Universal Path Following of Wheeled Mobile Robots

Reza Oftadeh

Abstract

The global market for service robotics has application potential in a wide variety of fields, such as agriculture, elderly care, security, among others. Given that a service robot has to be able to move to a place where the service is needed, most service robots are equipped with a Wheeled Mobile Robot (WMR). However, the multitude of the kinematic structures available for those platforms and their structural differences result in significant Research and Development (R&D) costs for the wide utilization of WMRs as bases for service robots; this is one of the main reasons that the market potential for service robotics has yet to materialize. To address this problem, in this study, we develop a nonlinear *universal path-following controller*; unlike previous algorithms, this is formally reconfigurable and solves the path-following problem for all common categories of holonomic and nonholonomic WMRs, such as omnidirectional, unicycle, car-like, and all wheels steerable types, among others. Moreover, it provides a closed-form solution for high performance, bounded actuation autonomous path following and goal tracking, which is also suitable for real-time implementation.

As part of the general solution, in this study, a major focus is the path-following of nonholonomic omnidirectional mobile platforms, which are equipped with several independently steerable wheels. Such platforms provide a robust and dexterous base for mobile robots serving in a variety of applications. However, the inherent nonholonomic constraints and singular configurations give rise to several challenges in exploiting the high maneuverability of such platforms. Unlike previous motion controllers that force the robot to stay outside of bulky regions around its singular points, it is shown that the utilization of the proposed universal controller provides efficient regulation of the robot velocity in the vicinity of the singular configurations. Thus, this approach allows the robot to realize wide ranges of complex maneuvers that were previously only possible with holonomic omnidirectional robots. For such platforms, we also solve the problem of time optimal path-following with accelerations by incorporating the simplified kinematic constraints as a basis of an online “Phase Plane" switching algorithm.

The implementation phase of this study covers the mechatronics design of a four-wheel-steered mobile manipulator. It presents the robot’s mechanical structure and electrical interfaces, designs low-level software architecture based on embedded PC-based controls, and proposes a systematic solution based on MatlaB and Simulink code generation products. This approach is used for an extensive experimental evaluation of the proposed path-following controller in several types of WMRs, thereby demonstrating the performance and efficacy of the method.

Preface

This dissertation was completed for the Department of Intelligent Hydraulics and Automation (IHA) at Tampere University of Technology (TUT) from 2011 – 2015. Three years of this work was supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network and carried out within the Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation (PURESAFE) framework under REA Grant Agreement Number 264336. The last year of this project was funded by the Academy of Finland and Doctoral Program of Concurrent Mechanical Engineering (DPCME).

Reza Oftadeh
Tampere, 2015

Contents

List of Figures

List of Abbreviations

TUT	Tampere University of Technology
IHA	Department of Intelligent Hydraulics and Automation
WMR	Wheeled Mobile Robot
WMM	Wheeled Mobile Manipulator
4WS	Four Wheel Independently Steered
ROS	Robot Operating System
ICR	Instantaneous Center of Rotation
PPA	Phase-Plane Approach
3D	Three-dimensional
POSIX	Portable Operating System Interface
OS	Operating System
SSH	Secure SHell
SOA	Service Oriented Architecture
TLC	Target Language Compiler

List of Publications

This thesis consists of the following publications. The first four publications, referred to as [**PI**],...,[**PIV**], are related to the theoretical development of the study, while the last three, referred to as [**PV**],...,[**PVII**], belong to the implementation phase.

PI **Oftadeh, Reza**, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Bounded-velocity motion control of four wheel steered mobile robots." In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pp. 255-260. IEEE, 2013.

PII **Oftadeh, Reza**, Reza Ghabcheloo, and Jouni Mattila. "A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels." In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 4845-4851. IEEE, 2013.

PIII **Oftadeh, Reza**, Reza Ghabcheloo, and Jouni Mattila. "Time optimal path following with bounded velocities and accelerations for mobile robots with independently steerable wheels." In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2925-2931. IEEE, 2014.

PIV **Oftadeh, Reza**, Reza Ghabcheloo, and Jouni Mattila. "A time-optimal bounded velocity path-following controller for generic Wheeled Mobile Robots." In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 676-683. IEEE, 2015.

PV **Oftadeh, Reza**, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink." In *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on*, pp. 274-279. IEEE, 2012.

PVI **Oftadeh, Reza**, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback." In *6th IFAC Symposium on Mechatronic Systems*, pp. 663-669. IFAC, 2013.

PVII **Oftadeh, Reza**, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "System integration for real-time mobile manipulation." *International Journal of Advanced Robotic Systems* 11 (2014).

1 Introduction

The major difference concerning service robots as compared to the well-established stationary industrial robots is their large, theoretically infinite, workspace requirement. A service robot has to be able to move to a place where the service is needed. Therefore, most service robots are equipped with a mobile platform fulfilling application-specific mobility and performance requirements. The diversity of such requirements has led to multiple custom-built mobile platform solutions, even for relatively similar applications. Conversely, different types of tasks require diverse mobile robots or autonomous vehicles with different morphologies [54].

As noted by [51], the global market for service robotics has significant potential in various domains, such as agriculture, elderly care, security, among others. However, so far, this market promise has not materialized. One of the main bottlenecks preventing service robotics market success has been the high costs involved in he extensive R&D work needed for the basic functionality of custom-built mobile platforms. Nevertheless, a service robot's added value seldom comes solely from its ability to move; rather, it stems from the services it can provide.

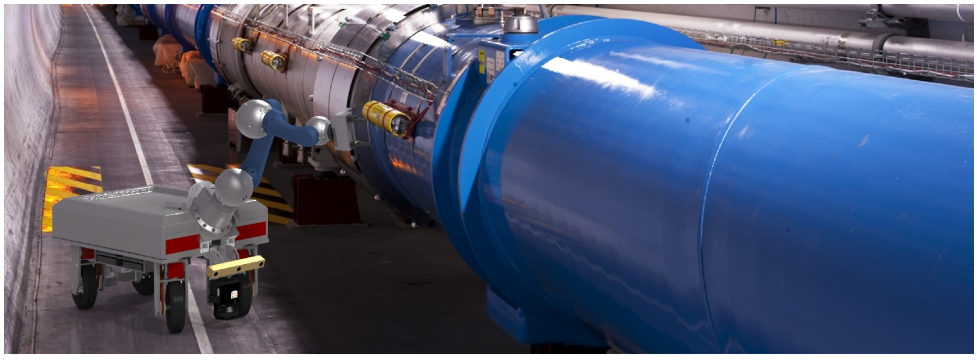


Figure 1.1: Graphical proof of concept of a *modular wheeled mobile manipulator*, created by the author for the PURESAFE project.

In this regard, there have been some notable achievements in the field of software architecture and abstraction layers for mobile robots, for example, the Robot Operating System (ROS) [60] and especially by MoveIt! [15]. In addition, modular wheels such as iMoro (Fig. 1.1) have made it possible to configure and build a Wheeled Mobile Robot(WMR) with different kinematic structure. However, the multitude of kinematic structures [55], the software and hardware architectures, and their structural differences represent the main hurdles to their wide utilization in the industry or in service robots. The kinematic structure determines the type and formation of the wheels, which in

turn defines the steering type of the platform; among many different types, differential, Ackerman (car-like), and independently steered mobile robots are the most popular.

1.1 Wheeled Mobile Robots: Definition and Classification

A conceptual definition of a general WMR with the accompanied assumptions, which represents the target platform of this thesis, is given below.

Definition 1 (WMR)**.** *The WMR considered in this study is equipped with n wheels attached to a main body called the* base*. It belongs to and possesses the minimum actuated wheels of one of those five kinematically feasible categories of WMRs, and the actuators provide velocity and position control. Each wheel can be categorized into the following types:* fixed wheels, standard steerable wheels, off-centered steerable wheels (Caster wheels), or Swedish wheels*. Furthermore, the following assumptions hold. The WMR traverses on a flat and horizontal plane. The base and the wheels are rigid, the tires are non-deformable, their contact surface with the ground can be approximated with a point, and there is no mechanical constraint for the steering of the steerable wheels, hence they are free-turn.*

In their seminal work, Campion et al. [11] formally define and classify the mobility, controllability, and nonholonomy of WMRs equipped with various types of wheels. Their work unifies the kinematic relations that arise from all wheels abiding by the rolling/ no skidding constraints in the planar case, and it describes the maneuverability by the pair: $\delta = (\delta_m, \delta_s)$, in which δ_m is the degree of mobility and δ_s is the degree of the WMR's steerability. The number of wheels, their types, and their arrangements determine δ_m, and δ_s and therefore the WMR's category. Here, δ_m represents the dimension of the tangent space of the configuration space, while δ_s corresponds to the ability to change (steer) the basis of the tangent space by means of the steering wheels. The degree of maneuverability, defined as $\delta_M = \delta_m + \delta_s$, analogous to the degrees of freedom for the mechanism, determines the total mobility of the WMR.

