



DEGREE PROJECT IN MECHANICAL ENGINEERING,  
SECOND CYCLE, 30 CREDITS  
*STOCKHOLM, SWEDEN 2020*

# **MPC based Caster Wheel Aware Motion Planning for Differential Drive Robots**

**JON ARRIZABALAGA  
AGUIRREGOMEZCORTA**




**KTH ROYAL INSTITUTE OF TECHNOLOGY  
SCHOOL OF INDUSTRIAL ENGINEERING AND MANAGEMENT**



---

# Abstract

---

 <b>KTH Industrial Engineering and Management</b>			<b>Master of Science Thesis ITM EX 2020:477</b>		
			<b>MPC based Caster Wheel Aware Motion Planning for Differential Drive Mobile Robots</b>		
			Jon Arrizabalaga Aguirregomezcorta		
Approved 2020-09-08		Examiner Lei Feng		Supervisor Tong Liu	
		Commissioner Robert Bosch GmbH		Contact person Niels van Duijkeren	

The inherited rotation in a caster wheel allows movement in any direction, but pays at the expense of reaction torques. When implemented in a mobile robot, these forces have a negative impact in its performance. One approach is to restrict rotations on the spot by attaching a filter to the output of the motion planner. However, this formulation compromises the navigation's completion in critical scenarios, such as parking, taking curves in narrow corridors or navigating at the presence of a high density of obstacles. Therefore, in this thesis we consider the influence of caster wheels in the motion planning stage, commonly presented as local planning.

This work proposes a Model Predictive Control (MPC) based local planner that integrates the caster wheel physics into the motion planning stage. A caster wheel aware term is combined with a reference tracking based navigation, which leads to the formulation of the Caster Wheel Aware Local Planner (CWAWLP). Since this method requires knowing the caster wheel's state and there is no sensor that provides this information, a caster wheel state observer is also formulated.

In order to evaluate the impact of the caster wheel aware term, CWAWLP is compared to a Caster Wheel based Agnostic Local Planner (CWAGLP) and a Caster Wheel based Agnostic Planner Local Planner with Path Filter (CWPFLP). After running simulations for three case studies in a virtual framework, two experimental case studies are conducted in an intra-logistics robot. These are evaluated according to the navigation's quality, motor torque usage and energy consumption.

According to the patterns observed in the evaluation, CWAWLP covers a longer distance than CWAGLP without decreasing the navigation's quality. At the same time, its motor torques are similar to the ones of CWPFLP. Therefore, CWAWLP is capable of considering caster wheel physics without sacrificing navigation capabilities. The formulated caster wheel aware term is compatible with any MPC based navigation algorithm and inherits the derivation of an observer capable of estimating caster wheel rotation angles and rolling speeds. Even if the caster wheel awareness has been implemented in a differential driven robot, this approach is also applicable to vehicles with an alternative drivetrain, such as car-like robots.

## **Keywords**


Mobile robots, differential drive, caster wheels, motion planning, MPC



---

# Sammanfattning

---

 <div><b>Examensarbete ITM EX 2020:477</b></div> <div><b>MPC-baserad Rörelseplanering med Integrerat Stöd för Svängbara Länkhjul Avsedd för Robotar med Differentialdrift</b></div> <div><b>Jon Arrizabalaga Aguirregomezorta</b></div>		
Godkänt 2020-09-08	Examinator Lei Feng	Handledare Tong Liu
	Uppdragsgivare Robert Bosch GmbH	Kontaktperson Niels van Duijkeren

Den ärvda rotationen i ett hjul möjliggör rörelse i vilken riktning som helst, men fås på bekostnad av reaktionsmoment. När de implementeras i en mobil robot har dessa krafter en negativ inverkan på dess prestanda. Ett tillvägagångssätt är att begränsa rotationer på plats genom att applicera ett filter på rörelseplanerns utgång. Denna formulering komprometterar dock navigeringens slutförande i kritiska scenarier, såsom parkering, kurvor i smala korridorer eller navigering i närheten av höga hinder. Därför beaktar vi i denna avhandling påverkan av hjul på hjulplaneringen, som ofta presenteras som lokal planering.

Detta arbete föreslår en Model Predictive Control (MPC) -baserad lokal planerare som integrerar svängbara länkhjuls fysik i rörelseplaneringsstadiet. En kugghjulmedveten term kombineras med en referensspårningsbaserad navigering, vilket leder till formuleringen av Caster Wheel Aware Local Planner (CWAWLP). Eftersom denna metod kräver kunskap om svängbara länkhjuls tillstånd och det inte finns någon sensor som ger denna information, formuleras också en hjulhjulstillståndsobserverator.

För att utvärdera effekten av det medvetna begreppet svängbara länkhjul jämförs CWAWLP med en Caster Wheel-baserad Agnostic Local Planner (CWAGLP) och en Caster Wheel-baserad Agnostic Planner Local Planner with Path Filter (CWPFLP). Efter att ha kört simuleringar för tre fallstudier i ett virtuellt ramverk genomförs två experimentella fallstudier i en intra-logistikrobot. Dessa utvärderas enligt navigeringens kvalitet, vridmomentanvändning och energiförbrukning.

Enligt de mönster som observerats i utvärderingen når CWAWLP ett längre avstånd än CWAGLP utan att sänka navigeringens kvalitet. Samtidigt liknar motorns vridmoment dem som CWPFLP. Därför kan

CWAWLP ta hänsyn till svängbara länkhjuls fysik utan att offra navigationsfunktionerna. Den formulerade medhjulningsmedveten termen är kompatibel med vilken MPC-baserad navigationsalgoritm som helst och ärver härledningen av en observatör som kan uppskatta hjulets rotationsvinklar och rullningshastigheter. Även om hjulhjälpmmedvetenheten har implementerats i en differentierad robot, är detta tillvägagångssätt också tillämpligt på fordon med ett alternativt drivsystem, såsom billiknande robotar.

## **Nyckelord**

Mobila robotar, differentiell drivning, hjul, rörelseplanering, MPC

## Title

MPC based Caster Wheel Aware Motion Planning for Differential Drive Robots (ENG)  
MPC-baserad Rörelseplanering med Integrerat Stöd för Svängbara Länkhjul Avsedd för Robotar med Differentialdrift (SWE)

## Author

Jon Arrizabalaga (jonarr@kth.se)  
Master of Science Engineering Design - Mechatronics  
KTH Royal Institute of Technology

## Examiner

Prof. Lei Feng  
lfeng@kth.se  
Machine Design - Mechatronics and Embedded Control Systems

## Academic supervisor

Phd Candidate Tong Liu  
tongliu@kth.se  
Machine Design - Mechatronics and Embedded Control Systems

## Industrial supervisors

Dr. Niels Van Duijkeren  
Niels.VanDuijkeren@de.bosch.com  
Corporate Research - Advance Autonomous Systems (CR/AAS4)

Dr. Ralph Lange  
ralph.lange@de.bosch.com  
Corporate Research - SW Systems Engineering (CR/AEE1)

## Place for Project

Robert Bosch GmbH Zentrum für Forschung und Vorauentwicklung  
Corporate Research - Advance Autonomous Systems (CR/AAS4)  
Robert Bosch Campus, Renningen, Germany, 71272

KTH Royal Institute of Technology  
Machine Design - Mechatronics and Embedded Control Systems  
Brinellvägen 83, Stockholm, Sweden, 10044



**BOSCH**  
Invented for life





---

# Acknowledgements

---

I would like to express my gratitude to the CR/AAS department in Robert Bosch GmbH for giving me the chance to write this thesis and providing all the tools and resources needed to complete this work. The spirit for innovation and progress, combined with the amount of expertise that concentrated at Renningen's campus for Corporate Research, have been a major motivating factor.

More specifically, I would like to thank Dr. Niels van Duijkeren and Dr. Ralph Lange for the weekly meetings and constant support. The challenges that you proposed in a socratic manner, where you made me doubt about every assumption, have encouraged me to push the limits. Besides the technical knowledge, your mentorship has brought in the skepticism needed to improve the quality of this thesis. None of this would have been possible without your guidance.

I would also like to appreciate Dr. Oliver Lenord's help for introducing me to Modelica language and explaining the previous work in an early stage prototype of the Active Shuttle.

Department students and interns also played a major role in this adventure, as they contributed to an exceptional working environment where lunch times and coffee breaks always converged into vivid conversations about random topics.

I cannot forget about the Machine Design department at KTH - Royal Institute of Technology for taking charge of this thesis. In particular, I would like to thank PhD Candidate Tong Liu for supervising this work and Prof. Lei Feng for being its examiner. Your immediate assistance, combined with your flexibility when help was needed, have allowed me to tailor the content's outline and maximize the learning outcome.

Last but not least, I want to give credit to every friend / student / colleague I have met during this beautiful journey called Masters. The encouragement I have received through talking to each of you has been the most valuable lecture I have taken during these last two years. The energy that you transmit has been the fuel for all the hard work. Thank you.

Of course, thanks also to my family, mom, dad and brother, for supporting me and giving feedback regardless of the circumstances.

KTH - Royal Institute of Technology  
August 4, 2020



Jon Arrizabalaga Aguirregomezcorta



---

# Contents

---

<b>Abstract</b>	<b>i</b>
<b>Acknowledgements</b>	<b>vii</b>
<b>List of Figures</b>	<b>xiii</b>
<b>List of Tables</b>	<b>xvii</b>
<b>Nomenclature</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Problem statement . . . . .	4
1.3 Purpose . . . . .	6
1.4 Methodology . . . . .	6
1.5 Delimitations and Limitations . . . . .	7
1.5.1 Delimitations . . . . .	7
1.5.2 Limitations . . . . .	7
1.6 Deposition . . . . .	8
<b>2 Fundamentals</b>	<b>9</b>
2.1 Modelling of DDMR with Caster Wheels . . . . .	9
2.1.1 DDMR . . . . .	9
2.1.2 Caster Wheels . . . . .	9
2.2 Caster wheel awareness in mobile robots . . . . .	10
2.2.1 Formulation . . . . .	11
2.2.2 Implementation . . . . .	11
2.3 MPC based motion planning . . . . .	11
2.3.1 Path following . . . . .	12
2.4 Case Identification . . . . .	13
2.4.1 Hypotheses statement . . . . .	13
2.4.2 Hypotheses simulation . . . . .	15
2.4.3 Hypotheses implication . . . . .	16

2.5	Summary . . . . .	16
<b>3</b>	<b>Concept</b>	<b>19</b>
3.1	Plant Model . . . . .	19
3.1.1	Differential Drive Mobile Robot . . . . .	19
3.1.2	Caster Wheel . . . . .	23
3.1.3	Formulation of the plant model . . . . .	25
3.2	Objective function . . . . .	26
3.2.1	Navigation term . . . . .	26
3.2.2	Caster wheel aware term . . . . .	28
3.2.3	Formulation of objective function . . . . .	31
3.3	Observer . . . . .	31
3.3.1	Observer model . . . . .	32
3.3.2	Stability analysis . . . . .	32
3.4	Horizon and sampling time . . . . .	37
3.4.1	Sampling time . . . . .	37
3.4.2	Time and control horizon . . . . .	38
3.5	Constraints . . . . .	38
3.6	Formulation of OCP . . . . .	39
3.7	Implementation of the MPC . . . . .	40
3.7.1	Integration . . . . .	40
3.7.2	OCP solver . . . . .	40
3.8	Extension of PF . . . . .	40
3.9	Summary . . . . .	41
<b>4</b>	<b>Simulations</b>	<b>43</b>
4.1	Observer . . . . .	43
4.1.1	Case study I: Navigation across a global path . . . . .	43
4.1.2	Case study II: Forward-Backward case . . . . .	46
4.2	Comparison of MPC based local planners . . . . .	47
4.2.1	Procedure . . . . .	47
4.2.2	Evaluation criteria . . . . .	47
4.2.3	Case Study I: Rotation on the spot . . . . .	49
4.2.4	Case Study II: Rotate and navigate in a straight line . . . . .	52
4.2.5	Case Study III: Navigation across given global paths . . . . .	55
4.3	Summary . . . . .	59
<b>5</b>	<b>Field Test</b>	<b>61</b>
5.1	Definition of case studies . . . . .	61
5.2	Experimental setup . . . . .	62
5.2.1	Layout . . . . .	62
5.2.2	Network . . . . .	63
5.2.3	Navigation algorithm . . . . .	64



5.2.4	Implementation . . . . .	65
5.3	Case study I: Rotation on spot . . . . .	66
5.3.1	Path Filter . . . . .	67
5.3.2	MPC based local planners . . . . .	68
5.3.3	Conclusions . . . . .	68
5.4	Case study II: Navigation across a given global path . . . . .	69
5.4.1	Validation of the observer . . . . .	69
5.4.2	Results . . . . .	70
5.4.3	Analysis . . . . .	72
5.4.4	Conclusions . . . . .	73
5.5	Summary . . . . .	74
<b>6</b>	<b>Conclusions</b>	<b>75</b>
6.1	Discussion . . . . .	75
6.2	Summary . . . . .	76
6.3	Future Work . . . . .	78
6.3.1	Research Tasks . . . . .	78
6.3.2	Research Directions . . . . .	78
	<b>References</b>	<b>81</b>
<b>A</b>	<b>Figures</b>	<b>85</b>
A.1	Fundamentals - Case Identification . . . . .	85
A.2	Experiments - Sensors . . . . .	86
<b>B</b>	<b>Scripts</b>	<b>87</b>
B.1	Local planner . . . . .	87
B.2	MPC local planner . . . . .	100
B.3	Navigation local planner . . . . .	106
B.4	Caster wheel estimator . . . . .	110
B.5	Caster wheel based Path Filter . . . . .	113



---

# List of Figures

---

1.1	Comparison of an AGV against a SDV. The SDV can adapt to changes in the environment, while the AGV relies on a set of instructions that obliges it to follow a fixed path. If an obstacle appears in its way, a human being needs clear the path, so that the AGV can keep driving. . .	2
1.2	Application cases for AS SDV produced by Bosch Rexroth AG. Source: Bosch Rexroth AG . .	2
1.3	Layout of the PF. It is introduced between the motion planner and the motor controller. . . .	3
1.4	Comparison of trajectories and motor currents when applying PF . . . . .	4
1.5	Robot's trajectory when following a L shape global path with a 90°clockwise rotation within a narrow corridor without (continuous) and with (dashed) PF. In the latter case, the robot collides with the corridor's wall. . . . .	5
1.6	Outline of the thesis. . . . .	8
2.1	Caster wheel's geometrical and bore torque representations - Source: [5] . . . . .	10
2.2	Description of PF's working principle - Source: [5] . . . . .	11
2.3	Identification of caster wheel reaction torques. Load sensitivity analysis for a cornering case of 90° while standing still. The velocity commands of the case study are given in Figure 2.3d. . . . .	14
2.4	Rolling speed of the left front caster-wheel for plant (kinematics) and OpenModelica (150 Kg and 250 Kg) models applicable to a cornering case of 90° while standing still (Figure 2.3d) . . . . .	15
2.5	Load sensitivity for a cornering case of 90° while standing still without bore frictions. . . . .	16
3.1	Diagram of the robot. Definition of global and local coordinate systems: $\{x^1, y^1, z^1\}$ and $\{x^2, y^2, z^2\}$ . Dimensions $\Delta x_{cw}, \Delta y_{cw}$ represent the distance from the robot's origin to the left front caster-wheel. $\Delta y_{dw}$ refers to the distance from the robot's origin to the driven wheels. . . . .	20
3.2	Model of the robot's motor controller in OpenModelica. . . . .	21
3.3	Diagram of the caster wheel. Point 0 is the robot's origin, $A$ is the linkage between robot-frame and overhang, $B$ is the connection between overhang and caster-wheel, and $C$ is the contact point between caster-wheel and environment. . . . .	23
3.4	Procedure to implement a virtual time based reference in a L shape global path. Notice that the reference given in Step 3 is a simplified version and it is intended only for illustration. . . .	27

3.5	Example of a <i>trajectory</i> given by the global planner that has been divided into <i>sections</i> , according to its <i>goal-points</i> and <i>check-points</i> . At the same time, a <i>section</i> contains several <i>lines</i> . This division dictates the behaviour of the time-based reference that the robot tries to follow. . . . .	28
3.6	Caster wheel rotation angles, $\varphi$ (blue and orange) and their respective steady state angles (green and red). Notice that the steady state values are specific for each set of input commands	30
3.7	Graphic of $\text{atan2}$ . Notice that its domain is $[-\pi, \pi]$ and it is discontinuous in the origin. . . .	30
3.8	Implementation of the estimator. . . . .	31
3.9	Possible equilibrium front-left caster-wheel angles, $\bar{x}$ , depending on longitudinal and lateral velocity commands, $v, \omega$ . . . . .	33
3.10	Green area represents front-left caster-wheel angle's equilibrium range, $\bar{\varphi} = \bar{x}$ , for any set of velocity commands $(v, \omega)$ that are within the limits . . . . .	34
3.11	Comparison of convergence for two estimators, without (blue) and with (green) oscillations, initialized at $\pi$ rad. Orange line represents the estimator's convergence with no oscillations, but real velocities as inputs. . . . .	36
3.12	Eigenvalues classified according to its respective longitudinal and rotation velocities, $V, W$ and caster wheel's rotation angle, $\varphi$ . The colors divide the eigenvalues depending on its speed. . .	38
4.1	Case-study for caster-wheel angle estimator's analysis . . . . .	43
4.2	Estimation value and error of left-front caster-wheel angle . . . . .	44
4.3	Estimation value and error of front-left caster-wheel under short disturbances . . . . .	44
4.4	Estimation value and error of front-left caster-wheel under long disturbances . . . . .	45
4.5	Estimation value and error of front-left caster wheel for different loads . . . . .	46
4.6	Estimation value and error of front-left caster-wheel for different loads in "forward-backward" case . . . . .	46
4.7	Trajectory deviations, $d_{\text{error}}$ , applied to all the points obtained from the robots trajectory when implementing CWAFLP to trajectories for "Navigation" simulation set. $d_{\text{error}}$ has been colored depending on its respective line in the global trajectory. . . . .	48
4.8	Comparison of front caster wheel angles, $\varphi_{\text{LF}}$ and $\varphi_{\text{LB}}$ , for "aligned" (discontinuous line) and "misaligned" (continuous) cases applied to a rotation on the spot with caster wheel agnostic MPC based local planner. . . . .	49
4.9	Performance when rotating on the spot with a caster wheel agnostic MPC local planner for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles $\varphi$ . . .	50
4.10	Performance when rotating on the spot with a caster wheel agnostic MPC local planner and a path filter for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles $\varphi$ . . . . .	51
4.11	Performance when rotating on the spot with a caster wheel aware MPC local planner for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles $\varphi$ . . .	52
4.12	Comparison of front caster wheel angles, $\varphi_{\text{LF}}$ and $\varphi_{\text{LR}}$ when rotating on the spot with different local planners . . . . .	52
4.13	Performance with CWAGLP (red), CWPFLP (green), CWAFLP (blue) for rotate and navigate case study. . . . .	53

4.14	Global paths for analyzing MPC based local planners performance when navigating. They are characterized for making the robot turn on the spot and being torque demanding. . . . .	55
4.15	Performance when navigation through global path 1 with CWAGLP (red), CWPFLP (green), CWAUWLP (blue) . . . . .	56
4.16	Performance when navigation through global path 2 with CWAGLP (red), CWPFLP (green), CWAUWLP (blue). . . . .	57
4.17	Comparison of OCP's cost value for global path 2 for differen planners. . . . .	59
5.1	Experimental setup for performing case studies on hardware. . . . .	62
5.2	Network of experimental setup for performing case studies on hardware. . . . .	63
5.3	Reflective markers attached to the robot's upper surface for tracking its pose. . . . .	65
5.4	Assembly of measurement system for caster wheel rotation angle, $\varphi$ , at AS's rear axle. . . . .	66
5.5	Comparison of velocity commands and torques in AS when rotating on the spot with and without PF. . . . .	67
5.6	Comparison of velocity commands and torques in Active Shuttle when rotating on the spot for CWAGLP and CWAUWLP. . . . .	68
5.7	Global path for conducting experiments to compare CWAGLP, CWPFLP and CWAUWLP. . . . .	69
5.8	Analysis of measurements (continuous) and estimations (dashed) of rear caster wheel rotation angles $\varphi$ . Rear left (BL) is shown in blue and rear right (BR) in red. . . . .	70
5.9	Comparison of CWAGLP (blue), CWPFLP (orange) and CWAUWLP (green) in Active Shuttle when navigating. The line with a stronger colour represents average value. Maximum values are given by the shade. . . . .	71
5.10	Comparison of applying CWPFLP in the front left (FL) or front-right (FR) in the AS when navigating. The line with a stronger colour represents average value. Maximum values are given by the shade. . . . .	72
A.1	Caster wheel angles, motor torques and bore torques for a cornering case of $90^\circ$ with longitudinal velocity of $0.1m/s$ (Figure A.1d) with respect to kinematics plant model. . . . .	85
A.2	Caster wheel angles, motor torques and bore torques for a cornering case of $90^\circ$ with longitudinal velocity of $0.3m/s$ (Figure A.2d) with respect to kinematics plant model. . . . .	86
A.3	Assembly of encoder to measure rear axle caster wheel rotation angles. . . . .	86



---

# List of Tables

---

2.1	Time difference between caster wheel angle transitions, maximum motor and bore torques depending on load carried by the robot and different longitudinal speeds when rotating in a 90° corner and carrying a load of 150Kg and 250Kg. . . . .	16
4.1	Measurements to evaluate navigation's performance when simulating CWAGLP (agnostic), CWPFLP (path filter) and CWAUPL (aware) for "rotate and navigate" case study . . . . .	54
4.2	Measurements to evaluate motor torques usage when simulating CWAGLP (agnostic), CWPFLP (path filter), CWAUPL (aware) for navigation after rotate and navigate case-study. . . . .	54
4.3	Measurements to evaluate navigation's performance when simulating CWAGLP (agnostic), CWPFLP (path filter) and CWAUPL (aware) for two different trajectories. . . . .	57
4.4	Measurements to evaluate motor torques usage when simulating CWAGLP (agnostic), CWPFLP (path filter), CWAUPL (aware) for two different trajectories. . . . .	58
5.1	Measurements to evaluate motor torques usage when rotating on the spot with and without PF in the Active Shuttle. . . . .	67
5.2	Measurements to evaluate motor torques usage when rotating on spot with CWAGLP (agnostic) and CWAUPL (aware) in the Active Shuttle. . . . .	68
5.3	Measurements to evaluate navigation's performance when applying CWAGLP (agnostic), CWPFLP (path filter) and CWAUPL (aware) for navigation case study in the AS. . . . .	72
5.4	Measurements to evaluate motor torques usage when applying CWAGLP (agnostic), CWPFLP (path filter), CWAUPL (aware) for navigation case-study in the AS. . . . .	73
5.5	Energy consumption when applying CWAGLP (agnostic), CWPFLP (path filter), CWAUPL (aware) for navigation case-study in the Active Shuttle. . . . .	73





---

# Nomenclature

---

## List of Abbreviations

<b>AGV</b>	Automated Guided Vehicle
<b>AMS</b>	Active Shuttle Management System
<b>AS</b>	Active Shuttle
<b>Bosch</b>	Robert Bosch GmbH
<b>CWAGLP</b>	Caster wheel agnostic MPC based local planner
<b>CWAWLP</b>	Caster wheel aware MPC based local planner
<b>CWPFLP</b>	Caster wheel agnostic MPC based local planner with path filter
<b>DDMR</b>	Differential Drive Mobile Robot
<b>DWA</b>	Dynamic Window Approach
<b>EB</b>	Elastic Band
<b>FMI</b>	Functional Mock-up Interface
<b>FMU</b>	Functional Mock-up Unit
<b>LB</b>	Left Back caster wheel
<b>LD</b>	Left Driven caster wheel
<b>LF</b>	Left Front caster wheel
<b>MAE</b>	Mean Absolute Error
<b>MCU</b>	Motor Control Unit
<b>MPC</b>	Model Predictive Control
<b>NLP</b>	Nonlinear problem
<b>OCP</b>	Optimal Control Problem
<b>ODE</b>	Ordinary differential equation
<b>PF</b>	Caster wheel based Path Filter

<b>RB</b>	Right Back caster wheel
<b>RCU</b>	Robot Control Unit
<b>RD</b>	Right Driven caster wheel
<b>RF</b>	Right Front caster wheel
<b>RMSE</b>	Root Mean Squared Error
<b>ROS</b>	Robot Operating System
<b>SCU</b>	Safety Control Unit
<b>SDV</b>	Self Driving Vehicle
<b>SSH</b>	Secure Shell
<b>TEB</b>	Time Elastic Band
<b>WMR</b>	Wheeled Mobile Robots

## List of Symbols

$\alpha$	Rotational acceleration of the robot's origin	rad/s <sup>2</sup>
$\Delta x_{cog}$	Distance in $x$ coordinate from robot's origin to center of mass	m
$\Delta x_{cw}$	Distance in $x$ coordinate from robot's origin to front-left caster-wheel	m
$\Delta y_{cw}$	Distance in $y$ coordinate from robot's origin to front-left caster-wheel	m
$\Delta y_{dw}$	Distance in $y$ coordinate from robot's origin to driven wheels	m
$\dot{\gamma}$	Rolling speed of caster wheel	rad/s
$\omega$	Rotational velocity of robot's origin	rad/s
$\theta$	Orientation of robot	rad
$\varphi$	Rotation angle of caster wheel	rad
$a$	Longitudinal acceleration of the robot's origin	m/s <sup>2</sup>
$F$	Plant model of the robot	
$H$	Constraints in the OCP	
$h_{cw}$	Height of caster wheel. Distance in $z$ coordinate from joint to rolling center	m
$I$	Inertia of the robot	Kg · m <sup>2</sup>
$i$	Motor current	A
$J$	Cost function in the OCP	
$M$	Mass of the robot	Kg
$N$	Control horizon of the MPC	

$p_x$	Position of robot in $x$ coordinate	m
$p_y$	Position of robot in $y$ coordinate	m
$Q_{cw}$	Weight for caster wheel aware term in cost function	-
$Q_{nav}$	Weight for navigation term in cost function	-
$Q_u$	Weight for input energy in cost function	-
$r_{cw}$	Radius of caster wheel	m
$r_{dw}$	Radius of driven wheel	m
$T$	Time horizon of the MPC	
$T_L$	Left motor torque	Nm
$T_R$	Right motor torque	Nm
$v$	Longitudinal velocity of robot's origin	m/s
$tr$	Trail of caster wheel. Distance in $x$ coordinate from joint to rolling center	m

## List of Indices

$(\cdot)_{cw}$	Value respective to caster wheels
$(\cdot)_{des}$	Desired value respective to velocity commands obtained from the motion planner before PF
$(\cdot)_{dw}$	Value respective to driven wheels
$(\cdot)_{filt}$	Value after being filtered by PF
$(\cdot)_{ss}$	Steady State value
$\hat{(\cdot)}$	Estimation value
$\overline{(\cdot)}$	Equilibrium value



## Introduction

---

This Chapter presents the need to look at caster wheel awareness in mobile robotics. A study of the tendencies in intra-logistic robots, along with the limitations of previous work leads to the formulation of a research question. The methodology applied for answering this question is followed by a summary of delimitations and limitations. To ease the report 's understanding, the outline of the thesis is given as a closure.

### 1.1 Background

Reductions in hardware costs, along with an increase of expertise in a variety of research fields, have led to an industrial revolution, also known as industry 4.0. Among others, this transformation is characterized by adding flexibility to the transport of goods and resources between workstations within factories or warehouses. Such strategies result in shorter lead times, tighter links between supply and demand, accelerating new product introduction and simplifying the manufacture of highly customized products. These advantages have put the focus on exploring alternative methods to fixed conveyor belts, [1].

During the past years Automated Guided Vehicles (AGVs) have played the main role in intra-logistic purposes. Since they rely on external guidance devices, such as magnetic tape, beacons, barcodes or predefined laser paths to navigate on predetermined trajectories, they cannot deal with uncertain scenarios. If the environment is manipulated and an obstacle appears along the path, an AGV needs to wait for a human being to remove it. Therefore, AGVs require infrastructure maintenance and do not provide the flexibility required for adjusting the flow of goods to the needs of the factory. The ability of SDVs to navigate freely without relying on fixed trajectories combined with the capacity to handle unexpected circumstances, such as obstacle avoidance, makes them a more appropriate solution than fixed conveyor belts or AGVs (see Figure 1.1). In fact, it is estimated that, in the upcoming years SDVs will become the main intra-logistics transport method. [2].

Such a SDV, called Active Shuttle (AS) [3], has recently been launched to the market by Bosch Rexroth AG, a wholly owned subsidiary of Robert Bosch GmbH (see Figure 1.2). It automates the internal flow of goods and materials depending on the supply chain's needs. Several units can be coordinated by a traffic management system that optimizes the assignation of tasks and maximizes the transport's efficiency. Since

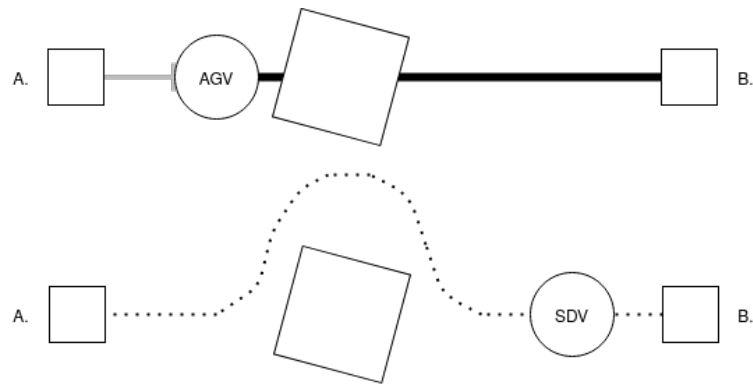
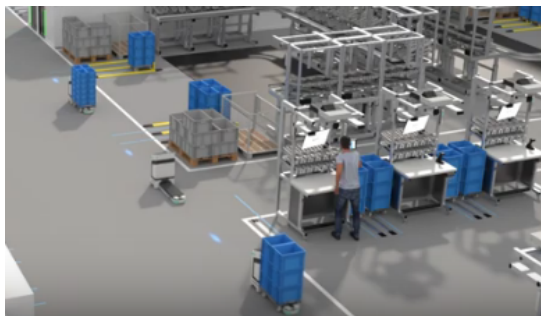
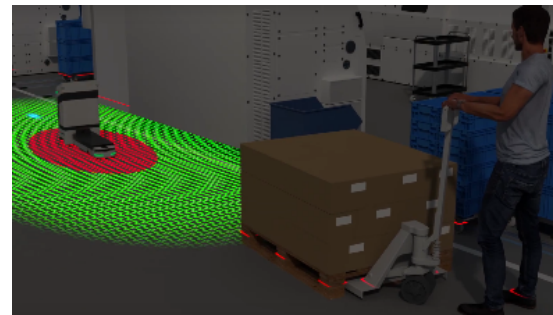


Figure 1.1: Comparison of an AGV against a SDV. The SDV can adapt to changes in the environment, while the AGV relies on a set of instructions that obliges it to follow a fixed path. If an obstacle appears in its way, a human being needs clear the path, so that the AGV can keep driving.

its implementation only requires the factory's reference map, the fleet's logistic routes automatically adapt to changes in infrastructure or modifications in the factory's layout. Laser based navigation allows the AS to safely drive around moving obstacles and share the operation area with other vehicles or human workers [2].



(a) Multiple ASs navigating through a factory



(b) AS navigating at the presence of dynamic obstacles



(c) Front view of AS operating in a real factory



(d) Side view of AS operating in a real factory

Figure 1.2: Application cases for AS SDV produced by Bosch Rexroth AG. Source: Bosch Rexroth AG

The AS is propelled by two electric motors, each of them coupled to a gearbox and a wheel. Robots with such a drive-train are known as Differential Drive Mobile Robots (DDMR). Because each wheel is independently actuated, the robot's heading direction depends on the ratio of the wheel's angular speeds and, consequently, an additional steering motion is not required. The high maneuverability combined with the compactness of this assembly accounts for its high presence in research and industry [4].

To be able to transport heavier loads, the weight of the load must be distributed across multiple points of

contact. That is why, the AS combines a differential drive system with four caster wheels. This results in a six-wheeled robot scattered in three rows. The driving wheels are located in the middle whilst the caster wheels are in the corresponding corners. Their ability to rotate freely allows movement in any direction, but generates reaction torques in the joints with the chassis. One can perceive this phenomena when pushing a supermarket trolley or moving an office chair. Even though both items can maneuver in any direction, in some scenarios, opposition arises.

When implementing caster wheels in mobile robots, reaction torques also arise and have severe consequences in the navigation's performance. Depending on the robot's geometry and power, the influence of the caster wheels can be critical. Firstly, unwanted orientations can cause blocking scenarios, where either the driving wheels spin, or the motors cannot provide sufficient force. Secondly, the high inertias involved when transporting heavy and bulky loads combined with the reaction forces generated in the caster wheels lead to abrupt motions degrading the performance of the components.

Due to these forces, not only torque demand on the actuators increases, but also the local planner needs to deal with unidentified disturbances. As a consequence, the lifetime of the motors, along with the navigation's quality, are compromised. On top of that, since higher torques are required for covering the same distance, more energy might be consumed [5].

The effects of caster wheels in mobile robots have been previously studied in paper [5], where its contributions are applied and tested in an early stage prototype of the AS. After modelling this vehicle in OpenModelica [6] and identifying rotations on the spot as the most critical motions regarding caster wheel reaction torques, it proposes a solution, denoted as Caster Wheel based Path Filter (PF).

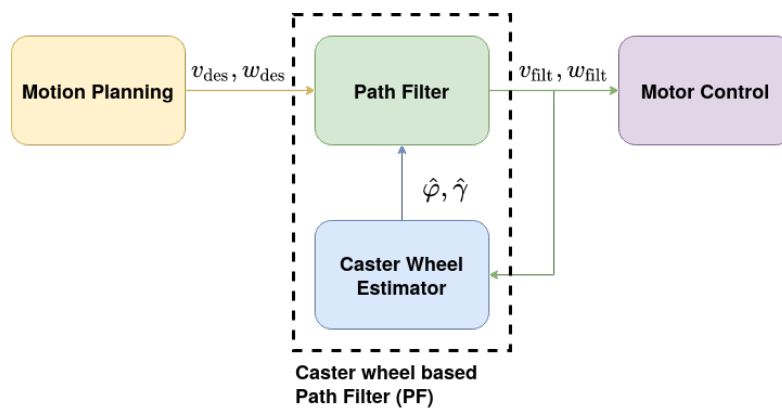


Figure 1.3: Layout of the PF. It is introduced between the motion planner and the motor controller.

This method consists on avoiding rotations on the spot when the robot is navigating. To this end, a filter combined with an estimator are introduced between the motion planner and motor controller (see Figure 1.3). In this way, velocity commands obtained from the motion planner are filtered by considering caster wheel state estimations. Instead of turning on the spot, the PF ensures that the caster wheels are rolling by converting pure rotational velocity commands into a combination of longitudinal and rotational commands, as depicted in Figure 1.4a. This ensures that there are no sudden changes in the caster wheels orientation and, consequently, the reaction torques generated in the caster wheels are not relevant to the robot's performance (see Figure 1.4b).

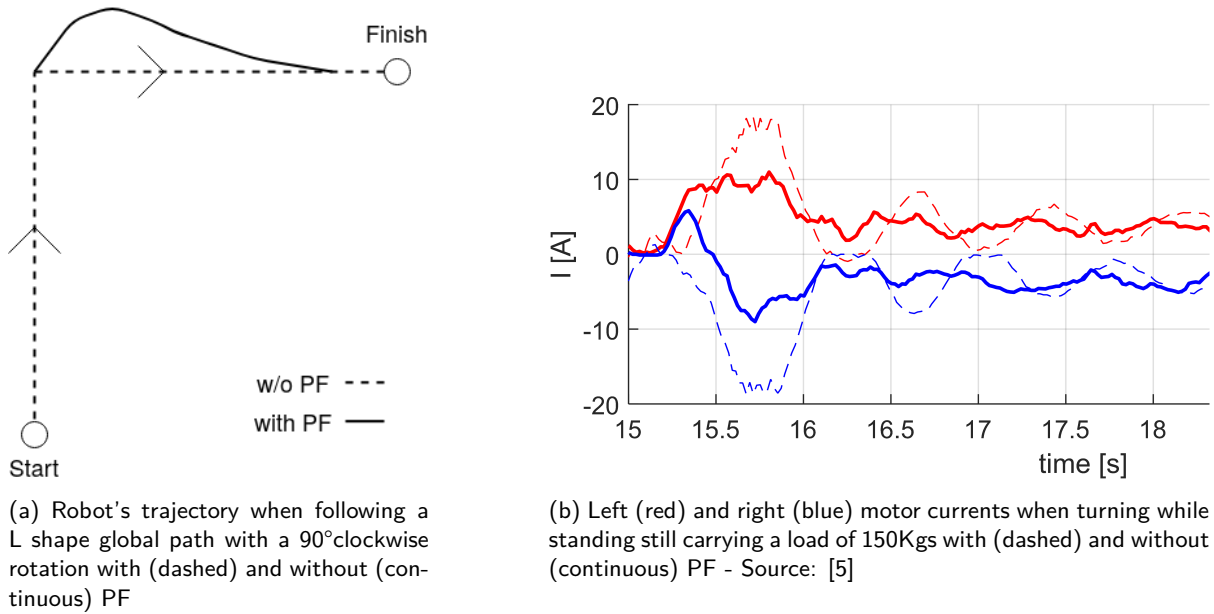


Figure 1.4: Comparison of trajectories and motor currents when applying PF

The resultant behavior of the robot ensures that, when the PF is activated, rotations with low longitudinal velocity commands are avoided, minimizing the influence of the reaction torques in the performance of the robot. Figure 1.4a shows an example where the robot needs to follow a global path that involves a 90°corner. It illustrates how does the robot's behavior change when attaching the PF to the output of the motion planner. Instead of rotating on the spot, the robot increases the radius of the curve. In this way, the caster wheels are rolling when there is an offset between the driving direction and the caster wheel's orientation. The respective impact on the motor currents is given in Figure 1.4b, where implementing the PF causes a decrease in amplitude and oscillations.

## 1.2 Problem statement

Despite the capability of the PF to mitigate reaction torques, its potential is limited by the fact that it is not integrated into the motion planner. Consequently, it modifies velocity commands without taking other constraints into account. This might have severe consequences in scenarios where rotations on the spot are required, such as parking or navigating in areas with limited free space. In these cases, not being able to rotate on the spot might lead to getting stuck or colliding with surrounding obstacles.

In order to illustrate this, the example shown in Figure 1.4a has been extended by limiting the available space to a narrow corridor (see Figure 1.5). Since the PF is not aware of the existence of obstacles, the filtering will not vary, increasing the radius of the curve and causing a collision against the corridor's wall. Even if Figure 1.4b demonstrates that the PF succeeds in minimizing caster wheel reaction torques, it cannot guarantee that the navigation will be completed. Consequently, there is a need of introducing the caster wheel awareness in the motion planning stage.

Motion planning is concerned with finding of a collision free trajectory that respects the kinematic and dynamic motion constraints [7]. If optimality is also desired, there needs to be a trade-off between control effort, control error or transition time between start and goal pose. Therefore, many researchers are focusing



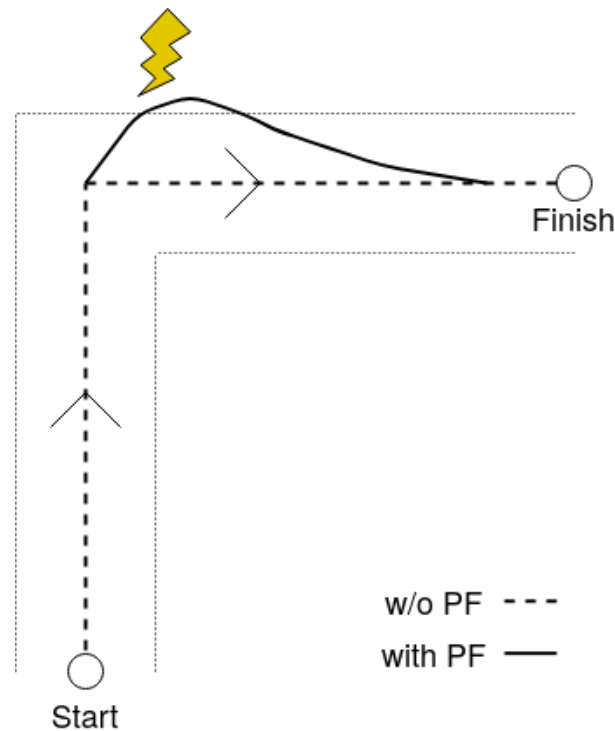


Figure 1.5: Robot's trajectory when following a L shape global path with a 90°clockwise rotation within a narrow corridor without (continuous) and with (dashed) PF. In the latter case, the robot collides with the corridor's wall.

on obtaining solutions or even approximations of the underlying trajectory optimization problem efficiently [8]. In this field, three common methods are Dynamic Window Approach (DWA), Elastic Band (EB) and Time Elastic Band (TEB).

