Motion Planning Under Differential Constraints

Clinton Freeman

Presentation Preview

- What is motion planning?
- What are differential constraints?
- Basic definitions
- Problem specification
- Solution strategies
- Specific algorithm: **KPIECE**
- Recommended reading, motion planning software

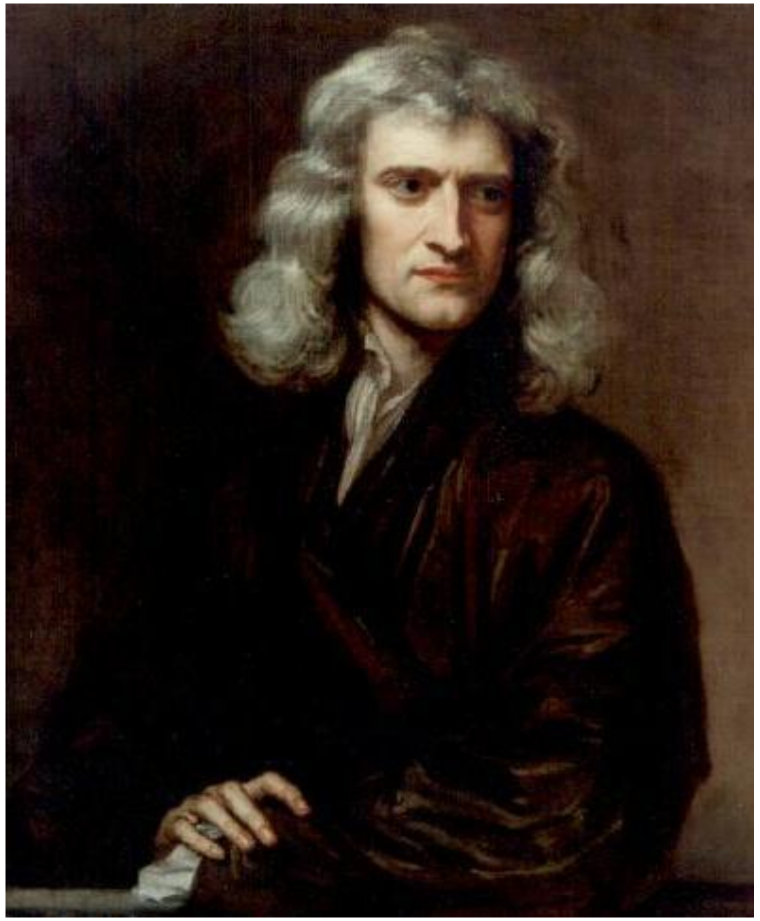
What is Motion Planning?

- Find a set of motions that will navigate a robot from an initial state to a goal region
- Applications:
  - Character animation
  - Driverless cars
  - Robotic surgery
  - Molecular design
  - AI Planning

GOAL
START
(u_1, t_1)
(u_2, t_2)
(u_3, t_3)
(u_4, t_4)

What are Differential Constraints?

- Equations that describe allowable robot motions
- Represent physical models
    - Paddling a boat on a river
    - Various car models
        - Reeds-Shepp: reverse, park, forward
        - Dubin’s: park, forward
    - Differential drive robots (two wheels with motors, one passive wheel)
    - Newtonian mechanics (particle physics)

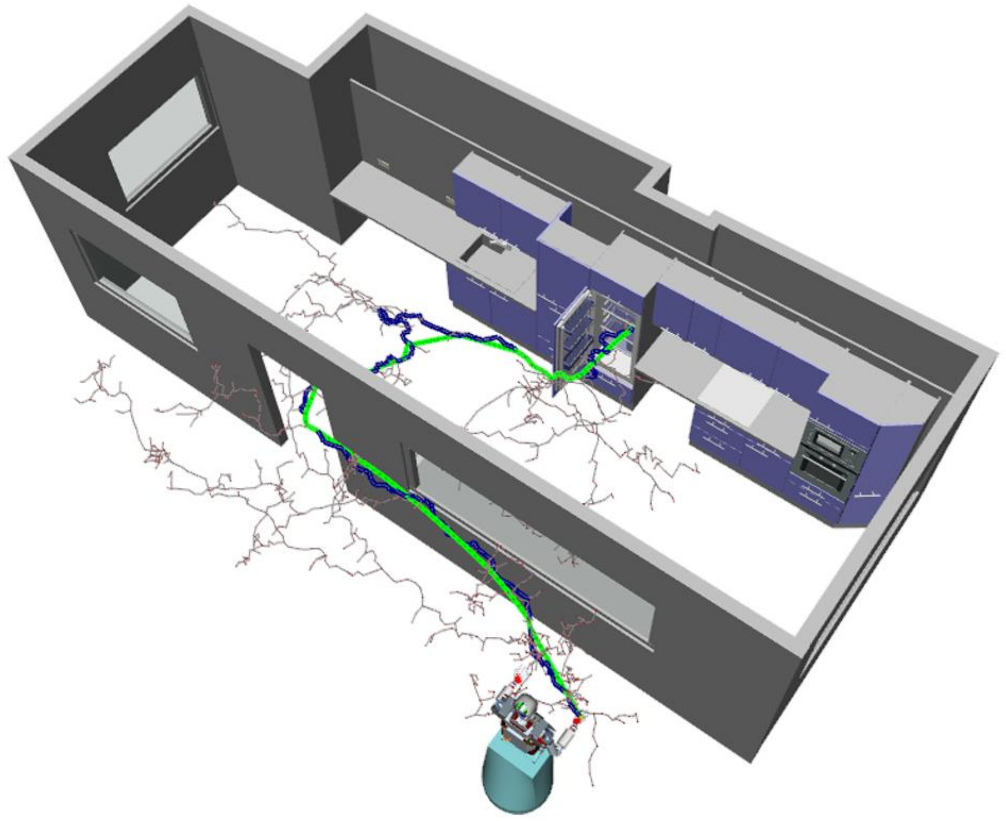


Problem/Solution Categories

- Motion Planning without Differential Constraints
  - Combinatorial
    - Complete, i.e. guaranteed to find solution if it exists
    - Exact, can solve most problems, but hard to implement and slow
  - Sampling-based
    - Probabilistically complete, i.e. guaranteed to find solution as sample size $\rightarrow \infty$
    - Weaker guarantees, but easy to implement and fast
- Motion Planning with Differential Constraints
  - Decoupled
    - Plan without physics first, then use path as basis for solution
  - Sampling-based
    - Account for physics up front

