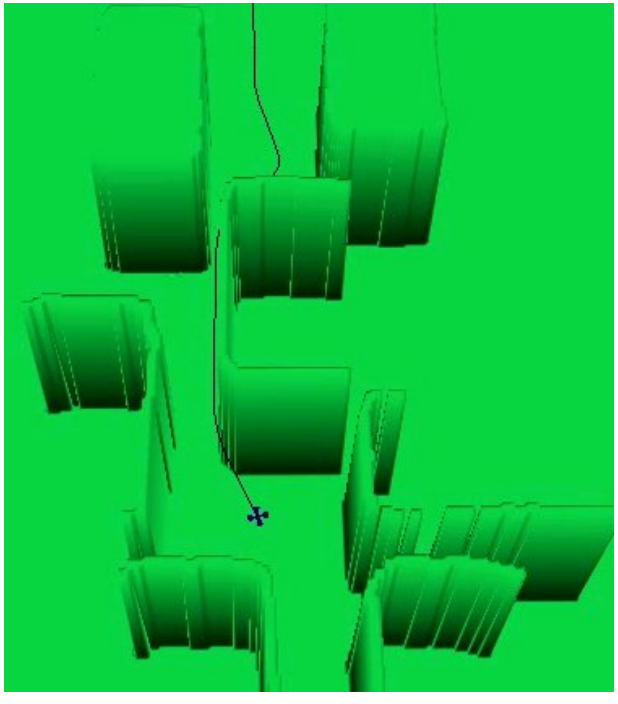
- high-dimensional underactuated system
- state parametrization using precomputed optimized primitives
- versatile trajectories can be generated instantly
- trajectory space explored using adaptive sampling
- performance scales with number of samples
- not suitable for narrow passages

Example: sampling-based methods, replanning, feedback

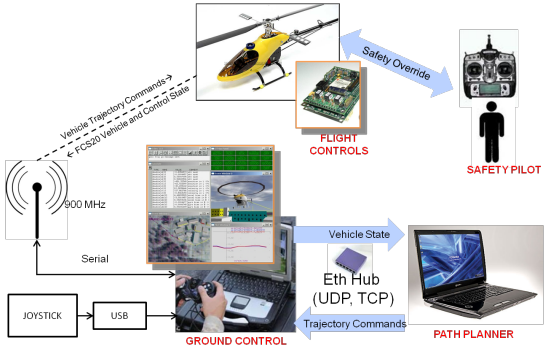
Map construction and optimal replanning



fast replanning



final path



experiment setup

Outline

Intro

Optimization
- Unconstrained
- Equality Constraints
- Inequality Constraints

Direct Methods
- Direct Shooting
- Direct Multiple-Shooting
- Direct Transcription/Collocation

Special cases
- Differentially flat systems
- Constraint Homotopy

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$
- First-order necessary condition for local optimality:

$$\nabla J(x^*) = 0$$

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$
- First-order necessary condition for local optimality:

$$\nabla J(x^*) = 0$$

- Second-order sufficient condition for a local optimum:

$$\nabla^2 J(x^*) > 0 \quad \text{(Hessian is positive definite)}$$

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$
- First-order necessary condition for local optimality:

$$\nabla J(x^*) = 0$$

- Second-order sufficient condition for a local optimum:

$$\nabla^2 J(x^*) > 0 \quad \text{(Hessian is positive definite)}$$

- Newton's method: iterate $x = x + \alpha d$ with *search direction* $d \in \mathbb{R}^n$ given by

$$\nabla^2 J(x) d = -\nabla J(x),$$

where $\alpha > 0$ is the *step-size*, ensuring cost function decreases

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$
- First-order necessary condition for local optimality:

$$\nabla J(x^*) = 0$$

- Second-order sufficient condition for a local optimum:

$$\nabla^2 J(x^*) > 0 \quad \text{(Hessian is positive definite)}$$

- Newton's method: iterate $x = x + \alpha d$ with *search direction* $d \in \mathbb{R}^n$ given by

$$\nabla^2 J(x) d = -\nabla J(x),$$

  where $\alpha > 0$ is the *step-size*, ensuring cost function decreases
- Sufficient conditions for a global optimum: J is convex, i.e. $\nabla^2 J(x) > 0$ for all x.

