RRT based Path Planning in Static and Dynamic Environment for Model Cars

David Gödicke
Freie Universität Berlin

Bachelorthesis presentation, 2018

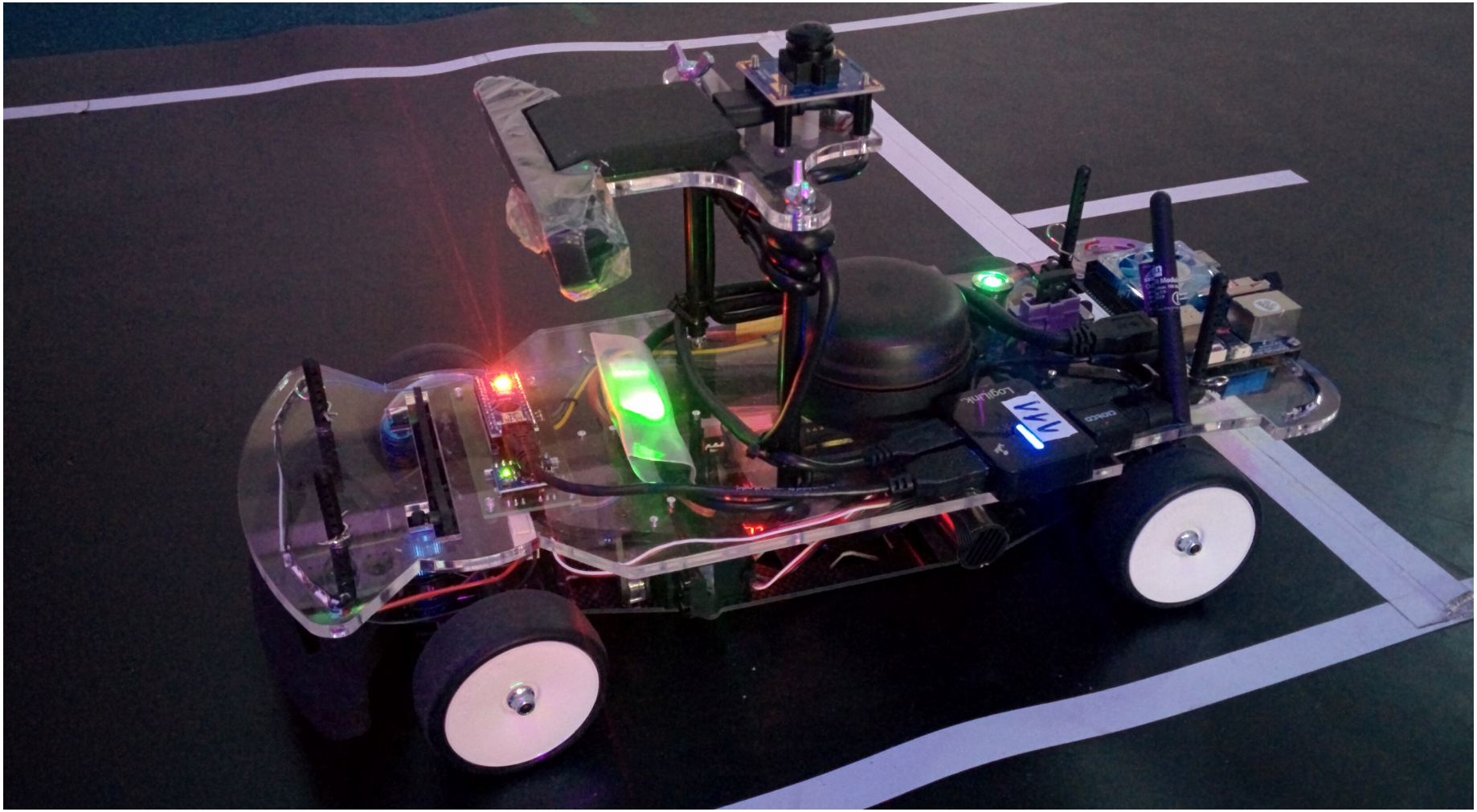
Outline

Outline

AutoNOMOS Mini v3.

- Developed at the FU-Berlin for educational purpose.
- Odroid board running ROS kinetic over Ubuntu 16.0.
- Sensors:
  - MPU6050
  - RPLIDAR A2 360
  - Fish-Eye Camera (ELP 1080p)
  - Intel RealSense SR300 Camera
  - Odometry (motor feedback)
- Motors:
  - FAULHABER 2232 (Speed)
  - HS-645MG (Steering)

AutoNOMOS Mini v3



Path Planning for cars

- Essential to be considered autonomous:
  - Computing a path between two positions.
  - The path should be optimal.
  - The path must avoid obstacles.
  - The car must be able to follow the path.
- Requires:
  - Planning algorithm
  - World model
  - Localization
  - Control model

Outline

Dijkstra, A*

- Optimal planning over a given graph
- Vertices and edges can have different cost
- A* is usually faster.
- The graph must be finite and build
- Common solution: Grid based maps
- Deterministic
- Consequences:
  - Discrete space
  - Predefined resolution
  - Does not scale well in high dimensions
  - Motion between cells are too sharp for the car

Rapidly-exploring Random Tree

- Presented by Steven M. LaValle in 1998.
- Incremental sampling based algorithm.
- Planning in continuous and high dimensional space.
- Vertices are connected with optimal motions.
- Each vertex has a parent.
- Following the parent gives the path to the initial state.
- The path is not optimal.

RRT

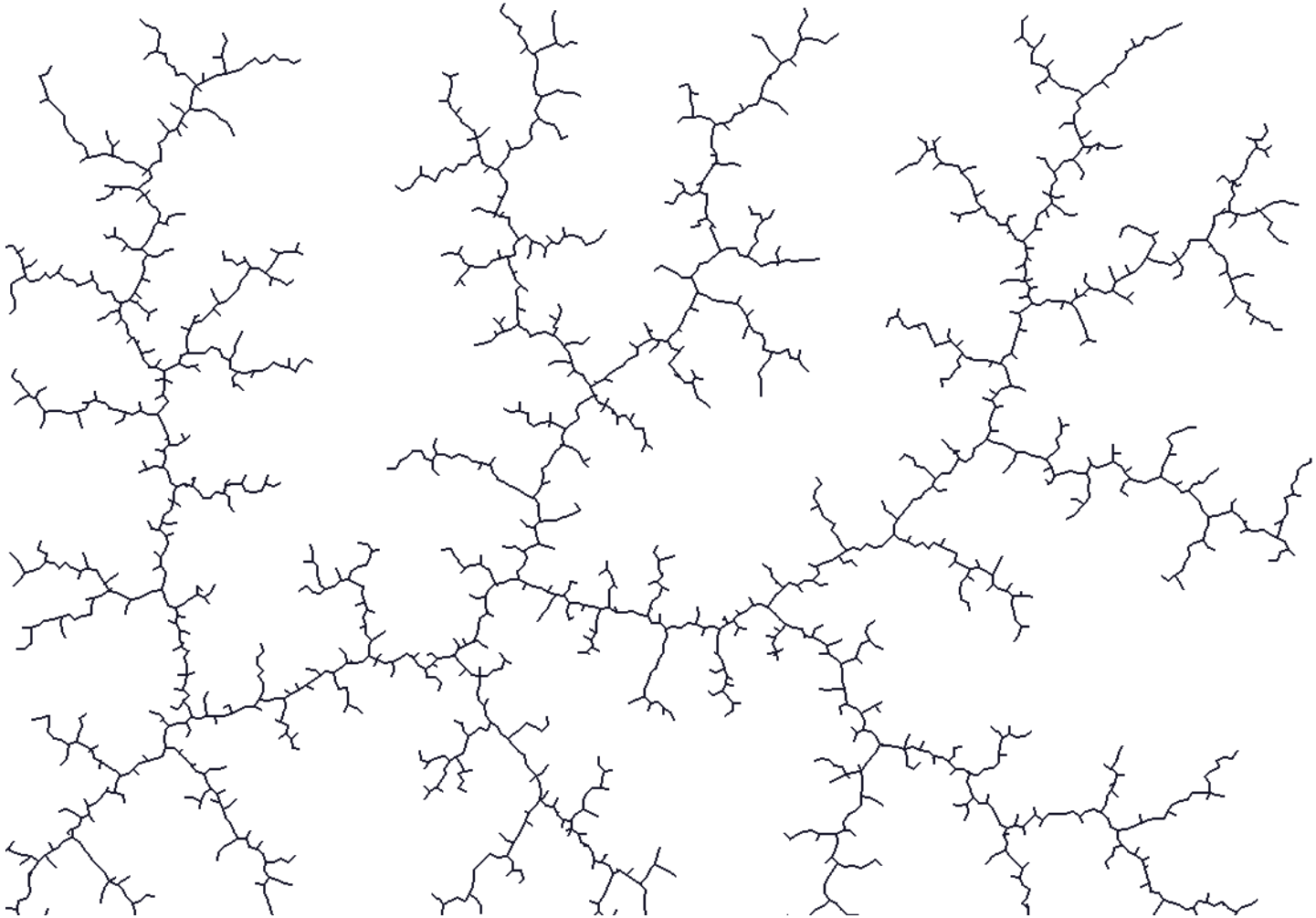


Figure: *RRT*

RRT star

- Presented by S. Karaman and E. Frazzoli.
- Same datastructure.
- Store cost values
- Rewires neighbors after each vertex insertion.
- Asymptotically optimal.
- Shrinking radius
- Complexity $O(n \log n)$

RRT star

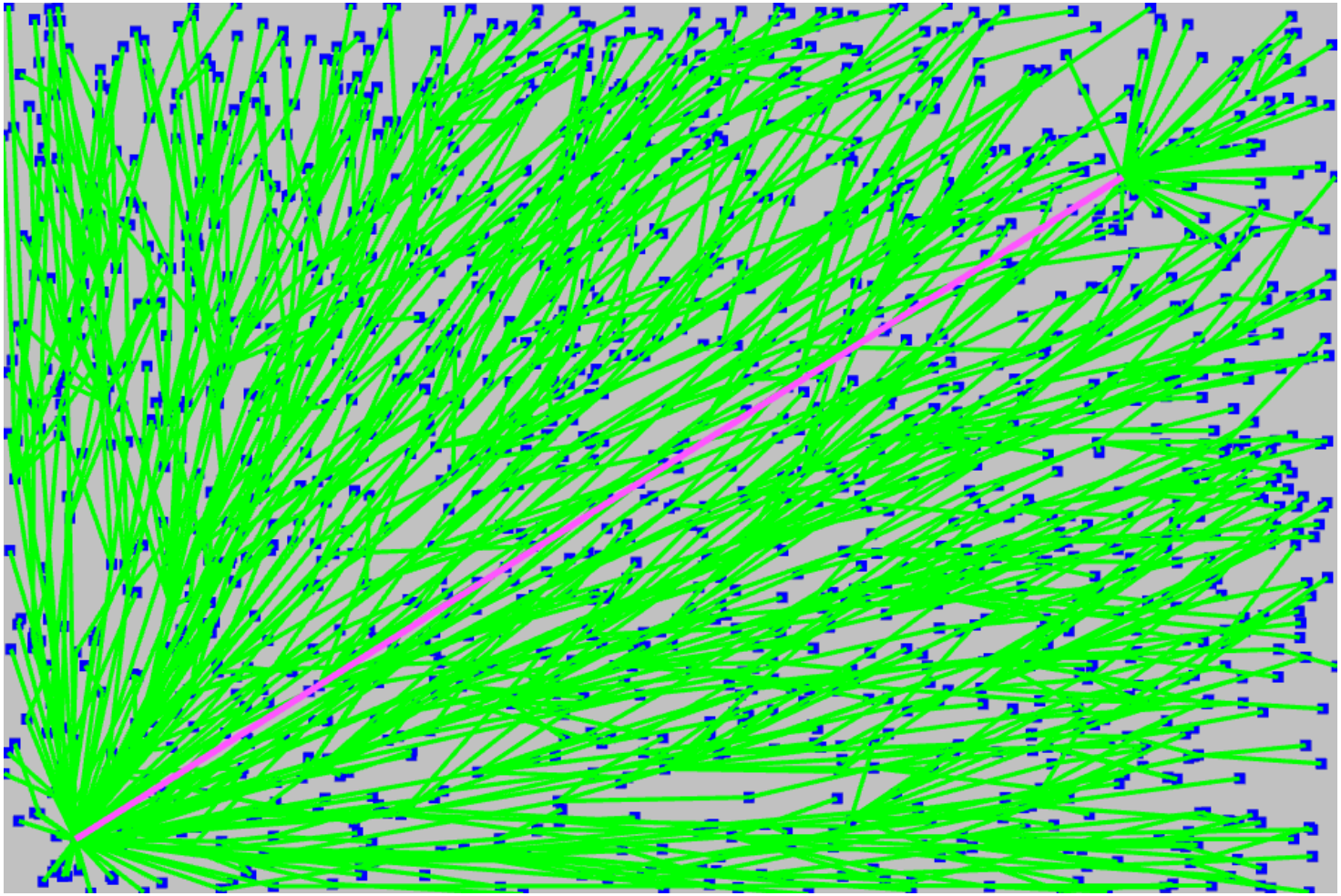


Figure: *RRT**

Dynamic Planning

- Previous algorithm are static planners.
- In real world situations obstacles are not predictable.
- The path has to be constantly updated when driving.
- Dynamic planners can repair the optimal path.
- Example:
  - D star
  - RRTx
- Hypothesis:
  - A dynamic planner could be more efficient