For each kinematic structure, various motion controllers have been designed and tested. However, each controller targets only one or two structures. The early works on the topic covering the methods developed during 1980s and early 1990s have been surveyed in [82]. The notable early methods that created the foundation for the modern approaches were carried out in [2, 48, 62, 73]; later, newer approached were developed for unicycle robots, such as [45], or the controllers based on the ICR for independently steered mobile platforms (e.g. [20]). A more thorough review of the recent works is provided in the next chapter.

Figure 1.2: iMoro: A four-wheel-steering mobile manipulator designed and fabricated at IHA/TUT.

In addition to this narrowness, there is no valid instruction or experiment on how one method can be adapted for other structures; even if such adaption is achieved,

there is no guarantee that the controller would be able to exploit all the mobility and/or maneuverability features of the new kinematic structure. In contrast, the motion controller should be robust yet computationally fast enough to comply with the real-time specifications of the designed hardware. For high-performance motion, the latter requirement is crucial.

Among the kinematic structures, interest in nonholonomic, omnidirectional mobile platforms, which utilize two or more actively steered standard wheels, has been growing exponentially. Such platforms provide a robust base for mobile robots serving in a wide range of practical fields, including service robotics [21, 29, 34], space robotics [64], agricultural applications [6, 13, 49], and others [32, 52]. These platforms, which owe their popularity to their high mobility and their use of standard wheels, provide more robustness against ground conditions [13], as well as a higher payload, compared with platforms that utilize omnidirectional wheels. Figure 1.2 depicts one example of this category of mobile robots. The early research on such robots has focused more on design [36], as well as the over-constrained nature and correction of wheel odometry errors for deadreconing localization [9, 61]. Advances in sensors and actuators, along with sensor-fusion-based algorithms for localization have solved many of these early issues [72]. The interest has now shifted to analyzing and developing more sophisticated control schemes that exploit the high maneuverability of those robots (e.g., [52, 65]).

1.2 Research Problem

WMRs are a major part of the ever growing area of Autonomous and Unmanned Ground Vehicles (AGVs and UGVs). In this industry, the everlasting demand for WMRs with higher performance and variability calls for more advanced motion controllers for these types of mobile robots. Each of the many motion controllers proposed for mobile robots targets one or two specific kinematic configuration and fails, at least without major modification, to be practically applicable for other architectures. Thus, the research problem addressed in this thesis involves the design of a motion controller for WMRs with the following features:

- Reusability: The motion controller should be easily reconfigured and adapted for different kinematic structures, such as a differential drive, Ackerman steering, and independently steerable wheels;
- High performance: The controller should provide high performance while abiding by the velocities and accelerations bounds of all of the actuators;
- Real-time capabilities: The solution should be computationally deterministic and fast enough so that its implementation abides by the hard real-time deadline; and
- Verification and testing: The controller should be practically implemented and tested on various types of WMRs;

The methodology of the thesis is to tackle the path-following problem in two stages; the first stage covers the path-following of the base divested of the wheel constraints, and the second stage maps the base control signals to that of the wheels in such way that it would be applicable to various kinematic configurations.

1.3 Overall Concept of the Proposed Method

The general concept of the implemented controller as a software module has two phases, as follows: the *Configuration phase* and the *Operational phase*. As shown in Fig 1.3, for each WMR, the configuration phase needs to be carried out only once or after any major change to the platform. In this phase, the design engineer sets the configurations, defining the type of kinematic structure, actuator bounds, geometry of the platform, dimensions, and some other configuration parameters. The software module is developed on Matlab/Simulink and C\C++ real-time services are then generated and loaded onto the embedded PC of the WMR via Ethernet. We have developed a semi-automatic solution for the code generation and loading process to ease development and maintenance.

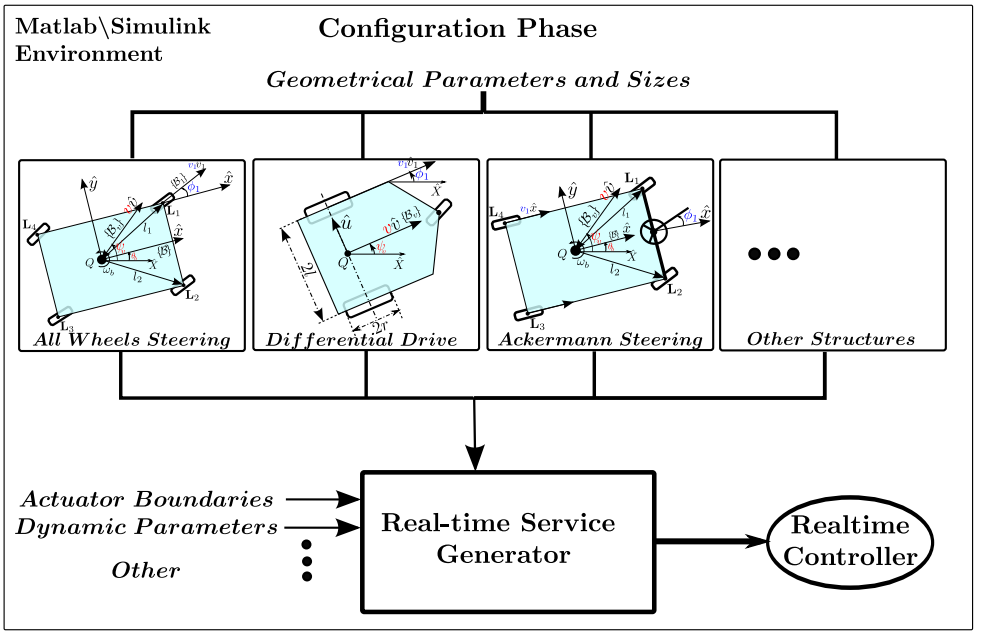


Figure 1.3: Configuration phase of the controller concept: Based on the WMR kinematic configuration, the universal path-following controller is customized (simplified), and the pertinent software module is generated.

As shown in Fig 1.4, in the operational phase, the service is run in the real-time environment; it receives the desired path and/or the target pose of the WMR and relevant obstacle data, and it generates the relevant control signals to the actuators.

Another benefit of the proposed framework is that it provides a coherent and reliable set-up for rapid prototyping of the controller on different WMRs, and tuning the controller gains in both the configuration and operational phases. This framework has been implemented on a Linux-based, real-time, service-oriented architecture using the automatic code-generation capabilities of Matlab and Simulink.

1.4 Contributions of this Study

The contributions of this paper are as follows:

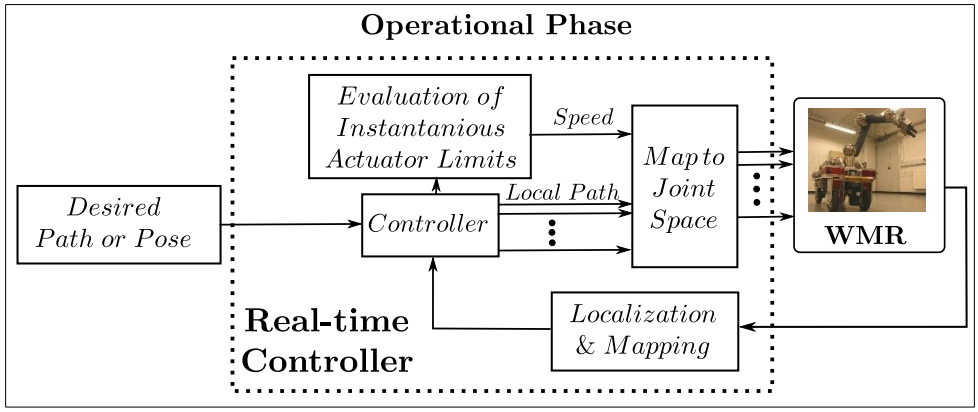


Figure 1.4: Operational phase of the Controller concept: The generated real-time controller is then integrated as a real-time service along with other operational modules.

