Introduction
ooo

DWA
oooooo

Custom controller
oooo

The experiment
ooooooooooooooo

Experimental results
oooooooo

Faced issues
oooo

Conclusion
oo

# DWA vs custom trajectory tracking controller
## A comparison in ROS

G. Chiari    L. Gargani    S. Salvi

Politecnico di Milano

Academic Year 2022/2023

# Table of Contents

# Table of Contents

## Project overview
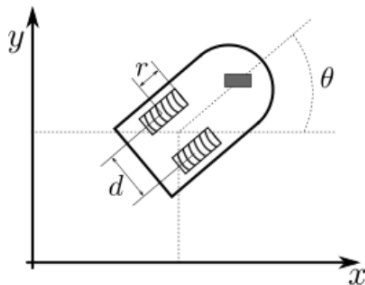
Goals of the project:

- implement a custom trajectory tracking controller
- use DWA standalone to control the robot
- compare the results of the two approaches above

The simulation is performed on a differential drive robot, using its kinematic model.

### Environment

Everything runs on *ROS Melodic* on *Ubuntu 18.04*.

# Differential drive



Kinematic model of the robot:

$$\begin{cases} \dot{x} = \frac{\omega_r + \omega_l}{2} r \cos(\theta) \\ \dot{y} = \frac{\omega_r + \omega_l}{2} r \sin(\theta) \\ \dot{\theta} = \frac{\omega_r - \omega_l}{d} r \end{cases}$$

# Table of Contents

Paper formulation (1)

Given a certain goal point to be reached by the robot, DWA tries
to find the optimal linear and angular velocities to go there.

This is mainly done in two sequential steps:

1. search space reduction
2. objective function optimization

# Paper formulation (2)

Search-space: all the possible $(v, \omega)$ tuples.

Search-space reduction steps:

1. consider only circular trajectories
2. consider only admissible velocities (stop before obstacles)
3. exclude velocities that cannot be reached given the limited accelerations of the robot

# Paper formulation (3)

The remaining velocities are fed into the following objective function to be maximized:

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

At this point the tuple $(v, \omega)$ leading to the highest score is chosen and the robot picks those velocities.

# ROS implementation (1)

DWA is already implemented in ROS in the dwa_local_planner package.

This package is ought to be used as the planner for move_base within the navigation stack.

## ROS implementation (2)

Cost function from ROS Wiki:

cost = path_distance_bias * (distance to path from the endpoint of the trajectory)
      + goal_distance_bias *(distance to local goal from the endpoint of the trajectory)
      + occdist_scale * (maximum obstacle cost along the trajectory in obstacle cost (0-254))

However, this function is used only for visualization purposes.

The real cost is computed instead as a weighted sum of the
following terms:

- oscillation_costs_
- obstacle_costs_
- goal_front_costs_
- alignment_costs_

- path_costs_
- goal_costs_
- twirling_costs_

# Table of Contents

Unicycle model control

When designing the controller it is possible to work on the unicycle (thus not differential) model, for simplicity:

$$\begin{cases} \dot{x} = v \cdot \cos(\theta) \\ \dot{y} = v \cdot \sin(\theta) \\ \dot{\theta} = \omega \end{cases}$$

In fact, the differential model can be easily reduced to the unicycle one with the transformations:

$$\begin{cases} v = \frac{\omega_r + \omega_l}{2} r \\ \omega = \frac{\omega_r - \omega_l}{2} r \end{cases}$$

# Controller's architecture (1)

The developed controller works by taking only the reference points (making up the full trajectory), and then computing everything else on its own.