Outline

Configuration Space

- Orientation: θ.
- Position: (x, y).
- Steering angle: ϕ.
- Wheelbase: L.
- Turning radius: p

- Configuration space: $C = \mathbb{R}^2 \times \mathbb{S}$.
- A configuration $q \in C$ is $q = (x, y, \theta)$
- We consider that ϕ can change instantly.
- $\phi = \arctan\left(\frac{L}{p}\right)$

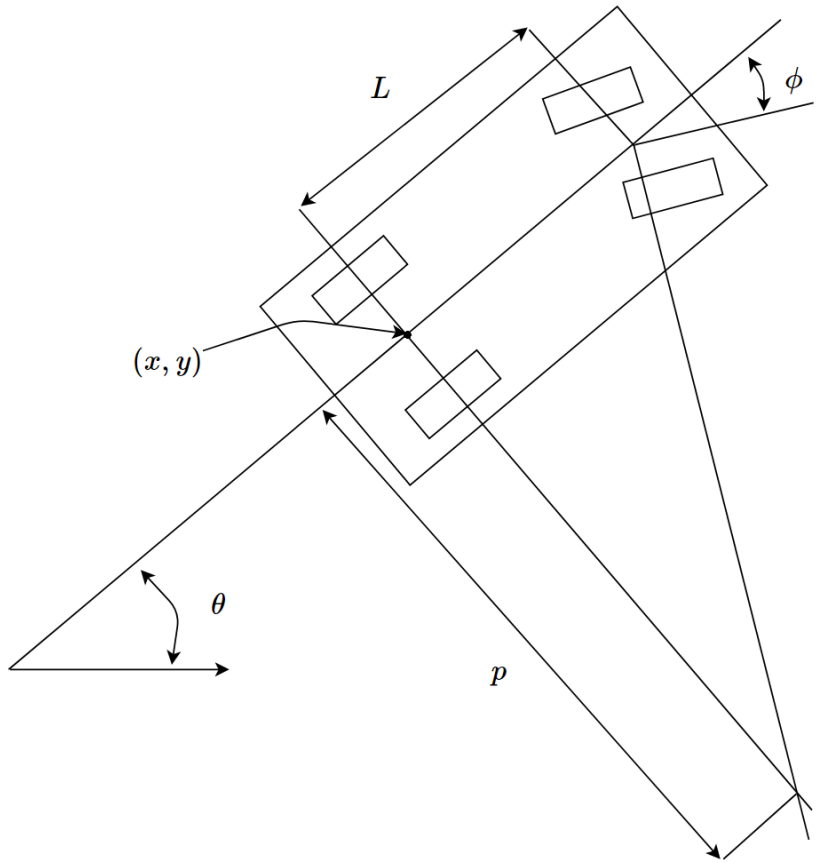


Figure: Simple car

Kinematic Constraints

- The car is a non-holonomic robot
- The car has two axis of freedom:
  - Speed along its own x axis
  - Steering angle ϕ
- ϕ is constrained by the mechanical limitations
- $|\phi| \leq \phi_{max}$
- Given two input variables u_s and u_ϕ we can express the car motion as:

$$\dot{x} = u_s \cos\theta$$

$$\dot{y} = u_s \sin\theta$$

$$\dot{\theta} = \frac{u_s}{L} \tan u_\phi$$

- With $|u_\phi| \leq \phi_{max}$ the steering angle and u_s the speed
- $\phi_{max} = \arctan\left(\frac{L}{p_{min}}\right)$

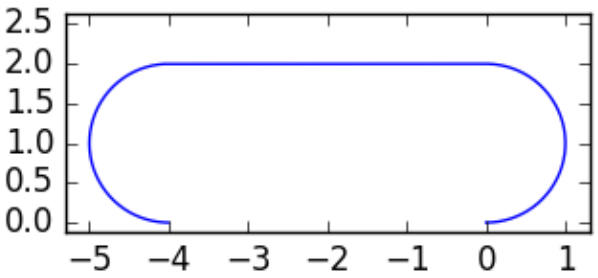
Dubins Curve

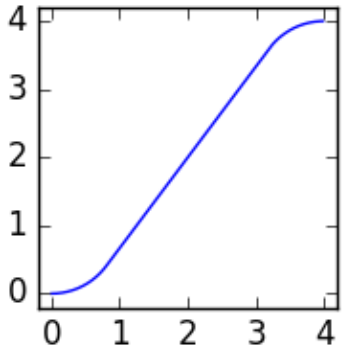
- Defined over $\mathbb{R}^2 \times \mathbb{S}$.
- Optimal curvature bounded path between two positions
- Each curve consist of 3 motion primitives called symbols:
  - (L) Left: $u_\phi = -\phi_{max}$.
  - (S) Straight: $u_\phi = 0$.
  - (R) Right $u_\phi = \phi_{max}$.
- The speed is constant and positive $u_s = 1$
- A sequence of 3 symbols is called a word.
- Dubins proves that 6 words are possibly optimal:

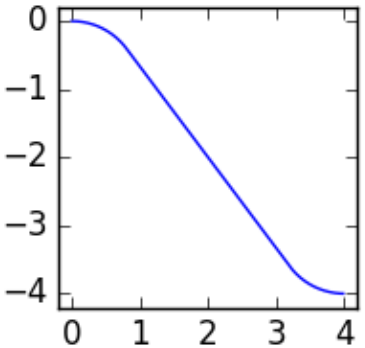
$$\{LRL, RLR, LSL, LSR, RSL, RSR\}$$

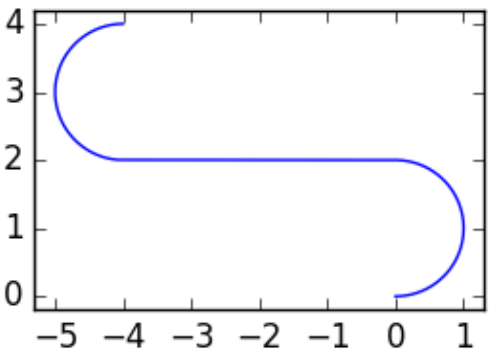
- Each word is tested, the smallest curve is the optimal Dubins curve.
- Dubins curve are not symmetric.
- Backward driving is not possible.

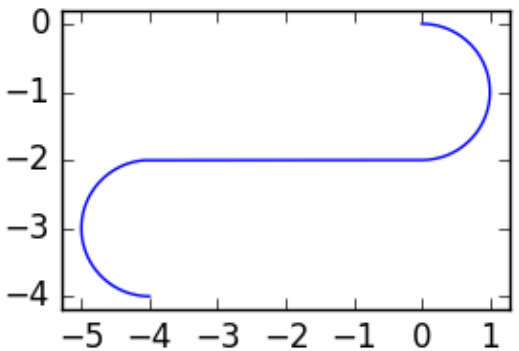
Dubins Curve

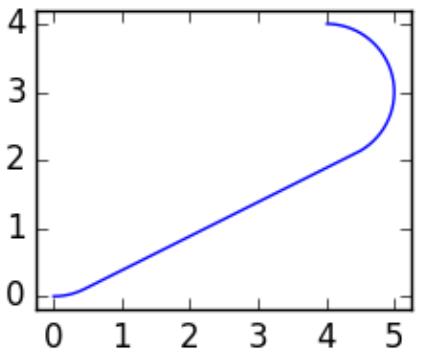












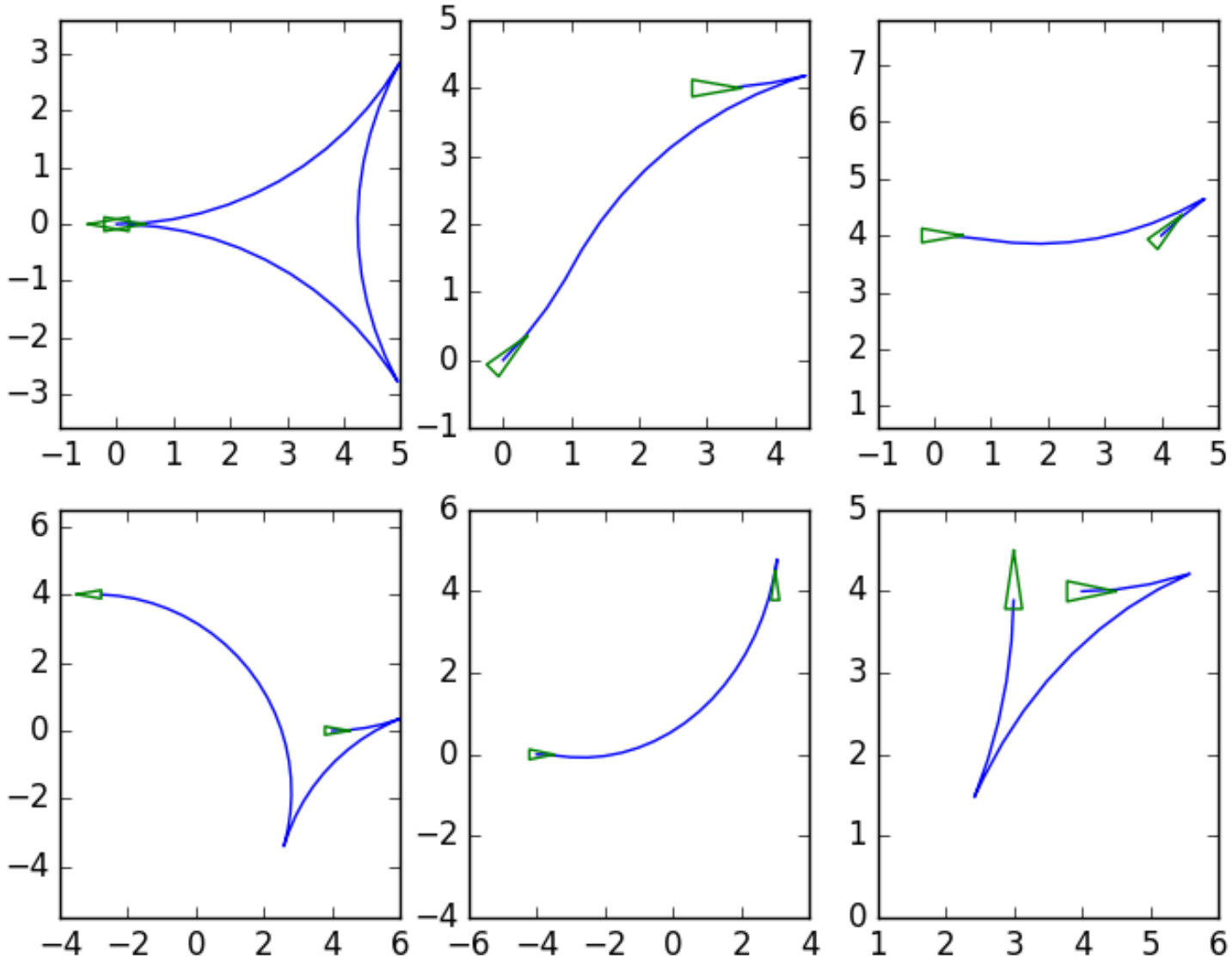
Reeds Shepp Curve

- Same symbols as Dubins curve.
- Speed input u_s can also be negative.
- Backward paths are possible.
- Reeds Shepp curves are symmetric
- The control system can be defined as:

$$\begin{aligned} \dot{x} &= u_1 \cos\theta \\ \dot{y} &= u_1 \sin\theta \\ \dot{\theta} &= u_1 u_2 \end{aligned} \tag{1}$$

- Speed: $u_1 \in \{-1, 0, 1\}$
- Steering: $u_2 \in \left\{\frac{1}{-p_{min}}, 0, \frac{1}{p_{min}}\right\}$

Reeds Shepp Curve



Problem Definition

- Formal definition

$$P : (x_{start}, x_{goal}) \rightarrow m^*(x_{start}, x_{goal})$$

- where

$$m^*(x_{start}, x_{goal}) = x_{start}, x_1, x_2, .., x_{goal}$$

- and each motion between two consecutive states is collision free.