DWA is a widely applied method for mobile robot navigation [9]. It samples simulated trajectories for a specific velocity search space according to a feasible velocity set. The control action is kept constant throughout the horizon and, consequently, complex motions, such as reversals, are not possible with this technique. After making predictions for all samples, the optimal candidate is selected on the basis of a cost function and desired constraints. Depending on the restrictions applied to the velocity set, computational efficiency can be achieved at the expense of obtaining sub-optimal results.

An alternative approach is EB [10], where the global path is deformed online according to internal and external forces that vary depending on the density of obstacles. Introducing temporal aspects in this formulation results in TEB [7]. This upgrade makes possible to consider dynamic constraints (such as velocities and accelerations) and its results are closer from being optimal than the ones from DWA. Nevertheless, TEB's downside is that constraints are only applied as penalties.

Model Predictive Control (MPC) is an advanced control method that can be used for motion planning. It consists on taking predictions by relying on a plant model over a finite horizon. Every prediction is evaluated according to a cost function and only the first input set of the optimal prediction is applied. In other words, it is a closed-loop optimization-based control approach that works in a receding horizon fashion and can explicitly optimize trajectories according to nonlinear goals, while considering desired constraints in states and inputs [11]. This accounts for its high presence in robotics tasks, such as collision avoidance in trajectory

planning for multi-robots [12] and regularization of nonholonomic robots [13].

Due to the nonlinear physics of a DDMR with caster wheels, MPC is a suitable framework, since it can find the optimal control action respective to nonlinear objectives. Moreover, the capability to include constraints in states and inputs when solving the Nonlinear Problem (NLP) fits to the need of limiting the robot's velocities and accelerations for trajectory planning. Therefore, MPC can address differential drive motions and, at the same time, counteract caster wheel reaction torques already in the trajectory planning stage.

### 1.3 Purpose

Considering that the PF is not aware of the existence of the constraints included in the motion planner, it sacrifices navigation capabilities and endangers its completion. That is why, there is a need to address the caster wheel's physics in the motion planning stage.

To do so, the circumstances under which caster wheel reaction torques arise need to be identified and a term that penalizes them has to be proposed. Since this term is implemented in a MPC framework, it needs to be finite and differentiable within the entire input range and cannot have a negative impact in the navigation's quality. Hence, the implications of including it into the motion planning stage have to be observed. To be clear, this process is broken down into the following three research steps:

**Research Objective 1:** *Identify the circumstances under which caster wheel reaction torques are relevant for the robot's performance.*

**Research Objective 2:** *Formulate a term that minimizes the scenarios stated in Obj.1 and is numerically compatible with an Optimal Control Problem (OCP).*

**Research Objective 3:** *Analyze the compatibility of the term formulated in Obj.2 with navigating capabilities of the local planner.*

The ultimate purpose of the three objectives mentioned above is to minimize the influence of the caster wheel reaction torques in the robot's navigation performance by addressing the caster wheel awareness in the motion planning stage. Accordingly, this thesis ought to answer the following research question:

**Research Question:** *Is it possible to formulate a local planner that considers caster wheel physics, so that maximum and average motor torques are reduced without compromising navigation capabilities?*

### 1.4 Methodology

The procedure followed for completing this thesis can be divided into three steps. A literature review leads to the formulation of a concept that is evaluated according to the trends observed in specific case studies. Below this process is described in detail.

In the first place, a literature review defining the background needed to answer the research question is carried out. The focus is set on understanding previous work regarding techniques of motion planning capable of dealing with nonlinear dynamics. For this purpose, approaches that can address additional objectives, other than navigation, are considered. This requires understanding the robot's physics upon which the local planner will be tested, leading to the analysis of modelling methods for a DDMR and a caster wheel.

Secondly, the theoretical content that stands behind the contributions to this thesis is presented. At the very beginning, a case identification that recognizes the circumstances under which caster wheel reaction torques occur is carried out. To do so, a hypotheses that validates an statement of previous work is tested by running simulations. Subsequently, the robot's plant model, a cost function that evaluates the predictions and a observer that estimates caster wheel states are combined into an OCP. When doing so, the horizon and controller's sampling time are also determined. Before evaluating this planner details on its implementations are given. Finally, an extension of a method presented in previous work is proposed .

Thirdly, the contributions are tested by analyzing specific case studies in a virtual environment and an experimental setup. Considering that field testing is time consuming and resource demanding, the initial analysis will be based on simulations. This increases the speed of testing and extends the range of opportunities, which, all together, lead to a more profitable research. The results obtained from the simulations are considered to be the basis for the experimental case studies. This provides a strong foundation to apply the knowledge acquired during the simulations once the motion planner is implemented in the real robot.

The analysis in the virtual framework can be divided into two simulation sets. The first one evaluates the observer at the presence of disturbances and load variations, while the second tests the caster wheel awareness through three case studies, where three local planners are compared against each other. Once the results in the virtual environment are satisfactory, the analysis is extended to the real world by replicating two case studies in the AS. The mixture of the results obtained in the simulations and experimental case studies provides a strong basis for drawing insightful conclusions about the capabilities and weaknesses of the proposed concept.

## 1.5 Delimitations and Limitations

### 1.5.1 Delimitations

Even though the content presented in this thesis has been tailored for the AS, the contributions are applicable to any DDMR with caster wheels. Moreover, the minimization of caster wheel reaction torques has been addressed in a modular manner, so that it is compatible with any MPC based navigation algorithm.

Therefore, the focus is set on the formulation of the caster wheel aware term, while the navigation algorithm is simplified to reference tracking across a sub-optimal global path. Its downside is that following a virtual time based reference biases the usage of motor torques, avoiding to visualize the full impact of the caster wheel aware term.

Apart from that, when conducting case studies in the experimental setup, certain weak points in the methods proposed in previous work have been observed. Even if a further extension could solve these, the resultant formulations would not be sufficient for overcoming the stated problem. That is why, these improvements are considered to be out of the scope of this thesis and are left as future work.

### 1.5.2 Limitations

Since the AS is not manufactured in the entity where research is conducted, there is a lack of knowledge regarding hardware and software specifications, which hinders the implementation of the local planner in the robot. For example, an explicit driver that connects the ROS network with the Robot Control Unit (RCU)

has to be written. Therefore, an efficient implementation cannot be guaranteed, compromising the results obtained when testing.

On top of that, the robot's Safety Control Unit (SCU) is active, even when the navigation stack is deactivated. Consequently, the control actions derived from the local planner can be modified or interrupted by the SCU. As a consequence, testing is exposed to unknown phenomena, slowing down the identification of bugs and reducing the test's outcome. In order to avoid this, it is necessary to guarantee that the distance to the nearest obstacle is below a threshold, the velocity commands are within a given range and the blinkers are coordinated with robot rotations. To this end, hardware specific constraints have to be included in the local planner, which increases the computational load and grows apart the theoretical formulation from the implemented one.

## 1.6 Deposition

This thesis is divided into 6 Chapters, as shown in Figure 1.6. After having introduced the topic and determined the research fields of interest in this chapter, Chapter 2 covers the required background to understand the contributions of this thesis. The theoretical formulation of the MPC based caster wheel aware local planner is given in Chapter 3. This is tested in Chapters 4 and 5, where case studies in a virtual environment and an experimental setup are conducted. Finally, Chapter 6 summarizes the steps taken to answer the research question, discusses the results, highlights the most relevant contributions of the thesis and closes by pointing out future research directions.

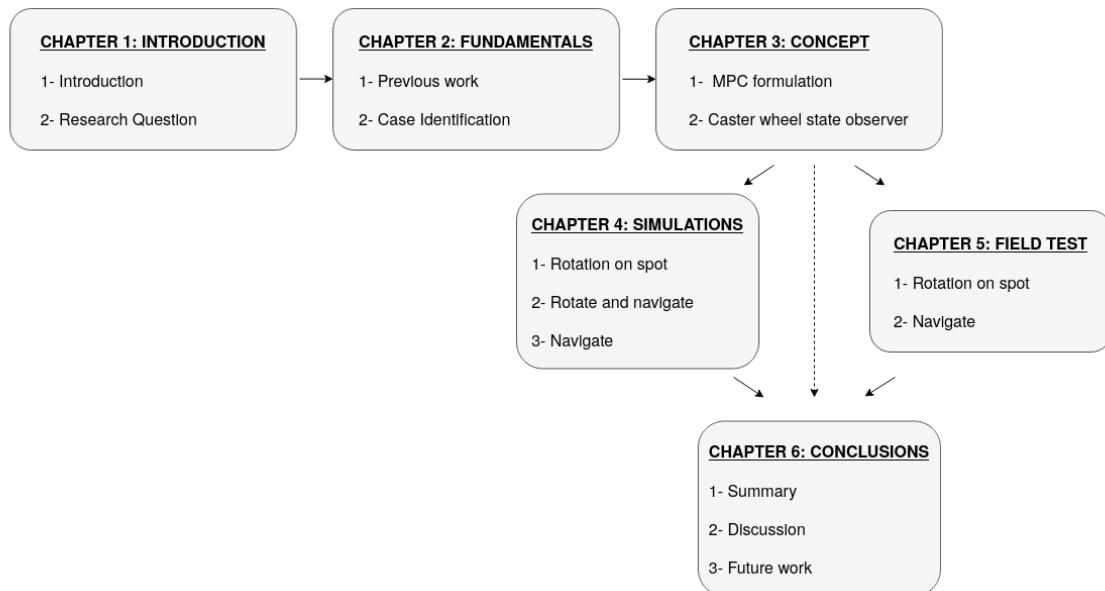


Figure 1.6: Outline of the thesis.

# Fundamentals

---

The purpose of this Chapter is to define the basic concepts required for answering the research question. For this purpose we start by examining previous work according to the research scope stated in Chapter 1. To this end, the focus is put on caster wheel awareness in mobile robotics and MPC based motion planning. Subsequently, a case identification is conducted, so that circumstances in which critical caster wheel reaction torques occur are detected. The highlighted content will be the basis upon which contributions to the thesis will be made.

## 2.1 Modelling of DDMR with Caster Wheels

MPC consists on applying the first control action respective to the optimal prediction. These predictions are taken according to a plant model based on Ordinary Differential Equations (ODE), which implies that the physics of the robot need to be modelled. Since the scope of this thesis is set on DDMRs with caster wheels, the modelling can be divided into two subsystems.

### 2.1.1 DDMR

The modelling of mobile robots with a differential drivetrain is well known and has been applied across several research fields. Paper [14] is an appropriate summary, since it proposes two different ODE formulations (Newton-Euler and Lagrange) for differential driven robots and proves their equivalence.

### 2.1.2 Caster Wheels

The kinematics and dynamics of the caster wheels are given in [15], where the principle of virtual work and second order Lagrange equations are applied in a DDMR robot with a centered caster wheel located at the front side.

On top of that, in the second Section of [5], the caster wheel and its respective reaction torque, defined as bore torque, are also modelled. For this purpose, it proposes a modified version of the tyre model suggested

in [16]. The respective expression is given in Equation 2.1.

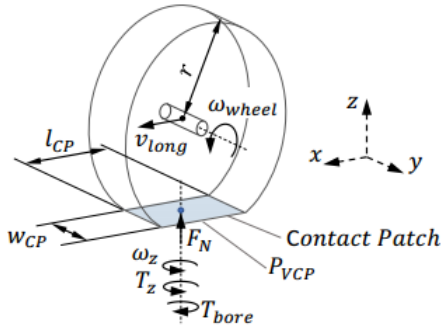
$$|T_{\text{bore}}| = \begin{cases} (|T_{\text{bore,max}}| - |T_{\text{bore,stic}}|) \cdot \frac{\lambda_{\text{bore}}}{\lambda_{\text{bore,lim}}} + |T_{\text{bore,stic}}|, & \text{if } |\omega_z| \cdot s_{\text{rep}} > \lambda_{\text{bore,lim}} \cdot |\omega_{\text{wheel}}| \cdot r \\ |T_{\text{bore,max}}| & \text{else} \end{cases} \quad (2.1a)$$

$$|T_{\text{bore,max}}| = F_N \cdot \mu_{\text{bore}} \cdot s_{\text{rep}} \quad (2.1b)$$

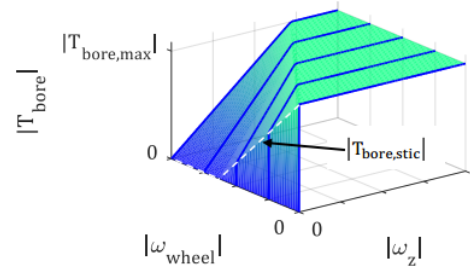
$$|T_{\text{bore,stic}}| = \max(0, |T_{\text{bore,max}}| - k_{\text{stic}} \cdot |\dot{\gamma}|) \quad (2.1c)$$

$$\lambda_{\text{bore}} = \frac{|\omega_z| \cdot s_{\text{rep}}}{|\omega_{\text{wheel}}| \cdot r} \quad (2.1d)$$

where  $\lambda_{\text{bore}}$  is the bore slip,  $\mu_{\text{bore}}$  is the bore friction,  $w_z$  and  $w_{\text{wheel}}$  are the caster wheel's rotating and rolling speeds,  $s_{\text{rep}}$  is a measurement regarding the contact patch and  $k_{\text{stic}}$  is a constant that softens the transition to maximum bore torque. These variables can be visualized in Figure 2.1a and Equation 2.1 is represented graphically in Figure 2.1b.



(a) Caster wheel's representation according to variables in paper [5]



(b) Graphical representation of Equation 2.1 of caster wheel's bore torque, rolling and rotating speeds

Figure 2.1: Caster wheel's geometrical and bore torque representations - Source: [5]

## 2.2 Caster wheel awareness in mobile robots

To the knowledge of the author, there is no previous work apart from paper [5] that accounts for caster wheel awareness in mobile robotics. As it has been mentioned in Section 1.1, [5] proposes to introduce a filter and an observer between the motion planner and the motor controller, so that the velocity commands obtained from the first are modified according to estimated caster wheel states and are fed into the latter. Even if this method has been validated in [5] by running experiments, in Section 1.2 it has been explained that it sacrifices navigation capabilities and jeopardizes its completion. In any case, it is the only work that tackles this problem and its closeness to the AS, thus, looking into its formulation provides insightful information for formulating a caster wheel aware local planner.

### 2.2.1 Formulation

As a first step, desired longitudinal and angular velocity commands ( $v$  and  $\omega$ ) are converted to their respective equilibrium caster wheel states (rotation angle,  $\bar{\varphi}$ , and rolling speed,  $\bar{\gamma}$ ). Simultaneously, the caster wheel state ( $\hat{\varphi}$  and  $\hat{\gamma}$ ) is estimated by applying a delay to the previously filtered caster wheel states. This delay introduces two tuneable parameters which need to be tailored to the dynamics of the robot. In order to ensure that sudden changes in the caster wheel's orientation do not occur, a filtered state is deduced by applying Equation 2.2a. Finally, this state is converted back to filtered velocity commands ( $v_{\text{filt}}, \omega_{\text{filt}}$ ). This process is depicted in Figure 2.2.

$$\varphi_{\text{filt}} = \hat{\varphi} + k \cdot (\varphi_{\text{des}} - \hat{\varphi}) \quad (2.2a)$$

where  $k$  is a filtering term given by

$$k = \min\left(1, \left| \frac{\hat{\gamma}}{\dot{\gamma}_{\text{max}}} \right| \right) \quad (2.2b)$$

Notice that  $\gamma_{\text{max}}$  is a third tuning parameter which defines the influence of the filter. High values will increase the time needed to reach the desired state, leading to a reduction in maneuverability but keeping reaction torques low. Apart from that, the filtered rolling speed is assigned the value of the desired one.

$$\dot{\gamma}_{\text{filt}} = \dot{\gamma}_{\text{des}} \quad (2.2c)$$

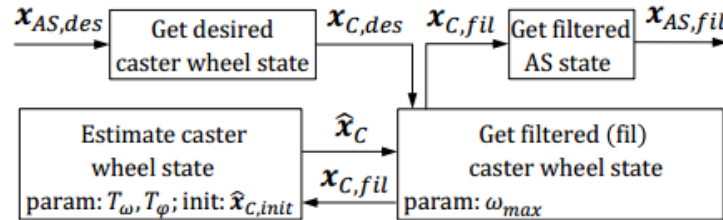


Figure 2.2: Description of PF's working principle - Source: [5]

### 2.2.2 Implementation

Paper [5] also presents a Functional Mock Up Interface (FMI) [17] adapter for coupling a Modelica [18] based model with the Robot Operating System (ROS) [19]. In this way, the PF can be implemented in hardware by modelling it in Modelica and exporting it to a Functional Mock-up Unit (FMU). This is proven in paper [5], where the entire AS robot, along with the PF, are modelled in Modelica. This implementation method is validated by running experiments in an early stage prototype.

## 2.3 MPC based motion planning

Model Predictive Control (MPC) is an optimization based closed loop control approach that works in a receding horizon fashion [11]. Predictions made over a plant model,  $F$ , for a finite horizon,  $N$ , are evaluated according to a cost function,  $J$ , and constraints in states and inputs,  $H$ . In other words, an Optimal Control

Problem (OCP) with a finite horizon is solved and only the first input-set of the optimal prediction,  $\mathbf{u}_1^*$ , is applied. Once the robot performs the motion respective to these inputs, the states are measured from sensors or estimated by an observer, so that they can be fed back as the starting point of the predictions in the next iteration. This process is repeated according to the controller's sampling time  $t_s$ .

$$U^* = \min_u \sum_{k=1}^{N+1} J(x_k, u_k) \quad (2.3a)$$

$$\text{subject to } x_{k+1} = F(x_k, u_k) \quad (2.3b)$$

$$0 \geq H(x_k, u_k) \quad (2.3c)$$

$$x_1 = x_e \quad (2.3d)$$

where  $U^*$  is the control sequence over the optimization horizon and consists of  $u_k$  with  $k \in \{1, 2, \dots, N+1\}$  at each stage.

Such a controller can be suitable for addressing nonlinear dynamics in the motion planning stage [20]. Non-linear model-predictive methods for motion planning enable simultaneous resolution of obstacle avoidance problems, feasible trajectory selection, and trajectory following, while complying with constraints in control inputs and state values [21]. In this way, caster wheel physics can be integrated in the motion planning.

For this purpose, the MPC not only has to ensure that the robot travels across the global path, but it also needs to mitigate caster wheel reaction torques while navigating. In other words, the cost function needs to have at least two terms. The first one relates to the navigation capabilities and the second one to the caster wheel reaction torques.

$$J = J_{\text{navigation}} + J_{\text{cw}} \quad (2.4)$$

Notice that the second term  $J_{\text{cw}}$  is only applied if caster wheel reaction torques want to be minimized, while the first term,  $J_{\text{navigation}}$ , is mandatory to ensure that the robot navigates across the global path. In other words, it addresses the motion planning problem, which is defined as the capability of a robot to plan its own motions, in order to achieve a task specified by initial and goal spatial arrangements of physical objects [22]. Considering that the focus of this work is put on the caster wheel awareness, the motion planning will be simplified to a path following problem, where the environment is static and the robot is the only moving object.

### 2.3.1 Path following

Path following consists on designing a control law that drives an object to reach and follow a geometric path by chasing a virtual signal. If this reference is updated according to a timing law, the "path following" problem is simplified to "trajectory tracking". In [23] a comparison between both leads to the conclusion that the latter one involves a performance limitation.

The ease of including bounds to inputs and states combined with the difficulties that arise from traditional closed loop control methods, converts MPC in an interesting approach for addressing the path following



and trajectory tracking problems. When doing so, it needs to be highlighted that non-integrable constraints contained in Wheeled Mobile Robots (WMR) add complexity with respect to omnidirectional robots.

Paper [24] extends and generalizes previous work in prediction based path following. Regarding trajectory tracking for WMR, [25] proposes, compares and validates two different approaches (linear and non-linear MPC). In [26] path following and trajectory tracking are combined into the same MPC framework. In this way, the respective benefits of both problems are merged.

## 2.4 Case Identification

This Section will focus on studying and motivating the scenarios when caster wheel reaction torques become relevant for the robot's dynamics. Considering that paper [5] identifies rotations on spot or with low longitudinal velocities as the most critical scenarios, the robot's cornering performance is going to be analyzed by running simulations in the AS's Modelica based model presented in [5]. These simulations will lead to stating and proving a hypotheses, which defines the basis for formulating a local planner capable of answering the research question.

### 2.4.1 Hypotheses statement

When it comes to analyzing how motor-torques react to specific robot motions, looking into cornering cases is specially relevant.

In order to state the hypotheses, sensitivity to load in a  $90^\circ$  corner, performed while standing still ( $v = 0$  m/s and  $\omega = -0.35$  rad/s) is considered. This is further developed by looking into bore friction parameter variations. Finally, the hypotheses is evaluated according to the results obtained from the simulations. Since the robot has to carry loads within the range of 150 to 250 Kg, both extremes are observed.

As Figure 2.3a shows, the caster-wheel angle changes when the robot starts rotating. The time difference depends on the load that is being carried. The heavier the load, the longer the period between both events. Once the rotation is completed, the caster wheels align with respect to the robot's orientation. Notice that no time difference occurs at this stage.

In Figure 2.3b it can be visualized that the caster wheel angle's delay is related to the peak torque generated by the motors at the beginning of the rotation. The existence of this peak can be explained by the caster wheel reaction torques given in Figure 2.3c. Thus, increasing the load augments caster wheel reaction torques and generates higher motor-torques. Notice that this only occurs in the first transient.

From Figure 2.3d, it can be concluded that the existence of a longitudinal velocity command causes the difference between the two transients. In the first case, at the beginning of the rotation, the robot has no longitudinal motion. However, when the caster wheels get aligned with the robot's orientation, the robot is driving forward. Thus, bore torques have a bigger impact on the robot's performance when turning while standing still.

Another way to see this is by looking into caster wheel rolling speeds in Figure 2.4. If the robot is rotating and the caster wheels are not rolling, the motors will have to provide higher torques. By looking into the

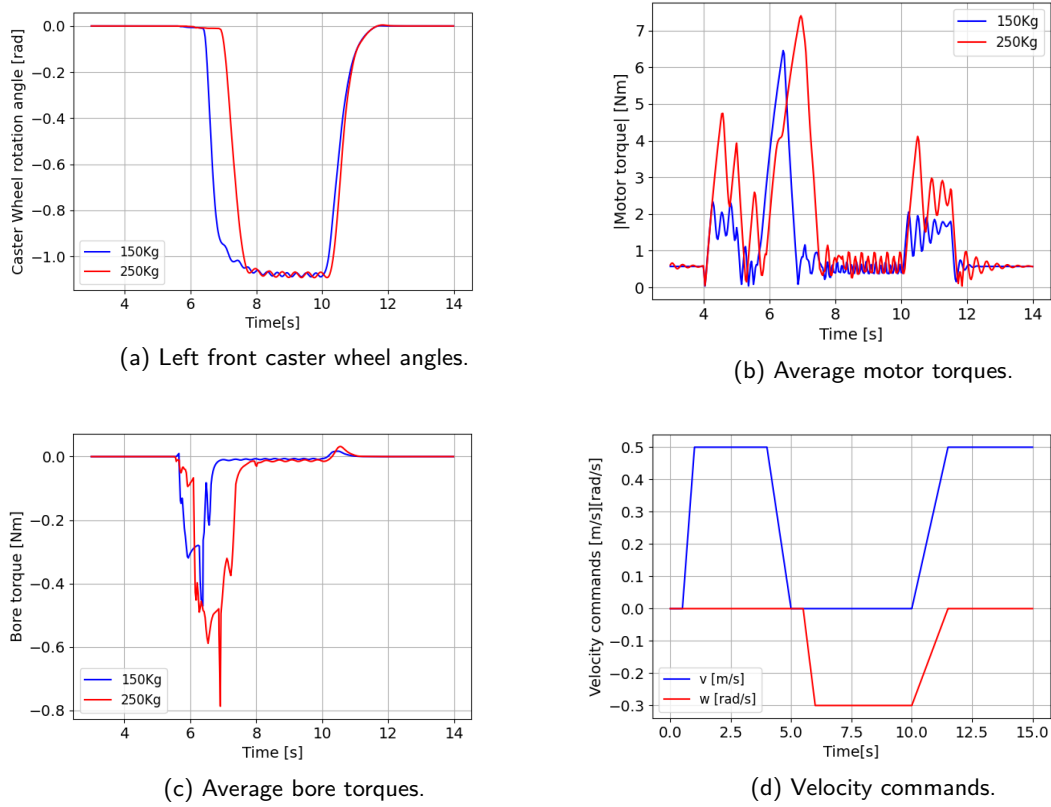


Figure 2.3: Identification of caster wheel reaction torques. Load sensitivity analysis for a cornering case of  $90^\circ$  while standing still. The velocity commands of the case study are given in Figure 2.3d.

timeline, it can be observed that the period when the caster wheel is not rolling (from  $t = 6$  to  $t = 7.5$  approximately) matches the peak torque given in Figure 2.3b.

From the arguments given in the previous paragraphs the following hypotheses can be stated:

*"The peak torque generated by the motors (1), the load that the robot is carrying (2) and the time difference between transients in caster wheel angles (3) are proportional for stand still rotations ( $v = 0, \omega \neq 0, \dot{\gamma} = 0$ ) with misaligned caster wheels ( $\Delta\varphi > 0$ )."*

The hypotheses can be further developed by looking into the model's sensitivity with respect to bore friction parameters. The proportionality given in the hypotheses only holds for stand-still rotations. In other words, it is only applicable for cases where the robot is rotating without moving forward and the caster wheel is not rolling. If we apply this to Equation 2.1, the bore torque acquires its maximum value ( $|T_{\text{bore}}| = |T_{\text{bore,max}}|$ ), which explains the existence of the reaction torque's peak when the caster-wheel is not rolling. This is highly relevant, since ensuring that the caster wheel is rolling guarantees that the respective bore torque does not reach its maximum value.

Before extending the hypotheses, it is convenient to sanity-check the ideas covered in the last paragraph by performing a simulation where the friction coefficient between caster wheels and environment ( $\mu_{\text{bore}}$ ) is 0. From Equation 2.1 it can be stated that this modification leads to  $|T_{\text{bore,max}}| = 0$ , which, at the same time involves that  $|T_{\text{bore,stic}}| = 0$ , and, finally, translates to  $|T_{\text{bore}}| = 0$ . If this is true, not only the torque spikes should disappear, but there also should be no time difference between caster wheel angle transients of both

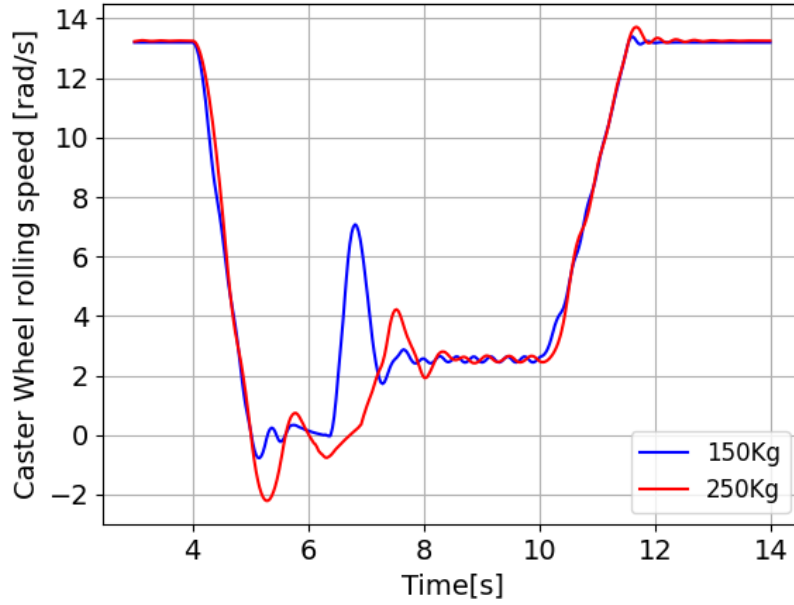


Figure 2.4: Rolling speed of the left front caster-wheel for plant (kinematics) and OpenModelica (150 Kg and 250 Kg) models applicable to a cornering case of  $90^\circ$  while standing still (Figure 2.3d)

loads. Figure 2.5 demonstrates that all these assumptions are true.

Considering the content covered in the previous paragraphs, the hypotheses could be reformulated as follows:

*"Motor-torques that are above nominal range are required for rotations on the spot ( $v = 0$ ,  $\omega \neq 0$ ,  $\dot{\gamma} = 0$ ) if caster wheels are misaligned with respect to the robot's longitudinal motion ( $\Delta\varphi > 0$ ). These peaks are proportional to the load that the robot is carrying and the time difference between transients in caster wheel angle values."*

## 2.4.2 Hypotheses simulation

The hypotheses given in 2.4.1 can be tested by simulating  $90^\circ$  cornering with low longitudinal velocities. If the hypotheses is correct, the torques obtained for these simulations should be lower than the ones given in Figure 2.3. In order to prove this, two sets of simulations have been fulfilled with a longitudinal velocity of 0.1m/s and 0.3m/s while the robot is rotating. The results are shown in Figures A.1 and A.2 and summarized in Table 2.1.

The results given in Figures A.1 and A.2 (see appendix), along with Table 2.1, prove that the hypothesis stated in 2.4.1 is correct. A very low longitudinal velocity (0.1m/s) is enough to reduce the maximum motor torque by 71% for a load of 150Kg. For the case of 250Kg, this reduction increases up to 82%. On top of that, the torque reductions are negligible when the longitudinal velocity is further increased. Increasing the longitudinal speed by 30% has caused a torque decrease of 17% for 150Kg and 11.1% for 250Kg.

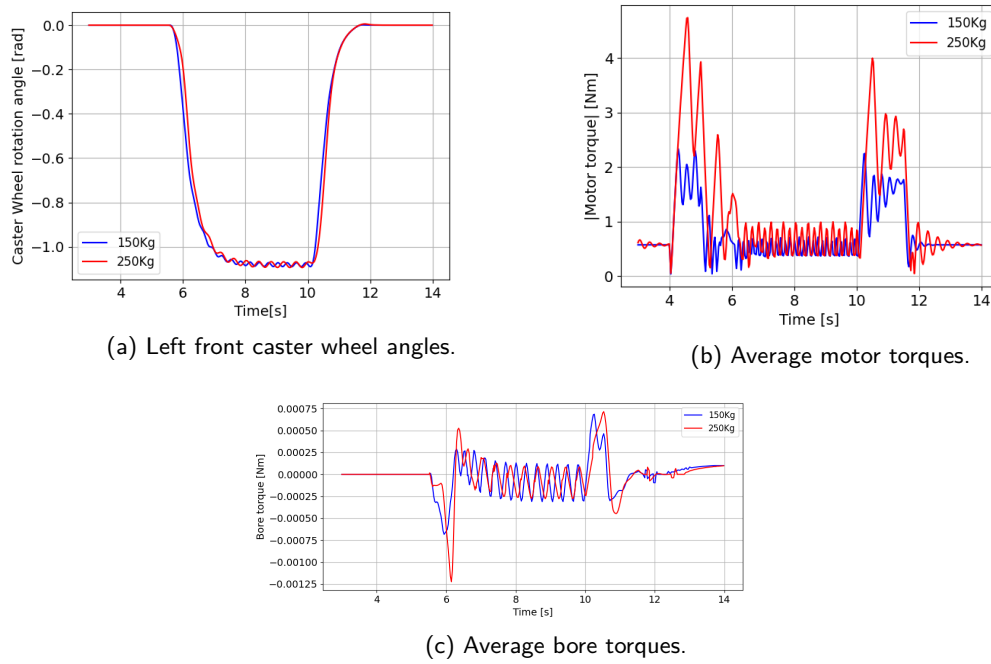


Figure 2.5: Load sensitivity for a cornering case of 90° while standing still without bore frictions.

Table 2.1: Time difference between caster wheel angle transitions, maximum motor and bore torques depending on load carried by the robot and different longitudinal speeds when rotating in a 90° corner and carrying a load of 150Kg and 250Kg.

Load [Kg]	$v$ [m/s]	$\Delta t(\hat{\varphi}, \varphi)$ [s]	$T_{\text{motor,max}}$ [Nm]	$T_{\text{bore,max}}$ [Nm]
150 Kg	0	0.5	5.9	1.45
	0.1	0.1	1.7	0.5
	0.3	0.05	1.4	0.0125
250 Kg	0	2	10	2.2
	0.1	0.1	1.8	0.7
	0.3	0.05	1.6	0.019

### 2.4.3 Hypotheses implication

The hypotheses stated and proven in this Section implies that the caster wheel reaction torques strongly rely on the tendency of the local planner to take curves with low longitudinal speed. Thus, the MPC based local planner is going to be formulated in such a way that it minimizes rotations on spot, as long as its trajectory does not excessively diverge from the path it needs to follow.

## 2.5 Summary

In this Chapter the background required to answer the research question has been covered. As a starting point, previous work in considering caster wheels for mobile robot navigation has been studied by looking into paper [5]. From this publication, the analytical expression for the caster wheel reaction torque and the PF solution have been explained. For the latter case, its limitations in specific scenarios, such as parking or high density of obstacles, has revealed the need of integrating caster wheel awareness in the motion planning stage.

It has been shown that MPC is a suitable control approach for this purpose, since it can address the motion planning problem and caster wheel physics simultaneously. Considering that this thesis focuses on minimizing caster wheel reaction torques, the motion planning problem has been reduced to path following. That is why, contributions in papers that implement MPC to solve path following and trajectory tracking problems have been highlighted.

Finally, the circumstances under which caster wheel reaction torques become critical for the performance of the robot have been analyzed by running simulations. In this way, a hypothesis that correlates motor torques with the robot's velocity profile is formulated and proven. It has been concluded that the reaction torques are associated to rotations on spot ( $v = 0, \omega \neq 0$ ), where caster wheels are not rolling ( $\dot{\gamma} = 0$ ) and they have an offset with respect to the steady state caster wheel angle ( $\Delta\varphi > 0$ ). In the following Chapter a local planner that addresses this phenomena is formulated.



# Concept

---

This Chapter covers the theoretical content upon which the research question will be answered. Considering the knowledge acquired from the analysis of previous work and the case identification performed in Chapter 2, an MPC based local planner that takes caster wheels into account is presented. To do so, the OCP that obtains the sequence of optimal control decisions, according to the real time state feedback is formulated. This requires defining the plant model, cost function, constraints, horizon and caster wheel state observer. On top of that, an extension to the PF explained in Chapter 2 is proposed.

## 3.1 Plant Model

The plant model is used for taking predictions over a finite horizon. Since the input command applied to the robot depends on those predictions, the plant model should reflect the significant dynamic characteristics of the real robot.

Taking into account that the AS is a DDMR with caster wheels, it can be modelled as two independent subsystems: a differential drive and a caster wheel. Kinematic and dynamic models are proposed for both of them. After being analyzed, both subsystems are combined, resulting in the formulation of the AS's plant model.

### 3.1.1 Differential Drive Mobile Robot

DDMRs have been a common tool in a wide range of research fields and its kinematic and dynamic modelling are mature. These are given in [14] and they are used as a basis for modelling the differential drive subsystem of the AS. After formulating both options (kinematics and dynamics), a comparison leads to choosing one of them.

The robot is modelled in a planar fashion, as given in Figure 3.1. To do so, two different frames are used. The first one  $\{x^1, y^1, z^1\}$  is a global frame that is fixed to the environment in which the robot moves in, while the second one  $\{x^2, y^2, z^2\}$  is a local frame attached to the robot's origin  $O$ .

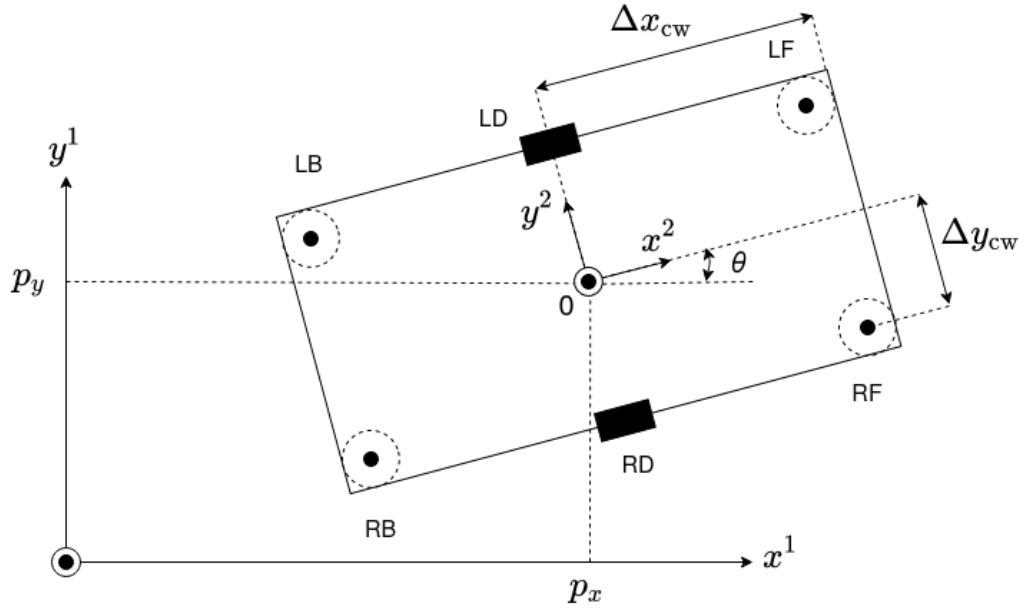


Figure 3.1: Diagram of the robot. Definition of global and local coordinate systems:  $\{x^1, y^1, z^1\}$  and  $\{x^2, y^2, z^2\}$ . Dimensions  $\Delta x_{cw}, \Delta y_{cw}$  represent the distance from the robot's origin to the left front caster-wheel.  $\Delta y_{dw}$  refers to the distance from the robot's origin to the driven wheels.

### 3.1.1.1 Kinematics of Diff. Drive

Considering longitudinal and rotational velocity ( $v$  and  $\omega$ ) as inputs, kinematics of a DDMR can be used to obtain the equations of motion that represent the origin's absolute position  $p_x, p_y, \theta$  with respect to reference frame  $\{x^1, y^1, z^1\}$ :

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega \end{bmatrix} \quad (3.1)$$

### 3.1.1.2 Dynamics of Diff. Drive

The model that is going to be proposed takes currents as inputs, converts them to torques and feeds them to the dynamics of a DDMR.

#### Estimation of motor-currents

Motor currents can be derived by applying a PI controller to desired and real velocities, as shown in the motor controller of the Modelica model developed in paper [5] and depicted in Figure 3.2).