The controller is composed of an inner linearisation law (based on the kinematic model) and an outer tracking law (based on a proportional integral controller with velocity feed-forward).
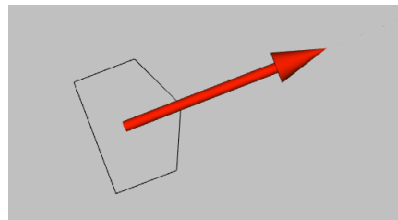
# Controller's architecture (2)
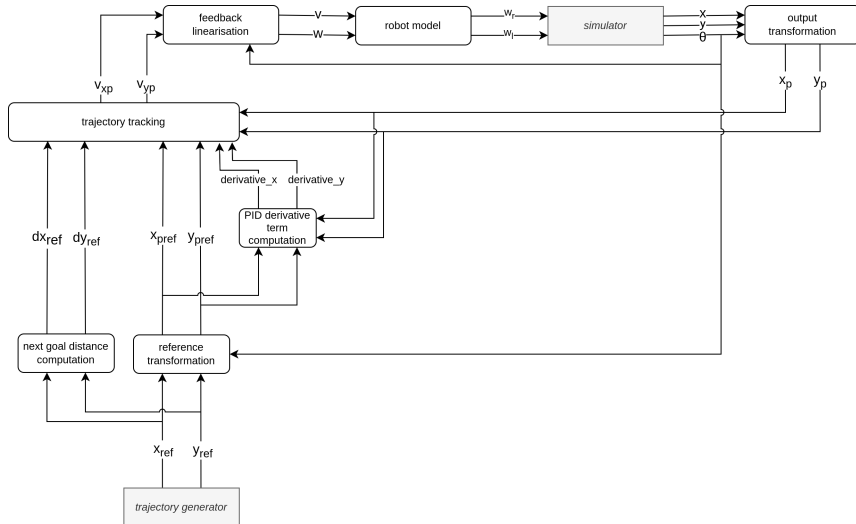
# Table of Contents

# Experiment setup - the robot
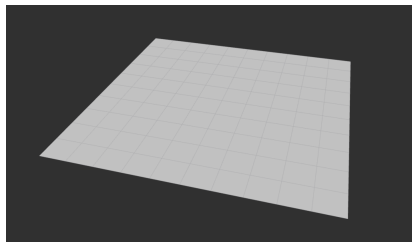
The robot:

- differential drive
- $d = 15$ cm (wheels distance)
- $r = 3$ cm (wheels radius)
- pentagonal footprint
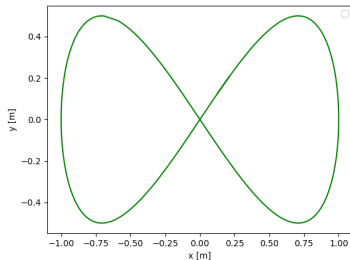
# Experiment setup - the map

The map:

- empty, no obstacles around
- global, no need for a local one

## Experiment setup - the trajectory

The trajectory:

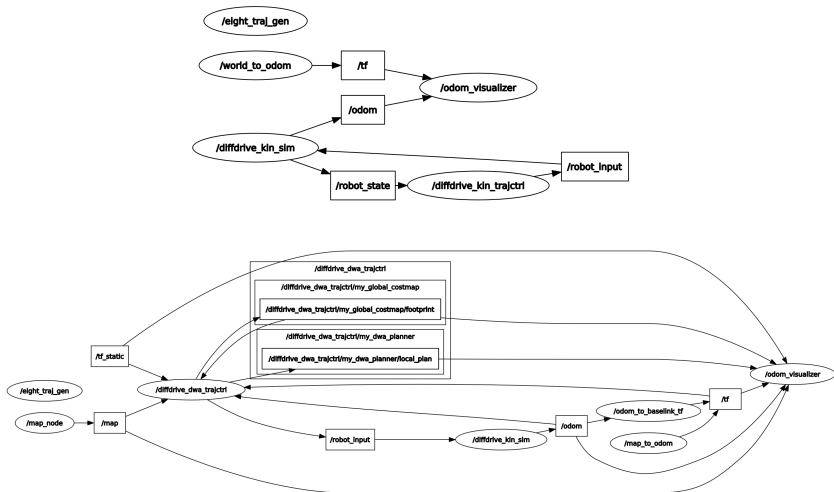- eight-shaped
- 2 x 1 meters
- discretized into multiple goals

# ROS architecture - overview (1)

Three packages:

- diffdrive_kin_sim $\rightarrow$ simulator
- diffdrive_kin_ctrl $\rightarrow$ custom controller
- diffdrive_dwa_ctrl $\rightarrow$ DWA controller

The two controllers are interchangeable and are meant to always be used together with the simulator, one at a time.