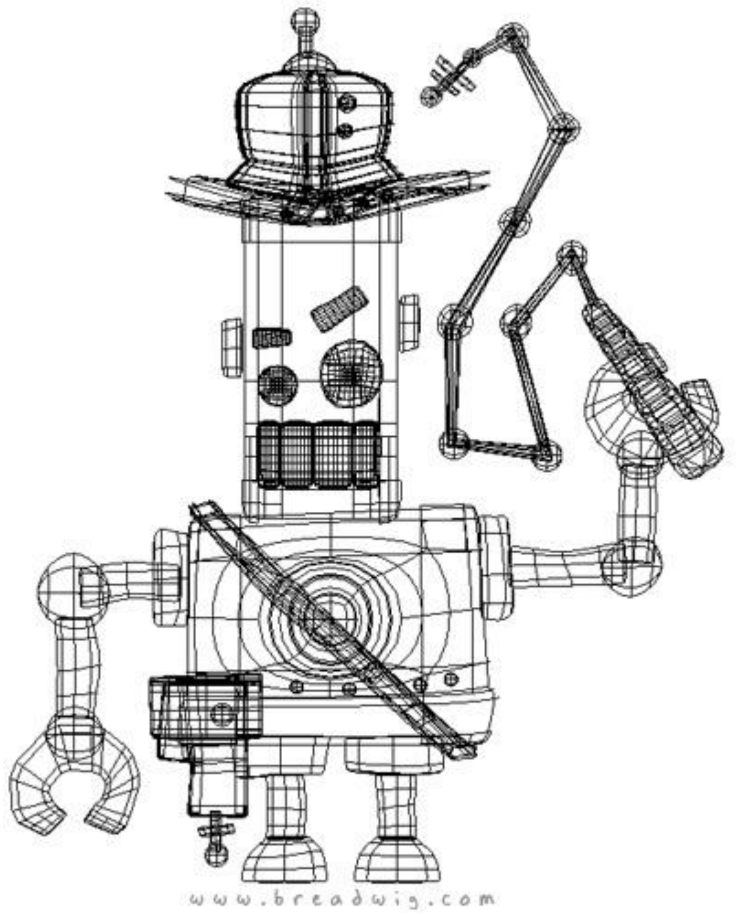
Problem Geometry

- *World* $\mathcal{W}$
    - Ambient space, typically $\mathbb{R}^2$ or $\mathbb{R}^3$
- *Robot* $\mathcal{A}$
    - Rigid body, or a chain of bodies
    - Mobile
- *Obstacle region* $\mathcal{O}$
    - "Occupied" subset of $\mathcal{W}$
    - Fixed



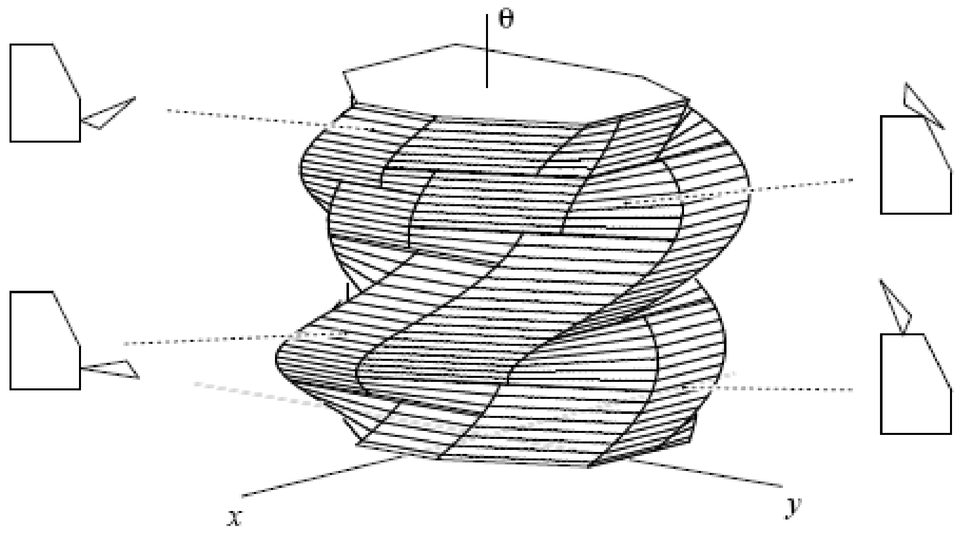
Representation of $\mathcal{A}$ and $\mathcal{O}$

- Combinatorial methods take geometry into account
- Sampling methods use geometry during collision detection
  - Most treat collision detection as a black box
  - We can abstract over geometry



Problem Kinematics, Configuration Space

- Robot $\mathcal{A}$ has certain allowable transformations, such as:
  - Translations = $\mathbb{R}^n$
  - Rotations = $SO(n)$ = special orthogonal group
- *Configuration space* $\mathcal{C}$ is set of all possible transformations
  - E.g. $\mathcal{C} = \mathbb{R}^n \times SO(n) = SE(n)$ = special Euclidean group
- http://youtu.be/SBFwgR4K1Gk

