POLITECNICO
MILANO 1863

Extended project work

A Gazebo car simulator, analysis and comparison with a single-track model

Colli Stefano, Pagani Mattia, Panelli Erica

"Control of Mobile Robots" Course

A.A. 2022/2023

Abstract

Contents

1 Notes on Installation and Launch **3**

1.1 Installation 3

1.1.1 Downloading material 3

1.1.2 Additional packages to be installed 3

1.1.3 Additional modifications 4

1.2 Launch 4

1.2.1 Original Project 4

2 General Project Structure **5**

2.1 Catkin Workspace Directories 6

2.1.1 Original MIT Racecar Packages 6

2.1.2 Added Packages 7

3 Original System Introduction **8**

4 (Our) System Description **9**

4.1 Scheme of the whole system 9

4.2 Topics 11

4.2.1 Scheme of topic publications/subscriptions 11

4.2.2 Topics meaning 13

5 Detailed Package Description **15**

5.1 Package (original) ackermann_msgs 15

5.2 Package (original) racecar 15

5.3 Package (original) racecar_gazebo 15

5.4 Package car_control 15

5.4.1 Intro 15

5.4.2 Configuration 16

5.4.3 Launch 17
5.4.4 Node car_control 17
5.5 Package car_kinematic_control 17
5.5.1 Intro 17
5.5.2 Configuration 18
5.5.3 Launch 19
5.5.4 Node car_kin_controller 19
5.6 Package trajectoy_tracker 20
5.6.1 Configuration 20
5.6.2 Node trajectory_tracker 22
5.6.3 Choiche of PID controller and parameters 25
5.7 Package CarCommndsFr 25
5.7.1 Intro 25
5.7.2 Configuration 25
5.7.3 Launch 25
5.7.4 Node car_commands_fr 25

A launch package inclusion **27**

Chapter 1

Notes on Installation and Launch

1.1 Installation

1.1.1 Downloading material

The project is based on the material of original MIT racecar. That's it, we have generated a new ROS environment copying MIT repository packages. In particular the following packages have been downloaded:

- ackermann_msgs
- racecar
- racecar_gazebo

They can be found at the link: `https://github.com/mit-racecar`

1.1.2 Additional packages to be installed

To be able to compile the project it is necessary to download two internal ROS packages which will be used by the racecar ones. Launch the following commands:

```
sudo apt install ros-noetic-ros-control
sudo apt install ros-noetic-ros-controllers
```

Otherwise an error will be thrown when `catkin_make` command is called.

1.1.3 Additional modifications

In some cases, to avoid conflicts, it's required to change Python environment to version 3 in each file of the original packages. In particular, if Python environment is set to 3, modifications are needed for `joy_teleop.py` file:

- Row 277: replace ',' with 'as'
- Row 282: replace `iteritems` with `items`

1.2 Launch

1.2.1 Original Project

In order to launch original project, once it's compiled following ROS guide, following steps should be followed:

- (run) `roscore`
- (run) `keyboard_teleop.py`
- (run) `racecar_gazebo racecar.tunnel`

If there are no errors the user should be able to see the racecar in a Gazebo environment. WASD keys on keyboard produce car moves.

Chapter 2

General Project Structure

2.1 Catkin Workspace Directories

2.1.1 Original MIT Racecar Packages

ackermann_cmd_mux (racecar folder)	...
ackermann_msgs	Contains definitions of **AckermannDrive** and **AckermannDriveStamped** messages, used by the racecar to compute movements.
racecar (racecar folder)	Directory which contains
racecar_control (racecar_gazebo folder)	Contains launch files to load controllers used to manage the motors of the racecar. Also load nodes which dispatch messages to controllers.
racecar_description (racecar_gazebo folder)	Contains a description of the racecar, in terms of models, meshes ecc... It will be used by Gazebo to represent it.
racecar_gazebo (racecar_gazebo folder)	Mainly contains launch scripts used to load all necessary nodes, worlds and other components to open a Gazebo instance with a controllable car.

2.1.2 Added Packages

car_control	Contains node which performs the linearization of the nonlinear bycicle **dynamic** model. It' receives desired velocities from trajectory tracker and sends Ackermann commands to the racecar.
car_kinematic_control	Contains node which performs the exact linearization of the nonlinear bycicle **kinematic** model. It' receives desired velocities from trajectory tracker and sends Ackermann commands to the racecar.
trajectory_tracker	Generates (or receives in input) a desired trajectory and actual car positions, than compute desired velocities to be sent to controllers.
CarCommandsFr	Interface used to replace inner wheel friction phisical model of ROS with a custom one.