Outline

Notation

- Parameters:
  - state space: X
  - maximum distance η
  - epsilon: ϵ
- Each state x has following values:
  - Cost-to-goal $g(x)$
  - Look-ahead cost $lmc(x)$
  - Parent $p(x)$
  - Children $C(x)$
  - Neighbors $N^+(x)$ and $N^-(x)$
- between two states we define:
  - motion: $m(x_1, x_2)$
  - distance: $d(x_1, x_2)$
  - motion cost: $c(x_1, x_2)$

Primitive Functions

- $Sample(X)$
- $Near(x, r)$
- $Nearest(x, r)$
- $Steer(v, u, r)$
- $CollisionFree(v, u)$

- Complexity of $O(\log n)$ is required for *Near* and *Nearest*
- Datastructure:
  - Kd-Tree
  - Geometric Near-neighbor Access Tree.
  - FLANN

Shrinking Ball Radius

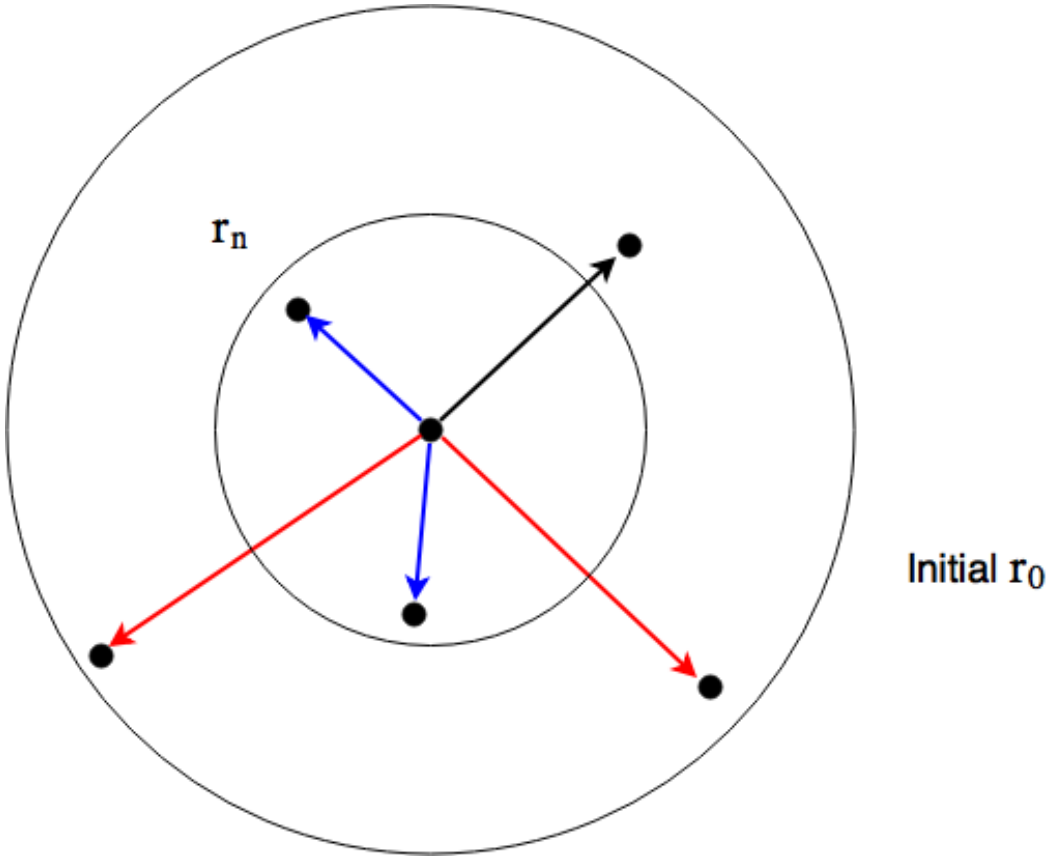
- Neighbors are selected in radius r
- $N(x) = N_0(x) \cup N_r(x)$
- Initial neighbors are set during insertion
- Running neighbors are set by other vertices
- The radius is reduced over time
- Running neighbors are culled

$$r = min\left(Y_{RRT^*}\left(\frac{\log|V|}{|V|}\right), \eta\right)$$

$$Y_{RRT^*} > \left(2\left(1+\frac{1}{d}\right)\right)^{1/d}\left(\frac{\mu(X_{free})}{C_d}\right)^{1/d}$$

- C_d is the volume of the unit ball with dimension d
- $\mu(X_{free})$ is the free space in X

Shrinking Ball Radius



- Red: Initial neighbors
- Blue: Running neighbors
- Black: Culled neighbors

Initialization

- Graph $G = (V, E)$ is rooted at x_{goal}
  - $V = \{v_{goal}\}$
  - $E = \{\emptyset\}$
  - $g(x_{goal}) = 0$
  - $lmc(x_{goal}) = 0$
- Sampled states are initialized with:
  - $g(x_{goal}) = \infty$
  - $lmc(x_{goal}) = \infty$

Expansion

- Starts at goal
- Insert new states one by one:
  - Update radius
  - Sample
  - Find nearest
  - Steer
  - Extend
  - Rewire neighbors
  - Reduce Inconsistency
- Follow parent from start to goal

Extend

- Extend a new vertex x
- Find neighbors near x in radius r
- Test motions for collision
- Find best parent
- Insert initial edges to x
- Insert running edges to neighbors

Rewiring and Reduce

- Use a priority queue Q
- Order vertices by *lmc* value
- Q contains inconsistent vertices
- Rewire neighbors of x:
  - Cull neighbors
  - Test if neighbors have better path through x
  - If yes, update *lmc* value and insert into Q
- Reduce inconsistency:
  - Take vertex u with lowest *lmc* value from Q
  - Rewire neighbors of u
  - Make u consistent $g(u) = lmc(u)$

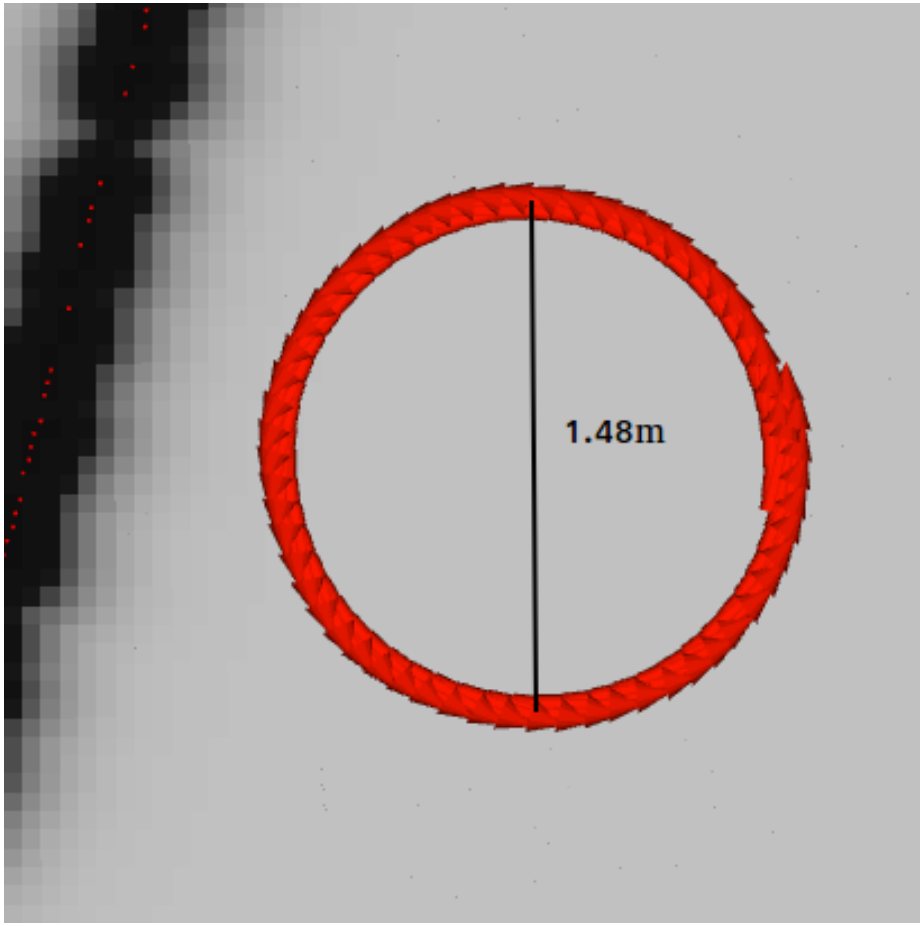
Re-planning

- Removing Obstacles:
  - Find edges (u, v) that became obstacle free.
  - Update cost $c(u, v)$.
  - Rewire u.
  - Reduce inconsistency.
- Add Obstacles:
  - Find edges (u, v) that collide with obstacle.
  - If $P(u) = v$, u becomes orphan.
  - Vertices with an orphan parent also become orphan.
  - For each orphan v, $lmc(v) = \infty$ and $g(v) = \infty$.
  - For each non orphan neighbor v, $lmc(v) = \infty$.
  - Insert non orphan neighbors into Q.
  - Reduce inconsistency.
- Updating position is simple: Sample and Extend one new state at the new position

Outline

Constraints

- Basic motion: Reeds Shepp curve
- Minimum turning radius: 0.74
- Car dimension: $0.30cm$ x $0.15cm$



Localization

- ROS frame convention:
  - World: earth
  - Map: map
  - Odometry: odom
  - Car: base_link
- Additional frames: laser

- Position in base_link $p = (0, 0, 0)$
- Transform between odom and base_link is published by odometry
- Transform between map and base_link is computed by GPS
- Transform between map and odom using transform chaining:

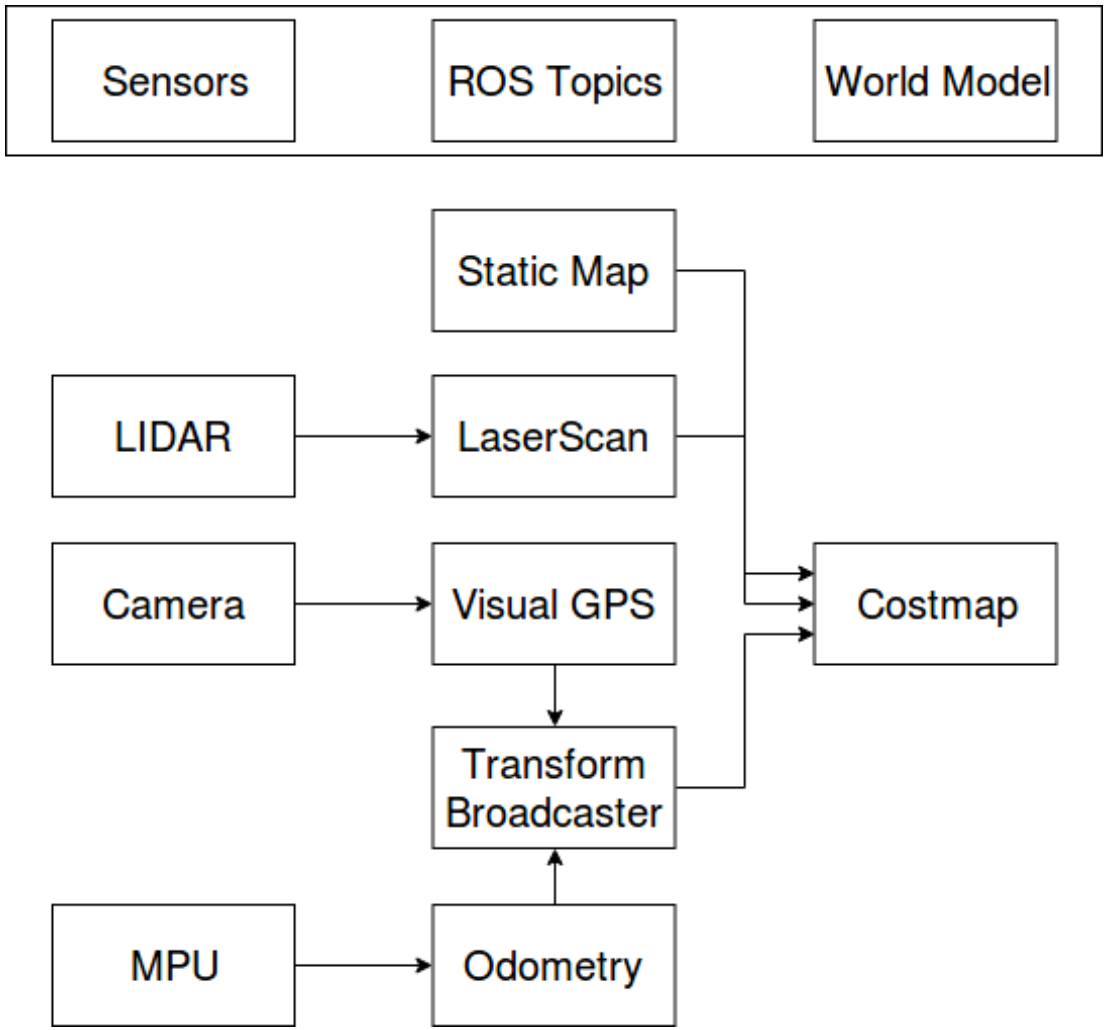
$$T_{map}^{odom} = T_{map}^{base\_link} * (T_{odom}^{base\_link})^{-1}$$