1. It solves the path-following problem for all of the categories of WMRs, allowing their wheels to roll without skidding. To the best knowledge of the author, this is the first study that coherently solves the path-following problem with this level of generality;
2. Unlike other path-followers, in this design, the control laws and the resultant closed-loop equations of motion are *linearly proportional* to the base speed.[1] In fact, the controller acts as a feedback path planner that minimizes the Lyapunov function of errors as its corresponding cost function;
3. The kinematic constraints of all types of wheels are rigorously derived in their most general form. We derive and prove the sufficient conditions for a path-following controller that simplify the kinematic and nonholonomic constraints into explicit relations between the speed of the base and that of the wheels;
4. Based on this setup, we present a *closed-form* solution for the speed of the WMR's base, so that all the wheels' steering and driving speeds remain within their respective bounds. We show that the solution is time-optimal because it provides a bang-bang velocity profile in which at least one of the wheels runs at its maximum speed at each time step;
5. This solution allows WMRs with active steering wheels to get close to and even pass their singular configurations by regulating the speed of the WMR and the steering velocities of the wheels. Hence, this method expands the allowable configuration space of such robots and allows them to exploit their whole maneuverability;
6. Taking advantage of the previous results, we incorporate a class of minimum time trajectory generation methods commonly known as the Phase-Plane Approach (PPA), which was first introduced by [8] and [67]. For serial manipulators, the method is commonly used *offline* to derive the time-optimal trajectory for the robot's *desired path* (see [16, 22] for a review on this topic). We show that our path-following controller and the resultant differential kinematic constraints can serve as a *dynamic model* for the PPA. Moreover, we present an *online* version of the PPA to generate a minimum time trajectory for the robot's *actual path* that switches between the maximum and minimum allowable *accelerations* and bounds the velocities of both driving and steering actuators; and
7. A coherent framework has been derived for Linux-based, real-time software development of mechatronic systems including the proposed path-following controller. The solution

[1] Technically, *speed* refers to the magnitude of velocity vectors.

is a service-oriented scheme that allows for the fast and reliable rapid prototyping of the software modules; it has been extensively used for the design and implementation of iMoro control software.

This thesis is composed of two parts. The first includes an introductory section, along with a literature review and experimental results. The second part consists of seven publications that provide the theoretical and experimental details of this study.

1.5 Restrictions

The autonomous motion control of WMRs encompasses an extensive field of research that has become even more diverse due to the utilization of numerous control methods, from classical linear and nonlinear control theory to soft computing approaches, such as fuzzy and neural network schemes. Moreover, the successful practical realization of such methods requires a solid integration of the controller with other necessary modules, such as localization and perception, along with higher-level units, such as state machine or task planner components. Needless to say, each of these modules and components has its own vast proprietary research field. Thus, it is worthwhile to delineate the context and scope in which the contributions listed in the previous section are applied.

- As specified in Definition 1, the methods presented in this study are applicable to WMRs under the assumption of pure rolling and no skidding wheels;
- The motion controller proposed in this study is a coherent autonomous path-follower and bounded velocity trajectory generator. Typically, the path-following approach belongs to a decoupled control architecture, in which a path planning method provides the desired geometric path and a path-following module with a trajectory generation scheme provides actuator commands that guide the WMR toward and onto the given path;
- This study only focuses on the path-follower and trajectory generation elements. Other modules and components, such as localization, perception, or the path planner, have not been explored. However, to implement the proposed design, the details related to the execution and integration of the controller with other components are discussed, especially in papers **PV**, **PVI**, and **PVII**;
- The controller is derived based on the kinematic model (in both velocity and acceleration space) of WMRs instead of the complete dynamic model. Therefore, actuator velocities serve as control inputs. As elaborated in [74, 34.2], the main reason for this approach is that the kinematic model is simpler and does not entail the knowledge of multiple vehicle-specific parameters, which are generally hard to measure accurately. Moreover, most of the small to mid-size WMRs, particularly for indoor applications, are equipped with electrically powered servo actuators that provide inner velocity control loops. These are able to compensate, at least at certain operational regions, for unmodeled dynamics and disturbances. Finally, in the absence such servo drivers, the kinematic controller may serve as a reference input that can be integrated into a controller that accounts for the unreduced state space; and
- A localization module based on wheel odometry has been used to perform the extended experiments described in Section 3.3.

1.6 The Author's Contribution to Publications

This section briefly explains the role of the author in each of the listed publications.

Publication **PI**: The author developed the presented theories and methods. M. Aref helped with the literature review and assessing the state of the art. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PII**: The author developed the initial idea and the theoretical framework and wrote the paper. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PIII**: The author wrote the paper and developed the methods presented in it. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PIV**: The author wrote the paper and developed the methods presented in it. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PV**: The author wrote the paper and developed the methods presented in it. M. Aref helped with the literature review and assessing the sate of the art. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PVI**: The author wrote the paper and developed the methods presented in it. M. Aref was responsible for the perception system and sensors integration. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections and suggestions.

Publication **PVII**: The author wrote the paper and developed the methods presented in it. M. Aref was responsible for the perception system and sensor integration, as well as helping with the literature review. As academic supervisors, Professors Reza Ghabcheloo and Jouni Mattila, reviewed the paper and made corrections, and suggestions.

2 Review of the State of the Art

This chapter reviews the state of the art in the field of design, integration, and implementation of various motion controllers for mobile robots, particularly in the area of path-following. The first section considers various path-following schemes and their incorporation into WMRs control. The second section reviews several integration and implementation schemes of motion controllers for mobile robots.

2.1 Path-Following of WMRs

WMRs form a significant subset of AGVs and UGVs. The continuing demand for more advanced and autonomous WMRs means that more reliable and higher-performance motion controllers are needed. The multitude of motion controllers that have been proposed for mobile robots, particularly those with nonholonomic constraints, may roughly be categorized into the three following groups[63]:

- Point stabilization [57];
- Trajectory tracking [39]; and
- Path-following [31].

As mentioned above, the path following approach is used under a decoupled control architecture, where a path planner provides the desired geometric path. Then, a path-following module maneuvers the robot toward the planned path and steers it so that it follows the path indefinitely, while at the same time considering the temporal and other intrinsic constraints of the system.

Path-following algorithms are classified into several branches, including but not limited to optimal control approaches[25, 76], feedback linearizion methods[3], line of sight guidance laws[4], pure pursuit techniques[80], and vector fields methods[56] (see [71] for a thorough review). Most of these algorithms incorporate a concept that goes by many names, including "Virtual Target Point," "Carrot," and "Rabbit." In this concept, a virtual point is selected and moved along the desired path, while a tracking controller, also called a guidance controller, makes the robot follow and converge to that point. The differences among and between those branches are mainly evident in the design of the guidance controller, the method for selecting the virtual point, and the strategy for its motion along the path.

The controller developed in this thesis belongs to a category of path-followers that use the dynamic projection of the mobile robot on the desired path as the virtual point and parametrize it using the path's natural parameter. A nonlinear guidance controller is

employed to track the virtual point based on an error space projected on the path through the path's Serret–Frenet frame.

Early notable works in this field were carried out by [62] for car-like WMRs, [2] for the unicycle type, and [48] for both unicycles and WMRs with two steering wheels. However, the projection scheme of this approach, which selects the path's closest point to the robot as the virtual point, results in singularities that will in turn impose stringent initial conditions on the system and the drivable path curvatures. This drawback was overcome by using the path natural parameter as an auxiliary state and therefore deriving a control law for its progression rate along the path [69]. There have been various extensions of the original problem, covering uncertainties [44], actuator saturations [42], obstacle avoidance [45], and an extension to aerial [40] and marine vehicles [30]. Comprehensive experimental results of some of these algorithms have been provided by [7]. However, most of the recent studies on the path-following of WMRs consider only a special type of WMR, namely the unicycle type, with some exceptions [41]. Nevertheless, there is no unified solution for the path-following of WMRs.

2.1.1 Nonholonomic Omnidirectional WMRs

Aside from the general difficulties of designing a universal path-follower for WMRs, this work tackles several challenges to design motion controllers that are *specific* to this type of WMR. Here, we focus on a specific category of MWRs that are both nonholonomic and omnidirectional.

Nonholonomic, omnidirectional mobile platforms are WMRs that utilize two or more actively steered standard wheels. Because of their kinematic configuration, such WMRs are referred to as nonholonomic, omnidirectional robots [50, 70] or pseudo-omnidirectional robots [19, 64]. As mentioned in the introduction, These have recently gained significant popularity and are now being used in a wide range of practical fields, including service robotics (PR2 [21], Care-O-bot [34], Rollin' Justin [29]), space robotics (Mars-Exo-Rover [64]), agricultural applications [6, 13], and so on [32]. Figure 2.1 depicts several of such robots. In what follows, we formally investigate the maneuverability of those platforms and employ it to summarize the most advanced research in designing motion controllers for such robots.

2.1.1.1 Kinematic Configuration

Based on Campion et al. [11], for such platforms, the degree of mobility, δ_m, is 1 and the degree of steerability, δ_s, is 2. Consequently, the degree of maneuverability, $\delta_M = \delta_m + \delta_s$, is three [68, Ch. 3]. The degree of steerabily of 2 indicates that only two of the wheels' headings, which are the steering angles, can be independently set; results in the formation of a velocity space that determines the instantaneous direction for the platform's velocity. The degree of mobility, δ_m, of 1 indicates that this velocity space is one-dimensional. Hence, by setting the steering angles, the driving velocity of only one wheel can be chosen arbitrarily and the rest of the driving velocities should be chosen accordingly.

The above statements have various kinematic and geometric interpretations and consequences. Kinematically, the mobile platform is able to realize any arbitrary and independent set of linear and angular velocities, but only after it has initially reoriented its wheels to a predetermined corresponding configuration. Thus, while the degree of maneuverability, δ_M, being 3 indicates that such platforms have three Degrees of Freedom (DoF) in motion, they are in fact over-actuated [55], with all of the actuators abiding by

the nonholonomic constraints of the robot. Therefore, such robots are called nonholonomic omnidirectional robots [50, 70] or pseudo-omnidirectional robots [19, 64].

2.1.1.2 Singularities and ICR

The Geometric interpretation of above maneuverability features incorporates the notion of Instantaneous Center of Rotation(ICR) [73]. The ICR of the robot body is defined on the horizontal Cartesian plane and with respect to the coordinate frame attached to the body. The ICR is determined by the intersection of horizontal lines normal to the wheels' vertical plane and passing through wheels' driving axles. Hence the steering of two wheels determine the ICR and the rest of the wheels' steering angles should be set such that the wheels' axles pass through the ICR. Consequently, each wheel is following an instantaneous circle with the ICR at its center. Hence, the common way to formulate the configuration space of such WMRs is through the notion of ICR[17].

Another prominent aspect of nonholonomic, omnidirectional platforms is the presence of singularities, both inherently [73] and in the presentation of the configuration space [18]. This makes the design and realization of motion controllers a challenging task. While several types of WMRs possess steering wheels, those singularities become a major issue for WMRs that utilize two or more actively steered standard wheels.

This issue may be demonstrated best using the platform's ICR. As the ICR approaches a wheel axis, the radius of that circle becomes smaller and the driving velocity of that wheel decreases, while its steering velocity unboundedly increases. When the ICR coincides with the wheel steering axis, the driving velocity of the wheel becomes zero and its steering angle becomes undefinable. This issue is shown in Figure 2.2, in which a WMR strives to follow a straight line, while it turns around itself. Hence, the 2D position vector of the ICR, along with the angular velocity, can serve as the state space for the platform, except when a wheel is singular and should be treated separately [73]. Moreover, when traversing along a straight line, the ICR remains at infinity, which can be regarded as a

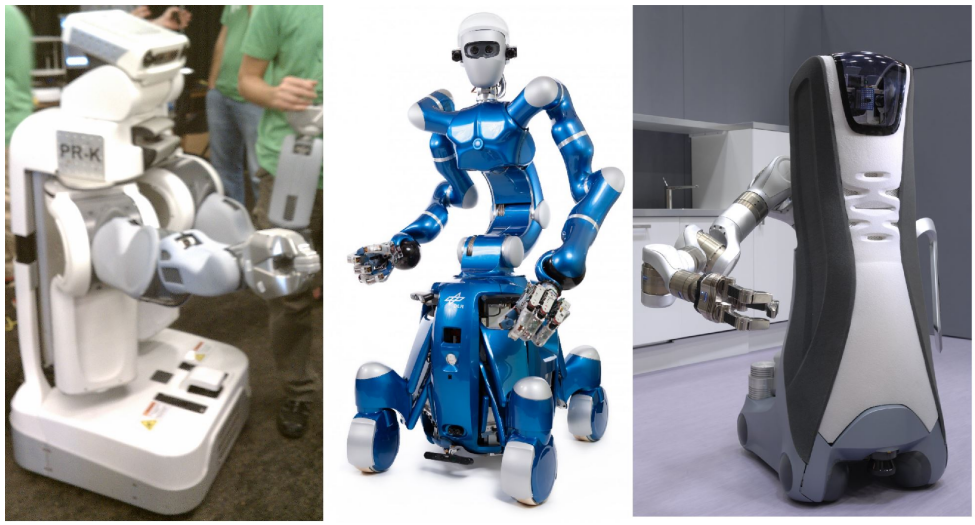


Figure 2.1: Several examples of nonholonomic omnidirectional WMRs. From left to right: PR2 [21], Rollin' Justin [29], Care-O-bot [34].

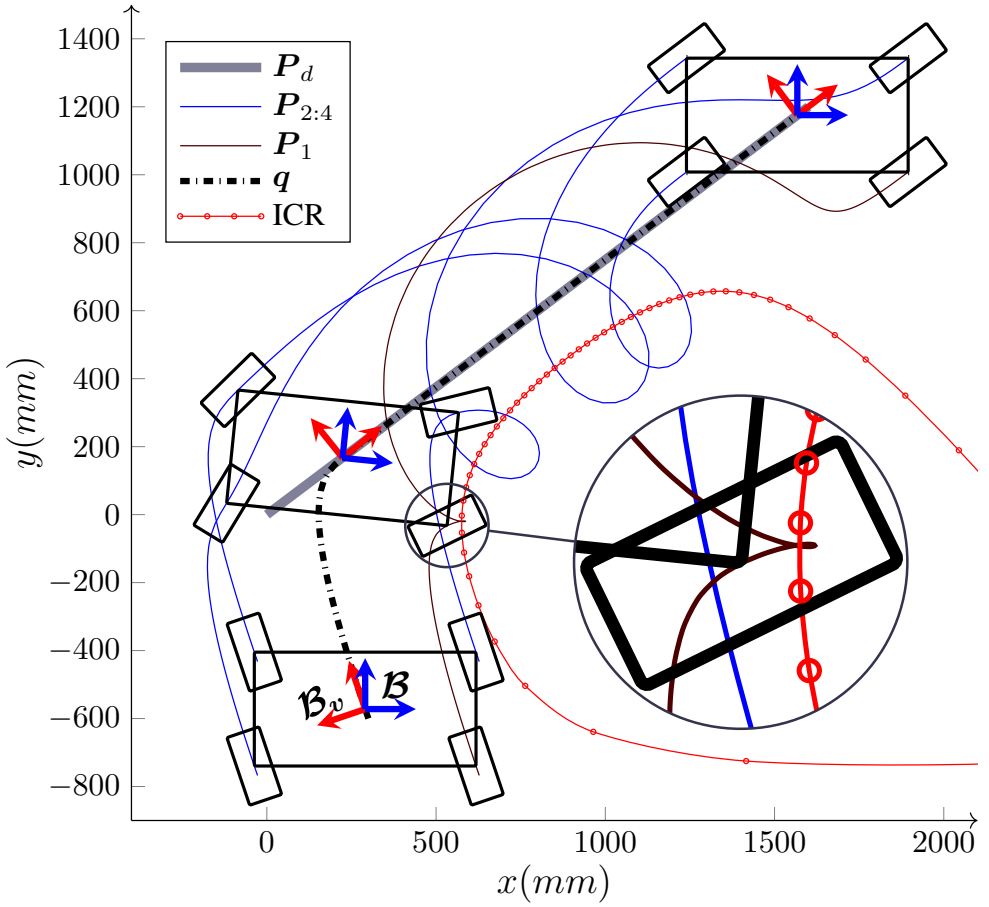


Figure 2.2: A nonholonomic, omnidirectional WMR with four steering wheels following the desired path $\boldsymbol{P}(d)$, which is a straight line, while turning around itself.

presentation singularity for that state space. The authors in [18] proposed an alternative ICR representation to avoid such singularities. Results in [18] were later extended in [17], where a singularity−free switching state space has been proposed that addresses both inherent and presentation singularities. In another approach, to avoid singularities [28], each wheel has been given an extra DoF, which makes the wheels' footprint variable. However, the problem of operating in the close neighborhood of the singular configurations remains unsolved.

Most of the solutions proposed so far have tried to plan ICR trajectories in singularity-free regions of the platform velocity space [73]. Other solutions [19, 20] treat the singular configurations and their neighborhood as obstacles and solve a navigation problem based on potential field and/or model predictive control methods. However, in all of these methods, considerable portions of the configuration space are avoided, thereby reducing the maneuverability of the platform. Even when realizing some simple maneuvers, at some points of the operation, the ICR will necessarily become relatively close to at least one wheel. One simple case is when the platform is moving on a straight line while changing its heading by 180 degrees. Furthermore, when the robot is required to follow a desired path and heading profile, the ICR position has already been determined; therefore, none

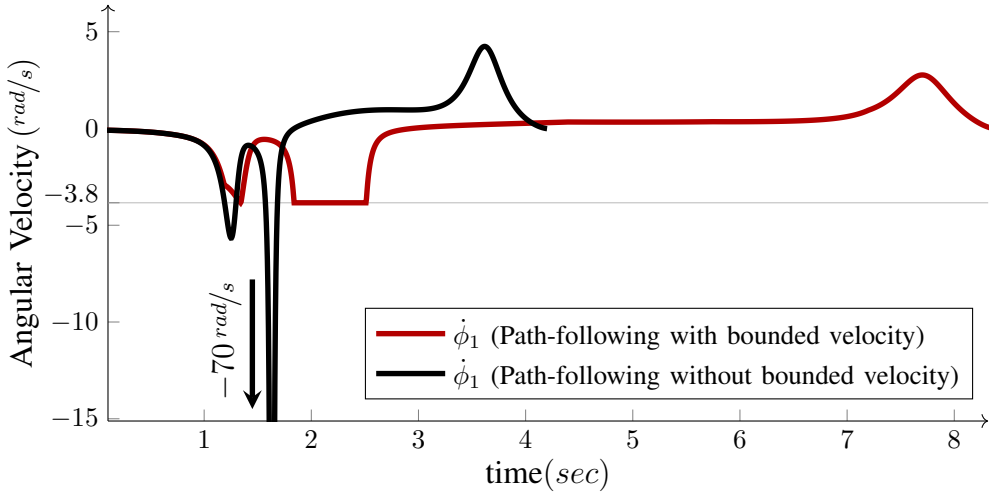


Figure 2.3: First wheel angular velocity $\dot{\phi}_1$ with and without bounded velocity.

of these approaches are suitable for path-following problems.

One of the significant benefits of the proposed bounded velocity path-following is that it properly handles steering wheel singularities. Figure 2.3 depicts the steering velocity of one of the wheels during the path-following operation for the scenario depicted in Figure 2.2. The figure clearly shows the jump in the steering velocity in the vicinity of singularities without the bounded velocity solution. However, with the proposed method, as the WMR approaches its singular configuration, it properly reduces its speed and allows the steering wheel to make its tight turn.

2.2 Integration and Implementation

This section gives a brief review of the integration and implementation of software and hardware modules in experimental setups that are commonly used in various mechatronic applications, including research platforms in the field of mobile robots and mobile manipulators.

2.2.1 Software Design for Real-time Controllers

Rapid prototyping of real-time software for complex mechatronic systems is a challenging but highly demanded task. In many cases, developers – and particularly control systems designers – find themselves immersed in the technical difficulties of real-time programming and hardware interfacing.

The inherent complexity of a mechatronic system as a multidisciplinary field necessitates efforts in various areas [24]. Such a system should include a common language between software engineers and control engineers at many design and implementation levels correlating with the development of such systems. In many cases, these systems are the basis for iterative verification and validation of different research projects or commercial products, or they are used for educational purposes [33].

In either case, certain parts of the system require the focus of designers and developers, while for other parts, utilizing already available subsystems is considered a fair deal.

The mechatronic system considered here consists of one or more embedded PCs with real-time Linux as the main processing unit connected to different hardware components via distinctive mediums with appropriate interfaces. Hence, these systems fall into the category of PC−based automation solutions [77].

There are different issues associated with the development of such mechatronic systems, especially at the low level, which covers device drivers and system controllers. One example of such difficulties is interfacing with hardware devices, which includes different communication mediums and protocols [81]. Moreover, developing such systems requires consecutive and numerous practical tests to verify and validate the functionality and integrity of the developed algorithms and codes. Ultimately, this should lead to optimized implementation of controllers that could receive sensor data punctually and produce control signals in guaranteed intervals of time [27]. In this regard, while software engineers prefer direct C\C++ programming, control engineers and system developers prefer Matlab and Simulink or similar software or toolboxes because of their ready-to-use, advanced mathematical functions and algorithms that could be utilized directly. However, this high-level programming makes the models developed in Matlab unsuitable for real-time tasks [10].

Several studies have been done for remediation of the real-time implementation of algorithms in Matlab [37] and [14] or its integration with Linux [59]. Moreover, there are applications of available software packages utilizing selected functionalities of Matlab in their real-time execution environment, such as xPC Targeting toolbox [53], RT-Lab [1], and RTAI-Lab [10]. Mathworks has addressed this problem by embedding code generator products that automatically generate optimized, independently and royalty-free C\C++ code from the Matlab codes and Simulink models. Comparison of the generated C\C++ codes with their original Matlab code is the subject of research in [12, 79].

Generally, one of the challenging issues in a mechatronic system is making operational interfaces between software that develop computational controllers like Matlab and real-world sensors and actuators. This challenge has been solved in dSpace® products by means of the complementary hardware in its package for rapid prototyping or code generation in many studies [46, 66]. Another comprehensive work has been carried out by the RTAI Linux team [10] involving the integration of RTAI-Lab into Simulink and translation of the code into the RTAI framework by means of *Real-Time-Workshop* (RTW). These systems are popular for a certain range of control and software requirements. However, their solutions enforce limitations compared to traditional C\C++ software development in terms of the reusability of existing codes. This problem is negligible if R&D teams use only compatible products for the development environment. In case of any unforeseen necessity of instruments, adaptation of a third-party driver to those environments is rife with difficulties. In contrast, a normal C\C++ application can easily use manufacturer-provided device drivers for C\C++.

During the procedure of designing a complex mechatronic system, several teams of researchers work simultaneously on design processes and the modeling of hardware and software platforms. Thus, even conceptual design and development of the product hardware can be subject to change in the same period as software design. Moreover, validation of the developed design models, methods, and tools are carried out with the prototype experiments. The desirable choice is to generate blocks, modules and patterns of the implemented components and written software that can be reused. Therefore, a real-time control, rapid-prototyping environment is necessary to overcome the control issues of the mechatronic system and at the same time consider other criteria, such as modularity,

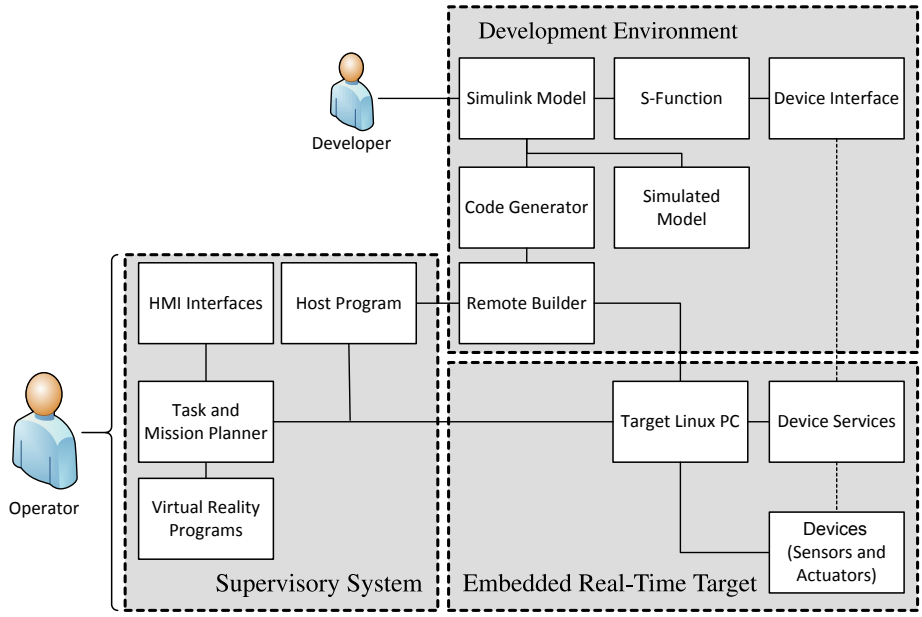


Figure 2.4: Schematic diagram of the rapid prototyping framework

flexibility, reusability, compatibility with commercially off-the-shelf components, while maintaining a dependable basis for future advances in safety measures, such as *Safety Integrity Level* (SIL) [47].

We aim to introduce a coherent framework for remote development of independent, hard real-time software controllers based on Matlab and Simulink products for the rapid prototyping of PC-based mechatronic systems. The author provided the technical details of the method in **PV**. Our proposed method consists of developing software modules and controllers in a Simulink project on a PC with a Windows operating system, with a target of one or several embedded PCs running real-time Linux (the "Target PC"). Then, the Code Generators toolbox is used to derive a stand-alone, hard real-time C\C++ code for the real-time Linux operating system. The resultant code is automatically transferred to the embedded target, where it is compiled and built. The final software is ready to debug, run, or verify on the Target PC. The conceptual implementation of this framework is depicted in Figure 2.4.

Note that this methodology could easily be expanded to develop multiple software programs to run on the Target PC simultaneously. For very complex mechatronic systems, in which software and controllers cannot be put into one *executable*, this method could be used simultaneously for multiple Simulink models to develop different parts of the whole software. Then, each model could be turned into a module or service on the Target PC and collaborate with other modules by means of different internal communication services, such as programming queues. Hence, developers would be able to benefit from Service-Oriented Architecture (SOA) principles [66] in developing their software. Furthermore, the code can be generated not just for Windows but also for a wide variety of operating systems that support the *Portable Operating System Interface* (POSIX) [79].

We are plan to addresses some of the issues and difficulties commonly associated with remote development and code generation with Matlab and Simulink products. One of

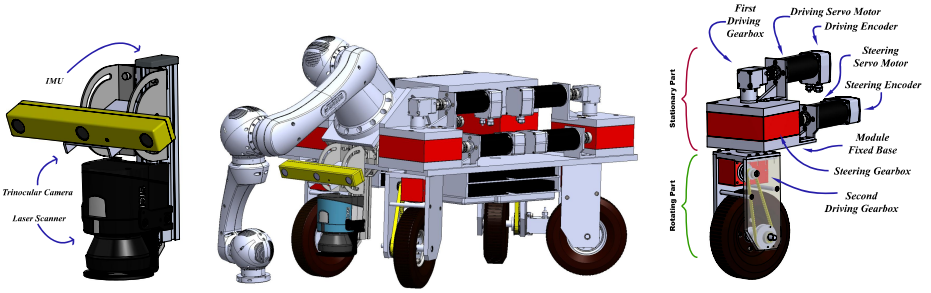


Figure 2.5: Three-dimensional solid model of iMoro and its components

the main issues is how to integrate device drivers and hardware interfaces into Simulink models. Technically, these interfaces should be turned into reusable Simulink block libraries. However, the drivers and their related API libraries are installed differently on the remote target system (real-time Linux) than they are on the operating system of the development environment (Windows). Hence, deriving such blocks demands more advanced treatment. Addressing this issue requires detailed knowledge of S-Function programming, as well as the development of *Target Language Compiler* (TLC) files that tell code generators how to generate C code out of the S-Functions.

2.2.2 iMoro Mobile Manipulator

A mobile manipulator refers to a robotic manipulator arm mounted on a mobile base. Mobility provides the manipulator with an unlimited workspace at the expense of added challenges [35, 38]. The *i*Moro, shown in Figure 2.5, is a Four Wheel Independently Steered (4WS) mobile manipulator that is under investigation for high maneuverability and flexibility in high-speed motions under the Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation (PURESAFE) project. The project, was initiated to advance theoretical and experimental knowledge on semi-autonomous mobile manipulation within the ionizing radiation-filled, and contaminated environments of accelerators, particularly within tunnels inside the European Organization for Nuclear Research (CERN).

A mobile manipulator is a complex mechatronics system that synergistically blends multidisciplinary solutions and methods from mechanics and embedded control systems [5, 75]. There are different theoretical [78] and practical issues associated with the development of such mechatronic systems, especially in high-level [58] and low-level architectures involving device drivers and system controllers. One example of such an issue is that of interfacing with hardware devices, including different communication mediums and protocols. Moreover, developing mechatronic systems requires numerous and consecutive practical tests to verify and validate the functionality and integrity of the developed algorithms and codes. Ultimately, it should lead to an optimized implementation of controllers that can punctually receive sensor data and produce control signals over guaranteed intervals of time.

2.3 Summary

This chapter provided a survey on several topics pertinent to mobile robots and manipulators. The main focus was on the motion control of mobile robots, specifically the

path-following problem [31]; this, along with the point stabilization [57], and trajectory tracking [39] approaches, represents most of the motion control topics for autonomous mobile platforms.

Aside from motion control of WMRs in general, special attention was given to certain types of WMRs, namely WMRs with active actively steered standard wheels. This type of WMR is both nonholonomic and omnidirectional [50, 70]; therefore, it is called pseudo-omnidirectional [19, 64]. While these WMRs have gained popularity in many practical fields [21, 34], there several challenges related to their motion control, including singularities that arise from the configuration of the steering wheels and their velocities [20].

Furthermore, this chapter reviewed several approaches for the design and rapid prototyping of real-time controllers for mechatronic systems, especially mobile robots and manipulators. Several challenges in this area include but are not limited to difficulties in interfacing with and integration of hardware devices [81] and implementation of controllers that could receive sensor data punctually and respond by producing control signals in guaranteed intervals of time while abiding by hard, real-time deadlines [27].

3 Overall Concept and Experimental Results

In this chapter, we demonstrate some extra simulation and experimental data from the presented path-following controller in action based on the theoretical results of **PIV**, which have not been included in the publications. The results are for four categories of WMRs.

3.1 Experimental Setup

As described in Section 1.1, WMRs are formally classified into five categories based on their kinematic structure and maneuverability using the pair: $\delta = (\delta_m, \delta_s)$, in which δ_m is the degree of mobility and δ_s is the WMR's degree of steerability. The simulation and

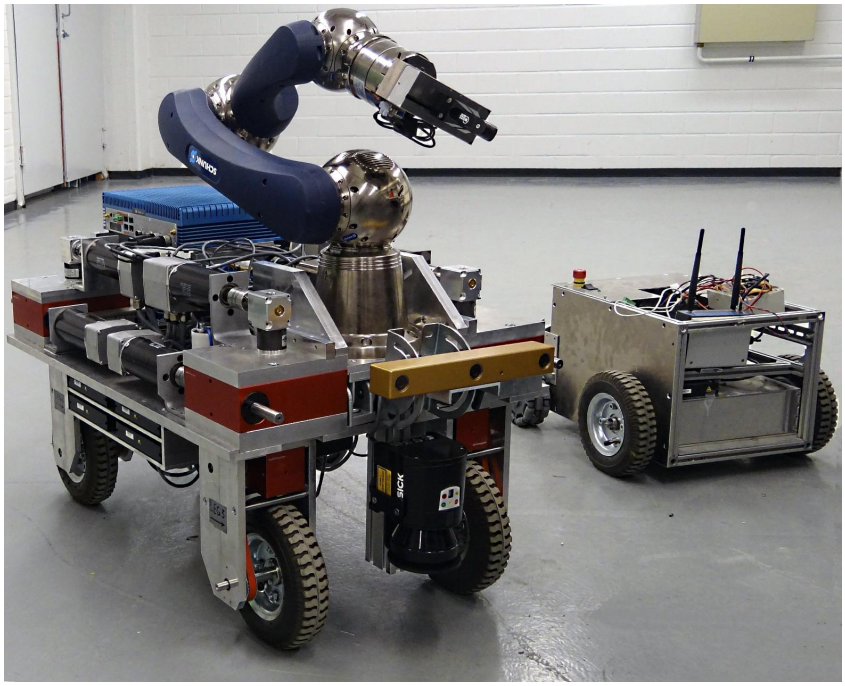


Figure 3.1: *Experimental Setups*: iMoro (left): a four wheel independently steering WMR ($\delta = (1, 2)$), and LabRat (right): a differential drive WMR($\delta = (2, 0)$) with active fixed wheels at the rear.

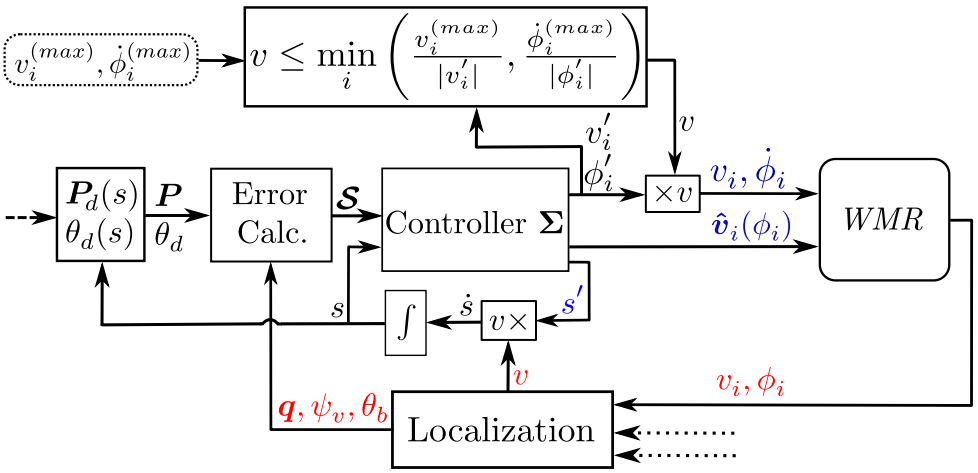


Figure 3.2: Schematic block diagram of the whole system

experimental results are for four of those categories. Specifically, the simulation has been performed on a holonomic, omnidirectional WMR ($\delta = (3, 0)$), and experiments are done for types $\delta = (1, 2)$, $\delta = (1, 1)$, and $\delta = (2, 0)$.

The experimental setups consist of two WMRs, as shown in Fig.3.1. The robot on the left is the iMoro mobile manipulator described in Section 1. As explained above, this is a nonholonomic, omnidirectional WMR ($\delta = (1, 2)$), also known as two-steer [68]. By manually fixing the steering of the rear wheels (setting $\dot{\phi}_i = 0$), it can also emulate the car-like type($\delta = (1, 1)$). In this case, the steering of the front wheels is naturally governed by the path-following based on the Ackerman principle. This feature has been incorporated to test the algorithm for the car-like WMRs. For the last type, the WMR on the right of Fig. 3.1, called LabRat, has been employed. This is a differential drive mobile robot ($\delta = (2, 0)$) and represents the unicycle kinematics. The path-following controller has been implemented on these WMRs in a real-time Linux environment based on **PV** and **PVII**.

The simulation of the path-following algorithm has been performed for a holonomic, omnidirectional WMR ($\delta = (3, 0)$), described hereafter as the holonomic type. The simulated robot has the same architecture as iMoro; the only difference is that the steering wheels are virtually replaced by Swedish wheels of the same radius.

3.2 Control Architecture

The control architecture and the relevant details have been explained in **PIV**. However, for the sake of continuity, a brief overview is provided here. Figure 3.2 depicts the schematic block digram of the whole system. The input to the system is the desired path $\boldsymbol{P}_d(s)$ and heading $\theta_d(s)$. Here, s is the natural parametrization (arc length) of the desired path $\boldsymbol{P}_d$ and serves as an auxiliary state of the system, where the rate $\dot{s}$ is a control signal. The driving velocity and steering direction of the i^{th} wheel are denoted as v_i and ϕ_i respectively. Correspondingly, $v_i^{\max}$ and $\dot{\phi}_i^{\max}$ are the maximum allowable driving and steering velocities, which are preset based on the actuators' specifications. Moreover, scalar v is the base speed of the WMR and an exogenous input for the controller.

The wheel odometry data and feedback from other on-board sensors are used by the localization block, which evaluates the platform's pose[1] information. Comparing these with the desired pose results in the error states, which are fed into the controller Σ. The natural parametrization of the path that is traversed by the WMR is denoted as λ. As described in **PIV**, the proposed controller Σ provides v_i', ϕ_i', and s', which are the partial differentiation of v_i, ϕ_i, and s with respect to λ. Therefore, as shown in the figure, the base speed v is selected such that $v \leq \min_i \left(\frac{v_i^{(max)}}{|v_i'|}, \frac{\dot{\phi}_i^{(max)}}{|\phi_i'|} \right)$. As demonstrated in **PIV**, this selection guarantees that wheel driving and steering velocities stay equal or less than their maximum bounds, which are $v_i^{\max}$ and $\dot{\phi}_i^{\max}$, respectively.

3.3 Case Studies and Results

For the simulation, Fig. 3.3 portrays the path-following scenario, the base footprint $\boldsymbol{q}$, and two of the wheels' paths. Similarly, the iMoro and LabRat footprints performing the same path-following scenario are illustrated in Fig. 3.5 for the two-steer type, in Fig. 3.7 for the car-like type, and in Fig. 3.11 for the unicycle type. As shown in the figures, the desired path is smooth but has sharp turns and hence large curvatures at some of its points, which challenges the agility of the controller.

We have intentionally set large initial errors to demonstrate the performance of the controller. The WMR is $2m$ off the starting point of the path and faces away from it; hence, the initial heading error is $-180°$. For the types with $\delta_M = 3$, which are two-steers and holonomic WMRs, independent heading control is possible, and in this scenario, the desired heading is set to be $360°$ by the end of the desired path. Conversely, for the types with $\delta_M = 2$, which are car-like, and differential drive WMRs, independent heading control is not possible. For those cases, the origin of the body frame has been placed on the common axis of the fixed wheels and therefore, the heading and the linear velocity angles are the same. Hence, in addition to the path-following of the base origin point $\boldsymbol{Q}$, the algorithm corrects the initial heading error and has the base heading follow the tangent of the path. As shown in these figures, the controller successfully manages to maneuver the robot toward and asymptotically onto the path.

In the simulation, as well as the experiments, the maximum driving and steering velocities have been set to be $600^{mm}/_{s}$ and $220^{\text{deg}}/_{s}$, respectively, and the base speed is selected to be $v = v^{(max)}$, as given by the bounded velocity trajectory generator. Figures 3.4, 3.6, 3.8, and 3.12 present the wheels' driving velocities v_i and the base speed v for the holonomic, two-steer, car-like and differential drive types, respectively. As shown in the figures, the bounded velocity algorithm duly scales the base speed so that none of the wheels exceed their maximum driving velocity. Note that for the wheels with steerings, the steering angle determines the driving direction, and therefore the driving velocities are always positive. In contrast, for the fixed wheels, the driving velocities may be positive or negative, which indicates the direction of the wheel's driving (forward or backward). Therefore, some wheels might reach their maximum velocity while driving backwards, which corresponds to the negative values of the velocities in the figure.

The base speed is selected to be the maximum. Hence, in this implementation at least, one of the wheels runs with its maximum driving or steering velocity. Comparing Figs. 3.6 and 3.9, it is clear that most of the time at least, one of the wheels drives at its maximum

[1] Position and Orientation.

driving velocity. However, when a tight turn is needed, the maximum velocity steadily changes from the driving to the steering. For the two types with steering wheels, Figs. 3.9 and 3.10 depict the steering velocities for the two-steer type and the car-like type (only the front wheels), respectively.