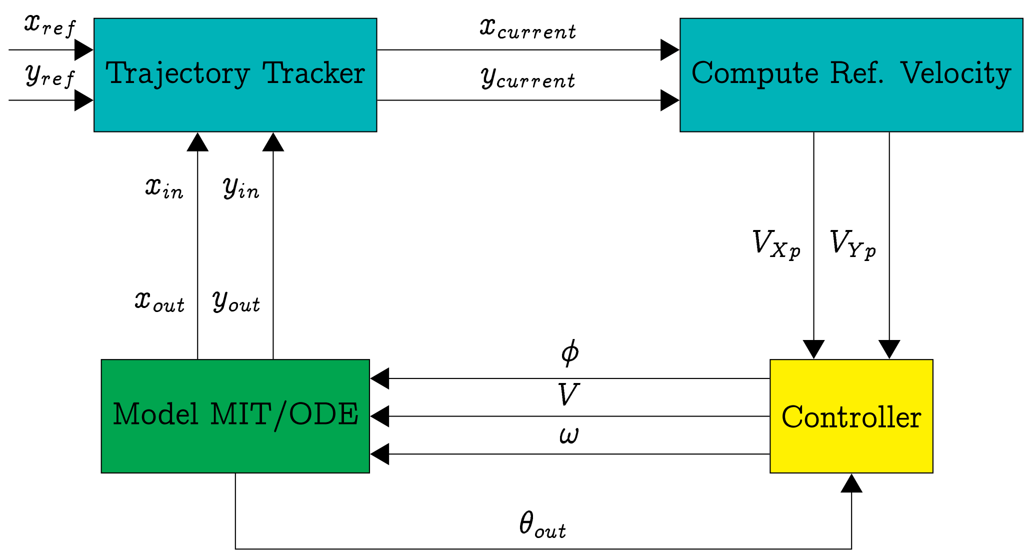
Chapter 3

Original System Introduction

Chapter 4

(Our) System Description

4.1 Scheme of the whole system



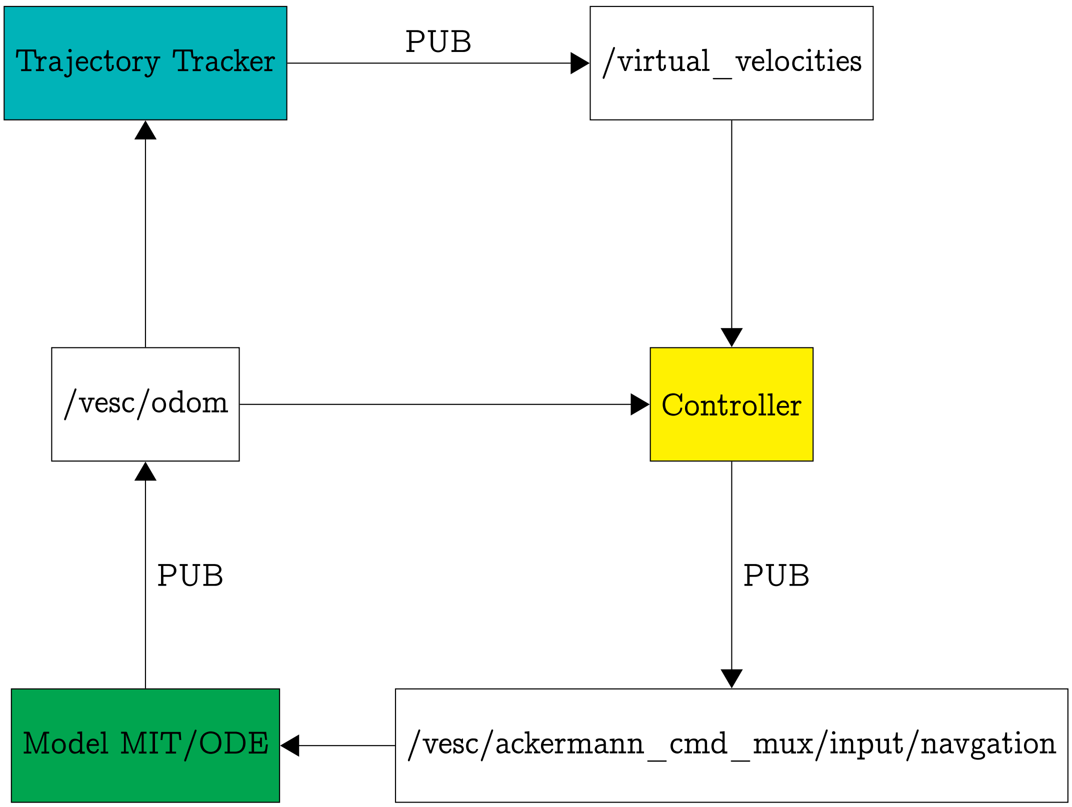
Symbol	Meaning
x_{ref}, y_{ref}	Ref. position of the trajectory
V_{Xp}, V_{Yp}	Required velocities of the point. They will be imposed by controller
ϕ	Steer degree of rotation
V	Vector velocity
ω	Steer speed of rotation
θ_{out}	Car pose: rotation around center axis
x_{out}, y_{out}	Car pose: x, y