- A Kalman filter can be used to avoid position jumps

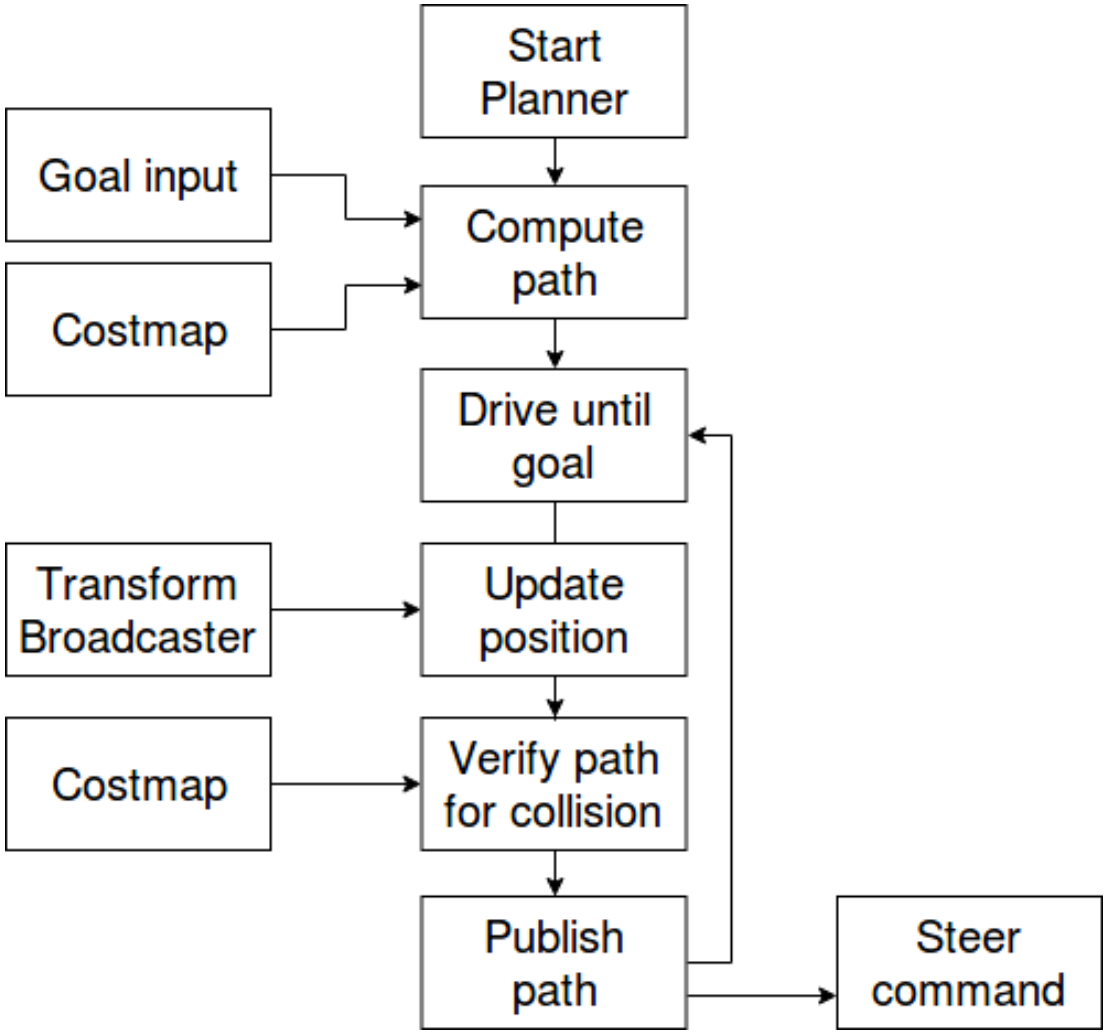
World Model

- LIDAR scanner for obstacle detection
- Odometry and GPS for localization
- Low resolution costmap as world model (Grid)

World Model


Sensors
ROS Topics
World Model
Static Map
LIDAR
LaserScan
Camera
Visual GPS
Costmap
Transform Broadcaster
MPU
Odometry

Planner

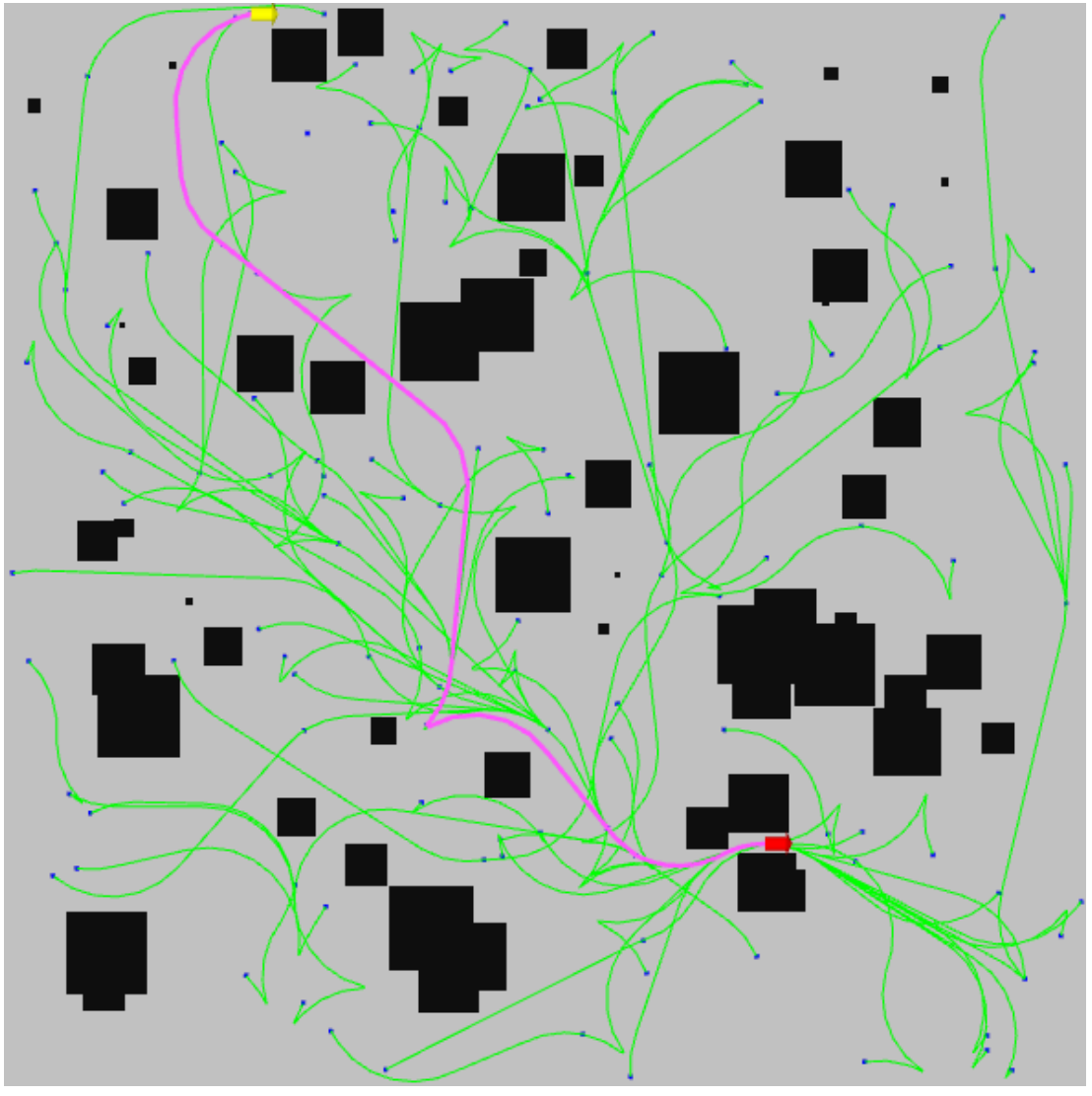

Start Planner
Goal input
Compute path
Costmap
Drive until goal
Transform Broadcaster
Update position
Costmap
Verify path for collision
Publish path
Steer command

