2024-1 개별연구 <다중 로봇 관제 시스템 개발>

Nav2 분석:
DWA Algorithm을 중심으로

박영하 (실제 로봇 구동)

Contents

D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.

1

Overview of Nav2

1. Overview of Nav2 (Components)

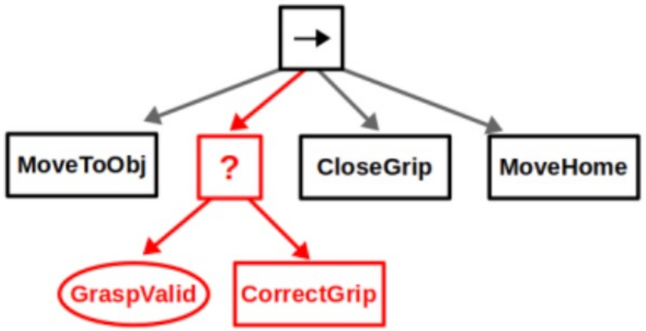
- Nav2 is a ROS2-based framework for robot navigation that provides capabilities for localization, mapping, path planning, and obstacle avoidance.

Global Planner	Local Planner	Behavior Tree (BT)
Lifecycle Manager	Costmap_2D	Sensor Interfaces
Map Server	Localization (AMCL)	Controller Server

1. Overview of Nav2 (Components)

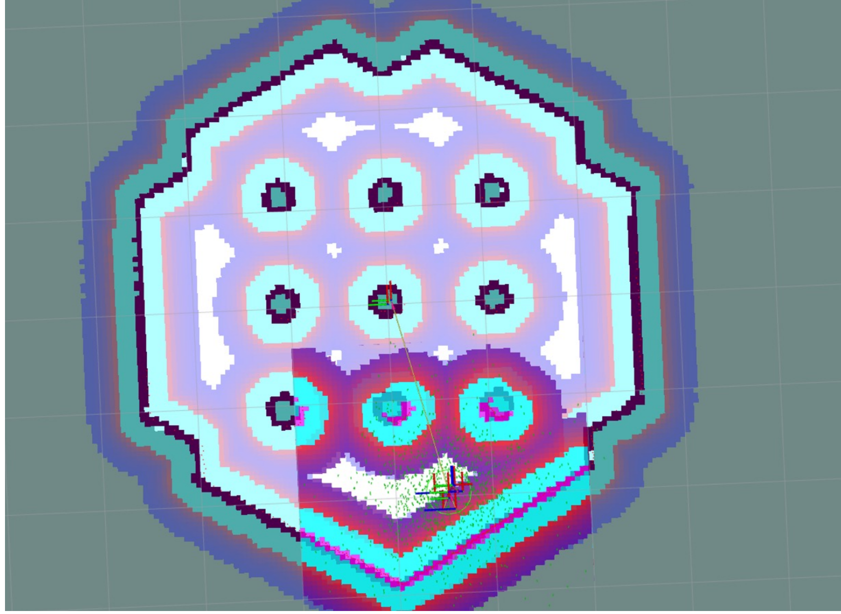
- **Global Planner**
  - Calculates a path from the robot's current position to the goal position considering the global map.
  - Includes various algorithms such as Dijkstra's, A*, and others.
- **Local Planner**
  - Computes the robot's movements in real-time to follow the global path while avoiding obstacles.
  - Common plugins include DWA (Dynamic Window Approach) and TEB (Timed Elastic Band).

1. Overview of Nav2 (Components)



- Behavior Tree (BT)
  - Manages the overall navigation behavior using behavior trees for flexible and modular decision-making.
  - Nodes for actions like navigating, recovering, and handling failures.
- Lifecycle Manager
  - Manages the lifecycle states of Nav2 nodes, ensuring they follow a defined state machine for robustness and reliability.

1. Overview of Nav2 (Components)



- **Costmap_2D**
  - Represents the environment as a 2D grid for obstacle avoidance and path planning.
  - Global costmap for the entire map, and local costmap for the robot's immediate vicinity.
- **Sensor Interfaces**
  - Integrates various sensors (e.g., LIDAR, cameras) to update the costmaps and assist in localization and obstacle detection.

1. Overview of Nav2 (Components)

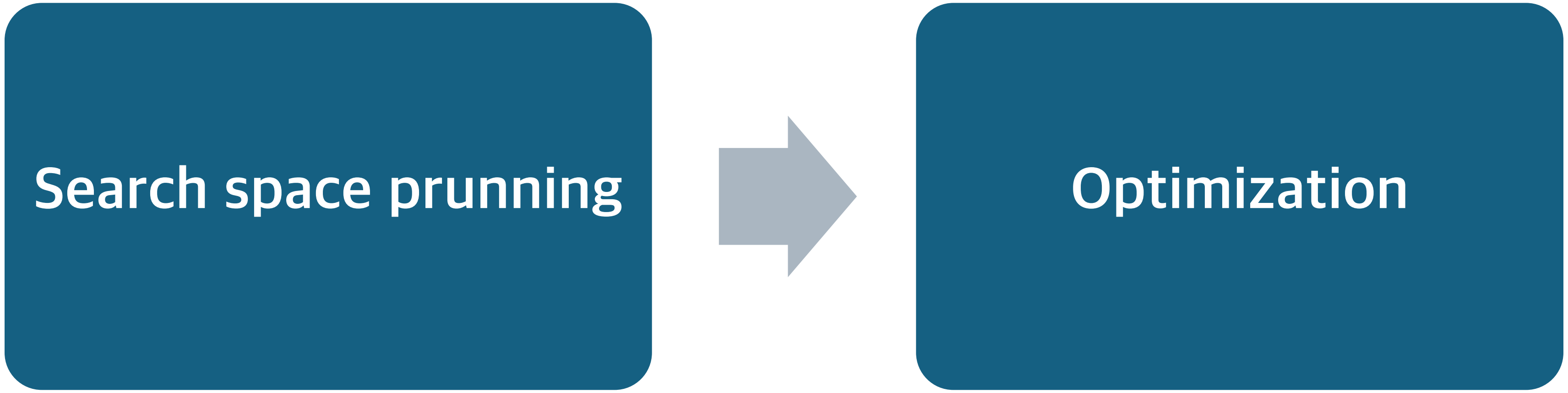
- **Map Server**
  - Loads and provides the global map to other components.
- **Localization (AMCL)**
  - Uses Adaptive Monte Carlo Localization (AMCL) to estimate the robot's pose on the map.
- **Controller Server**
  - Hosts the local planner and controls the robot's movements to follow the path generated by the global planner.

2

“The dynamic window approach to collision avoidance”

Dynamic Window Approach

- Local trajectory planning algorithm used for real-time obstacle avoidance and navigation.



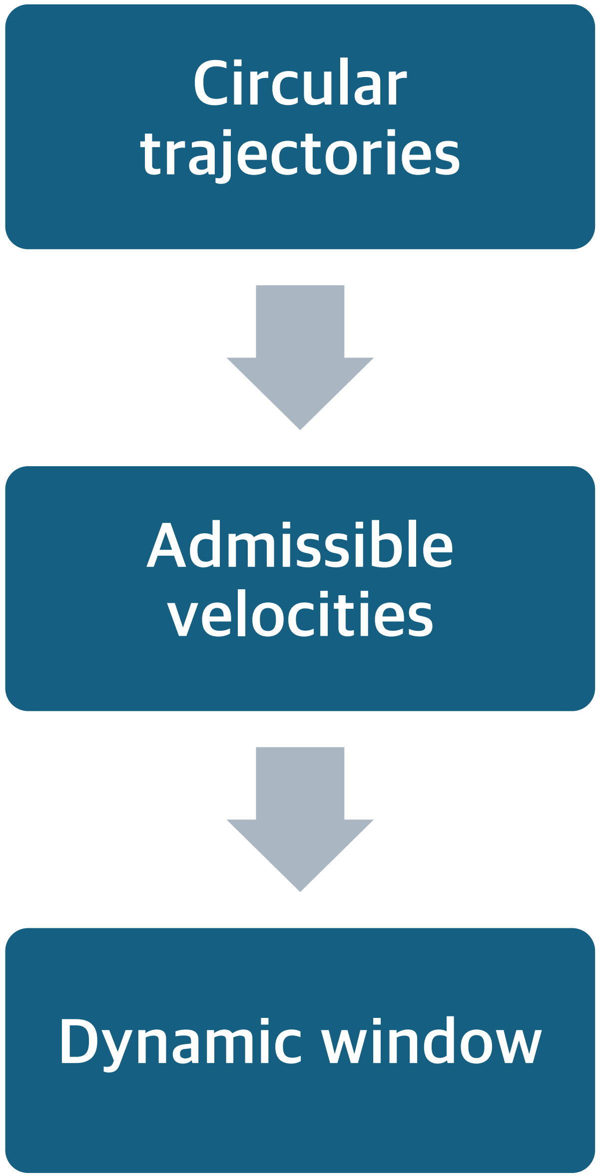
1. Search space prunning

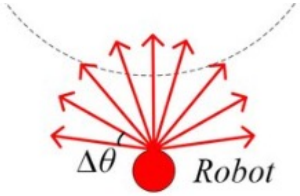
1. **Search space:** The search space of the possible velocities is reduced in three steps:

   (a) **Circular trajectories:** The dynamic window approach considers only circular trajectories (curvatures) uniquely determined by pairs (v,ω) of translational and rotational velocities. This results in a two-dimensional velocity search space.