Note that "trajectory tracker" generated trajectories are hard-coded, even if x_{ref} and y_{ref} are shown as input parameters. The user can select the trajectory using YAML configuration file (which will be explained in the relative section).

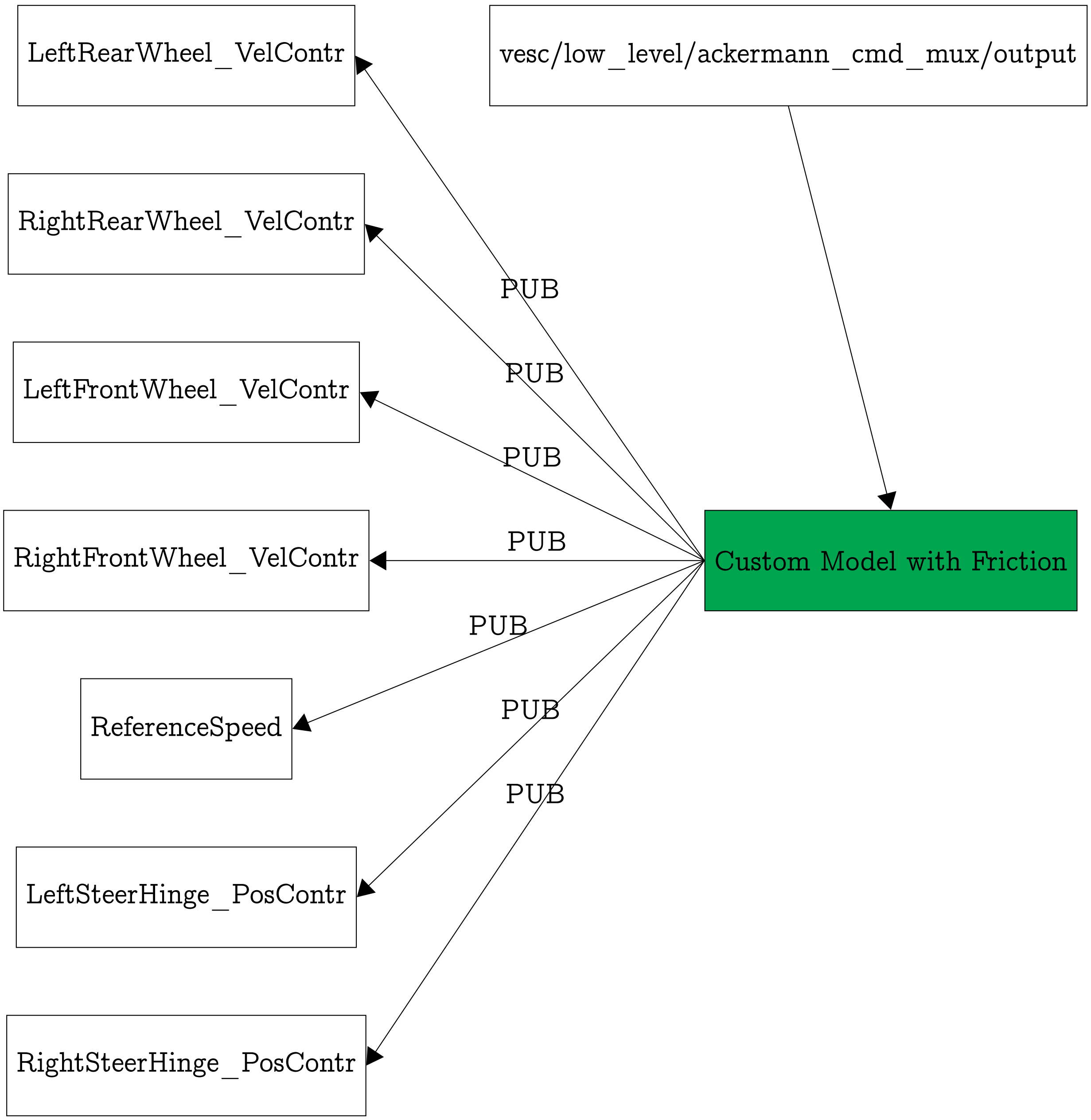
4.2 Topics

4.2.1 Scheme of topic publications/subscriptions

ROS Friction Model



Custom Friction Model



4.2.2 Topics meaning

Common Topics

/virtual_velocities	Used by "trajectory tracker" to publish desired velocity components. These are read by controller in order to perform linearization and compute instructions for the model.
/vesc/ackermann_cmd_mux /input/navgation	Contains AckermannDriveStamped messages sent by controller. These messages contains information for the racecar, about velocity and steering.
/vesc/odom	The model uses this topic to publish odometry information of the racecar (position and orientation). These data are used both by tracker and controller. The first one compute differences between actual car position and desired position imposed by trajectory. The last one reads z-axis orientation useful to perform linearization.

There is another topic in which "trajectory tracker" publish, the */reference_trajectory*. This is used to read trajectory information to perform debug and register data for analysis.

Specific Topics

/vesc/low_level/ ackermann_cmd_mux/output	xyz
/racecar/ left_rear_wheel _velocity_controller/command	xyz
/racecar/ right_rear_wheel _velocity_controller/command	xyz
/racecar/ left_front_wheel _velocity_controller/command	xyz
/racecar/ right_front_wheel _velocity_controller/command	xyz
/reference_speed	xyz
/racecar/ left_steering_hinge _position_controller/command	xyz