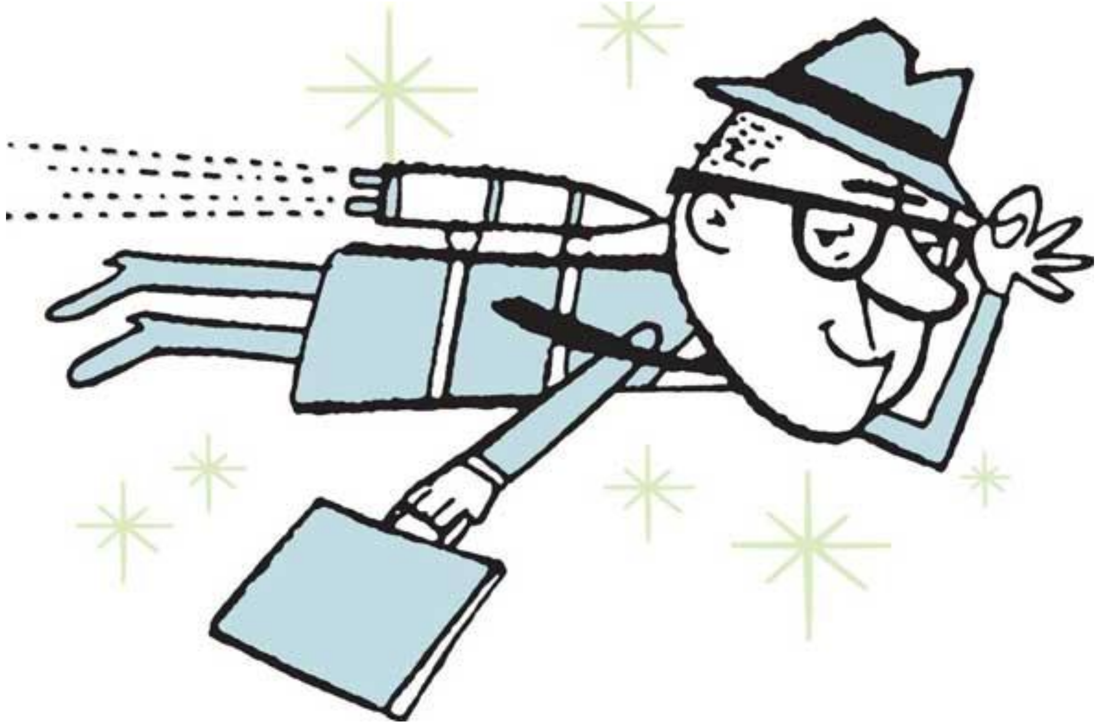


Differential Constraints, Phase/State Space

- In a first-order world…
    - *Differential constraints* restrict the allowable velocities of the robot at each point in $\mathcal{C}$
- Need higher-order for dynamics (e.g. Newtonian mechanics)…
    - *Differential constraints* also restrict higher-order derivatives
    - *Phase space* pairs configurations with derivatives, sometimes called the *tangent bundle*
    - A *state space* $\mathcal{X}$ can be either $\mathcal{C}$ or the associated phase space

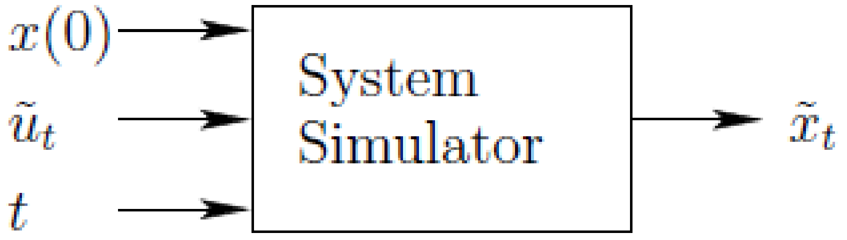
Actions and Action Space

- An *action* (or *control*) u is a variable which can be applied to exert a force on the robot $\mathcal{A}$
- An *action trajectory* is a function mapping time to actions
- *Action space* $U(x)$ is the set of all possible actions in a given state
    - Not necessarily homogeneous
    - Includes termination action u_T
    - Example: Jetpack with fuel



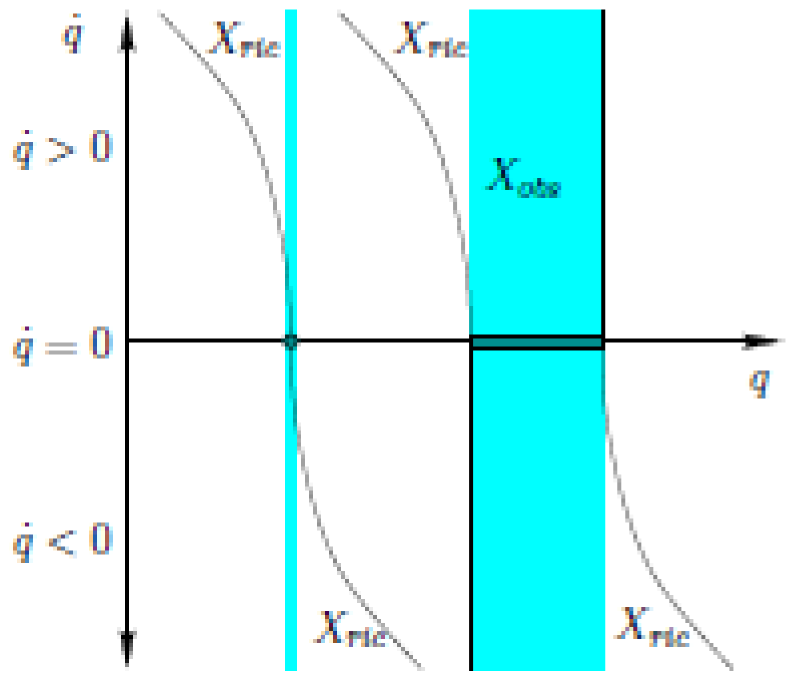
State Transition Equation

- *State transition equation* $\dot{x} = f(x, u), \forall x \in \mathcal{X}, \forall u \in U(x)$
- f contains the differential constraint equations
- Integration of f for a particular state and action pairing yields a new state
- *State trajectory* is a mapping of time to a series of states