Outline

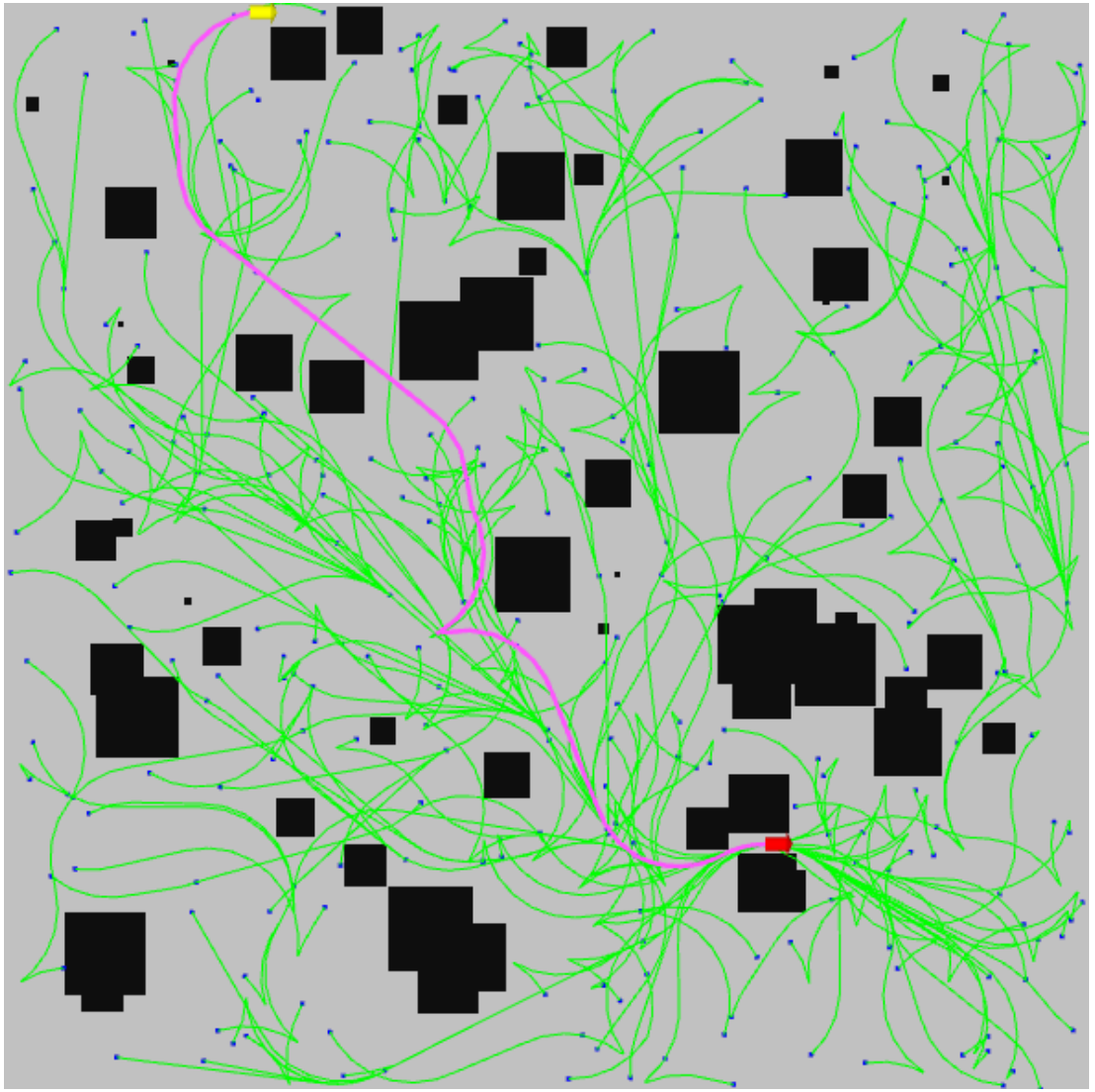
Sampling Time

- Finding a path between obstacles
- Fast sampling vs long sampling

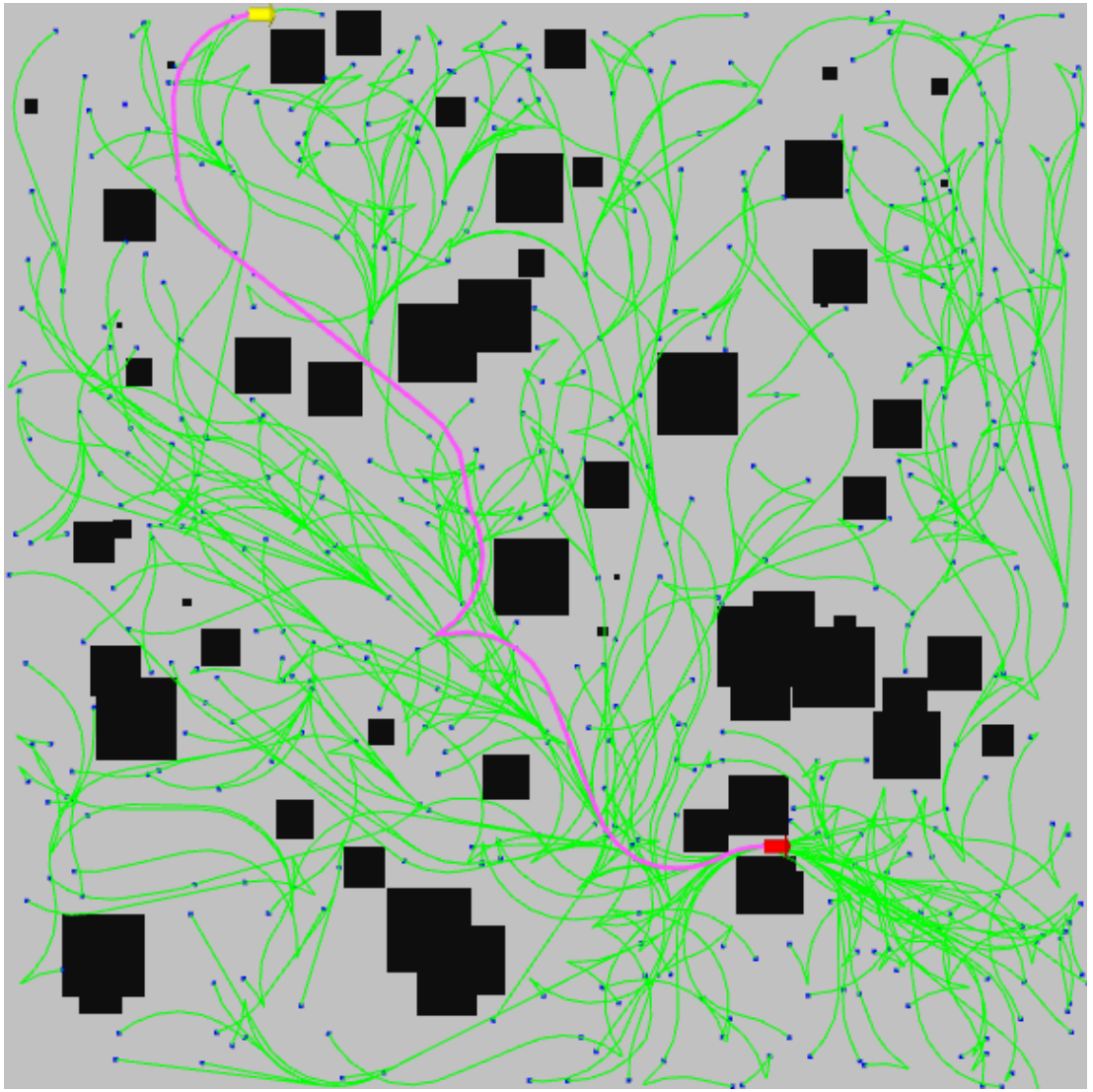
Sampling Time: 0.5*s*



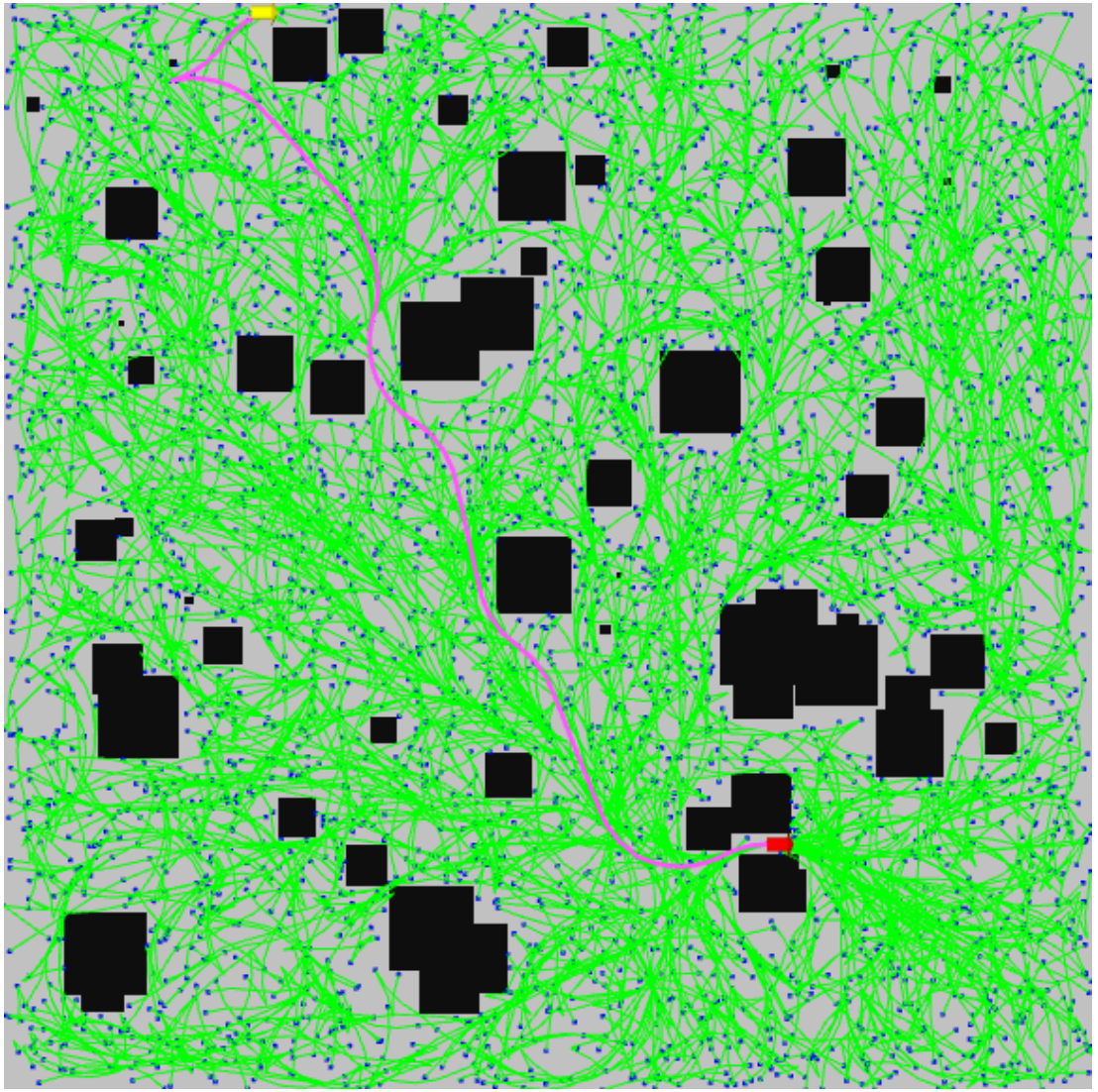
Sampling Time: 1.5*s*



Sampling Time: 2.5*s*



Sampling Time: 35*s*



Sampling Time

Table: Optimal path cost

Time	0.5s	1.5s	2.5s	35s
Cost	14.72	14.39	14.39	14.0

- An optimal path is found at $t = 0.5s$
- Only 5% gain between $t = 0.5s$ and $t = 35s$

