An Extension to the Dynamic Window Approach for arbitrarily shaped Robots

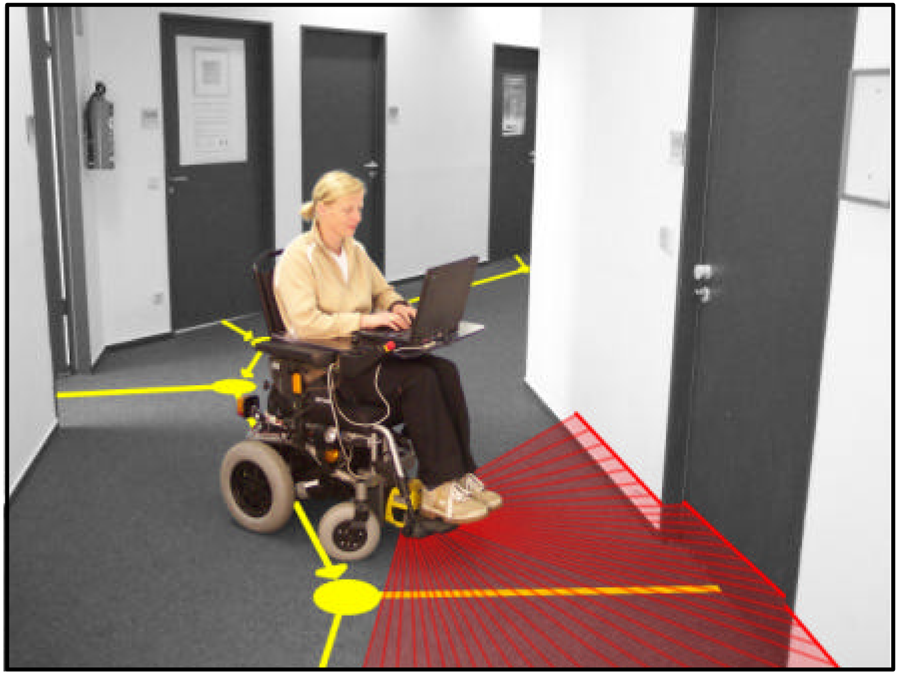
Christian Mandel A1[RoboMap], 09/17/04

Overview

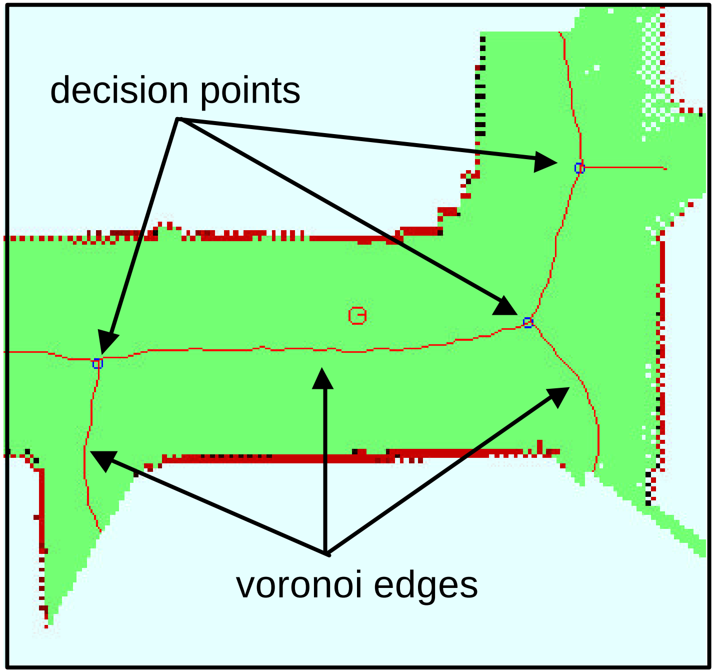
- Rolland: new hardware platform
- Motivation
- Basic principles of the **D**ynamic **W**indow **A**pproach
- DWA & the problem with non circular shaped robots
- Implementation issues: Curve Segments Table, Collision Table
- Computing the Trajectory
- Computing the Velocity Profile
- What remains to do
- Preliminary Experiments

- Rolland: hardware platform
  - Meyra Champ wheelchair
  - omnidirectional camera system
  - controlling laptop connected via single usb cable
  - emergency stop button
  - 2 laser range finder
  - 2 incremental encoders