Consequence: Region of Inevitable Collision

- States in which no action trajectory can avoid collision with $\mathcal{C}_{obs}$
- Generally not computed, unknown how to detect collisions
- Problematic for high momentum systems
- Example: plane flying into a cave



Problem Formulation

- Input
    - *Time interval* $T = [0, \infty)$
    - *Initial state* $x_1 \in \mathcal{X}_{free}$
    - *Goal region* $\mathcal{X}_G \subset \mathcal{X}_{free}$
- Output
    - *Action trajectory* $\bar{u} : T \rightarrow U$
    - *State trajectory* $\bar{x}$ satisfies $\bar{x}(0) = x_1$, $\exists\ t > 0$ for which $\bar{u}(t) = u_T$ and $\bar{x}(t) \in \mathcal{X}_G$

Solution Strategies

- Exact solutions
  - Planning under just geometric constraints is already P-SPACE hard [3]
  - Generally unknown whether planning with differential constraints is decidable [4]
  - Polynomial time solution available for point-mass moving in 3 space
- Plan and Transform
  - Feasible
  - Brief description
- State Sampling
  - Many methods use this framework
  - Example: RRTs
  - Algorithm: KPIECE

Solution Strategy: Plan and Transform

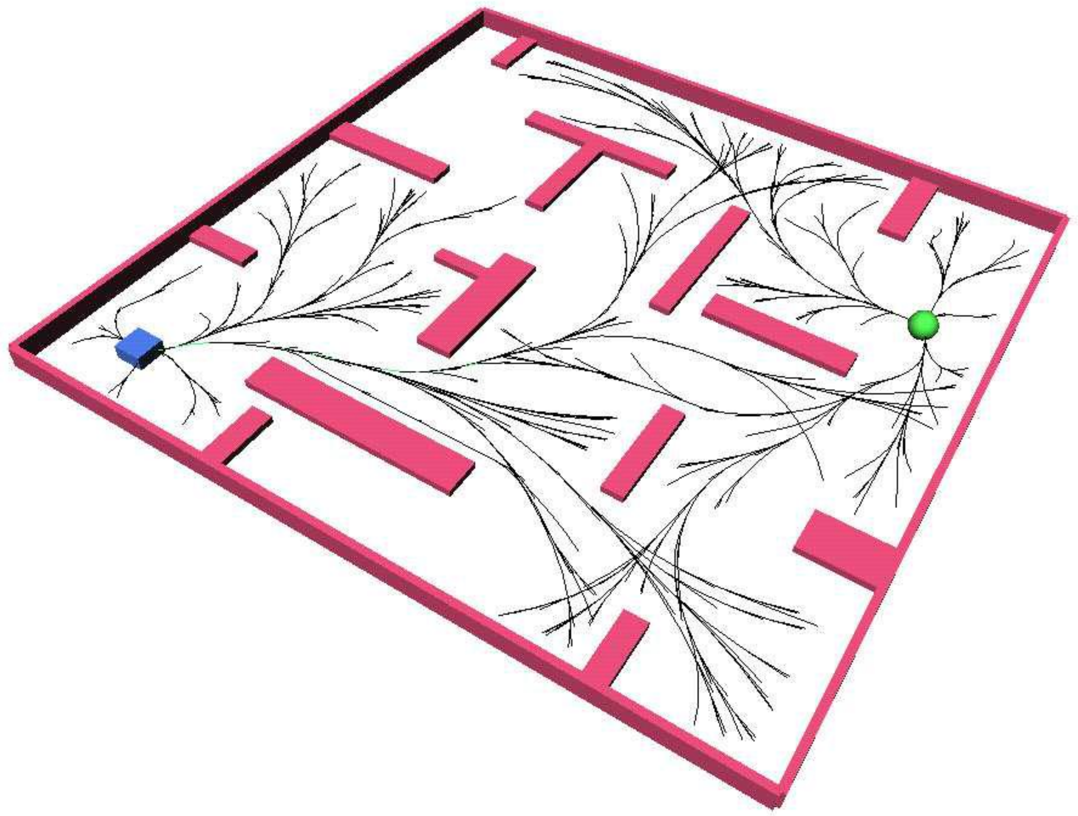
1. **Compute:** a path $\tau : [0, 1] \rightarrow \mathcal{C}_{free}$ ignoring constraints
2. **Transform:** τ into a new path τ' to satisfy constraints
    - Example: subdivide τ and use a local planning method to replace the subdivision with a path that accounts for constraints
3. **Check:** If τ satisfies constraints over the whole interval then terminate, otherwise go to 2

Solution Strategy: State Sampling

1. **Initialize**: let $\mathcal{G}(V, E)$ represent an undirected search graph, where $V \subset \mathcal{X}_{free}$ and E is a set of edges that store action trajectories
2. **Select**: choose a vertex $x_{cur} \in \mathcal{G}$ for expansion
3. **Plan**: generate a *motion primitive* $\bar{u}^p : [0, t_F] \rightarrow \mathcal{X}_{free}$ such that $u(0) = x_{cur}$ and $u(t_F) = x_r$ for some $x_r \in \mathcal{X}_{free}$.
4. **Insert**: insert $\bar{u}^p$ into E, and x_r into V
5. **Check**: return if $\mathcal{G}$ encodes a solution path
6. **Repeat**

