# Motion Planning Lecture 10

Optimization-Based Motion Planning

Wolfgang Hönig (TU Berlin) and Andreas Orthey (Realtime Robotics)

June 26, 2024

# Recap Last Week

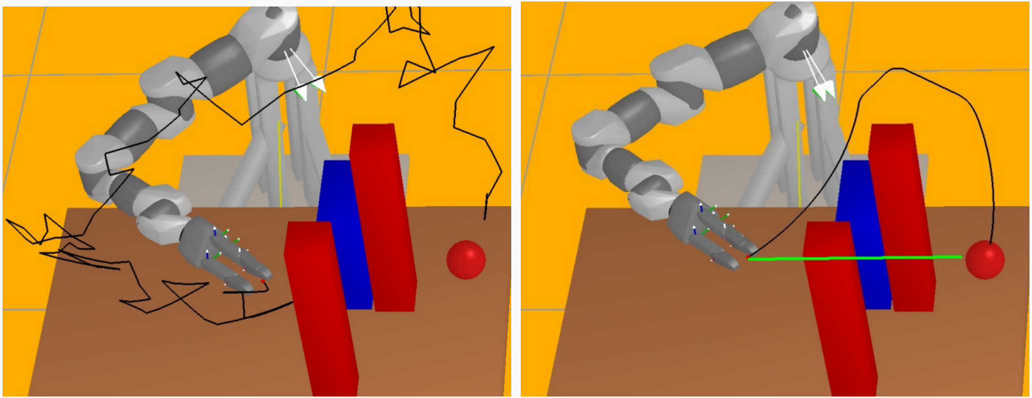
**Last Week**

- Completeness and Convergence of RRT
- Proof of asymptotic optimality
- Advanced sampling-based planners (LazyPRM, FMT*)

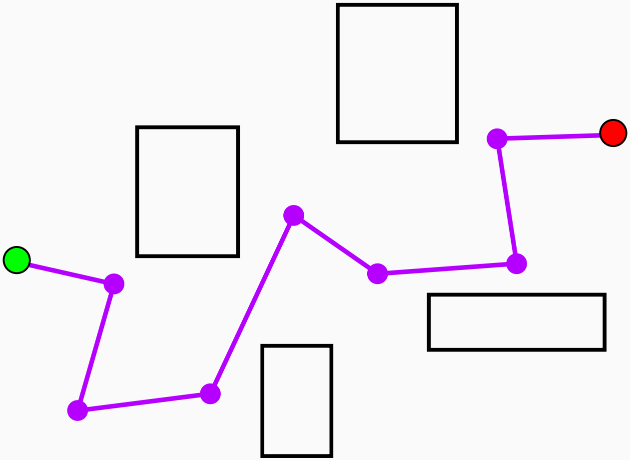
**Today**

- Introduction to Convex optimization
- Optimization-based motion planning
- Splines, Signed-Distance Field, Gradients

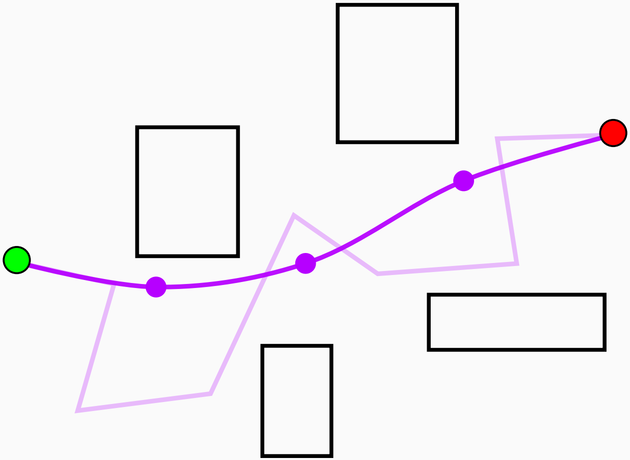
# Optimization-based Motion Planning



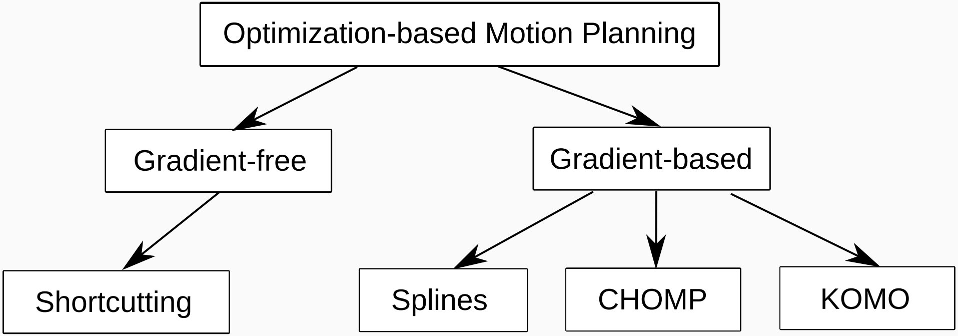
# Optimization-based Motion Planning



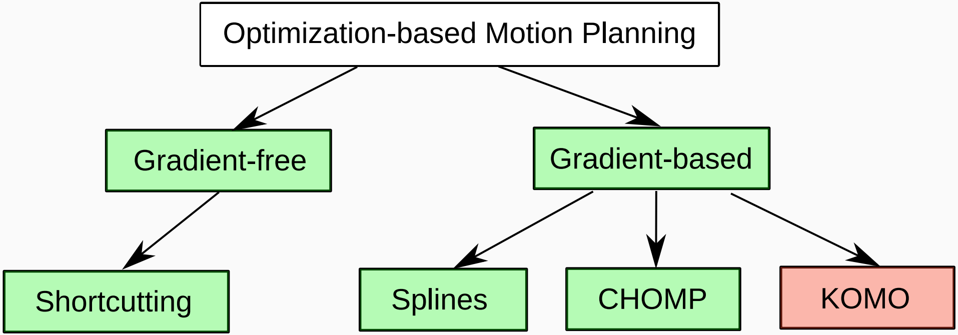
# Optimization-based Motion Planning



# Optimization-based Motion Planning



# Optimization-based Motion Planning



# Optimization using Shortcutting

# Geometric Path Improvement

## Shortcutting

- Vanilla version
  - Sample two waypoints
  - Try to connect them and update path
  - Repeat until timeout or convergence

- Easy to implement
- Does not take complex cost functions into account
- Too few waypoints:
- Too many waypoints:

# Geometric Path Improvement

## Shortcutting

- Vanilla version
  - Sample two waypoints
  - Try to connect them and update path
  - Repeat until timeout or convergence

- Easy to implement
- Does not take complex cost functions into account
- Too few waypoints: Hard to connect
- Too many waypoints:

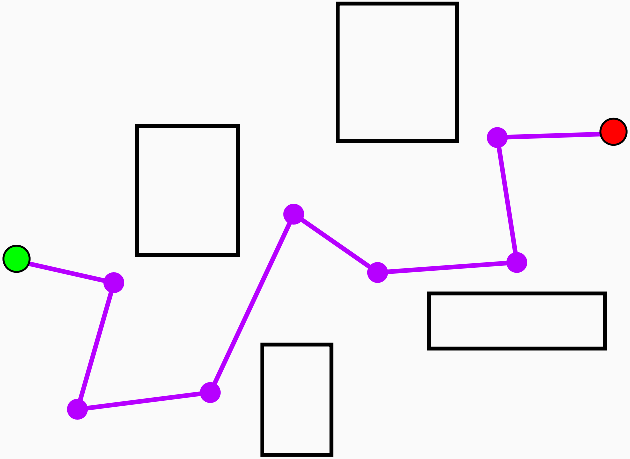
# Geometric Path Improvement

## Shortcutting

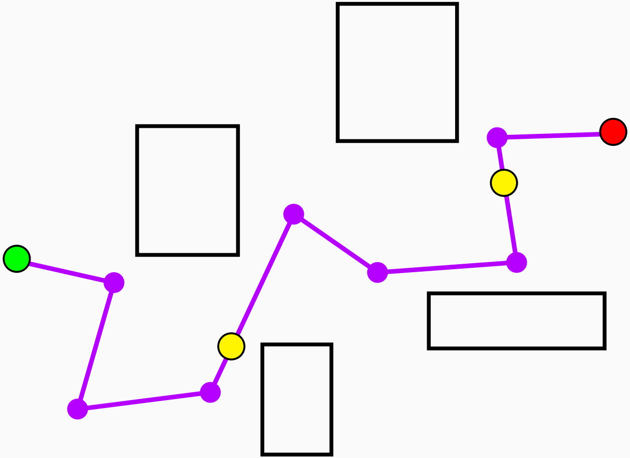
- Vanilla version
  - Sample two waypoints
  - Try to connect them and update path
  - Repeat until timeout or convergence

- Easy to implement
- Does not take complex cost functions into account
- Too few waypoints: Hard to connect
- Too many waypoints: Long computational time

