Motion Planning and Control for Robotics

Trajectory Optimization

Kyle Stachowicz

October 5, 2020

RoboJackets

Getting Started

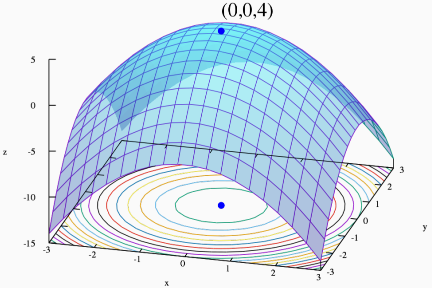
Exercises: *https://bit.ly/2S2v58j*

Review

Review: Optimization

Mathematical optimization is the process of finding the minimum or maximum value of some function, as well as the argument that gives that minimum or maximum.

$$4 = \max_{x,y} 4 - x^2 - y^2$$

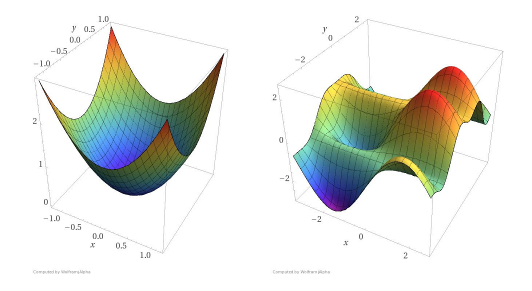


Review: Convexity

A function is **convex** if for any two points p and q on the function, the line between p and q lies entirely above the graph of the function.

A convex function has a single global minimum.

$$q \leq x \leq p \implies f(x) \leq (1-t)f(q) + tf(p)$$

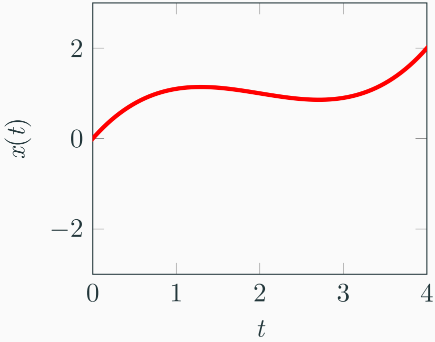


Review: Terminology

Name	Symbol	Description
Dynamics	$f(x_t, u_t)$	Evolution of the state.
Running cost	$\ell(x, u)$	The instantaneous cost (rate).
Horizon	T	How many timesteps to compute.
Terminal cost	$\Phi(x)$	One-time cost for the final state.
Cost-to-go	$J_t(x)$	The minimum remaining total cost if the system is in state x at time t.

Review: Bellman Equation and Cost-to-go

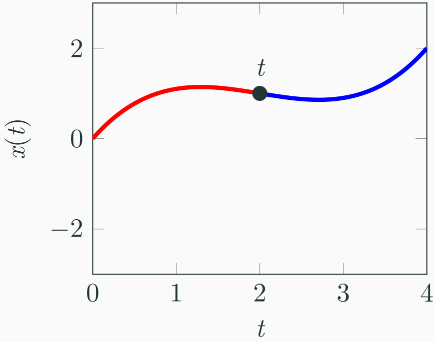
If we have an optimal trajectory $x^*(t)$...



Review: Bellman Equation and Cost-to-go

If we have an optimal trajectory $x^*(t)$...

And we break it up into parts...

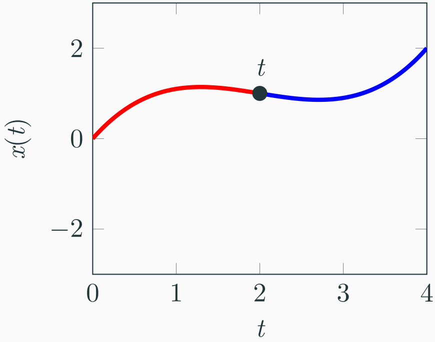


Review: Bellman Equation and Cost-to-go

If we have an optimal trajectory $x^*(t)$...

And we break it up into parts...

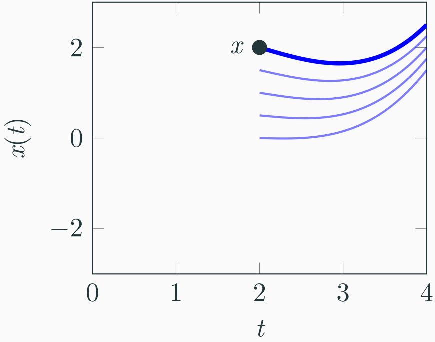
The piece starting at t and ending at T is optimal, for any trajectory starting in x at t.



Review: Bellman Equation and Cost-to-go

The **cost-to-go** at time t is the function $J_t(x)$ that gives you the optimal cost, for starting in an arbitrary state x, at time t.

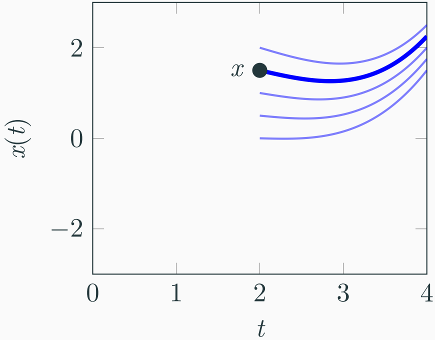
$$J_t(x) = \min_u \left[\int_t^T \ell(x, u)\, dt + \Phi(x_T) \right]$$



Review: Bellman Equation and Cost-to-go

The **cost-to-go** at time t is the function $J_t(x)$ that gives you the optimal cost, for starting in an arbitrary state x, at time t.

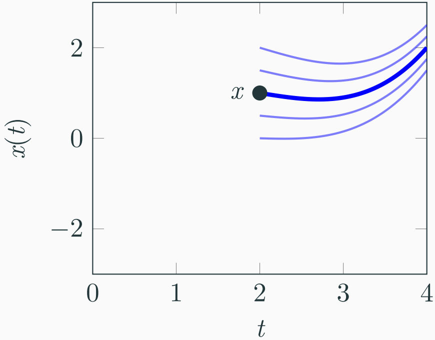
$$J_t(x) = \min_u \left[\int_t^T \ell(x,u)\,dt + \Phi(x_T) \right]$$



Review: Bellman Equation and Cost-to-go

The **cost-to-go** at time t is the function $J_t(x)$ that gives you the optimal cost, for starting in an arbitrary state x, at time t.

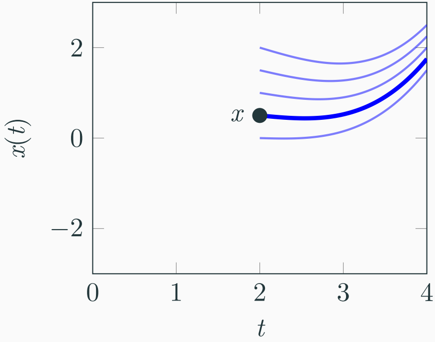
$$J_t(x) = \min_u \left[\int_t^T \ell(x, u)\, dt + \Phi(x_T) \right]$$



Review: Bellman Equation and Cost-to-go

The **cost-to-go** at time t is the function $J_t(x)$ that gives you the optimal cost, for starting in an arbitrary state x, at time t.

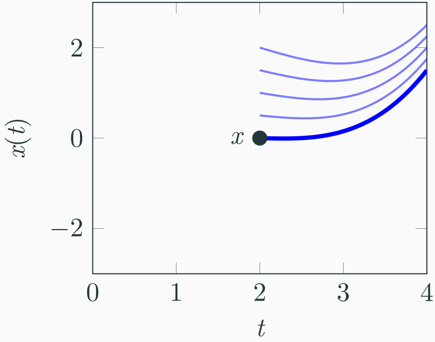
$$J_t(x) = \min_u \left[\int_t^T \ell(x,u)\,dt + \Phi(x_T) \right]$$



Review: Bellman Equation and Cost-to-go

The **cost-to-go** at time t is the function $J_t(x)$ that gives you the optimal cost, for starting in an arbitrary state x, at time t.

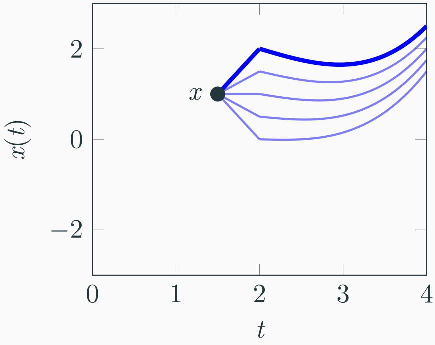
$$J_t(x) = \min_u \left[\int_t^T \ell(x,u)\,dt + \Phi(x_T) \right]$$



Review: Bellman Equation and Cost-to-go

If we know all possible best-case paths at t, we can find the best cases for a particular state at x_{t-1} (i.e., calculate $J_{t-1}(x)$ for a particular value of x).

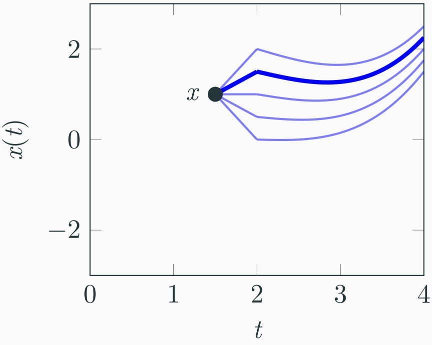
$$J_{t-1}(x) = \min_u \left[\ell(x, u) + J_t(f(x, u))\right]$$



Review: Bellman Equation and Cost-to-go

If we know all possible best-case paths at t, we can find the best cases for a particular state at x_{t-1} (i.e., calculate $J_{t-1}(x)$ for a particular value of x).

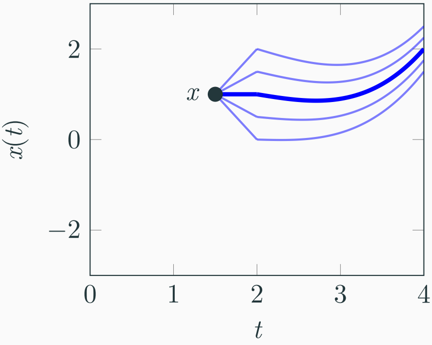
$$J_{t-1}(x) = \min_u \left[\ell(x, u) + J_t(f(x, u))\right]$$



Review: Bellman Equation and Cost-to-go

If we know all possible best-case paths at t, we can find the best cases for a particular state at x_{t-1} (i.e., calculate $J_{t-1}(x)$ for a particular value of x).

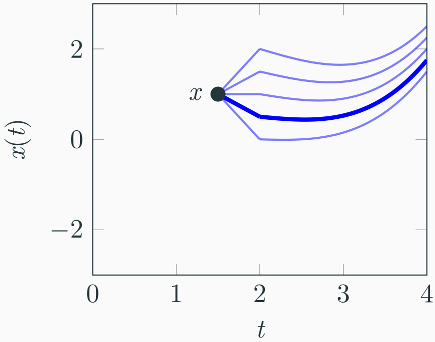
$$J_{t-1}(x) = \min_u \left[\ell(x, u) + J_t(f(x, u))\right]$$



Review: Bellman Equation and Cost-to-go

If we know all possible best-case paths at t, we can find the best cases for a particular state at x_{t-1} (i.e., calculate $J_{t-1}(x)$ for a particular value of x).

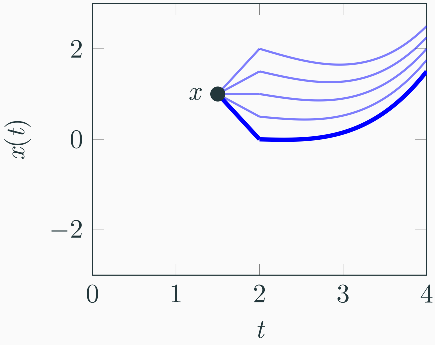
$$J_{t-1}(x) = \min_u \left[\ell(x, u) + J_t(f(x, u))\right]$$



Review: Bellman Equation and Cost-to-go

If we know all possible best-case paths at t, we can find the best cases for a particular state at x_{t-1} (i.e., calculate $J_{t-1}(x)$ for a particular value of x).

$$J_{t-1}(x) = \min_u \left[\ell(x,u) + J_t(f(x,u))\right]$$



Review: LQR

$$x_{t+1} = f(x_t, u_t) = Ax_t + Bu_t$$

$$\ell(x, u) = \frac{1}{2}x^T Qx + \frac{1}{2}u^T Ru$$

$$\Phi(x_T) = \frac{1}{2}x_t^T Q_f x_t$$

$$J_t(x) = \frac{1}{2}x_t^T S_t x_t$$

Update equation:

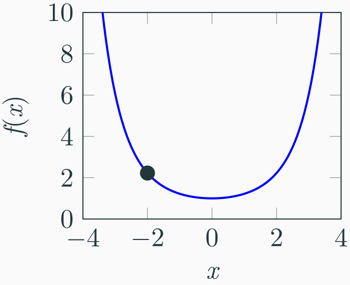
$$S_T = Q_f$$

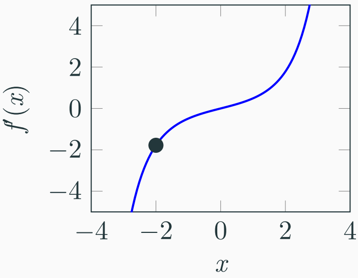
$$S_{t-1} = A^T S_t A - A^T S_t B(R + B^T S_t B)^{-1} B^T S_t A + Q$$

Iterative Methods and iLQR

Newton's Method: Iterative Approximation

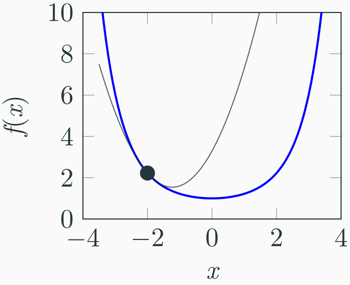
To minimize a function, we find its derivative and find where it is zero. Newton's method is a way of *iteratively* solving that (or any) equation.

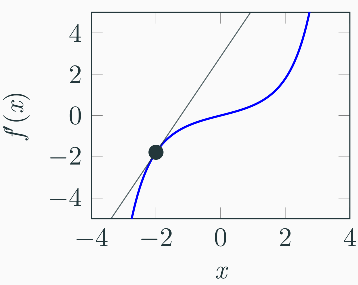




Newton's Method: Iterative Approximation

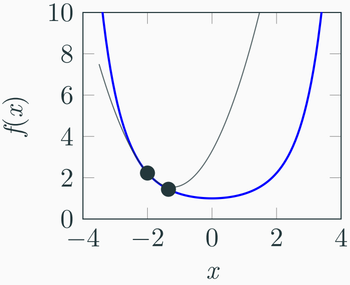
To minimize a function, we find its derivative and find where it is zero. Newton's method is a way of *iteratively* solving that (or any) equation.

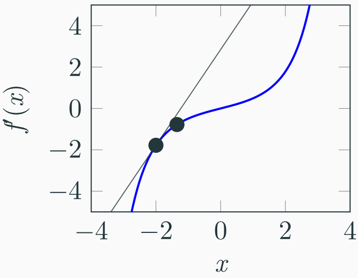




Newton’s Method: Iterative Approximation

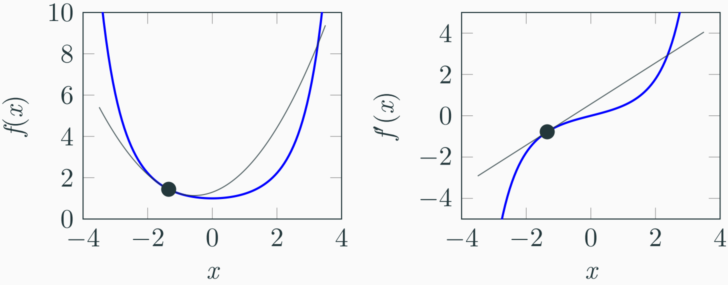
To minimize a function, we find its derivative and find where it is zero. Newton’s method is a way of *iteratively* solving that (or any) equation.





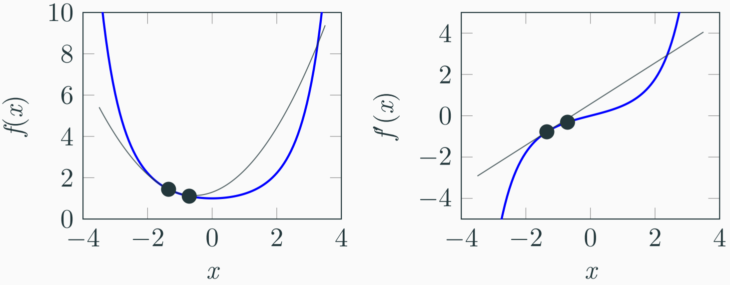
Newton’s Method: Iterative Approximation

To minimize a function, we find its derivative and find where it is zero. Newton’s method is a way of *iteratively* solving that (or any) equation.



Newton’s Method: Iterative Approximation

To minimize a function, we find its derivative and find where it is zero. Newton’s method is a way of *iteratively* solving that (or any) equation.



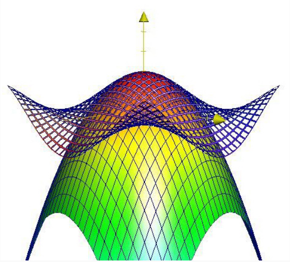
Taylor Series in Multiple Dimensions

For a function $f : \mathbb{R}^n \to \mathbb{R}$, we can approximate it around a point x_0 as...

$$f(x) \approx \frac{1}{2}(x - x_0)^T \text{hess}(f)(x - x_0) + \text{grad}(f)(x - x_0) + f(x_0)$$

In this case, hess f is like the "second derivative" (matrix of partial derivatives $\frac{\partial^2 f}{\partial x_i \partial x_j}$):

$$\text{hess} f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix}$$



Jacobian Matrix

We can also approximate a function $f\colon \mathbb{R}^n \to \mathbb{R}^n$ with the *Jacobian matrix* $\frac{\partial f}{\partial x}$, which is the matrix $\frac{\partial f_i}{\partial x_j}$.

$$\frac{\partial}{\partial x} f = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$$

This gives us $f(x) \approx \frac{\partial f}{\partial x}(x - x_0) + f(x_0)$.

Iterative LQR

Idea: approximate dynamics as linear and cost as quadratic, then solve LQR!

Iterative LQR

Idea: approximate dynamics as linear and cost as quadratic, then solve LQR!

Problem: Where do we linearize/quadratize around?

Iterative LQR

Idea: approximate dynamics as linear and cost as quadratic, then solve LQR!

Problem: Where do we linearize/quadratize around?

Answer: The trajectory from the last iteration!

Iterative LQR

Intuition: set up some "nominal trajectory" that follows the full nonlinear dynamics. Then create a "delta trajectory" $\delta x, \delta u$ that evolves based on the *linearized* dynamics (around the nominal trajectory).

1. $\delta x = 0, \delta u = 0$ is a fixed point
2. $\delta \ell$ is the *change* in running cost from the nominal trajectory
3. δJ_t is the *change* in cost-to-go from the nominal trajectory

We still want to minimize δJ_t (make as large of change as possible, in the negative direction).

We will propagate *cost-to-go* backwards (as usual), and then propagate *dynamics* forwards to find a new nominal trajectory.

Iterative LQR

Notation: drop time index subscripts for t ($x = x_t$), let prime denote $t+1$ (i.e. $J_{t+1} = J'$). Denote partial derivatives with subscript x and u.

$$\delta u = u - u^*, \delta\ell(\delta x, \delta u) = \ell(x, u) - \ell(x^*, u^*), \delta J = J(x) - J(x^*)$$

$$\delta x_{t+1} = x_{t+1} - x^*_{t+1} \approx f_x \delta x + f_u \delta u$$

$$\delta\ell(x, u) \approx \frac{1}{2}\delta x^T \ell_{xx}\delta x + \ell_x \delta x + \frac{1}{2}\delta u^T \ell_{uu} + \ell_u \delta u + \delta x^T \ell_{xu}\delta u$$

Iterative LQR

Bellman Equation:

$$\delta J_t(x) \approx \min_{\delta u}[\frac{1}{2}\delta x^T \ell_{xx}\delta x + \ell_x \delta x + \frac{1}{2}\delta u^T \ell_{uu}\delta u + \ell_u \delta u + \delta x^T \ell_{xu}\delta u + \delta J_{t+1}(f_x \delta x + f_u \delta u)]$$

Iterative LQR

We can solve in the same way we solved LQR, but it's just a bunch of messy algebra...

Define Q as the the inside of the $\min$ above:

$$Q(\delta x, \delta u) = \frac{1}{2}\delta x^T \ell_{xx} \delta x + \ell_x \delta x + \frac{1}{2}\delta u^T \ell_{uu} \delta u + \ell_u \delta u + \dots$$

And group terms:

$$\begin{aligned} Q_{uu} &= \ell_{uu} + f_u^T J'_{xx} f_u \quad & Q_u &= \ell_u + J'_x f_u \\ Q_{xx} &= \ell_{xx} + f_x^T J'_{xx} f_x \quad & Q_x &= \ell_x + J'_x f_x \end{aligned}$$

$$Q_{xu} = \ell_{xu} + f_x^T J'_{xx} f_u$$

Exercise (4 minutes): Calculate Q in the notebook.

Find the minimizing value of δu by taking the gradient and setting it to zero:

$$0 = \partial_u Q(x, u) = \delta u^T Q_{uu} + \delta x^T Q_{xu} + Q_u$$

$$\delta u = -Q_{uu}^{-1}(Q_{xu}^T \delta x + Q_u^T)$$

We call $K\delta x = -Q_{uu}^{-1} Q_{xu}^T \delta x$ the *feedback* term and $k = -Q_{uu}^{-1} Q_u^T$ the *feedforward* term.

Exercise (2 minutes): Calculate K and k in the notebook.

Then substitute the our new value for δu into the Bellman equation for δJ:

$$\begin{aligned}\delta J_t(x) &= \frac{1}{2}\delta u^T Q_{uu}\delta u + Q_u\delta u + \frac{1}{2}\delta x^T Q_{xx}\delta x + Q_x\delta x + \delta x^T Q_{xu}\delta u \\ &= \frac{1}{2}\delta x^T(Q_{xx} - K^T Q_{uu}K)\delta x + (Q_x + k^T Q_{xu}^T)\delta x - k^T Q_{uu}k\end{aligned}$$

Iterative LQR

The forward pass is easy: just simulate the system forwards to calculate new values for x^* and u^* using dynamics equations and the feedback rule $u = Kx + k$. We keep a running copy of x called x^*, this is the simulated state (x_t^* is updated in-place after we calculate u^*).

Algorithm 1: iLQR Forwards Pass

x^* = initial value of x;
$K_t = 0$; $k_t = 0$;
for *each timestep* $t = 0..T-1$ **do**
    $\delta u_t^* = K_t(x^* - x_t^*) + k_t$;
    $u_t^* \mathrel{+}= \delta u_t^*$;
    $x_t^* = x^*$;
    $x^* = f(x_t^*, u_t^*)$;
end

The backwards pass is a little longer, but it's mostly just combining things we already have. We also have to keep track of $J_{xx,t}$ and $J_{x,t}$.

Algorithm 2: iLQR Backwards Pass

$\delta J_{xx,T} = \frac{\partial^2 \Phi}{\partial x^2}|_{x_T^*}$;
$\delta J_{x,T} = \frac{\partial \Phi}{\partial x}|_{x_T^*}$;
for *each timestep* $t = T - 1..0$ *(backwards)* **do**
  $Q_{xx}, Q_x, Q_{uu}, Q_u, Q_{xu}$ = CalculateQ($J_{xx,t+1}, J_{x,t+1}, x_t^*, u_t^*$);
  K_t, k_t = OptimizeQ($Q_{xx}, Q_x, Q_{uu}, Q_u, Q_{xu}$);
  $J_{xx,t}, J_{x,t}$ = CalcPrevC2G($Q_{xx}, Q_x, Q_{uu}, Q_u, Q_{xu}, K_t, k_t$);
end

Addendum: Off-the-Shelf Solvers and Model-Predictive Control

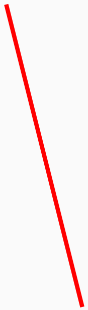
Constrained Optimization

1. So far, no way to directly constrain inputs

Constrained Optimization

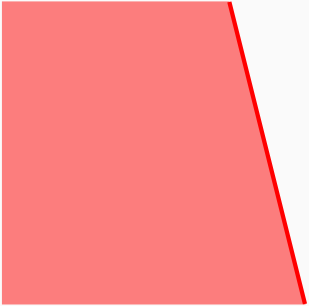
1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



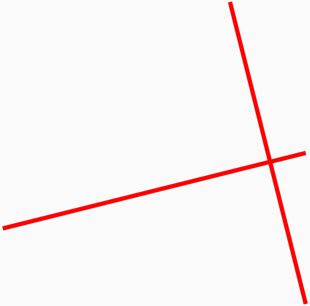
Constrained Optimization

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



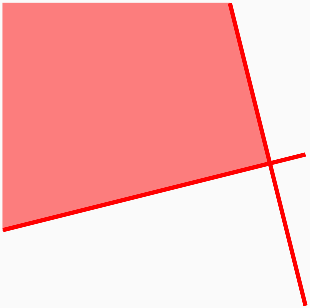
Constrained Optimization

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



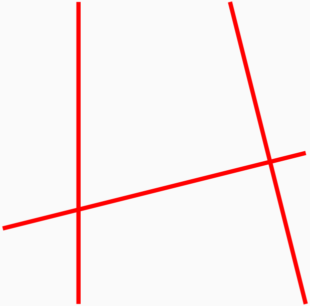
Constrained Optimization

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



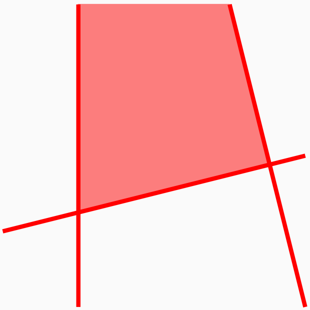
Constrained Optimization

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



Constrained Optimization

1. So far, no way to directly constrain inputs
2. We'd like to be able to specify *input polytopes* with $Hx \leq b$ (all elements of Hx must be less than the corresponding element in b).



Constrained Optimization for Linear Problems

We can write our original optimal control problem as a “Quadratic Program” (quadratic cost, linear inequality and inequality constraints):

$$\min_{x,u} \sum_{i=1}^{T} \left[\frac{1}{2} x^T Q x + q^T x + \frac{1}{2} u^T R u + r^T u \right]$$

Subject to:

$$x_{t+1} = A x_t + B u_t + c$$

$$Hu \leq k$$

Off-the-Shelf Solvers

There are solvers that will just *solve* non-quadratic optimization problems with non-linear constraints!

(Remember: if we can express nonlinear constraints we can have nonlinear dynamics. If we can express non-quadratic cost, we can have better cost functions)

Limits of Direct Trajectory Optimization

What are the drawbacks?

Limits of Direct Trajectory Optimization

What are the drawbacks?

- Can be slow, especially for complex problems.

Limits of Direct Trajectory Optimization

What are the drawbacks?

- Can be slow, especially for complex problems.
- Can fail (return “infeasible”), even if there is a solution that satisfies all of your constraints.

Limits of Direct Trajectory Optimization

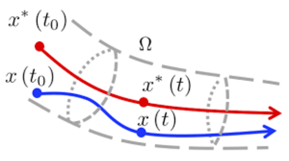
What are the drawbacks?

- Can be slow, especially for complex problems.
- Can fail (return "infeasible"), even if there is a solution that satisfies all of your constraints.
- **Doesn't give us a feedback policy.** (Open-loop only)

Limits of Direct Trajectory Optimization

What are the drawbacks?

- Can be slow, especially for complex problems.
- Can fail (return "infeasible"), even if there is a solution that satisfies all of your constraints.
- **Doesn't give us a feedback policy.** (Open-loop only)



Replanning as Control

Strategy:

- Plan a trajectory from $t = 1$ to $t = T$

(Model-Predictive Control)

Replanning as Control

Strategy:

- Plan a trajectory from $t = 1$ to $t = T$
- Execute u_1

(Model-Predictive Control)

Replanning as Control

Strategy:

- Plan a trajectory from $t = 1$ to $t = T$
- Execute u_1
- Find the new state x_2

(Model-Predictive Control)

Replanning as Control

Strategy:

- Plan a trajectory from $t = 1$ to $t = T$
- Execute u_1
- Find the new state x_2
- Plan a trajectory from $t = 2$ to $t = T + 1$, starting at x_2

(Model-Predictive Control)

Replanning as Control

Strategy:

- Plan a trajectory from $t = 1$ to $t = T$
- Execute u_1
- Find the new state x_2
- Plan a trajectory from $t = 2$ to $t = T + 1$, starting at x_2
- Execute u_2 (the first control input from the new sequence)

(Model-Predictive Control)

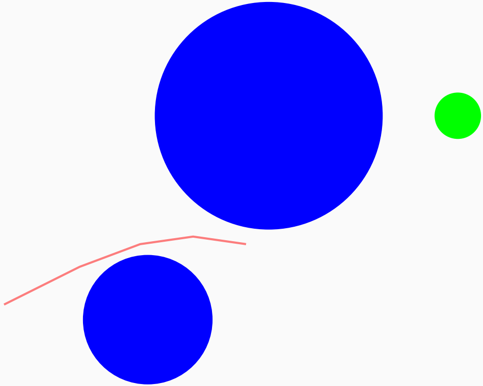
Replanning as Control

Strategy:

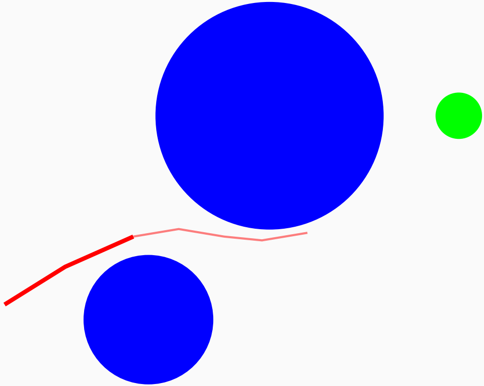
- Plan a trajectory from $t = 1$ to $t = T$
- Execute u_1
- Find the new state x_2
- Plan a trajectory from $t = 2$ to $t = T + 1$, starting at x_2
- Execute u_2 (the first control input from the new sequence)
- Repeat...

(Model-Predictive Control)

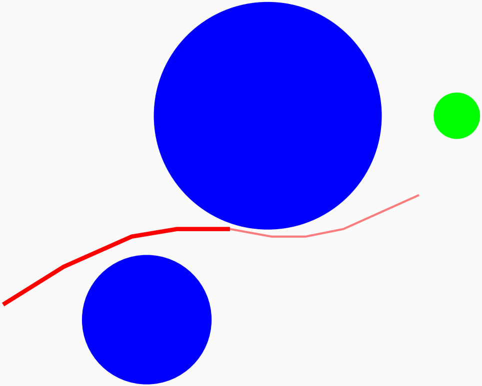
Model-Predictive Control



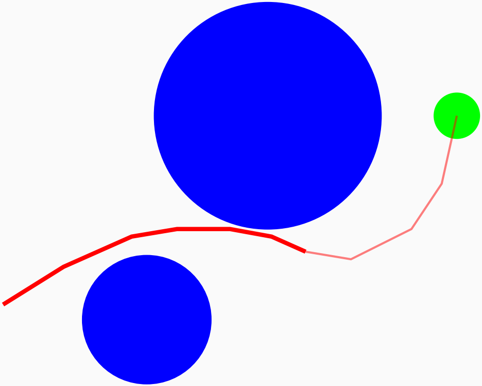
Model-Predictive Control



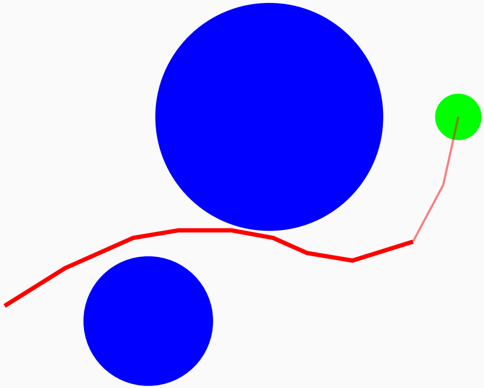
Model-Predictive Control



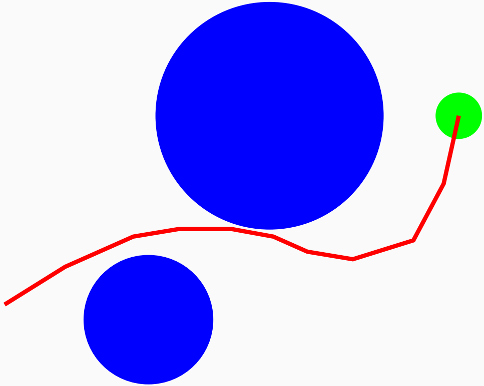
Model-Predictive Control



Model-Predictive Control



Model-Predictive Control



Conclusion

- Optimization-based motion planning is a powerful technique
- We can use LQR (repeatedly) to find good *local* solutions for nonlinear systems
- Off-the-shelf optimizers can be excellent…but sometimes slow
- Constant replanning (MPC) can turn open-loop sequences into a feedback technique

Resources

- Russ Tedrake's lectures: *http://underactuated.mit.edu*
- UIUC motion planning notes: *http://planning.cs.uiuc.edu*