/racecar/ right_steering_hinge _position_controller/command	xyz

Chapter 5

Detailed Package Description

5.1 Package (original) ackermann_msgs

5.2 Package (original) racecar

5.3 Package (original) racecar_gazebo

5.4 Package car_control

5.4.1 Intro

Even if this package is not used, it's correct to do a description of the objective it should have reached.

The aim was to implement a dynamic controller, which perform exact linearization of the nonlinear bicycle dynamic model. To do this it needs more parameter respect to the kinematic one.

In addition, we put a scheme (5.1) of the principal parameters and variables used for linearization.

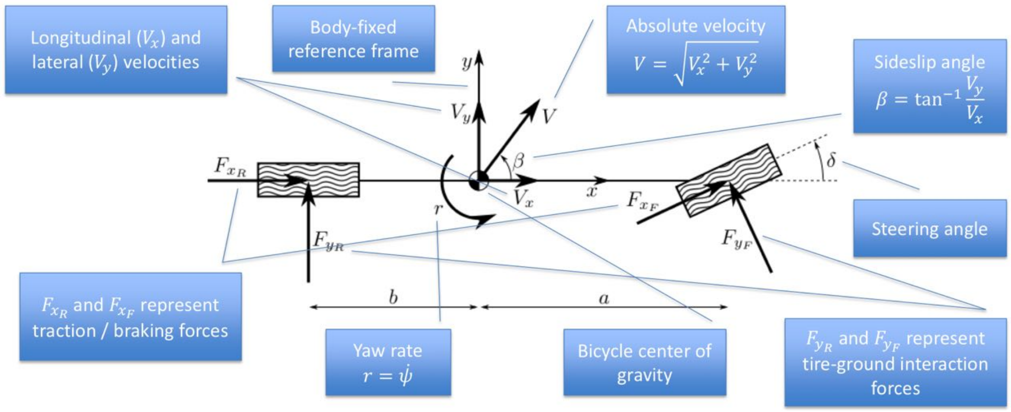


Figure 5.1: dynamic model with main parameters and variables

5.4.2 Configuration

Input values	
Vp_x	Point velocity x
Vp_y	Point velocity y
ψ	Yaw[1]
$\dot{\psi}$	Yaw rate[2]

Model parameters	
C_f, C_r	Viscous friction coefficients
a, b	Distance between wheels center and Center of Gravity
M	Vehicle mass
ϵ	Distance between Center of Gravity and a point P, along the velocity vector. Linearization is done around point P. This parameter should be chosen empirically

[1] In the System Scheme, this is represented by θ_{out}

[2] In the System Scheme, this is **not** represented (as we have used, for tests, only the kinematic model)

Intermediate computed values	
β	Sideslip angle: $\tan^{-1}\left(\frac{Vp_y}{Vp_x}\right)$

Output values	
V	Point absolute velocity
δ	Steering angle
ω	Steering speed

5.4.3 Launch

There is a lunch file which **should be used to execute the node**. This contains also information about debugging level and loads configuration file.