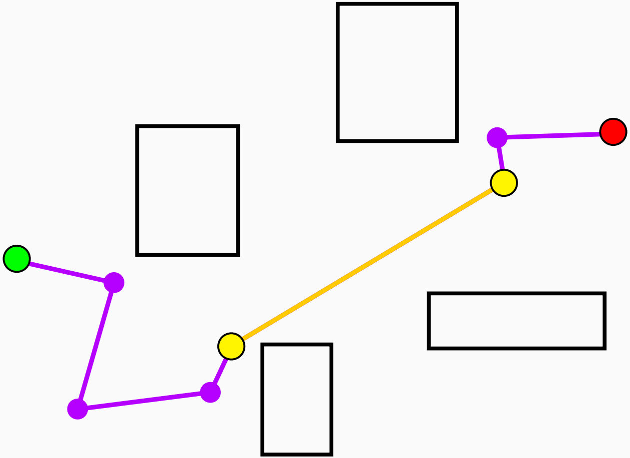
# Shortcutting



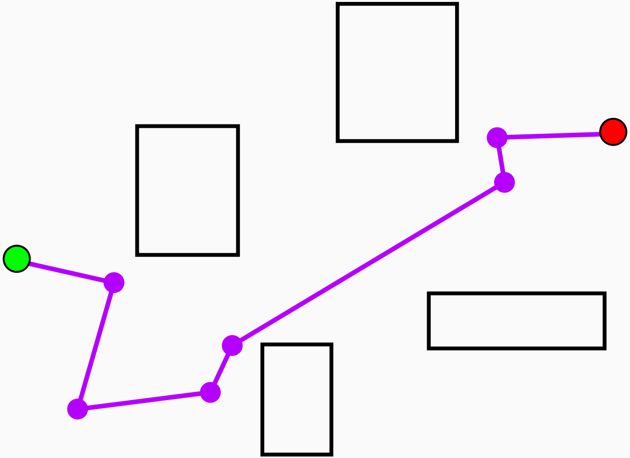
# Shortcutting



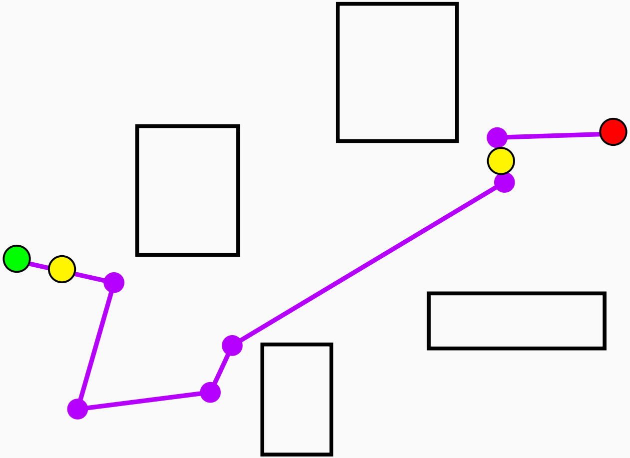
# Shortcutting



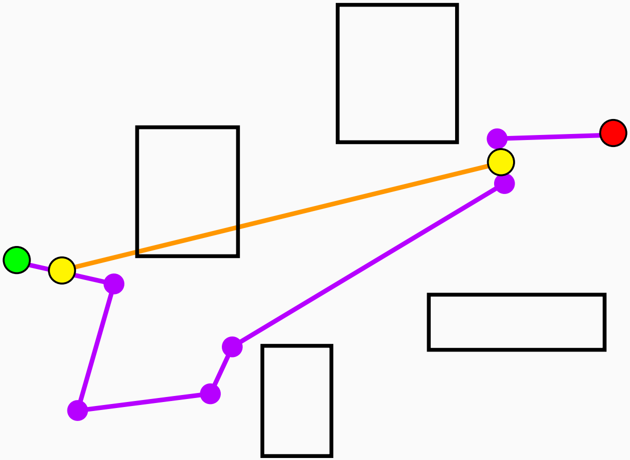
# Shortcutting



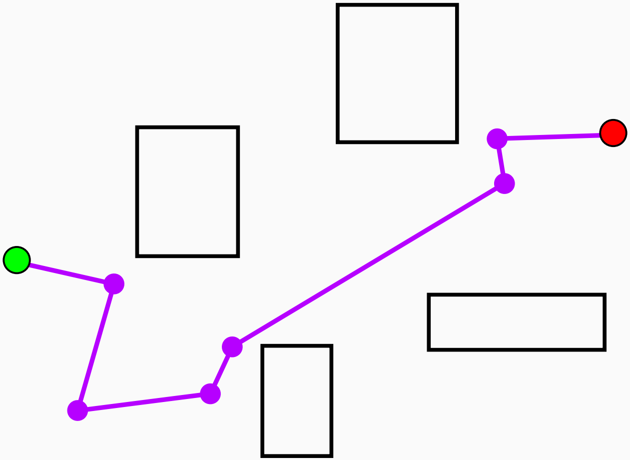
# Shortcutting



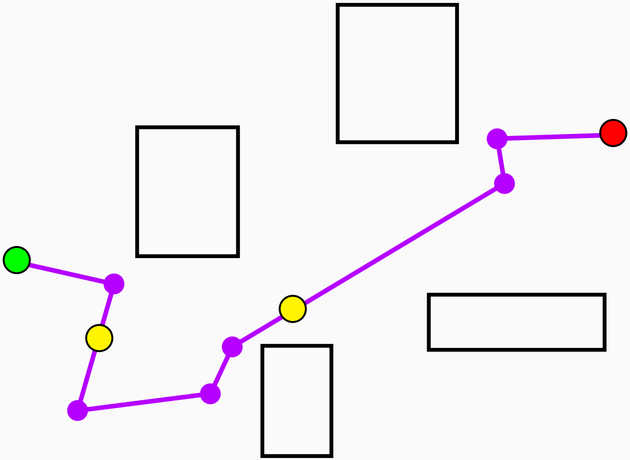
# Shortcutting



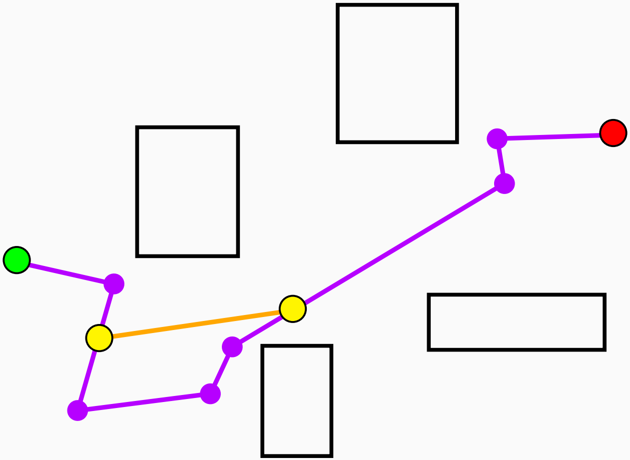
# Shortcutting



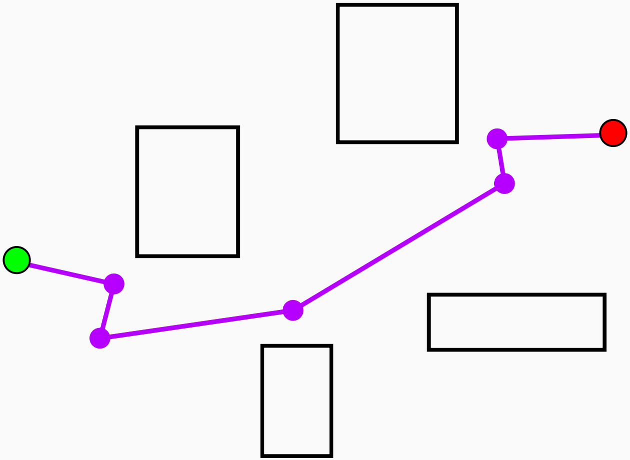
# Shortcutting



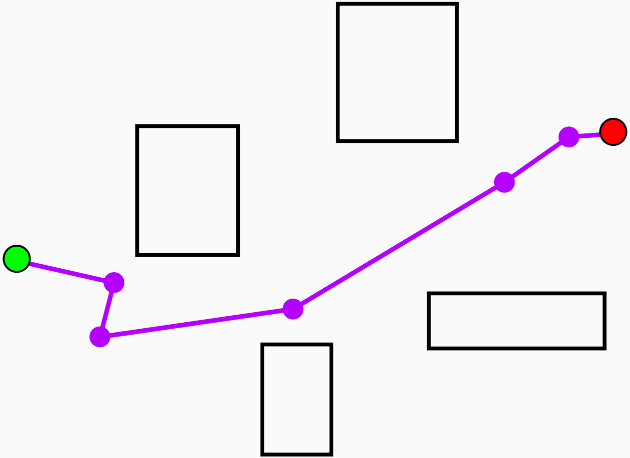
# Shortcutting



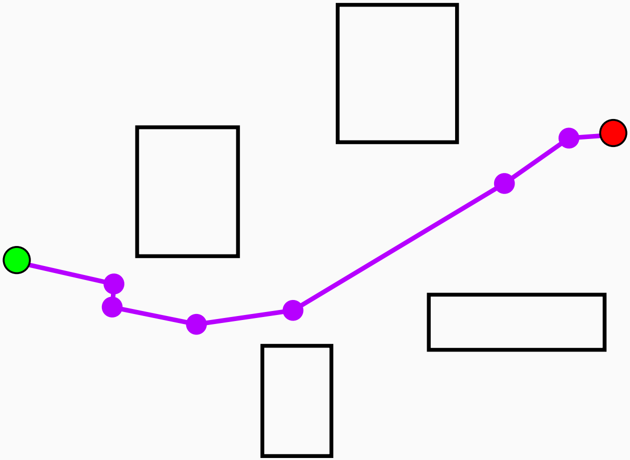
# Shortcutting



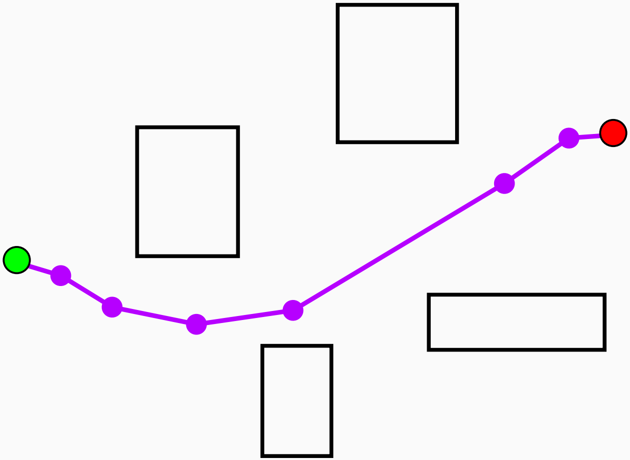
# Shortcutting



# Shortcutting



# Shortcutting



### Shortcutting

- Question: Can you do shortcutting with an arbitrary cost function ?

# Geometric Path Improvement

**Shortcutting**

- Question: Can you do shortcutting with an arbitrary cost function ?
- Question: Can you do shortcutting on e.g. a sphere ?

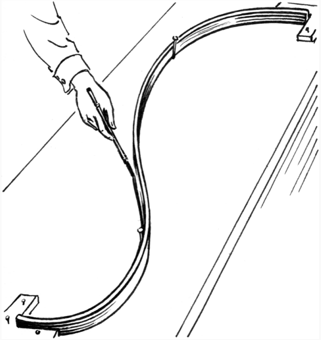
# Geometric Path Improvement

## Shortcutting

- Question: Can you do shortcutting with an arbitrary cost function ?
- Question: Can you do shortcutting on e.g. a sphere ?
- Question: Can you do shortcutting on kinodynamic systems ?