Motivation



Wheelchair in its environment

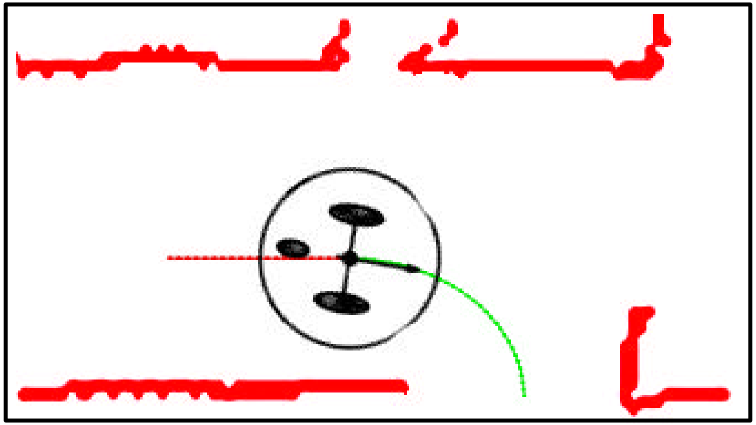


Metrical & topological representation.

How to navigate between decision points while taking care of dynamic obstacles?

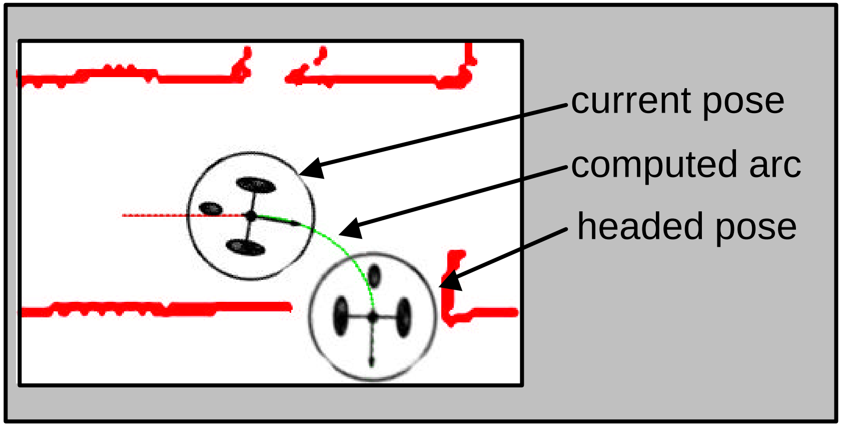
Basic principles of the **D**ynamic **W**indow **A**pproach[1]

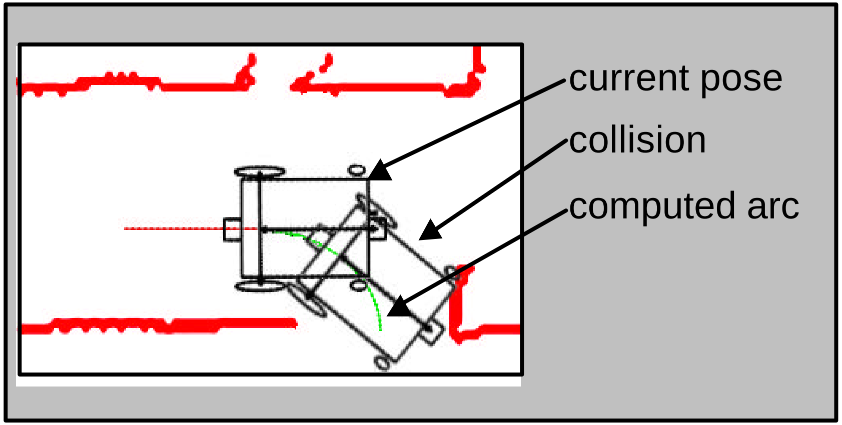
- Local navigation combined with reactive collision avoidance.
- **DWA** assumes: Robot velocity is a piecewise constant function in time.
- **DWA** considers: Robot has initial velocity and limited accelerations.
- **DWA** computes optimal circular arc in every time step.
- **DWA** looks one curve ahead.