5.4.4 Node car_control

$$\beta = \tan^{-1}\left(\frac{Vp_y}{Vp_x}\right)$$

$$\delta = \frac{MV}{C_f}\omega + \frac{C_f + C_r}{C_f}\beta - \frac{bC_r - aC_f}{C_f}\frac{\dot{\psi}}{V}$$

$$\begin{bmatrix} V \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\beta+\psi) & \sin(\beta+\omega) \\ -\frac{\sin(\beta+\psi)}{\epsilon} & \frac{\cos(\beta+\psi)}{\epsilon} \end{bmatrix} \begin{bmatrix} Vp_x \\ Vp_y \end{bmatrix}$$

5.5 Package car_kinematic_control

5.5.1 Intro

Before starting the explanation, we add a brief high level description of Quaternions, which are used in messages to represent orientations. Even a distinction between pose and position is done.

Quaternion: a different way to describe the orientation of a frame only. It's an alternative to Yaw, Pitch and Roll. A quaternion has four parameters: x, y, z, w. Pay attention, they are NOT a position vector.

Position: position of the robot in a 3D space.

Pose: position (3 DOF) + orientation (3 DOF).

In conclusion the pose has 6 D.O.F. which are: x, y, z, roll, pitch, yaw. Euler angles can be converted to quaternions, which are better. Transformation functions of ROS can do this conversion and the reverse one.

In addition, we put a scheme (5.2) of the principal parameters and variables used for linearization.

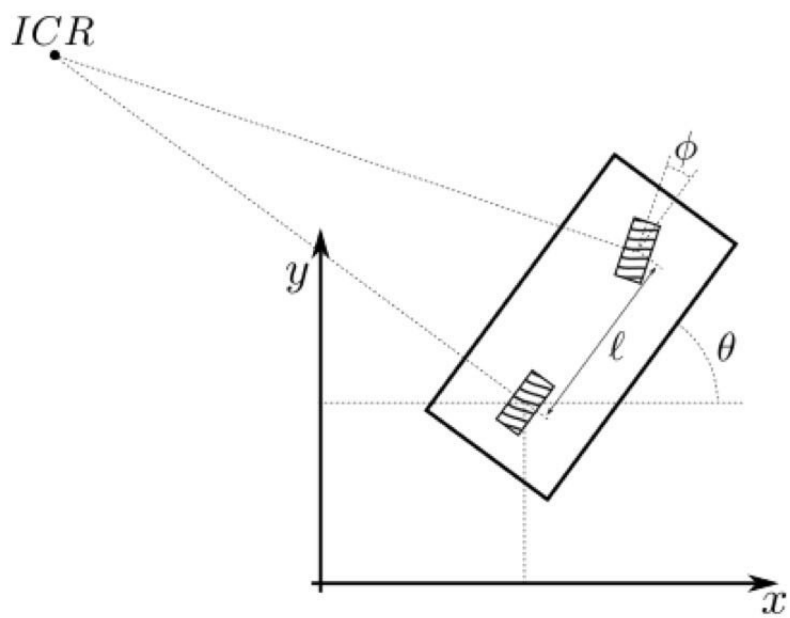


Figure 5.2: bicycle vehicle with main parameters and variables

5.5.2 Configuration

In the package there is a configuration file, containing: the parameter L, which represents distance between rear and front wheels; the parameter ϵ, the distance between Center of Gravity and a point P, along the velocity vector. Linearization is done around point P. This parameter should be chosen empirically.

Both are used in the linearization.

5.5.3 Launch

There is a lunch file which **should be used to execute the node**. This contains also information about debugging level and loads configuration file.

5.5.4 Node car_kin_controller

Node requirements: distance between rear and front wheels as parameter.

The node has two callbacks:

- One used to retrieve desired velocities of the point. These velocities are computed by trajectory tracker and published in `/virtual_velocities` topic, subscribed by the controller node.
- One used to retrieve the orientation of the car around `z` axis. This is done reading from `/vesc/odom`. The information retrieved are in the form of a quaternion and are converted into roll, pitch and yaw. Yaw is taken. In addition, even the speed around `z` axis is read (`twist.angular.z`).