   (b) **Admissible velocities:** The restriction to admissible velocities ensures that only safe trajectories are considered. A pair (v,ω) is considered admissible, if the robot is able to stop before it reaches the closest obstacle on the corresponding curvature.

   (c) **Dynamic window:** The dynamic window restricts the admissible velocities to those that can be reached within a short time interval given the limited accelerations of the robot.

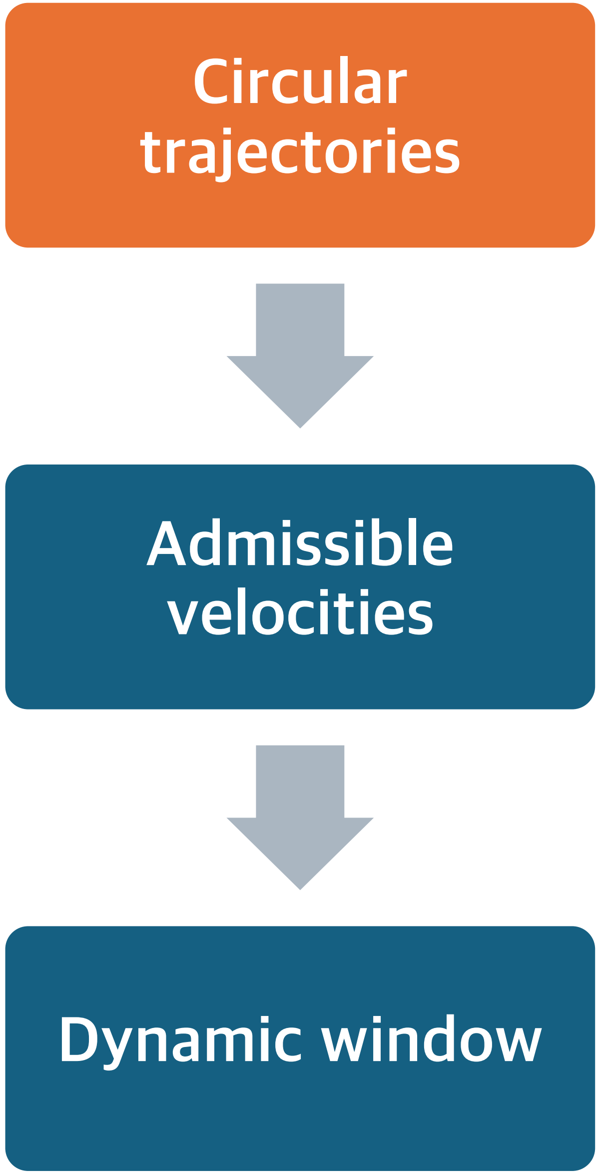




v_i: translational velocities
w_i: rotational velocities

- Each curvature is uniquely determined by the velocity vector (v_i, w_i)
- To generate a trajectory to a given goal point for the next n time intervals the robot has to determine velocities (v_i, w_i), one for each of the n intervals between t_0 and t_n.
  - Premise: the resulting trajectory does not intersect with an obstacle.
- The search space for these vectors is exponential.

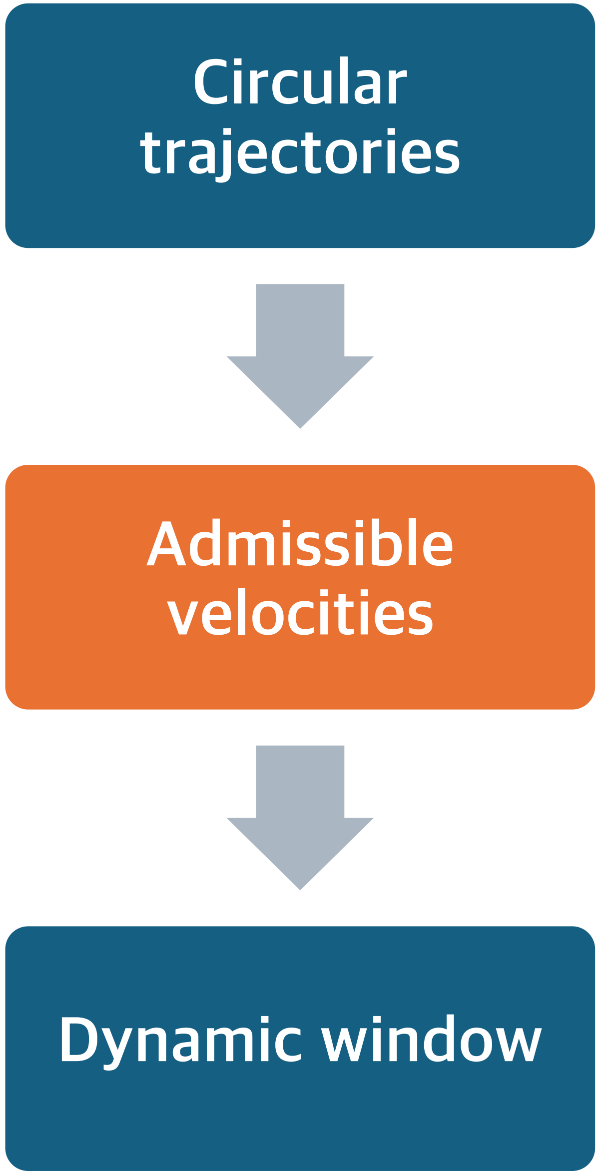
- Considers exclusively the first time interval, and assumes that the velocities in the remaining n – 1 time intervals are constant.
  - (a) the reduced search space is two-dimensional and thus tractable.
  - (b) the search is repeated after each time interval.
  - (c) the velocities will automatically stay constant if no new commands are given.



- Obstacles in the closer environment of the robot impose restrictions on the rotational and translational velocities.
- For example, the maximal admissible speed on a curvature depends on the distance to the next obstacle on this curvature.
- A velocity is considered admissible, if the robot is able to stop before it reaches this obstacle.

$$V_a = \left\{ v, \omega \,\middle|\, v \leq \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{v}_b} \wedge \omega \leq \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{\omega}_b} \right\}.$$

- V_a: the set of velocities (v, w) which allow the robot to stop without colliding with an obstacle.