# ROS architecture - overview (2)
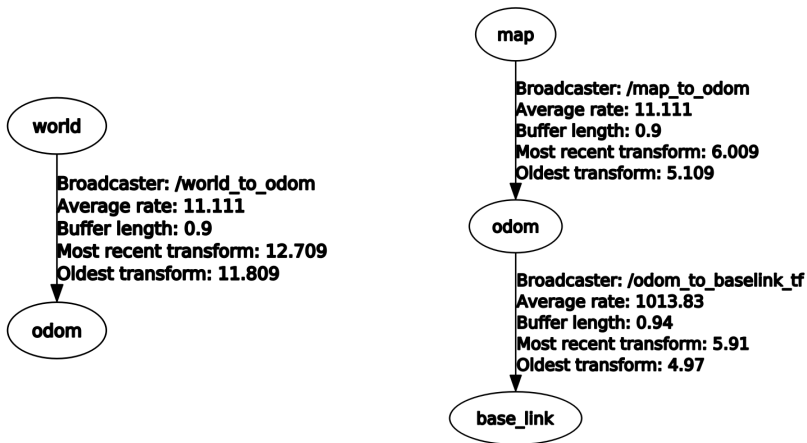
# ROS architecture - service & frames (1)

One service:

- generate_desired_path_service $\rightarrow$ used to generate the eight-shaped trajectory as soon as it is invoked by one of the two controllers

Three frames:

- map $\rightarrow$ for the empty map provided by the map server
- odom $\rightarrow$ representing the global reference system
- base_link $\rightarrow$ representing the moving reference system

# ROS architecture - service & frames (2)



world

Broadcaster: /world_to_odom
Average rate: 11.111
Buffer length: 0.9
Most recent transform: 12.709
Oldest transform: 11.809

odom

map

Broadcaster: /map_to_odom
Average rate: 11.111
Buffer length: 0.9
Most recent transform: 6.009
Oldest transform: 5.109

odom

Broadcaster: /odom_to_baselink_tf
Average rate: 1013.83
Buffer length: 0.94
Most recent transform: 5.91
Oldest transform: 4.97

base_link

# ROS architecture - nodes

Four nodes:

- diffdrive_kin_sim_node, $\rightarrow$ integration logic to update the robot pose given $(\omega_r,\omega_l)$

- diffdrive_kin_trajctrl_node, $\rightarrow$ computation of $(\omega_r,\omega_l)$ to reach the next point in the trajectory, using the custom controller

- diffdrive_dwa_trajctrl_node, $\rightarrow$ interface between DWA library and simulator to compute $(\omega_r,\omega_l)$

- odom_to_baselink_tf_node, link between the odom and the base_link coordinate frames through a dynamic tf

# ROS architecture - topics

Five topics:

- `/clock`, $\rightarrow$ synchronization of all the nodes in the simulation
- `/odom`, $\rightarrow$ communication of the odometry information of the robot to DWA
- `/robot_state`, $\rightarrow$ communication of the odometry information of the robot to the custom controller
- `/controller_state`, $\rightarrow$ for visualization purposes
- `/robot_input`, $\rightarrow$ communication of $(\omega_r, \omega_l)$ computed by the controllers

# ROS architecture - launch files

Two launch files:

- diffdrive_kin_trajctrl.launch → start the simulation of the robot's behavior with the custom controller
- diffdrive_dwa_trajctrl.launch → start the simulation of the robot's behavior with DWA

## Parameters tuning - trajectory

Equations describing the eight-shaped trajectory:

$$\begin{cases} x = \mathtt{a} \cdot \sin(\mathtt{w} \cdot t) \\ y = \mathtt{a} \cdot \sin(\mathtt{w} \cdot t) \cdot \cos(\mathtt{w} \cdot t) \end{cases}$$