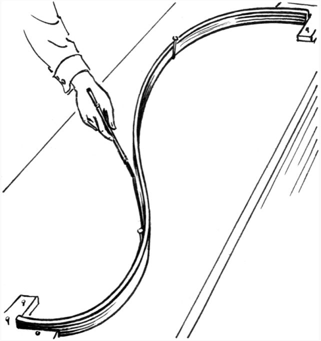
# Optimization using Splines

# Splines



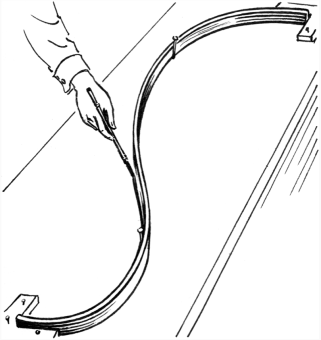
spline (n.)—long, thin piece of wood [Online Etymology Dictionary]

# Splines



Mathematically: A piecewise polynomial function.

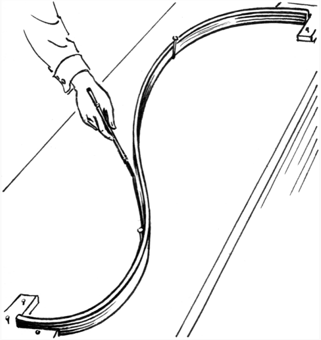
# Splines



Mathematically: A piecewise polynomial function.

Why do we want that?

# Splines



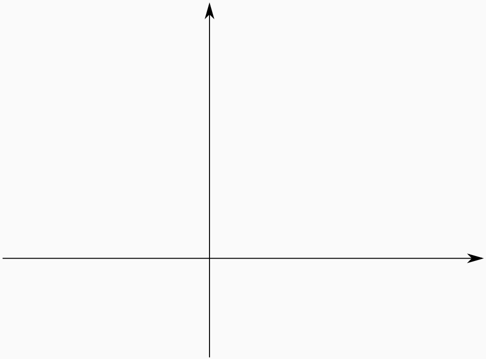
Mathematically: A piecewise polynomial function.

Why do we want that? Smoothness, Differentiability, Comfort (car)

# Splines

**Smoothness**

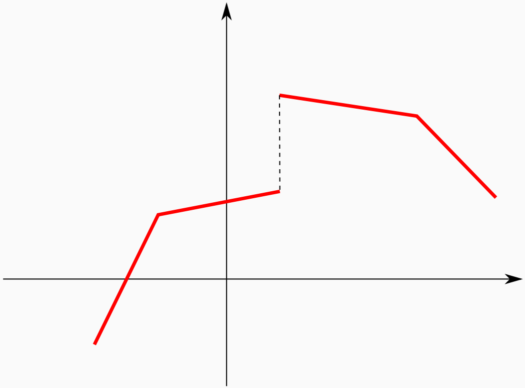
A path which is at least $n$-times differentiable (differentiability classes $C^0, C^1, \ldots, C^\infty$).



# Splines

**Smoothness**

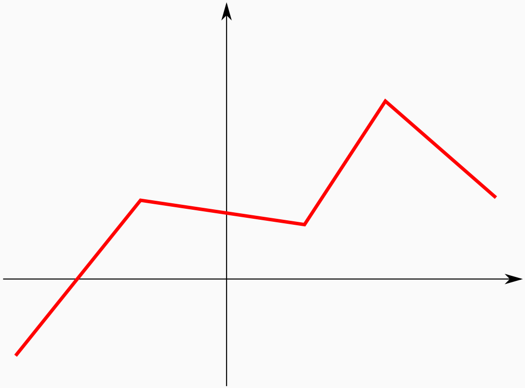
A path which is at least $n$-times differentiable (differentiability classes $C^0, C^1, ..., C^\infty$).



Discontinuous function

**Smoothness**

A path which is at least $n$-times differentiable (differentiability classes $C^0$, $C^1$, ..., $C^\infty$).

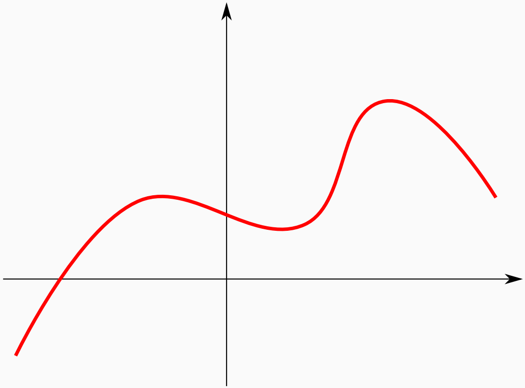


$C^0$ function

# Splines

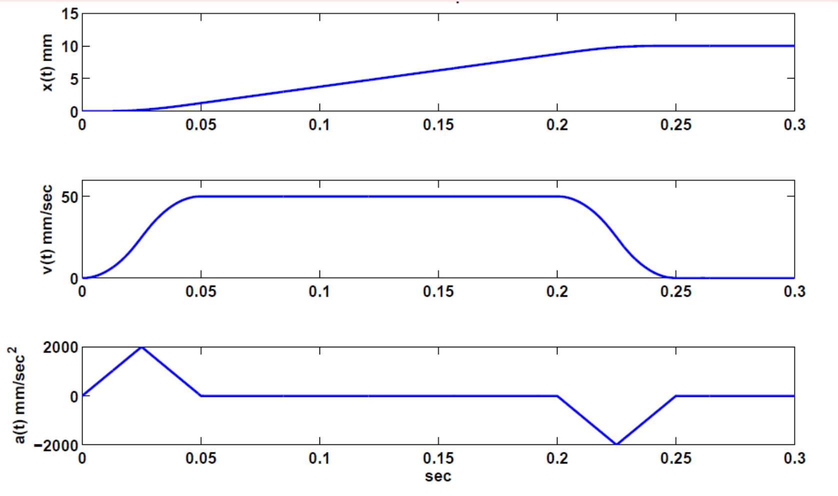
**Smoothness**

A path which is at least $n$-times differentiable (differentiability classes $C^0$, $C^1$, ..., $C^\infty$).



$C^1$ function

# Splines



x(t) mm
15
10
5
0
0
0.05
0.1
0.15
0.2
0.25
0.3
v(t) mm/sec
50
0
a(t) mm/sec^2
2000
0
−2000
sec

# Splines

- Basis splines (B-Splines)
- Polynomial splines
- Geometric planning with polynomial splines
- Bézier curves
- Safe planning with Bézier curves

# Optimization using Splines

## B-Splines

# B-Splines

Basis Splines (BSplines)

- A path is represented by $M$ control points $p_m$ and corresponding basis functions

$$p(t) = \sum_{m=1}^{M} B_m(t) p_m, \text{ s.t. } \sum_{m=1}^{M} B_m(t) = 1 \text{ for all } t$$

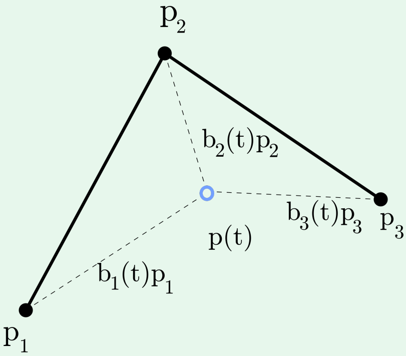
- Interpretation basis function: Influence of control point $p_m$ at point $t$.

# B-Splines

## Blending of basis functions

$$p(t) = \sum_{m=1}^{M} B_m(t) p_m$$

$$\text{s.t.} \sum_{m=1}^{M} B_m(t) = 1$$



# B-Splines

## Knot vector

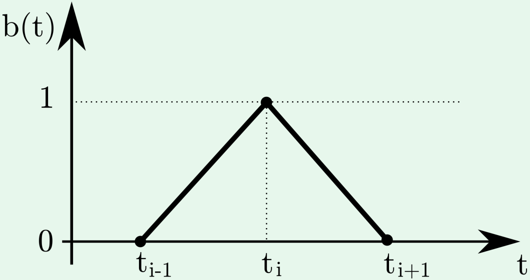
Let $p : [0,1] \to X$ be a path.

- Idea: Take the input space $[0,1]$ and cover it with $M$ subintervals $[t_i, t_{i+1}], i = 0, \dots, M$, and with $t_i < t_{i+1}$.
- Step 2: For each $t_i$ define a basis function centered at $t_i$.
- $t_i$ is called the $i$-th knot, and $t = (t_0, \dots, t_M)$ the knot vector ($M + 1$ elements).

# B-Splines

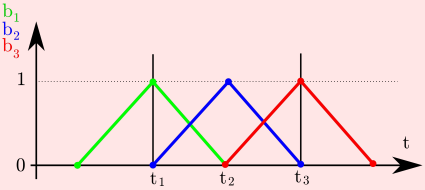
## Linear B-Splines

$$b_i(t) = \begin{cases} \dfrac{t - t_{i-1}}{t_i - t_{i-1}} & \text{, if } t_{i+1} < t \leq t_i \\ \dfrac{-t + t_{i+1}}{t_{i+1} - t_i} & \text{, if } t_i < t \leq t_{i+1} \\ 0 & \text{, otherwise.} \end{cases}$$