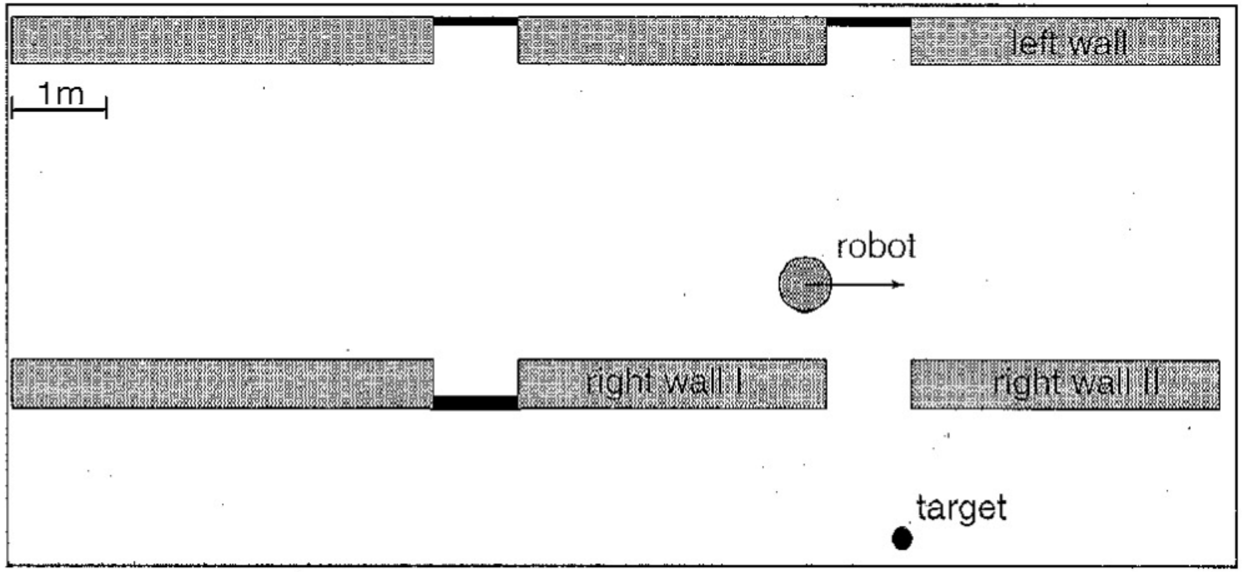


Figure 2. Example Situation.

dark shaded area: non-admissible velocities

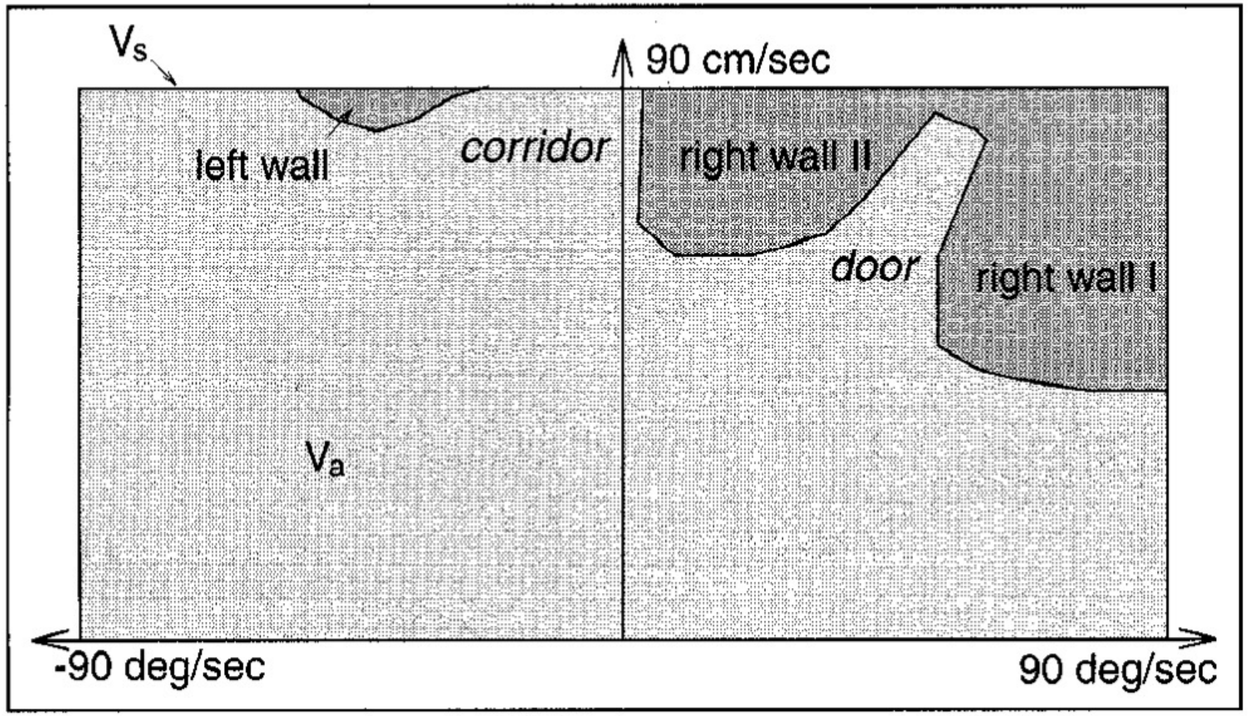
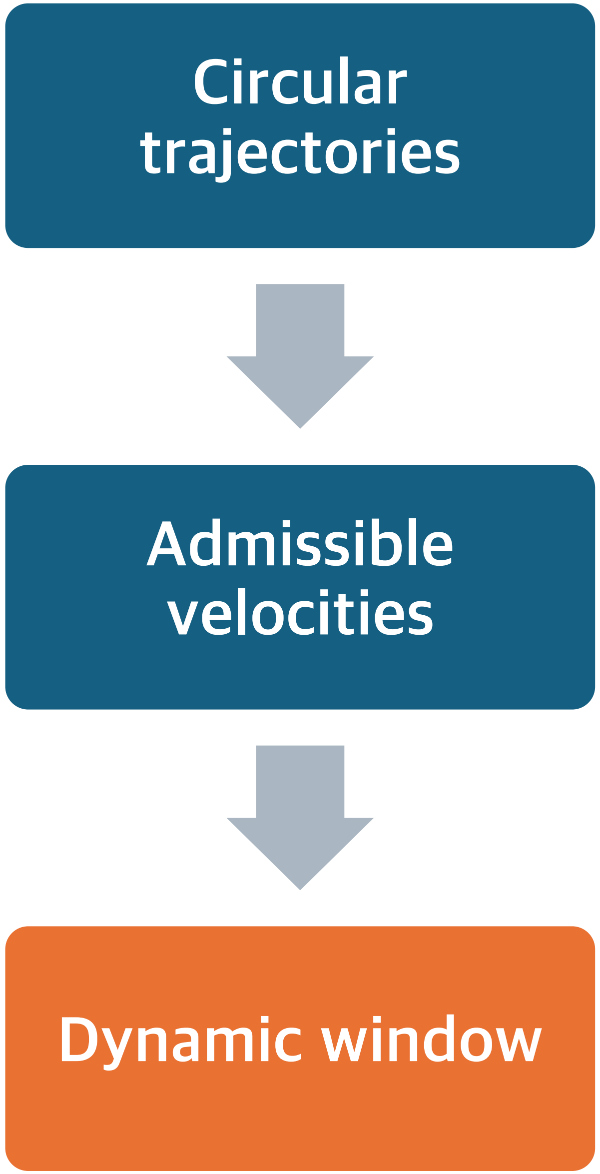


Figure 4. Velocity Space.

- The overall search space is reduced to the dynamic window which contains only the velocities that can be reached within the next time interval.

$$V_d = \left\{(v,\omega) \mid v \in \left[v_a - \dot{v}\cdot t, v_a + \dot{v}\cdot t\right] \wedge \omega \in \left[\omega_a - \dot{\omega}\cdot t, \omega_a + \dot{\omega}\cdot t\right]\right\}. \tag{15}$$



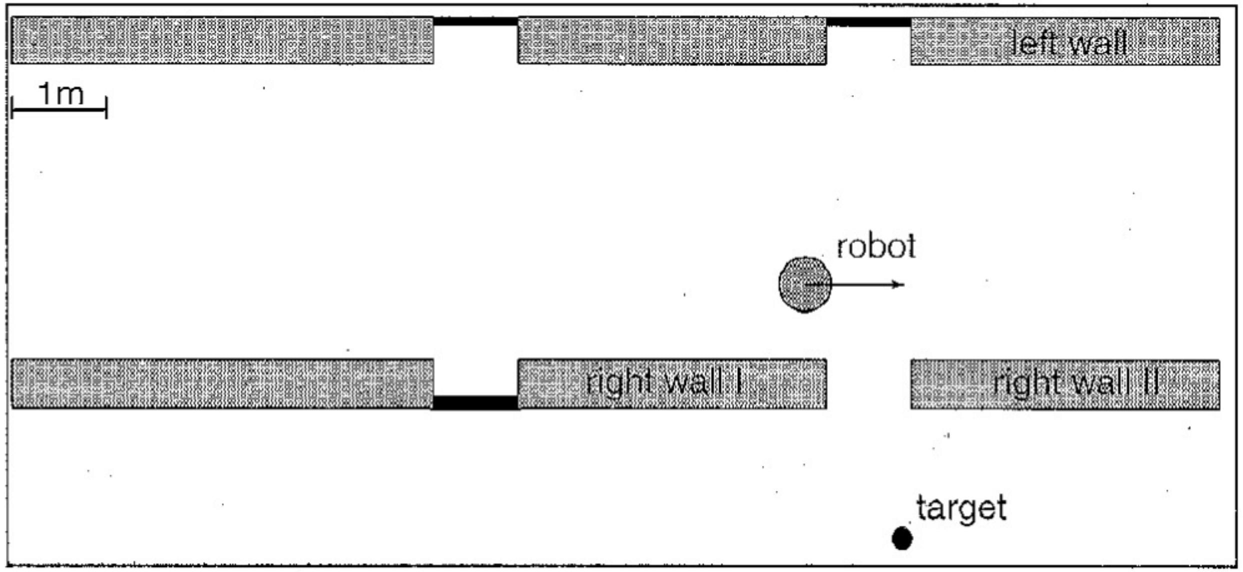


Figure 2. Example Situation.