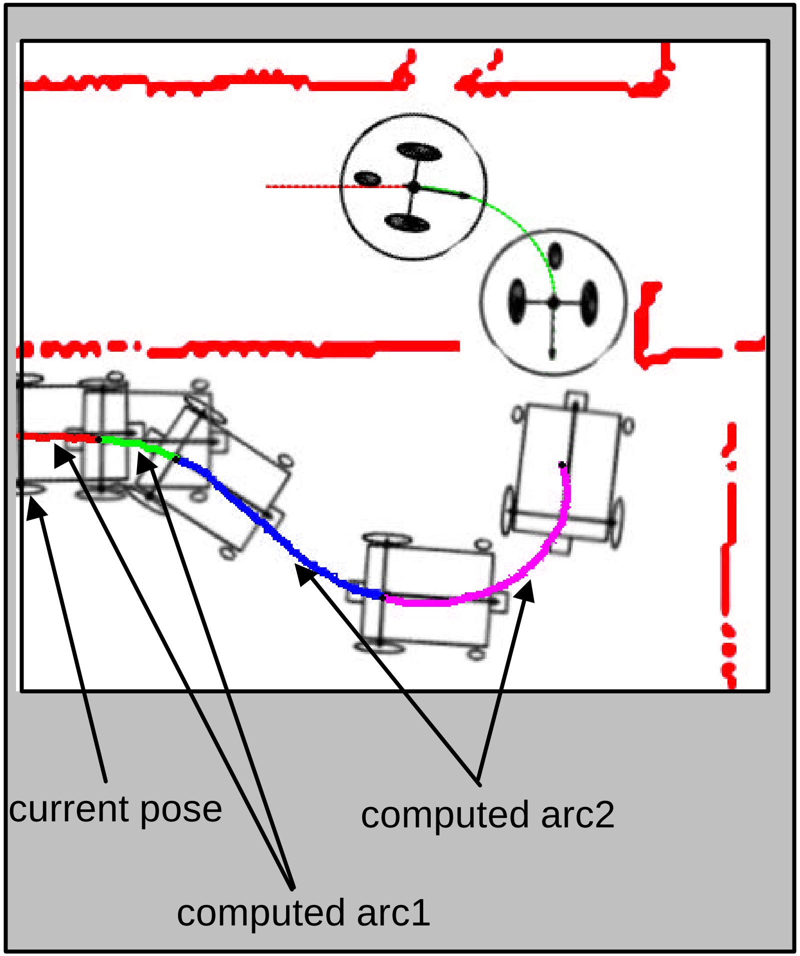


[1] [Fox, Burgard, Thrun] „The Dynamic Window Approach To Collision Avoidance“

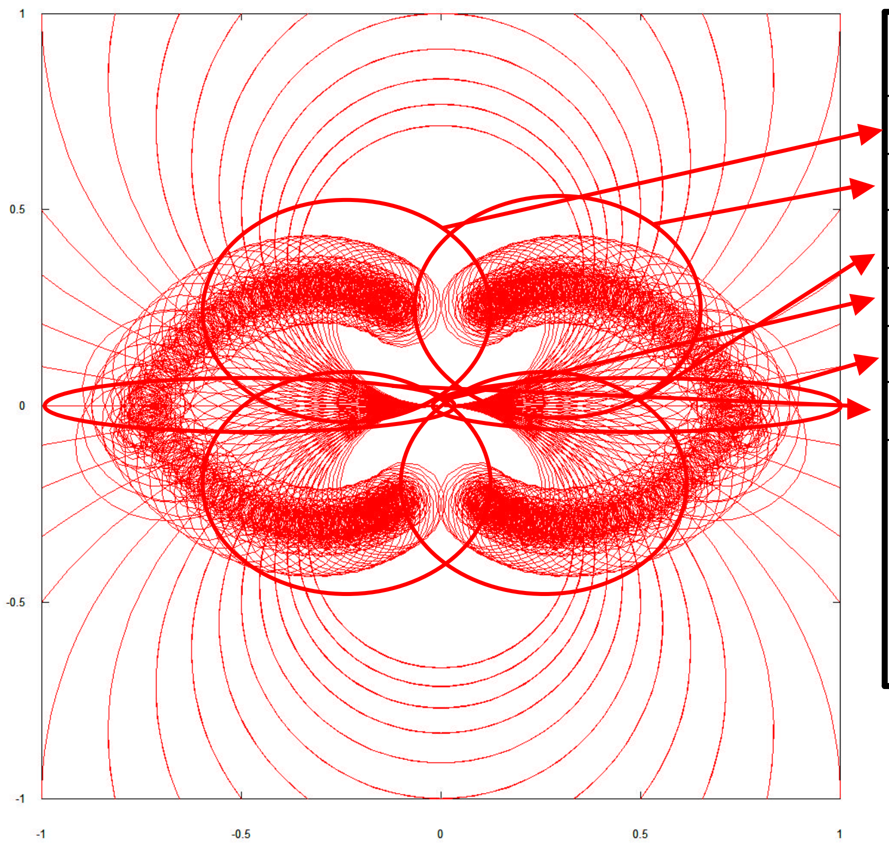
DWA & the problem with non circular shaped robots