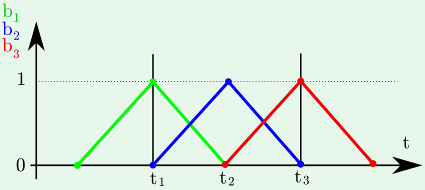
# B-Splines

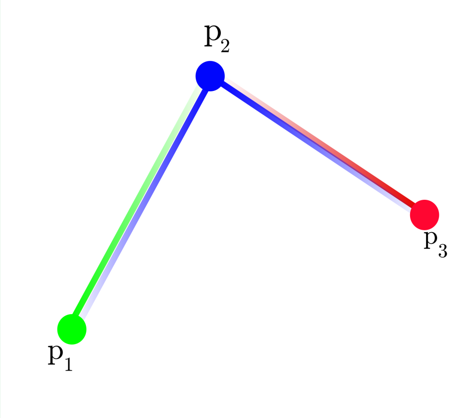
## Linear B-Splines



# B-Splines

## Linear B-Splines

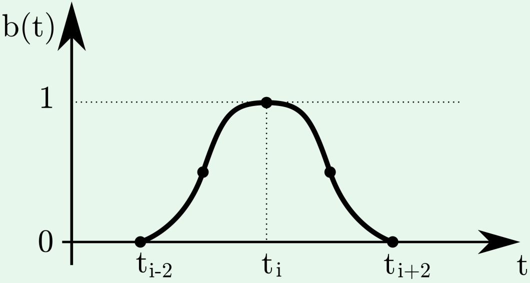


# B-Splines

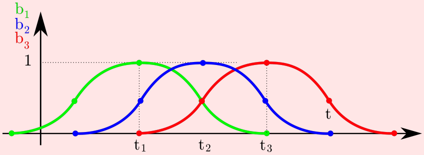
## Hermit B-Splines

$$b_i(t) = \begin{cases} 3x^2 - 2x^3 & \text{, if } t_{i-2} < t \leq t_i, \\ & \quad x = \dfrac{t - t_{i-2}}{t_i - t_{i-2}} \\ 3x^2 - 2x^3 & \text{, if } t_i < t \leq t_{i+2}, \\ & \quad x = 1 - \dfrac{t - t_i}{t_{i+2} - t_i} \\ 0 & \text{, otherwise.} \end{cases}$$

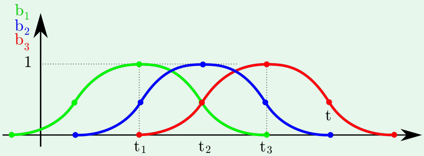


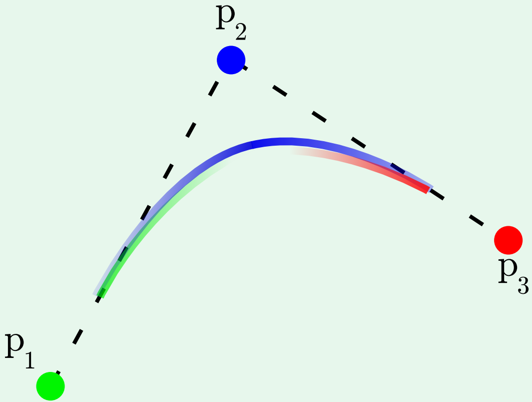
# B-Splines

## Hermit B-Splines



## Hermit B-Splines



# B-Splines

**Cox-de Bour Recurrence**

Recursive basis function (allows you to tune dimensionality and smoothness).

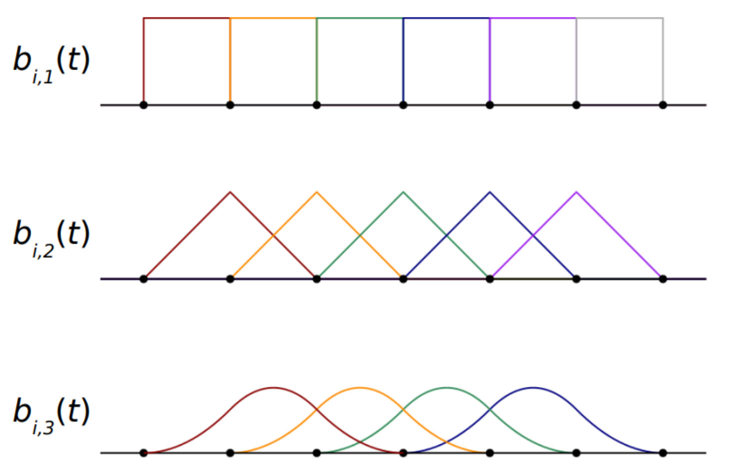
Choose an integer $k$. Then

$$b_{i,1}(t) = \begin{cases} 1 & \text{, if } t_i \leq t < t_{i+1} \\ 0 & \text{, otherwise.} \end{cases}$$

$$b_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} b_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} b_{i+1,k-1}(t)$$

# B-Splines

## Cox-de Bour Recurrence



# Optimization using Splines

## Polynomial Splines

# Polynomial Splines

## Polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n = \sum_{k=0}^{n} a_k t^k$$

- Cubic polynomial: $p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

# Polynomial Splines

## Polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n = \sum_{k=0}^{n} a_k t^k$$

- Cubic polynomial: $p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

## What are the values at $t = 0$ and $t = 1$?

# Polynomial Splines

## Polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n = \sum_{k=0}^{n} a_k t^k$$

- Cubic polynomial: $p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

## What are the values at $t = 0$ and $t = 1$?

$p(0) = a_0$

$p(1) = a_0 + a_1 + a_2 + a_3$

# Polynomial Splines

## Polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n = \sum_{k=0}^{n} a_k t^k$$

- Cubic polynomial: $p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

## What are the values at $t = 0$ and $t = 1$?

$p(0) = a_0$

$p(1) = a_0 + a_1 + a_2 + a_3$

## What are the derivatives with respect to t at $t = 0$ and $t = 1$?

# Polynomial Splines

## Polynomial

$$p(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_n t^n = \sum_{k=0}^{n} a_k t^k$$

- Cubic polynomial: $p(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3$

## What are the values at $t = 0$ and $t = 1$?

$p(0) = a_0$ $\quad$ $p(1) = a_0 + a_1 + a_2 + a_3$

## What are the derivatives with respect to t at $t = 0$ and $t = 1$?

| | | |
|---|---|---|
| $p'(t) = a_1 + 2a_2 t + 3a_3 t^2$ | $p'(0) = a_1$ | $p'(1) = a_1 + 2a_2 + 3a_3$ |
| $p''(t) = 2a_2 + 6a_3 t$ | $p''(0) = 2a_2$ | $p''(1) = 2a_2 + 6a_3$ |
| $p'''(t) = 6a_3$ | $p'''(0) = 6a_3$ | $p'''(1) = 6a_3$ |

# Polynomial Splines

Assume we represent the trajectory using polynomial splines.

**What is the acceleration cost?**

# Polynomial Splines

Assume we represent the trajectory using polynomial splines.

**What is the acceleration cost?**

$$
\begin{aligned}
J &= \int_{t=0}^{1} p''(t)dt \\
&= \int_{t=0}^{1} 2a_2 + 6a_3 \\
&= 2a_2 t + 3a_3 t^2 \Big|_{t=0}^{1} \\
&= 2a_2 + 3a_3
\end{aligned}
$$

# Polynomial Splines

Let's try to connect 3 numbers $x_1, x_2, x_3 \in \mathbb{R}$ using 2 cubic polynomials (with coefficients $a$ and $b$):

$$
\begin{aligned}
\underset{a_0,a_1,a_2,a_3,b_0,b_1,b_2,b_3}{\operatorname{argmin}} \quad J_a + J_b \quad \text{s.t.} \\
p_a(0) &= x_1 \\
p_a(1) &= x_2 \\
p_b(0) &= x_2 \\
p_b(1) &= x_3 \\
p_a'(1) &= p_b'(0) \\
p_a''(1) &= p_b''(0)
\end{aligned}
$$

# Polynomial Splines

Let's try to connect 3 numbers $x_1, x_2, x_3 \in \mathbb{R}$ using 2 cubic polynomials (with coefficients $a$ and $b$):

$$\begin{aligned}
\underset{a_0,a_1,a_2,a_3,b_0,b_1,b_2,b_3}{\operatorname{argmin}} \quad & J_a + J_b \quad \text{s.t.} \\
p_a(0) &= x_1 \\
p_a(1) &= x_2 \\
p_b(0) &= x_2 \\
p_b(1) &= x_3 \\
p_a'(1) &= p_b'(0) \\
p_a''(1) &= p_b''(0)
\end{aligned}$$

$$\begin{aligned}
\underset{a_0,a_1,a_2,a_3,b_0,b_1,b_2,b_3}{\operatorname{argmin}} \quad & 2a_2 + 3a_3 + 2b_2 + 3b_3 \quad \text{s.t.} \\
a_0 &= x_1 \\
a_0 + a_1 + a_2 + a_3 &= x_2 \\
b_0 &= x_2 \\
b_0 + b_1 + b_2 + b_3 &= x_3 \\
a_1 + 2a_2 + 3a_3 &= b_0 \\
2a_2 + 6a_3 &= 2b_2
\end{aligned}$$

# Polynomial Splines

**What kind of optimization problem is that?**

# Polynomial Splines

**What kind of optimization problem is that?**

So far LP. (For higher-orders it can become a QP).

# Polynomial Splines

**What kind of optimization problem is that?**

So far LP. (For higher-orders it can become a QP).

**How can we handle the 2D or 3D case?**

# Polynomial Splines

**What kind of optimization problem is that?**

So far LP. (For higher-orders it can become a QP).

**How can we handle the 2D or 3D case?**

Optimization can be done for each dimension independently.

# Polynomial Splines: Challenges

**Poor Numerical Stability**

When using high order ($\geq 8$) and/or many ($\geq 50$) pieces, it is difficult to solve in practice.

One solution: formulate as unconstrained QP where decision variables are endpoint derivatives of segments [1].

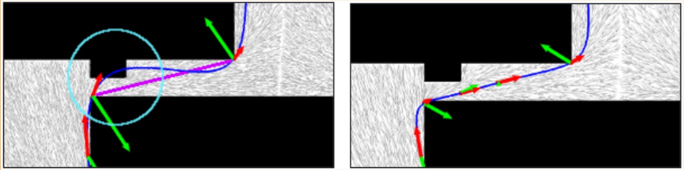
# Polynomial Splines: Challenges

## Poor Numerical Stability

When using high order ($\geq 8$) and/or many ($\geq 50$) pieces, it is difficult to solve in practice.

One solution: formulate as unconstrained QP where decision variables are endpoint derivatives of segments [1].