- The dynamic window is centered around the actual velocity and the extensions of it depend on the accelerations that can be exerted.
- All curvatures outside the dynamic window cannot be reached within the next time interval and thus are not considered for the obstacle avoidance.

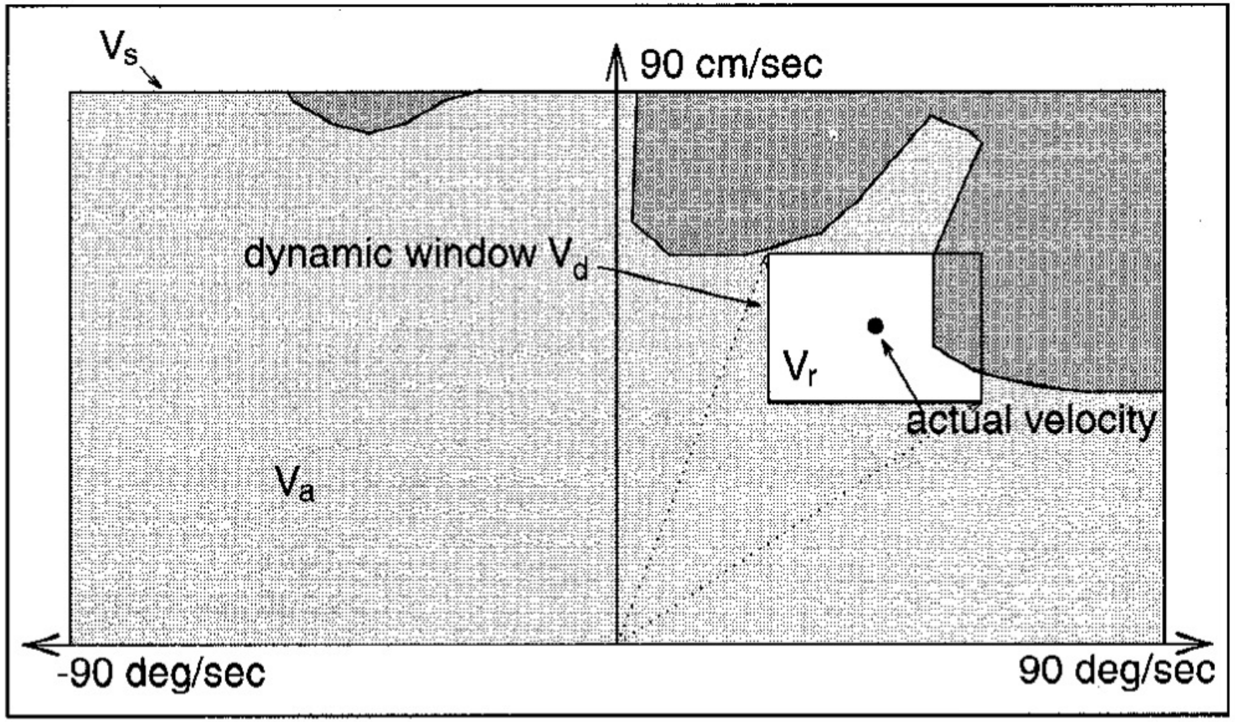
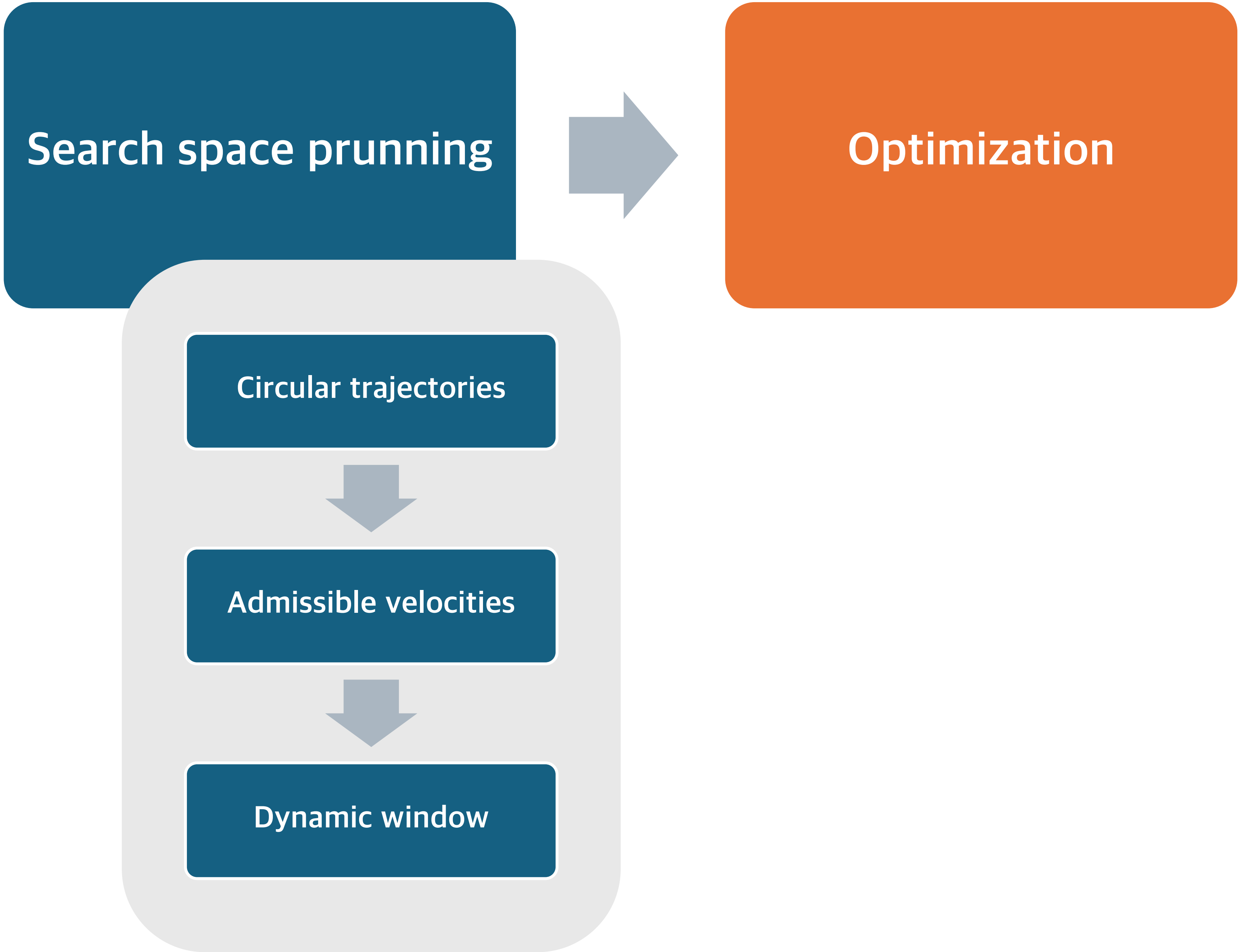


Figure 5. Dynamic Window.