Example: Rapidly Exploring Random Tree

- Selecting a vertex:
    - Define a metric on the state space
    - Generate a random state
    - Choose vertex in graph that is closest to random state (nearest neighbor)
- Expanding the graph:
    - Heuristically choose controls that "pull" chosen vertex to random state
    - Integrate transition equation to produce new state
    - Add new vertex and edge to graph



KPIECE Introduction

- **K**inodynamic Motion **P**lanning by **I**nterior-**E**xterior **C**ell **E**xploration
- *"A Sampling-Based Tree Planner for Systems with Complex Dynamics,"* Sucan and Kavraki, 2012
- Why am I presenting this algorithm?
  - Authors emphasize it is designed specifically for systems with complex dynamics
  - Uses physics engine to compute state transitions
  - Implemented in Open Motion Planning Library

KPIECE Algorithm Pseudocode

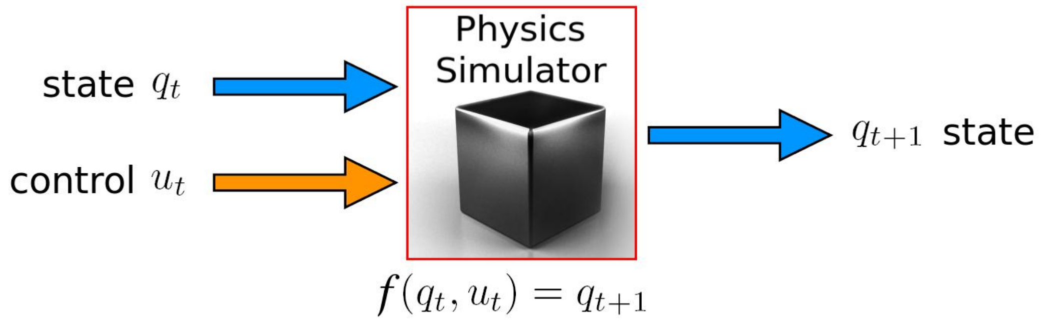
```
1. $T = \mathbf{InitializeTree}(x_1)$
2. $\mathbf{for}\ i \leftarrow 1 \ldots N_{iterations}\ \mathbf{do}$
3. $v = \mathbf{SelectMotion}(T)$
4. $\mathbf{ExpandTree}(T, v)$
5. $\mathbf{if}\ solution\ is\ found\ \mathbf{then}$
6. $\mathbf{return}\ solution$
7. $\mathbf{EvaluateProgress}()$
8. $\mathbf{return}\ no\ solution$
```

How Does KPIECE Relate to Previous Work?

- Uses standard state sampling framework
  - Uses concept of a “motion tree” instead of a “state graph” but this is a technicality
- Uses a physics engine to integrate state transition equation
  - This is not new, but minimizing number of evaluations is a high priority
- Discretizes a projection of state space into cells
  - Not the same as “cell decomposition” for geometric planning
  - Distinction between interior and exterior cells is new
- Contributions
  - Way in which they use information collected during sampling decreases number of evaluations needed relative to other methods

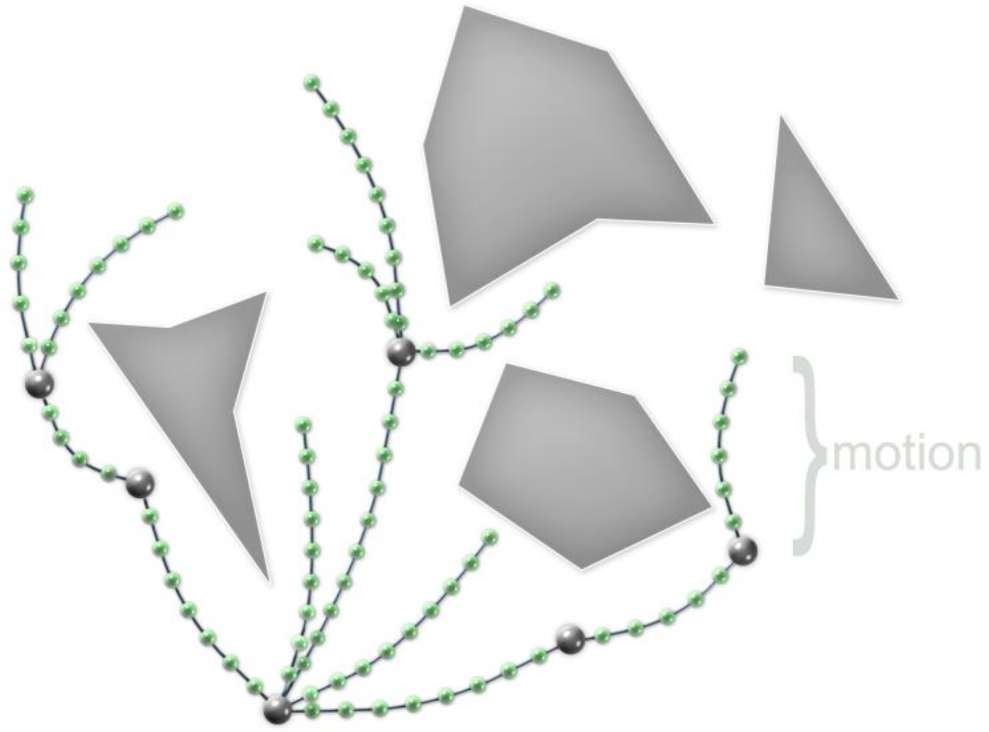
Physics Simulation

- Simulator evaluate f over a *simulation step size* r
- Pros:
  - Physically accurate, can be treated as black-box
  - Can use different simulators
- Cons:
  - Requires uni-directional planning
  - Cost of simulation > cost of integration