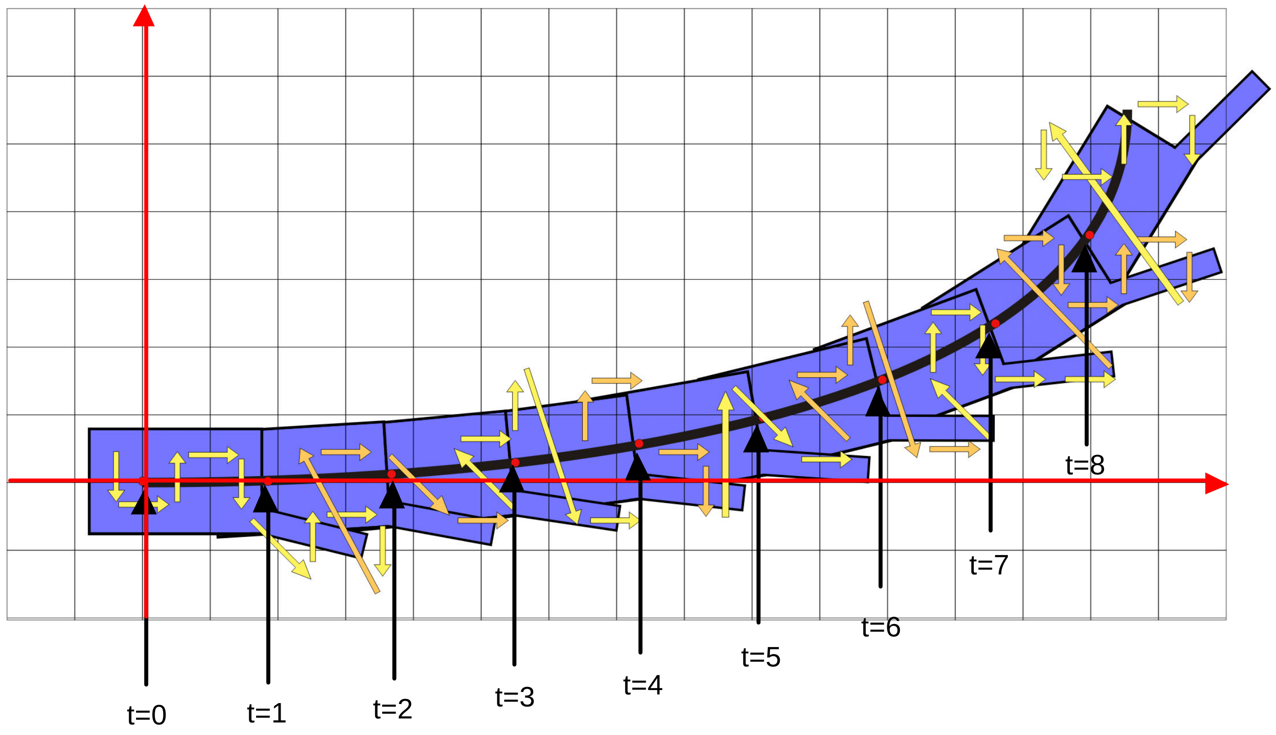
Curve Segments Table



start curvature	curvature prime	direction	poses
0	+ MAX	backwards	$(x_0, y_0, q_0), \ldots$
0	- MAX	forwards	$(x_0, y_0, q_0), \ldots$
0	+ MAX	forwards	$(x_0, y_0, q_0), \ldots$
0	- MAX	backwards	$(x_0, y_0, q_0), \ldots$
0	0	forwards	$(x_0, y_0, q_0), \ldots$
0	0	backwards	$(x_0, y_0, q_0), \ldots$
...	...	...	...

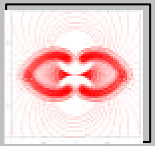
Collision Table

arclength (t)	0	...
offset between occupied cells ({$(x_1,y_1),\ldots,(x_n,y_n)$})	{(0,-1),(1,0),(0,1),(1,0),(0,-1),(1,-1), (0,1),(1,0),(0,-1)}	...

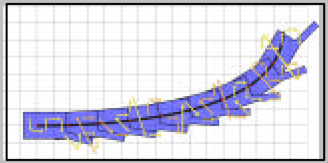


Algorithmic Refinements 1

Goal:
reduce Size of



Curve **S**egments **T**able



Collision **T**able

Precompute and store only paths whose first pose has zero heading.

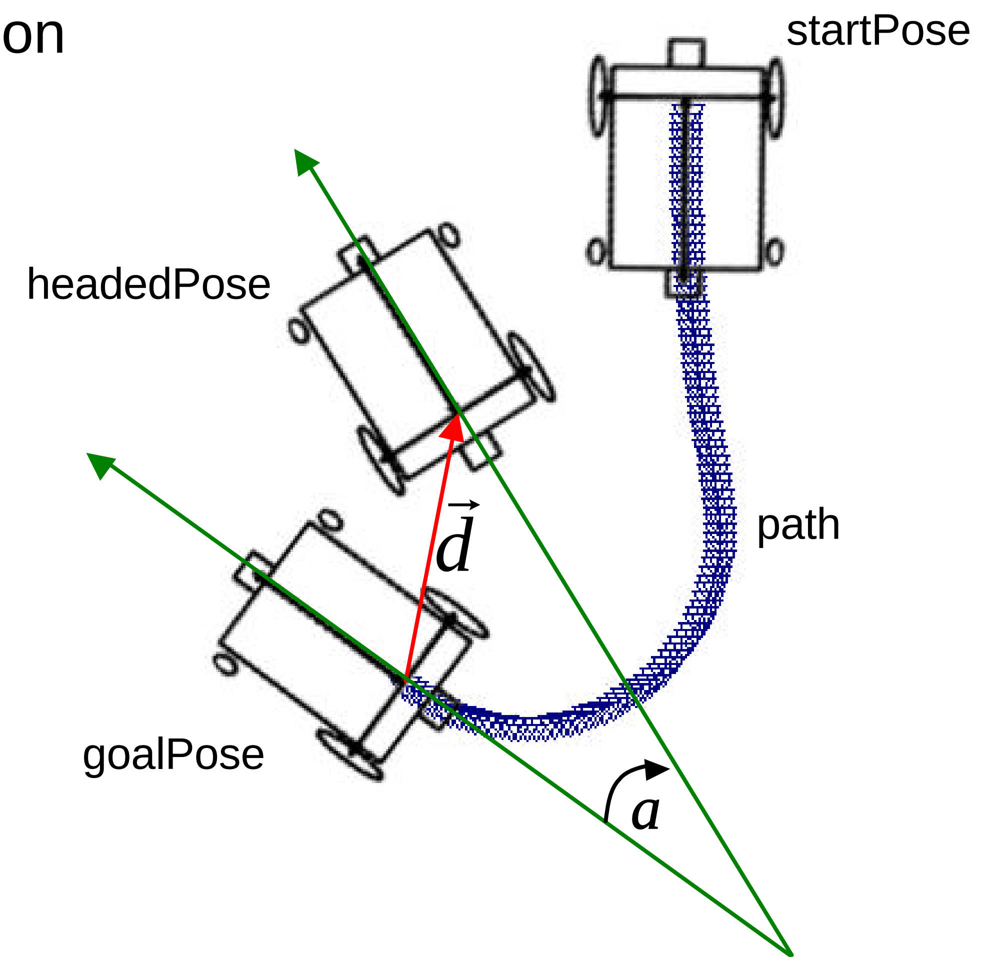
↓

Test-for-Collision-Operation has to rotate CT-entries.

↓

Precompute additional table which holds rotated offsets between occupied cells.

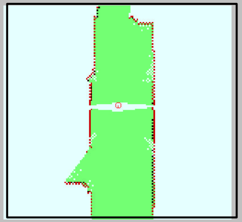
Score Function



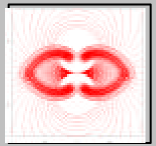
$$score(path, startPose, headedPose) = \left|\vec{d}\right| + c_1 * a + c_2 * path.length$$

Computing the Optimal Path

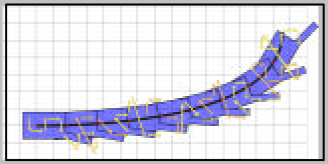
Input



Local **E**vidence **G**rid,
Odometry Pose,
Goal Pose

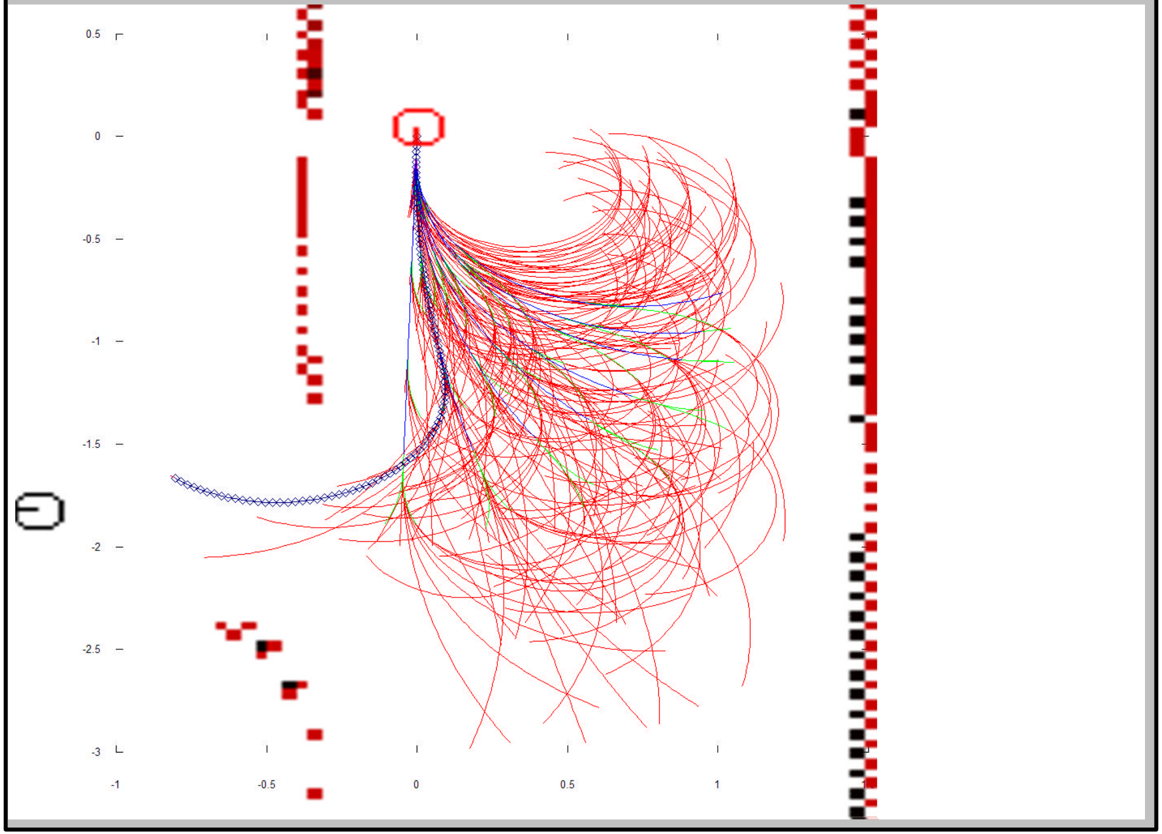


Curve **S**egments **T**able



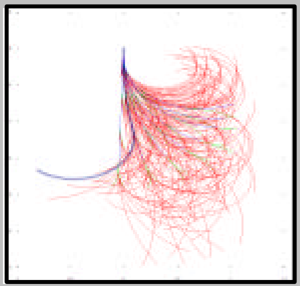
Collision **T**able

Algorithm (basic idea)



Algorithmic Refinements 2

Goal:
reduce complexity of computation



Computing the optimal path w.r.t. objective function

→

Algorithm (with constant arc2.length)

set *arc2.length* = *MAX_arc2.length*

∀ *arc1.curvature*, *arc1.length*, *arc1.direction* do

∀ *arc2.curvature*, *arc2.direction* do

- construct *path*
- test for collision and prune if necessary
- minimise *path.score*
- if (*path.score* < *bestPath.score*)
  set *bestPath* = *path*

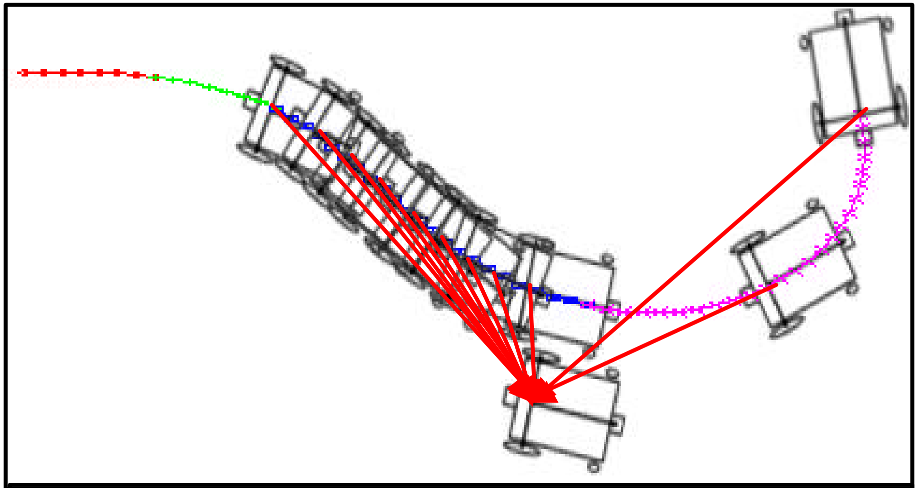
Algorithmic Refinements 3

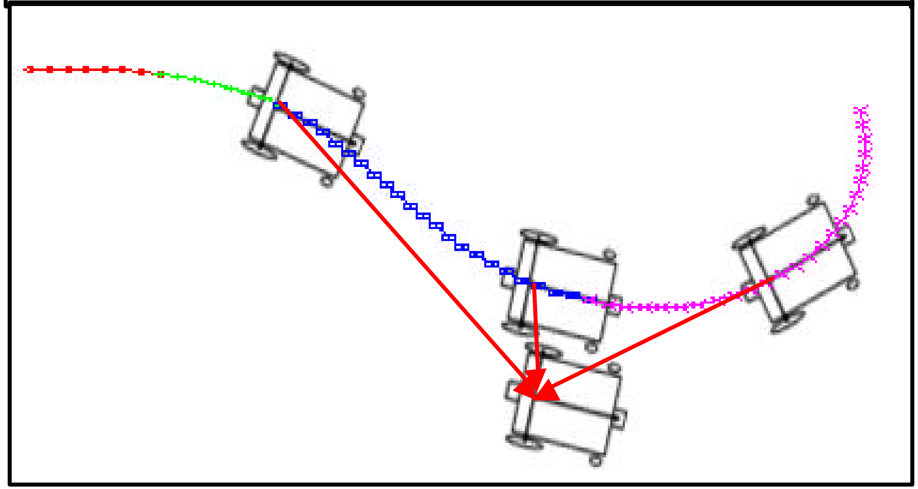
Algorithm
(witch constant
arc2.length)

minimise
path.score

∀ potential *goalPose*
∈ *arc2* do

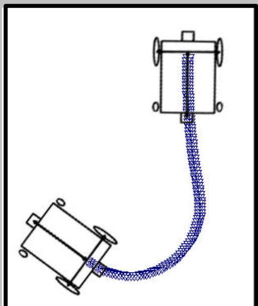
- calc *score*
- store *goalPose* with minimal *score*





Computing the Velocity Profile

Input



Solution Path
$path = \{pose_0, \dots, pose_n\}$

Velocities in Start & Goal
v_{start}, v_{goal}

Lateral Acceleration Limit
$a_{\max}$

Longitudinal Acceleration Limit
$v'_{\max}$

Rotational Velocity Limit
$w_{\max}$

Longitudinal Velocity Limit
$v_{\max}$

Algorithm

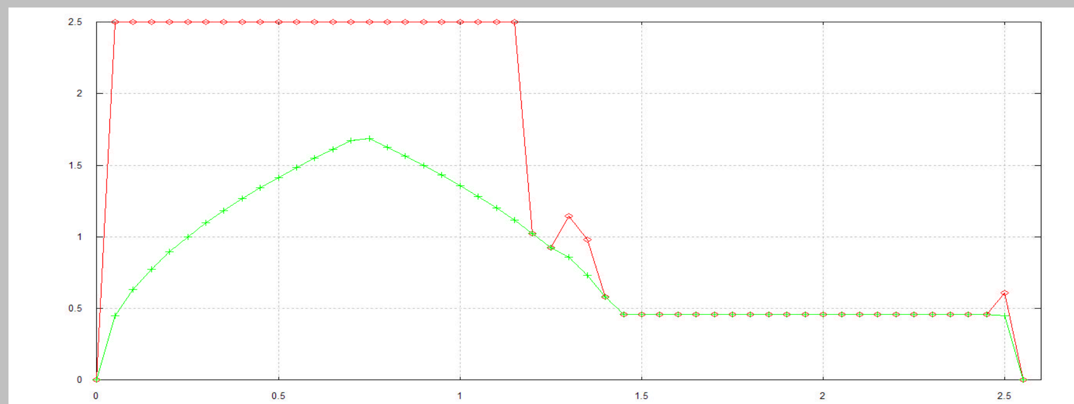
FOR EVERY $pose_i \in path$ DO:

COMPUTE MAXIMUM velocity v_i FOR WHICH HOLDS:

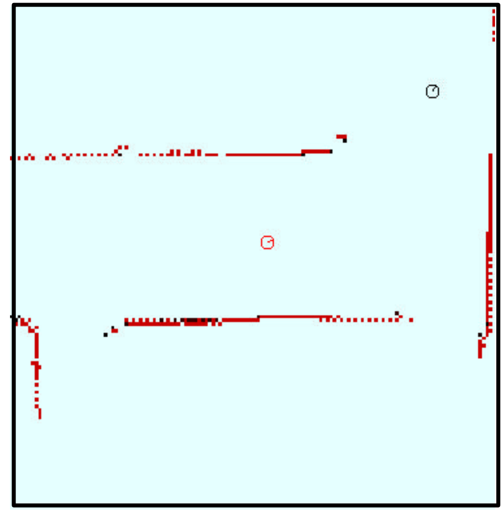
$$|v_i| = \min\left(v_{\max}, \sqrt{\frac{a_{\max}}{pose_i.c}}, \frac{w}{pose_i.c}\right)$$

FOR EVERY v_i DO:

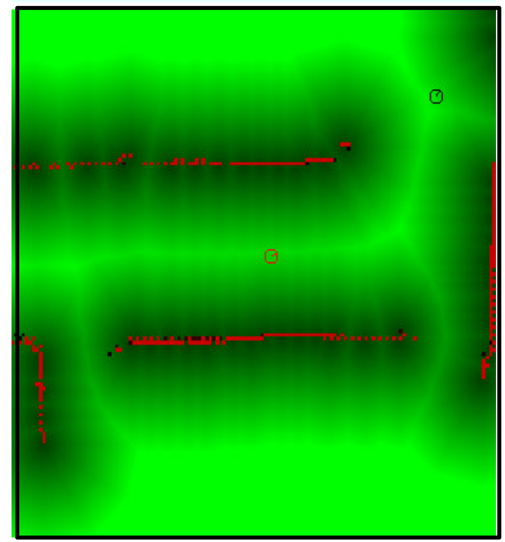
INCORPORATE LONGITUDINAL ACCELERATION LIMIT v'_i:



What remains to do



- current implementation considers only binary information from the evidence grid while doing the collision test



- better: collision test should also return minimal distance to obstacles for tested path

$$score(path, startPose, headedPose) =$$

$$|\vec{d}| + c_1 * a + c_2 * path.length + c_3 * distToObstacles$$

Preliminary Experiments

SFB/TR 8 SPATIAL COGNITION