Search space prunning
Optimization
Circular trajectories
Admissible velocities
Dynamic window

2. Optimization

2. **Optimization:** The objective function

$$G(v,\omega) = \sigma(\alpha \cdot \mathrm{heading}(v,\omega) + \beta \cdot \mathrm{dist}(v,\omega) + \gamma \cdot \mathrm{vel}(v,\omega)) \tag{13}$$

is maximized. With respect to the current position and orientation of the robot this function trades off the following aspects:
(a) **Target heading:** *heading* is a measure of progress towards the goal location. It is maximal if the robot moves directly towards the target.
(b) **Clearance:** *dist* is the distance to the closest obstacle on the trajectory. The smaller the distance to an obstacle the higher is the robot's desire to move around it.
(c) **Velocity:** *vel* is the forward velocity of the robot and supports fast movements.
The function σ smoothes the weighted sum of the three components and results in more side-clearance from obstacles.

Target heading

Clearance

Velocity

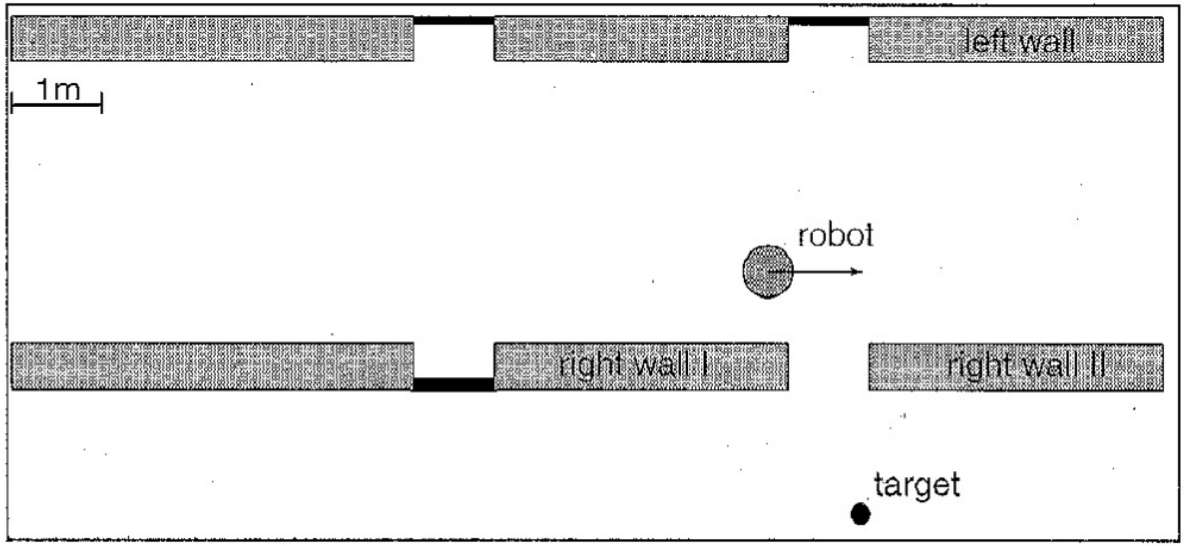


Figure 2. Example Situation.

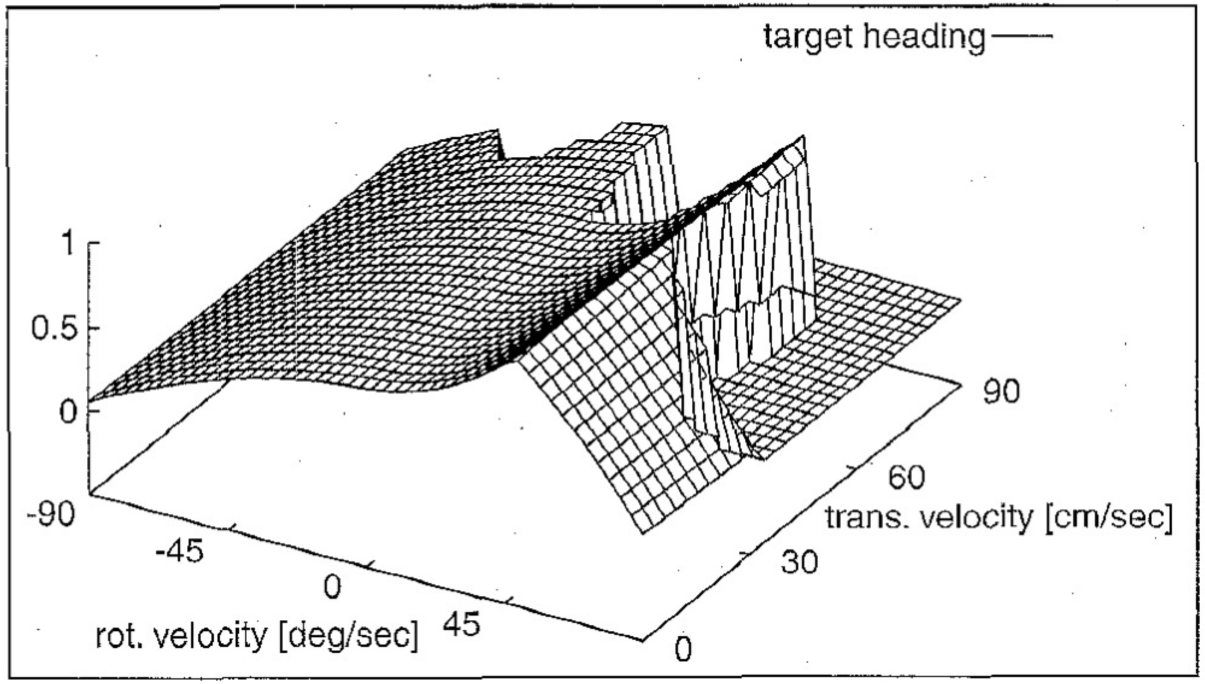


Figure 7. Evaluation of the Target Heading.

- The target heading **heading(v, w)** measures the alignment of the robot with the target direction.

Target heading

Clearance

Velocity

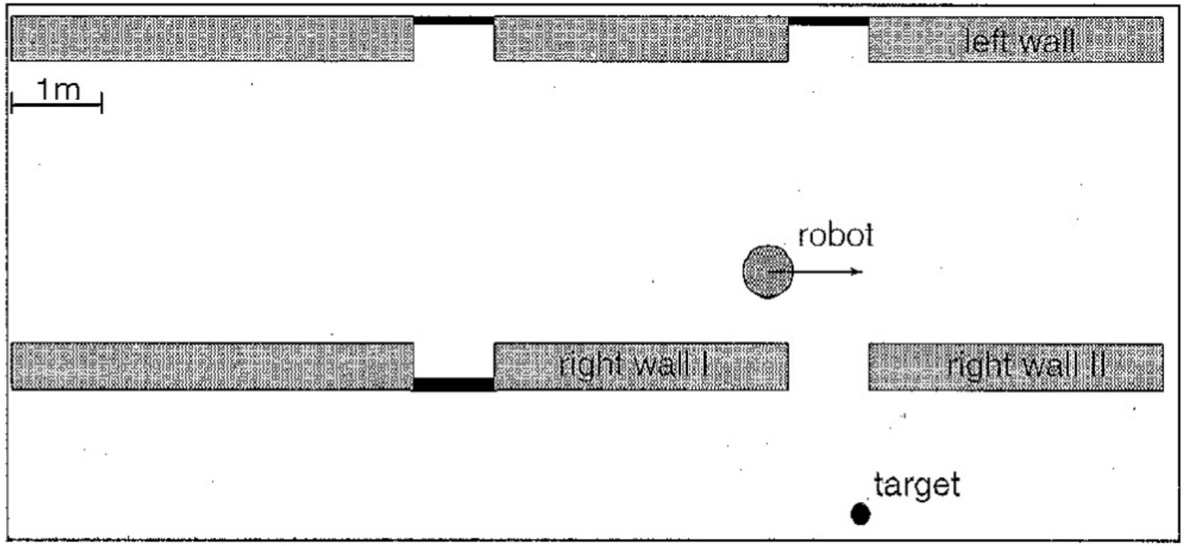


Figure 2. Example Situation.