Two main parameters:

- $\mathtt{a} \rightarrow$ amplitude of the eight-shaped trajectory
- $\mathtt{w} \rightarrow$ ratio $\frac{2 \cdot \pi}{T}$ where T is the time duration of each lap

Assigned values

$$\mathtt{a} = 1 \qquad\qquad \mathtt{w} = 1$$

## Parameters tuning - custom controller

Step response of a PID controller:

$$u(t) = K_{\mathrm{p}}e(t) + K_{\mathrm{i}}\int_0^t e(\tau)\,\mathrm{d}\tau + K_{\mathrm{d}}\frac{\mathrm{d}e(t)}{\mathrm{d}t}$$

Three main parameters:

- $K_p \rightarrow$ proportional gain of the PID controller
- $K_i \rightarrow$ integral gain of the PID controller
- $K_d \rightarrow$ derivative gain of the PID controller

### Assigned values

| | | |
|---|---|---|
| $K_p = 0.8$ | $K_i = 0.8$ | $K_d = 0.0$ |

# Parameters tuning - DWA (1)

Two of the many predefined parameters:

- vth_samples $\rightarrow$ number of the evaluated samples for $\omega$
- vx_samples $\rightarrow$ number of the evaluated samples for $v$

### Assigned values

vth_samples $= 100$      vx_samples $= 30$

# Parameters tuning - DWA (2)

One custom parameter:

- skipped_goals $\rightarrow$ number of points to skip when feeding the trajectory to DWA

### Assigned values

$$\text{skipped\_goals} = 15$$

# Table of Contents

## Python scripts

To visualize the experimental results of the experiment, two Python scripts can be used to analyze the recorded *bag* files:

- plot_result.py $\rightarrow$ plot the result of a single *bag* file
- plot_comparison.py $\rightarrow$ compare the results of two *bag* files

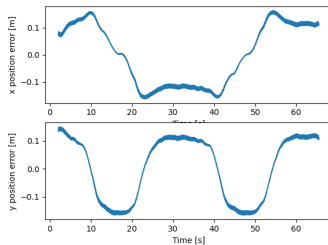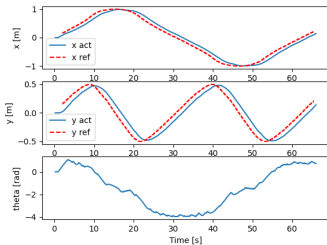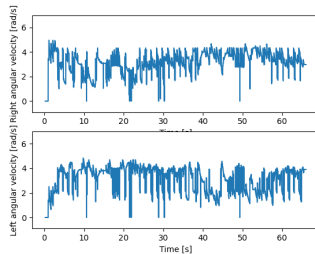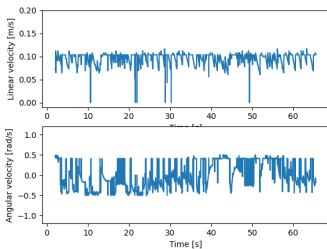The following plots are obtained with an optimal parameters choice.