The node perform an exact linearization of the nonlinear bicycle cinematic model. The change of coordinates is applied as follows:

$$V = V_{Xp} cos(\theta) + V_{Yp} sin(\theta)$$

$$\phi = \arctan\left(\frac{l}{\epsilon}\frac{V_{Yp} cos(\theta) - V_{Xp} sin(\theta)}{V_{Xp} cos(\theta) + V_{Yp} sin(\theta)}\right)$$

Where

- l is the distance between rear and front wheels
- ϵ is the distance between Center of Gravity and a point P
- V_{Xp} and V_{Yp} are the desired point velocities
- θ is the car orientation around z-axis
- ϕ is the steering angle

- V is the driving velocity of the front wheel

In addition, the program compute the steering speed as $\omega = \frac{V}{l}\tan(\phi)$. This value is not used in the construction of the message because it's ignored by the model.

Once the linearization is performed an `AckermannDriveStamped` message is built, containing V and ϕ. This message is published on `/vesc/ackermann_cmd_mux/input/navigation` topic, which is read by the model to make the car move. Linearization and command sending operations are repeated in a loop, which is the core of the node.

5.6 Package trajectoy_tracker

5.6.1 Configuration

In the YAML configuration file are present many parameters which are used in the node.

Node configuration parameters	
trajectory_type	Desired trajectory (0 = linear; 1 = parabolic; 2 = circular; 3 = eight shape; 4 = cycloidal; 5 = fixed point)

Linear trajectory parameters. The trajectory is parallel to the direction vector (a_coeff, b_coeff)	
a_coeff	
b_coeff	

Parabolic trajectory parameters	
parabola_convexity	Convexity a of the parabola having equation $y = ax^2$

Circular trajectory parameters	
R	Radius of the circle described by the trajectory
W	Angular speed for the lap ($W = 2\pi/T$, where T is the time duration of each lap)

Eight-shape trajectory parameters	
a	Trajectory amplitute (the eight-shape goes from $-a$ to a)
w	Angular speed for the lap ($w = 2\pi/T$, where T is the time duration of each lap)

Cycloidal trajectory parameter	
cycloid_radius	Radius of the wheel describing the cycloid

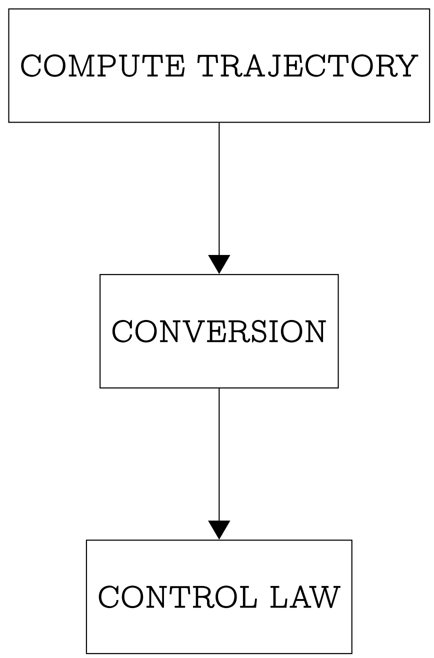
Controller parameters	
Kp	Proportional gain
Ki	Integral gain
Kd	Derivative gain
$FFWD$	Use velocity feedforward (1 = feed-forward used; 0 = feedforward NOT used)

Robot parameters	
$PL_{distance}$	Distance from the odometric centre of the robot to the selected point P

5.6.2 Node trajectory_tracker

The main loop executes three actions:

1. Computation of reference point for the desired trajectory
2. Translation of trajectory point from point L to point P
3. Computation of control action as virtual velocities



Compute Trajectory

$$t = t_{new} - t_{initial}$$

<u>Linear</u>

a_ coeff in configuration parameters is indicated as a_{circle}.

b_ coeff in configuration parameters is indicated as b_{circle}.

$$X_{ref} = a_{circle}t$$
$$dx_{ref} = a_{circle}$$

$$Y_{ref} = b_{circle} t$$
$$dy_{ref} = b_{circle}$$

<u>Parabolic</u>

PC := Parabola Convexity

$$X_{ref} = t$$
$$dx_{ref} = 1$$

$$Y_{ref} = PC \cdot \cos(wt)^2$$
$$dy_{ref} = 2 \cdot PC \cdot t$$

<u>Circle</u>

In parameters file parameter W is uppercase.

$$X_{ref} = R\cos(Wt)$$
$$dx_{ref} = -WR\sin(Wt)$$

$$Y_{ref} = R\sin(Wt)$$
$$dy_{ref} = WR\cos(Wt)$$

<u>Eight</u>

In parameters file parameter w is lowercase.