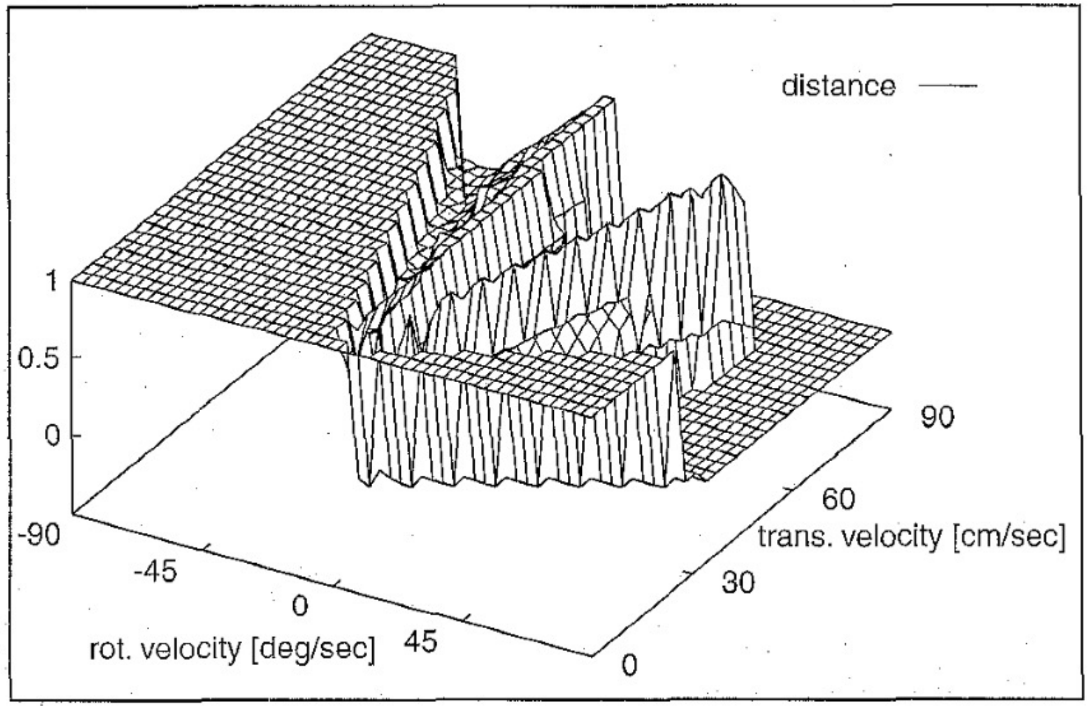


Figure 8. Evaluation of the Distances.

- The function **dist(v, w)** represents the distance to the closest obstacle that intersects with the curvature.

Target heading

Clearance

Velocity

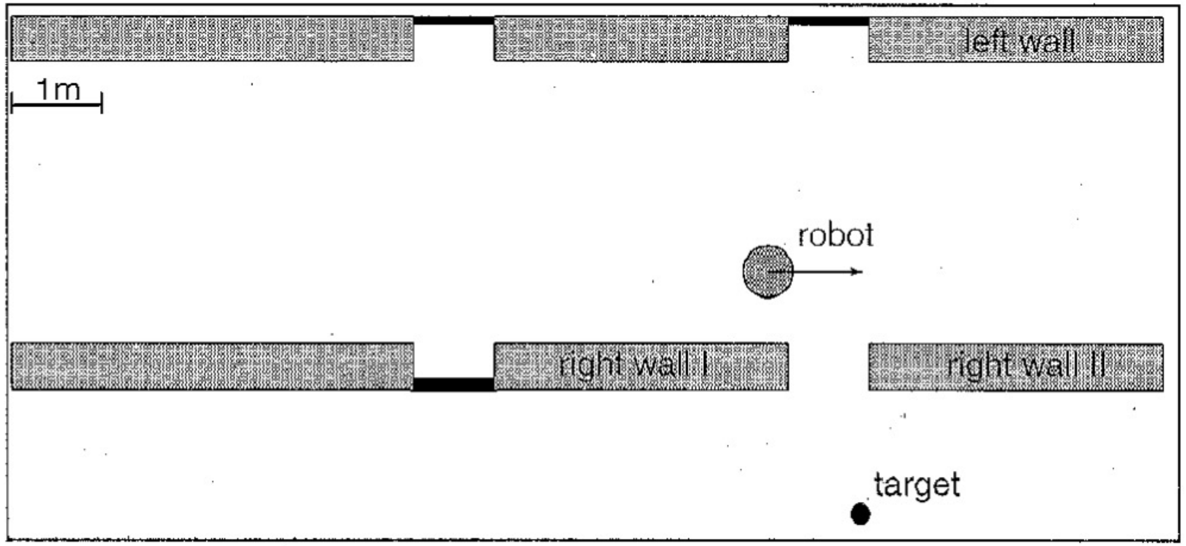


Figure 2. Example Situation.

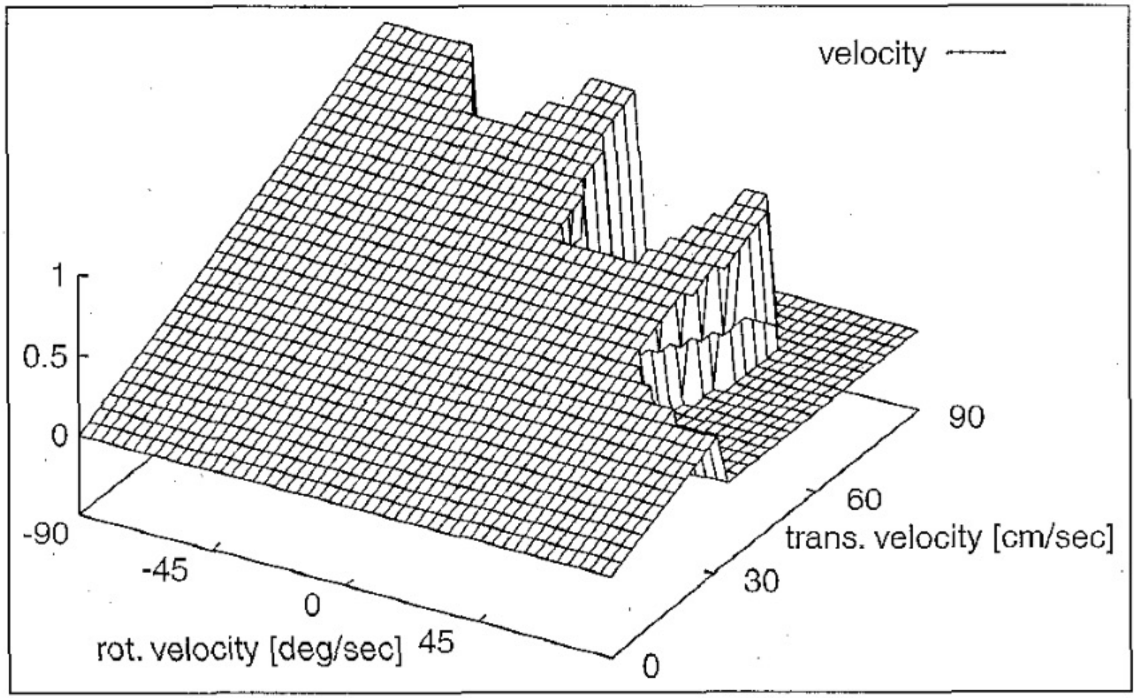


Figure 9. Evaluation of the Velocities.

- The function **velocity(v, w)** is used to evaluate the progress of the robot on the corresponding trajectory.

Target heading

Clearance

Velocity

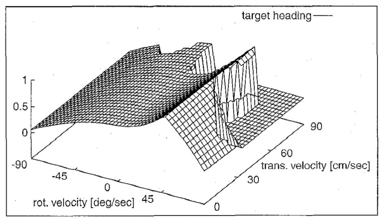


Figure 7. Evaluation of the Target Heading.

Target heading

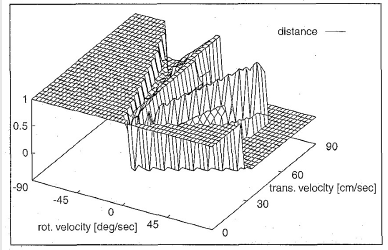


Figure 8. Evaluation of the Distances.

Clearance

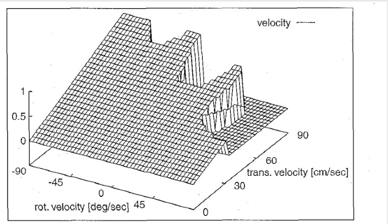


Figure 9. Evaluation of the Velocities.