First step is to convert velocity commands to rotational speeds of the driven wheels.

$$\omega_{dwl|t} = \frac{1}{r_{dw}} \cdot (w_t \cdot \Delta y_{dw} + v_t) \quad (3.2a)$$

$$\omega_{dwr|t} = \frac{1}{r_{dw}} \cdot (-w_t \cdot \Delta y_{dw} + v_t) \quad (3.2b)$$

Since the procedure is analogue for both motor-currents (left and right),  $\omega_{dwl,t}$  and  $\omega_{dwr,t}$  will be simplified



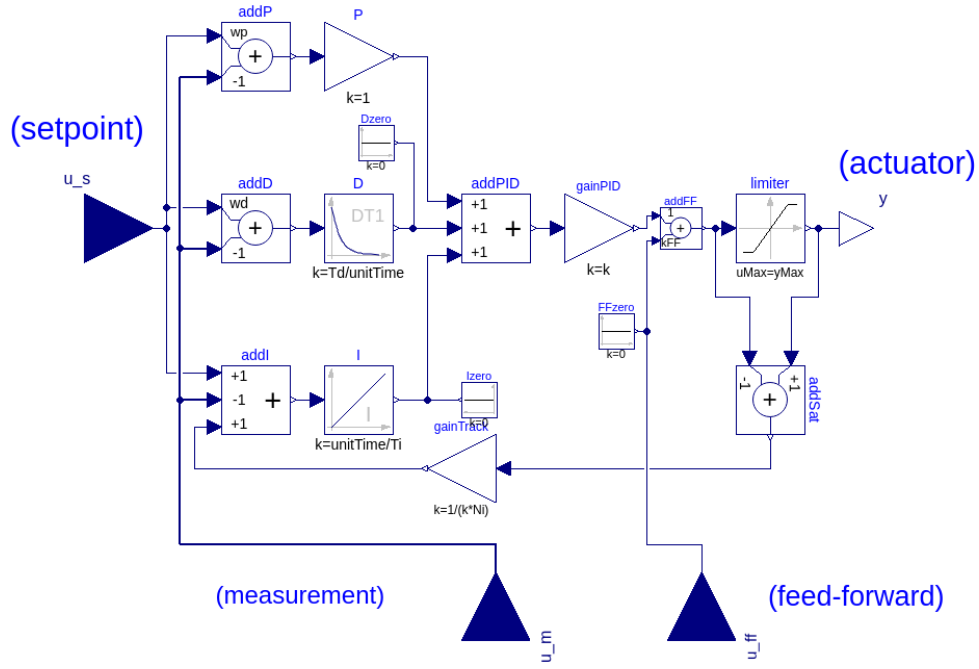


Figure 3.2: Model of the robot's motor controller in OpenModelica.

to  $\omega_{dw,t}$ . Secondly, the error between desired and real rotational speeds is defined.

$$e_t = \omega_{dw|t} - \omega_{dw,fb|t} \quad (3.2c)$$

where  $\omega_{dw,fb|t}$  is the rotational speed subtracted as feedback. Adding an anti-windup term to this error results in the controller's integration part.

$$\zeta_t = e_t + \frac{1}{T_{sat}} \cdot (i_{t-1} - h_{t-1}) \quad (3.2d)$$

$T_{sat}$  is the time constant of anti-windup compensation,  $h_{t-1}$  and  $i_{t-1}$  are motor-currents before and after saturation. This term can be integrated using Forward Euler's method.

$$\int \zeta_t = \int \zeta_{t-1} + t_s \cdot \zeta_t \quad (3.2e)$$

where  $t_s$  is the step size. Combining all the equations given above with the respective proportional  $T_p$  and integral  $T_i$  constants, results in expressing motor currents as shown below:

$$h_t = \frac{1}{T_p} \cdot e + \frac{1}{T_i} \cdot \int \zeta_t \quad (3.2f)$$

Since the current that can be pulled out the motors is limited to  $i_{max}$ , bounds need to be set on  $h$ . This can

be done by a saturation function.

$$i_t = \begin{cases} i_{\max}, & \text{if } h_t > i_{\max} \\ -i_{\max}, & \text{else if } h_t \leq -i_{\max} \\ h_t & \text{else} \end{cases} \quad (3.2g)$$

Both motor currents can be obtained by following this procedure. These values will be the inputs for the ODEs that are explained in the following subsection.

### Dynamic ODEs

The estimated motor currents are the inputs to the ODE system that describes the dynamics of a DDMR. That is why, the ODE can be divided into two Sections.

The first one consists of two equations, which convert currents,  $i$ , into torques,  $T$ . This is done by a first order delay that comes after the PI motor-controller.

$$\dot{T} = \frac{1}{T_{ti}} \cdot (i - T) \quad (3.3a)$$

$T_{ti}$  is the time constant regarding the conversion from current to torque. The second group involves the dynamics of DDMR. These have been widely implemented across several research fields and are given in [14].

$$-\frac{1}{r_{dw}} \cdot (T_L + T_R) + M \cdot (a - \Delta x_{\text{cog}} \cdot \omega^2) = 0 \quad (3.3b)$$

$$\frac{2 \cdot \Delta y_{dw}}{r_{dw}} \cdot (T_L - T_R) - \Delta x_{\text{cog}} \cdot M \cdot v \cdot \omega + (I_{zz} + \Delta x_{\text{cog}} \cdot M^2) \cdot \alpha = 0 \quad (3.3c)$$

Putting everything together results in the following ODE system:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} \frac{1}{T_{ti}} \cdot (u_1 - x_1) \\ \frac{1}{T_{ti}} \cdot (u_2 - x_2) \\ \frac{1}{M \cdot r_{dw}} \cdot (x_1 + x_2) + \Delta x_{\text{cog}} \cdot x_4^2 \\ \frac{1}{I_{zz} + \Delta x_{\text{cog}} \cdot M^2} \cdot \left( \frac{-2 \cdot \Delta y_{dw}}{r_{dw}} \cdot (x_1 - x_2) + \Delta x_{\text{cog}} \cdot M \cdot x_3 \cdot x_4 \right) \end{bmatrix} \quad (3.4a)$$

where

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} T_L \\ T_R \\ v \\ \omega \end{bmatrix}, \quad \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} i_L \\ i_R \end{bmatrix} \quad (3.4b)$$

#### 3.1.1.3 Model comparison

Any of the two systems (kinematics in Equation 3.1 or dynamics in 3.4a) proposed above is valid to be implemented in the plant model. In order to choose which is the best option, several points need to be considered.

The kinematic alternative is simpler, since it has less equations and they are pure integrators. Its downside is

that it is not aware of the load and inertia that the robot is carrying.

The dynamics based system does address this topic, at the expense of adding four extra equations, and several modelling and tuning parameters. On top of that, it also requires a motor current estimator attached to its input. This complexity makes it more prone to differing from the robot's behavior. Furthermore, the DDMR dynamics introduce two poles, which convert it into a faster system than the kinematic one. Due to all these disadvantages, the kinematic system has been chosen to replicate the robot's differential drive behavior in the plant model.

### 3.1.2 Caster Wheel

The four caster wheels located at each corner of the robot's footprint can also be modelled through kinematic and dynamic relationships.

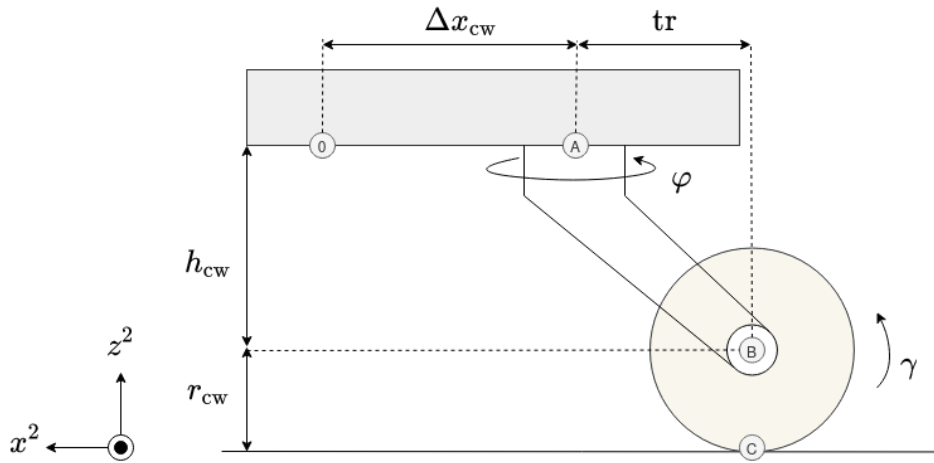


Figure 3.3: Diagram of the caster wheel. Point 0 is the robot's origin, A is the linkage between robot-frame and overhang, B is the connection between overhang and caster-wheel, and C is the contact point between caster-wheel and environment.

#### 3.1.2.1 Kinematics of the caster wheel

By applying kinematic relationships, equations of motion respective to the caster wheel's rotational and rolling angles  $\varphi, \gamma$  can be derived. This is demonstrated in the upcoming paragraphs for the front left caster wheel.

As a starting point, it is necessary to define the velocity of B in two different manners. Considering that the frame's velocity is known, B's velocity is

$$\vec{v}_B = \vec{v}_O + \vec{\omega}^1 \times \vec{OA} + \vec{\omega}^2 \times \vec{AB} = \begin{bmatrix} v \\ 0 \\ \omega \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\theta} \end{bmatrix} \times \begin{bmatrix} \Delta x_{cw} \\ \Delta y_{cw} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dot{\varphi} \end{bmatrix} \times \begin{bmatrix} -tr \\ 0 \\ -h_{cw} \end{bmatrix} \quad (3.5)$$

which results in

$$\vec{v}_B = \begin{bmatrix} v - \dot{\theta} \cdot \Delta y_{cw} + \dot{\varphi} \cdot \sin(\varphi) \cdot tr \\ \dot{\theta} \cdot \Delta x_{cw} - \dot{\varphi} \cdot \cos(\varphi) \cdot tr \end{bmatrix} \quad (3.6)$$

If spinning in the caster wheel is neglected and pure rolling is assumed, the following equations also define  $B$ 's velocities in the local frame:

$$\vec{v}_B = \vec{w}^{cw} \times \vec{CB} = \begin{bmatrix} \dot{\gamma} \cdot r_{cw} \\ 0 \\ \dot{\varphi} \end{bmatrix} \quad (3.7)$$

Translating this velocity to the fixed frame results in

$$\vec{v}_B = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \dot{\gamma} \cdot r_{cw} \\ 0 \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} \dot{\gamma} \cdot r_{cw} \cdot \cos \varphi \\ \dot{\gamma} \cdot r_{cw} \cdot \sin \varphi \\ \dot{\varphi} \end{bmatrix} \quad (3.8)$$

From combining equations 3.6 and 3.8, the following two equations of motion arise, which describe the behavior of the caster wheel's rotating and rolling angles  $(\varphi, \gamma)$ :

$$\begin{aligned} \dot{\gamma} \cdot r_{cw} \cdot \cos \varphi &= v - \dot{\theta} \cdot \Delta y_{cw} + \dot{\varphi} \cdot \sin(\varphi) \cdot \text{tr} \\ \dot{\gamma} \cdot r_{cw} \cdot \sin \varphi &= \dot{\theta} \cdot \Delta x_{cw} - \dot{\varphi} \cdot \cos(\varphi) \cdot \text{tr} \end{aligned} \quad (3.9)$$

Equation 3.9 can be transformed into

$$\dot{\varphi} = -\frac{1}{\text{tr}} \cdot [(v - \omega \cdot \Delta y_{cw}) \cdot \sin(\varphi) - (\omega \cdot \Delta x_{cw}) \cdot \cos(\varphi)] \quad (3.10a)$$

$$\dot{\gamma} = \frac{1}{r_{cw}} \cdot [(v - \omega \cdot \Delta y_{cw}) \cdot \cos(\varphi) + (\omega \cdot \Delta x_{cw}) \cdot \sin(\varphi)] \quad (3.10b)$$

Notice that equations 3.10a and 3.10b are applied to the front left caster wheel and can be extended to other wheels by varying the signs of  $\Delta x_{cw}$  and  $\Delta y_{cw}$ .

### Equilibrium state

Equations 3.10a and 3.10b describe the rotation and rolling angles  $\varphi, \gamma$  of the caster wheels. Getting the analytical expressions of the respective equilibrium angles provides information about its steady state values for given velocity commands  $v, \omega$ . This is of great use for anticipating to undesired caster wheel motions, which might provoke an increase in bore torques, and consequently, in motor torques. Hence, the equations obtained in this subsection, along with 3.10a and 3.10b, play a crucial role in the formulation of the MPC based local planners.

#### Rotation angle

This term defines the angle at which the caster wheel stops rotating and acquires a steady state value for given input commands ( $v$  and  $\omega$ ). It can be obtained by taking the derivative of Equation 3.10b with respect to  $\varphi$  and making it equal to 0.

$$\frac{\partial \dot{\gamma}}{\partial \varphi} = \frac{1}{r_{cw}} \cdot [-(v - \omega \cdot \Delta y_{cw}) \cdot \sin(\varphi_{ss}) + (\omega \cdot \Delta x_{cw}) \cdot \cos(\varphi_{ss})] = 0 \quad (3.11)$$

the caster wheel's steady state angle can be described as:

$$\varphi_{ss} = \text{atan} \left( \frac{\omega \cdot \Delta x_{cw}}{v - \omega \cdot \Delta y_{cw}} \right) \quad (3.12)$$

#### Rolling angle

Combining Equation 3.10b with the term obtained from Equation 3.12 leads to the definition of the caster-wheel's rolling speed once it rotates to a steady state angle.

$$\dot{\gamma}_{ss} = \frac{1}{r_{cw}} \cdot [(v - \omega \cdot \Delta y_{cw}) \cdot \cos(\varphi_{ss}) + (\omega \cdot \Delta x_{cw}) \cdot \sin(\varphi_{ss})] \quad (3.13)$$

Considering the following trigonometric relationships

$$\begin{cases} \sin(\text{atan}(x)) = \frac{x}{\sqrt{1+x^2}} \\ \cos(\text{atan}(x)) = \frac{1}{\sqrt{1+x^2}} \end{cases} \quad (3.14)$$

Equation 3.13 can be simplified into

$$\dot{\gamma}_{ss} = \frac{1}{r_{cw}} \cdot \sqrt{(v - \omega \cdot \Delta y_{cw})^2 + (\omega \cdot \Delta x_{cw})^2} \quad (3.15)$$

#### **3.1.2.2 Dynamics of the caster wheel**

Even if Equations 3.10a and 3.10b, provide very valuable information about the caster wheel's motion, the counter acting force that it generates, also defined as bore torque, remains unknown. According to paper [5] this reaction force is given by Equation 2.1.

### **3.1.3 Formulation of the plant model**

The AS can be divided into two independent subsystems: a DDMR and four caster wheels. The kinematics and dynamics of both subsystems have been developed. Putting them together leads to the plant model that is going to be used to formulate a MPC based local planner.

Notice that the DDMR's ODE system given in Section 3.1 has been extended to the accelerations domain. This gives the chance to penalize longitudinal and lateral accelerations, ensuring a smooth navigation.

Combining this system with the caster wheel angle's equation of motion defined in 3.10a and applied to the front left and right caster wheels, results in a seven state model that takes longitudinal and angular

accelerations as inputs.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \\ \dot{x}_5 \\ \dot{x}_6 \\ \dot{x}_7 \end{bmatrix} = \begin{bmatrix} x_4 \cdot \cos(x_3) \\ x_4 \cdot \sin(x_3) \\ x_5 \\ u_1 \\ u_2 \\ -\frac{1}{\text{tr}} \cdot [(x_4 - x_5 \cdot \Delta y_{\text{cw}}) \cdot \sin(x_6) - (x_5 \cdot \Delta x_{\text{cw}}) \cdot \cos(x_6)] \\ -\frac{1}{\text{tr}} \cdot [(x_4 + x_5 \cdot \Delta y_{\text{cw}}) \cdot \sin(x_7) - (x_5 \cdot \Delta x_{\text{cw}}) \cdot \cos(x_7)] \end{bmatrix} \quad (3.16a)$$

where

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \\ v \\ \omega \\ \varphi_{\text{FL}} \\ \varphi_{\text{FR}} \end{bmatrix}, \quad \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} a \\ \alpha \end{bmatrix} = \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} \quad (3.16b)$$

## 3.2 Objective function

In order to implement the first input sample according to the optimal forecast, each prediction is evaluated according to a cost function. In other words, the cost function establishes the criteria for evaluating every prediction. As explained in Section 2.3, the research question defines two main goals for the motion planner's formulation: navigating as close as possible to the global path and considering caster wheel reaction torques. Therefore, the objective function consists of two terms. One of them ensures that the robot follows the desired path, while the other is in charge of minimizing caster wheel reaction torques. This concept has already been summarized in Equation 2.4.

### 3.2.1 Navigation term

In Section 2.3 it has already been mentioned that motion planning is simplified to path following. When doing so, references to papers that apply MPC for solving path following and trajectory tracking problems have been analyzed. In order to ensure that the robot navigates across the global path, a simplified version of the trajectory tracking method proposed in [25] is applied. According to this approach, the robot is obliged to follow the global path by penalizing the distance from the robot to a virtual time based reference.

The global path and the reference's navigation velocity are assumed to be known. This information is used to discretize the global path and convert it into a grid of points with time labels. Once this is done, the reference navigates across the grid through time interpolation. This procedure can be visualized in Figure 3.4.

The time based reference,  $r_k$ , is a set of points with a length equal to  $N + 1$ , where  $N$  is the control horizon. By penalizing the distance from the robot to the reference, it is forced to chase it. For this purpose, the following term needs to be included in the OCP's cost function.

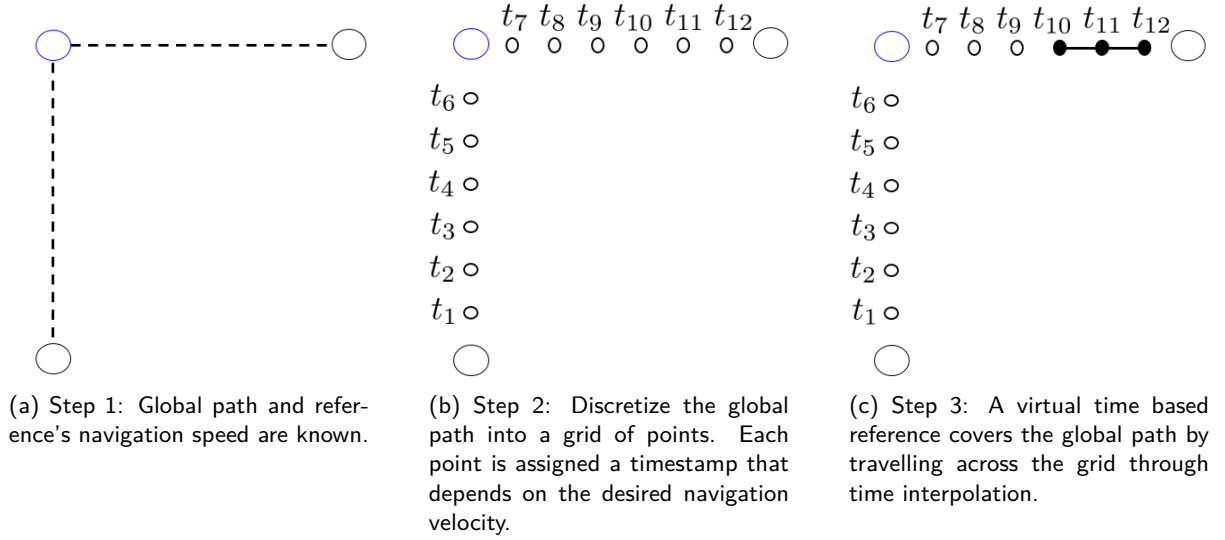


Figure 3.4: Procedure to implement a virtual time based reference in a L shape global path. Notice that the reference given in Step 3 is a simplified version and it is intended only for illustration.

$$\forall k \in \{1, 2, \dots, N + 1\}$$

$$e_k = r_k - x_k = \begin{bmatrix} x_{\text{goal},k} \\ y_{\text{goal},k} \\ \theta_{\text{goal},k} \end{bmatrix} - \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}. \quad (3.17)$$

In order to fully define this term, the functioning principle of the time based reference,  $r_k$ , has to be explained. Apart from the discretization given in Figure 3.4, it relies on a chopping procedure that re-organizes the path received from the global planner. Before getting into the details about the time interpolation strategy, the procedure of restructuring the global path will be explained.

### 3.2.1.1 Processing of global path

The path given by the global planner will be denoted as a *trajectory* and will be considered to be defined as a set of points. These points can be separated into *check-points* and *goal-points*. The latter ones divide the *trajectory* into *sections*. If a *section* does not contain any *check-points*, the *section* will be a single *line* that connects both *goal-points*. If a *section* contains  $q$  *check-points*, the *section* will be defined by  $q + 1$  *lines*. Furthermore, each *section* is related to a navigation velocity, which is considered to be given. Looking into Figure 3.5 might be helpful to visualize these concepts.

The time based reference will navigate along each of the *sections*. Once it gets to the end, it will wait for the robot to arrive. As soon as this happens, the moving reference will start navigating through the following *section*. Every time that the reference gets into a new *section*, it updates its speed. This procedure is repeated until the reference gets to the end of the last *section*.

### 3.2.1.2 Time based reference

The time reference consists of  $N + 1$  points  $R_{0,1,\dots,N+1}^t = (r_0^t, r_1^t, \dots, r_{N+1}^t)$ , where each of them has three fields  $r_0^t = (x_0^t, y_0^t, \theta_0^t)$ . Every time step  $t_s$ ,  $r_0^t$  and  $r_{N+1}^t$  are updated. In order to update  $r_0^t$ , it is enough to substitute with the second point of the reference ( $r_0^{t+t_s} = r_1^t$ ). Updating  $r_{N+1}^t$  is not that simple. For

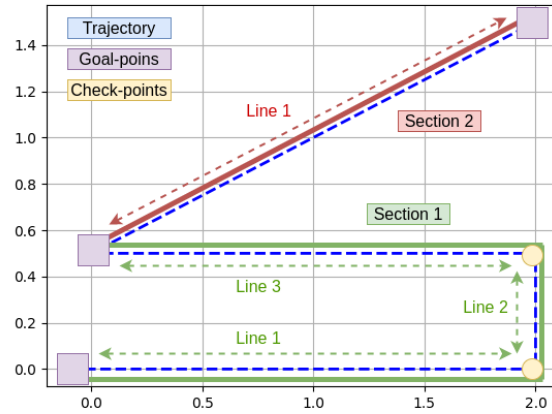


Figure 3.5: Example of a *trajectory* given by the global planner that has been divided into *sections*, according to its *goal-points* and *check-points*. At the same time, a *section* contains several *lines*. This division dictates the behaviour of the time-based reference that the robot tries to follow.

this purpose, each *section* is divided into a set of grid-points before the navigation starts, as depicted in Figure 3.4. Depending on the *section's* navigation speed, a time is assigned to each point that is part of this grid. During navigation, the reference's last value  $r_{N+1}^t$  is updated through time interpolation along the grid points. Once  $r_0^{t+t_s}$  and  $r_{N+1}^{t+t_s}$  are defined, the rest of the reference points are linearly distributed.

In this way, the time based reference follows every *line* of the *section* with the assigned navigation speed. Once it gets to the end of a *line*, it will either continue or stop. This depends on the point-group to whom the end of the *line* belongs. If it is a *check-point*, the reference will be updated and it will continue navigating through the next *line*. If it is a *goal-point*, the reference will wait until the robot gets close enough. Once this condition is met, the reference will update and the navigation will continue. This loop keeps going until the the robot gets to the last *goal-point*. As a consequence, *goal-points* are mandatory locations that the robot cannot miss, while *check-points* only define the trajectory's shape. This process is summarized in the following algorithm:

### 3.2.1.3 Formulation of navigation term

Assigning a weight,  $Q_{\text{nav}}$ , to the distance between the robot and the time based reference given in Equation 3.17, results in the cost function's term that is in charge of the navigation part.

$$J_{\text{navigation}} = \sum_{k=1}^{N+1} \left\| \begin{bmatrix} x_{\text{goal},k} - x_{1,k} \\ y_{\text{goal},k} - x_{2,k} \\ \theta_{\text{goal},k} - x_{3,k} \end{bmatrix} \right\|_{Q_{\text{nav}}}^2 \quad (3.18)$$

### 3.2.2 Caster wheel aware term

Once the navigation term of the cost function has been defined, the term that gives the local planner awareness of the caster wheels is formulated. Consequently, the content exposed in this subsection is highly relevant to the thesis' outcome.



---

Time based reference

---

**Initialization:**

```

grid = discretize_global_path(global_path)
labeled_grid = assign_time_stamps(navigation_speed, grid_points)
section_finished = false
cont_section = 0

```

**Navigate:**

```

while True do
  t = t + t_s
  if section_finished = false then
    r_{N+1}^t = time_interpolation(labeled_grid, t)
  end if
  r_0^t = r_1^{t-t_s}
  r_1^t, ..., r_N^t = linear_distribution(r_0^t, r_{N+1}^t)
  goal_point = update_goal_point(cont_section)
  if r_{N+1}^t = goal_point then
    section_finished = true
  else if |position_robot - goal_point| ≤ tolerance then
    section_finished = false
    cont_section = cont_section + 1
  end if
end while

```

---

According to the case identification in Section 2.4, peak reaction torques arise when caster wheels are misaligned with respect to the robot's driving direction ( $\Delta\varphi > 0$ ). In other words, if caster wheels are kept aligned with respect to the navigation's direction, reaction torques are not critical and severe consequences are omitted.

Therefore, the most intuitive approach to tackle this problem is by penalizing the difference,  $\Delta\varphi$ , between steady state and current caster wheel rotation angles ( $\varphi_{ss}$ ,  $\varphi$ ). In this way, it can be guaranteed that the alignment mentioned in the previous paragraph is fulfilled. Both elements of the subtraction can be visualized in Figure 3.6. In this Section it will be assumed that the current or real caster wheel states,  $(\varphi, \dot{\gamma})$  are known and the focus will be put in the steady states  $(\varphi_{ss}, \dot{\gamma}_{ss})$ .

The mentioned difference is expressed by

$$\Delta\varphi = \varphi_{ss} - \varphi \quad (3.19a)$$

where  $\varphi_{ss}$  is given in Equation 3.12 and represents the steady state caster wheel rotation angle for a given set of input commands  $(v, \omega)$ . Since this expression includes an atan, which is only defined in the range of  $[-\pi/2, \pi/2]$ , and it is desired to project the rotation angle in an entire revolution, atan is replaced by atan2. This results in extending Equation 3.12 as follows:

$$\varphi_{ss} = \text{atan2} \left( \frac{\omega \cdot \Delta x_{cw}}{v - \omega \cdot \Delta y_{cw}} \right) \quad (3.19b)$$

Despite that substituting atan with atan2 has succeeded in translating  $\varphi_{ss}$ , into a range within  $[-\pi, \pi]$ , it has the disadvantage of being discontinuous in the origin (see Figure 3.7). Moreover, the fraction inside atan2 is not defined in the entire input range. More specifically, the fraction becomes infinite if both longitudinal and rotational velocity commands,  $(v, \omega)$  are 0. Due to these two facts, such a term is not numerically compatible with an OCP solver. Even if penalizing the angle difference given in Equation 3.19a might have a positive

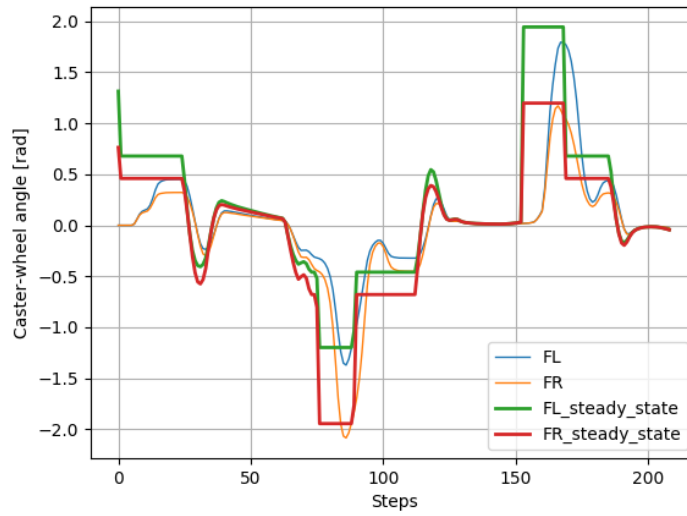


Figure 3.6: Caster wheel rotation angles,  $\varphi$  (blue and orange) and their respective steady state angles (green and red). Notice that the steady state values are specific for each set of input commands

impact in the mitigation of caster wheel reaction torques, it numerically is problematic.

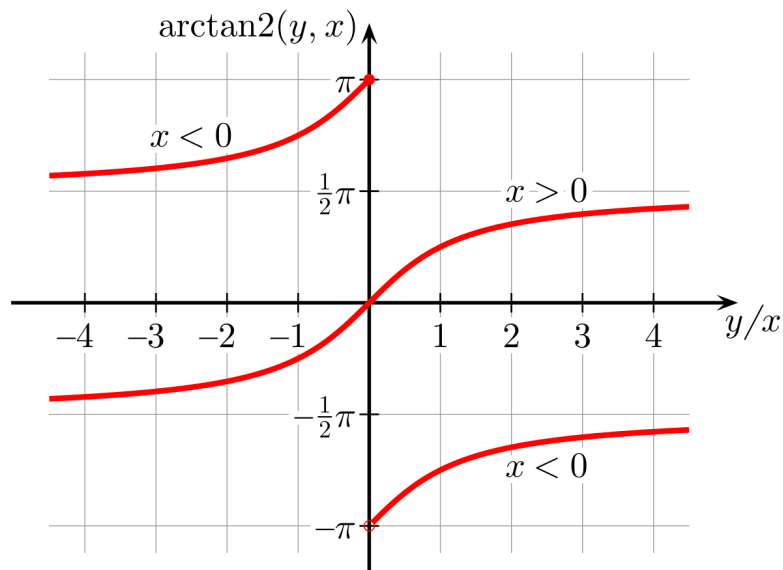


Figure 3.7: Graphic of atan2. Notice that its domain is  $[-\pi, \pi]$  and it is discontinuous in the origin.

Instead of penalizing the difference in rotation angles, a workaround is to penalize the difference in rolling speeds  $\Delta\dot{\gamma}$ . For this purpose, the analytical expression of  $\dot{\gamma}_{ss}$  has to be applied. This has already been derived in Equation 3.15 by combining  $\varphi_{ss}$  with the Equation of motion of the caster wheel's rolling speed,  $\dot{\gamma}$ .

When doing so, the numerical issues related to the atan2 and the fraction have been omitted and replaced by a square root that is defined and finite in the input's entire domain. In order to make it also differentiable, a small constant  $\epsilon$  is added. Otherwise, its derivative in the origin would be infinite.

$$\Delta\dot{\gamma} = \dot{\gamma}_{ss} - \dot{\gamma} \quad (3.20a)$$

$$\dot{\gamma}_{ss} \approx \frac{1}{r_{cw}} \cdot \sqrt{(v - \omega \cdot \Delta y_{cw})^2 + (\omega \cdot \Delta x_{cw})^2} + \varepsilon \quad (3.20b)$$

### 3.2.2.1 Formulation of caster wheel aware term

Adding a term to the cost function that accounts for the difference between steady state and current caster wheel rolling speeds guarantees that changes in orientation are smooth, hindering motions that generate reaction torques. The weight allocated to this penalization is  $Q_{cw}$ .

$$J_{cw} = \sum_{k=1}^{N+1} \|\Delta \dot{\gamma}_k\|_{Q_{cw}}^2 \quad (3.21)$$

### 3.2.3 Formulation of objective function

Adding the recently formulated navigation and caster wheel aware terms and including a third term that minimizes the energy introduced in the system results in the following cost function:

$$J = \sum_{k=1}^{N+1} \|p_{robot,k} - p_{ref,k}\|_{Q_{nav}}^2 + \|\Delta \dot{\gamma}_k\|_{Q_{cw}}^2 + \|u_k\|_{Q_u}^2 \quad (3.22)$$

In this way, each prediction is going to be evaluated according to the distance between the robot's trajectory to the global path (1), the difference between the caster wheel's real and steady state rolling speeds (2) and the energy consumption (3). The relevance of each goal depends on the ratio between the weights. The first sample of the input corresponding to the prediction that deals best with these three objectives is the one applied in the robot.

## 3.3 Observer

In the previous Section it has been assumed that the caster wheel's angle,  $\varphi$ , is known. However, the AS has no sensors that provides such information, which implies the need of an observer. The estimation takes place by integrating another model, denoted as "observer model", in parallel with the plant model. Figure 3.8 shows how the observer is implemented:

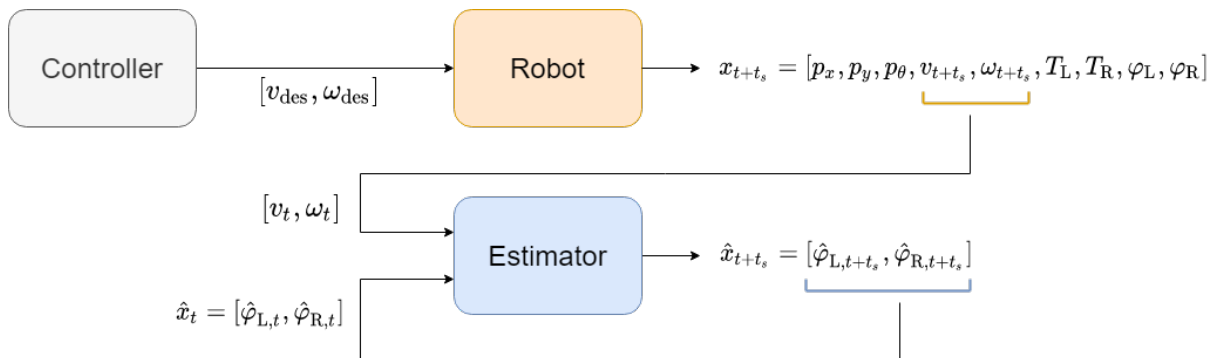


Figure 3.8: Implementation of the estimator.

Feeding the estimations from the last iteration  $[\hat{\varphi}_{L,t}, \hat{\varphi}_{R,t}]$ , along with measured longitudinal and rotational

velocities  $[v_t, \omega_t]$ , to the observer is sufficient to obtain a new set of estimations. Notice that the velocities that the estimator takes as inputs are velocities measured by odometry. Since the previous estimations need also to be fed, this implementation involves knowing the initial value of the caster-wheel angles. There are two different ways to tackle this. Even if the assumption that these are known could be made, it would be possible to start navigating with constant command velocities until the caster-wheels align with respect to the equilibrium angle, which can be derived from Equation 3.12.

### 3.3.1 Observer model

The observer consists on integrating the equation of motion respective to the caster wheel's rotation angle derived in Section 3.1.2 and given in Equation 3.10a applied to the front left and right caster wheels.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} -\frac{1}{tr} \cdot [(u_1 - u_2 \cdot \Delta y_{cw}) \cdot \sin(x_1) - (u_2 \cdot \Delta x_{cw}) \cdot \cos(x_1)] \\ -\frac{1}{tr} \cdot [(u_1 + u_2 \cdot \Delta y_{cw}) \cdot \sin(x_2) - (u_2 \cdot \Delta x_{cw}) \cdot \cos(x_2)] \end{bmatrix} \quad (3.23a)$$

where

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \hat{\varphi}_L \\ \hat{\varphi}_R \end{bmatrix}, \quad \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix} \quad (3.23b)$$

### 3.3.2 Stability analysis

The purpose of this Section is to proof that the steady state rotation angle,  $\varphi_{ss}$  is an asymptotically stable point for the estimator presented above. Since the estimator's model consists of two analogous ODEs based on the caster-wheel kinematics given in Equation 3.10a, demonstrating its stability is enough. The demonstration will be done for the front left caster wheel and is applicable to any of the four wheels.

#### 3.3.2.1 Stability of caster wheel's rotation angle

From the caster-wheel's kinematics the Equation of motion respective to its rotation angle can be derived. This has already been done Section 3.1.2, which led to Equation 3.10a. For the sake of simplicity, it will be simplified to the following form:

$$\dot{x} = f(x, u) = -a \cdot [(u_1 - u_2 \cdot b) \cdot \sin(x) - (u_2 \cdot c) \cdot \cos(x)] \quad (3.24a)$$

, where

$$x = \varphi, \quad \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} v \\ \omega \end{bmatrix}, \text{ and } \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \frac{1}{tr} \\ \Delta y_{cw} \\ \Delta x_{cw} \end{bmatrix} \quad (3.24b)$$

The equilibrium points,  $\bar{x}$  can be obtained by  $\dot{x} = 0$ , which results in

$$\bar{x} = \text{atan} \left( \frac{\bar{u}_2 \cdot c}{\bar{u}_1 - \bar{u}_2 \cdot b} \right) \quad (3.25)$$

Notice that this expression is equivalent to Equation 3.12. Considering that inputs are limited, all possible

equilibrium angles can be computed, as shown in Figure 3.9. According to this graph, any value of velocity commands will make the front left caster wheel converge to an angle between  $-1.19$  rad and  $1.97$  rad. The alternative representation shown in the Figure 3.10 might be helpful to visualize this concept.

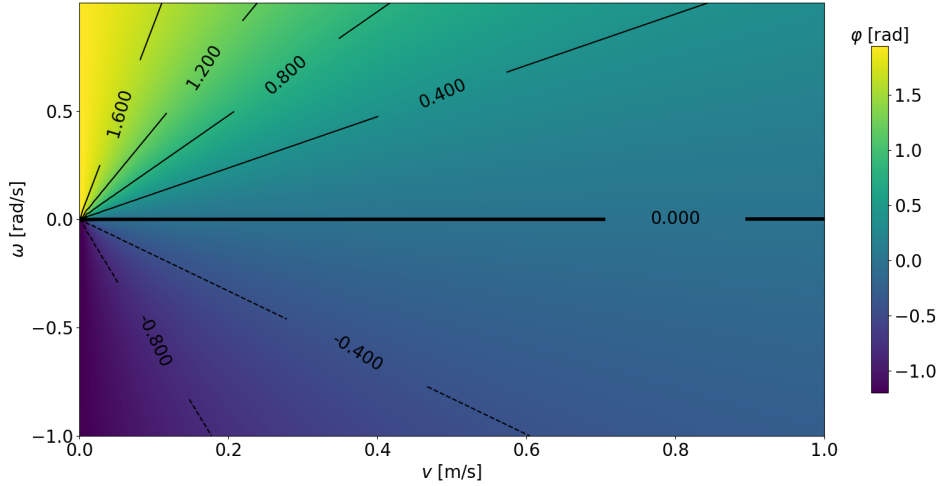


Figure 3.9: Possible equilibrium front-left caster-wheel angles,  $\bar{\phi}$ , depending on longitudinal and lateral velocity commands,  $v, \omega$ .

No matter which are the velocity commands, the front left caster-wheel's rotation will converge to an angle within the green area. In other words, any angle between  $-1.19$  rad and  $1.97$  rad could be an equilibrium angle and each of them has its respective set of velocity commands  $\bar{v}, \bar{\omega}$ .

### 3.3.2.2 Stability of caster wheel angle's error

In subsection 3.3.2.1 it has been shown that the caster-wheel's rotation angle has got multiple equilibrium points. Since this implies that convergence should be demonstrated for each of them, it is convenient to find an alternative representation with fewer equilibrium points. That is why, Equation 3.24a is going to be converted, so that it represents the difference between the estimated and equilibrium caster-wheel angle.

$$e = x - \bar{x} \quad (3.26a)$$

$$\dot{e} = \dot{x} - \dot{\bar{x}} \quad (3.26b)$$

Considering that the equilibrium point is given for specific inputs,

$$\dot{\bar{x}} = 0 \quad (3.26c)$$

$$\dot{e} = \dot{x} = f(x, u) = f(e + \bar{x}, u) = g(e, u) \quad (3.26d)$$

$$\dot{e} = -a \cdot [(u_1 - u_2 \cdot b) \cdot \sin(e + \bar{x}) - (u_2 \cdot c) \cdot \cos(e + \bar{x})] \quad (3.26e)$$

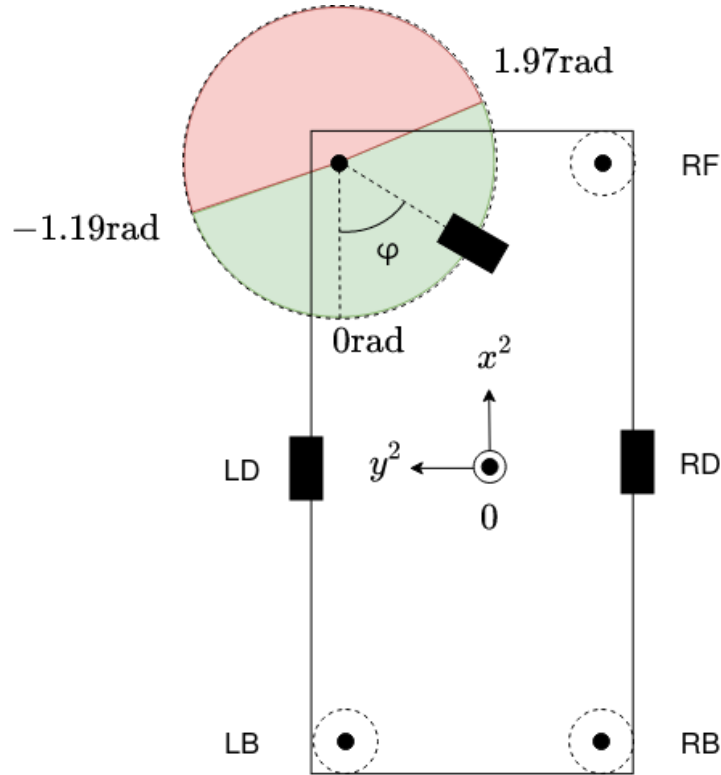


Figure 3.10: Green area represents front-left caster-wheel angle's equilibrium range,  $\bar{\varphi} = \bar{x}$ , for any set of velocity commands  $(v, \omega)$  that are within the limits

Using the trigonometric properties given in Equation 3.14 enables to simplify Equation 3.26e to

$$\dot{e} = g(e, u) = -a \cdot \sqrt{(u_1 - u_2 \cdot b)^2 + (u_2 \cdot c)^2} \cdot \sin(e) = -d \cdot \sin(e) \quad (3.26f)$$

where  $d$  is a positive real number. Equation 3.26e represents the difference between estimated and equilibrium caster-wheel angle. If this Equation converges to  $e = 0$  for any initial starting state  $e_0 \neq 0$ , the estimated angle converges to the respective steady state angle, thus, the observer is asymptotically stable for every equilibrium angle shown in Figures 3.9 and 3.10. This will be proved by applying two lemmas and a theorem.