Motion Structure 1

- A *motion* v_i is defined as (x_i, u_i, t_i, p_i)
- $x_i \in \mathcal{X}$ is the starting state of the motion
- $u_i \in U$ is the action applied at that state, for duration $t_i \in \mathbb{R}^{\geq 0}$
- t_i is an integer multiple of the simulation step size r
- p_i is the motion's parent motion



Motion Structure 2

- t_i is obtained by sampling a number of simulation steps to perform
- $States(\nu)$ is the set of states along a motion ν
- $AS = \bigcup_{\nu} States(\nu)$ is the set of all states in the tree
- Neither $States(\nu)$ nor AS are explicitly stored
- New motions expanded from an existing motion ν can start at any state in $States(\nu)$
- T is a *tree of motions*
- T is initialized with a motion $\nu_1 = (x_1, null, 0, null)$

State Space Projection

- Let $Proj : \mathcal{X} \rightarrow \mathbb{R}^k$ be a projection from $\mathcal{X}$ to a Euclidean space
- Input to algorithm, random default provided
- Purpose of $Proj$ is to provide a space in which to estimate coverage
- The space should be representative for the problem being solved
- To generate a projection from n-dimensional $\mathcal{X}$ to $\mathbb{R}^k$:
    - k vectors in $\mathbb{R}^n$ are randomly sampled from a normal distribution
    - Vectors are orthonormalized, placed into matrix V
- For a state x, and a projection V, the projection of x is $p = \mathrm{V}^T x$

Projection Discretization into Cells

- Define $Coord : \mathbb{R}^k \to \mathbb{Z}^k$ as:

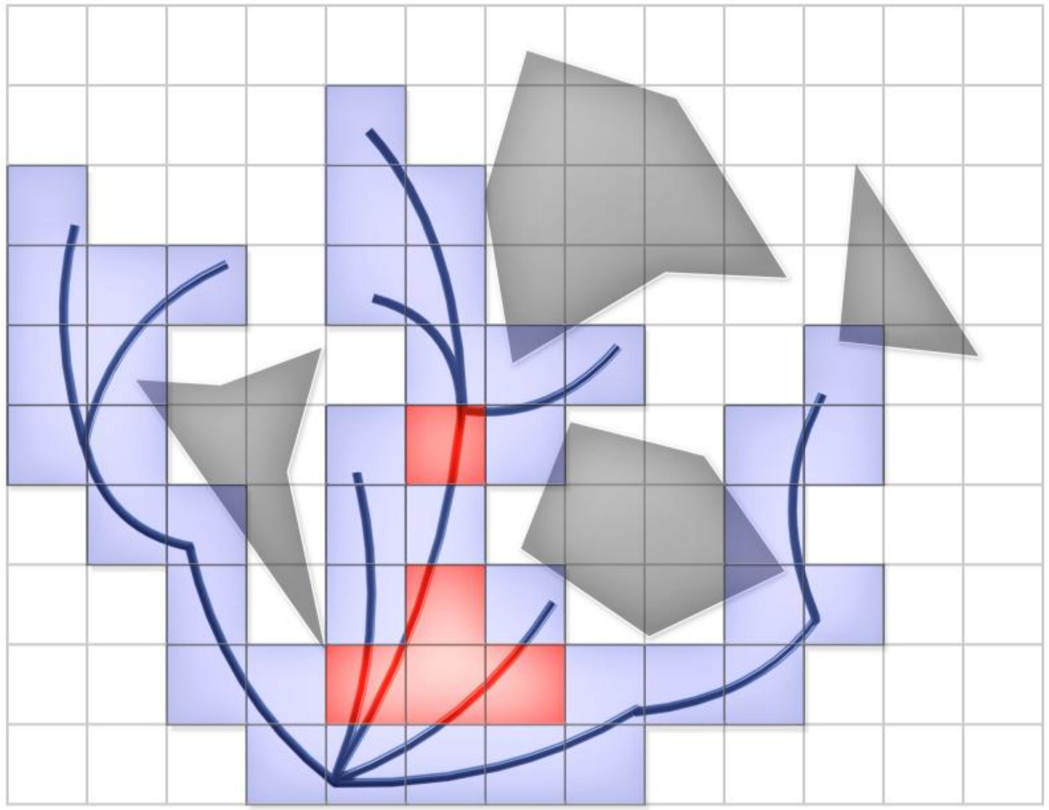
$$Coord(\mathbf{p}) = \left(\left\lfloor \frac{p_1 - o_1}{d_1} \right\rfloor, \dots, \left\lfloor \frac{p_k - o_k}{d_k} \right\rfloor \right) = \mathbf{z}$$

- $\mathbf{p} = (p_1, \dots, p_k) \in \mathbb{R}^k$
- $\mathbf{o} = (o_1, \dots, o_k) \in \mathbb{R}^k$ is an arbitrary point designated as the origin
- How to select d_i is discussed later
- $\forall \mathbf{z} \in \mathbb{Z}^k$, define the corresponding cell in $\mathcal{X}$ to be:

$$Cell(\mathbf{z}) = \{x \in \mathcal{X} \mid Coord\big(Proj(x)\big) = \mathbf{z}\}$$

Assigning Motions to Cells

- Motions are part of a cell if all their states are in the cell
- Every motion resides in one cell
- Motions are split before adding them to the tree of motions
- When a motion ν is to be added, $States(\nu)$ is generated
- For every $x \in State(\nu)$, $Coord(Proj(x))$ is computed
- Decide which parts of the motion go to which cells



State Space Coverage Estimation

- For every cell coordinate $\mathbf{z} \in \mathbb{Z}^k$, the coverage of $Cell(\mathbf{z})$ is

$$Coverage(\mathbf{z}) = \sum_{\nu=(x,u,t)\in Motions(\mathbf{z})} \left(1 + \frac{t}{r}\right)$$

- In other words, coverage is the number of states in a cell
- Assume coverage estimates for cells are relevant for coverage of $\mathcal{X}$
- Let $M_c \subset \mathbb{Z}^k$ denote the minimum # of cells needed to cover AS
- Cells included in M_c are called *instantiated cells*

Distinguishing Interior and Exterior Cells

- For every $\mathbf{z} = (\mathrm{z}_1, \dots, \mathrm{z}_k) \in \mathbb{Z}^k$, the neighbors of $Cell(\mathbf{z})$ are
$$Neighbors(\mathbf{z}) = \{Cell(w) \in M_c \mid w = (\mathrm{z}_1, \dots, \mathrm{z}_{i-1}, y, \mathrm{z}_{i+1}, \dots, \mathrm{z}_k), \\ \text{for } y = \mathrm{z}_i - 1 \text{ or } y = \mathrm{z}_i + 1\}$$
- The max # of neighbors is $2k$
- Cells with max # of neighboring cells are considered interior
- Cells with less than max # of neighboring cells are considered exterior
- Focusing the exploration on exterior cells allow the motion planner to cover state space faster
- For high dimensional spaces the definition of interior cells can be relaxed

Assigning Cells Importance 1

- Importance of cells is used in selection of motions
- Cells are maintained in heaps for quick access
- Prefers:
  - expanding from cells that are less covered
  - cells that have been selected for expansion fewer times
  - cells with fewer neighbors
  - cells that have been instantiated more recently
  - cells that led to “good” progress (i.e. more coverage per simulation time spent)

Assigning Cells Importance 2

- $Importance(\mathbf{z}) = \frac{\log(\mathcal{I})\cdot\text{score}}{\mathcal{S}\cdot(1+|Neighbors(\mathbf{z})|)\cdot Coverage(\mathbf{z})}$
- $\mathcal{I}$ is the number of the iteration at which the cell was instantiated
- $\mathcal{S}$ is the number of times the cell was selected for expansion
- score reflects exploration progress
- score is computed by

$$P = \alpha + \beta \cdot \left(\frac{C_{after} - C_{before}}{T_{after} - T_{before}}\right)$$

$$\text{score} = \text{score} \cdot \min(P, 1)$$

Selecting α and β

- α represents the multiplicative penalization of a cell's score if no progress is made from that cell
- α should be less than 1 but not close to zero
- $\alpha = 0.7$ is the default value
- β is chosen such that P ends up being larger than 1 for only expansions that have led to significant progress
- β depends on how coverage is computed
- $\beta = 5$ is the default value

Selecting Cell Sizes

- Depends on the projection used
- Guidelines:
  - $<$ 10% of the motions cover more than 2 cells in one simulation step
  - $>$ 50% of the motions need to be 3 simulation time-steps or longer
  - Avg. number of parts in which a motion is split should be small (1-4)
  - At least some interior cells need to be created
  - Avg. number of samples per cell should range from tens to hundreds
- Info collected during execution is used to alter cell sizes
- Algorithm is restarted
- This process converges to “good” cell sizes after a few iterations

$\nu = \mathbf{SelectMotion}(T)$

1. Choose whether to select from an internal or external cell, with preference given largely to external cells (75%)
2. Choose the cell with highest importance from appropriate heap
3. Increase the cell's selection count
4. Randomly select a motion ν from within the cell
5. Return ν

ExpandTree(T, ν)

1. Randomly select a state from $States(\nu)$ to create a new motion ν_n
2. Set ν_n's parent to ν
3. Randomly sample an action
4. Randomly sample a number of steps
5. Run the simulator with the selected action starting at ν_n
6. For each new state in ν_n, compute its cell in $\mathbb{Z}^k$
7. If the motion crosses a cell boundary, break it up
8. Add the motion(s) to appropriate cells

$\mathbf{EvaluateProgress}()$

1. Compute $P = \alpha + \beta \cdot (ratio\ of\ coverage\ to\ time)$
2. Multiply the score of the initially selected cell by $\min(P, 1)$

Performance Comparison

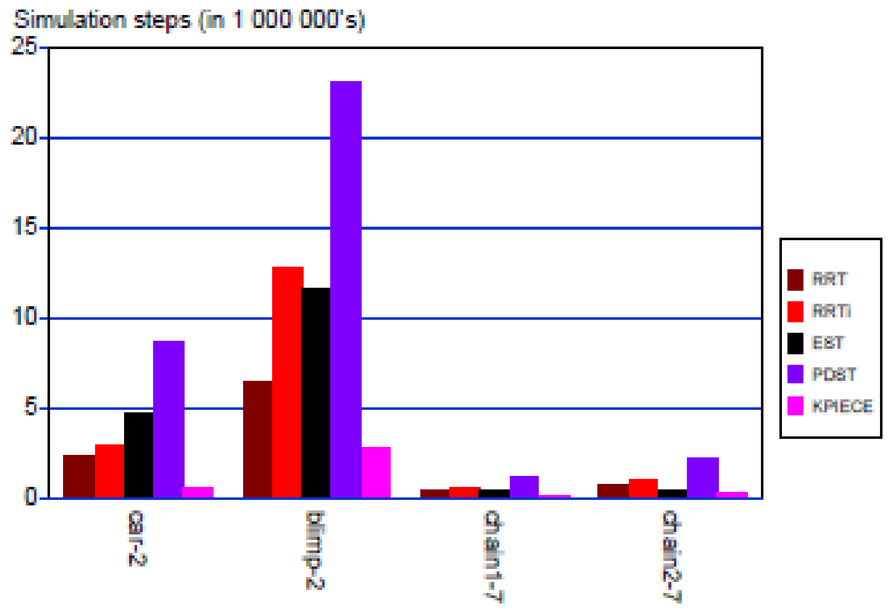


Fig. 8. Averaged number of simulation steps (50 runs) for different algorithms. Notice the similarity to Figure 7. This similarity serves to prove that the runtime of sampling-based planning algorithms is dominated by physics-based simulation, so minimizing the number of simulation steps leads to speedup.

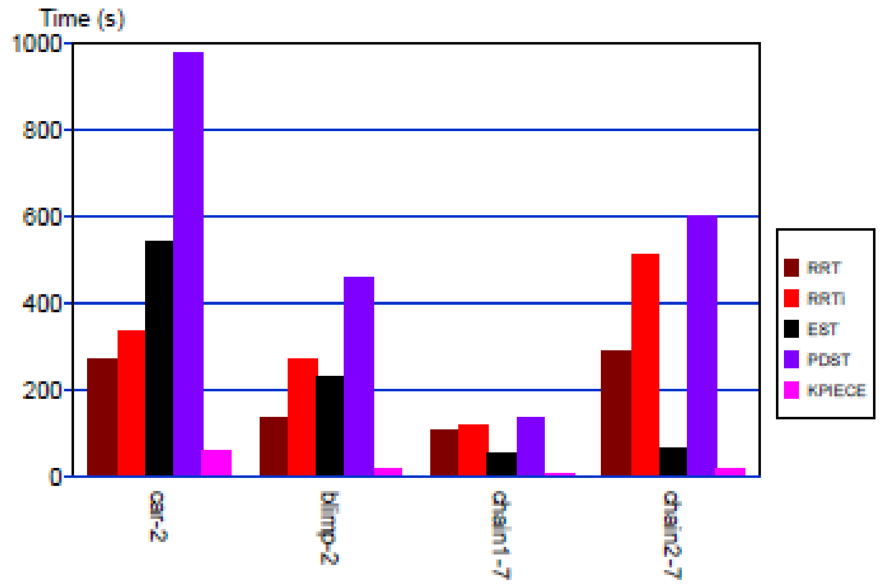
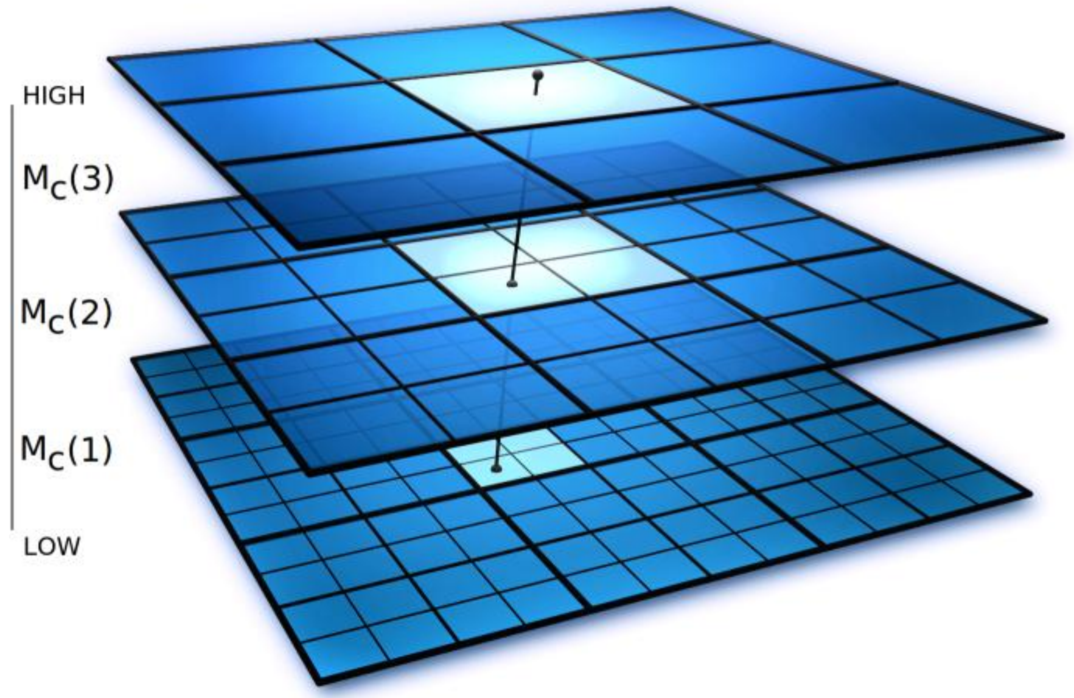


Fig. 7. Averaged runtimes (50 runs) of different algorithms on some of the tested models. The achieved speedup is shown in Table I.

KPIECE Algorithm Extensions

- Multiple levels of discretization
- Goal biasing
  - Algorithm should greedily attempt to reach the goal
- Parallel implementation



Recommended Reading and MP Software

- Books
  - Planning Algorithms, LaValle 2006
  - Robot Motion Planning, LaTombe 1998
- Papers
  - http://gamma.cs.unc.edu/research/robotics/
  - http://robotics.cs.unc.edu/publications.html
- Software
  - Open Motion Planning Library, http://ompl.kavrakilab.org/

References

1. Sucan, IA, Kavraki LE. 2012. **A Sampling-Based Tree Planner for Systems With Complex Dynamics**. IEEE Transactions on Robotics.
2. LaValle, S. 2006. **Planning Algorithms**. Cambridge Press.
3. J. H. Reif. 1979. **Complexity of the mover's problem and generalizations.** IEEE Symposium on Foundations of Computer Science.
4. P. Cheng, G. Pappas, and V. Kumar. 2007. **Decidability of motion planning with differential constraints**. IEEE International Conference on Robotics and Automation.

Thanks!