$$X_{ref} = a\sin(wt)$$
$$dx_{ref} = wa\cos(wt)$$

$$Y_{ref} = a\sin(wt)\cos(wt)$$

$$dy_{ref} = wa(\cos(wt)^2 - \sin(wt)^2)$$

Cycloidal

CR := Cycloid Radius

$$X_{ref} = CR(t - \sin(t))$$
$$dx_{ref} = CR - \cos(t)$$

$$Y_{ref} = CR(1 - \cos(t))$$
$$dy_{ref} = \sin(t)$$

Conversion

$$Xp_{ref} = X_{ref} + PL_{distance} \cdot \cos(\theta)$$
$$Yp_{ref} = Y_{ref} + PL_{distance} \cdot \sin(\theta)$$

Control Law

Compute position error

$$Xp_{err} = Xp_{ref} - Xp$$
$$Yp_{err} = Yp_{ref} - Yp$$

Difference between current time and initial one

$$\Delta T = T_{new} - T$$

Compute integral term

$$X_{int} = X_{int} + Xp_{err} \cdot \Delta T$$
$$Y_{int} = Y_{int} + Yp_{err} \cdot \Delta T$$

Compute derivative term

$$X_{der} = (Xp_{err} - Xp_{err\_old})/\Delta T$$
$$Y_{der} = (Yp_{err} - Yp_{err\_old})/\Delta T$$

PI + FFWD controller

$$V_{Xp} = FFWD \cdot dx_{ref} + K_p \cdot Xp_{err} + K_i \cdot X_{int} + K_d \cdot X_{der}$$
$$V_{Yp} = FFWD \cdot dy_{ref} + K_p \cdot Yp_{err} + K_i \cdot Y_{int} + K_d \cdot Y_{der}$$

5.6.3 Choiche of PID controller and parameters

5.7 Package CarCommndsFr

5.7.1 Intro

5.7.2 Configuration

Surface Parameters	
surface_type	Select surface type among 1:DRY, 2:WET, 3:SNOW, 4:ICE, 5:CUSTOM (parameters calculated with racecar values)

5.7.3 Launch

5.7.4 Node car_commands_fr

Compute surface type

Values are assigned to coefficients depending on the chosen ground type.

	B_x	B_y	C_x	C_y	D	E
Dry	10	10	1.9	1.9	1	0.97
Wet	12	12	2.3	2.3	0.82	1
Snow	5	5	2	2	0.3	1
Ice	4	4	2	2	0.1	1
Custom	0.21	0.017	1.65	1.3	-548	0.2

Compute desired speed

S = slip

V = |speed|

R_w = wheel radius

M = mass

Longitudinal Slip

$$\omega = \frac{V}{R_w}$$

Sempre 0?