### Proof of stability

From  $\dot{e} = 0$ , equilibrium points  $\bar{e}_1 = 0$  and  $\bar{e}_2 = \pi$  are obtained. Following the argument given above, the proposed observer is asymptotically stable if Equation 3.26e is unstable for  $e = \pi$  (lemma 1) and stable for  $e \neq \pi$  (lemma 2). Since  $e = \pi$  is an equilibrium angle, the error will stay in  $\pi$ , unless the system is externally disturbed. Adding oscillations to the estimated angles will avoid  $e$  from staying in  $\pi$  and will ensure its convergence from  $e = \pi$  to  $e = 0$  (theorem 1).

#### Lemma 1

- **Statement:** "Equation 3.26e is unstable for  $e = \pi$ ".

- **Proof:** Equilibrium around  $e = \pi$  can be analyzed by linearizing Equation 3.26e at this point and looking

into its eigenvalues. If these are strictly positive, Equation 3.26e is unstable in  $e = \pi$ .

$$\dot{e} = g(e, u) = A \cdot \partial e + B \cdot \partial u \quad (3.27a)$$

where  $A = \frac{\partial f(e, \dot{e}, u)}{\partial e}$  and  $B = \frac{\partial f(e, \dot{e}, u)}{\partial u}$ . The eigenvalues of Equation 3.26e in  $e = \pi$  are defined by

$$\lambda = \text{eig}(A) = \text{eig}\left(\left[\frac{\partial f(e = \pi, \dot{e} = 0, u)}{\partial e}\right]\right) = \text{eig}(-d \cdot \cos(e)) = d \quad (3.27b)$$

Since  $d = a \cdot \sqrt{(u_1 - u_2 \cdot b)^2 + (u_2 \cdot c)^2}$ , it can be concluded that

$$\lambda(e = \pi) = d > 0, \forall u_1, u_2 \in U \quad (3.27c)$$

Consequently, it can be generalized that Equation 3.26e is unstable in  $e = \pi$  for any longitudinal and rotational velocity commands within the limits.

#### Lemma 2

- **Statement:** "Equation 3.26e is stable  $\forall e \neq \pi$ ".

- **Proof:** Equation 3.26e is a nonlinear system, whose asymptotic stability  $\forall e \in (-\pi, \pi)$  with respect to  $e = 0$  can be proofed by the existence of a Lyapunov function,  $V(e)$ , with a strictly negative total derivative [27]. This function needs to fulfill the following conditions:

$$V(e) = \begin{cases} V(e) > 0, & \forall e \in (-\pi, \pi) \wedge e \neq 0 \\ V(e) = 0, & e = 0 \\ \frac{dV(e)}{dt} < 0, & \forall e \in (-\pi, \pi) \end{cases} \quad (3.28a)$$

$$(3.28b)$$

$$(3.28c)$$

Consider the following Lyapunov function

$$V(e) = \frac{1}{2} \cdot e^2 \quad (3.29)$$

which fulfills requirements 3.28a and 3.28b. Taking the derivative of Equation 3.29 and combining it with 3.24a, results in Equation 3.30, which is strictly negative  $\forall e \in (-\pi, \pi)$ , thus, it also satisfies Equation 3.28c.

$$\dot{V}(e) = e \cdot \dot{e} = -e \cdot d \cdot \sin(e) \quad (3.30)$$

Since the Lyapunov function proposed in 3.29 accomplishes conditions 3.28a, 3.28b and 3.28c, Equation 3.26e is asymptotically stable  $\forall e \in (-\pi, \pi)$ .

#### Theorem 1

- **Statement:** "The observer proposed in 3.23a is asymptotically stable  $\forall e \in [-\pi, \pi]$  at  $\varphi_{ss}$ , if small oscillations are added to its output."

- **Proof:** Lemma 1 and Lemma 2 have concluded that the estimator is unstable for  $e = \pi$  and asymptotically

stable  $\forall e \in (-\pi, \pi)$  with respect to  $e = 0$ . Since  $e = \pi$  is an equilibrium point, the current estimator does not converge from  $e = \pi$  to  $e = 0$ . However, this point is unstable, which means that a small disturbance is enough to make it converge. That is why, adding small oscillations to the estimated angles will make the estimator asymptotically stable  $\forall e \in [-\pi, \pi]$ . Equation 3.24a can be modified, so that it includes these oscillations.

$$\dot{x} = -a \cdot [(u_1 - u_2 \cdot b) \cdot \sin(x) - (u_2 \cdot c) \cdot \cos(x)] + \zeta \cdot \sin(w_\zeta \cdot t) \quad (3.31)$$

where term  $\zeta \cdot \sin(w_\zeta \cdot t)$  represents oscillations with an amplitude of  $\zeta$  and an angular frequency of  $w_\zeta$ . It is relevant to highlight that the oscillation's amplitude has to be minimized, so that the impact on the estimator's performance is negligible, while being able to deviate the caster-wheel angle from  $e = \pi$ .

In order to demonstrate this theorem, a new case has been studied, where the robot drives longitudinally ( $v = 0.5\text{m/s}$ ,  $\omega = 0\text{rad/s}$ ) with an offset of  $\pi\text{rad}$  in the caster-wheel angle's initialization. Consequently, the difference between estimated and equilibrium angles is  $e = -\pi$ . As it can be seen in Figure 3.11, only for the case where the oscillations are included (green), the error converges from  $e = \pi$  to  $e = 0$ .

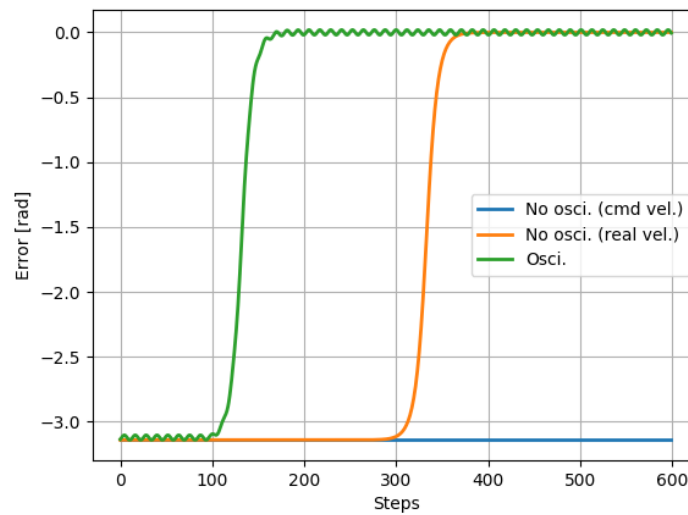


Figure 3.11: Comparison of convergence for two estimators, without (blue) and with (green) oscillations, initialized at  $\pi$  rad. Orange line represents the estimator's convergence with no oscillations, but real velocities as inputs.

For this demonstration the estimator's layout exposed in Figure 3.8 has been modified. According to this diagram, the estimator takes the robot's real velocities as inputs. In order to avoid deviations from  $e = \pi$ , these have been replaced by the command velocities. Otherwise, the variations included in the velocity measures disturb the system, which make it converge to  $e = 0$ , as the orange line indicates. Notice that this convergence is slower than the one achieved by adding oscillations.

#### Outcome of proof

By transforming the observer's Equation of motion into an expression that stands for the error,  $e$ , between steady state and real angle, it has been demonstrated that the caster wheel's steady state angle,  $\varphi_{ss}$ , is an asymptotically stable point in the observer's system for any set of commands within the range.



### 3.4 Horizon and sampling time

The plant model over which predictions are made, together with an objective function that evaluates them and an observer that estimates caster wheel states, have already been defined. In this Section, the length of time taken into account when making predictions is determined.

This period is denoted as "time horizon",  $T$ , and the quantity of input sets applied during this interval is defined as "control horizon",  $N$ . The ratio between them is given by the sampling time,  $t_s = \frac{T}{N}$ . By taking its inverse, the frequency of the MPC can be obtained.

$$f_{\text{MPC}} = \frac{1}{t_s} = \frac{N}{T} \quad (3.32)$$

The MPC needs to run faster than the dynamics that is trying to control. A rule of thumb is that the controller's rate needs to be five to ten times higher than the system's fastest pole [28]. Consequently, the choice of time and control horizon will depend on the plant model's poles.

#### 3.4.1 Sampling time

The AS's plant model consists of two subsystems: the DDMR and the caster wheels. Since the first one is made out of pure integrators, only the poles of the latter are relevant. These can be obtained by linearizing Equation 3.10 and looking into its eigenvalues.

$$\dot{\varphi} = f(\varphi, u) = A \cdot \partial\varphi + B \cdot \partial u \quad (3.33a)$$

where  $A = \frac{\partial f(\varphi, u)}{\partial \varphi}$  and  $B = \frac{\partial f(\varphi, u)}{\partial u}$ . The eigenvalues of Equation 3.10 in  $\varphi = \varphi^*$  for inputs  $v = v^*$  and  $\omega = \omega^*$  are defined by

$$\lambda = \text{eig}(A) = \text{eig}\left(\left[\frac{\partial f(\varphi = \varphi^*, u = u^*)}{\partial \varphi}\right]\right) \quad (3.33b)$$

$$\lambda = -\frac{1}{\text{tr}} \cdot [(v^* - \omega^* \cdot \Delta y_{\text{cw}}) \cdot \cos(\varphi^*) + (\omega^* \cdot \Delta x_{\text{cw}}) \cdot \sin(\varphi^*)] \quad (3.33c)$$

Considering the bounds of the inputs, the fastest eigenvalue is

$$\lambda_{\text{max}} = \frac{1}{\text{tr}} \cdot (|v_{\text{max}}| + |\omega_{\text{max}}| \cdot \Delta y_{\text{cw}}) \quad (3.33d)$$

which can be converted into the respective frequency of the caster wheel's rotation angle

$$f_{\varphi, \text{max}} = \frac{\lambda_{\text{max}}}{2\pi} = \frac{1}{2\pi \cdot \text{tr}} \cdot (|v_{\text{max}}| + |\omega_{\text{max}}| \cdot \Delta y_{\text{cw}}) \quad (3.33e)$$

Applying Equation 3.33e to the AS's dimensions and velocity limits results in a frequency of 3.08Hz. Hence, in order to ensure that the caster wheels motions are addressed at every moment and no aliasing occurs, MPC needs to run at a frequency between 15Hz and 30Hz.

The eigenvalues obtained from computing Equation 3.33b for a grid of inputs are given in Figure 3.12. The cloud of eigenvalues has been classified according to their respective rotation angle  $\varphi$  and velocity commands  $v, \omega$ . The fastest eigenvalues (red color) take place for maximum velocity commands and rotation angles of

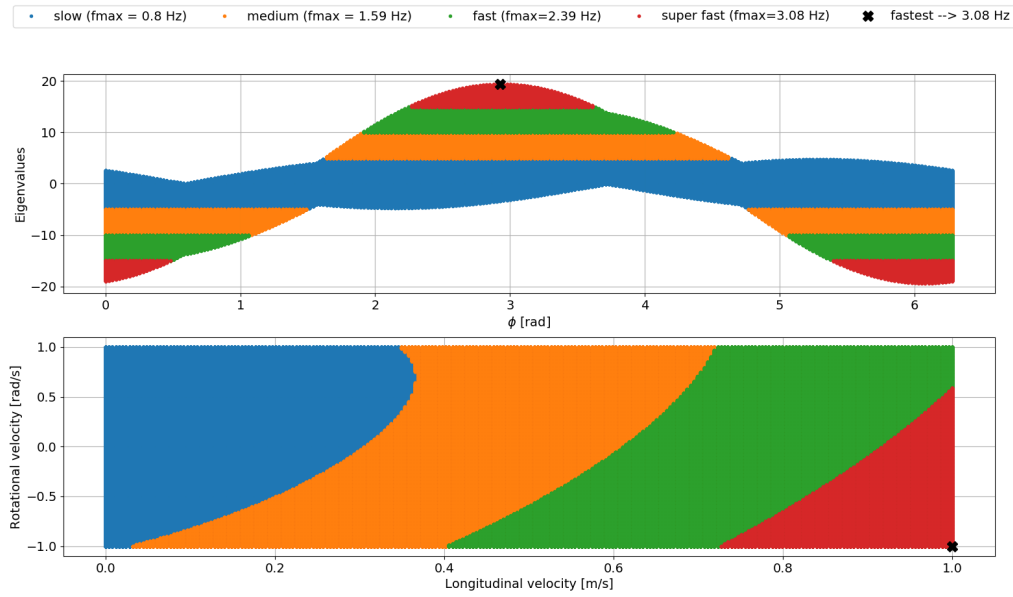


Figure 3.12: Eigenvalues classified according to its respective longitudinal and rotation velocities,  $V$ ,  $W$  and caster wheel's rotation angle,  $\varphi$ . The colors divide the eigenvalues depending on its speed.

$\varphi = 0$  and  $\varphi = \pi$ . The fastest eigenvalue has been marked with a black cross and proves the correctness of Equation 3.33e.

### 3.4.2 Time and control horizon

Since, the AS's focus is set in intralogistics, the local planner needs to deal with unexpected static and dynamic obstacles. Consequently, its time horizon has to be sufficiently long to find a way through the obstacles, while staying as close as possible to the global path. Extending the horizon increases the computation load and memory overhead and does not guarantee a better control performance [29]. Consequently, finding the right balance between performance and computation overhead is crucial to define these two parameters.

Considering that the longitudinal and angular speeds are limited to 1 m/s and 1 rad/s and the behaviour of potential obstacles in an industrial site, predictions between 2 s and 5 s are sufficient. A time horizon lower than 1 s might lead to instabilities, while going over 5 s increases computational load without having a significant contribution to the planner's performance.

Due to the frequency range obtained out of Equation 3.33e, the facts given in the previous paragraphs and restrictions on computational capacity, a frequency of  $f_{MPC} = 20$  Hz with a time horizon of  $T = 2$  s have been chosen. From Equation 3.32, it is known that this is equivalent to a control horizon of  $N = 40$ .

## 3.5 Constraints

The constraints defined in the OCP ensure that the MPC's optimal output is not out of the robot's operation range. These conditions can also guarantee other requirements, such as safety or hardware limitations. That is why, a total of four constraints have been added to the OCP.

The robot's longitudinal and rotational velocities always remain within the bounds, thanks to the following

expressions.

$$v_{\min} \leq v \leq v_{\max} \quad (3.34a)$$

$$w_{\min} \leq \omega \leq w_{\max} \quad (3.34b)$$

For the case of the AS,  $[v_{\min}, v_{\max}] = [0, 1]$  and  $[w_{\min}, w_{\max}] = [-1, 1]$ . Furthermore, due to the robot's nature, accelerations are also limited by their respective mass and inertias. Moreover, the motors have restrictions on rotational accelerations. These are also taken into account in the OCP, by applying the following restrictions:

$$a_{dw,\min} \leq a - \alpha \cdot \Delta y_{dw} \leq a_{dw,\max} \quad (3.34c)$$

$$a_{dw,\min} \leq a + \alpha \cdot \Delta y_{dw} \leq a_{dw,\max} \quad (3.34d)$$

where  $a_{dw}$  is the longitudinal acceleration in the driving wheel's center of rotation (point B in Figure 3.3, applicable to left and right driving wheels).

## 3.6 Formulation of OCP

The plant model presented in Section 3.1 evaluated according to the cost function in Section 3.2 over the horizon chosen in Section 3.4 and considering constraints in Section 3.5 leads to the definition of the OCP that can be solved by the MPC:

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_N} \sum_{k=1}^{N+1} \left\| \begin{bmatrix} x_{\text{goal},k} - x_{1,k} \\ y_{\text{goal},k} - x_{2,k} \\ \theta_{\text{goal},k} - x_{3,k} \end{bmatrix} \right\|_{Q_{\text{nav}}}^2 + \left\| \begin{bmatrix} \dot{\gamma}_k^l - \dot{\gamma}_{\text{ss},k}^l \\ \dot{\gamma}_k^r - \dot{\gamma}_{\text{ss},k}^r \end{bmatrix} \right\|_{Q_{\text{cw}}}^2 + \|\mathbf{u}_k\|_{Q_u}^2 \quad (3.35a)$$

$$\text{subject to } \mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{u}_k), \quad (3.35b)$$

$$v_{\max} \geq x_{4,k} \geq v_{\min}, \quad (3.35c)$$

$$w_{\max} \geq x_{5,k} \geq w_{\min}, \quad (3.35d)$$

$$a_{dw,\max} \geq u_{1,k} - u_{2,k} \cdot \frac{\Delta y_{dw}}{2} \geq a_{dw,\min}, \quad (3.35e)$$

$$a_{dw,\max} \geq u_{1,k} + u_{2,k} \cdot \frac{\Delta y_{dw}}{2} \geq a_{dw,\min} \quad (3.35f)$$

where  $F$  is the plant model and the caster wheel rolling speeds,  $\dot{\gamma}$  and  $\dot{\gamma}_{\text{ss}}$ , are defined by equations 3.10b and 3.15. States  $x_{6,k}$  and  $x_{7,k}$  are estimated from the observer proposed in Section 3.3.

$$\dot{\gamma}_k^l = \frac{1}{r_{\text{cw}}} \cdot [(x_{4,k} - x_{5,k} \cdot \Delta y_{\text{cw}}) \cdot \cos(x_{6,k}) + (x_{5,k} \cdot \Delta x_{\text{cw}}) \cdot \sin(x_{6,k})] \quad (3.35g)$$

$$\dot{\gamma}_k^r = \frac{1}{r_{cw}} \cdot [(x_{4,k} + x_{5,k} \cdot \Delta y_{cw}) \cdot \cos(x_{7,k}) + (x_{5,k} \cdot \Delta x_{cw}) \cdot \sin(x_{7,k})] \quad (3.35h)$$

$$\dot{\gamma}_{ss,k}^l = \frac{1}{r_{cw}} \cdot \sqrt{(x_{4,k} - x_{5,k} \cdot \Delta y_{cw})^2 + (x_{4,k} \cdot \Delta x_{cw})^2} \quad (3.35i)$$

$$\dot{\gamma}_{ss,k}^r = \frac{1}{r_{cw}} \cdot \sqrt{(x_{4,k} + x_{5,k} \cdot \Delta y_{cw})^2 + (x_{4,k} \cdot \Delta x_{cw})^2} \quad (3.35j)$$

## 3.7 Implementation of the MPC

The MPC has been developed in an open source tool for optimal control called CasADi [30]. It offers several methods for nonlinear optimization and algebraic differentiation. Its usage can be divided into integration of ODEs and finding a solution for the OCP.

### 3.7.1 Integration

The plant and observer models are integrated by using "Range Kutta" [31] with 1 finite element and a step size equal to the MPC's sampling time,  $t_s$ . Since the optimal inputs obtained from the MPC are accelerations, they need to be converted to velocities. This is done by applying Forward Euler's integration [32] at a frequency of 50Hz. This value has its roots in the Motor Control Unit's (MCU) characteristics.

### 3.7.2 OCP solver

The OCP is solved by applying the interior-point method [33]. To do so, a software package called "Ipopt" (Interior Point Optimizer) [34] capable of solving large scale nonlinear optimization problems has been used. The amount of tunable parameters that this solver includes allows the user to customize its performance. Finding the optimal combination of these parameters is out of the scope of these thesis. However, the OCP is the most critical part to ensure that the MPC runs at the desired frequency. That is why, an effort has been made to spot the settings that have the biggest impact on decreasing the solving time. According to user's guide [35], the linear solver's choice plays a crucial role. "MA27" [36] has ended up being the one with best performance. Using "warm start" [37] and tuning its respective setting have also turned out to be crucial.

## 3.8 Extension of PF

The equations obtained in the kinematics based analysis done in Section 3.1.2 can be applied to further develop the PF.

### Estimator

The PF's estimator can be fully replaced by the one developed in Section 3.3. In this way, two tuning parameters can be omitted, estimations do not exclusively rely any more on the equilibrium state, and transient scenarios are considered.

### Filtering ratio

Parameter  $\dot{\gamma}_{\max}$  in Equation 2.2b, can be replaced by  $Q_{\text{PF}} \cdot \dot{\gamma}_{\text{ss}}$ , where  $Q_{\text{PF}}$  is a weight that defines how much the PF should modify desired velocity commands and  $\dot{\gamma}_{\text{ss}}$  is the caster wheel's steady state rolling speed given for the desired velocity commands, as described in Equation 3.15. The purpose of this modification is that the filtering ratio,  $k$ , does not rely on the global maximum rolling speed, but adapts to the desired velocities. In this way, the tuning process for parameter  $Q_{\text{PF}}$  ( $\approx 1$ ) is more intuitive than for  $\dot{\gamma}_{\max}$ .

## 3.9 Summary

In this Chapter a caster wheel aware MPC has been formulated and the PF explained in the previous Chapter has been extended. The focus is put in the first case. As a starting point, kinematics of a differential drive and caster wheel's rotation angle have been combined to a plant model,  $F$ . Secondly, the cost function,  $J$ , that evaluates predictions according to navigation capabilities, caster wheel awareness and energy usage has been presented. For this purpose, it is necessary to derive an observer that estimates caster wheel rotation angles,  $\varphi$ , and rolling speeds,  $\dot{\gamma}$ . Moreover, a Lyapunov function has been found that proves that the steady state rotation angle,  $\varphi_{\text{ss}}$ , is an asymptotically stable equilibrium point. Subsequently, the horizon  $(N, T)$  and sampling time,  $t_s$ , have been determined by looking into the eigenvalues of the caster wheel angle's equations of motion. Putting all this information together results in an OCP that the MPC will implement for deriving optimal input commands. Having formulated the MPC, it is time to evaluate it by running simulations (Chapter 4) and experiments (Chapter 5).



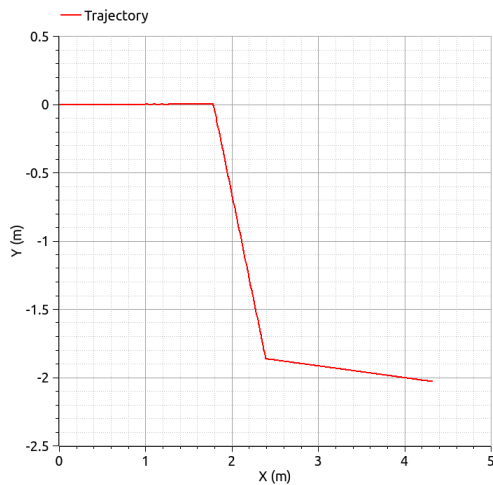
# Simulations

The objective of this Chapter is to explain results obtained from simulating the previously proposed caster wheel aware MPC based motion planner in a Modelica based AS's model. The evaluation takes place in two steps. Firstly, the observer's estimations are validated. Secondly, the performance of three different local planners are compared against each other. In this way, the caster wheel aware term's influence is studied.

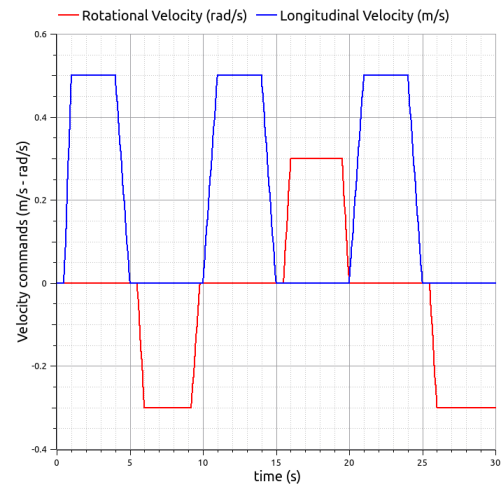
## 4.1 Observer

### 4.1.1 Case study I: Navigation across a global path

The caster wheel angles measured from the model have been compared to the estimated ones in the case study shown in Figure 4.1. The respective trajectory and velocity profiles can be visualized in Figures 4.1a and 4.1b.



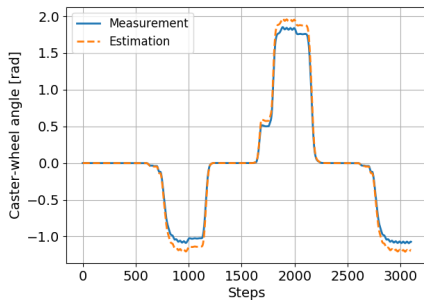
(a) Trajectory for estimator's analysis



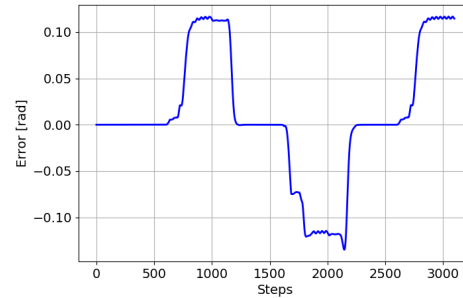
(b) Velocity commands for estimator's analysis

Figure 4.1: Case-study for caster-wheel angle estimator's analysis

This case-study involves three turns on the spot at three different points. The rotations are concatenated, which make the observer vulnerable to the accumulation of error. In order to test the estimator's performance in both directions, clockwise and counter-clockwise rotations are involved.



(a) Real and estimated front-left caster-wheel angle



(b) Error between real and estimated caster-wheel angles

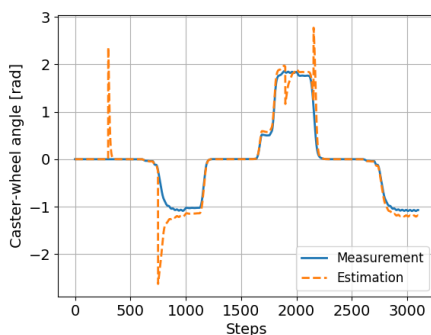
Figure 4.2: Estimation value and error of left-front caster-wheel angle

As it is depicted in Figure 4.2, apart from the steady-state error of 0.11 rad, the difference between measurements and estimations is negligible. In fact, the Root Mean Square Error (RMSE) between both data sets is 0.06988.

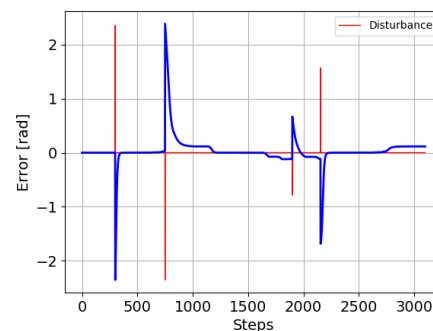
#### 4.1.1.1 Disturbance sensitivity

The estimator's behaviour under disturbances plays a key role when studying its convergence with respect to the caster wheel angle steady state value,  $\varphi_{ss}$ . That is why the effect of adding a disturbance to the estimated angle in the form of an impulse is observed. Since the resultant value is the input of the integrator in the following step, the respective estimation is disrupted. Once the impulse is over, the caster wheel angle converges back to the equilibrium point. In other words, the action of the impulse is equivalent to rotating the caster wheel externally by an angle and a duration equal to the impulse.

Figures 4.3 and 4.4 show the estimator's behaviour under disturbances. In order to consider the phenomena mentioned in the previous paragraph, disturbances of different lengths (0.01 s and 0.1 s) are going to be considered.



(a) Real and estimated front-left caster wheel angle under disturbances that are 0.01 s long



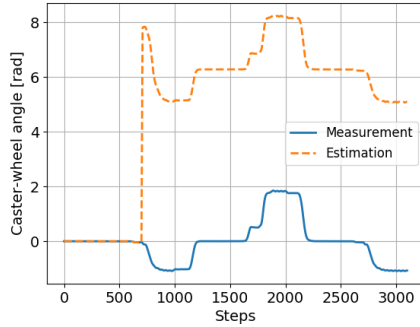
(b) Disturbances and errors caused in front-left caster-wheel angle

Figure 4.3: Estimation value and error of front-left caster-wheel under short disturbances

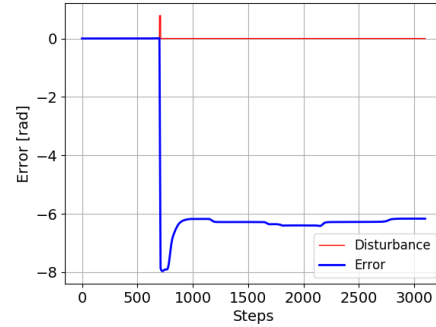


The disturbances shown in Figure 4.3 are impulses of a length of 0.01 s with different sizes ranging from  $-\pi$  to  $\pi$  rad, which take place in steady state and transient cases. As shown in this Figure, the estimation converges to the equilibrium angle regardless of the instant at which the disturbance occurs.

If the disturbance's amplitude is larger than  $\pi$  rad, the estimation converges to the closest value from the origin  $\varphi = 2n\pi$  with  $n = 0, \pm 1, \pm 2, \dots$ . Another way to replicate this scenario is to increase the disturbance's length.



(a) Real and estimated front-left caster wheel angle under disturbance that are 0.1 s long



(b) Disturbances and estimation errors in front-left caster-wheel angle

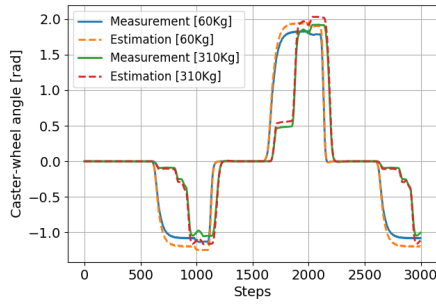
Figure 4.4: Estimation value and error of front-left caster-wheel under long disturbances

The disturbance given in Figure 4.4 is 0.1 s long, ten times longer than in Figure 4.3. Following the analogy mentioned above, this implies that the caster-wheel angle is externally rotated during a longer time. This generates an offset with respect to the origin, which, for the case shown in Figure 4.4, is equal to  $2\pi$  rad. As it was expected in the last paragraph, once the disturbance is over, the estimation converges to the closest value with respect to  $\varphi = 2n\pi$ .

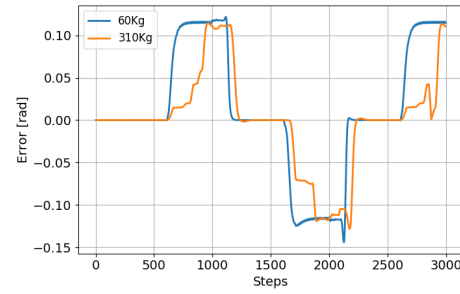
#### 4.1.1.2 Parameter variation

The estimator needs to be applicable to different robot setups, such as changes in geometry and mass. Considering that the previous analysis has only been applied to the cases with average load (150 Kg), it is worth looking into the applications with possibly maximum and minimal load mass (0 Kg and 250 Kg). The mass of the robot's main body (60 Kg) should be added to the load that it is carrying and thus, the robot's total mass can vary from 60 Kg to 310 Kg and the previous analysis has only considered a mass of 210 Kg.

Figure 4.5 shows that the behaviour of the caster wheel changes considerably depending on the load that the robot is carrying. The caster wheel rotation angle deviates from the origin much faster for the case of 60 Kg than for 250 Kg. This is also reflected on the error's graph in Figure 4.5b. The error is negligible, except for the moments when the caster-wheel angles deviate from the origin, which happens in a faster manner for lighter weights. However, the estimator adapts to this phenomena and its values are very close from the measurement. This demonstrates that the estimator's performance is not sensitive to the load that the robot is carrying.



(a) Real and estimated front-left caster-wheel for 60 Kg and 310 Kg



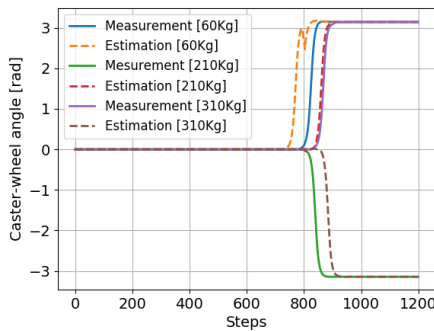
(b) Estimation errors in front-left caster-wheel angle for 60 Kg and 310 kg

Figure 4.5: Estimation value and error of front-left caster wheel for different loads

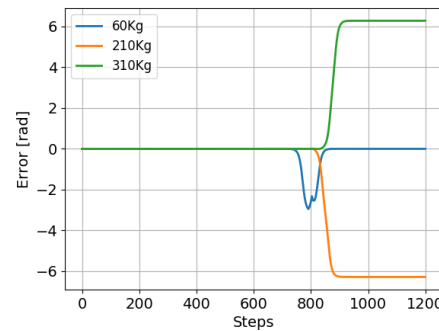
### 4.1.2 Case study II: Forward-Backward case

Since the estimator is based on the kinematics of the caster-wheel, the effects of dynamics are neglected. Considering the positive performance observed in the previous section, there is no need for such an extension. This simplification compromises the estimator's accuracy in some critical cases.

The "forward-backward" case would be such a case. It consists on a scenario where the robot is driving longitudinally ( $\omega = 0 \text{ rad/s}$ ,  $\alpha = 0 \text{ rad/s}^2$ ) and switches the direction by reversing. This implies that the caster-wheels will flip  $\pi \text{ rad}$ . Specifying the instant when this event is going to take place involves precise modelling of the dynamics not only from the robot, such as mass, inertia and caster-wheel friction parameters, but also environment, such as road's surface or wind. It also is highly sensitive to the integrator that is used to solve the ODEs.



(a) Real and estimated front-left caster-wheel for different loads in "forward-backward" case



(b) Estimation errors in front-left caster-wheel angle for different loads in "forward-backward" case

Figure 4.6: Estimation value and error of front-left caster-wheel for different loads in "forward-backward" case

This scenario has been replicated by running a simulation where the robot drives longitudinally with a speed of  $v = 0.5 \text{ m/s}$ ,  $\omega = 0 \text{ rad/s}$  during the first half and reverses direction by driving  $v = -0.5 \text{ m/s}$ ,  $\omega = 0 \text{ rad/s}$  in the second half. The results are shown in Figure 4.6b. The estimator detects this variation with a time difference, which varies depending on the load (530 ms for 60 kg, 270 ms for 210 kg and 170 ms for 310 kg) that it is carrying. These are considerably higher than the 20 ms transients analyzed previously. The direction of the estimation's rotation also varies depending on the load (it is correct only for the case of 60

kg). These facts confirm the sensitivity of this case to slight variations in the robot's parameters.

Since the AS is not allowed to drive backwards, this case is not applicable to this use case. Furthermore, extending the estimator, so that it would consider this particular scenario, would increase the complexity of the estimator's computational cost. Due to these reasons, finding a solution to this case is outside the scope of this thesis.

## 4.2 Comparison of MPC based local planners

### 4.2.1 Procedure

The MPC formulated in Section 3.6 is tested by running three sets of simulation. Each simulation compares three local planners: caster wheel agnostic, caster wheel agnostic with PF and caster wheel aware. A hypotheses is formulated at the beginning of each simulation set. The results are used to fulfill an analysis that leads to an evaluation of the hypotheses. The robot is replaced by a FMU exported out of the Modelica based AS's model developed in paper [5]. The FMU and the simulation framework are linked by the usage of a Python library called FMPy [38]. The MPC's implementation is explained in Section 3.7.

The first set of simulations consists on making the robot rotate on the spot. The case identification stated in Section 2.4, reveals the importance of this scenario when referring to reaction torques, and, consequently, motor torques. That is why, it is worth studying how torque term  $J_{cw}$  influences the behavior of the robot. The second set of simulations makes the robot follow a time based reference which navigates in the opposite direction of the robot's initial orientation. This case provides insightful information regarding its capability to cope with critical motions and, at the same time, follow a time based reference. The third simulation set consists on following two global paths that concatenate complex motions. Out of here it is expected to compare how all three local planners can handle the trade-off between chasing the moving reference and minimizing motor torques.

Before getting started clarifications in nomenclature should be made. "CWAGLP" stands for "Caster Wheel Agnostic MPC based Local Planner" and does not include the term that minimizes motor torques in the OCP's cost function. For this purpose, term "CWAWL", "Caster Wheel Aware MPC based Local Planner", has to be used. The last term, "CWPFLP", "Caster Wheel Agnostic MPC based Local Planner with Path Filter", consists on attaching the PF to "Agnostic"'s output. This notation is summarized in the following Equation.

$$\begin{aligned} \text{CWAGLP} &\rightarrow J = J_{\text{nav}} \\ \text{CWAWL} &\rightarrow J = J_{\text{nav}} + J_{\text{cw}} \\ \text{CWPFLP} &\rightarrow [J = J_{\text{nav}}] + \text{PF} \end{aligned} \tag{4.1}$$

### 4.2.2 Evaluation criteria

Before comparing the navigation capability of each planner, it is worth defining on which criteria are the results adopted to evaluate the simulation results.

### 4.2.2.1 Navigation

The local planner's navigation capabilities are going to be judged according to two statistical measurements: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) of the trajectory's deviation. Other parameters, such as distance covered and time spent during navigation, are also going to be considered.

#### Trajectory's deviation

The navigation's success depends on the similarity between the robot's trajectory and the global path. In order to quantify this, MAE and RMSE are going to be applied over the orthogonal distance,  $d_{error}$ , from every point of the robot's trajectory to its respective line in the global path.

Since the sampling of the robot's pose runs at 125 Hz,  $d_{error}$  is updated every 8 ms. Thus, the sum of all distances approximates the area between the global path and the robot's trajectory. This area is a measurement of how much has the robot deviated from the desired path. Figure 4.7 helps to visualize this measure.

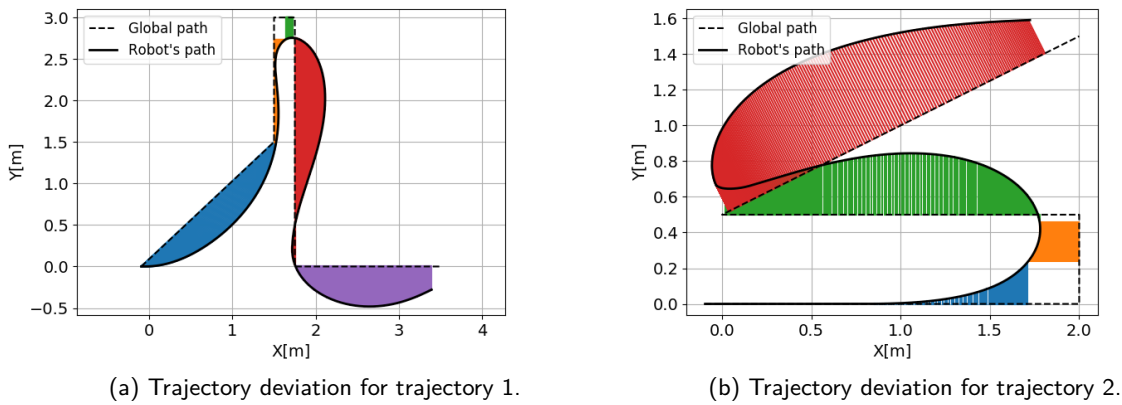


Figure 4.7: Trajectory deviations,  $d_{error}$ , applied to all the points obtained from the robots trajectory when implementing CWAWLPL to trajectories for "Navigation" simulation set.  $d_{error}$  has been colored depending on its respective line in the global trajectory.

#### MAE and RMSE

The areas colored in Figure 4.7 are a measure of the navigation's discrepancy with respect to the desired trajectory. Since the value of these areas consist of a set of distances ( $d_{error}$ ), they can be approximated by applying MAE and RMSE to the mentioned set.

Both terms provide information regarding the average distance error. On the one hand, RMSE is more appropriate for normally distributed errors, which means that it penalizes large deviations more severely. On the other hand, MAE provides more intuitive results, since it is the average of all distances [39].

The navigation's success rate depends on these two measurements. The lower their values, the higher the resemblance between the robot's trajectory and the global path. Thus, the planner with the lowest MAE and RMSE is the one with the best navigation capabilities.