Cost over Time

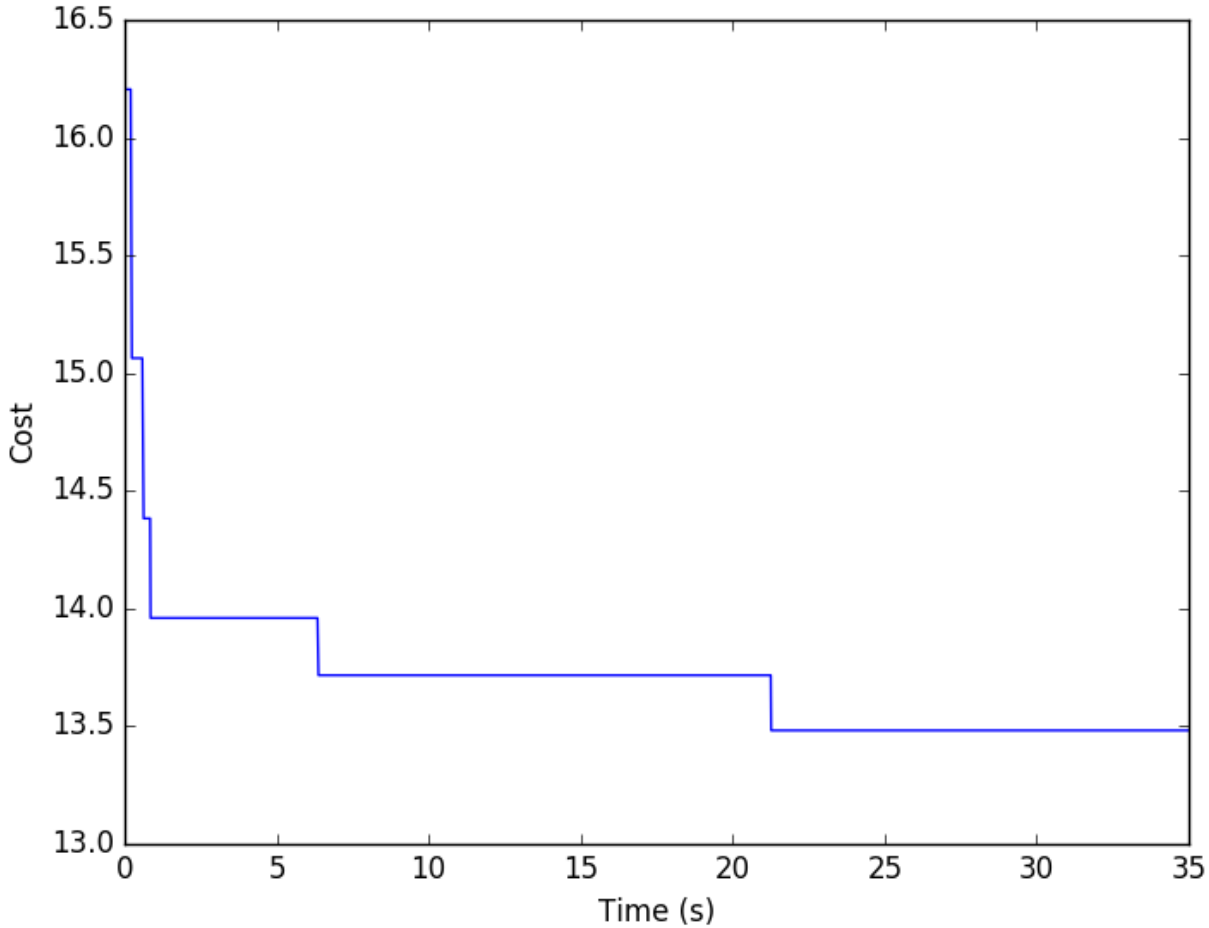


Figure: Cost over Time

Insertion Complexity

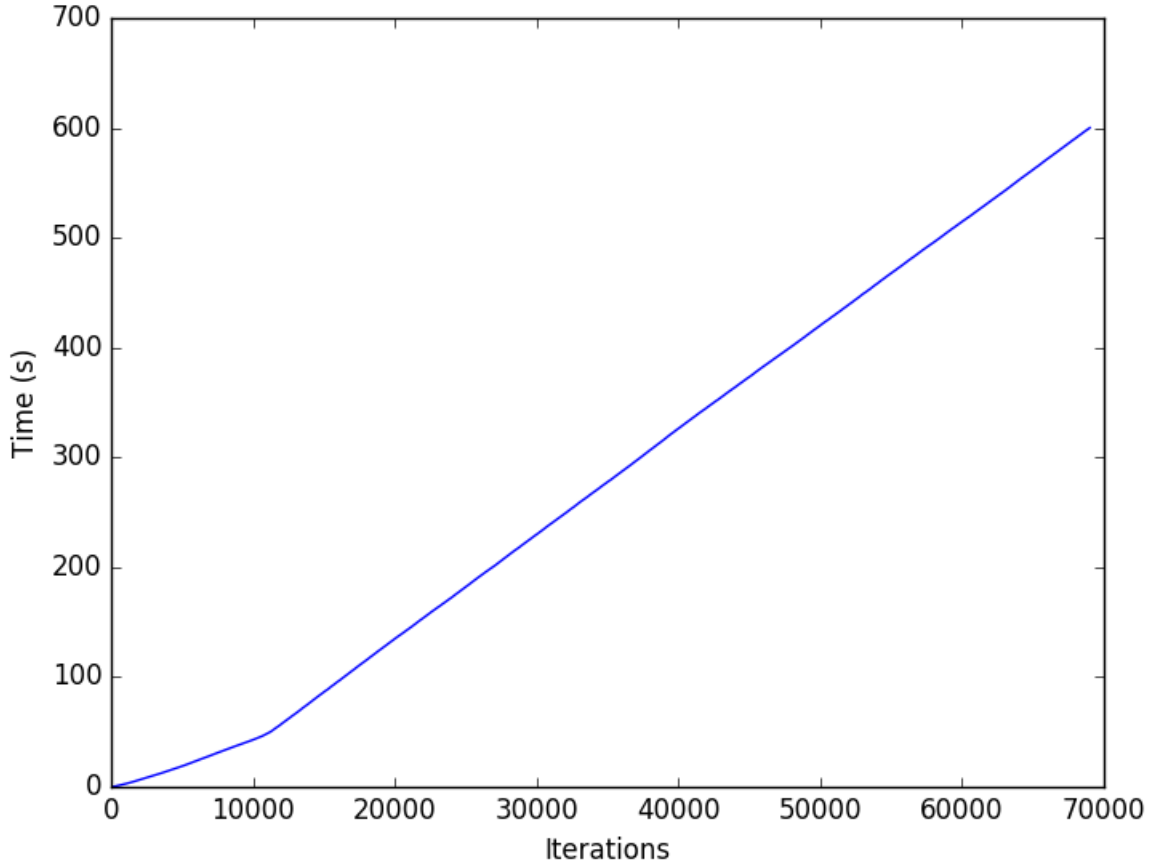


Figure: Time per Iterations

Shrinking Ball Radius

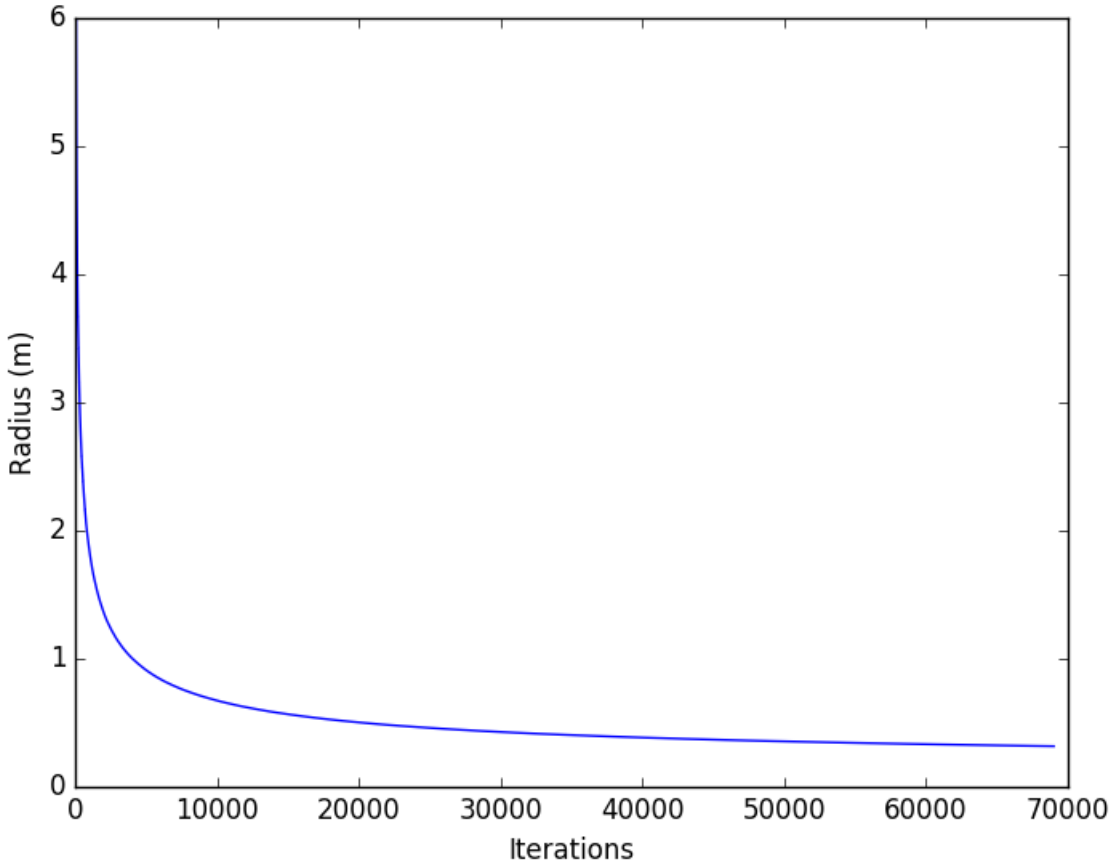


Figure: Radius over Iterations

Sampling Distance

- Benefits of long distances:
  - The tree expands faster.
  - The path has less curves when sampling for a small time.
- Small trajectories have less chances to collide with an obstacle.

Sampling Distance: $0.5m$

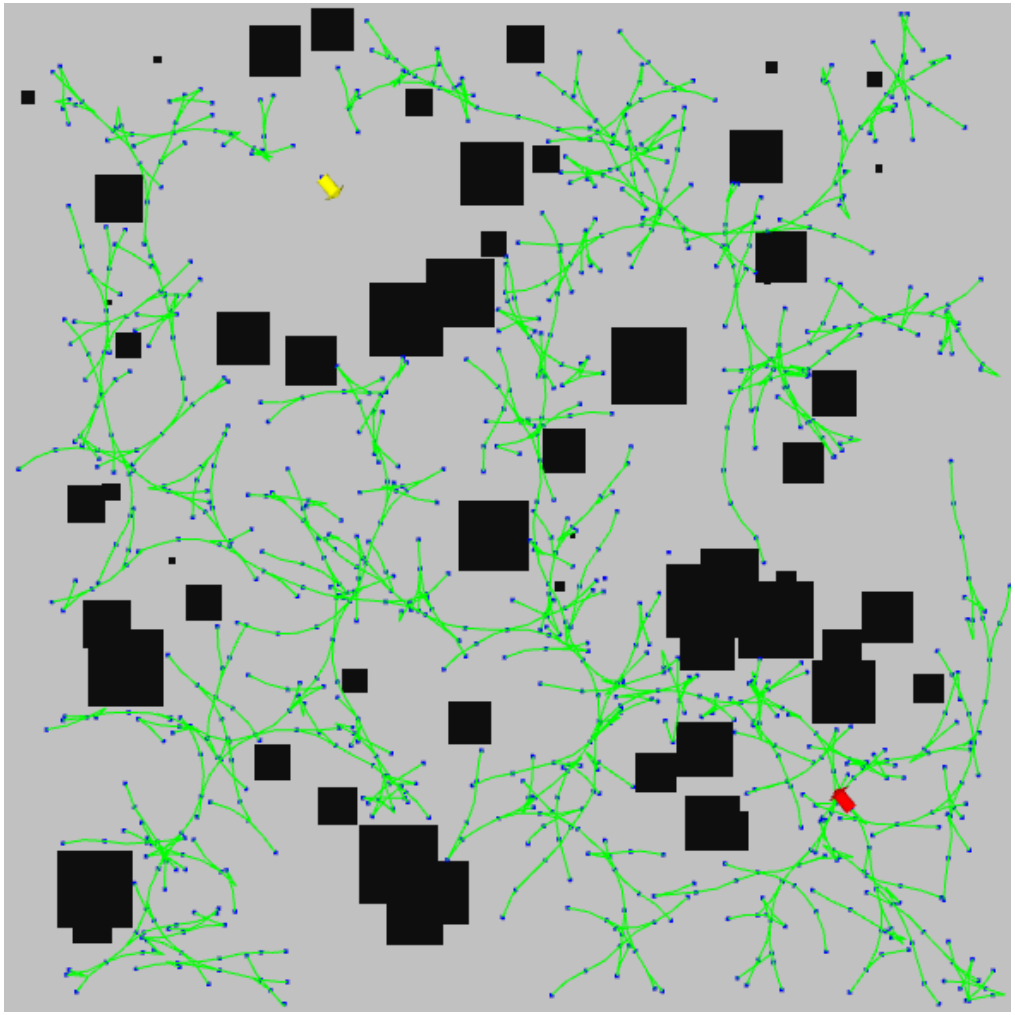


Figure: Optimal Tree at $t = 1s$ with $\eta = 0.5m$

Sampling Distance: $5m$

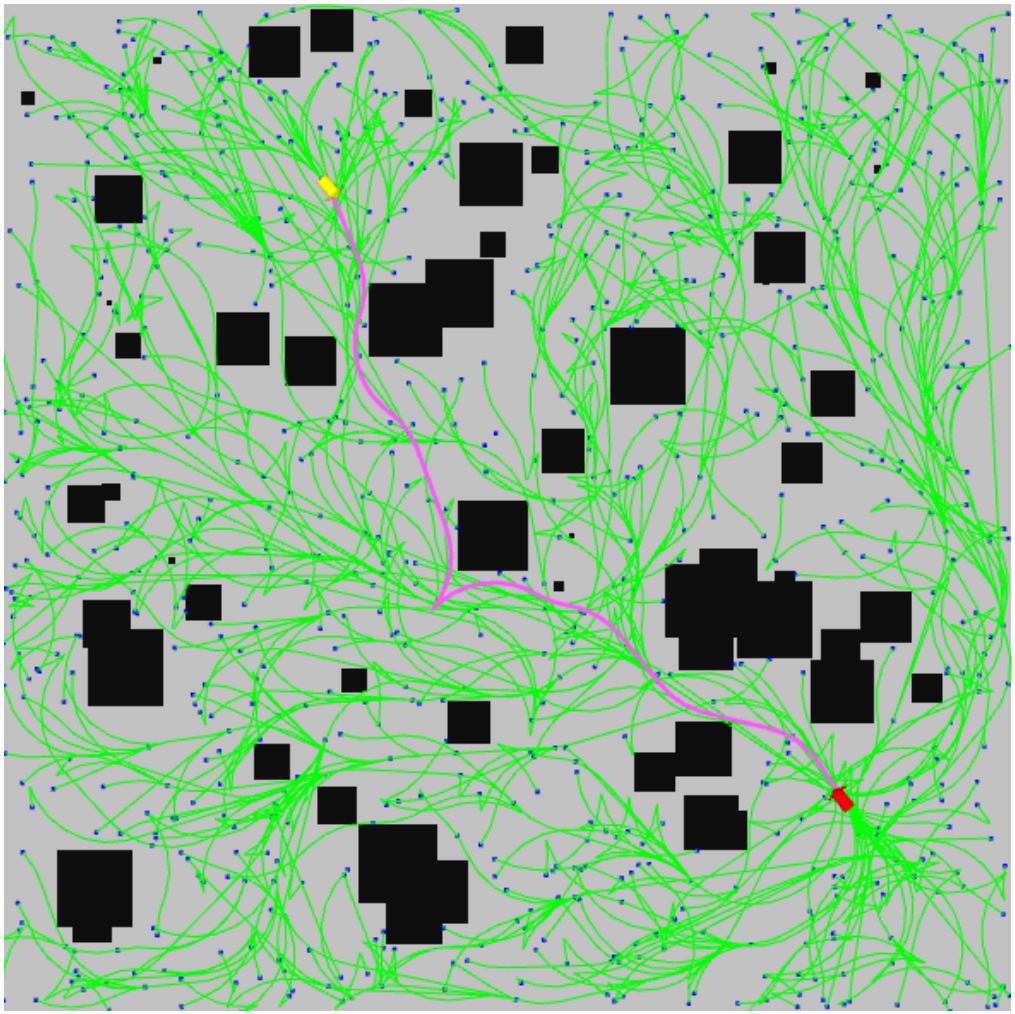


Figure: Optimal Tree at $t = 1s$ with $\eta = 5m$

Backward Driving

- The car should drive forward when possible.
- We don't want to forbid backward paths.
- Solution: Motion cost penalty.