Unconstrained Optimization

- Find the minimum $x^* = \arg\min J(x)$ of a given function $J : \mathbb{R}^n \to \mathbb{R}$
- First-order necessary condition for local optimality:

$$\nabla J(x^*) = 0$$

- Second-order sufficient condition for a local optimum:

$$\nabla^2 J(x^*) > 0 \quad \text{(Hessian is positive definite)}$$

- Newton's method: iterate $x = x + \alpha d$ with *search direction* $d \in \mathbb{R}^n$ given by

$$\nabla^2 J(x) d = -\nabla J(x),$$

where $\alpha > 0$ is the *step-size*, ensuring cost function decreases

- Sufficient conditions for a global optimum: J is convex, i.e. $\nabla^2 J(x) > 0$ for all x.
- In practice, the direction d is computed according to:

$$\widetilde{H}(x) d = -\nabla J(x),$$

where $\widetilde{H}$ is a positive definite approximation to the Hessian

Equality Constraints

- Cost function $J : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $F : \mathbb{R}^n \to \mathbb{R}^{m \leq n}$

Equality Constraints

- Cost function $J : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $F : \mathbb{R}^n \to \mathbb{R}^{m \leq n}$
- Find $x^* = \operatorname{argmin} J(x)$ such that $F(x^*) = 0$

Equality Constraints

- Cost function $J : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $F : \mathbb{R}^n \to \mathbb{R}^{m \leq n}$
- Find $x^* = \operatorname{argmin} J(x)$ such that $F(x^*) = 0$
- Define the Lagrangian

$$L(x, \lambda) = J(x) + \lambda^T F(x),$$

where $\lambda \in \mathbb{R}^m$ are *Lagrange multipliers*.

Equality Constraints

- Cost function $J : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $F : \mathbb{R}^n \to \mathbb{R}^{m \leq n}$
- Find $x^* = \operatorname{argmin} J(x)$ such that $F(x^*) = 0$
- Define the Lagrangian

$$L(x, \lambda) = J(x) + \lambda^T F(x),$$

  where $\lambda \in \mathbb{R}^m$ are *Lagrange multipliers*.
- the constrained problem is equivalent to unconstrained minimization of L over (x, λ)

Equality Constraints

- Cost function $J : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $F : \mathbb{R}^n \to \mathbb{R}^{m \leq n}$
- Find $x^* = \operatorname{argmin} J(x)$ such that $F(x^*) = 0$
- Define the Lagrangian

$$L(x, \lambda) = J(x) + \lambda^T F(x),$$

  where $\lambda \in \mathbb{R}^m$ are *Lagrange multipliers*.
- the constrained problem is equivalent to unconstrained minimization of L over (x, λ)
- Necessary conditions: (x^*, λ^*) must satisfy

$$\nabla L(x^*, \lambda^*) = 0 \quad \Leftrightarrow \quad \begin{bmatrix} \nabla J(x^*) + \nabla F(x^*)^T \lambda^* \\ F(x^*) \end{bmatrix} = 0 \qquad (6)$$

Equality Constraints (cont.)

- Solved using Newton's method over $(x, \lambda) \in \mathbb{R}^{n+m}$
- Results in the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{bmatrix} H_L & \nabla F^T \\ \nabla F & 0 \end{bmatrix} \begin{bmatrix} p \\ \bar{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla J \\ -F \end{bmatrix}, \tag{7}$$

where the Hessian $H_L = \nabla_x^2 L$ is defined by

$$H_L = \nabla^2 J(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 F_i(x)$$

Equality Constraints (cont.)

- Solved using Newton's method over $(x, \lambda) \in \mathbb{R}^{n+m}$
- Results in the Karush-Kuhn-Tucker (KKT) conditions

$$\begin{bmatrix} H_L & \nabla F^T \\ \nabla F & 0 \end{bmatrix} \begin{bmatrix} p \\ \bar{\lambda} \end{bmatrix} = \begin{bmatrix} -\nabla J \\ -F \end{bmatrix}, \tag{7}$$

where the Hessian $H_L = \nabla_x^2 L$ is defined by

$$H_L = \nabla^2 J(x) + \sum_{i=1}^{m} \lambda_i \nabla^2 F_i(x)$$

- Equivalent quadratic programming (QP) form: find d to minimize

$$\frac{1}{2} d^T H_L d + \nabla J^T d, \qquad \textit{subject to}: \ \nabla F^T d = -F.$$

- Step-size again determined by line-search or trust region.
- *Trust region*: constrain d such that $\|Dd\| \leq \Delta$ where D is a diagonal scaling matrix and Δ is the radius of the trust region.

Inequality Constraints

- Minimize $J(x)$ subject to inequality constraints $F(x) \geq 0$
- the dimension of which can now be greater than n
- At the optimum x^* we have:
  - *active* constraints: $F_i(x^*) = 0$
  - *inactive* constraints: $F_i(x^*) > 0$.

Inequality Constraints

- Minimize $J(x)$ subject to inequality constraints $F(x) \geq 0$
- the dimension of which can now be greater than n
- At the optimum x^* we have:
  - *active* constraints: $F_i(x^*) = 0$
  - *inactive* constraints: $F_i(x^*) > 0$.
- During iteration, at each $\bar{x}$, the simplest scheme:
  - find active constraints at $\bar{x}$
  - ignore inactive, and solve the equality-constraint problem
- Methods such as (Sequential Quadratic Programming) SQP maintain an active set index.
- Computing the active set or predicting the active/inactive switch is nontrivial.

Outline

Intro

Optimization
- Unconstrained
- Equality Constraints
- Inequality Constraints

Direct Methods
- Direct Shooting
- Direct Multiple-Shooting
- Direct Transcription/Collocation

Special cases
- Differentially flat systems
- Constraint Homotopy

Direct Methods

- finite-dimensional representation of the continuous optimal control problem
- discretizing time and solving a discrete-time optimal control problems
- or by parametrizing the controls using a finite set of parameters
- a finite-dimensional nonlinear optimization problem
- solution through *nonlinear programming* (NLP) methods.

Direct Shooting

Parametrize the control signal $u(t)$ using a finite number of parameters $p \in \mathbb{R}^c$. For instance, in simple cases (e.g. a point-mass vehicle in $\mathbb{R}^2$):

$$u(t) = p_1 + p_2 t,$$

where $p = (p_1, p_2) \in \mathbb{R}^4$. More generally:

$$u(t) = \sum_{k=1}^{M} p_k B_k(t),$$

where M is finite and $B_k(t)$ are a set of basis functions, such as B-splines. The unknowns in the problem are then

$$\text{NLP variables} = [p, t_f]$$

The state $x(t)$ is obtained using forward integration. The NLP problem:

$$J = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t), \tag{8}$$

subject to:

$$\psi(x(t_f), t_f) \leq 0, \tag{9}$$

Note that there is no systematic way to incorporate path constraints.

Direct Shooting

The pros and cons of the methods are:

Pros.

- *low dimension*: often difficult large-scale problems can be formulated using relatively low number of parameters p
- *simplicity*: straightforward to implement

Cons.

- *sensitivity*: small changes early in the trajectory result in large deviations at later times and cause problems for gradient based methods which try to stabilize the resulting boundary conditions
- *instability*: when the dynamics is unstable, trajectories could vary substantially, numerical round-off errors are also an issue
- *path constraints*: difficult to treat since they introduce constraints sub-arcs that must be handled with additional multipliers which must be set up defined a-priori
- *loss of sparsity*: the resulting nonlinear equations to be solved are typically complicated and non-sparse so it is not possible to gain efficiency using sparse solvers

Direct Multiple-Shooting

- choose discrete times $[t_0, t_1, t_2, \ldots, t_N]$ where $t_N = t_f$, as well as discrete trajectory $[x_0, x_1, \ldots, x_N]$ and a discrete set of parametrized control $[u(p_0), u(p_1), \ldots, u(p_N)]$.
- perform a direct shooting step within each interval $[t_i, t_{i+1}]$ by integrating the dynamics starting from x_i and obtaining, say $\bar{x}_i$ at time t_{i+1}, for each i.
- then we solve the root finding problem with optimization variables

$$p_0, p_1, \ldots, p_{N-1}, x_1, x_2, \ldots, x_N, t_f$$

and the equations to be solved are

$$\begin{bmatrix} \bar{x}_0 - x_1 \\ \vdots \\ \bar{x}_{N-1} - x_N \\ \psi(x_N, t_N) \end{bmatrix} = 0, \tag{10}$$

Direct Multiple-Shooting

The pros and cons of the methods are similar to indirect shooting
Pros.

- *robustness*: issues related to stability are alleviated
- *efficiency*: sparsity is now introduced through the extra NLP variables since the problem is dense only within segments, and block-sparse overall

Cons.

- *dimensionality*: the problem dimension is increased and becomes more computationally expensive, but one can exploit *sparsity* is the root-finding problem
- *path constraints:* it is still necessary to define constrained-unconstrained arcs in advance (unless a whole interval $[t_i, t_{i+1}]$ can be regarded as either constrained or unconstrained, but this might requires very fine discretization)

Direct Transcription/Collocation

We start by choosing discrete times $[t_0, t_1, t_2, \ldots, t_N]$ and defining the optimization variables

$$\text{NLP variables} = [u_0, x_1, u_1, \ldots, x_N, u_N, t_f].$$

We replace the nonlinear dynamics with a finite difference approximation, e.g. the trapezoidal rule

$$x_{i+1} - x_i - h_i \frac{f(x_i, u_i, t_i) + f(x_{i+1}, u_{i+1}, t_{i+1})}{2} = 0,$$

or midpoint

$$x_{i+1} - x_i - h_i f\left(\frac{x_i + x_{i+1}}{2}, \frac{u_i + u_{i+1}}{2}, \frac{t_i + t_{i+1}}{2}\right) = 0,$$

or another higher-order method.

Direct Transcription/Collocation

Similarly, the cost function is approximated, e.g. by

$$\int_{t_i}^{t_{i+1}} L(x,u,t)dt \approx h_i \frac{L(x_i,u_i,t_i)+L(x_{i+1},u_{i+1},t_{i+1})}{2},$$

or

$$\int_{t_i}^{t_{i+1}} L(x,u,t)dt \approx h_i L\left(\frac{x_i+x_{i+1}}{2}, \frac{u_i+u_{i+1}}{2}, \frac{t_i+t_{i+1}}{2}\right),$$

or another higher-order method. We can then solve the NLP:

$$\phi(x_N,t_N)+\sum_{i=0}^{N-1} L_i(x_i,x_{i+1},u_i,u_{i+1}), \tag{11}$$

subject to:

$$S_i(x_i,x_{i+1},u_i,u_{i+1})=0 \tag{12}$$

$$c(x_i,u_i,t_i)\leq 0, \text{ for all } i=0,\ldots,N \tag{13}$$

$$\psi(x_N,t_N)\leq 0, \tag{14}$$

where S_i and L_i correspond to the discrete approximations of the dynamics and cost function, e.g. using the midpoint, trapezoidal, or a higher-order scheme. The solution can generally be computed using SQP or interior-point (IP).

Direct Transcription/Collocation

The pros and cons of the methods can be summarized as follows Pros.

- *efficiency*: the problem is complex, large-scale but still sparse
- *simplicity*: no need to derive adjoint equations in terms of λ, can use x directly (although technically SQP internally uses multipliers)
- *stability*: unstable dynamics remedied since all x are simultaneously varied
- *path constraints*: easier to handle using existing NLP methods (SQP, IP)

Cons.

- *adaptation*: time grid must be chosen in advance and often difficult to adapt (e.g. where higher accuracy is required) during optimization, since this changes the problem dimension, i.e. more points can't be easily added
- *careful implementation*: to obtain efficiency one still needs derivatives and sparsity structure which must be carefully specified. Recent software packages are starting to provide automated ways to deal with this though.

Outline

Intro

Optimization
- Unconstrained
- Equality Constraints
- Inequality Constraints

Direct Methods
- Direct Shooting
- Direct Multiple-Shooting
- Direct Transcription/Collocation

Special cases
- Differentially flat systems
- Constraint Homotopy

Differentially flat systems

- Find flat outputs $y(t) \in \mathbb{R}^m$ such that

$$x = a(y, \dot{y}, y^{(2)}, \ldots, y^{(q)}), \quad u = b(y, \dot{y}, y^{(2)}, \ldots, y^{(r)})$$

- Construct finite-dimensional flat trajectory representation

$$y(t) = \sum_{k=1}^{K} B_k(t) p_k,$$

where $B_k(t)$ are basis functions (e.g. B-splines) and $p_k \in \mathbb{R}^m$ are free parameters. This can also be written as $y(t) = Pb(t)$, where $P \in \mathbb{R}^{m \times K}$ is a parameter matrix and $b(t) = (B_1(t), \ldots, B_K(t)) \in \mathbb{R}^K$

- Set $x(t) = a(Pb(t), P\dot{b}(t), \ldots), \quad u(t) = b(Pb(t), P\dot{b}(t), \ldots)$ and solve:

$$\min_{P, t_f} J = \phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t) dt, \tag{15}$$

subject to:

$$x(t_0) = x_0, \quad x(t_f) \text{ and } t_f \text{ free} \tag{16}$$

$$c(x(t), u(t), t) \leq 0, \text{ for all } t \in [t_0, t_f] \tag{17}$$

$$\psi(x(t_f), t_f) \leq 0, \tag{18}$$

Optimization on SO(3): three parametrization methods

Optimize curve $R(t) \in SO(3)$ but avoid Euler angle singularities?