### 4.2.2.2 Torques

Motor torques are evaluated based on relative percentage change of mean and maximum values. Left and right motors are analyzed separately for the rotation on spot case, while the average will be taken as a measure for the case studies related to navigation.

### 4.2.3 Case Study I: Rotation on the spot

The purpose of these simulations is to study the performance of the torque term  $J_{cw}$  by making the robot rotate on the spot. When doing so, two different scenarios have been considered for the initial angles of the caster-wheels  $\varphi_0$ . Each of the scenarios ("aligned" and "misaligned") is going to be simulated twice (with and without torque term  $J_{cw}$ ).

The first case is denoted as "aligned" and consists on starting the rotation with the caster wheels aligned with respect to the steady state value that they are going to achieve once the rotation takes place. From Equation 3.12 and considering that a rotation in the spot involves  $v = 0$  and  $\omega = \omega_{\text{rotation}}$ , the initial caster wheel angles can be defined as

$$\varphi_0^{\text{aligned}} = \varphi_{ss}^{\text{rotationSpot}} = \text{atan} \left( \frac{\Delta x_{cw}}{\Delta y_{cw}} \right) \quad (4.2)$$

When referring to the other case, term "misaligned" is used. Instead of having the caster wheels aligned with respect to the steady state values, their angle is equal to the robot's orientation.

$$\varphi_0^{\text{misaligned}} = 0 \quad (4.3)$$

Figure 4.8 depicts the difference between "aligned" and "misaligned" simulations. For the latter case there is a transition from the initial caster wheel angle to the desired one. This does not occur for the second case, because the caster-wheels initial angle matches the rotation's steady state value.

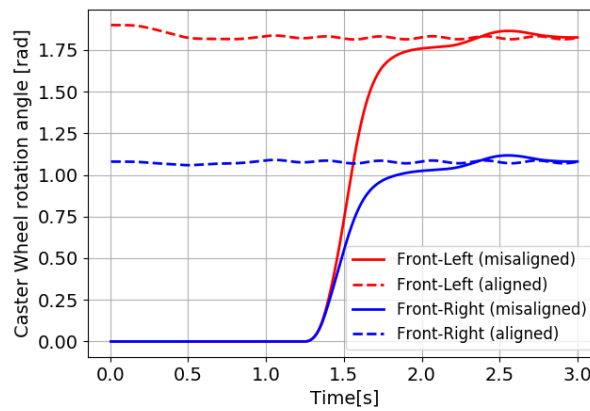


Figure 4.8: Comparison of front caster wheel angles,  $\varphi_{LF}$  and  $\varphi_{LB}$ , for "aligned" (discontinuous line) and "misaligned" (continuous) cases applied to a rotation on the spot with caster wheel agnostic MPC based local planner.

### 4.2.3.1 Caster wheel agnostic - CWAGLP

#### Hypotheses

"When implementing the MPC based local planner in CWAGLP to a rotation on the spot, the optimal velocity commands obtained from the OCP do not vary depending on the initial caster wheel angles  $\varphi_0$ . This implies that the motors have to provide an extra amount of torque to overcome the transition from  $\varphi_0$  to  $\varphi_{ss}$  that is required for the misaligned case."

#### Results

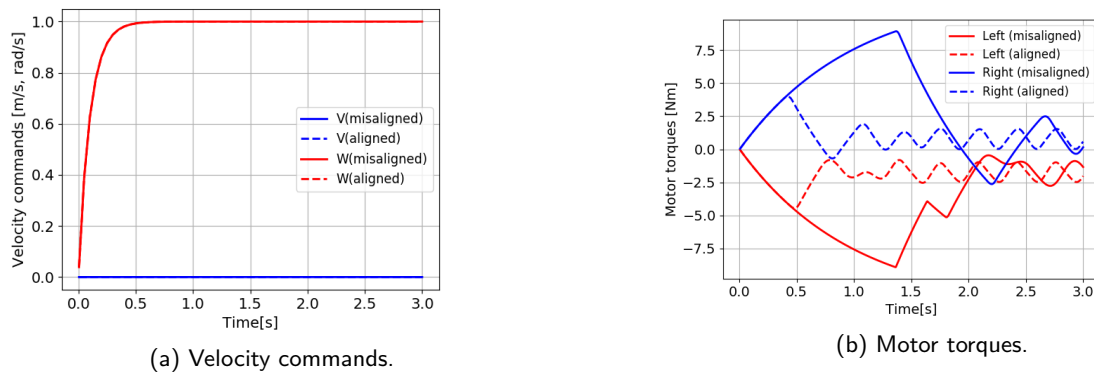


Figure 4.9: Performance when rotating on the spot with a caster wheel agnostic MPC local planner for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles  $\varphi$ .

Notice that the caster wheel angle's change for the front caster wheels is given in Figure 4.8.

#### Analysis

Figure 4.9a shows that the robot rotates on the spot ( $v = 0$  m/s and  $\omega = 1$  m/s) without varying its behaviour depending on the caster wheel's initial angle. As the hypotheses has predicted, this induces an increase in motor torques for the "misaligned" case. According to Figure 4.9b, the raise in the maximum motor torque can be quantified to an increase of 123.25% (from 4.0 Nm to 8.93 Nm). These peak values are related to the caster wheel angle transition shown in Figure 4.8. Once the motors overcome the necessary torque to align the caster wheel with respect to the steady state value, which happens approximately at  $t = 1.375$  s, the torques start decreasing.

### 4.2.3.2 Path Filter after caster wheels agnostic - CWPFLP

#### Hypotheses

"For CWPFLP the optimal commands obtained from the MPC are filtered according to the caster wheel's rotation angle  $\varphi$  and rolling speed  $\dot{\gamma}$ . Consequently, when applied to a rotation on the spot, the velocity commands that are fed to the robot differ depending on the initial caster wheel angles  $\varphi_0$ . On the one hand, if these are misaligned, it drives forward before it starts rotating. On the other hand, if they are aligned, the PF does not filter the commands given by the MPC."

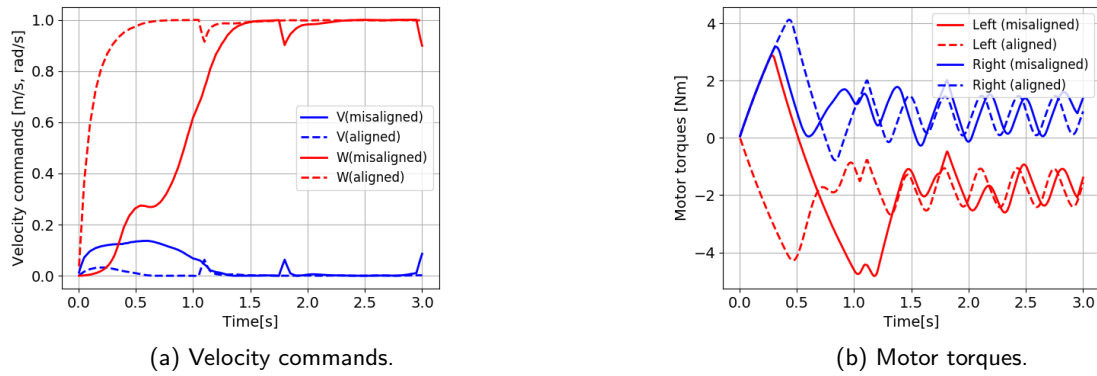


Figure 4.10: Performance when rotating on the spot with a caster wheel agnostic MPC local planner and a path filter for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles  $\varphi$ .

## Results

### Analysis

As the hypotheses has foreseen, the performance of the local planner differs depending on the initial caster wheel rotation angles. From the second term of Equation 2.2a,  $k \cdot (\varphi_{ss} - \hat{\varphi})$ , the PF relies on the difference between current and steady state caster wheel rotation angles,  $\varphi_{ss} - \hat{\varphi}$ , and the ratio between caster wheel's current and steady state rolling speeds,  $k = \frac{\dot{\gamma}}{Q_{PF} \cdot \dot{\gamma}_{ss}}$ . At the beginning of the rotation the robot is standing still, hence,  $\dot{\gamma} = 0$  and  $k = 0$ . As a consequence, once the caster wheel starts rolling, the filtering relies on  $\Delta\varphi$ , which is 0 for the aligned case and  $\varphi_{ss} - \hat{\varphi}$  for the misaligned one. This explains why driving forward is only necessary for the latter one and why the aligned one is a replica of the CWAGLP. This can be visualized in Figure 4.10a.

From Figure 4.10b it can be stated that the PF is capable of varying velocity commands, so that motor torques are minimized. For the misaligned case, the maximum value is 4.32 Nm, 48.4% of the CWAGLP's peak torque.

### 4.2.3.3 Caster Wheel aware

#### Hypotheses

*"When implementing CWAWLPL, the optimal inputs obtained from the OCP vary depending on the initial value of the caster wheel angles  $\varphi_0$ . On the one hand, for the "aligned" case, the robot rotates on the spot in a similar manner to the caster wheel agnostic case. On the other hand, if the caster wheels are misaligned, the robot is expected to drive forward, so that term  $J_{cw}$  is mitigated. This leads to a decrease in motor torques."*

## Results

### Analysis

The behaviour of the robot varies depending on the initial caster wheel angles. Only if these are not aligned with the respective steady state angles, the robot drives slightly forward. This can be seen in Figure 4.11a, where the longitudinal velocity command is 0.2 m/s. This results in a maximum torque value of 4.52 Nm, which can be quantified as a decrease of 49.4% with respect to the maximum motor torque obtained with CWAGLP. The fact that the longitudinal command for the misaligned case is negligible proves that this

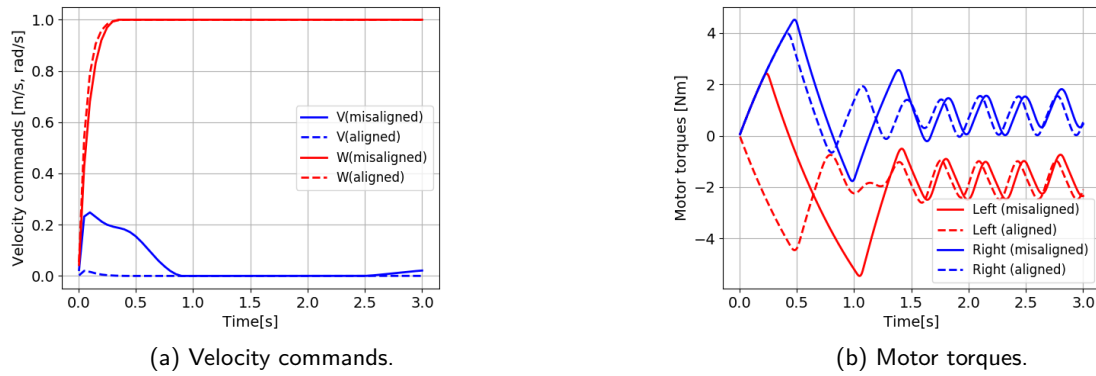


Figure 4.11: Performance when rotating on the spot with a caster wheel aware MPC local planner for aligned (discontinuous line) and misaligned (continuous line) initial caster wheel angles  $\varphi$ .

formulation is capable of mitigating torques depending on the caster wheel's state.

Minimizing torque term  $J_{cw}$  means that the difference between the caster wheel's real and desired rolling speed ( $\dot{\gamma} - \dot{\gamma}_{ss}$ ) is kept low. From Equation 2.1, it can be concluded that the bore torque that the motors need to overcome decreases. Comparing Figures 4.8 and 4.12b provides an alternative approach to understand the reason behind this phenomena. Firstly, the transition of  $\varphi$  starts 1.1 s earlier and lasts 0.4 s longer for CWAFLP. Since driving forward forces the caster wheels to roll, the bore torques that the motors need to overcome is decreased. Consequently, the lowered torque limit is achieved earlier and the caster wheels are capable to start rotating towards a steady state angle.

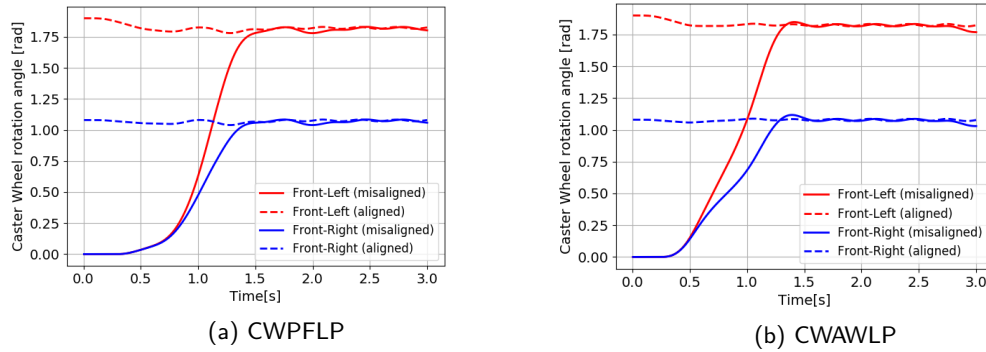


Figure 4.12: Comparison of front caster wheel angles,  $\varphi_{LF}$  and  $\varphi_{LR}$  when rotating on the spot with different local planners

#### 4.2.4 Case Study II: Rotate and navigate in a straight line

These simulations replicate a scenario where the robot's starting orientation is opposite to the navigation's one. Consequently, the robot needs to turn  $180^\circ$  before navigating in a straight line. Analyzing this use case provides information about the performance of each planner to juggle rotations on the spot and navigation capability.



#### 4.2.4.1 Hypotheses

"The execution of the rotation on the spot that takes place before starting to navigate depends on each planner. Similar behaviour to the one observed in subsection 4.2.3 is expected. Thus, CWAFLP and CWPFLP result in a longitudinal velocity command that causes a longitudinal motion prior to the rotation. Consequently, the robot does not turn while standing still, which leads to a significant decrease in motor torques. Since the time based reference keeps navigating while the robot is rotating, the curves in the trajectories of CWPFLP and CWAFLP might have a bigger radius than the CWAGLP's one. In other words, at the expense of minimizing torques, the length of the trajectory increases.

#### 4.2.4.2 Results

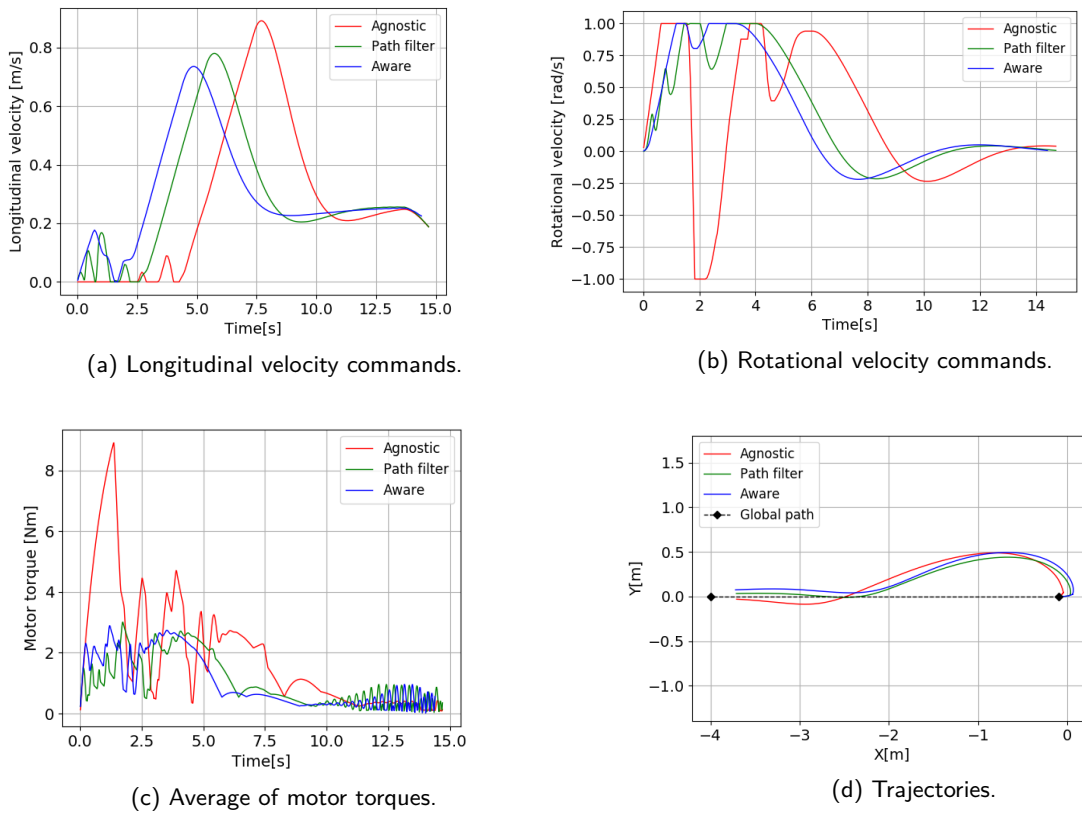


Figure 4.13: Performance with CWAGLP (red), CWPFLP (green), CWAFLP (blue) for rotate and navigate case study.

Before getting into the details of the results, some clarifications regarding this case study should be made by referring to Figure 4.13d. The robot starts in position  $[x, y] = [-0.095, 0]$  facing the X-axis' positive direction. However, the time based reference navigates from  $-0.095$  m to  $-4$  m, negative direction of X-axis, with  $v = 0.5$  m/s along the "global path" (black dashed line in Figure 4.13d).

Even if all three trajectories given in Figure 4.13d are close from each other, there are two differences that need to be highlighted. Firstly, at the beginning of the navigation, CWPFLP (green) and CWAFLP (blue) drive longitudinally from  $-0.095$  m to  $0.04$  m. This does not occur for CWAGLP (red). Secondly, in the case of CWAGLP, there is a small overshoot at  $3$  m, when the robot gets back to the global path.

All these motions can also be observed in the velocity profile given in Figures 4.13a and 4.13b. The longitudinal velocity command that takes place from  $t = 0$  s to  $t = 2$  s for CWPFLP and CWAFLP explains the forward motion at the beginning of the navigation.

Figure 4.13c shows the average of both motor torques required to fulfill these trajectories. CWAGLP's motions result in a peak torque at the initial stage of the navigation, which does not happen for CWPFLP and CWAFLP. Notice that the oscillations that appear after  $t = 11$  s are caused by the time based reference's compression once it gets to the end point.

#### 4.2.4.3 Analysis

##### Navigation

Table 4.1: Measurements to evaluate navigation's performance when simulating CWAGLP (agnostic), CWPFLP (path filter) and CWAFLP (aware) for "rotate and navigate" case study

Local planner	Distance [m]	Time [s]	MAE [m]	RMSE [m]
Agnostic	4.05	14.7	0.1154	0.1844
Path filter	4.16	14.7	0.09	0.1646
Aware	4.26	14.4	0.1378	0.1989

From the first column in Figure 4.1, it can be stated that the hypotheses has correctly foreseen that CWAGLP would cover a shorter distance than CWPFLP and CWAFLP. Regarding MAE and RMSE columns, the deviation with respect to the global path is very similar for all three planners. Even if CWPFLP navigates closest to the global path (lowest MAE and RMSE values), the maximum relative difference with respect to the other planners is only 4.78 cm.

##### Torques

Table 4.2: Measurements to evaluate motor torques usage when simulating CWAGLP (agnostic), CWPFLP (path filter), CWAFLP (aware) for navigation after rotate and navigate case-study.

Local planner	$T_{L,max}$ [Nm]	$T_{L,mean}$ [Nm]	$T_{R,max}$ [Nm]	$T_{R,mean}$ [Nm]
Agnostic	8.89	1.59	8.94	2.23
Path filter	4.85	0.71	4.46	1.39
Aware	4.81	0.76	4.48	1.31

The content shown in Figure 4.13c is covered in detail in Figure 4.2. As the hypotheses has predicted CWPFLP and CWAFLP have reduced the peak torques that the CWAGLP's rotation on the spot implies. This change can be quantified as a decrease of 54% in both motors. Moreover, the average torques have also been lowered to 52% for the left motor and 62% for the right one.

#### 4.2.4.4 Conclusions

This case study has analyzed an scenario where the robot needs to rotate before navigating in a straight line. The results provide insightful information regarding each planner's navigation capabilities and torque requirements. The behaviour observed in Section 4.2.3 has been replicated and confirms that CWPFLP and CWAFLP avoid rotating on the spot, which decreases motor torques considerably. When doing so, the navigation's quality is not sacrificed, since all three trajectories are very close from each other. Consequently,

this Section has proved that CWPFLP and CWAFLP are capable of mitigating torques without requiring meaningful variations in the robot's trajectories.

### 4.2.5 Case Study III: Navigation across given global paths

The purpose of these simulations is to extend the analysis made in Section 4.2.4 by navigating through more complex paths. This will lead to the evaluation of each motion planner's performance when following a specific path and, at the same time, its capability to mitigate motor torques. To do so, two different trajectories are considered.

In Sections 4.2.3 and 4.2.4 the performance of MPC formulations when rotating on the spot has been analyzed. It has been proven that CWAFLP and CWPFLP can minimize motor torques when rotating on the spot according to caster wheel states. These simulations have validated the torque term  $J_{cw}$  of CWAFLP and the PF included in CWPFLP. However, their capability to navigate is still unknown.

The purpose of the following simulations is to study how CWAFLP and CWPFLP perform when navigating along two given trajectories, whose main attributes are short distances between turning points and curves featured by sharp angles. Consequently, torques above the nominal range are required to complete these trajectories.

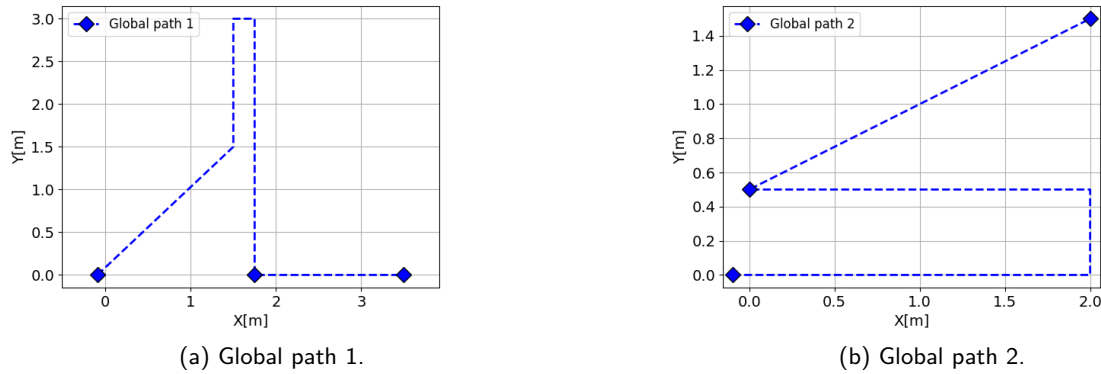


Figure 4.14: Global paths for analyzing MPC based local planners performance when navigating. They are characterized for making the robot turn on the spot and being torque demanding.

#### 4.2.5.1 Hypotheses

*"Considering that CWAFLP and CWPFLP try to minimize motor torques, these planners tend to increase the radius of the curves. Therefore, CWAGLP is the formulation that keeps the closest distance with respect to the global path. Softening the trajectory by taking longer curves leads CWAFLP and CWPFLP to have a lower average motor torque than CWAGLP. Taking into account that the CWPFLP's PF is not included in the OCP, but attached to its output, CWAFLP handles better than CWPFLP the trade-off between navigation and torque mitigation."*

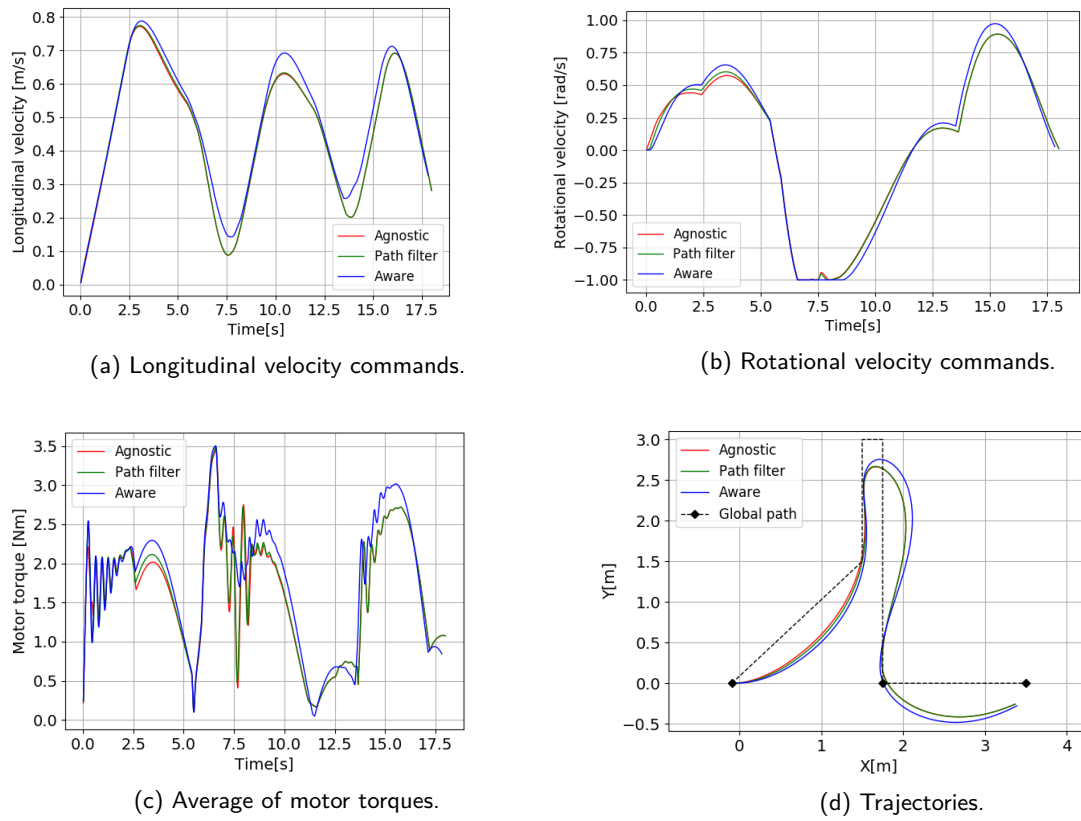


Figure 4.15: Performance when navigation through global path 1 with CWAGLP (red), CWPFLP (green), CWAWLP (blue)

#### 4.2.5.2 Results

##### Global path 1

Figures in 4.15 show that all three planners behave in a very similar manner. No meaningful differences can be identified between CWAGLP and CWPFLP. Nevertheless, in Figure 4.15d it can be seen that CWAWLP slightly increases the radius of the curves. This is related to higher longitudinal velocity commands shown in Figure 4.15a. The average motor torques are also alike for all three planners, but there are some small differences for the case of CWAWLP in  $t = 7.5$  s and  $t = 15$  s. In the first case oscillations are reduced, while in the second there is a small increase in the average torque value.

##### Global path 2

From Figure 4.16d it can be seen that CWAGLP covers the shortest distance, while CWPFLP's and CWAWLP's trajectories are very close from each other. Regarding motor torques, even if all three planners seem to follow the same pattern, CWAWLP stands out for getting rid of the spike in  $t = 9$  s and the oscillations around  $t = 10$  s.

#### 4.2.5.3 Analysis

In order to study the navigation's quality and the respective motor torques separately, the analysis is divided into two groups. This is going to be based on different measures explained in subsection 4.2.2 and references to Figures 4.15 and 4.16.

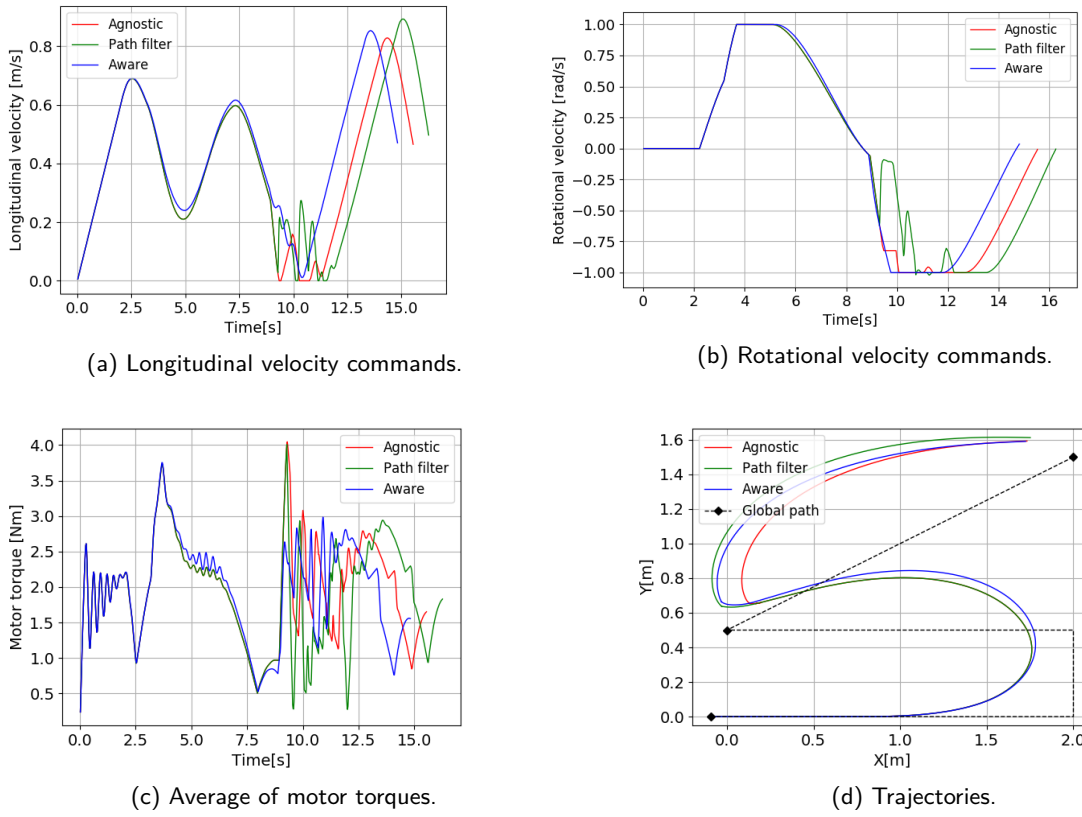


Figure 4.16: Performance when navigation through global path 2 with CWAGLP (red), CWPFLP (green), CWAJLP (blue).

### Navigation

The results shown in Figures 4.15d and 4.16d have been summarized in Table 4.3.

Table 4.3: Measurements to evaluate navigation's performance when simulating CWAGLP (agnostic), CWPFLP (path filter) and CWAJLP (aware) for two different trajectories.

Trajectory	Local planner	Distance [m]	Time [s]	MAE [m]	RMSE [m]
1	Agnostic	8.13	18.0	0.1561	0.2035
	Path filter	8.17	18.0	0.1602	0.2081
	Aware	8.57	17.82	0.1881	0.2400
2	Agnostic	6.06	15.54	0.1846	0.2364
	Path filter	6.39	16.26	0.2061	0.2690
	Aware	6.34	14.82	0.2097	0.2674

Table 4.3 reveals that in both case studies CWAGLP is the planner which covers the shortest distance. Regarding navigation time, CWAJLP is the fastest to the end of both trajectories. When it comes to the trajectory's closeness with respect to the time based reference's global path, CWAGLP performs best (smallest MAE and RMSE values), while CWAJLP and CWPFLP tend to deviate more.

### Torques

The torques of left and right motors have been measured separately. The respective information its summarized below.

Table 4.4: Measurements to evaluate motor torques usage when simulating CWAGLP (agnostic), CWPFLP (path filter), CWAUWLP (aware) for two different trajectories.

Trajectory	Local planner	$T_{L,max}[\text{Nm}]$	$T_{L,mean}[\text{Nm}]$	$T_{R,max}[\text{Nm}]$	$T_{R,mean}[\text{Nm}]$
1	Agnostic	3.98	1.35	5.32	1.88
	Path filter	4.01	1.36	5.35	1.9
	Aware	4.27	1.52	4.96	2.0
2	Agnostic	5.2	2.15	5.38	1.78
	Path filter	5.2	2.12	5.09	1.7
	Aware	5.08	2.11	5.12	1.84

Table 4.4 and Figures 4.15c, 4.16c show that all three planners apply very similar torques when navigating through both trajectories. Considering the inconsistency of the torques obtained for each global path and the similarity between different planners, no clear trend can be observed. The cause of this phenomena is explained in the following subsection.

#### 4.2.5.4 Conclusions

The results and analysis given during Subsection 4.2.5, combined with the ones done in 4.2.3 and 4.2.4 provide enough information to observe the strong and weak points of each planner in a simulation framework.

The navigation's analysis has confirmed that CWAGLP is the planner that sticks best to the global path. However, the difference with respect to CWAUWLP and CWPFLP is negligible if the AS's dimensions and the deviation's mean values are compared. Furthermore, its relative deviation is persistent for both case studies.

From subsections 4.2.3 and 4.2.4, it was expected that CWPFLP and CWAUWLP would decrease the torques obtained in CWAGLP. However, out of the torque's analysis, all three planners apply very similar values. The command velocities in Figures 4.15 and 4.16 are helpful to explain this phenomena. In these figures, it can be perceived that CWAGLP's motion profile is close to the optimal one. That is why, CWPFLP and CWAUWLP are not capable of making significant reductions. Since CWAGLP's tuning ensures that the time based reference is chased smoothly, the robot will never be submitted to abrupt accelerations which imply critical movements that could cause undesired motor torques. Thus, applying a time based reference to complex global paths hinders visualizing the impact of the PF and caster wheel aware term.

Nevertheless, Figures 4.15c and 4.16c demonstrate that CWAUWLP is capable of removing torque oscillations that appear in CWPFLP and CWAGLP. On top of that, CWAUWLP also decreases the time required for getting to the end point.

Looking into the cost of the OCP provides an alternative approach for understanding the impact of the caster wheel aware term. Adding a quadratic term to the objective function, should increase its value. However, its physical implications in the AS's behaviour causes the opposite if the caster wheels are in a blocking scenario, such as the rotation on the spot that takes place from  $t = 8$  s to  $t = 12.5$  s in Figure 4.17.

Putting all these arguments together, it can be stated that CWPFLP and CWAUWLP are capable of mitigating torques, without sacrificing the navigation's quality. On top of that, CWAUWLP requires less time to reach the goal point. Finally, it has been learnt that a time based reference navigation policy is not a suitable approach to compare torque mitigation planners in complex global paths.

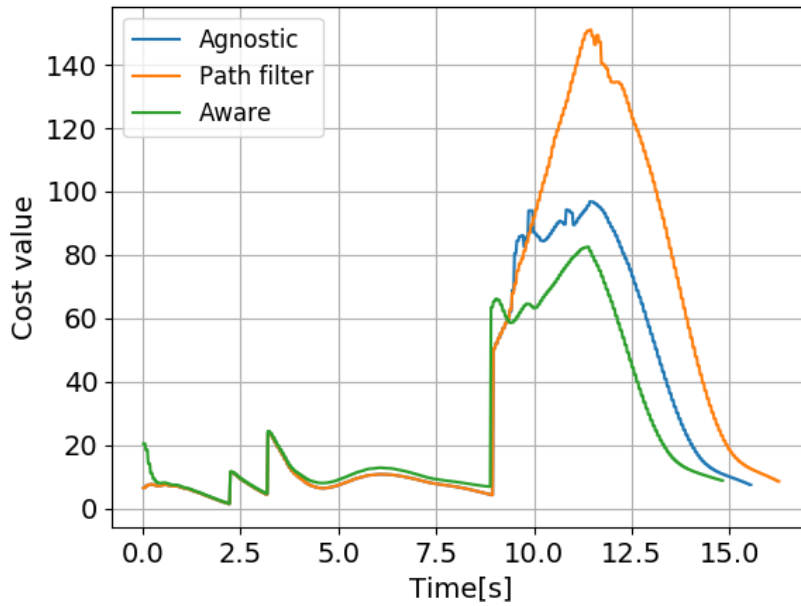


Figure 4.17: Comparison of OCP's cost value for global path 2 for differen planners.

Consequently, the navigation capabilities presented in this section combined with the capacity to deal with critical motions in Subsections 4.2.3 and 4.2.4 prove that the caster wheel aware term formulated in Section 3.6 and PF extended in Section 3.8 are tools that can be implemented to minimize motor torques in DDMR with caster wheels.

### 4.3 Summary

In this Chapter the observer and the MPC formulation have been tested by running simulations. In the first half, measured and estimated caster wheel rotation angles have been compared. When doing so, the observer's convergence with respect to steady state values has been demonstrated by adding disturbances and varying the load. In the second half, the MPC formulated in Section 3.6 has been evaluated. For this purpose, three simulation sets have been carried out. It has been shown that adding the caster wheel aware term,  $J_{cw}$ , to the cost function causes a similar behaviour to attaching the PF to the output of the agnostic planner. For both cases, mean and maximum torques decrease without degrading the navigation's quality. It has also been learnt that following a time based reference biases the mitigation of reaction torques. In the upcoming chapter this validation is extended from a virtual framework to an experimental setup, where the proposed motion planners are integrated in the AS.





# Field Test

---

The MPC based local planners formulated in Section 3.6 combined with the PF's extension in subsection 3.8, have led to a comparison of three different local planners in a virtual framework. Given that some important indexes for real application cannot be validated just by simulations, two case studies are conducted in an experimental setup. To this end, this chapter covers the implementation of the MPC in the AS and the analysis of the results obtained in each case.

## 5.1 Definition of case studies

Two different sets of experiments are conducted. The first one consists on a rotation on the spot, where the PF's and CWAFLP's performances are studied. Secondly, a modified version of the global path presented in Subsection 4.2.4 is applied for analyzing torque mitigation when following a time based reference with CWAGLP, CWPFLP and CWAFLP. Moreover, estimated and measured caster wheel rotation angles are compared.

According to the hypotheses stated in Section 2.4, torques out of nominal range are required when rotating on the spot. As a first experimental step, this hypotheses, along with the MPC and PF solutions developed in Sections 3.6 and 3.8 are tested.

Simulations in the previous Chapter have shown that following a time based reference with a properly tuned MPC has an influence on the applied motor torques and hinders the impact of the caster wheel aware term. Hence, looking into long global paths does not provide any valuable information regarding motor torques. Therefore, the second set of tests consists on an extended version of the case study in Subsection 4.2.4, while the case study in Subsection 4.2.5 is discarded.

When doing so, the caster wheel state observer has been validated by comparing measured and estimated rotations angles. For this purpose, a sensor-system to measure caster wheel rotation angles has been designed and assembled for both wheels at the rear axle.

## 5.2 Experimental setup

### 5.2.1 Layout

Converting simulations running in a virtual framework into a field test requires a neutral environment that will not interfere in the AS's outcome. Furthermore, there is a need for an infrastructure capable of measuring the parameters that have been considered to be known during simulations. The experiments were executed in a test field that fulfilled all these conditions. An overview of the setup is given in Figure 5.1.

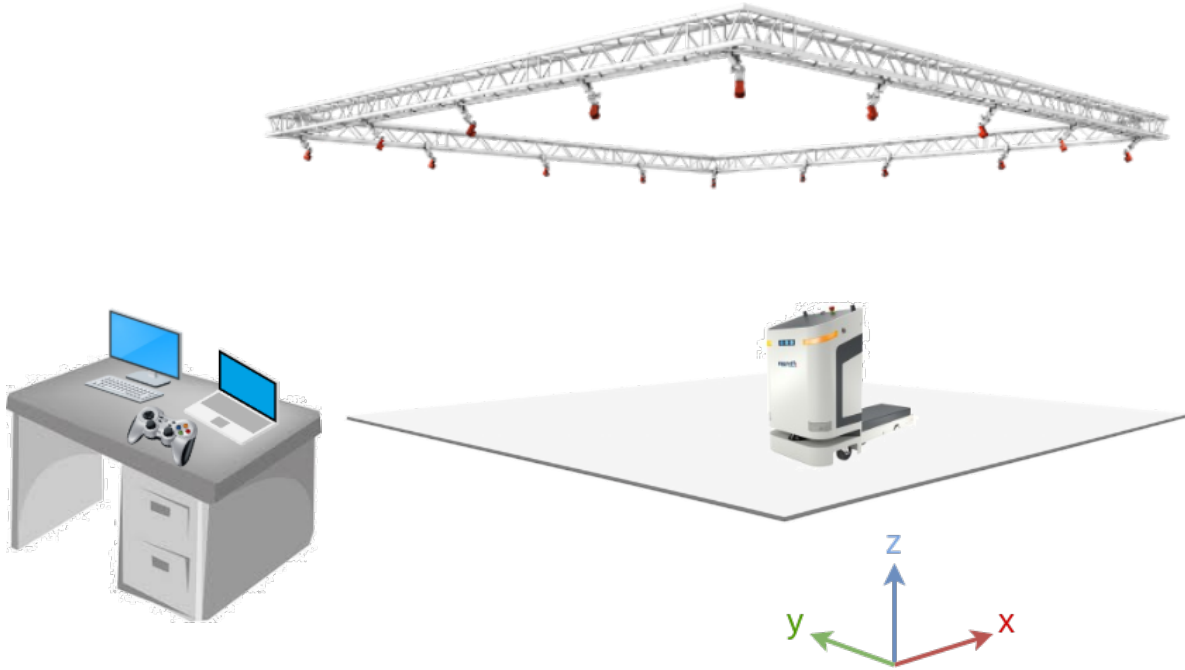


Figure 5.1: Experimental setup for performing case studies on hardware.