Velocity

$$G(v,\omega) = \boxed{\sigma}(\alpha \cdot \mathrm{heading}(v,\omega) + \beta \cdot \mathrm{dist}(v,\omega) + \gamma \cdot \mathrm{vel}(v,\omega))$$

$\boldsymbol{\alpha = 2.0}$
$\boldsymbol{\beta = 0.2}$
$\boldsymbol{\gamma = 0.2}$

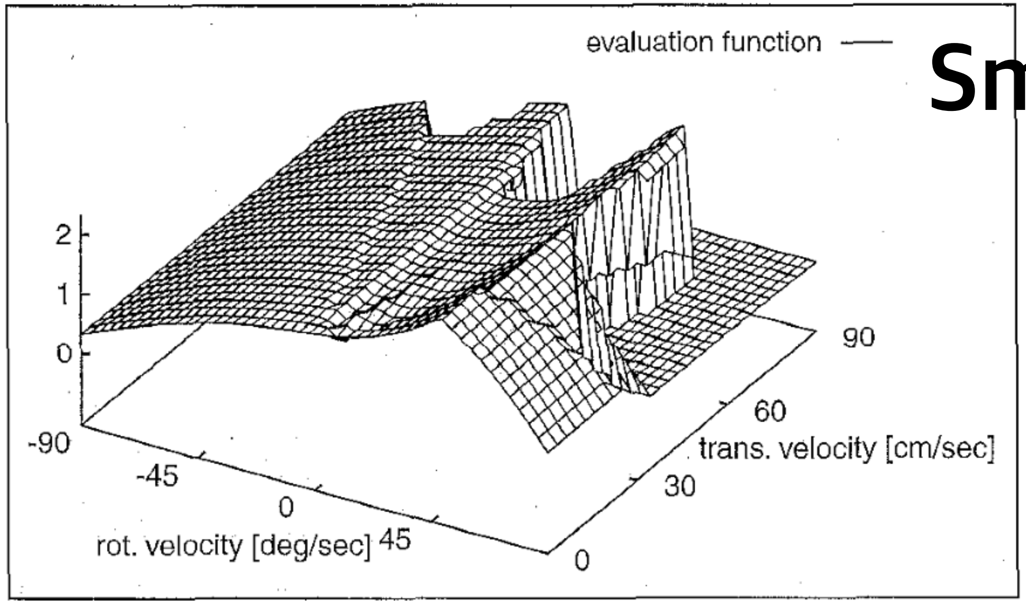


Figure 10. Combined Evaluation Function.

Smoothing

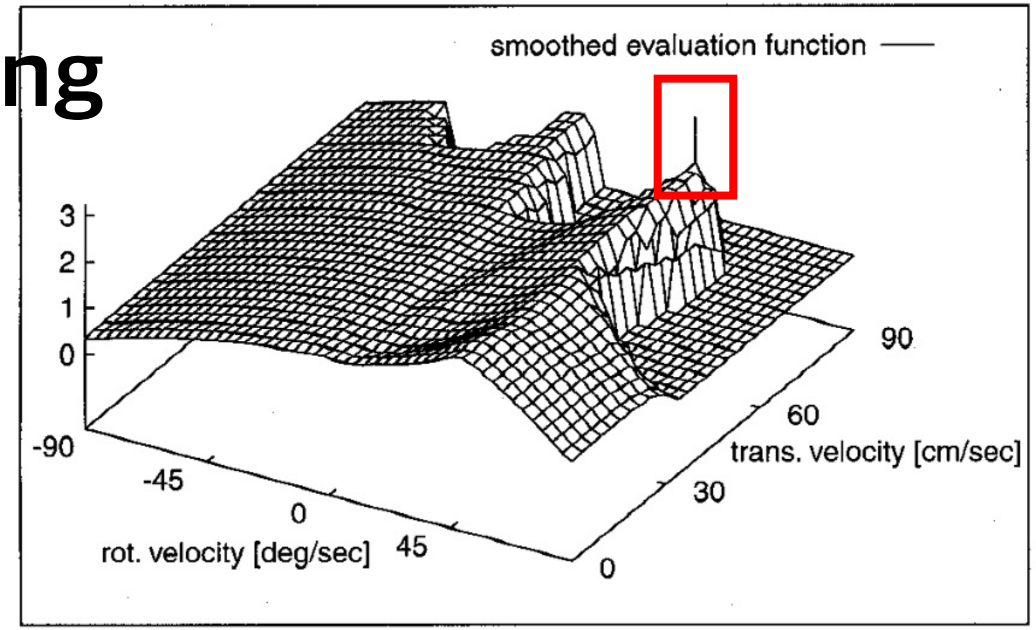
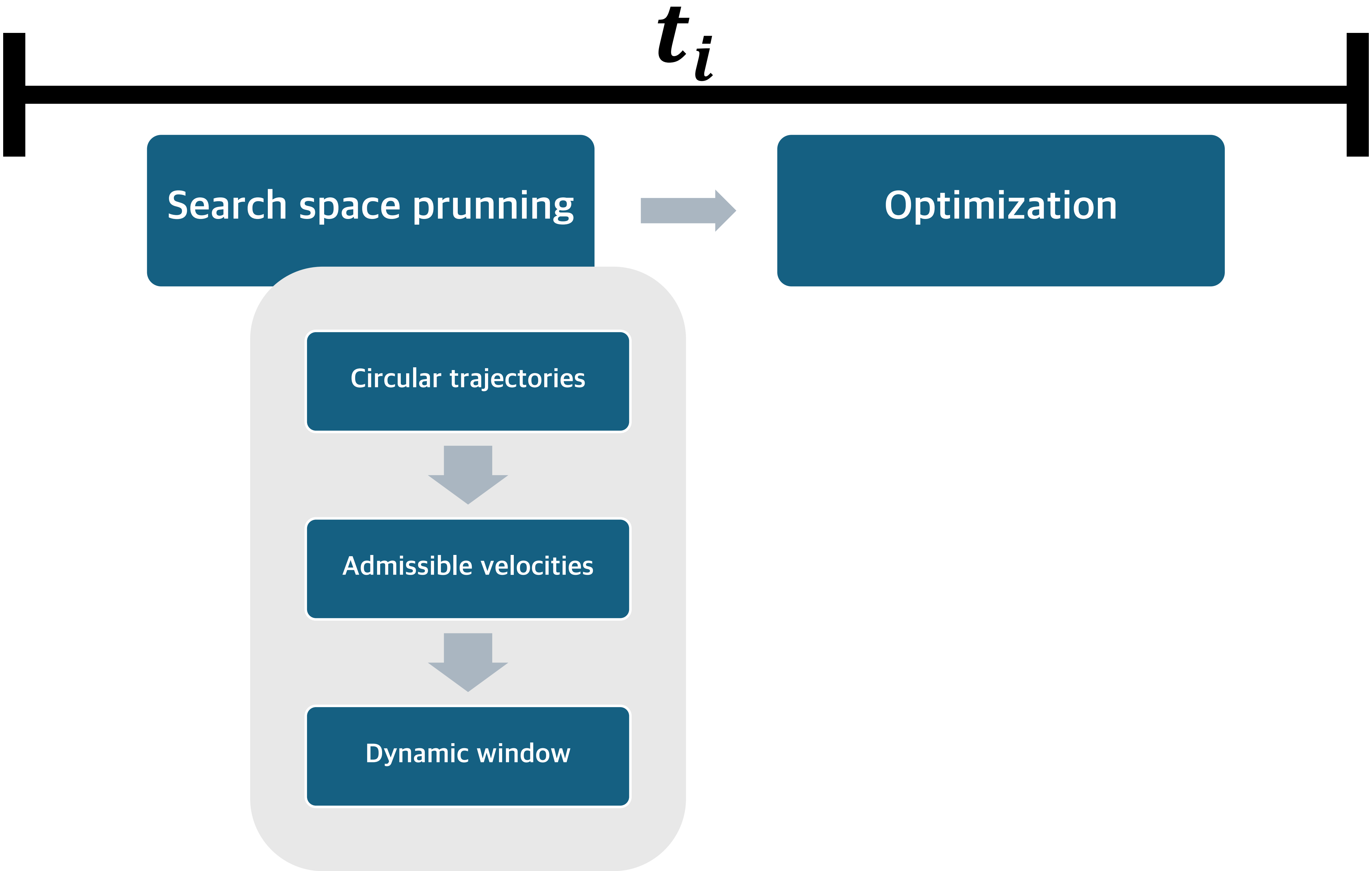


Figure 11. Objective Function.