1. Use quaternions $c = \phi(R) \in S^3$, add constraint $c^T c - 1 = 0$.
2. Sequence of Lie algebra flows: $\xi_0(\cdot), \xi_1(\cdot), \cdots \in \mathbb{R}^3$
   - first let $R_0, R_1, \ldots, R_N \in SO(3)$ be discrete rotations along $R(t)$
   - define $\xi_i : [0, \Delta_i] \to \mathbb{R}^3$ on the interval $t \in [t_i, t_{i+1}]$ so that

     $$R(t) = R_i \exp(\xi_i(t - t_i)), \text{ with } \xi_i(0) = 0, \ \xi_i(\Delta t_i) = \log(R_i^{-1} R_{i+1}),$$

     for $\Delta t_i \triangleq t_{i+1} - t_i$ and derivative conditions on the knots defined by

     $$R_i J(\xi_i(\Delta t_i)) \dot{\xi}_i(\Delta t_i) = R_{i+1} \dot{\xi}_{i+1}(0),$$

     where $J(\xi)$ is the left (translated) Jacobian of the exp defined as

     $$J(\xi) \cdot \exp(\xi)\delta = \partial \exp(\xi) \cdot \delta$$

3. Instead of optimizing over $R(t)$, we define $R(t) = \bar{R}(t) \exp(\eta(t))$ and optimize over $\eta(t) \in \mathbb{R}^3$ where $\bar{R}(t)$ is an initial guess.
   In the discrete setting we have
   $R(\cdot) \approx (R_0, R_1, \ldots, R_N) = (\bar{R}_0 \exp(\eta_0), \bar{R}_1 \exp(\eta_1), \ldots, \bar{R}_N \exp(\eta_N))$,
   so the NLP variables are $(\eta_0, \eta_1, \ldots, \eta_N)$.

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

- Example: *distance function* $F(x) = \rho(x, \mathcal{O}_i) \geq 0$ to obstacle $\mathcal{O}_i$

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

- Example: *distance function* $F(x) = \rho(x, \mathcal{O}_i) \geq 0$ to obstacle $\mathcal{O}_i$
  - Consider the homotopy, where $x_c \in X$ as the centroid of the obstacle

$$\rho^\lambda(x, \mathcal{O}_i) = \lambda \|x - x_c\| + (1 - \lambda)\rho(x, \mathcal{O}_i),$$

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

- Example: *distance function* $F(x) = \rho(x, \mathcal{O}_i) \geq 0$ to obstacle $\mathcal{O}_i$
  - Consider the homotopy, where $x_c \in X$ as the centroid of the obstacle

$$\rho^\lambda(x, \mathcal{O}_i) = \lambda \|x - x_c\| + (1 - \lambda)\rho(x, \mathcal{O}_i),$$

  - ρ^λ "smooths" ρ by enclosing the obstacle in a ball

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

- Example: *distance function* $F(x) = \rho(x, \mathcal{O}_i) \geq 0$ to obstacle $\mathcal{O}_i$
  - Consider the homotopy, where $x_c \in X$ as the centroid of the obstacle

$$\rho^\lambda(x, \mathcal{O}_i) = \lambda \|x - x_c\| + (1 - \lambda)\rho(x, \mathcal{O}_i),$$

  - ρ^λ "smooths" ρ by enclosing the obstacle in a ball
  - gradually growing it back to its original form as λ goes from 1 to 0.

Dealing with complex constraints

- Non-convex or non-smooth constraints complicate the optimization
- Alleviated through constraint *homotopy* $H : \mathbb{R}^{n+1} \to \mathbb{R}^m$

$$H(x, 1) = \widetilde{F}(x), \quad H(x, 0) = F(x),$$

where the smooth map $\widetilde{F} : \mathbb{R}^n \to \mathbb{R}^m$ is a relaxed version of F.

- For example, a commonly used homotopy is

$$H(x, \lambda) = \lambda \widetilde{F}(x) + (1 - \lambda) F(x).$$

The goal is to trace the implicitly defined curve $c(s) \in H^{-1}(0)$ from the initial solution $(x_1, 1)$ to the actual solution $(\bar{x}, 0)$.

- Example: *distance function* $F(x) = \rho(x, \mathcal{O}_i) \geq 0$ to obstacle $\mathcal{O}_i$
  - Consider the homotopy, where $x_c \in X$ as the centroid of the obstacle

$$\rho^\lambda(x, \mathcal{O}_i) = \lambda \|x - x_c\| + (1 - \lambda)\rho(x, \mathcal{O}_i),$$

  - ρ^λ "smooths" ρ by enclosing the obstacle in a ball
  - gradually growing it back to its original form as λ goes from 1 to 0.
- To implement: use H instead of F; add λ to optimization vector $\xi' = (\xi, \lambda)$; initialize $\lambda = 1$ and enforce final constraint $\lambda = 0$.