$$S_{long} = \frac{V - R_w \omega}{V}$$

Lateral Slip

$$S_{lat} = \arctan\left(\frac{V_y}{V_x}\right)$$

Acceleration

Pacejka Magic Formula

$$y(x) = D \sin\{C \arctan[Bx - E(Bx - \arctan(Bx))]\}$$

$$a_{long} = \frac{y(S_{long})}{M}$$

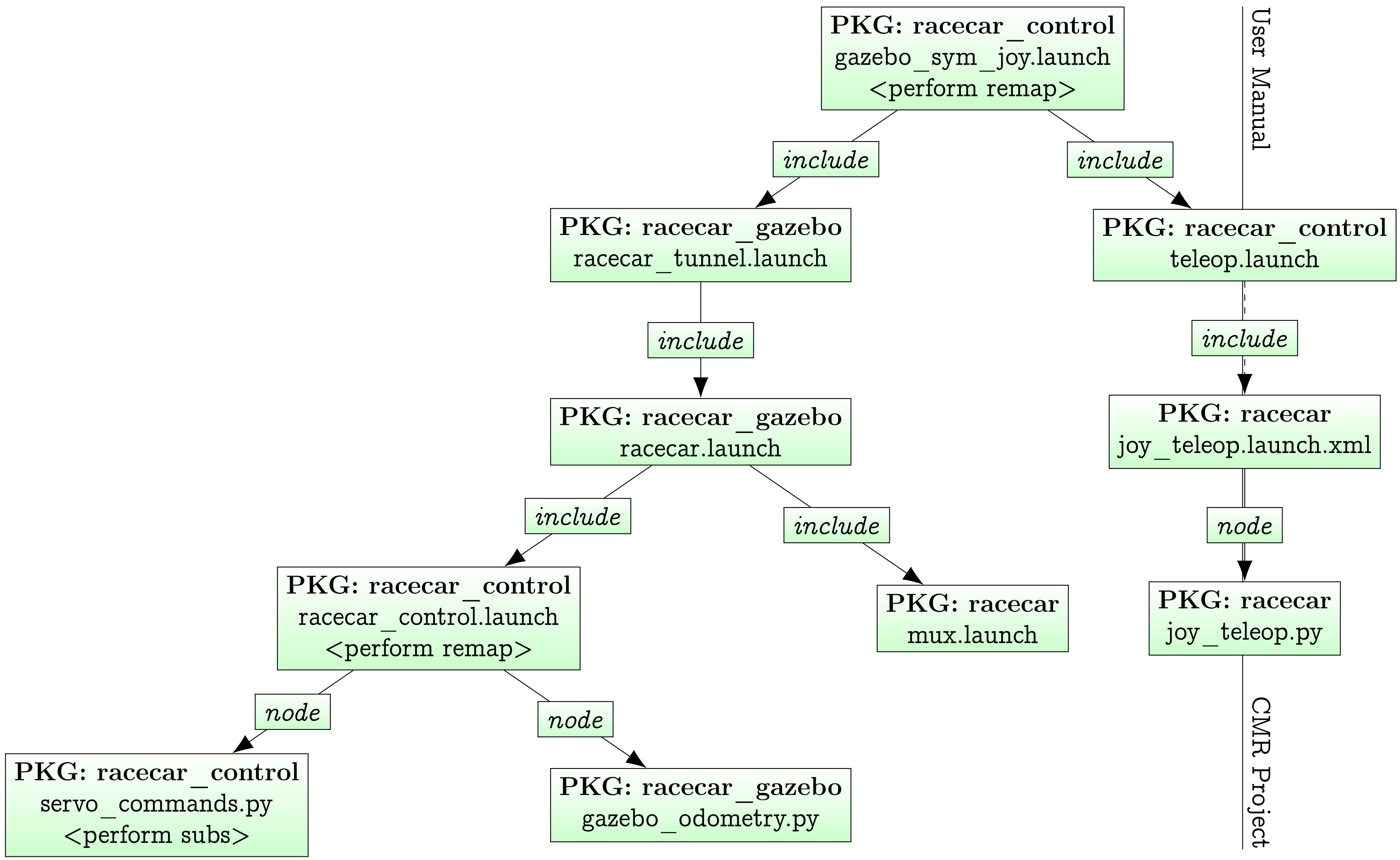
$$a_{lat} = \frac{y(S_{lat})}{M}$$

$$a = \sqrt[2]{a_{long}^2 + a_{lat}^2}$$

$$V_{desired} = \frac{\left(V + a\left(\frac{1}{100}\right)\right)}{0.1}$$

Appendix A

launch package inclusion


PKG: racecar_control
gazebo_sym_joy.launch
<perform remap>
include
include
PKG: racecar_gazebo
racecar_tunnel.launch
PKG: racecar_control
teleop.launch
include
include
PKG: racecar_gazebo
racecar.launch
PKG: racecar
joy_teleop.launch.xml
include
include
node
PKG: racecar_control
racecar_control.launch
<perform remap>
PKG: racecar
mux.launch
PKG: racecar
joy_teleop.py
node
node
PKG: racecar_control
servo_commands.py
<perform subs>
PKG: racecar_gazebo
gazebo_odometry.py

PKG: racecar_control
keyboard_teleop.py
<perform publish>

Package	File	Remap
racecar	joy_teleop.launch.xml	(none)
racecar	joy_teleop.py	(none)
racecar	mux.launch	(none)
racecar_control	gazebo_sim_joy.launch	REMAP /ackermann_cmd_mux/input/teleop TO /racecar/ackermann_cmd_mux/input/teleop
racecar_control	teleop.launch	(none)
racecar_control	racecar_control.launch	REMAP /racecar/ack/output TO /vesc/low_level/ack/output
racecar_control	servo_commands.py	SUBSCRIBE /racecar/ackermann_cmd_mux/output
racecar_control	keybpard_teleop.py	PUBLISH /vesc/achermann_cmd_mux/input/teleop
racecar_gazebo	racecar_tunnel.launch	(none)
racecar_gazebo	racecar.launch	(none)
racecar_gazebo	gazebo_odometry.py	(none)