- $d(v,u) = d_f(v,u) + d_b(v,u)$
- Forward: $d_f(v,u)$
- Backward: $d_b(v,u)$
- Motion cost:

$$c(v,u) = d_f(v,u) + penalty * d_b(v,u)$$

Backward Driving

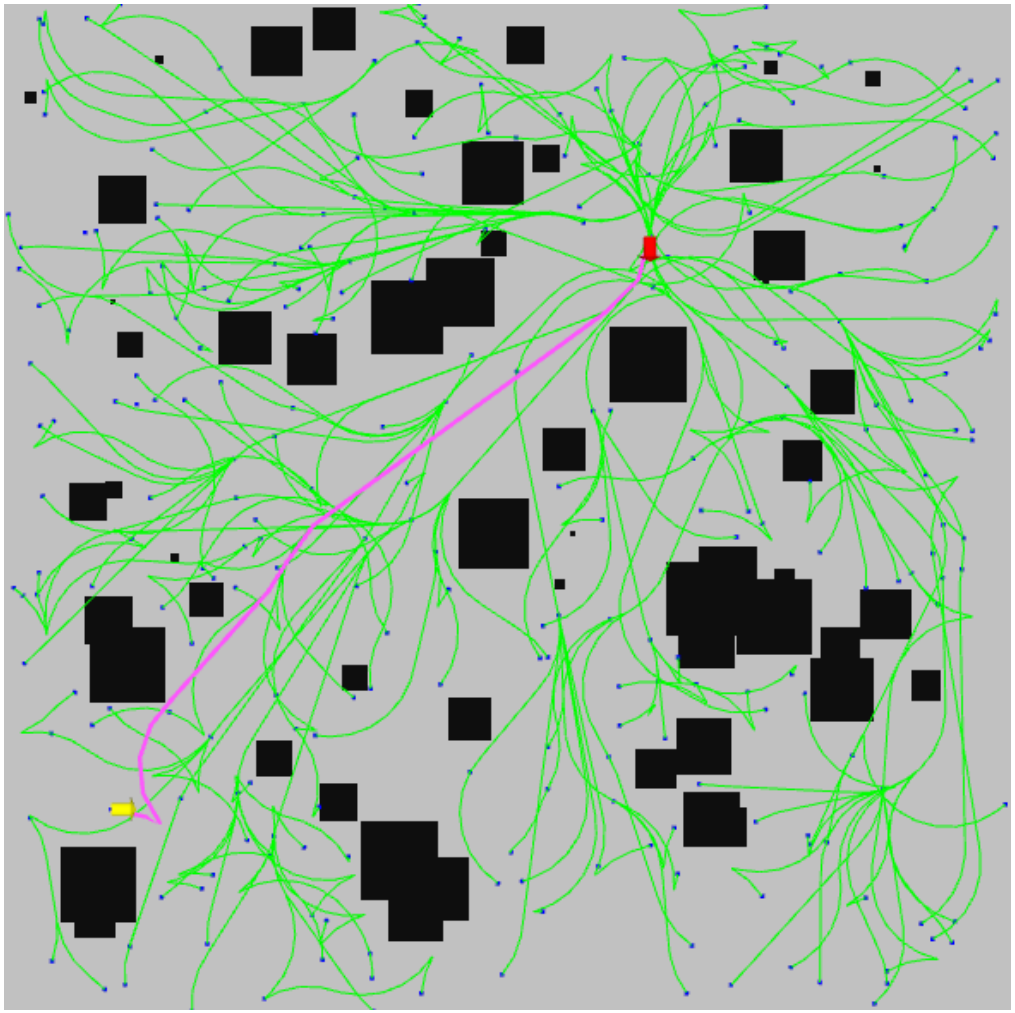


Figure: Driving without motion penalty

Backward Driving

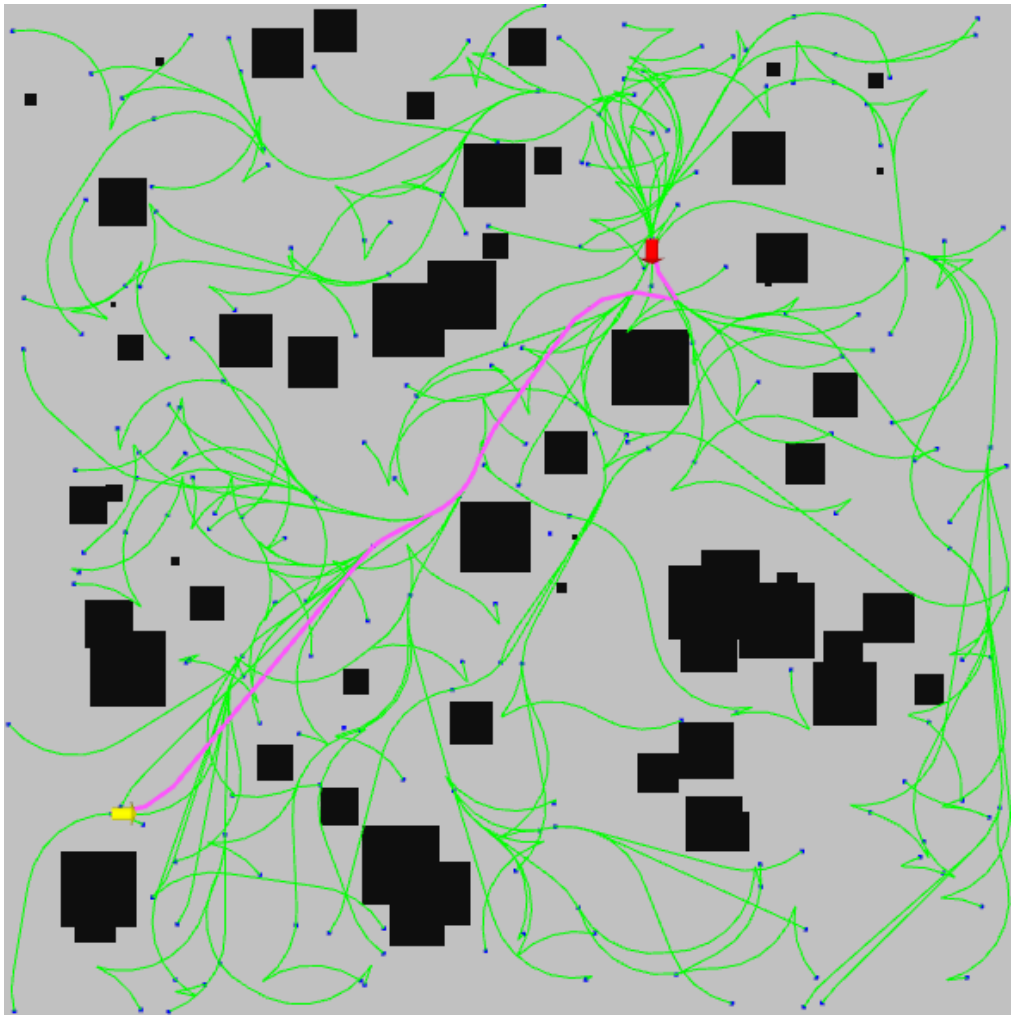


Figure: Driving with motion penalty

Maneuver

- U-turn in constrained environment
- The car has to switch between forward and backward driving
- Problem: Many direction changes

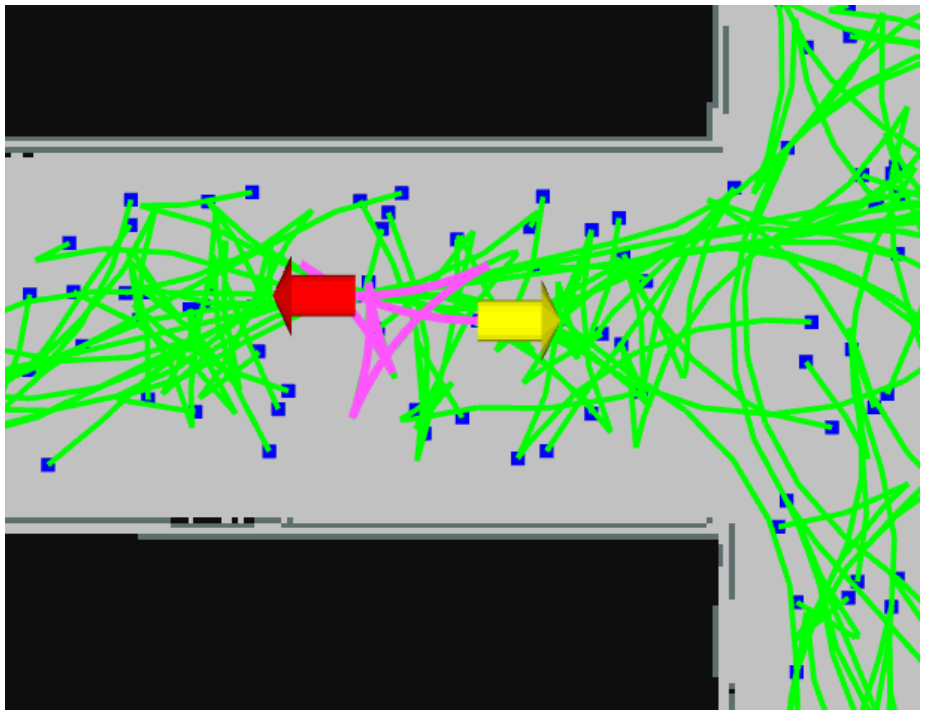


Figure: U-turn without penalty

Maneuver

- Solution: Motion cost penalty on direction changes

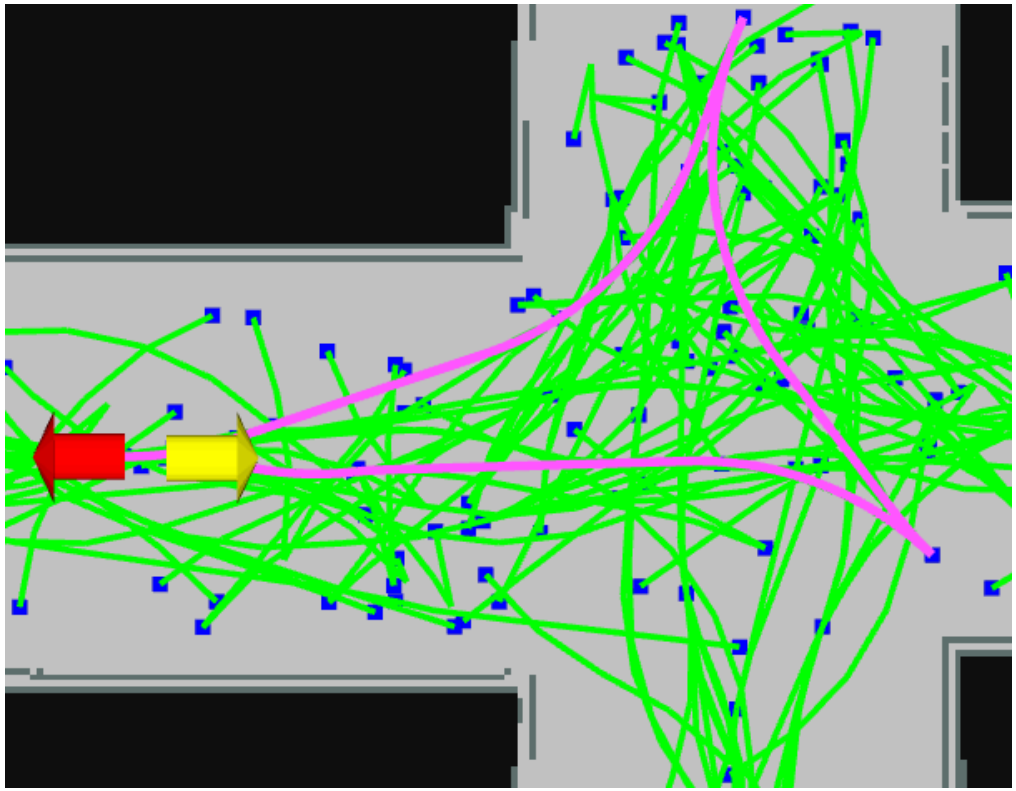


Figure: U-turn with penalty

Re-Planning

- How to detect new obstacles ?
  - Scan all the Costmap.
  - Scan some part of the Costmap.
- Sampling time vs Re-planning time