## Handling of Obstacles



- Add additional waypoints

# Optimization using Splines

## Bézier Curves

# Bézier Curves

## Bézier Curve

A Bézier curve $\mathbf{p} : [0, 1] \to \mathbb{R}^d$ of degree $n$ is defined by $n+1$ control points $\mathbf{p}_0, \ldots, \mathbf{p}_n \in \mathbb{R}^d$ as follows:

$$\mathbf{p}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{p}_i$$

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} .$$

# Bézier Curves

## Bézier Curve

A Bézier curve $\mathbf{p} : [0, 1] \to \mathbb{R}^d$ of degree $n$ is defined by $n+1$ control points $\mathbf{p}_0, \ldots, \mathbf{p}_n \in \mathbb{R}^d$ as follows:

$$\mathbf{p}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{p}_i$$

$$b_{i,n}(t) = \binom{n}{i} t^i \,(1-t)^{n-i} \, .$$

## Cubic Bézier Curve

$$\mathbf{p}(t) = (1-t)^3\mathbf{p}_0 + 3t(1-t)^2\mathbf{p}_1 + 3t^2(1-t)\mathbf{p}_2 + t^3\mathbf{p}_3$$

# Bézier Curves

## Bézier Curve

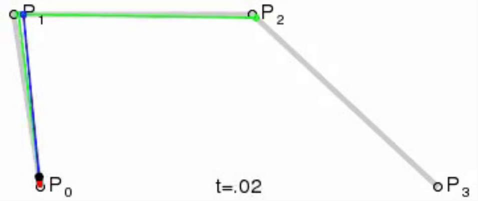
A Bézier curve $\mathbf{p} : [0,1] \to \mathbb{R}^d$ of degree $n$ is defined by $n+1$ control points $\mathbf{p}_0, \ldots, \mathbf{p}_n \in \mathbb{R}^d$ as follows:

$$\mathbf{p}(t) = \sum_{i=0}^{n} b_{i,n}(t)\mathbf{p}_i$$

$$b_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

## Cubic Bézier Curve

$$\mathbf{p}(t) = (1-t)^3\mathbf{p}_0 + 3t(1-t)^2\mathbf{p}_1 + 3t^2(1-t)\mathbf{p}_2 + t^3\mathbf{p}_3$$



# Bézier Curves Properties (1)

- Endpoint interpolation: The curve connects $\mathbf{p}_0$ and $\mathbf{p}_n$, i.e., $\mathbf{p}(0) = \mathbf{p}_0$ and $\mathbf{p}(1) = \mathbf{p}_n$

# Bézier Curves Properties (1)

- Endpoint interpolation: The curve connects $\mathbf{p}_0$ and $\mathbf{p}_n$, i.e., $\mathbf{p}(0) = \mathbf{p}_0$ and $\mathbf{p}(1) = \mathbf{p}_n$
- $C^n$ smoothness

**Derivative of Bézier Curve**

$$\mathbf{p}'(t) = n \sum_{i=0}^{n-1} b_{i,n-1}(t)(\mathbf{p}_{i+1} - \mathbf{p}_i).$$

## Bézier Curves Properties (1)

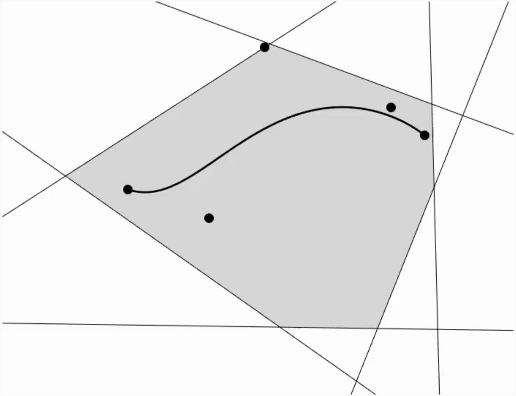
- Endpoint interpolation: The curve connects $\mathbf{p}_0$ and $\mathbf{p}_n$, i.e., $\mathbf{p}(0) = \mathbf{p}_0$ and $\mathbf{p}(1) = \mathbf{p}_n$
- $C^n$ smoothness

**Derivative of Bézier Curve**

$$\mathbf{p}'(t) = n \sum_{i=0}^{n-1} b_{i,n-1}(t)(\mathbf{p}_{i+1} - \mathbf{p}_i).$$

- Convex hull property: The curve lies inside the convex hull of their control points, i.e., $\mathbf{p}(t) \in ConvexHull\{\mathbf{p}_0, \ldots, \mathbf{p}_n\} \ \forall t \in [0, 1]$ [2]

# Convex Hull Property



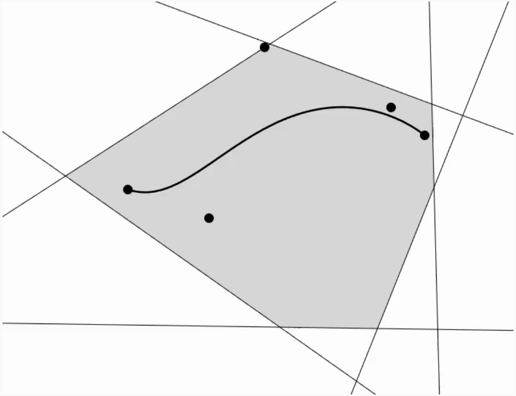
Separating hyperplane

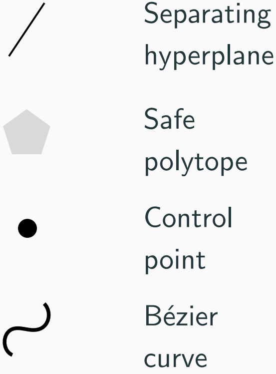
Safe polytope

Control point

Bézier curve

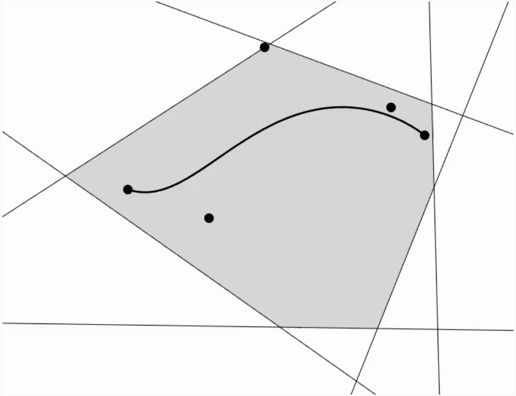
# Convex Hull Property

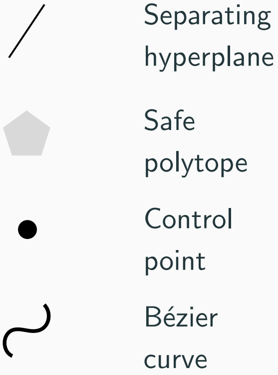





**Why is the convex hull property useful?**

# Convex Hull Property



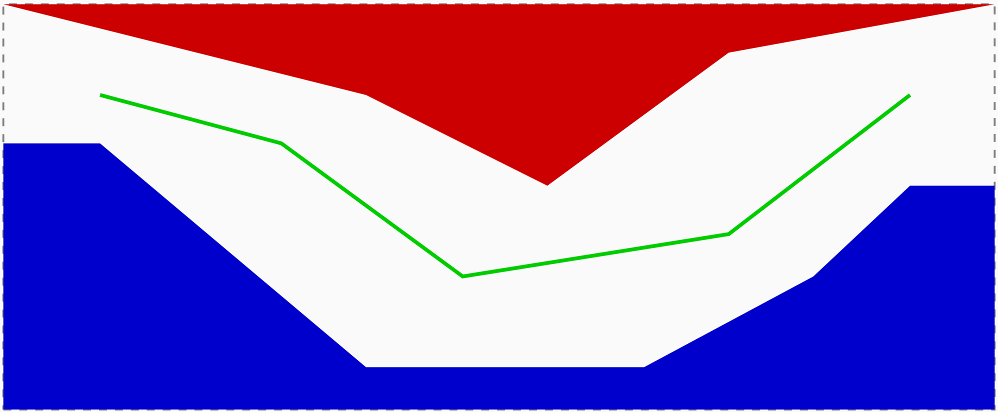



**Why is the convex hull property useful?**

If $\mathbf{p}_i$ are decision variables, we can constrain them to be in $\mathcal{Q}_{free}$. Then, the curve is guaranteed to be collision-free.

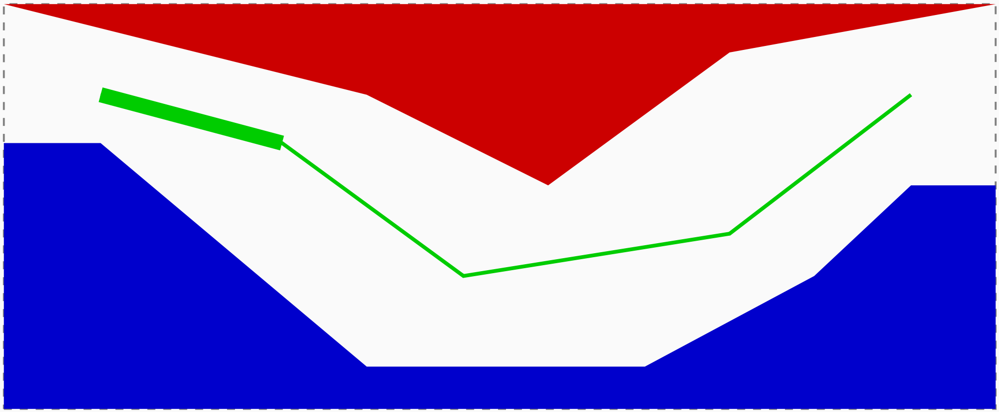
# Planning in Safe Polyhedra

robot path (green), obstacles (blue and red)



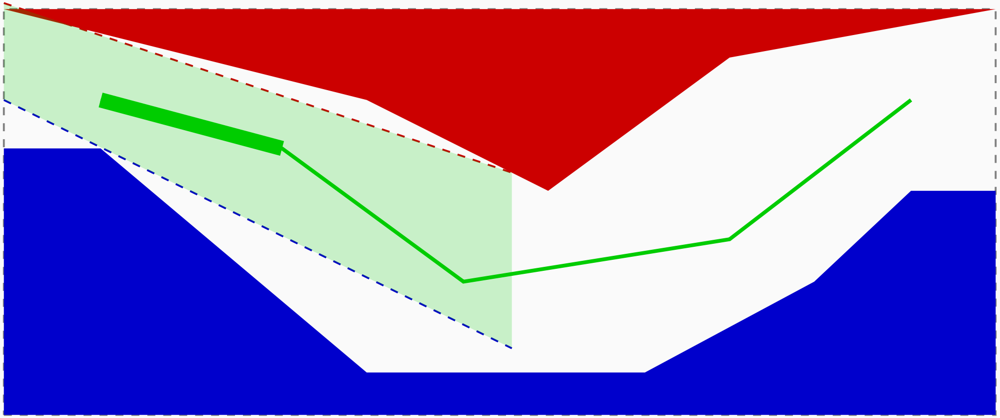
# Planning in Safe Polyhedra

time=1



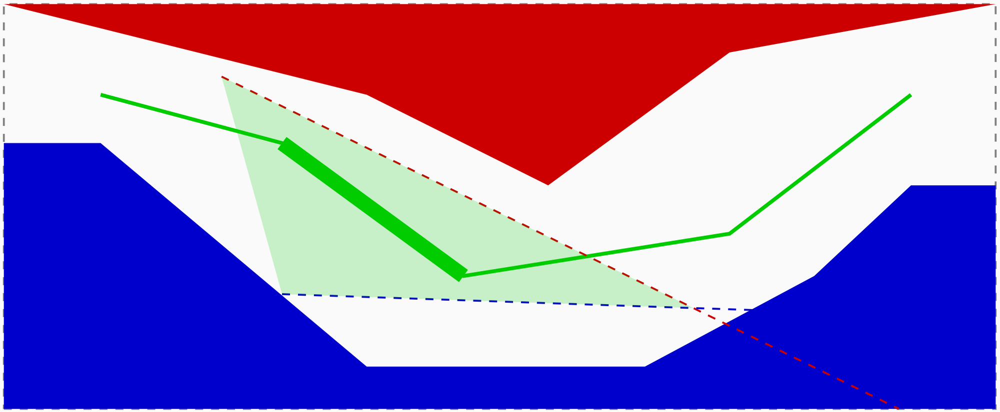
# Planning in Safe Polyhedra

time=1



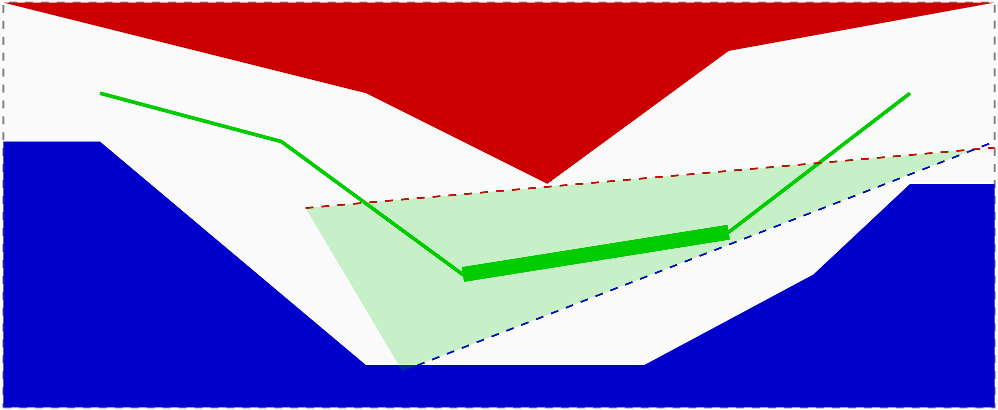
# Planning in Safe Polyhedra

time=2



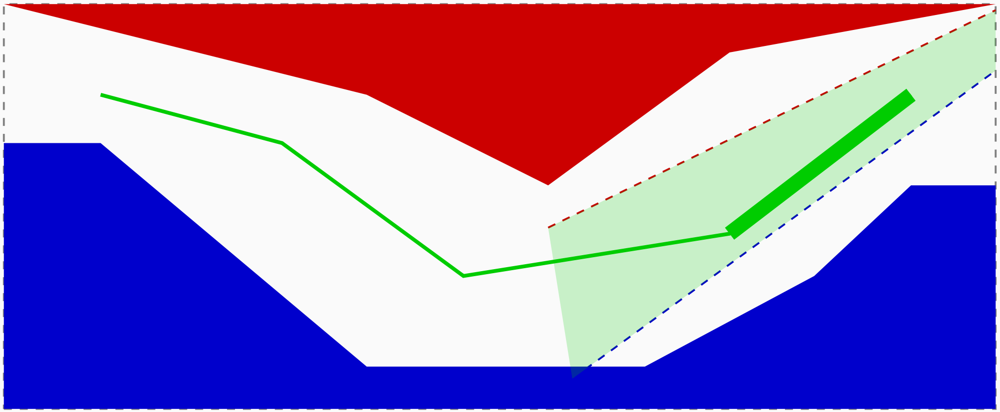
# Planning in Safe Polyhedra

time=3



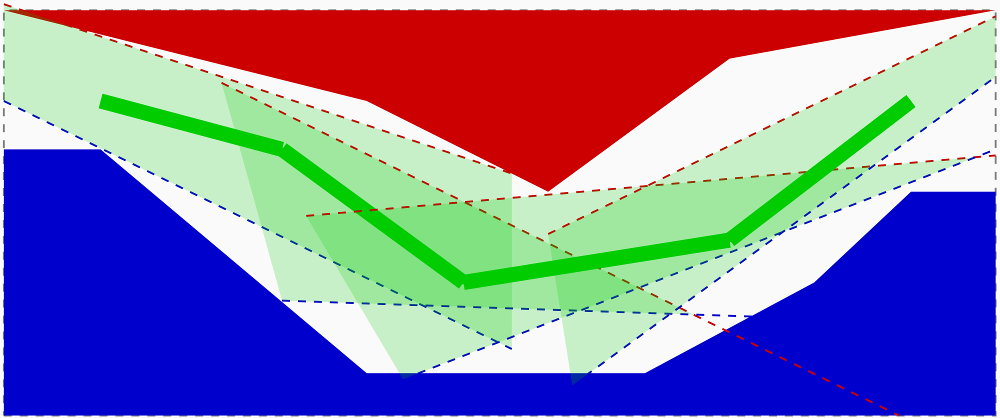
# Planning in Safe Polyhedra

time=4

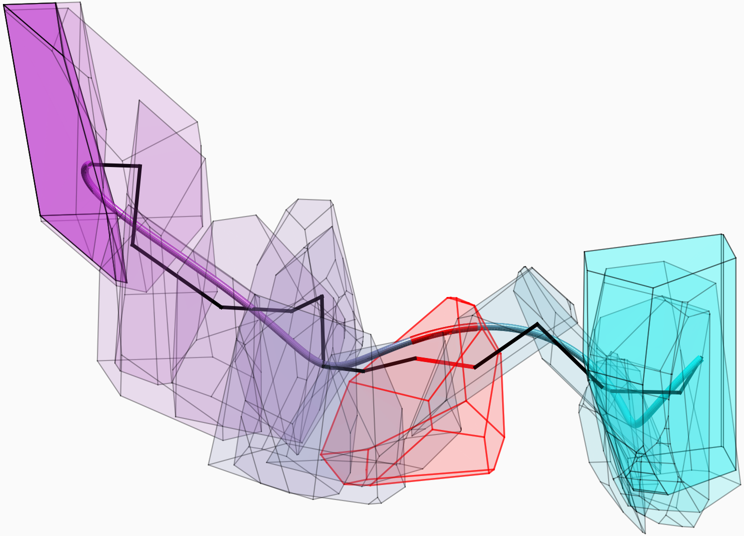


# Planning in Safe Polyhedra

robot path (green), obstacles (blue and red)



# Planning in Safe Polyhedra (3D Example)



# Summary Splines

- Optimization for smoothness and energy
- Base splines
- Polynomial splines and convex optimization
- Bézier curves for safe planning

# Gradient-based optimization on differentiable costs

# Gradients

- Idea: Improve path by deforming it into the direction of a lower cost
- Functional gradients as generalization of directions for paths

## Gradients

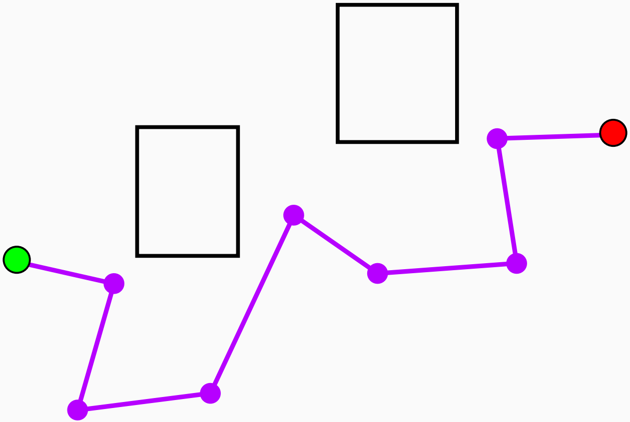
### Cost functionals

Given a path $p : [0,1] \to X$, define cost *functionals* like

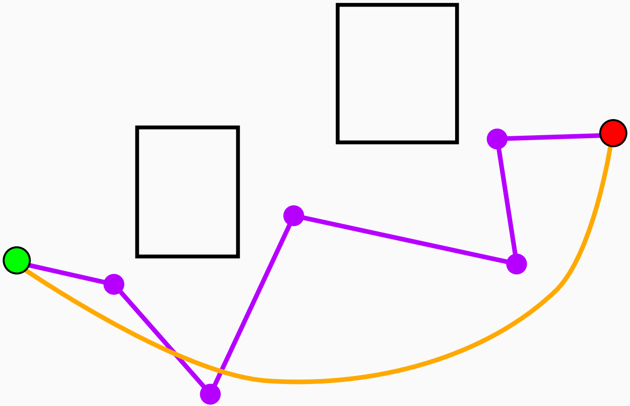
- Obstacle cost $U_{obs}[p]$
- Smoothness cost $U_{smooth}[p]$
- Path length cost $U_{length}[p]$
- Then compute gradients $\nabla U[p]$ and do gradient descent

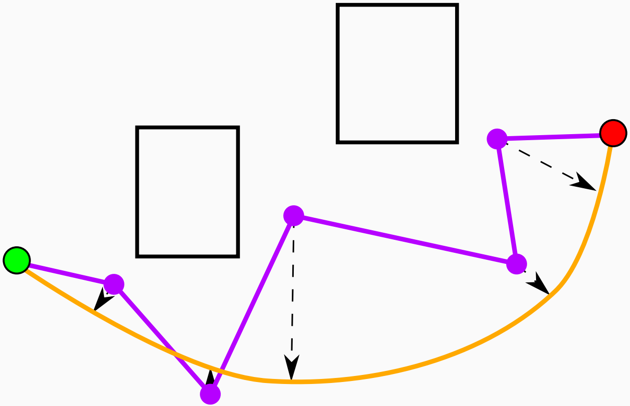
$$p' = p - \lambda \nabla U[p]$$

# Gradients

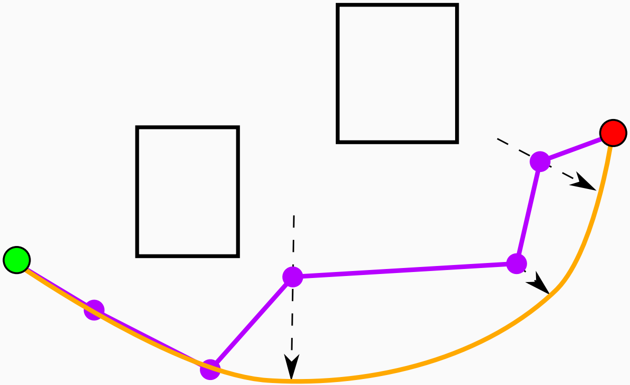


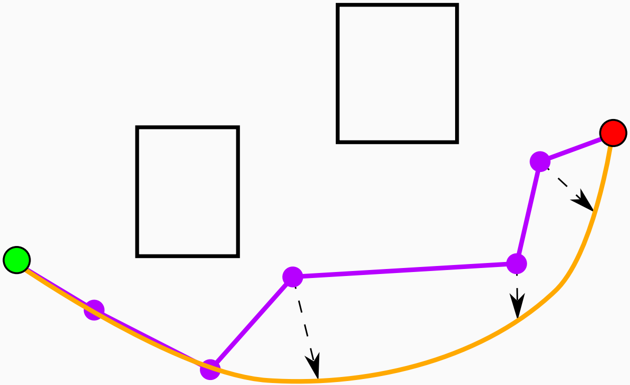
# Gradients



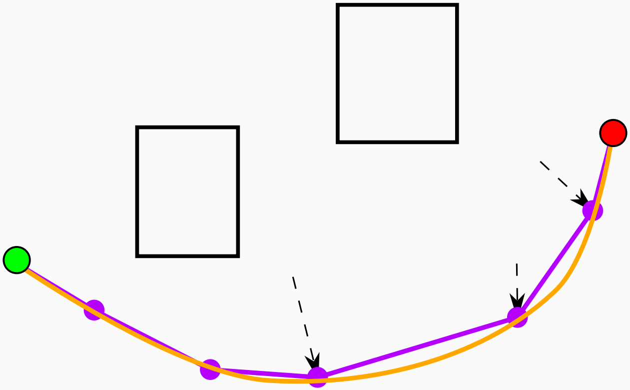


## Gradients

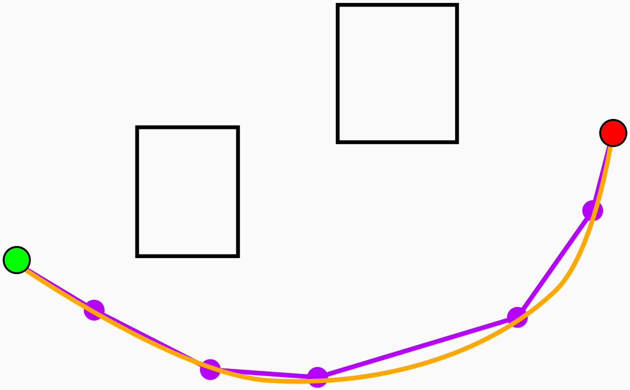




# Gradients



# Gradients



# Gradient-based optimization on differentiable costs

## Computing Obstacle Cost Gradients

# Gradients

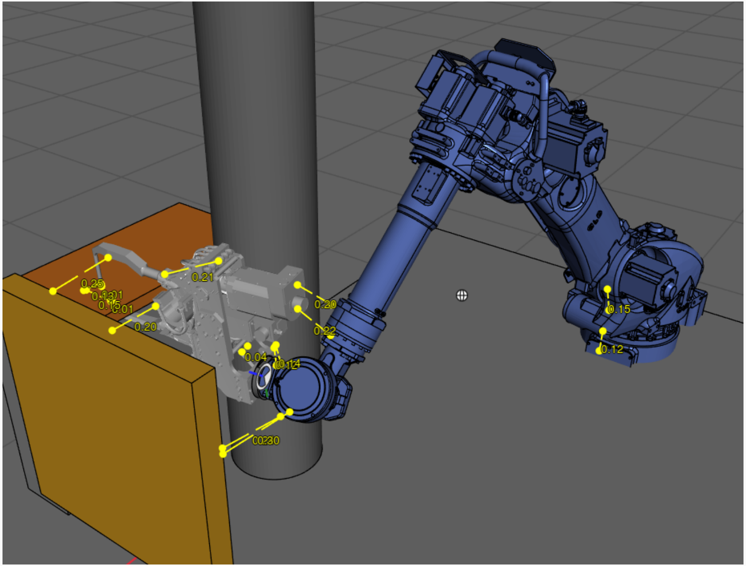
## Obstacle cost

- Definition of obstacle cost as clearance from environment
- Obstacle cost

$$U_{obs}[p] = \int_0^1 \int_{x \in B} c_x(p(s)) \, \mathrm{d}x \, \mathrm{d}s$$

- with $x$ being a prespecified point on the robot body $B$, $c_x$ being the minimum clearance to the environment and $s$ being the index of the path

# Robot point distances



# Clearance definition

## Obstacle cost

$$U_{obs}[p] = \int_0^1 \int_{x \in B} c_x(p(s))\,\mathrm{d}x\,\mathrm{d}s$$

## Robot clearance

$$c_x(q) = \begin{cases} \frac{1}{\epsilon}(d_x(q) - \epsilon)^2, & \text{if } d_x(q) < \epsilon \\ 0, & \text{otherwise.} \end{cases}$$

whereby $d_x(q)$ is the distance of point $x$ on robot at configuration $q$ to the nearest point in the environment.

# Obstacle gradients

**Requirements**

1. Efficient computation of distances (Signed distance fields)
2. Efficient obstacle cost gradient

# Gradient-based optimization on differentiable costs

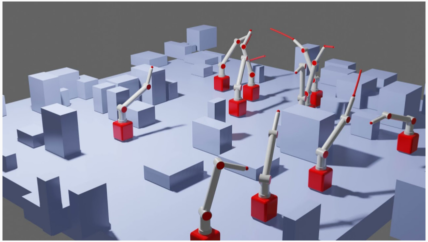
## Signed Distance Fields

# Obstacle gradients

**Problem**

Computing distance of a point to the environment is not trivial

- For each point, you need to compute the clearance
- In the worst case, you would need to run GJK once for all obstacles and all points

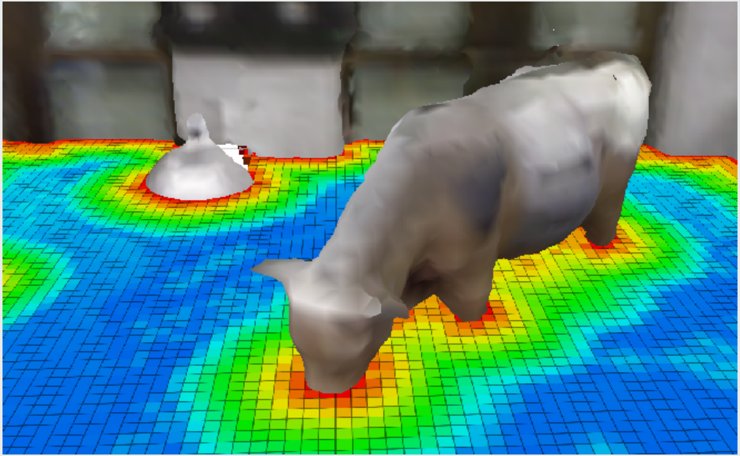


# Signed Distance Field

Assumption: All obstacles are static

Then we can precompute a signed distance field (SDF), which assigns for each point in space its clearance value

# Signed Distance Field



Can be computed efficiently in 2-d or 3-d.

# Signed Distance Field

**Signed distance field**

- Voxelize environment
- For each voxel, compute nearest obstacle distance
- Store this information, and use it as a look-up-table to compute clearances

### Marching parabola algorithm

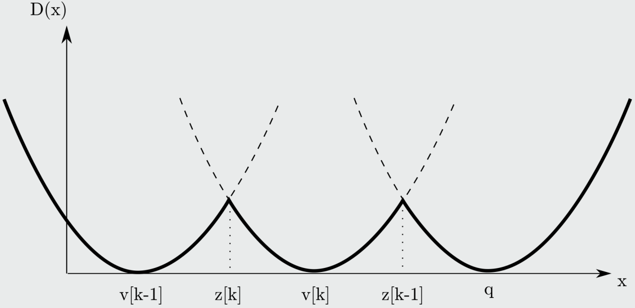
Marching parabola algorithm for euclidean distance fields [3] by Felzenszwalb and Huttenlocher, 2012

- Let $V$ be a 1-d grid, and $P$ be occupied voxels on the grid
- Then we search for $D_P(v) = \min\limits_{p \in P} d(v, p)$
- Idea: Computer lower envelope parabolas for all occupied voxels
- For each point, check height of lower envelope, and store this value.

# Marching parabola algorithm

## Step 1: Computer lower envelope parabolas

- k: number of parabolas in lower envelope
- p[i]: location of i-th parabola in lower envelope
- z[i], z[i+1]: range of i-th parabola in lower envelope lower envelope



# Marching parabola algorithm

## Step 1: Computer lower envelope parabolas

- $p[0] = 0$, $z[0] = -\infty$, $z[1] = +\infty$, $k = 0$
- for $p$ in $P$
  1. $s = \texttt{IntersectionPoint}(p, p[k])$
  2. If $s \leq z[k]$
     - $k = k - 1$
     - Goto 1
  3. Else
     - $k = k + 1$
     - $p[k] = p$
     - $z[k] = s$
     - $z[k+1] = +\infty$

# Marching parabola algorithm

**Step 2: Assign each point its height on the lower envelope**

- $k = 0$
- for $v$ in $G$
    - while $z[k+1] < v$
        - k+1
    - $D_P(v) = (v - p[k])^2$
- Return $D_P$

# Marching parabola algorithm

- Multidimensional case can be reduced to 1-dimensional case
- Let $(x, y) \in G$ be an element of a 2-d grid

$$
\begin{aligned}
D_P(x, y) &= \min_{x', y' \in P} [(x - x')^2 + (y - y')^2] \\
&= \min_{x'} [(x - x')^2 + \min_{y'} (y - y')^2] \\
&= \min_{x'} [(x - x')^2 + D_P(y)]
\end{aligned}
$$

- Computational complexity $O(dN)$, with $d$ being the dimension and $N$ being the number of points

Publication: "Distance transforms of sampled functions", PF Felzenszwalb, DP Huttenlocher, Theory of computing, 2012

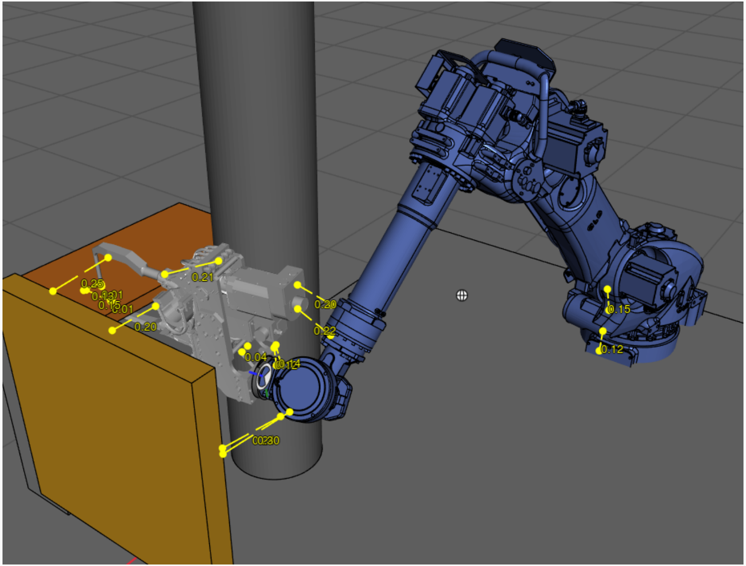
# Gradient-based optimization on differentiable costs

## Obtacle cost gradient

# Gradient of obstacle cost

- Given $U_{obs}[p]$, let us compute $\nabla U_{obs}[p]$.
- Three steps
  - Compute gradient of clearance
  - Consider gradient as virtual forces
  - Map forces into configuration space

# Robot point distances



# Gradient of obstacle cost

**Step 1: Compute gradient of clearance**

- Clearance $c_x(q)$ will be improved by moving point $x$ away from obstacle
- Let $n_E$ be the nearest point on the environment to $x$.
- Then

$$\frac{x - n_E}{\|x - n_E\|}$$

# Gradient of obstacle cost

**Step 2: Consider gradient as virtual forces**

- Pushing at each $x$ along the direction of $\dfrac{x - n_E}{\|x - n_E\|}$ would increase clearance!
- Let us define virtual forces at each $x$ and map them into the configuration space

# Gradient of obstacle cost

**Step 3: Map forces into configuration space**

- A virtual force will move each point $x$ by an infinitesimal small $dx$.
- This will create likewise an infinitesimal small $dq$ in configuration space
- A $dq$ is the direction in configuration space, which (locally) increases clearance.

# Gradient of obstacle cost

## Step 3: Map forces into configuration space

- How to do it? By taking the derivative of the forward kinematics.
- Let $x$ be a point in the workspace, $q$ be the joint space values, and $T$ the forward kinematics.

$$\begin{aligned} x &= T(q) \\ \dot{x} &= \frac{dT(q)}{dq}\dot{q} \\ &= J \cdot \dot{q} \end{aligned}$$

- Taking the inverse leads to $\dot{q} = J^{-1}(q)\dot{x}$ (transpose, pseudoinverse)

# Gradient of obstacle cost

## Steepest Descent Step

- Functional gradient

$$\nabla U[p] = \int_0^1 \sum_{x \in R(p(s))} J^{-1}(p(s)) \nabla c_x p(s) \, \mathrm{d}s$$

- Gradient descent step in functional space:

$$p' = p - \lambda \nabla U[p]$$

# Gradient of obstacle cost

**Functional Gradient Descent**

```
Let p be a path
While not converged
    p' = p - lambda * dU[p]
Return p'
```

# Summary of obstacle cost gradient

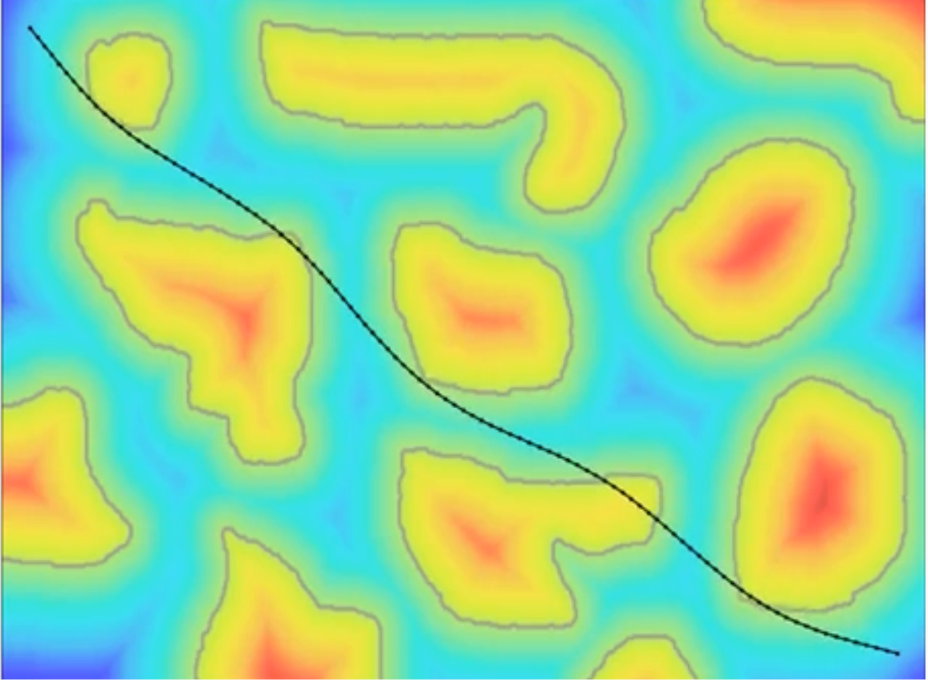
- (1) Formulate cost from distances
- (2) Compute virtual displacements in direction of decreasing cost
- (3) Map virtual displacements into configuration space using jacobian
- (4) Take those displacements as directions for gradient step
- (5) Apply gradient descent using the directional steps in configuration space

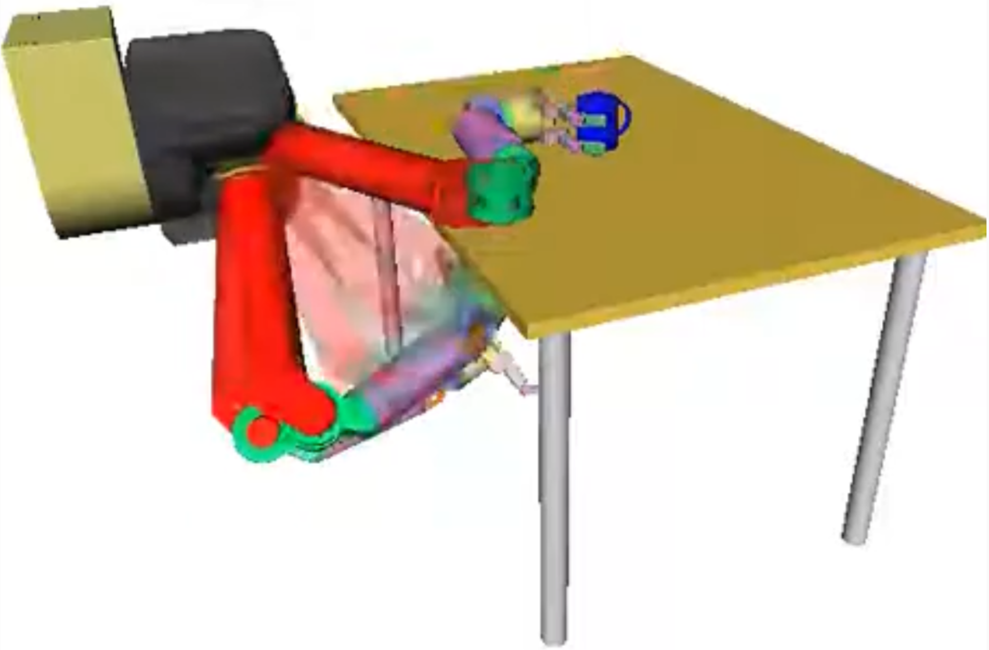
# CHOMP

Covariant Hamiltonian Optimization for Motion Planning (CHOMP)

- Initialize a path
- Discretize the path (waypoints)
- Use cost functional $U[p] = U_{obs}[p] + \lambda U_{smooth}[p]$
- Apply derivative to waypoints
- Repeat or terminate if magnitude of gradient falls below threshold

M Zucker et al., "Chomp: Covariant hamiltonian optimization for motion planning", 2013 [4]





Completeness and Optimality?

# Conclusion

## Summary

- Optimization, gradient-free vs. gradient-based
- Shortcutting
- B-splines
- Polynomial splines (acceleration optimized)
- Bézier curves (obstacle-free)
- Computing obstacle costs (Signed distance field, obstacle gradients)
- Gradient descent and CHOMP

[1] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments". In: *International Symposium on Robotics Research (ISRR)*. Vol. 114. Springer Tracts in Advanced Robotics. Springer, 2013, pp. 649–666. DOI: 10.1007/978-3-319-28872-7_37.

[2] Rida T. Farouki. "The Bernstein polynomial basis: A centennial retrospective". In: *Computer Aided Geometric Design* 29.6 (2012), pp. 379–419. DOI: 10.1016/j.cagd.2012.03.001.

[3] Pedro F Felzenszwalb and Daniel P Huttenlocher. "Distance transforms of sampled functions". In: *Theory of computing* 8.1 (2012), pp. 415–428.

[4] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. "Chomp: Covariant hamiltonian optimization for motion planning". In: *The International Journal of Robotics Research* 32.9-10 (2013), pp. 1164–1193.