# Trajectory tracking controller (1)
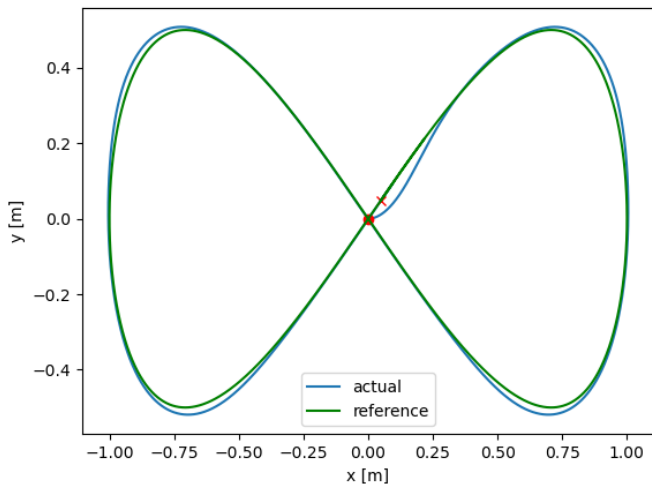
# Trajectory tracking controller (2)
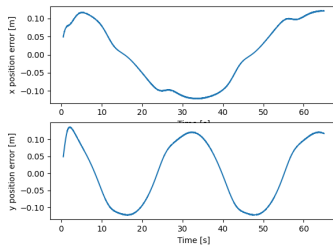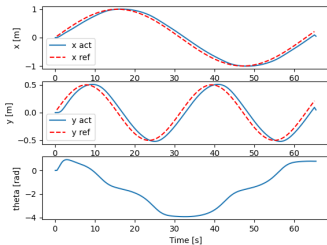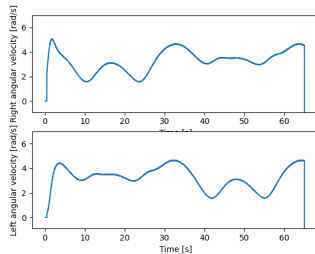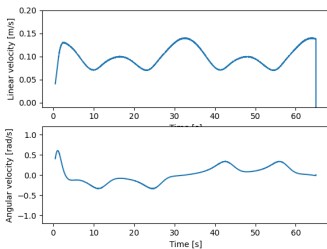
# DWA (1)
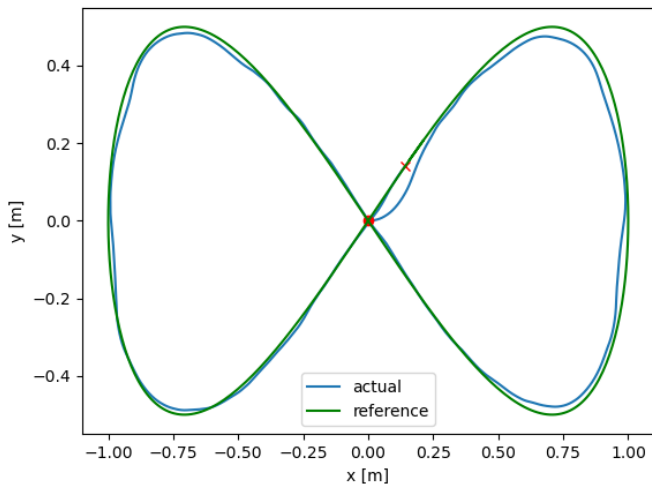
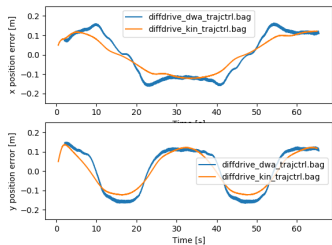# DWA (2)

# Comparison (1)
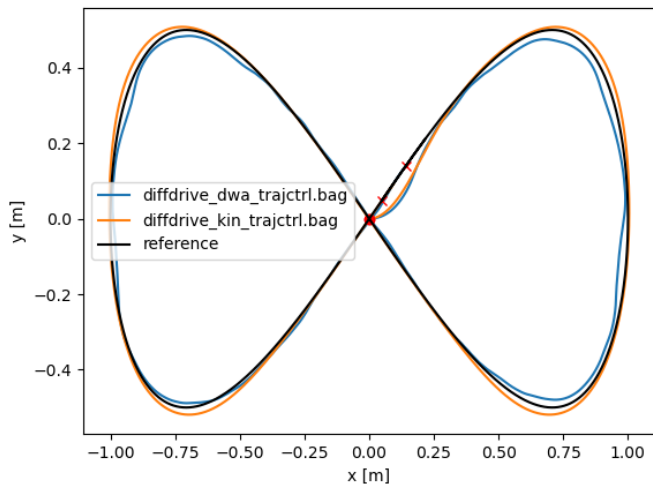
# Comparison (2)
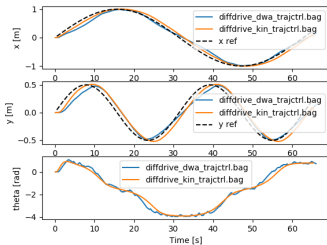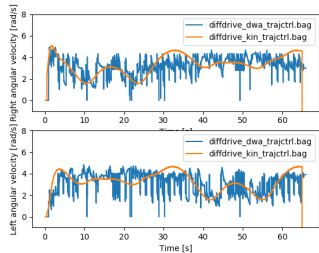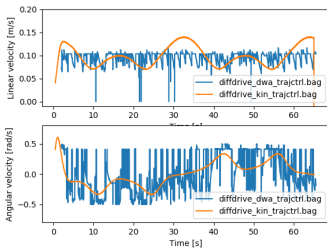
## Table of Contents