Re-Planning

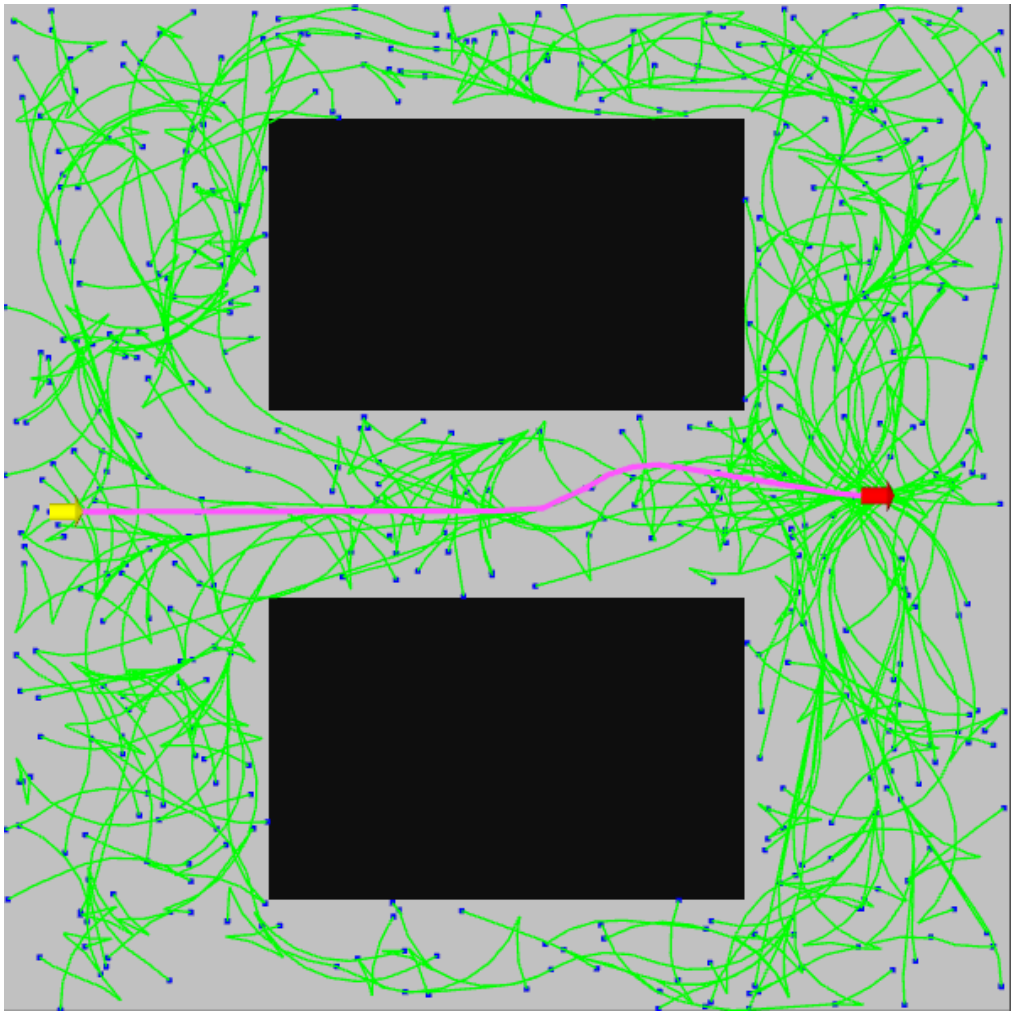


Figure: Initial Expansion

Re-Planning

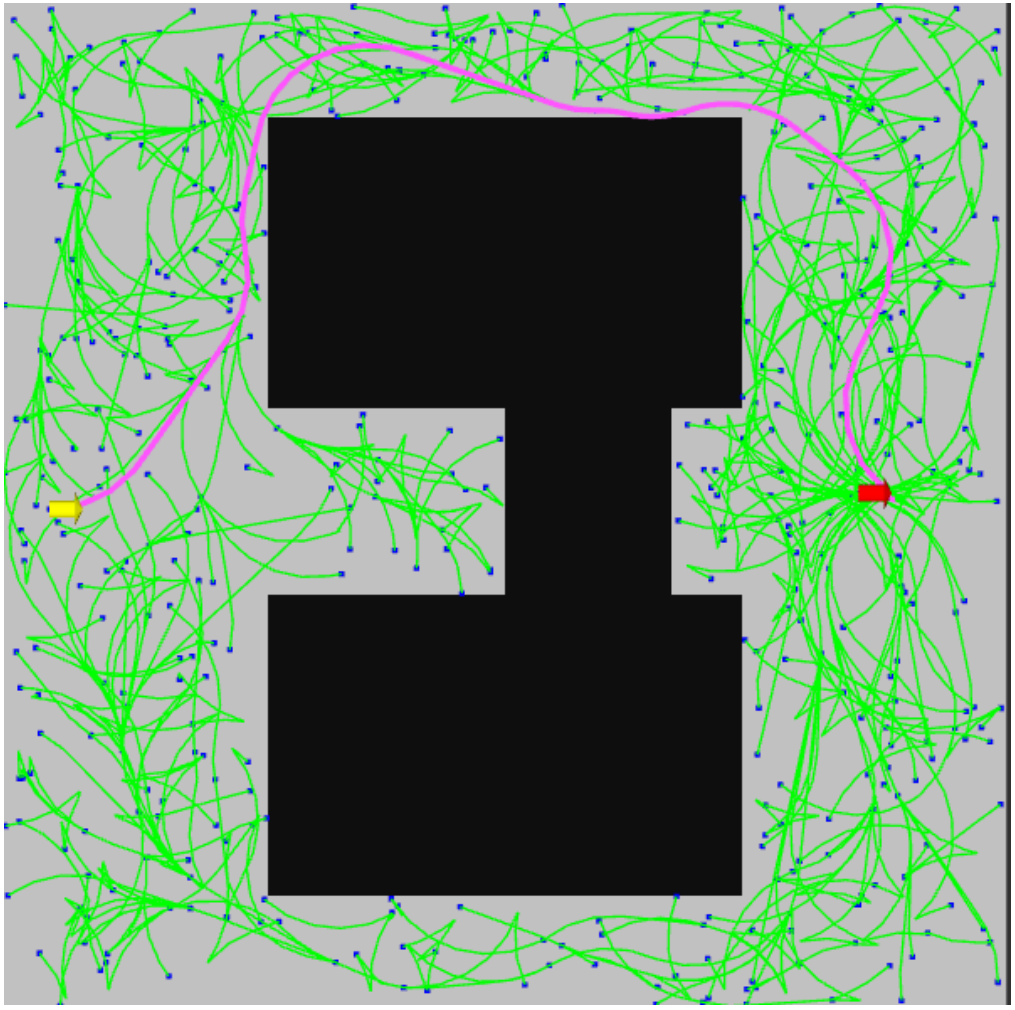


Figure: Corrected path

Re-Planning

- For small sampling time the re-planning benefits are minor.
- Scanning less edges reduces the re-planning time.
- The scanned zone cannot be smaller than the ball with radius η.

Table: Re-planning time

Expansion (s)	1	2	4	8
Scan 100%	0.70	1.1	1.37	1.81
Scan 50%	0.24	0.35	0.63	0.73
Scan 25%	0.15	0.23	0.36	0.40

Driving

- RRT^x is slow on the car.
- Sampling time of minimum $2s$.
- Maximum distance $\eta = 2m$.

Driving: Avoid static obstacle

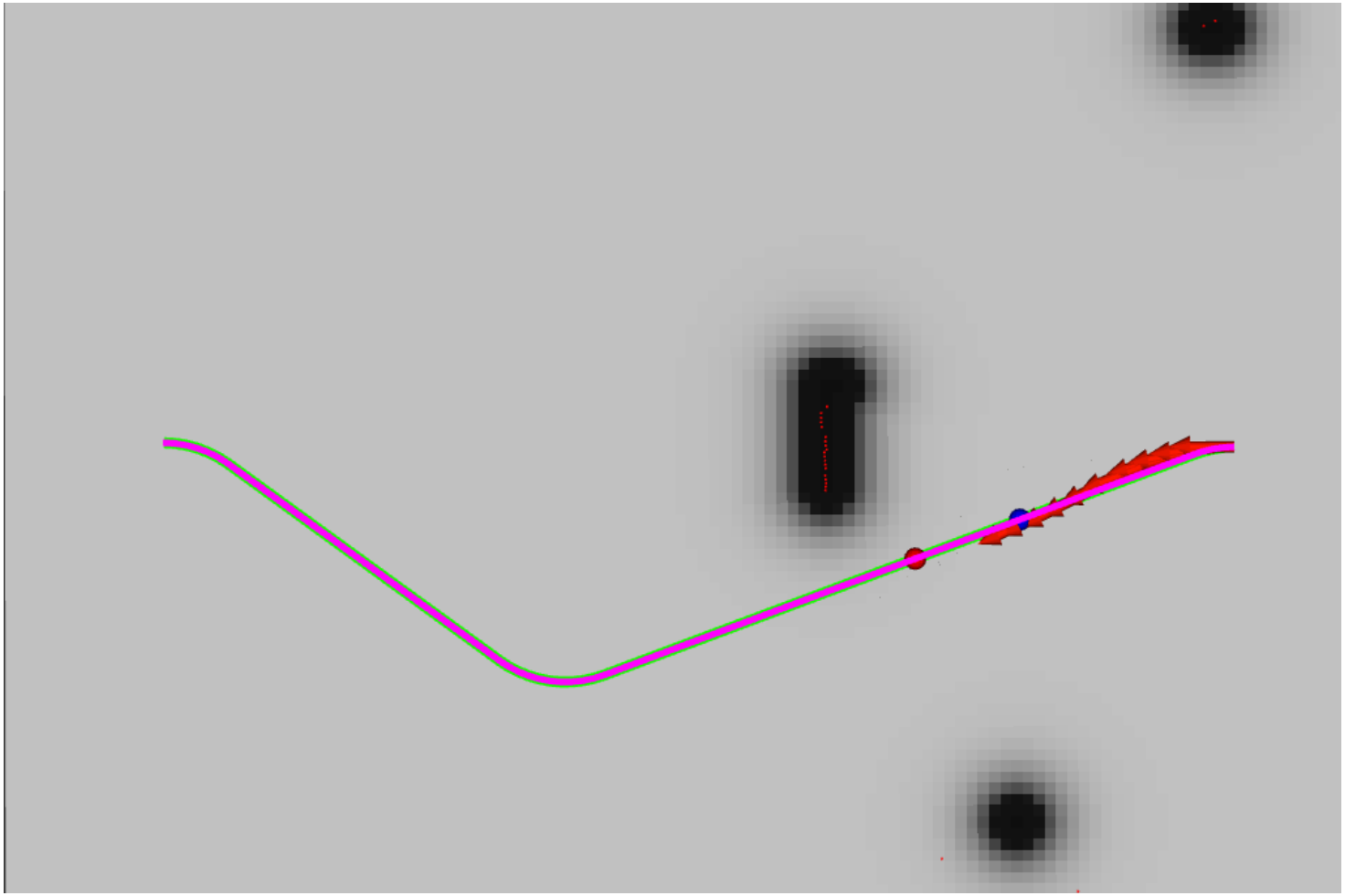


Figure: Path around obstacle

Driving: Avoid dynamic obstacle

- Re-planning time is still long $0.9s$.
- More than 60% of the edges are tested.
- The car should drive slowly and stop to avoid obstacle.
- Hypothesis: RRT^* may be better implemented in OMPL.
- Result: RRT^* computes a good result in $0.5s$.
- RRT^* is recommended for small sampling times.

Limitations

- The implementation of RRT^x does not give better performance than $RRT*$.
- Planning in Time seems not possible with dynamic planners.
- Reeds Shepp curve have instant curvature changes.

Outline

Conclusion

- Both RRT^x and $RRT*$ are able to find a kinematically feasible path
- Re-planning is more efficient for long sampling times.
- Both algorithm are not suited to real time planning.
- Most of the sampled vertices are not improving the path.
- Dynamic planners can not be used to plan in time for the car.

For Further Reading I


Steven M. LaValle.
Planning Algorithms


Lester E. Dubins.
On Plane Curves With Curvature.


J. A. Reeds And L. A. Shepp.
Optimal paths for a car that goes both forwards and backwards.


Steven M. LaValle.
Rapidly Exploring Random Trees: A New Tool For Path Planning


S. Karaman and E. Frazzoli.
Sampling-based Algorithms for Optimal Motion Planning


J. Otte and E. Frazzoli.
RRTx: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles

For Further Reading II


Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot.
Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic


Luigi Palmieri, Sven Koenig, Kai O. Arras
RRT-Based Nonholonomic Motion Planning Using Any-Angle Path Biasing


Wim Meeussen
REP 105: Coordinate Frames for Mobile Platforms
http://www.ros.org/reps/rep-0105.html


*Sergey Brin**
Near Neighbor Search in Large Metric Spaces


Marius Muja and David G. Lowe
Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration