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 Dynamic Window Approach
- 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 **Dynamic Window Approach**\(^1\)

- 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

- current pose
- computed arc
- headed pose

- current pose
- collision
- computed arc

- current pose
- computed arc1
- computed arc2
Curve Segments Table

<table>
<thead>
<tr>
<th>start curvature</th>
<th>curvature prime</th>
<th>direction</th>
<th>poses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>+ MAX</td>
<td>backwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
<tr>
<td>0</td>
<td>- MAX</td>
<td>forwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
<tr>
<td>0</td>
<td>+ MAX</td>
<td>forwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
<tr>
<td>0</td>
<td>- MAX</td>
<td>backwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>forwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>backwards</td>
<td>((x_0, y_0, \theta_0))....</td>
</tr>
</tbody>
</table>
Collision Table

<table>
<thead>
<tr>
<th>arclength (t)</th>
<th>0</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>offset between occupied cells</td>
<td>{(0,-1),(1,0),(0,1),(1,0),(0,-1),(1,-1), (0,1),(1,0),(0,-1)}</td>
<td>...</td>
</tr>
</tbody>
</table>
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

\[ score(path, startPose, headedPose) = \mid \vec{d} \mid + c_1 \cdot \alpha + c_2 \cdot path.length \]
Computing the Optimal Path

**Input**
- Local Evidence Grid
- Odometry Pose
- Goal Pose
- Curve Segments Table
- Collision Table

**Algorithm (basic idea)**

![Diagram showing the optimal path computation process](image-url)
Algorithmic Refinements 2

Goal: reduce complexity of computation

Algorithm (with constant arc2.length)

set \( arc2.length = MAX_{arc2.length} \)

\( \forall \ arc1.curvature, \ arc1.length, \ arc1.direction \) do

\( \forall \ arc2.curvature, \ arc2.direction \) 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 (with constant \( arc2.length \))

<table>
<thead>
<tr>
<th>minimise ( path.score )</th>
</tr>
</thead>
</table>

\[ \forall \text{ potential } goalPose \in \text{arc2} \text{ do} \]

- calc score
- store \( goalPose \) with minimal score
An Extension to the Dynamic Window Approach for arbitrarily shaped Robots.

Computing the Velocity Profile

Input

- Solution Path
  \( path = \{ pose_0, ..., pose_n \} \)
- Velocities in Start & Goal
  \( v_{\text{start}}, v_{\text{goal}} \)
- Lateral Acceleration Limit
  \( a_{\text{max}} \)
- Longitudinal Acceleration Limit
  \( v_{\text{max}} \)
- Rotational Velocity Limit
  \( \omega_{\text{max}} \)
- Longitudinal Velocity Limit
  \( v_{\text{max}} \)

Algorithm

FOR EVERY \( pose_i \in path \) DO:

COMPUTE MAXIMUM velocity \( v_i \) FOR WHICH HOLDS:

\[
|v_i| = \min \left( v_{\text{max}}, \sqrt{\frac{a_{\text{max}}}{\text{pose}_i.c}}, \frac{\omega}{\text{pose}_i.c} \right)
\]

FOR EVERY \( v_i \) DO:

INCORPORATE LONGITUDINAL ACCELERATION LIMIT \( v_i' \):

![Graph showing velocity profile](image-url)
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

\[
\text{score}(\text{path, startPose, headedPose}) = \left| \bar{d} \right| + c_1 \alpha + c_2 \text{path.length} + c_3 \text{distToObstacles}
\]
Preliminary Experiments