The testing site consists of an obstacle-free flat surface located below a set of cameras that track the robot's position. This vision system is called "Opti-Track" [40] and is connected to a desktop machine denoted as "Vision Unit". The robot's location is sent from the "Vision Unit" to the "Navigation Unit" (Lenovo Thinkpad T470 with a processor called Intel®Core i5-7300U operating at 2.5 GHz and 8 GB of RAM), which runs the navigation algorithm that controls the robot's motions by sending velocity commands to the robot. These are converted to motor commands by the driver running inside the RCU.

All three computers are connected through a ROS network, whose master is the RCU and can be accessed from the "Navigation Unit" through Secure Shell protocol (SSH). Details on the ROS network are given in Section 5.2.2.

The navigation algorithm is listening to a joystick ("Logitech F710"), which can either be used as a remote control to drive around, or can activate the "autonomous mode", so that the local planner follows a specified global path. The PF can also be activated/deactivated from the remote control. The algorithm's working principle is covered in 5.2.3.

### 5.2.2 Network

The experimental framework explained in 5.2.1 involves three different computers which need to operate under a network capable of exchanging information in a synchronized manner. This was achieved by applying the ROS network shown in Figure 5.2.

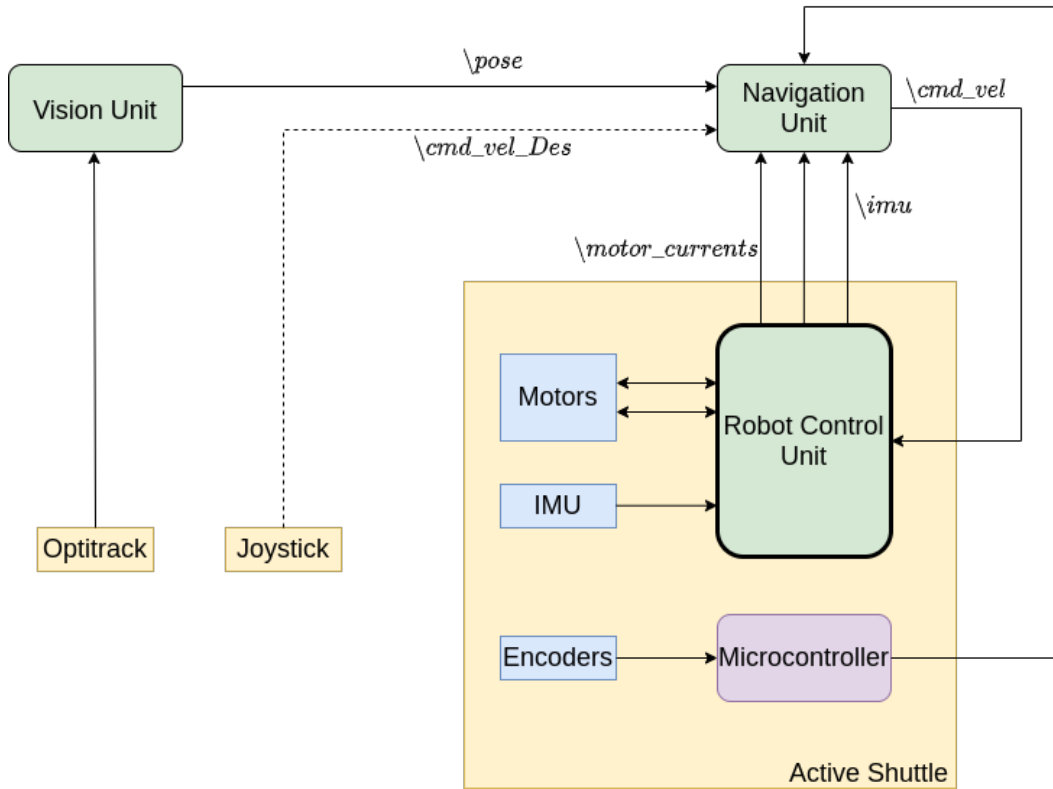


Figure 5.2: Network of experimental setup for performing case studies on hardware.

The components can be classified according to the color that they have been assigned. Hardware devices, such as the AS, joystick and opti-track cameras, can be visualized in yellow. The three computers are shown in green. Since the RCU is the master of the network, its outline is thicker. Relevant components inside the robot are shown in blue and the micro-controller in charge of publishing sensor values is purple. Arrows represent the transfer of messages between two devices and the names refer to their respective ROS-topic. The machine that is linked to the arrow's origin contains a ROS-publisher, while the one attached to its end has a subscriber.

The network's master is the RCU and runs a script that connects the robot's hardware with the ROS network. The velocity commands received from the "Navigation Unit" are stored and converted to motor commands, which are sent with a frequency of 50 Hz. As soon as these are received by the motors, the RCU publishes a feedback message containing, motor currents, IMU and odometry data to the ROS network.

The "Navigation Unit" runs the navigation algorithm, which can drive the robot in manual or autonomous mode. It is subscribed to the feedback information provided by the motors and the robot's location coming from the "Vision Unit". If the manual mode is activated, it also listens to the joystick's velocity commands. Apart from that, for analysis purposes, it logs sensor values published by the micro-controller.

### 5.2.3 Navigation algorithm

The navigation algorithm is subscribed to different buttons on the joystick. This way, not only the robot can operate in a manual or autonomous mode, but the PF can also be activated/deactivated by pushing a single button.

Once the autonomous mode is activated, the robot will drive around a previously defined global path. This is done by following the logic exposed in the pseudo code shown below.

---

#### MPC local planner

---

##### Initialization:

```
mpc = MPC_local_planner
navigation = navigation_local_planner
observer = caster_state_observer
pf = caster_path_filter
HLC.start [20Hz]
LLC.start [200Hz]
```

##### HLC(*x*):

```
while True do
  ref = navigation.update(x, time)
  a,  $\alpha$  = mpc.NLPsolver(x, ref)
end while
```

##### LLC(*a*, $\alpha$ ):

```
while True do
  if navigation.check = finished then
    HLC.stop
    break
  end if
  x = UpdateState(pose, odometry, est)
  v,  $\omega$  = ForwardEuler(a,  $\alpha$ )
  est = observer.estimate(odometry, v,  $\omega$ )
  if pf.on = True then
    v,  $\omega$  = pf.filter(est)
  end if
  Publish(v,  $\omega$ , est)
end while
manual_mode.activate
```

---

The local planner (App. B.1) is divided into four classes: the MPC (App. B.2), the time based reference (App. B.3), the observer (App. B.4) and the path filter (App. B.5). In this way, the functions required to follow the global path are implemented in a modular fashion, which eases the correction of bugs and facilitates the formulation of new algorithms. Notice that all the scripts have been added to App. B.

These classes are applied in two independent loops that run at different frequencies. Since the MPC's horizon choice in Section 3.4 implies a rate of 20 Hz and the computation time required for solving the NLP varies between 20 ms and 50 ms, parallelization is required. This is done by applying a Python package called "Multiprocessing". Its working principle is similar to the one of a thread, but avoids the "Global Interpreter Lock" by using subprocesses instead of threads, which results in an increase on speed [41].

Once the three classes are initialized, a process, named "High Level Control" (HLC), that runs the MPC at the desired frequency is started. It takes the robot's most recent state,  $\mathbf{x}$ , as input and returns the optimal linear and angular accelerations ( $a$  and  $\alpha$ ).

Simultaneously, another loop, which represents the "Low Level Control" is running at a rate of 200Hz. This loop starts by verifying if the robot has reached its destination or the autonomous mode has been deactivated. If none of this is true, it updates the state vector. To do so, it replaces the position and velocity states with the member variables stored in the last calls of the subscriber-callback functions for topics `\pose` and `\odometry`. The estimations are updated by applying the ones obtained in the previous iteration.

Notice that LLC's inputs are accelerations ( $a$  and  $\alpha$ ), obtained from the HLC's process. Since the RCU takes velocity commands ( $v$  and  $\omega$ ) as inputs, Forward Euler integration is applied to the accelerations. If the path filter is activated, the velocity commands are filtered out depending on the estimated caster wheel state, *est*. Finally, the resultant velocity commands are published to the ROS network.

## 5.2.4 Implementation

### 5.2.4.1 Tracking system

The pose of the AS is obtained through a motion tracking system [42]. A set of cameras, which lay above the testing area, capture the light that is reflected from four markers attached to the robot's upper surface (see Figure 5.3). "Motive" is a software running in the "Vision Unit" and converts the data sampled by the cameras into coordinates. Its broadcasting options allow to publish the robot's pose to the ROS network.

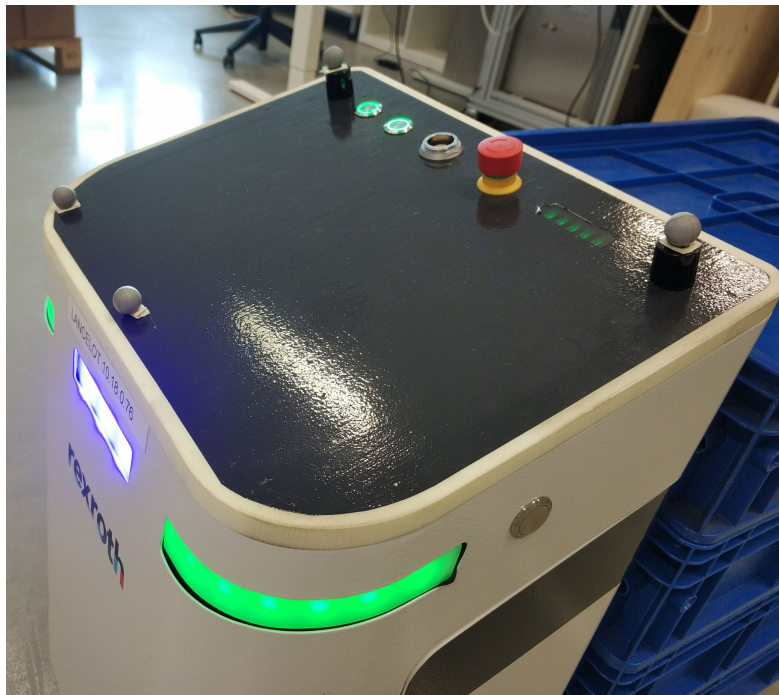


Figure 5.3: Reflective markers attached to the robot's upper surface for tracking its pose.

When calibration takes place, a "Calibration Result Report" is provided by the software. According to this, the "Overall Reprojection 3D Error" is 0.478mm and the mean triangulation error is 0.6mm (the recommended maximum is 2.8mm). The software classifies this calibration as "Exceptional", the highest quality among the possible ranks [40].

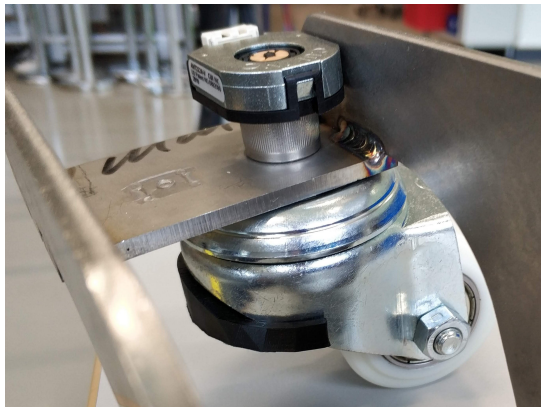
### 5.2.4.2 Active Shuttle

As it was mentioned in Section 1.5, apart from the RCU, inside the robot a SCU is also running. It ensures that safety requirements, such as minimum distance to surrounding obstacles and synchronization between motions and blinkers, are always satisfied. Since the robot is not allowed to operate above certain combinations of longitudinal and angular velocities, new linear restrictions need to be included into the OCP formulated in Section 3.6. Considering that these provide confidential information regarding hardware characteristics, they are not documented in this thesis.

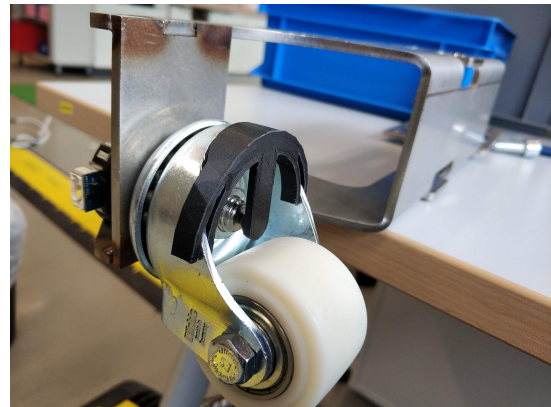
### 5.2.4.3 Sensors

Since CWPFLP and CWAFLP modify the robot's trajectory depending on the states of the caster wheels, these planners are considered to be caster wheel aware approaches. Both rely on the kinematic model of the observer which estimates the caster wheel's rotation angle,  $\varphi$ , and rolling speed,  $\dot{\gamma}$ . That is why, comparing estimated and real caster wheel states provides meaningful information to validate, not only the observer formulated in Section 3.3, but also the "caster wheel aware" term,  $J_{cw}$ , included into the OCP's cost function in Section 3.6.

In order to measure the rotating angle of the caster wheels, encoders ("AMT22" - CUI devices [43]) have been placed at the rear axle. The rotational movement of the caster wheel is transferred to the encoder through a 3D printed shaft that is attached to the caster wheel's frame. This design requires replacing the caster wheel assembly's bolts with screws that have holes through them. Both encoders are connected to a micro-controller (Teensy 3.6 [44]), which is plugged to the RCU through USB. In this way, the micro-controller reads the values of both encoders and publishes them to the ROS network. Notice that the encoders are powered from the USB's 5V line. An overview of the assembly is given in Figure 5.4. More details about this measurement system are given in Figure A.3 in the appendix.



(a) Isometric view.



(b) Bottom view.

Figure 5.4: Assembly of measurement system for caster wheel rotation angle,  $\varphi$ , at AS's rear axle.

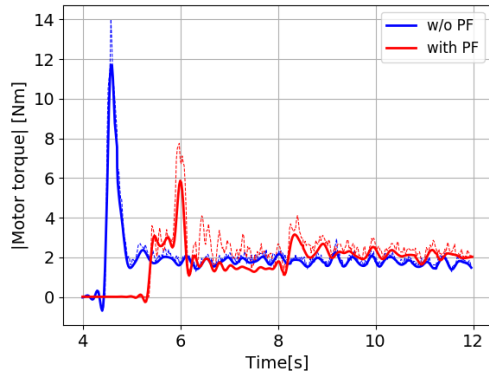
## 5.3 Case study I: Rotation on spot

The case identification in Section 2.4 and results obtained in Subsection 4.2.3 reveal the importance of rotations on the spot when mitigating motor torques in a DDMR with caster wheels. That is why, the

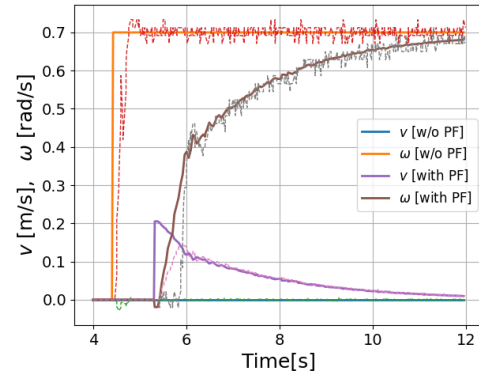
behaviour of the PF's extension proposed in Section 3.8 and CWAWLP formulated in Section 3.6 have been observed in the experimental setup presented in Section 5.2.

### 5.3.1 Path Filter

In order to examine the PF when rotating on the spot, a command of  $[v, \omega] = [0, 0.7]$  is published to topic `\cmd_vel_Des`. If the PF is activated, this command is filtered depending on caster wheel estimations and published to topic `\cmd_vel`, so that a subscriber callback in the RCU converts it to motor references.



(a) Average (continuous) and maximum (dashed) motor torques.



(b) Command (continuous) and odometry (dashed) velocities.

Figure 5.5: Comparison of velocity commands and torques in AS when rotating on the spot with and without PF.

Figure 5.5 shows very similar results to the ones obtained in 4.2.3, where the initial torque spike is reduced by driving forward instead of rotating on the spot. In Figure 5.5a it can be visualized that the average and maximum motor torques are reduced to the half. This can be understood by looking into the motion profile given in Figure 5.5b. When the PF is activated, the robot moves forward (purple) and rotates (brown) simultaneously, instead of turning (orange) while standing still (blue). Notice that the dashed thinner lines represent the robot's real velocity given by the motor's odometry feedback.

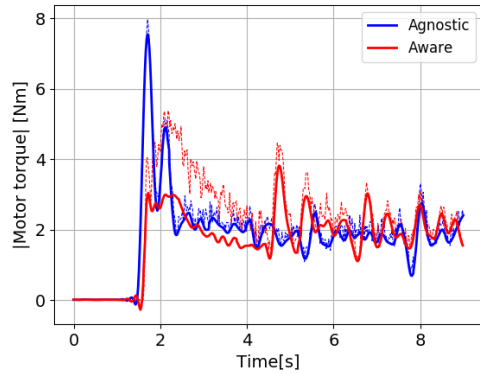
Table 5.1: Measurements to evaluate motor torques usage when rotating on the spot with and without PF in the Active Shuttle.

Local planner	$T_{L,max}$ [Nm]	$T_{L,mean}$ [Nm]	$T_{R,max}$ [Nm]	$T_{R,mean}$ [Nm]
without PF	11.16	2.03	13.99	1.84
with PF	4.4	1.61	7.75	1.87

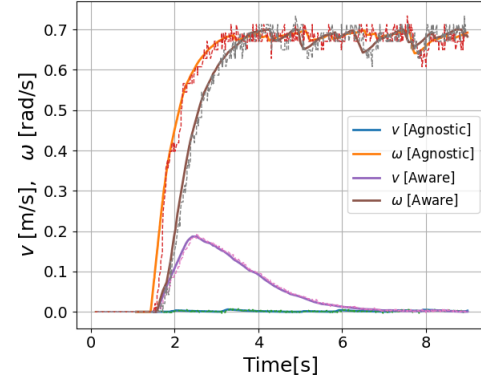
Since there is no navigation involved in this experiment set, the focus is put on the analysis of motor torques. The results shown in Figure 5.5a have been summarized in Table 5.1. According to these results, maximum torques have been reduced by 60.57% and 44.6% for the left and right motors. Regarding mean values, the left motor decreases by 20.7%, while an increase of 1.6% happens for the right side. If mean values of both motors are averaged, the PF reduces torques by 52%.

### 5.3.2 MPC based local planners

So that the performance of MPC based planners when rotating on the spot in hardware is analyzed, CWAGLP and CWAWLP have been compared in a similar manner to the simulations performed in Subsection 4.2.3.



(a) Average (continuous) and maximum (dashed) motor torques.



(b) Command (continuous) and odometry (dashed) velocities.

Figure 5.6: Comparison of velocity commands and torques in Active Shuttle when rotating on the spot for CWAGLP and CWAWLP.

The results given in Figure 5.6a show analogous patterns to the ones obtained in the simulations. The same way that happened in the PF's case, a longitudinal motion gets rid of the initial torque spike. This is shown in Figure 5.6a, where the decrease in average and maximum motor torques can be visualized. The velocity commands in Figure 5.6a justify this difference. The explanation given for Figure 5.5b is applicable to this case. Notice that CWAWLP's torque oscillations that start at  $t = 5$  s are caused by the lack of smoothness in the rotational velocity command (brown).

Information regarding motor torques has been summarized in Table 5.2. The decrease in peak torques is quantified to 44.04% and 26.6% for left and right maximum motor torques. Regarding mean torques, the left motor has been reduced by 21.97% and the right one increased 14.95%. Converting these results to the average of both motor torques, the decrease provoked by the caster wheel aware term is 36%.

Table 5.2: Measurements to evaluate motor torques usage when rotating on spot with CWAGLP (agnostic) and CWAWLP (aware) in the Active Shuttle.

Local planner	$T_{L,max}[\text{Nm}]$	$T_{L,mean}[\text{Nm}]$	$T_{R,max} [\text{Nm}]$	$T_{R,mean}[\text{Nm}]$
Agnostic	7.97	1.82	7.33	1.72
Aware	4.46	1.42	5.38	1.97

### 5.3.3 Conclusions

Both CWAWLP and CWPFLP minimize mean and maximum motor torques according to the caster wheel's state by converting a pure rotational command into a combination of longitudinal and rotational commands. In this way, the robot drives forward instead of rotating on the spot, and consequently, reaction torques in



the caster wheels are minimized. For CWPFLP the reduction on average of both motors is 52% and for CWAFLP is 36%. However, it needs to be highlighted that these values cannot be compared, since they are from experiments. In the first case, the PF was attached to a pure rotational command, while in the latter one, CWAGLP was compared against CWAFLP. However, the responses of both methods were the same. The step-shaped rotational command is smoothed and a low longitudinal command ensures that the robot is moving forward, while it is rotating. From this case study it can be stated that CWPFLP and CWAFLP have the same behavior when rotating on the spot.

## 5.4 Case study II: Navigation across a given global path

In order to observe the local planner's capacity to trade-off between torque mitigation and navigation capabilities, the global path is defined and shown in Figure 5.7.

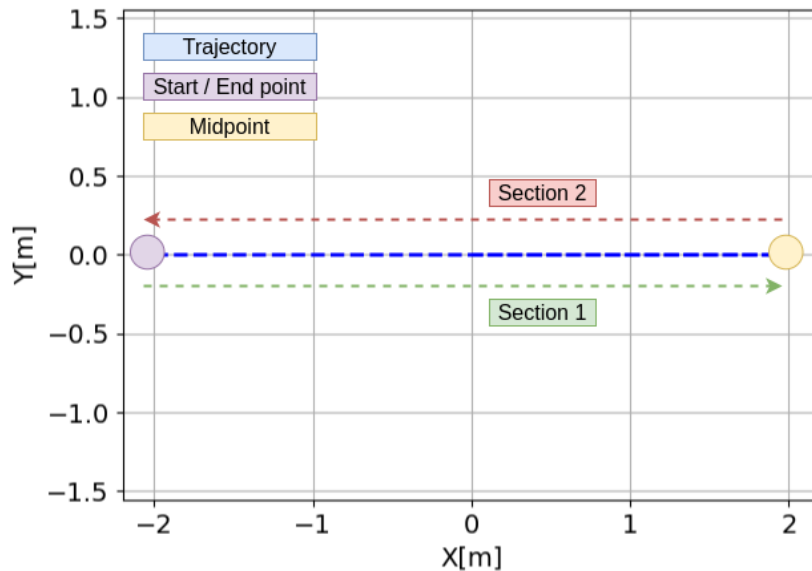
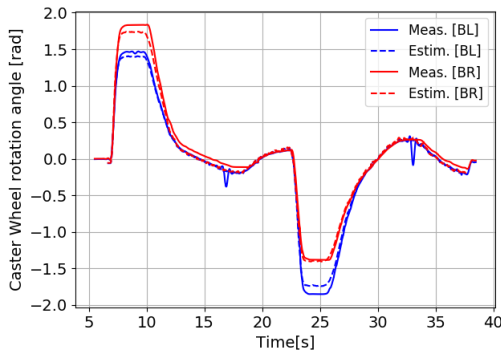


Figure 5.7: Global path for conducting experiments to compare CWAGLP, CWPFLP and CWAFLP.

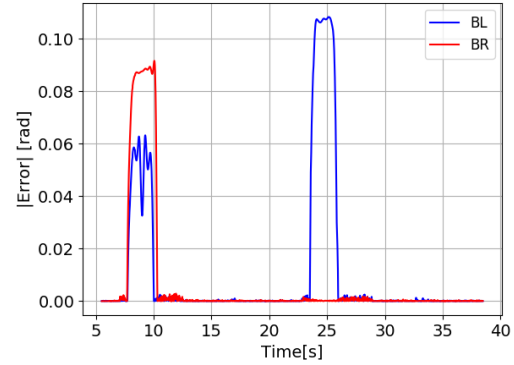
It consists of two sections. Once the time based reference covers the first section (green) and gets to the midpoint (yellow), it waits for the robot to arrive. As soon as this happens, it starts Section 2 (red) and gets back to the origin (purple). Consequently, it involves two rotations on the spot. In order to ensure that uncertainties are included, each planner navigates across the global path 10 times in a continuous manner.

### 5.4.1 Validation of the observer

The caster wheel rotation angles have been measured through the system presented in 5.2.4.3. The measured and estimated angles, along with the respective absolute errors, are given in Figure 5.8



(a) Comparison between measurements and estimations.



(b) Absolute error between measurements and estimations.

Figure 5.8: Analysis of measurements (continuous) and estimations (dashed) of rear caster wheel rotation angles  $\varphi$ . Rear left (BL) is shown in blue and rear right (BR) in red.

Figure 5.8 shows that the estimations and measurements only differ in steady state equilibrium angles by a magnitude of 0.1 rad. The results obtained out of this comparison are very similar to the ones in the simulation based validation in Subsection 4.1. In fact, this error also occurred when testing the observer in the virtual framework (see Figure 4.2). Otherwise, Figure 5.8b proves that the error is negligible ( $\approx 0$  rad) for the rest of the cases. Computing the RMSE values results in 0.0292 rad and 0.0230 rad for rear-left and -right caster wheels. Thus, the observer presented in Section 3.3 is validated and the estimations obtained out of it can be considered to be an accurate measurement of the caster wheel rotation angle.

## 5.4.2 Results

As mentioned above, each planner has been executed 10 times continuously across the global path. In this way, the recorded data includes more cases, uncertainties are minimized and conclusions arising from the results are more generic.

The average trajectory from the 10 iterations is represented by a continuous line in Figure 5.9a, while the possible deviations are included in the shade with the respective color. The same representation has been applied for the velocity commands Figure in 5.9c. When it comes to motor torques, the mean of the averaged torques along the 10 iterations are given by a continuous line and the shades stand for the range of the maximum torques between both motors.

In Figure 5.9a it can be observed that CWPFLP and CWAFLP increase the radius of the curves. However, deviations seem to be similar for all planners. Regarding maximum motor torques, there are two points that should be highlighted. Firstly, CWAFLP's values are lower than those of the other two. Secondly, CWPFLP has two severe peaks at  $t = 0$  s and  $t = 30$  s. In the case of mean torques, visually relevant differences cannot be recognized.

The increase of the curves' radius mentioned in the previous paragraph can be understood by looking into the velocity command Figure in 5.9c. For both cases ( $t = 0$  s,  $t = 30$  s for the first curve and  $t = 15$  s for the second), CWPFLP and CWAFLP decrease the time span where  $v = 0$  m/s. Consequently, the robot drives longitudinally for a longer time and the rotations on spot are neglected, augmenting the trajectory's

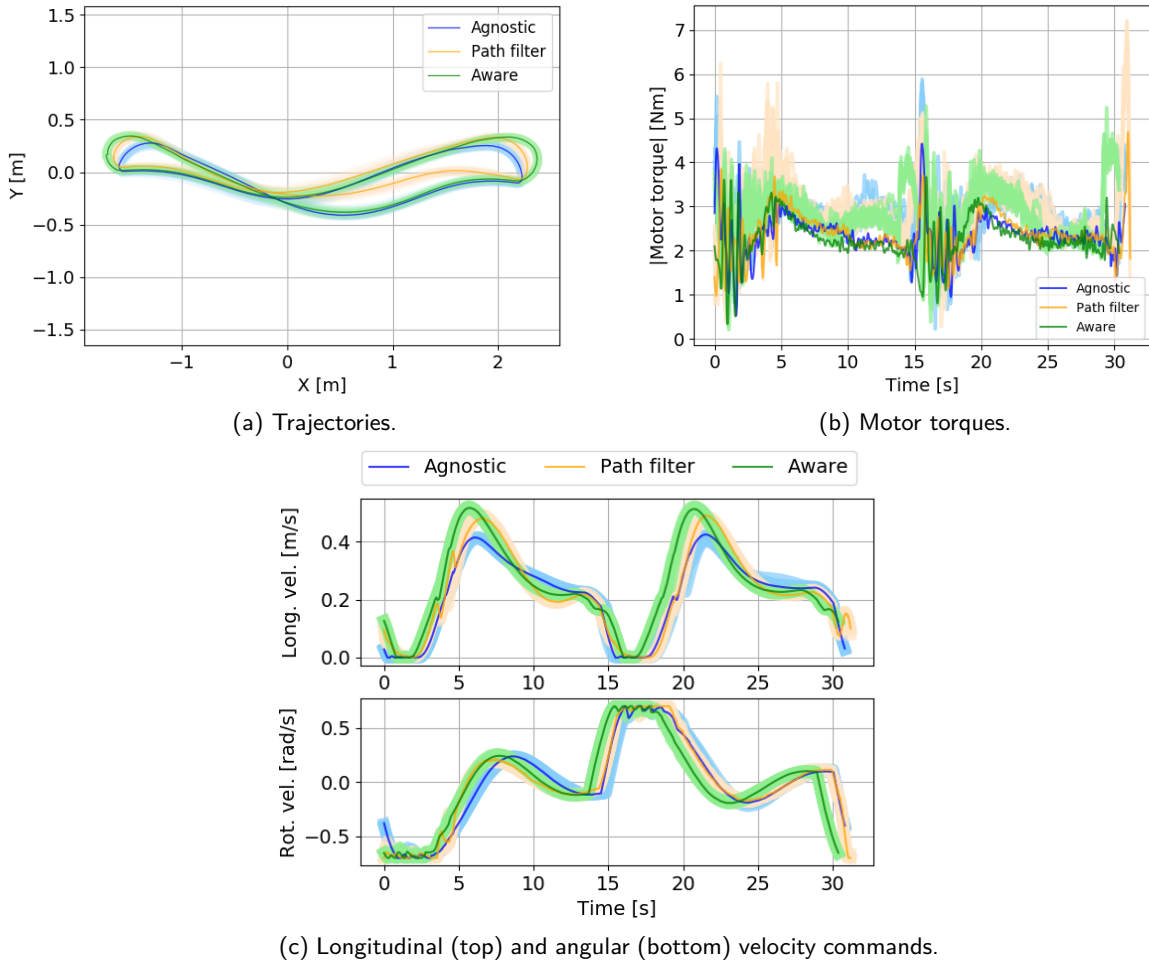


Figure 5.9: Comparison of CWAGLP (blue), CWPFLP (orange) and CWAUWP (green) in Active Shuttle when navigating. The line with a stronger colour represents average value. Maximum values are given by the shade.

radius.

As far as CWPFLP is concerned, there are two points that deserve a detailed explanation. The first relates to the existence of abrupt changes in the longitudinal velocity commands in Figure 5.9c around  $t = 3$  s and  $t = 4$  s. When the PF filters velocity commands, it violates the SCU's constraints explained in Subsection 5.2.4.2, causing this unit to be actuated by sending an interrupt. This scenario is another representation of the problem stated in Section 1.2. The PF is not capable of considering other circumstances and it cannot guarantee that all constraints are fulfilled, obliging to trigger the SCU.

The second observation refers to the aforementioned spikes in maximum motor torques during the first curve ( $X = -2$  m,  $t = 0$  s and  $t = 30$  s). These are caused because the PF is implemented in a single caster wheel, the front-left one in this case. Consequently, velocity commands are filtered only according to the state of this wheel, while those of the other three are not considered. As a consequence, the motor that is in the opposite side is sacrificed. In order to verify this, the PF has been moved to the front-right caster wheel and the case study has been replicated. The obtained results are given in Figure 5.10. When doing so, not only the trajectory changes, but also the spikes of the maximum torque are moved from the first curve ( $t = 0$

s and  $t = 30$  s) to the second ( $t = 15$  s). Since rotations in these curves occur in opposite directions, the PF only favours one of the rotations and generates torque-peaks in the other case. This issue can be solved by extending the PF presented in paper [5], so that the states of all four caster wheels are considered when velocity commands are filtered.

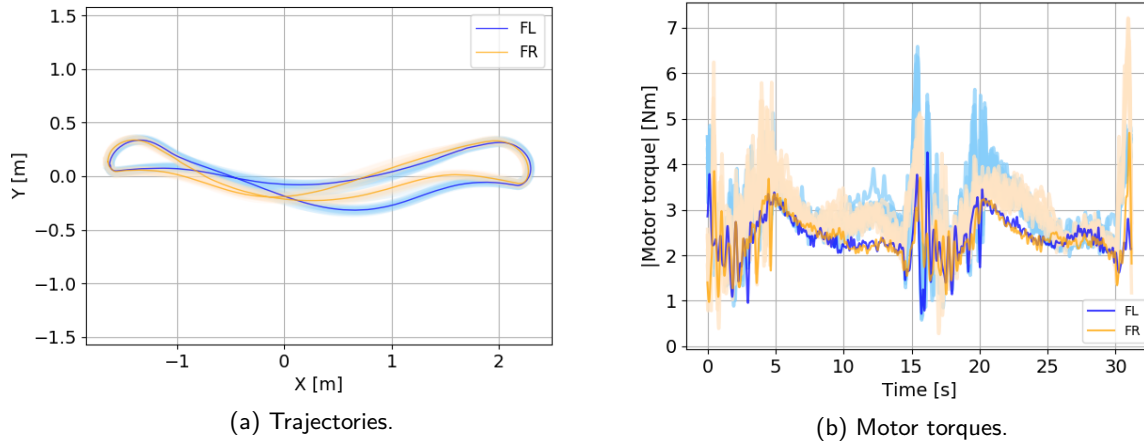


Figure 5.10: Comparison of applying CWPFLP in the front left (FL) or front-right (FR) in the AS when navigating. The line with a stronger colour represents average value. Maximum values are given by the shade.

### 5.4.3 Analysis

The results obtained above are analyzed in three separate blocks. Firstly, navigation characteristics of each planner are studied and a comparison is conducted. Secondly, motor torque usage is evaluated by looking into mean and maximum torques. Thirdly, the energy consumed by each planner is analyzed.

#### 5.4.3.1 Navigation

Trajectories obtained in Figure 5.9a are quantified in Table 5.3, where covered distance, navigation's time and quality (MAE, RMSE) for the average of the 10 repetitions are given. Deviations obtained out of the other repetitions are included as uncertainties.

Table 5.3: Measurements to evaluate navigation's performance when applying CWAGLP (agnostic), CWPFLP (path filter) and CWAWLPL (aware) for navigation case study in the AS.

Local planner	Distance [m]	Time [s]	MAE [m]	RMSE [m]
Agnostic	$8.28 \pm 0.03$	$30.73 \pm 0.06$	$0.1654 \pm 0.0093$	$0.1973 \pm 0.0099$
Path filter	$8.51 \pm 0.02$	$31.08 \pm 0.1$	$0.13 \pm 0.0022$	$0.1657 \pm 0.002$
Aware	$8.91 \pm 0.03$	$30.33 \pm 0.03$	$0.1699 \pm 0.0033$	$0.2057 \pm 0.0047$

The first column of Table 5.3 shows that CWPFLP and CWAWLPL cover longer distances than CWAGLP. This is related to the fact that they take wider curves. For the first case, the difference is 0.23 m, while for the latter it is 0.63 m. Even if CWAWLPL navigates across a longer path, it needs 0.4 s less than CWAGLP. This does not hold for CWPFLP, because it requires 0.35 s more. Regarding navigation's quality, CWAGLP and CWAWLPL are very close to each other. The maximum difference in RMSE considering uncertainties is 4%. In terms of average deviation (MAE), this is equivalent to a difference of 8.1 mm. CWPFLP performs

better, but the deviation 4 cm with respect to the other two still remains insignificant when considering the robot's dimensions. Notice that all the results that have been obtained by field test have also been observed when doing simulations in the virtual framework (see Table 4.3).

#### 5.4.3.2 Motor torques

The numerical results in Figure 5.9b are summarized in Table 5.4, where the mean and maximum motor torques are quantified. CWAWLPL reduces mean torques by 0.04 Nm and maximum torques by 0.9 Nm compared to CWAGLP. In percentages, these decrements are equivalent to 3.83% and 15.86%. In the case of CWPFLP, the already explained implementation issues combined with the interrupt of the SCU, result in an increase of mean and maximum torques. The first rise is negligible (1.3%), while the second is severe (20%).

Table 5.4: Measurements to evaluate motor torques usage when applying CWAGLP (agnostic), CWPFLP (path filter), CWAWLPL (aware) for navigation case-study in the AS.

Local planner	$T_{\text{mean}}[\text{Nm}]$	$T_{\text{max}}[\text{Nm}]$
Agnostic	$2.35 \pm 0.04$	$5.64 \pm 0.25$
Path filter	$2.38 \pm 0.08$	$6.77 \pm 0.44$
Aware	$2.26 \pm 0.05$	$4.74 \pm 0.53$

#### 5.4.3.3 Energy

The energy consumption has also been measured by adding the product of the force and the distance covered at every sample.

$$E = \sum_{t=0}^{t_{\text{end}}} \frac{T_t}{r_{\text{dw}}} \cdot d_t \quad (5.1)$$

where  $T$  is the motor torque and  $d_t$  is the distance covered during every sampling interval. The absolute energy consumption, along with the ratio respective to the CWAGLP are given in Table 5.5.

Table 5.5: Energy consumption when applying CWAGLP (agnostic), CWPFLP (path filter), CWAWLPL (aware) for navigation case-study in the Active Shuttle.

Local planner	$E_{\text{total}}[\text{J}]$	$E_{\text{ratio}}$
Agnostic	$394.89 \pm 5.28$	1.0
Path filter	$419.79 \pm 13.6$	$1.063 \pm 0.049$
Aware	$418.83 \pm 9.37$	$1.061 \pm 0.038$

Due to the fact that CWPFLP covers a longer distance and the average motor torque is also higher, its energy consumption is the largest of all. In the case of CWAWLPL, the reduction in mean motor torques is not sufficient when compared to the increase in the covered distance and the decrease in navigation time, causing the PF to consume more energy. Consequently, both, CWPFLP and CWAWLPL, consume approximately 24 J more, equivalent to an rise of 6%.

#### 5.4.4 Conclusions

When navigating across a global path in a continuous manner some differences have been identified. CWAWLPL covers more distance in less time. However, the deviation of the robot's trajectory with respect to the global

path remains very similar for all three planners. The most significant differences have been observed in motor torques, where CWAFLP has been able to minimize both, mean and maximum values. This cannot be achieved on CWPFLP, since both quantities have increased.

In fact, regarding CWPFLP, two weak-points have been spotted. Firstly, applying the PF to a single caster wheel neglects the state of other caster wheels and endangers the motor on the opposite side. Secondly, it has been observed that the PF cannot guarantee the fulfillment of other constraints, since it has infringed the restrictions related to the SCU. In a similar manner, restrictions regarding obstacle avoidance could be violated and thus, it can be stated that the PF cannot guarantee the navigation's completion.

Even if the energy consumptions for all three planners are very similar, the ones of CWPFLP and CWAFLP are slightly higher. This can be explained by the fact that the navigation's duration and distance are different for the three planners.

## 5.5 Summary

In this Chapter the motion planner formulated in Section 3.6 has been tested in an experimental setup. The focus has been put on two case studies. Before executing the experiments, the setup, along with other implementation concepts have been explained. Moreover, the design and assembly of the caster wheel angle's measurement system have been briefly introduced.

From the first case study, a rotation on the spot, it has been learnt that the CWPFLP and CWAFLP behave in an analogous manner. Both planners convert the angular velocity command into a combination of longitudinal and angular commands. In this way, the robot drives forward before it starts to rotate, which guarantees that the caster wheel is rolling once it changes the orientation, decreasing the reaction torques. The influence of this motion has shown to have a positive impact on the motor torques.

The second case study consists on driving back and forth within a 4m long space. As a starting point, the observer has been validated by comparing the estimations against the measurements obtained from the system. When it comes to case study's results, CWAFLP is the planner whose mean and maximum torques are the lowest. At the same time, it covers the longest distance in the shortest time. This does not hold for CWPFLP, because the current implementation of the PF sacrifices one motor. Even if an extension can solve this issue, it has been confirmed that it cannot be aware of constraints included in the motion planning stage.

# Conclusions

---

The last Chapter summarizes the content of the thesis, highlights the most relevant points and proposes an answer to the research question formulated in Section 1.3. To do so, a summary that puts together all the steps leads to the listing of key contributions that have been crucial for answering the research question. Possible extensions are analyzed after discussing the capabilities and limitations of the presented concept by proposing future directions for research.

## 6.1 Discussion

In the first case study of the simulations it has been shown that CWPFLP and CWAFLP make the robot drive forward instead of turning while standing still. In this way, caster wheels are rolling when their orientation changes, resulting in a decrease of reaction torques. However, in the second case study, improvements of CWPFLP and CWAFLP with respect to CWAGLP have not been noticed. The reason is that the applied torques depend on how the reference tracking is tuned, hindering the visualization of the caster wheel aware term's influence in motor torque minimization.

Regarding experiments, the simulation's results for rotations on the spot have been replicated. When it comes to navigation, due to timing and hardware requirements, significant changes in implementation were required. One of these modifications consisted on adding hardware related constraints to the OCP, which caused issues in the CWPFLP. Otherwise, the outcome is very similar to the one in simulations.

Considering that the analysis in Section 5.4 is particularized for the global path given in Figure 5.7, the quantitative results are not applicable for other cases. Changes in the desired trajectory or modifications in the tuning parameters, would cause significant variations in the recorded data. There are however some patterns that are always fulfilled and can be generalized.

$$\begin{aligned} \text{Navigation [RMSE]} &\rightarrow \text{CWAFLP} \approx \text{CWAGLP} > \text{CWPFLP} \\ T_{\text{mean}}, T_{\text{max}} [\text{Nm}] &\rightarrow \text{CWAGLP} > \text{CWPFLP} \approx \text{CWAFLP} \\ \text{Energy cons. [J]} &\rightarrow \text{CWPFLP} \approx \text{CWAFLP} > \text{CWAGLP} \end{aligned} \tag{6.1}$$

Firstly, CWAJLP covers a longer distance in less time without reducing the navigation's quality. Even if the CWPJLP's trajectory tends to be a little closer to the global path than the other two, differences are insignificant when compared to the robot's dimensions. Secondly, CWAJLP and CWPJLP, are capable of minimizing mean and maximum motor torques. For both cases, the maximum motor torques are reduced in a higher rate than mean values. Even if both perform in a similar manner, it has been shown that CWPJLP sacrifices one of the motors and is not aware of the constraints included in the planning stage. Since the decrease in mean torques is not sufficient to overcome the increase in the covered distance, CWPJLP and CWAJLP consume more energy than CWJLP.

Putting together the trends observed in simulations and experiments, the research question can be answered by stating that the caster wheel aware MPC based motion planner proposed in this thesis (CWAJLP) is capable of considering caster wheel reaction torques in the motion planning stage without compromising navigation capabilities.

## 6.2 Summary

Caster wheels offer high maneuverability in a compact and cheap assembly at the expense of reaction torques. By looking into the effects of these in differential drive mobile robots, it has been identified that they significantly increase the motor torque demand. This phenomena, not only jeopardizes the motor's lifetime, but also increases the energy consumption and degrades the navigation's performance.

When looking into what has been done previously in this field, the focus has been put in paper [5], where a path filter (PF) is attached to, but not integrated into the planning stage. Thus, it cannot guarantee the fulfillment of constraints, such as obstacle avoidance or hardware demands, endangering the navigation's completion. This has led to a research question which aims to integrate caster wheel physics into a local planner.

The ease to solve nonlinear dynamics and include constraints make MPC a suitable control approach for considering caster wheel motions and, simultaneously, following the global path. Since the focus is put in the first, the motion planning problem has been simplified to reference tracking.

Having defined the research question and the methodology, every term involved in the MPC's OCP has been determined. As a starting point, a plant model with seven states and longitudinal and angular accelerations as inputs is presented. To this end, the differential drive and caster wheel models have been analyzed separately. In the latter case, the equations of motion of the caster wheel's rotation angle and rolling speed have been derived.

Subsequently, the OCP's cost function has been formulated by splitting it up into two terms: the first one is in charge for the navigation, while the second involves caster wheel reaction torques. The first one consists on penalizing the distance from the robot to a time based reference, while the second minimizes the difference between steady state and current caster wheel rolling speeds. Since the caster wheel state (rotation angle and rolling speed) is unknown, an observer that estimates it has also been proposed. Finally the MPC's sampling time and horizon have been established by considering the fastness of the equations of motion of the caster wheel rotation angle. Putting everything together has determined the OCP that stands behind the MPC.



This formulation has been tested in a virtual framework, where the robot is substituted by a FMU exported from the AS's Modelica model. After verifying that the estimations obtained from the observer are similar to the angles measured from the model, three planners (CWAGLP, CWPFLP and CWAUPL) have been compared against each other in three different testing cases. The first one consisted on rotating on the spot and was extended by the second one, where the robot had to rotate on the spot and navigate. Thirdly, the robot had to navigate across two different global paths. Combining the results obtained in all three case studies, it is concluded that CWPFLP and CWAUPL are capable of considering caster wheel physics without causing severe deviations with respect to CWAGLP's trajectory.

In order to validate the results obtained in the simulations, all three planners have been implemented in an experimental setup. Before getting into the case studies, the observer has been validated by comparing estimations against real caster wheel rotation angles measured from two sensors assembled at the rear axle of the robot. In the first case study, which consisted on rotating on the spot, PF and CWAUPL have shown a similar behavior. However, some differences have been identified in the second one, which consisted on driving the robot back and forth by following a time based reference in a continuous manner for 10 times. CWAUPL has shown to be the only planner capable of minimizing mean and maximum torques. Omitting implementation and conceptual issues of CWPFLP, results obtained in experiments are very similar to the ones in simulations.

In the list below, the contributions that were crucial for integrating caster wheel physics into a local planner are summed up. To the writer's knowledge there is no prior work that has addressed these topics.

1. **Formulation of a caster wheel aware term compatible with any MPC framework.** Instead of penalizing the difference between steady state and current caster wheel rotation angles, an alternative expression for differences in rolling speeds has been found. In this way, the caster wheel aware term is finite, defined and differentiable in the entire input range, which generalizes its usage to other OCP solvers and is compatible with any MPC framework.
2. **The caster wheel aware term can be combined with any MPC based navigation algorithm.** Given that the navigation and the caster wheel aware terms are decoupled in the cost function, the caster wheel aware term can be implemented in any MPC based navigation algorithm, providing modularity and facilitating its generalization to other cases.
3. **The caster wheel aware term is applicable to any vehicle with caster wheels.** The caster wheel aware term is adjustable to other vehicles by replacing the DDMR motion equations in the plant model with the vehicle's drivetrain equations. For example, this attribute enables to implement the caster wheel awareness in car-like mobile robots.
4. **Formulation of a caster wheel state (rotation angle and rolling speed) observer.** Since the caster wheel aware term requires knowing the caster wheel's state, an observer that estimates rotation angles and rolling speeds has been derived. When analyzing the observer's stability, a Lyapunov function has been found that proves that the steady state angle is an asymptotically stable equilibrium point. Therefore, every estimation converges to the steady state angle respective to the applied velocity commands. This observer is also applicable to estimate the states of a caster wheel in any vehicle whose longitudinal and angular velocities are known.

## 6.3 Future Work

A brief outlook is given in two subsections to further develop the motion planner presented by this thesis. Firstly, solutions to weak points or minor bugs are presented. Secondly, paths for further research in this field are proposed.

### 6.3.1 Research Tasks

#### Extending PF to both sides

When running experiments, it has been discovered that implementing the PF to a single caster wheel does not take the status of other wheels into account, sacrificing the motor in the opposite side. An extension of the PF, so that it filters velocity commands by considering the states of all the caster wheels would solve this issue. Notice that such an improvement would ensure that both motors are equally treated, but the PF would still remain being incapable of fulfilling constraints introduced in the planning stage.

#### Extend caster wheel aware term to all wheels

The plant model applied in the thesis has 7 states, out of which the last two are front-left and -right caster wheel rotation angles. Adding two more equations would be sufficient for including both wheels of the rear axle. Considering that the AS's front wheels are significantly bigger than the rear ones, they have more influence in the robot's overall performance. However, if the proposed motion planner wants to be implemented in other robots whose wheels have the same size, implementing the equations in all four wheels might be interesting. Notice that such an extension would increase the OCP's solving time, compromising the frequency at which the MPC runs.

#### Re-evaluate experiments

In order to evaluate navigation capabilities in the experiments, three parameters have been observed: distance, duration and quality (RMSE and MAE). Instead of navigating across the global path 10 times continuously and observing all three parameters vary with respect to each other, fixing one of them would result in a more thorough comparison. This would be relevant for comparing the consumed energy if either navigation's distance or duration are fixed.

### 6.3.2 Research Directions

#### Including dynamics in the plant model

Even if extending the kinematic based plant model to the field of dynamics increases the number of equations and parameters, this opens the possibility to modify the caster wheel aware term, so that a simplified version of the bore torque's analytical expression is introduced in the cost function. For this purpose, it would be necessary to modify Equation 2.1, since it includes discontinuities and fractions that would cause numerical issues when solving the OCP.

Applying this upgrade to the observer's model might solve the issue related to identifying the change in caster wheel orientation for the forward-backward case (Section 4.1.2). Moreover, when including inertias and a tyre model for the caster wheels, the observer's steady state error of 0.1rad might disappear. Never-

theless, this modification hinders the applicability of the model, since it becomes vulnerable to the fitting of parameters.

### **Replace the navigation term**

As mentioned above, the caster wheel aware term is compatible with other MPC navigation algorithms and frameworks. Considering that the navigation part of the motion planner presented in this thesis consists on reference tracking, the navigation strictly relies on the tuning of how the moving reference is followed.

Therefore, it would be interesting to replace the reference tracking navigation with a more mature MPC based navigation algorithm, such as Tunnel Following NMPC [45], where the distance to a tunnel is penalized and any trajectory within the corridor is equally good. In such a framework, the caster wheel aware term would have a greater influence, since the trajectory with the lowest reaction torques among the ones inside the corridor would be chosen. In addition, new scenarios, such as obstacle avoidance, could be observed.



---

# References

---

- [1] Jonathan Tilley. "Automation, Robotics, and the Factory of the Future". In: *McKinsey*. <https://www.mckinsey.com/business-functions/operations/our-insights/automation-robotics-and-the-factory-of-the-future> (2017).
- [2] Badea Sorin-Ionut. "Intelligent Devices for Transporting Parts Between Processing Systems, Ultra-precise Cyber-Mechatronic Systems for Industrial and Laboratory Control (For Molded Parts in the Automotive Industry) and the Assembly Line". In: *International Conference of Mechatronics and Cyber-Mixmechatronics*. Springer. 2020, Bucharest, Romania, pp. 165–172.
- [3] Bosch Rexroth. *Active Shuttle - setting your intralogistics in motion*. 2020 (accessed August 3, 2020). URL: <https://www.boschrexroth.com/en/xc/products/product-groups/assembly-technology/topics/intralogistics/template-neuprodukt-seite-6>.
- [4] Yuan Ping Li, Teresa Zielinska, Marcelo H Ang, and Wei Lin. "Vehicle dynamics of redundant mobile robots with powered caster wheels". In: *Romansy 16*. Springer, 2006, pp. 221–228.
- [5] Nikolas Schröder, Oliver Lenord, and Ralph Lange. "Enhanced Motion Control of a Self-Driving Vehicle Using Modelica, FMI and ROS". In: *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019*. 157. Linköping University Electronic Press. 2019.
- [6] Hilding Elmqvist, Sven Erik Mattsson, and Martin Otter. "Modelica-a language for physical system modeling, visualization and interaction". In: *Proceedings of the 1999 IEEE international symposium on computer aided control system design (Cat. No. 99TH8404)*. IEEE. 1999, Kohala Coast, HI, USA, pp. 630–639.
- [7] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. "Trajectory modification considering dynamic constraints of autonomous robots". In: *ROBOTIK 2012; 7th German Conference on Robotics, Munich Germany*. VDE. 2012, pp. 1–6.
- [8] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. "Integrated online trajectory planning and optimization in distinctive topologies". In: *Robotics and Autonomous Systems* 88 (2017), pp. 142–153.
- [9] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1 (1997), pp. 23–33.
- [10] Sean Quinlan. *Real-time modification of collision-free paths*. 1537. Stanford University Stanford, 1994.
- [11] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

- [12] David H Shim, H Jin Kim, and Shankar Sastry. "Decentralized nonlinear model predictive control of multiple flying robots". In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475), Maui, Hawaii, USA*. Vol. 4. IEEE. 2003, pp. 3621–3626.
- [13] Dongbing Gu and Huosheng Hu. "A stabilizing receding horizon regulator for nonholonomic mobile robots". In: *IEEE Transactions on Robotics* 21.5 (2005), pp. 1022–1028.
- [14] Rached Dhaouadi and A Abu Hatab. "Dynamic modelling of differential-drive mobile robots using lagrange and newton-euler methodologies: A unified framework". In: *Advances in Robotics & Automation* 2.2 (2013), pp. 1–7.
- [15] Stefan Staicu. "Dynamics equations of a mobile robot provided with caster wheel". In: *Nonlinear Dynamics* 58.1-2 (2009), p. 237.
- [16] Dirk Zimmer and Martin Otter. "Real-time models for wheels and tyres in an object-oriented modelling framework". In: *Vehicle system dynamics* 48.2 (2010), pp. 189–216.
- [17] Torsten Blochwitz, Martin Otter, Martin Arnold, Constanze Bausch, H Elmqvist, A Junghanns, J Mauß, M Monteiro, T Neidhold, Dietmar Neumerkel, et al. "The functional mockup interface for tool independent exchange of simulation models". In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical University; Dresden; Germany*. 063. Linköping University Electronic Press. 2011, pp. 105–114.
- [18] Peter Fritzson and Vadim Engelson. "Modelica—A unified object-oriented language for system modeling and simulation". In: *European Conference on Object-Oriented Programming*. Springer. 1998, Brussels, Belgium, pp. 67–90.
- [19] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan. 2009, p. 5.
- [20] Frank Allgöwer and Alex Zheng. *Nonlinear model predictive control*. Vol. 26. Birkhäuser, 2012.
- [21] Kun Zhang, Jonathan Sprinkle, and Ricardo G Sanfelice. "A hybrid model predictive controller for path planning and path following". In: *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*. 2015, New York, NY, United States, pp. 139–148.
- [22] Jean-Claude Latombe. *Robot motion planning*. Vol. 124. Springer Science & Business Media, 2012.
- [23] A Pedro Aguiar, João P Hespanha, and Petar V Kokotović. "Performance limitations in reference tracking and path following for nonlinear systems". In: *Automatica* 44.3 (2008), pp. 598–610.
- [24] Timm Faulwasser and Rolf Findeisen. "Nonlinear model predictive control for constrained output path following". In: *IEEE Transactions on Automatic Control* 61.4 (2015), pp. 1026–1039.
- [25] F Kühne, J Gomes, and W Fetter. "Mobile robot trajectory tracking using model predictive control". In: *II IEEE latin-american robotics symposium*. Vol. 51. Citeseer. 2005.
- [26] Kiattisin Kanjanawanishkul, Marius Hofmeister, and Andreas Zell. "Smooth Reference Tracking of a Mobile Robot using Nonlinear Model Predictive Control." In: *ECMR*. 2009, pp. 161–166.
- [27] Jamal Daafouz, Pierre Riedinger, and Claude lung. "Stability analysis and control synthesis for switched systems: a switched Lyapunov function approach". In: *IEEE transactions on automatic control* 47.11 (2002), pp. 1883–1887.

- [28] Karl J Åström and Björn Wittenmark. *Computer-controlled systems: theory and design*. Courier Corporation, 2013.
- [29] Karl Worthmann. "Estimates on the prediction horizon length in model predictive control". In: *Proceedings of the 20th International Symposium on Mathematical Theory of Networks and Systems, CD-ROM, MTNS2012*. Vol. 112. 2012, Melbourne, Australia.
- [30] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. "CasADi – A software framework for nonlinear optimization and optimal control". In: *Mathematical Programming Computation* 11.1 (2019), pp. 1–36. DOI: 10.1007/s12532-018-0139-4.
- [31] John C Butcher. "Coefficients for the study of Runge-Kutta integration processes". In: *Journal of the Australian Mathematical Society* 3.2 (1963), pp. 185–201.
- [32] Francis Begnaud Hildebrand. *Introduction to numerical analysis*. Courier Corporation, 1987.
- [33] Sanjay Mehrotra. "On the implementation of a primal-dual interior point method". In: *SIAM Journal on optimization* 2.4 (1992), pp. 575–601.
- [34] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.
- [35] C Laird and A Wächter. "Introduction to IPOPT: a tutorial for downloading, installing, and using IPOPT. Revision No. 1863". In: *Electronically published: <http://www.coin-or.org/Ipopt/documentation/> (accessed at 11.02. 2013) ()*.
- [36] Iain S Duff and John K Reid. *MA27-a set of Fortran subroutines for solving sparse symmetric sets of linear equations*. UKAEA Atomic Energy Research Establishment, 1982.
- [37] E Alper Yildirim and Stephen J Wright. "Warm-start strategies in interior-point methods for linear programming". In: *SIAM Journal on Optimization* 12.3 (2002), pp. 782–810.
- [38] Dassault Systèmes. *FMPy - Free Python library to simulate Functional Mock-up Units (FMUs)*. 2020 (accessed August 6, 2020). URL: <https://github.com/CATIA-Systems/FMPy>.
- [39] Tianfeng Chai and Roland R Draxler. "Root mean square error (RMSE) or mean absolute error (MAE)?—Arguments against avoiding RMSE in the literature". In: *Geoscientific model development* 7.3 (2014), pp. 1247–1250.
- [40] Inc. DBA OptiTrack. *Opti Track Motion Cameras*. 2020 (accessed August 6, 2020). URL: <https://optitrack.com/>.
- [41] Jan Palach. *Parallel programming with Python*. HPDC '19: Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing - Packt Publishing Ltd, 2014, New York, NY, United States.
- [42] Joshua S Furtado, Hugh HT Liu, Gilbert Lai, Herve Lacheray, and Jason Desouza-Coelho. "Comparative analysis of optitrack motion capture systems". In: *Advances in Motion Sensing and Control for Robotic Applications*. Springer, 2019, pp. 15–31.
- [43] CUI Devices. *AMT22 - Absolute Encoders*. 2020 (accessed September 7, 2020). URL: <https://www.cuidevices.com/product/resource/amt22.pdf>.

- [44] PJRC Electronic Projects. *Teensy 3.6*. 2020 (accessed September 7, 2020). URL: <https://www.pjrc.com/teensy/techspecs.html>.
- [45] Niels van Duijkeren. "Online Motion Control in Virtual Corridors-for Fast Robotic Systems". PhD thesis. MECO Research Team, Katholieke Universiteit Leuven, 2019.

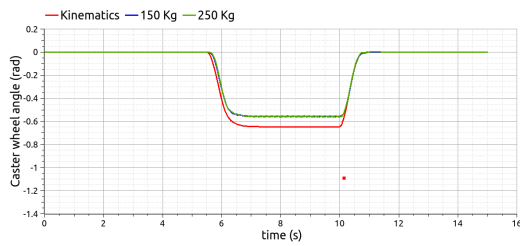


# Appendix A

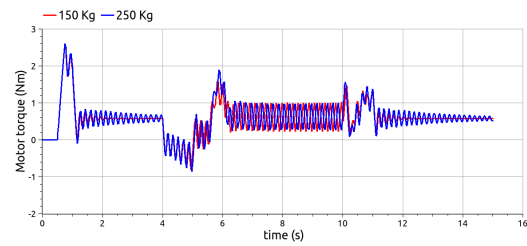
## Figures

The following figures have been referenced during the thesis and might help the reader to understand the presented content. Notice that each Figure was added according to the Section in which it was cited.

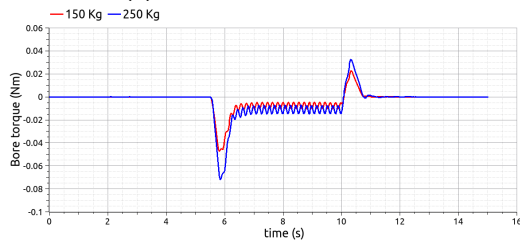
### A.1 Fundamentals - Case Identification



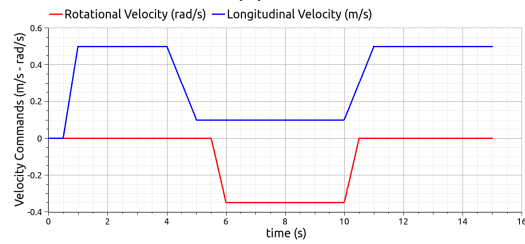
(a) Left front caster wheel angle.



(b) Left motor torque.



(c) Left front bore torque.



(d) Velocity commands for cornering with  $V = 0.1 \text{ m/s}$ .

Figure A.1: Caster wheel angles, motor torques and bore torques for a cornering case of  $90^\circ$  with longitudinal velocity of  $0.1 \text{ m/s}$  (Figure A.1d) with respect to kinematics plant model.

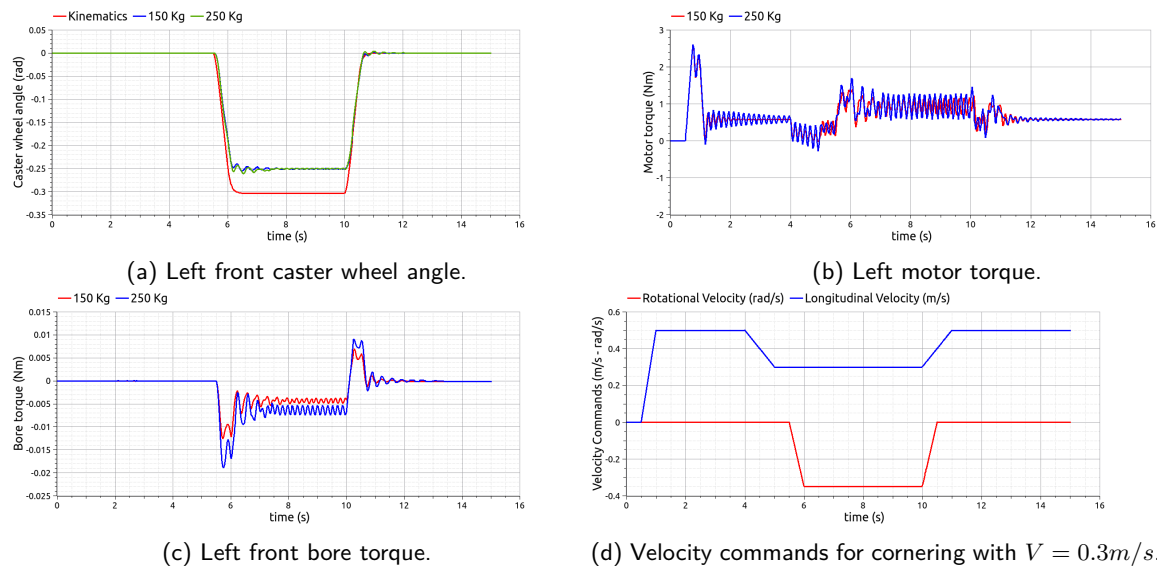
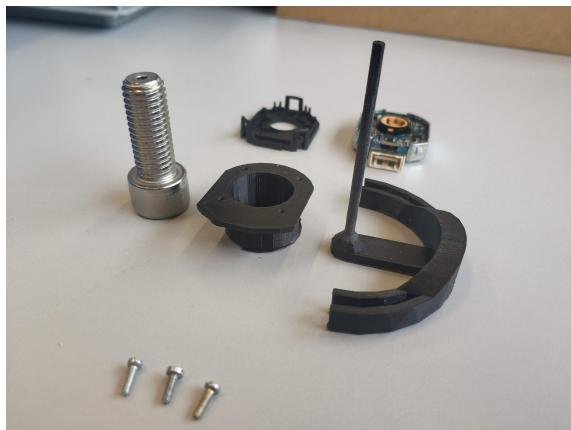
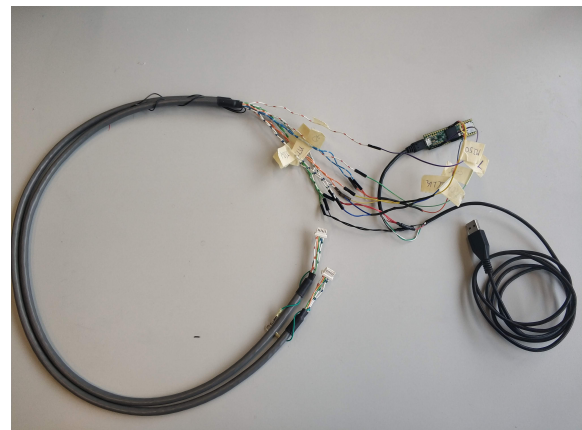


Figure A.2: Caster wheel angles, motor torques and bore torques for a cornering case of  $90^\circ$  with longitudinal velocity of  $0.3m/s$  (Figure A.2d) with respect to kinematics plant model.

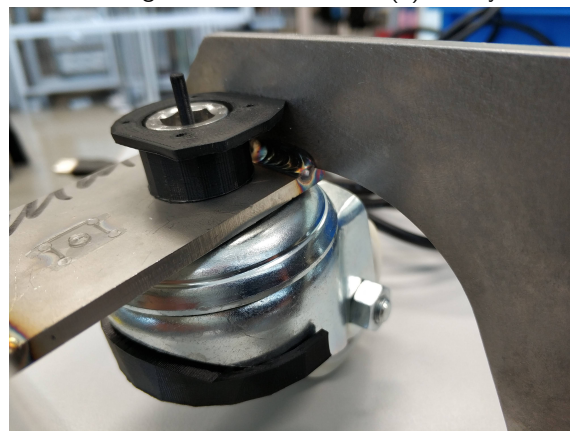
## A.2 Experiments - Sensors



(a) Components before assembling.



(b) Teensy 3.6 microcontroller with wiring.



(c) Top view of sensor's assembly without encoder.

Figure A.3: Assembly of encoder to measure rear axle caster wheel rotation angles.

## Appendix B

---

### Scripts

---

In Section 5.2.3 the navigation algorithm is divided into several classes, providing modularity and ease of use. The pseudo code given in this Section shows how these classes interact with each other. This logic has been applied through the ROS network presented in Section 5.2.2. The respective scripts have been added to this appendix.

#### B.1 Local planner

It combines all the classes by following the procedure explained in the pseudo code. The global path and the controller's tuning parameters are defined by the user in an external text file.

```
1  #!/usr/bin/env python
2
3  import numpy as np
4  import multiprocessing as mp
5  #import threading
6  #import time
7
8  import rospy
9  import tf
10 from std_msgs.msg import Float32
11 from geometry_msgs.msg import Twist, PoseStamped
12 from nav_msgs.msg import Odometry, Path
13 from sensor_msgs.msg import Joy
14 from casterwheelaware_mpc_ROS.msg import caster_state
15
16
17 from MPC_local_planner import *
18 from caster_estimator import *
19 from navigation_local_planner import *
20 from caster_estimator import *
21 from caster_path_filter import *
22
23 class local_planner_ROS():
```

```

24
25
26 def __init__(self, x0, horizon, model_parameters, f_LLC, tolerance, trajectory_path,
    mpcParameter_path):
27
28     #####General variables#####
29     #Opti-tracker marker displacement
30     self.dxOpti = -0.31688          # x-distance from origin of opti marker to center of
    robot
31     self.dyOpti = 0.034213          # x-distance from origin of opti marker to center of
    robot
32
33
34     #Step sizes
35     self.ts_MPC = horizon[0]/horizon[1]    #step size of MPC (20Hz)
36     self.ts_LLC = 1/float(f_LLC)          #step size of low level controller (motor commands
    , 50Hz)
37
38     #Flags
39     self.odom_fbk = False                #flag to define if feedback from odom was received
40     self.pose_fbk = False                #flag to define if feedback from odom was received
41     self.cmdvelDes_fbk = False           #flag to define if feedback from twistDes was
    received
42     self.pf_flag = False                  #flag to define if path filter is active
43     self.drive_mode = "joy"              #flag to activate mpc navigation
44
45     #initial states
46     self.x = x0
47     self.X = x0[0]
48     self.Y = x0[1]
49     self.TETA = x0[2]
50     self.v_odom = x0[3]
51     self.w_odom = x0[4]
52     self.cmdV = x0[3]
53     self.cmdW = x0[4]
54     self.vDes = x0[3]
55     self.wDes = x0[4]
56
57     #mpc
58     self.mpcParameter_path = mpcParameter_path
59     self.horizon = horizon
60     self.MPC_reinit = True
61     self.pf_update = True
62     self.model_parameters = model_parameters
63     self.f_LLC = f_LLC
64
65     #navigation
66     self.trajectory_path = trajectory_path
67     self.tolerance = tolerance
68
69
70     #####Initialize ROS#####

```

```

71     rospy.init_node('MPC_local_planner', anonymous=True)
72     rospy.loginfo("Initializing ROS...")
73
74
75     #robot
76     self.odometry_sub = rospy.Subscriber('odometry_data', Odometry, self.odometryCallback)
77     self.pose_sub = rospy.Subscriber('vrpn_client_node/ActiveShuttle/pose', PoseStamped,
self.poseCallback)
78     self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=50)
79
80
81     #path filter
82     self.cmd_velDes_sub = rospy.Subscriber('cmd_vel_Des', Twist, self.cmdvelDesCallback)
83     self.pf_trigger_sub = rospy.Subscriber('joy', Joy, self.joyTriggerCallback)
84
85     #mpc
86     self.cmd_velDes_pub = rospy.Publisher('cmd_vel_Des', Twist, queue_size=50)
            # desired velocity by MPC
87     self.mpc_horizon_pub = rospy.Publisher('mpc/horizon', Path, queue_size=1000)
            # horizon estimation of MPC
88     self.stopPt_pub = rospy.Publisher('moving_reference/stop_points', Path, queue_size
=1000)
            # stopping points of path
89     self.trajectory_pub = rospy.Publisher('moving_refence/trajectory', Path, queue_size
=1000)
            # current trajectory
90     self.moving_reference_pub = rospy.Publisher('moving_reference/reference', Path,
queue_size=50)
            # moving reference
91
92     #others
93     self.caster_state_pub = rospy.Publisher('estimator/caster_state', caster_state,
queue_size=50)
94
95     #####Initialize estimator#####
96
97     rospy.loginfo("Initializing estimator...")
98
99     #Solver parameters
100     dae_solver = 'rk'
101     dae_opts = {'tf':self.ts_LLC, 'simplify':True, 'number_of_finite_elements':4}
102     # jit_opts = {"jit": True, "compiler":"shell", "jit_options": {"compiler": "ccache gcc
", "compiler_flags":["-O3"]}}
103     # self.dae_opt.update(jit_opts)
104
105     #Definition of estimator
106     self.est = estimator(    initial_state = [x0[5],x0[6],0,0],                #we suppose it
starts standing still --> gammadot=0rad/s
107                             f_controller = self.f_LLC,
108                             dae_solver = dae_solver,
109                             dae_opts = dae_opts,
110                             model_parameters = self.model_parameters)
111
112
113     #####launch#####

```

```

114
115     #start navigaton depending on drive_mode
116     try:
117         self.choose_drive_mode()
118     except rospy.ROSInterruptException:
119         pass
120
121     def initMPC(self):
122
123         rospy.loginfo("Initializing MPC...")
124
125         #Import settings
126         labels = np.genfromtxt(self.mpcParameter_path, delimiter=',', usecols=0, dtype=str)
127         raw_data = np.genfromtxt(self.mpcParameter_path, delimiter=',')[1:]
128         data = {label: row for label, row in zip(labels, raw_data)}
129
130         weights = [data['Qacc'], data['Qalph'], data['Qd'], data['Qteta'], data['Qvref'], data
131                     ['Qgammadot'], data['Qslack']]
132
133         self.boundaries = [data['vmin'], data['vmax'], data['wmin'], data['wmax'], data['accmin'],
134                             data['accmax'], data['alphmin'], data['alphmax']]#, data['alphmin'], data['alphmax']]
135
136         #Solver parameters
137         dae_solver = 'rk'
138         dae_opts = {'tf': self.ts_MPC, 'simplify': True, 'number_of_finite_elements': 1}
139         ocp_solver = 'ipopt'
140         ocp_opts = { "ipopt.sb": "yes", "ipopt.print_level": 0, "print_time": False,
141                     "ipopt.linear_solver": "ma27", "ipopt.warm_start_init_point": "yes",
142                     "ipopt.warm_start_bound_push": 1e-9, "ipopt.warm_start_bound_frac": 1e
143                     -9,
144                     "ipopt.warm_start_slack_bound_frac": 1e-5, "ipopt.
145                     warm_start_slack_bound_push": 1e-5,
146                     "ipopt.warm_start_mult_bound_push": 1e-5, #"ipopt.
147                     warm_start_mult_init_max": 1e6,
148                     }
149
150         #torque term
151         if data['torque_term'] == 0.0:
152             weights[-1] = 0.0
153
154         #definition of mpc
155         self.mpc = MPC_local_planner(x0 = x0[:5],
156                                     weights = weights,
157                                     horizon = self.horizon,
158                                     boundaries = self.boundaries,
159                                     dae_solver = dae_solver, dae_opts = dae_opts,
160                                     ocp_solver = ocp_solver, ocp_opts = ocp_opts,
161                                     model_parameters = self.model_parameters,
162                                     jit_on = False)
163
164     def initPF(self):

```

```

161     rospy.loginfo("Initializing path filter...")
162
163     #Import settings
164     labels = np.genfromtxt(self.mpcParameter_path, delimiter=',', usecols=0, dtype=str)
165     raw_data = np.genfromtxt(self.mpcParameter_path, delimiter=',')[1:]
166     data = {label: row for label, row in zip(labels, raw_data)}
167
168     #definition of pf
169     self.pf = path_filter(model_parameters = self.model_parameters, tuning_parameter =
data['pf_ratio'])
170
171     def odometryCallback(self,odom_msg):
172
173         self.v_odom = odom_msg.twist.twist.linear.x
174         self.w_odom = odom_msg.twist.twist.angular.z
175
176         self.odom_fbk = True
177
178     def poseCallback(self,opti_msg):
179
180         #if self.odom_fbk == True: #odom is received slower, thus, wait for it
181
182         self.X = opti_msg.pose.position.x+self.dxOpti
183         self.Y = opti_msg.pose.position.y+self.dyOpti
184         quaternion = [opti_msg.pose.orientation.x, opti_msg.pose.orientation.y, opti_msg.pose.
orientation.z, opti_msg.pose.orientation.w]
185         self.TETA = tf.transformations.euler_from_quaternion(quaternion = quaternion)[2]+np.pi
/2
186
187         self.odom_fbk = False
188
189     def cmdvelDesCallback(self,twistDes_msg):
190
191         self.vDes = twistDes_msg.linear.x
192         self.wDes = twistDes_msg.angular.z
193
194         self.cmdvelDes_fbk = True
195
196     def joyTriggerCallback(self,joy_msg):
197
198         #pathfilter
199         if joy_msg.buttons[7] == 1:                                     #"RT" button (index 7) from logitech
joy activates pf
200             self.pf_flag = True
201             rospy.loginfo("Path filter activated")
202
203         if joy_msg.buttons[6] == 1:                                     #"LT" button (index 7) from logitech
joy deactivates pf
204             self.pf_flag = False
205             rospy.loginfo("Path filter de-activated")
206
207         #mpc navigation

```

```

208     if (joy_msg.buttons[4] == 1) and (joy_msg.buttons[5] == 1):  #"RB" and "LB" buttons (
index 4,5) from logitech joy activates mpc
209         rospy.loginfo("MPC activated")
210         self.drive_mode = "mpc"
211
212     if joy_msg.buttons[0] == 1:                                     #"X" button (index 0) from
logitech joy deactivates pf
213         rospy.loginfo("MPC de-activated")
214         self.drive_mode = "joy"
215
216     #mpc torque term
217     if joy_msg.axes[5] == 1.0:                                     #"arrow-down" button (index
5) from logitech joy activates reinitialization of mpc
218         rospy.loginfo("MPC reinitialization activated")
219         self.MPC_reinit = True
220
221
222     if joy_msg.axes[5] == -1.0:                                     #"arrow-down" button (
index 5) from logitech joy activates reinitializatio of pf
223         rospy.loginfo("PF reinitialization activated")
224         self.pf_update = True
225
226 def casterState_publish(self):
227
228     #casterstate
229     caster_state_msg = caster_state()
230
231     caster_state_msg.phi_est_FL = self.est.FL.phi_est
232     caster_state_msg.phi_est_FR = self.est.FR.phi_est
233     caster_state_msg.phi_des_FL = self.est.FL.phi_des
234     caster_state_msg.phi_des_FR = self.est.FR.phi_des
235
236     caster_state_msg.gammadot_est_FL = self.est.FL.gammadot_est
237     caster_state_msg.gammadot_est_FR = self.est.FR.gammadot_est
238     caster_state_msg.gammadot_des_FL = self.est.FL.gammadot_des
239     caster_state_msg.gammadot_des_FR = self.est.FR.gammadot_des
240
241     #publish
242     self.caster_state_pub.publish(caster_state_msg)
243
244 def navigation_publish(self):
245     #mpc horizon estimation and moving reference
246     #mpc_horizon_msgs = Path()
247     moving_reference_msgs = Path()
248
249     #loc1_list = []
250     loc2_list = []
251
252     for i in range(self.mpc.N):
253         loc1 = PoseStamped()
254         #loc1.pose.position.x = self.x_estim[i]
255         #loc1.pose.position.y = self.y_estim[i]

```



```

256         #loc1_list.append(loc1)
257
258         loc2 = PoseStamped()
259         loc2.pose.position.x = self.navigation.goal_pt[0,i]
260         loc2.pose.position.y = self.navigation.goal_pt[1,i]
261         loc2_list.append(loc2)
262
263         #mpc_horizon_msgs.poses = loc1_list
264         moving_reference_msgs.poses = loc2_list
265
266         #publish
267         #self.mpc_horizon_pub.publish(mpc_horizon_msgs)
268         self.moving_reference_pub.publish(moving_reference_msgs)
269
270     def cmdVel_publish(self):
271
272         cmd_vel_msg = Twist()
273         cmd_vel_msg.linear.x = self.cmdV
274         cmd_vel_msg.angular.z = self.cmdW
275         self.cmd_vel_pub.publish(cmd_vel_msg)
276
277         #if self.drive_mode == "mpc":
278         #     cmd_velDes_msg = Twist()
279         #     cmd_velDes_msg.linear.x = self.vDes
280         #     cmd_velDes_msg.angular.z = self.wDes
281         #     self.cmd_velDes_pub.publish(cmd_velDes_msg)
282
283     def stop_robot(self):
284         #update drive_mode
285         self.drive_mode = "joy"
286
287         #Stop smoothly the robot
288         while self.v_odom > 0.05:
289
290             t_start = rospy.get_time()
291             #####
292             self.cmdV = self.v_odom-0.5* self.ts_MPC
293             self.cmdW = 0
294             self.vDes = self.cmdV
295             self.wDes = self.cmdW
296             self.cmdVel_publish()
297             #####
298             t_delta = rospy.get_time()-t_start
299
300             if t_delta < self.ts_LLC:
301                 rospy.sleep(self.ts_LLC-t_delta)
302
303         #Stop completely
304         self.cmdV = 0
305         self.cmdW = 0
306         self.vDes = self.cmdV
307         self.wDes = self.cmdW

```

```

308
309 def LLC(self, acc, alph, maxV, maxW, cycle_MPC, navigation_finished):
310
311     rospy.init_node('MPC_local_planner_LLC', anonymous=True)
312     vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=50)
313
314     #Declare variables for LLC
315     cmd_vel_msg = Twist()
316     cmdV = self.cmdV
317     cmdW = self.cmdW
318     ts_LLC = self.ts_LLC
319     boundaries = self.boundaries
320     ratio = int(np.ceil(self.ts_MPC/self.ts_LLC))
321     cycle_LLC = ratio+1
322     cycle_MPC_old = 0
323
324     #start LLC+publishing to motors
325     while navigation_finished.value == False:
326
327         # if cycle_MPC.value > cycle_MPC_old:
328         #     cycle_MPC_old = cycle_MPC.value
329         #     cycle_LLC = 0
330
331         if cycle_MPC.value >= 1:# cycle_MPC_old:#cycle_LLC < ratio:#3*cycle_MPC.value >
cycle_LLC and 3*(cycle_MPC.value-1) <= cycle_LLC :#
332
333             #accel = acc.value
334             #alpha = alph.value
335             #cycle_MPC_old = cycle_MPC.value
336
337             #for i in range(ratio):
338
339                 t_start = rospy.get_time()
340
341                 #####
342                 #Converting optimal accelerations to velocities
343                 cmdV = cmdV + ts_LLC*acc.value
344                 cmdW = cmdW + ts_LLC*alph.value
345                 cmdV = np.clip(cmdV, boundaries[0], boundaries[1])#-abs(maxV.value), abs(maxV.
value)) #limit cmdV#
346                 cmdW = np.clip(cmdW, boundaries[2], boundaries[3])#-abs(maxW.value), abs(maxW.
value)) #limit cmdW#
347                 #cmdV = maxV.value
348                 #cmdW = maxW.value
349                 rospy.loginfo("LLC --> acc: %.2f, alph: %.2f, cmdV: %.2f, cmdW: %.2f", acc.
value, alph.value, cmdV, cmdW)
350
351                 #Estimating caster state
352                 self.est.estimate( t_start = rospy.get_time(), t_end = rospy.get_time()+
ts_LLC,
353
                                     vel_odom = [self.v_odom, self.w_odom], vel_des = [cmdV
, cmdW])

```

```

354         cmd_vel_msg.linear.x = cmdV
355         cmd_vel_msg.angular.z = cmdW
356         vel_pub.publish(cmd_vel_msg)
357         #####
358
359
360         t_delta = rospy.get_time()-t_start
361
362         if t_delta < ts_LLC:
363             rospy.sleep(ts_LLC-t_delta)
364         else:
365             rospy.loginfo("LLC too slow: %.2f instead of %.2f",t_delta*1e3, ts_LLC*1e3
366     )
367
368     def joy_navigate(self):
369
370         rospy.loginfo_once("Joy navigation activated")
371
372         #checking if feedback received
373         while True:
374
375             if (self.odom_fbk == True) and (self.cmdvelDes_fbk == True):
376
377                 #Reset flags
378                 self.odom_fbk = False
379                 self.cmdvelDes_fbk = False
380
381                 #Continue
382                 break
383
384             #rospy.loginfo("Waiting for feedback...")
385
386             #Estimating caster state
387             self.est.estimate( t_start = rospy.get_time(), t_end = rospy.get_time()+self.ts_LLC,
388                             vel_odom = [self.v_odom, self.w_odom], vel_des = [self.vDes, self.
389 wDes])
390
391             #Filter velocity commands
392             if self.pf_flag == True:
393
394                 if self.pf_update == True:
395                     self.pf_update = False
396                     self.initPF()
397
398                 self.pf.filter( phi_est          = [self.est.FL.phi_est, self.est.FR.phi_est],
399                               phi_des           = [self.est.FL.phi_des, self.est.FR.phi_des],
400                               gammadot_est      = [self.est.FL.gammadot_est, self.est.FR.
401 gammadot_est],
402                               gammadot_des      = [self.est.FL.gammadot_des, self.est.FR.
403 gammadot_des])
404
405                 self.cmdV = self.pf.FR.v_filt

```

```

402         self.cmdW = self.pf.FR.w_filt
403
404     elif self.pf_flag == False:
405
406         self.cmdV = self.vDes
407         self.cmdW = self.wDes
408
409     #Defining new state
410     self.x = [self.X, self.Y, self.TETA,
411              self.v_odom, self.w_odom,
412              self.est.FL.phi_est, self.est.FR.phi_est]
413
414     #Publish information
415     self.cmdVel_publish()
416     self.casterState_publish()
417
418     def mpc_navigate(self):
419
420         rospy.loginfo_once("MPC navigation activated")
421
422         #####Re-initialize pf#####
423         if (self.pf_flag == True):
424             self.pf_update = False
425             self.initPF()
426
427         #####Re-initialize MPC#####
428         if self.MPC_reinit == True:
429             self.MPC_reinit = False
430             self.initMPC()
431
432         #####Initialize navigation#####
433         self.x = [ self.X, self.Y, self.TETA,
434                  self.v_odom, self.w_odom]#,
435                  #self.est.FL.phi_est, self.est.FR.phi_est]
436
437         self.navigation = navigation_local_planner(path_filename = self.trajectory_path,
438                                                    tolerance = self.tolerance,
439                                                    initial_states = self.x,
440                                                    N = self.horizon[1], ts = self.ts_MPC)
441
442         #####MPC running#####
443
444         rospy.loginfo("Starting MPC navigation...")
445
446         #declare variables for processes
447         acc = mp.Value('d',0.0)
448         alph = mp.Value('d',0.0)
449         maxV = mp.Value('d',0.0)
450         maxW = mp.Value('d',0.0)
451         cycle_MPC = mp.Value('i',0)
452         #cycle_tref = mp.Value('i',0)
453         #x = mp.Array('d',self.x)

```

```

454     #p = mp.Array('d',np.zeros((4*(self.horizon[1]+1))))
455     navigation_finished = mp.Value('b',False)
456
457     #declare processes
458     #process_ref = mp.Process(target=self.moving_reference, args = (x,p,
navigation_finished,cycle_tref))    #declare moving reference process
459     process_LLC = mp.Process(target=self.LLC, args = (acc,alpha,maxV,maxW,cycle_MPC,
navigation_finished))                #declare HLC process
460     process_LLC.start()
461
462     #start LLC+publishing to motors
463     t_navigation_start = rospy.get_time()
464
465     while True:
466
467         t_start = rospy.get_time()
468
469         #####
470         #update states (for processes, for processes, otherwise in separate thread)
471         x = [    self.X, self.Y, self.TETA,
472                self.v_odom, self.w_odom],#,#,cmdV,cmdW,#
473                #self.est.FL.phi_est, self.est.FR.phi_est]
474
475         #Updating goal point
476         self.navigation.navigation_update(states = x, time = rospy.get_time()-
t_navigation_start)
477
478         #check if robot needs to be stopped
479         if (self.navigation.finished == True) or (self.drive_mode == "joy"):
480             navigation_finished.value = True #stops HLC process
481             self.stop_robot()
482             break
483
484         t_nav = rospy.get_time()-t_start
485
486         #Finding optimal accelerations
487
488         [acc.value , alpha.value],[maxV.value, maxW.value] = self.mpc.nlp_solver(x0 = x, p
= self.navigation.goal_pt) #[self.x_estim, self.y_estim]
489         rospy.loginfo("MPC --> acc: %.2f, alph: %.2f", acc.value, alpha.value)
490
491         #update cycle flag
492         cycle_MPC.value += 1
493
494         #rate.sleep()
495
496         #####
497
498         t_delta = rospy.get_time()-t_start
499
500         if t_delta < self.ts_MPC:
501             rospy.sleep(self.ts_MPC-t_delta)

```

```

502         else:
503             #self.drive_mode = "joy"
504             rospy.loginfo("MPC too slow: %.2f instead of %.2f --> nav: %.2f and mpc: %.2f"
, t_delta*1e3, self.ts_MPC*1e3, t_nav*1e3, (t_delta-t_nav)*1e3)
505
506
507
508         #Continue joy navigation
509         rospy.loginfo("MPC navigation de-activated")
510
511     def choose_drive_mode(self):
512
513         while not rospy.is_shutdown():
514
515             #checking if MPC navigation activated
516             if self.drive_mode == "mpc":
517                 self.mpc_navigate()
518             elif self.drive_mode == "joy":
519                 self.joy_navigate()
520
521 if __name__ == '__main__':
522
523     #Get parameters from launch file
524     trajectory_path = rospy.get_param('/local_planner/trajectory_file')
525     mpcParam_path = rospy.get_param('/local_planner/MPC_parameters')
526
527
528     #Initial state
529     x0 = [0,0,0,0,0,0,0]
530
531     #Horizon for MPC
532     T = 2.0           #time horizon of MPC
533     N = 40            #number of control intervals of MPC
534     ts = T/N          #step size of MPC
535     f_LLC = 50.0      #frequency of LLC (low level controller)
536
537
538     #Motion planner parameters
539     tolerance = 0.2
540
541     #Model parameters (all distances assume that the robot's origin is the axis of DD wheels)
542     dxCaster = 0.241212           #X-distance from origin to front caster wheel (m)
543     #0.3065 (devkit front)       0.360860 (AS-rear)
544     dyCaster = 0.159              #Y-distance from origin to front caster wheel (m)
545     #0.12 (devkit front)         0.0614 (AS-rear)
546     hCaster = 0.0611              #Overhang of caster wheel front (m)
547     #0.038 (devkit front)        0.0449 (AS-rear)
548     rCaster = 0.040               #Radius of front caster wheels
549     #0.0375 (devkit front)       0.025 (AS-rear)
550     dyDrive = 0.183              #Distance to left drive wheel
551
552     #Local planner class

```

```
549     local_planner_ROS(      x0 = x0,  
550                           horizon = [T,N],  
551                           model_parameters = [dxCaster,dyCaster, hCaster, rCaster, dyDrive],  
552                           f_LLC = f_LLC,tolerance = tolerance,  
553                           trajectory_path = trajectory_path, mpcParameter_path =  
                           mpcParam_path)
```

Listing B.1: Class for local planner

## B.2 MPC local planner

Defines the a MPC with the respective formulations of the plant model and the OCP.

```

1 from casadi import*
2 import numpy as np
3 import time
4
5 class MPC_local_planner():
6
7
8     def __init__(self, x0, horizon, weights, boundaries, dae_solver, dae_opts, ocp_solver,
9         ocp_opts, model_parameters, jit_on):
10
11         #Horizon and states
12         self.T = horizon[0]           #time horizon
13         self.N = horizon[1]           #control horizon
14         self.ts = self.T/self.N       #step size
15         self.nx = 7                   #number of states in plant model
16         self.nu = 2                   #number of inputs in plant model
17         self.ns = 2                   #number of slack variables in for safety
18
19         #OCP parameters
20         self.Qacc = weights[0]         #weight for XXXXX
21         self.Qalph = weights[1]       #weight for XXXXX
22         self.Qd = weights[2]          #weight for XXXXX
23         self.Qteta = weights[3]       #weight for XXXXX
24         self.Qvref = weights[4]       #weight for XXXXX
25         self.Qgammadot = weights[5]   #weight for XXXXX
26         self.Qslack = weights[6]
27
28         self.vmin = boundaries[0]      #minimum longitudinal velocity [m/s]
29         self.vmax = boundaries[1]      #maximum longitudinal velocity [m/s]
30         self.wmin = boundaries[2]      #minimum rotational velocity [rad/s]
31         self.wmax = boundaries[3]      #ocp_optsmaximum rotational velocity [rad/s]
32         self.accmin = boundaries[4]
33         self.accmax = boundaries[5]
34         self.alphmin = boundaries[6]
35         self.alphmax = boundaries[7]
36
37         #Solver parameters
38         self.dae_solver = dae_solver   #solver for ode
39         self.dae_opts = dae_opts       #options for ode solver
40         self.ocp_solver = ocp_solver   #solver for ocp
41         self.ocp_opts = ocp_opts       #options for ocp solver
42
43
44         #Model parameters
45         self.dxCaster = model_parameters[0] #X-distance from origin to front caster wheel
46         (m)

```



```

46     self.dyCaster = model_parameters[1]      #Y-distance from origin to front caster wheel
(m)
47     self.hCaster = model_parameters[2]      #Overhang of caster wheel front (m)S
48     self.rCaster = model_parameters[3]      #Radius of front caster wheels
49     self.dyDrive = model_parameters[8]      #Y-distance from origin to left driven wheel
50
51     #others
52     self.numb = 1e-3                        #fix for square root jacobian in cost
function
53     self.jit_on = jit_on                    #compilation method (if false C code
generated and compiled)
54
55     #functions
56     self.MPC_definition(x0 = x0)            #function to define OCP PROBLEM (
defines function "solver")
57
58
59 def MPC_definition(self, x0):
60
61     #####PLANT MODEL#####
62     #Variables
63     x = SX.sym('x',self.nx)
64     u = SX.sym('u', self.nu)
65
66     #Equations x = 'X', 'Y', 'TETA', 'V', 'W'
67     ode = SX.zeros(self.nx)
68     ode[0] = x[3]*cos(x[2])
69     ode[1] = x[3]*sin(x[2])
70     ode[2] = x[4]
71     ode[3] = u[0]
72     ode[4] = u[1]
73     ode[5] = -1/(self.hCaster)*((x[3]-x[4]*self.dyCaster)*sin(x[5])-x[4]*self.dxCaster*cos
(x[5]))
74     ode[6] = -1/(self.hCaster)*((x[3]+x[4]*self.dyCaster)*sin(x[6])-x[4]*self.dxCaster*cos
(x[6]))
75
76     #define integrator
77     dae = {'x': x, 'ode': ode, 'p': u}
78     plant_model = integrator('intg', self.dae_solver, dae, self.dae_opts)
79
80     #####FORWARD INTEGRATOR#####
81     # #Variables
82     # u_opt = SX.sym('u_opt',2)
83
84     # #Equations x = 'X', 'Y', 'TETA', 'V', 'W'
85     # ode_f = SX.zeros(2)
86     # ode_f[0] = u_opt[0]
87     # ode_f[1] = u_opt[1]
88
89     # #define integrator
90     # dae = {'x': u_opt, 'ode': ode_f, }
91     # self.forward_integr = integrator('intg', self.dae_solver, dae, self.dae_opts)

```

```

92
93 #####NLP#####
94
95 # Start with an empty NLP
96
97 w = []
98 lbw = []
99 ubw = []
100
101 g = []
102 lbg = []
103 ubg = []
104
105 J = 0
106
107 #Optimization
108 x = SX.sym("x", self.nx)
109 par = SX.sym("param",4,self.N+1) #parameter
110
111 #x = SX.sym('x', nx)
112 w += [x]
113 lbw += [x0]
114 ubw += [x0]
115
116 for k in range(self.N):      #Optimization problem for a horizon (w is a optimization
variable--> "Multiple shooting")
117
118     # New NLP variable for the control
119     u = SX.sym('u', self.nu)
120     w += [u]
121     lbw += [DM([self.accmin, self.alphamin])]#[DM([-inf, -inf])] #
122     ubw += [DM([self.accmax, self.alphmax])]#[DM([inf, inf])] #
123
124     # Integrate until the end of the interval
125     xn_dae = plant_model(x0=x, p=u)['xf']
126
127     # New NLP variable for state at end of interval
128     xn = SX.sym('x', self.nx)
129     w += [xn]
130     lbw += [DM([-inf, -inf, -inf, self.vmin, self.wmin, -inf, -inf])]
131     ubw += [DM([inf, inf, inf, self.vmax, self.wmax, inf, inf])]
132
133     # New NLP variable for slack variable in safety constraints
134     s = SX.sym('s',self.ns)
135     w += [s]
136     lbw += [DM([0, 0])]
137     ubw += [DM([inf, inf])]
138
139     # Add equality constraint
140     g += [xn_dae - xn]
141     lbg += [DM.zeros(self.nx)]
142     ubg += [DM.zeros(self.nx)]

```

```

143
144
145     # # Add input constraints
146     g += [u[0]-self.dyDrive/2*u[1], u[0]+self.dyDrive/2*u[1]]
147     lbq += [DM([self.accmin]), DM([self.accmin])]
148     ubq += [DM([self.accmax]), DM([self.accmax])]
149
150     #Add safety constraints
151     a0 = 3.76454364e-1
152     a1 = -1.09537003e-16
153     a2 = 5.34941652e-1
154     xt1 = sqrt(2)/2*(x[4]-x[3])
155     yt1 = sqrt(2)/2*(x[4]+x[3])
156     xt2 = sqrt(2)/2*(x[4]+x[3])
157     yt2 = sqrt(2)/2*(-x[4]+x[3])
158     ft1 = a0*xt1*xt1+a1*xt1+a2
159     ft2 = a0*xt2*xt2-a1*xt2+a2
160     g += [(yt1-ft1)-s[0], (yt2-ft2)-s[1]]#[(yt1-ft1), (yt2-ft2)]
161     lbq += [DM([-inf, -inf])]
162     ubq += [DM([0, 0])]
163
164     #####
165     #Cost function
166     ref_error = x[:4]-par[:,k]
167
168     Vref = x[3]*par[3,k]
169     Wref = x[4]*0
170     V = x[3]
171     W = x[4]
172     phi_L = x[5]
173     phi_R = x[6]
174
175     gammadot_ss_L = 1/self.rCaster*sqrt((Vref-Wref*self.dyCaster)**2+(Wref*self.
176     dxCaster)**2+self.numb)
177     gammadot_ss_R = 1/self.rCaster*sqrt((Vref+Wref*self.dyCaster)**2+(Wref*self.
178     dxCaster)**2+self.numb)
179     gammadot_L = 1/self.rCaster*((V-W*self.dyCaster)*cos(phi_L)+W*self.dxCaster*
180     sin(phi_L))
181     gammadot_R = 1/self.rCaster*((V+W*self.dyCaster)*cos(phi_R)+W*self.dxCaster*
182     sin(phi_R))
183
184     ref_error_3 = vertcat(gammadot_ss_L-gammadot_L, gammadot_ss_R-gammadot_R)
185
186     J += mtimes(mtimes(ref_error.T, diag(vertcat(self.Qd, self.Qd, self.Qteta, self.
187     Qvref))),ref_error)
188     J += mtimes(mtimes(ref_error_3.T, diag(vertcat(self.Qgammadot, self.Qgammadot))),
189     ref_error_3)
190     J += mtimes(mtimes(u.T, diag(vertcat(self.Qacc, self.Qalph))), u)
191     J += mtimes(s.T, vertcat(self.Qslack, self.Qslack))#mtimes(mtimes(s.T, diag(
192     vertcat(self.Qslack, self.Qslack))), s)
193
194     # Update state

```

```

188         x = xn
189
190         #Cost for last iteration
191
192         ref_error = x[:4]-par[:,self.N]
193
194         Vref = x[3]#par[3,self.N]
195         Wref = x[4]#0
196         V = x[3]
197         W = x[4]
198         phi_L = x[5]
199         phi_R = x[6]
200
201         gammadot_ss_L = 1/self.rCaster*sqrt((Vref-Wref*self.dyCaster)**2+(Wref*self.dxCaster)
202         **2+self.numb)
203         gammadot_ss_R = 1/self.rCaster*sqrt((Vref+Wref*self.dyCaster)**2+(Wref*self.dxCaster)
204         **2+self.numb)
205         gammadot_L = 1/self.rCaster*((V-W*self.dyCaster)*cos(phi_L)+W*self.dxCaster*sin(
206         phi_L))
207         gammadot_R = 1/self.rCaster*((V+W*self.dyCaster)*cos(phi_R)+W*self.dxCaster*sin(
208         phi_R))
209
210         ref_error_3 = vertcat(gammadot_ss_L-gammadot_L, gammadot_ss_R-gammadot_R)
211
212         J += mtimes(mtimes(ref_error.T, diag(vertcat(self.Qd, self.Qd, self.Qteta, self.Qvref)
213         )),ref_error)
214         J += mtimes(mtimes(ref_error_3.T, diag(vertcat(self.Qgammadot, self.Qgammadot))),
215         ref_error_3)
216
217         # Concatenate decision variables and constraint terms
218         w = vertcat(*w)
219         g = vertcat(*g)
220         self.lbg = vertcat(*lbg)
221         self.ubg = vertcat(*ubg)
222         self.lbw = vertcat(*lbw)
223         self.ubw = vertcat(*ubw)
224
225         #####COMPILER#####
226         # Declare solver
227         nlp = {'f': J, 'x': w, 'g': g, 'p': par}
228         if self.jit_on == True:
229             print("MPC initialization: JIT compiler...")
230             jit_opts = {"jit": True, "compiler": "shell", "jit_options": {"compiler": "ccache
231             gcc", "compiler_flags": ["-O3"]}}
232             self.ocp_opts.update(jit_opts)
233             self.solver = nlpsol('nlp', self.ocp_solver, nlp, self.ocp_opts)
234         else:
235             self.solver = nlpsol('nlp', self.ocp_solver, nlp, self.ocp_opts)
236             #self.solver.generate_dependencies("nlp.c")
237             #print("MPC initialization: Generating C code...")
238             #os.system("gcc -fPIC -O3 -shared nlp.c -o nlp.so")

```

```

233         #self.solver = nlpsol("solver", "ipopt", "./nlp.so")
234
235     def nlp_solver(self, x0, p):
236
237         #Define initial conditions (warm guess)
238         self.lbw[:,self.nx] = vertcat(x0)
239         self.ubw[:,self.nx] = vertcat(x0)
240
241         #nlp solve
242         t_start = time.time()
243         sol = self.solver(lbg = self.lbg, ubg = self.ubg, lbx = self.lbw, ubx = self.ubw, p =
244         p)
245         self.time_nlp = time.time()-t_start
246
247         # print( "Objective:", round(self.solver.stats()['iterations']['obj'][-1],5),
248         #         "\tIterations:",self.solver.stats()['iter_count'],
249         #         "\t\tTime(ms):", round(self.time_nlp*1000,2))
250
251         # if self.solver.stats()['success'] == False:
252         #     print("NLP failed --> Return status: ",self.solver.stats("return_status"))
253
254         #convert output
255         wopt = sol['x'].full()
256         wopt_1toN = wopt[:self.N*(self.nx+self.nu+self.ns)].reshape((self.nx+self.nu+self.ns,
257         self.N), order='F')
258         xopt = np.concatenate((wopt_1toN[:self.nx, :], wopt[self.N*(self.nx+self.nu+self.ns)
259         :]), axis=1).transpose()
260         uopt = wopt_1toN[self.nx:-self.ns, :].transpose()
261
262         # wopt = sol['x'].full()
263         # wopt_1toN = wopt[:self.N*(self.nx+self.nu)].reshape((self.nx+self.nu, self.N), order
264         # = 'F')
265         # xopt = np.concatenate((wopt_1toN[:self.nx, :], wopt[self.N*(self.nx+self.nu):]),
266         # axis=1).transpose()
267         # uopt = wopt_1toN[self.nx:, :].transpose()
268
269         #Forward integrate
270         cmdV = xopt[0,3]
271         cmdW = xopt[0,4]
272         V = cmdV + self.ts*uopt[0,0]
273         W = cmdW + self.ts*uopt[0,1]
274         #vopt = self.forward_integr(x0=uopt[0])['xf']
275         #v_opt = [vopt.full()[0][0], vopt.full()[1][0]]
276
277         return uopt[0], [V,W]#, [V,W]#v_opt###, [tuple(xopt[:,0]), tuple(xopt[:,1])]#[xopt
278         [1,3], xopt[1,4]]

```

Listing B.2: Class for MPC local planner

## B.3 Navigation local planner

The local planner follows a time dependant reference which moves along the trajectory. This class contains the functions that dictate how the reference is going to move along the path.

```

1 from casadi import *
2 import numpy as np
3
4 class navigation_local_planner():
5
6     def __init__(self, path_filename, tolerance, initial_states, N, ts):
7
8         self.traj = np.genfromtxt(path_filename, delimiter=',', names=True)    #obtain path
9         self.tolerance = tolerance                                           #tolerance
10        self.x = initial_states                                              #initial
11        self.N = N                                                            #control
12        self.ts = ts                                                         #control
13
14        self.cont_traj = 0 #counter for number of trajectories
15        self.t0_traj = 0 #timer for time spent in current trajectory
16
17        self.goal_pt = vertcat(np.ones((1,N+1))*self.x[0],
18                               np.ones((1,N+1))*self.x[1],
19                               np.ones((1,N+1))*self.x[2]) #initial goal point
20
21        self.get_trajectories() #divide path to follow into trajectories
22
23        self.finished = False #flag to know when the navigation is finished
24
25    def get_trajectories(self):
26        traj = self.traj
27        ts = self.ts
28
29        info = [traj['x'], traj['y']]
30        ind = np.where(traj["flag_goal"] == 1)[0]
31
32        #separate the information into trajectories
33        d = []
34        v_nav = traj['v_nav'][ind]
35        for k in range(len(info)):
36            c = []
37            for i in range(len(ind)-1):
38                b = []
39                for j in range(ind[i], ind[i+1]+1): b.append(info[k][j])
40                c.append(b)
41
42            if ind[-1] < len(info[k])-1:

```

```

43         b = []
44         for i in range(ind[-1], len(info[k])):
45             b.append(info[k][i])
46             c.append(b)
47         d.append(c)
48
49     #get distance and scattering of each trajectory
50     tf = [0]
51     distT = []
52     tT = []
53     scT = []
54     n_info = len(d)
55     n_traj = len(d[0])
56     for i in range(n_traj):
57         n_traj_points = len(d[0][i])
58         distance = []
59         tf = [0]
60         scatter_chk = []
61
62         for k in range(n_traj_points-1):
63             distance.append(np.sqrt(((d[0][i][k+1]-d[0][i][k])**2)+((d[1][i][k+1]-d[1][i][k])**2)))
64             tf.append(tf[-1]+distance[-1]/v_nav[i])
65             scatter_chk.append(int(np.ceil(tf[-1]/ts)))
66
67         distT.append(distance)
68         scT.append(scatter_chk)
69         tT.append(tf)
70
71     #get checkpoints of each trajectory
72     d_type = np.dtype([('t', np.float64), ('x', np.float64), ('y', np.float64), ('teta', np.float64)])
73     trajectories = []
74     for i in range(n_traj):
75         n_traj_points = len(d[0][i])
76         checkpoints = []
77         chk = []
78         for k in range(n_traj_points-1):
79             chk_t = np.linspace(tT[i][k], tT[i][k+1], scT[i][k]+2)
80             chk_X = np.linspace(d[0][i][k], d[0][i][k+1], scT[i][k]+2)
81             chk_Y = np.linspace(d[1][i][k], d[1][i][k+1], scT[i][k]+2)
82             angle = atan((d[1][i][k+1]-d[1][i][k])/(d[0][i][k+1]-d[0][i][k]))
83
84             if angle == 0 and d[0][i][k]<d[0][i][k+1]: angle = 0
85             if angle == 0 and d[0][i][k]>d[0][i][k+1]: angle = np.pi
86             if np.isnan(angle) and d[1][i][k]<d[1][i][k+1] and d[0][i][k]==d[0][i][k+1]:
87                 angle = np.pi/2
88             if np.isnan(angle) and d[1][i][k]>d[1][i][k+1] and d[0][i][k]==d[0][i][k+1]:
89                 angle = -np.pi/2
90
91             chk_teta = np.ones(scT[i][k]+2)*angle

```

```

91         for z in range(scT[i][k]+2): chk.append((chk_t[z], chk_X[z], chk_Y[z],
chk_teta[z]))
92
93         checkpoints.append(np.array(chk,dtype=d_type))
94
95         trajectories.append(checkpoints)
96
97         self.trajectories = trajectories
98         self.ind = ind
99
100     def navigation_update(self,states,time):
101         self.t = time
102         self.x = states
103         self.check_status()
104         if self.finished == False:
105             self.get_goal()
106
107     def check_status(self):
108         if self.check_end():
109             self.cont_traj += 1
110             self.t0_traj = self.t
111             if self.cont_traj > len(self.trajectories)-1:
112                 self.finished = True
113             else: print("--Trajectory: " + str(self.cont_traj+1)+", References velocity: ",
str(self.traj['v_nav'][self.ind[self.cont_traj]])+"m/s")
114
115     def check_end(self):
116         posX = self.x[0]
117         posY = self.x[1]
118         objX = self.trajectories[self.cont_traj][0]['x'][-1]
119         objY = self.trajectories[self.cont_traj][0]['y'][-1]
120         distance = np.sqrt(((posX-objX)**2)+((posY-objY)**2))
121         #print("Dist", distance)
122         if distance < self.tolerance: return True
123         else: return False
124
125     def get_goal(self):
126         self.v_ref = self.traj['v_nav'][self.ind[self.cont_traj]]
127         t = self.t-self.t0_traj
128         checkpoints = self.trajectories[self.cont_traj][0]
129         N = self.N
130         ts = self.ts
131
132         objX_old = float(self.goal_pt[0,1])
133         objY_old = float(self.goal_pt[1,1])
134         objTeta_old = float(self.goal_pt[2,1])
135
136         objX = np.interp(t+N*ts, list(checkpoints['t']),list(checkpoints['x']))
137         objY = np.interp(t+N*ts, list(checkpoints['t']),list(checkpoints['y']))
138         objTeta = np.interp(t+N*ts, list(checkpoints['t']),list(checkpoints['teta']))
139

```



```

140     # if (objX == checkpoints['x'][-1]) and (objY == checkpoints['y'][-1]): #if the time
    based reference has got to the end of the section, move its origin to robot's location
141     #     objX = tuple(np.linspace(self.x[0], objX, N+1))
142     #     objY = tuple(np.linspace(self.x[1], objY, N+1))
143     #     objTeta = tuple(np.linspace(objTeta_old, objTeta, N+1))
144     #     objVel = tuple(np.ones(N+1)*self.v_ref)
145     # else: #else normal time based reference
146     objX = tuple(np.linspace(objX_old, objX, N+1))
147     objY = tuple(np.linspace(objY_old, objY, N+1))
148     objTeta = tuple(np.linspace(objTeta_old, objTeta, N+1))
149     objVel = tuple(np.ones(N+1)*self.v_ref)
150
151     self.goal_pt = horzcat(objX,objY,objTeta,objVel).T
152     self.objX = objX
153     self.objY = objY

```

Listing B.3: Class for navigation local planner

## B.4 Caster wheel estimator

Contains the ODE equations to estimate the caster wheel states (rotation angle,  $\varphi$ , and rolling speed,  $\dot{\gamma}$ ). It also calculates the steady states of those two variables ( $\varphi_{ss}, \dot{\gamma}_{ss}$ ).

```

1 from casadi import*
2 import numpy as np
3
4 class caster_estimator():
5
6     def __init__(self, initial_state, f_controller, dae_solver, dae_opts, model_parameters):
7
8         self.v_des = 0                #DESIRED longitudinal velocity
9         self.w_des = 0                #DESIRED rotational velocity
10
11        self.v_odom = 0                #ODOMETRY longitudinal velocity
12        self.w_odom = 0                #ODOMETRY rotational velocity
13
14        self.phi_est = initial_state[0]    #initial ESTIMATED caster-wheel rotation
15        self.gammadot_est = initial_state[1] #initial ESTIMATED caster-wheel rolling
16        self.angle
17
18        self.phi_des = 0                #DESIRED caster-wheel rotation angle
19        self.gammadot_des = 0           #DESIRED caster-wheel rolling angle
20
21        self.t_start = 0                #start time for integration
22        self.t_end = 0                 #finish time for integration
23        self.ts = 1/f_controller        #step size of integration
24
25        self.dxCaster = model_parameters[0] #X-distance from origin to front caster wheel
26        self.dyCaster = model_parameters[1] #Y-distance from origin to front caster wheel
27        self.hCaster = model_parameters[2]  #Overhang of caster wheel front (m)S
28        self.rCaster = model_parameters[3]  #Radius of front caster wheels
29
30        self.dae_solver = dae_solver      #solver for ode
31        self.dae_opts = dae_opts          #options for ode solver
32
33        self.ODE_definition()              #function to define INTEGRATOR
34
35    def ODE_definition(self):
36
37        #Variables
38        x = SX.sym('x',1)
39        u = SX.sym('u',2)
40
41        #Equations --> x = ['phiL', 'phiR', 'gammaL', 'gammaR']
42        ode = SX.zeros(1)
43        ode[0] = -1/(self.hCaster)*((u[0]-u[1]*self.dyCaster)*sin(x[0])-u[1]*self.dxCaster*cos
44        (x[0]))

```

```

43
44
45     #Settings
46     dae = {'x': x, 'ode': ode, 'p': u}
47
48     intg = integrator('intg', self.dae_solver, dae, self.dae_opts)
49     x_next = intg(x0=x, p=u)['xf']
50     self.F = Function('F', [x,u], [x_next], ['x','u'], ['x_next'])
51
52     def TwistToCasterDesired(self, v_des, w_des):
53
54         #desired caster wheel rotation angles
55         self.phi_des = arctan2((w_des*self.dxCaster),(v_des-w_des*self.dyCaster))
56
57         #desired caster wheel rolling angles
58         self.gammadot_des = 1/self.rCaster*np.sqrt((v_des-w_des*self.dyCaster)**2+(w_des*self.
59         dxCaster)**2)
60
61     def CasterStateEstimate(self, v_odom, w_odom):
62
63         U = [v_odom, w_odom]
64         phi_est = self.phi_est
65
66         #estimate rotation angle (integration)
67         for T in np.arange(self.t_start, self.t_end, self.ts):
68             phi_est = self.F(phi_est, U)
69             self.phi_est = float(phi_est.full())
70
71         #estimate rolling angle (analytical equations)
72         self.gammadot_est = 1/(self.rCaster)*((U[0]-U[1]*self.dyCaster)*cos(phi_est[0])+U[1]*
73         self.dxCaster*sin(phi_est[0]))
74
75     def estimate(self, t_start, t_end, vel_des, vel_odom):
76
77         #safe variables
78         self.t_start = t_start
79         self.t_end = t_end
80
81         #convert desired command velocities to caster wheel angles
82         self.TwistToCasterDesired(v_des = vel_des[0] , w_des = vel_des[1])
83
84         #estimate caster wheel rotation and rolling angles
85         self.CasterStateEstimate(v_odom = vel_odom[0], w_odom = vel_odom[1])
86
87     class estimator():
88
89         def __init__(self, initial_state, f_controller, dae_solver, dae_opts, model_parameters):
90
91             #Model parameters
92             dxCaster          = model_parameters[0]          #X-distance from origin to front caster
93             wheel (m)

```

```

91     dyCaster      = model_parameters[1]      #Y-distance from origin to front caster
wheel (m)
92     hCaster      = model_parameters[2]      #Overhang of caster wheel front (m)S
93     rCaster      = model_parameters[3]      #Radius of front caster wheels
94     dxCasterRear = model_parameters[4]
95     dyCasterRear = model_parameters[5]
96     hCasterRear  = model_parameters[6]
97     rCasterRear  = model_parameters[7]
98
99     #initialize a path filter for each caster-wheel
100    self.FL = caster_estimator(initial_state = initial_state[:2],
101                                f_controller = f_controller,
102                                dae_solver  = dae_solver,
103                                dae_opts    = dae_opts,
104                                model_parameters = [dxCaster,dyCaster,hCaster,rCaster])
105
106    self.FR = caster_estimator(initial_state = initial_state[2:],
107                                f_controller = f_controller,
108                                dae_solver  = dae_solver,
109                                dae_opts    = dae_opts,
110                                model_parameters = [dxCaster,-dyCaster,hCaster,rCaster])
111
112    self.BL = caster_estimator(initial_state = initial_state[:2],
113                                f_controller = f_controller,
114                                dae_solver  = dae_solver,
115                                dae_opts    = dae_opts,
116                                model_parameters = [-dxCasterRear,dyCasterRear,hCasterRear,
rCasterRear])
117
118    self.BR = caster_estimator(initial_state = initial_state[2:],
119                                f_controller = f_controller,
120                                dae_solver  = dae_solver,
121                                dae_opts    = dae_opts,
122                                model_parameters = [-dxCasterRear,-dyCasterRear,hCasterRear,
rCasterRear])
123
124    def estimate(self,t_start, t_end, vel_des, vel_odom):
125
126        #estimate states for all caster_wheels
127        self.FL.estimate(t_start = t_start, t_end = t_end, vel_odom = vel_odom, vel_des =
vel_des)
128        self.FR.estimate(t_start = t_start, t_end = t_end, vel_odom = vel_odom, vel_des =
vel_des)
129        self.BL.estimate(t_start = t_start, t_end = t_end, vel_odom = vel_odom, vel_des =
vel_des)
130        self.BR.estimate(t_start = t_start, t_end = t_end, vel_odom = vel_odom, vel_des =
vel_des)

```

Listing B.4: Class for caster wheel observer

## B.5 Caster wheel based Path Filter

Implements in Python an the extended version of the PF proposed in paper [5] as explained in Section 3.8.

```

1 from casadi import*
2 import numpy as np
3
4 class caster_path_filter():
5
6
7     def __init__(self, model_parameters, tuning_parameter):
8         #Tuning parameters
9         self.tuning_parameter = tuning_parameter
10
11         #Model parameters
12         self.dxCaster = model_parameters[0]      #X-distance from origin to front caster wheel
13         self.dyCaster = model_parameters[1]      #Y-distance from origin to front caster wheel
14         self.hCaster = model_parameters[2]      #Overhang of caster wheel front (m)S
15         self.rCaster = model_parameters[3]      #Radius of front caster wheels
16
17     def CasterStateFilter(self, phi_est, gammadot_est, phi_des, gammadot_des):
18
19         #conversion to -pi,pi
20         deltaPhi_raw = phi_des-phi_est
21         n = np.floor((deltaPhi_raw-np.pi)/(2*np.pi))
22         deltaPhi = deltaPhi_raw-2*(n+1)*np.pi
23
24         #filtering term
25         k = min(1.0, abs(gammadot_est/(gammadot_des*self.tuning_parameter)))
26         self.correction_term = k*deltaPhi
27
28         #filtered caster wheel rotation angle
29         self.phi_filt = phi_est+self.correction_term
30
31         #filtered caster wheel rolling angle
32         self.gammadot_filt = gammadot_des
33
34
35
36     def CasterStatetoTwist(self):
37         numb = 1e-3
38
39         #conversion to -pi,pi
40         n = np.floor((self.phi_filt-np.pi)/(2*np.pi))
41         phi_helper = self.phi_filt-2*(n+1)*np.pi
42
43         #calculate w_filt
44         self.w_filt = self.gammadot_filt*sin(self.phi_filt)*self.rCaster/self.dxCaster
45

```

```

46     #calculate v_filt
47     if phi_helper >= -np.pi+numb and phi_helper <= -numb:
48         ycc = self.dyCaster-self.dxCaster*tan(phi_helper+np.pi/2)
49         self.v_filt = self.w_filt*ycc
50
51     elif phi_helper > numb and phi_helper < np.pi-numb:
52         ycc = self.dyCaster-self.dxCaster*tan(phi_helper-np.pi/2)
53         self.v_filt = self.w_filt*ycc
54
55     elif phi_helper > -numb and phi_helper < numb:
56         self.v_filt = self.gammadot_filt*self.rCaster
57
58     else:
59         self.v_filt = -self.gammadot_filt*self.rCaster
60
61     #gammadot_filt*cos(phi_filt)*self.rCaster*(1+tan(phi_filt)*self.dyCaster/self.dxCaster
62     )
63
64 def filter(self, phi_est, phi_des, gammadot_est, gammadot_des):
65
66     #filter caster wheel rotation and rolling angles
67     self.CasterStateFilter( phi_est = phi_est,
68                             phi_des = phi_des,
69                             gammadot_est = gammadot_est,
70                             gammadot_des = gammadot_des)
71
72     #convert filtered caster wheel rotation and rolling angles to filtered command
73     velocities
74     self.CasterStatetoTwist()
75
76 class path_filter():
77
78     def __init__(self, model_parameters, tuning_parameter):
79
80         #Model parameters
81         dxCaster = model_parameters[0]      #X-distance from origin to front caster wheel (m)
82         dyCaster = model_parameters[1]      #Y-distance from origin to front caster wheel (m)
83         hCaster  = model_parameters[2]      #Overhang of caster wheel front (m)S
84         rCaster  = model_parameters[3]      #Radius of front caster wheels
85
86         #initialize a path filter for each caster-wheel
87         self.FL = caster_path_filter(model_parameters = [dxCaster,dyCaster,hCaster,rCaster],
88                                     tuning_parameter = tuning_parameter)
89         self.FR = caster_path_filter(model_parameters = [dxCaster,-dyCaster,hCaster,rCaster],
90                                     tuning_parameter = tuning_parameter)
91
92     def filter(self, phi_est, phi_des, gammadot_est, gammadot_des):
93
94         self.FL.filter( phi_est = phi_est[0],
95                         phi_des = phi_des[0],
96                         gammadot_est = gammadot_est[0],

```

```
94         gammadot_des = gammadot_des[0])
95
96     self.FR.filter( phi_est = phi_est[1],
97                    phi_des = phi_des[1],
98                    gammadot_est = gammadot_est[1],
99                    gammadot_des = gammadot_des[1])
```

Listing B.5: Class for caster wheel based Path Filter







TRITA ITM-EX 2020:477