# Deprecated parameters

There are some DWA parameters that are named differently
between the ROS Wiki and the library source code.

```
// Warn about deprecated parameters -- remove this block in N-turtle
nav_core::warnRenamedParameter(private_nh, "max_vel_trans", "max_trans_vel");
nav_core::warnRenamedParameter(private_nh, "min_vel_trans", "min_trans_vel");
nav_core::warnRenamedParameter(private_nh, "max_vel_theta", "max_rot_vel");
nav_core::warnRenamedParameter(private_nh, "min_vel_theta", "min_rot_vel");
nav_core::warnRenamedParameter(private_nh, "acc_lim_trans", "acc_limit_trans");
nav_core::warnRenamedParameter(private_nh, "theta_stopped_vel", "rot_stopped_vel");
```

## Unadvertised warning!

Without using `nav_core`, the warnings are not raised.

## DWA used standalone

DWA expects information on certain topics (/odom, /goal, /map).
When used standalone, we must provide such information manually.

Moreover, the few lines of code in the ROS Wiki explaining how to
setup DWA standalone are no longer working:

```
#include <tf/transform_listener.h>
#include <costmap_2d/costmap_2d_ros.h>
#include <dwa_local_planner/dwa_planner_ros.h>

...

tf::TransformListener tf(ros::Duration(10));
costmap_2d::Costmap2DROS costmap("my_costmap", tf);

dwa_local_planner::DWAPlannerROS dp;
dp.initialize("my_dwa_planner", &tf, &costmap);
```

**Figure:** Snippet in the ROS Wiki

```
#include <tf2_ros/transform_listener.h>
#include <costmap_2d/costmap_2d_ros.h>
#include <dwa_local_planner/dwa_planner_ros.h>

...

tf2_ros::Buffer tfBuffer(ros::Duration(10));
tf2_ros::TransformListener tfListener(tfBuffer);
costmap_2d::Costmap2DROS my_global_costmap("my_global_costmap", tfBuffer);
my_global_costmap.start();

dwa_local_planner::DWAPlannerROS dp;
dp.initialize("my_dwa_planner", &tfBuffer, &my_global_costmap);
```

**Figure:** Modified snippet

## Multiple goals

Generally, DWA takes only a single final goal and computes the full trajectory on its own.

In this project the trajectory is predefined and must be explicitly forced. This is done by continuously changing the goal.

We must find a suitable 'density' for the goals vector:

- too many goals $\Rightarrow$ unsteady profile in the velocities
- too few goals $\Rightarrow$ large deviation from the reference trajectory

# Table of Contents

1. Introduction

2. DWA

3. Custom controller

4. The experiment

5. Experimental results

6. Faced issues

7. **Conclusion**

## Conclusion

Regarding the trajectory tracking aspect only, the two control
methods behave in a similar way even though the custom
controller performs moderately better than DWA.

However, the most noticeable difference is in the smoothness of
the plots of linear and angular velocities: the custom controller
produces velocities that are much smoother and realistic than
those produced by DWA.