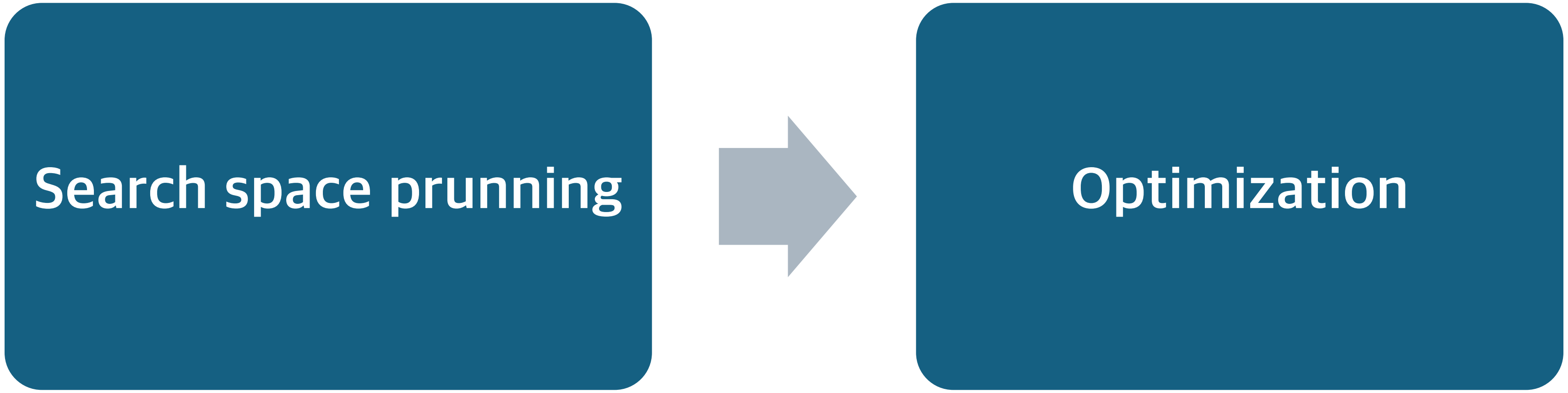

t_i
Search space prunning
Optimization
Circular trajectories
Admissible velocities
Dynamic window

3

Code Implemention of DWA Algorithm in Nav2

Dynamic Window Approach

- Local trajectory planning algorithm used for real-time obstacle avoidance and navigation.



https://github.com/ros-navigation/navigation2/tree/main/nav2_dwb_controller/dwb_plugins

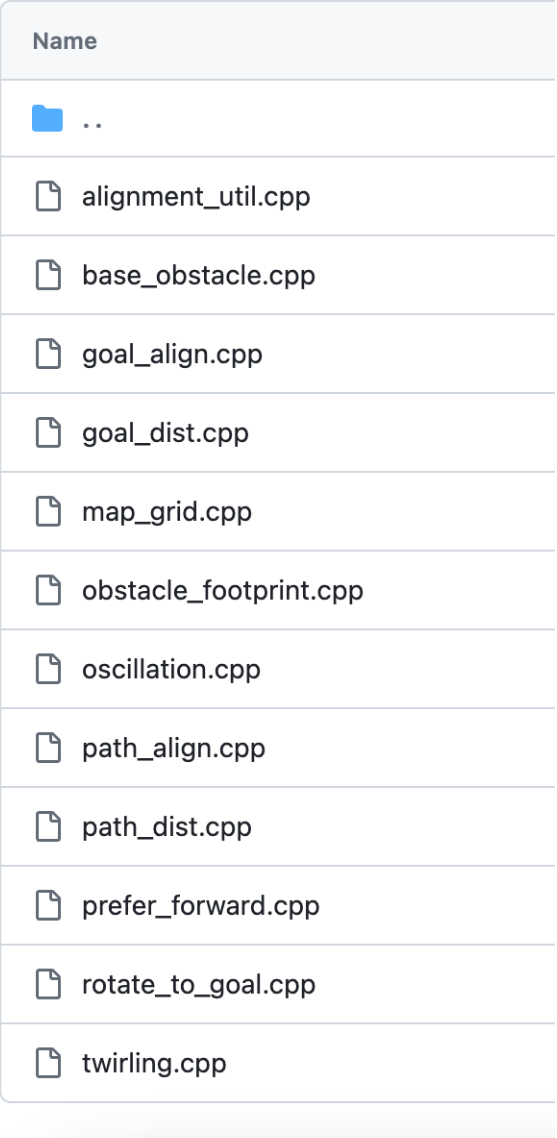
DWB Plugins

1. Trajectory Generator Plugins

- Generate the set of possible trajectories that should be evaluated by the critics
- The trajectory with the best score determines the output command velocity.
- Only one can be loaded at a time.
- StandardTrajectoryGenerator
- LimitedAccelGenerator

2. Critic Plugins

- Score the trajectories generated by the trajectory generator.
- Multiple plugins can be loaded and the sum of their scores determines the chosen command velocity.



<Critic Plugins>

Trajectory Generator Plugins:

StandardTrajectoryGenerator vs. LimitedAccelGenerator

```
class LimitedAccelGenerator : public StandardTrajectoryGenerator
{
public:
  void initialize(
    const nav2_util::LifecycleNode::SharedPtr & nh,
    const std::string & plugin_name) override;
  void startNewIteration(const nav_2d_msgs::msg::Twist2D & current_velocity) override;

protected:
  /**
   * @brief Calculate the velocity after a set period of time, given the desired velocity and acceleration limits
   *
   * Unlike the StandardTrajectoryGenerator, the velocity remains constant in the LimitedAccelGenerator
   *
   * @param cmd_vel Desired velocity
   * @param start_vel starting velocity
   * @param dt amount of time in seconds
   * @return cmd_vel
   */
  nav_2d_msgs::msg::Twist2D computeNewVelocity(
    const nav_2d_msgs::msg::Twist2D & cmd_vel,
    const nav_2d_msgs::msg::Twist2D & start_vel,
    const double dt) override;
  double acceleration_time_;
  std::string plugin_name_;
};
}  // namespace dwb_plugins
```

Critic Plugins

- **BaseObstacle** – Scores a trajectory based on where the path passes over the costmap. To use this properly, you must use the inflation layer in costmap to expand obstacles by the robot's radius.
- **ObstacleFootprint** – Scores a trajectory based on verifying all points along the robot's footprint don't touch an obstacle marked in the costmap.
- **GoalAlign** – Scores a trajectory based on how well aligned the trajectory is with the goal pose.
- **GoalDist** – Scores a trajectory based on how close the trajectory gets the robot to the goal pose.
- **PathAlign** – Scores a trajectory based on how well it is aligned to the path provided by the global planner.
- **PathDist** – Scores a trajectory based on how far it ends up from the path provided by the global planner.
- **PreferForward** – Scores trajectories that move the robot forwards more highly
- **RotateToGoal** – Only allows the robot to rotate to the goal orientation when it is sufficiently close to the goal location
- **Oscillation - Prevents the robot from just moving backwards and forwards.**
- **Twirling** – Prevents holonomic robots from spinning as they make their way to the goal.

```cpp
double OscillationCritic::scoreTrajectory(const dwb_msgs::msg::Trajectory2D & traj)
{
  if (x_trend_.isOscillating(traj.velocity.x) ||
    y_trend_.isOscillating(traj.velocity.y) ||
    theta_trend_.isOscillating(traj.velocity.theta))
  {
    throw dwb_core::
          IllegalTrajectoryException(name_, "Trajectory is oscillating.");
  }
  return 0.0;
}
```

```cpp
bool OscillationCritic::CommandTrend::isOscillating(double velocity)
{
  return (positive_only_ && velocity < 0.0) || (negative_only_ && velocity > 0.0);
}
```

Thank You

Reference

- Nav2 Github. https://github.com/ros-navigation/navigation2.
- DWB Controller — Nav2 1.0.0 documentation. https://docs.nav2.org/configuration/packages/dwb-params/controller.html.
- D. Fox, W. Burgard and S. Thrun, "The dynamic window approach to collision avoidance," in IEEE Robotics & Automation Magazine, vol. 4, no. 1, pp. 23-33, March 1997, doi: 10.1109/100.580977.
- 개인블로그, “[논문 리뷰] Dyanamic Window Approach (DWA) 알고리즘”. https://oxcarxierra.com/dwa-paper-review/.
- 개인블로그, “[ROS2::Nav2::DWB] Dynamic Window Approach”. https://robonote.tistory.com/93.