Notice that in the two-steer mode, iMoro approaches its singular configurations a couple of times, which corresponds to some of the peaks in the steering velocities in Fig .3.9. In Fig. 3.5, the robot is shown at one of those near singular configurations, demonstrating the high curvature of the wheel's path and the closeness of the body ICR to the steering axis of the wheel. However, the increase in steering velocities is not solely due to the singular configurations and closeness of the body ICR to the steering axes. As shown in Fig .3.5 the high curvature of the desired path could also lead to the same effect and occurs when the robot steers to change the direction of its base linear velocity.

This can also be studied analytically, which has been done in **PII** using the closed-loop equation for the path curvature of the i^{th} steering wheel. Unlike in the case of singularities that are due to the decrease in the numerator of Eq. 12b in **PII**, this case occurs because of the increase in the denominator, namely κ_v. In this case, while the curvature of the wheel's path increases, the body ICR is not necessarily near the wheel's steering axis.

This matter is more apparent from the steering velocities in the car-like mode, where there is no independent heading control. Fig. 3.10 shows that the car-like WMR reduces its speed and increases its steering velocity to change the direction of its linear velocity and make the sharp turns imposed by the desired path. In contrast, in the unicycle case, there are no steering wheels. As shown in Fig. 3.12, LabRat makes the sharp turns by setting the velocities of both wheels to their maximum values but with different signs, which implies that one is driving forward and another is running backward. Hence, the base speed becomes almost zero and the velocity difference leads to the angular velocity needed for the turn. Collectively, these experiments demonstrate the agility of the proposed path-following algorithm to steadily realize sharp maneuvers without relying on switching procedures.

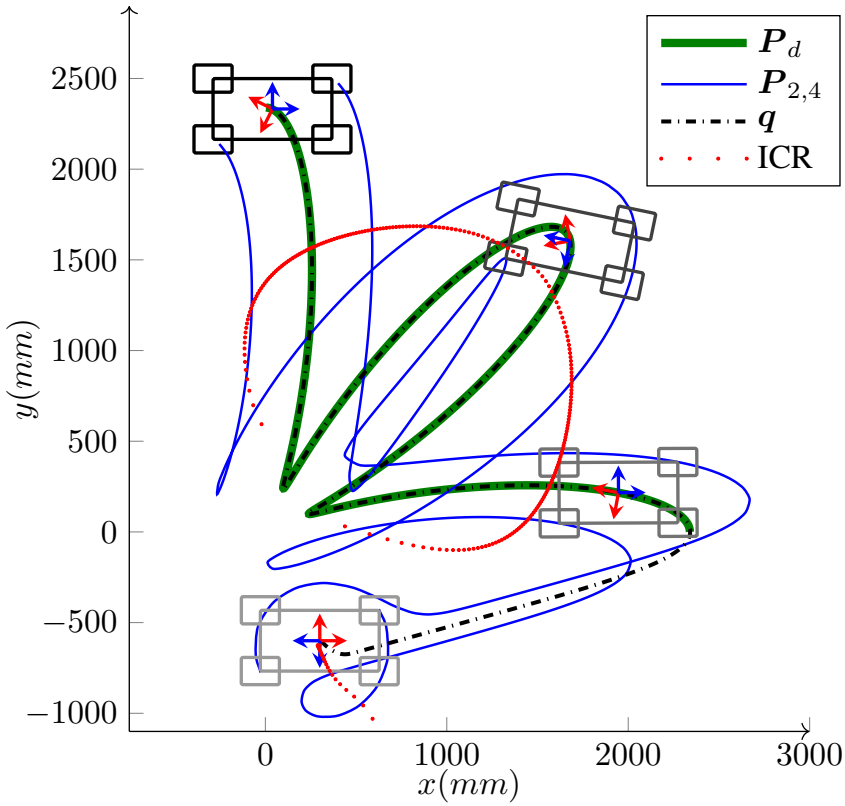


Figure 3.3: *Simulation*: Path-following with large initial errors of a WMR with four Swedish wheels ($\delta = (3, 0)$). It seeks and follows the path $\boldsymbol{P}_d$ while correcting its heading from the initial error of $-180°$ to the desired heading of $360°$ at the end of the path.

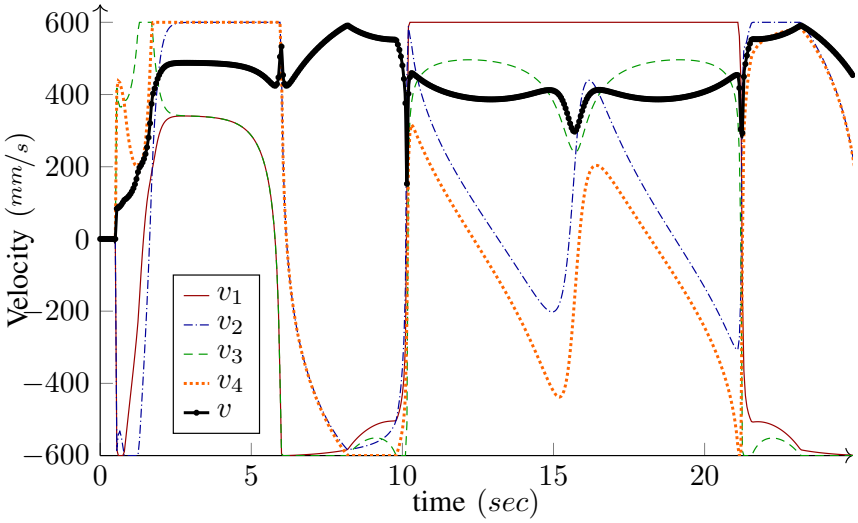


Figure 3.4: *Simulation*: Driving velocities v_i, and the base speed v, for path-following of the WMR with four Swedish wheels depicted in Fig.3.3.

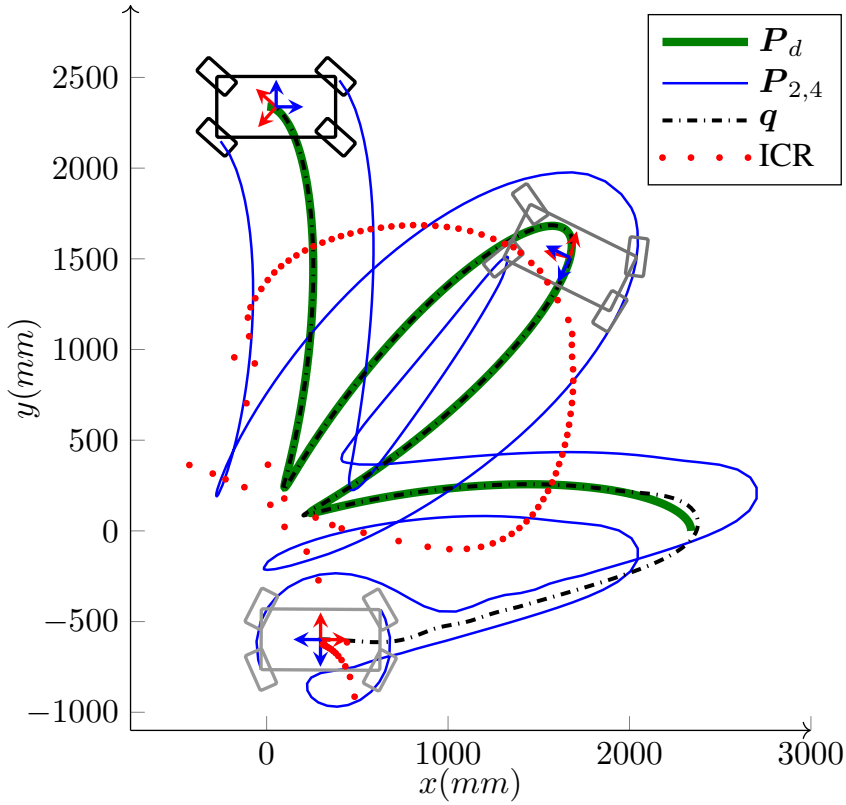


Figure 3.5: *Experiment*: Bounded velocity path following with large initial errors and independent heading control of the iMoro WMR ($\delta = (1, 2)$). It seeks and follows the path $\boldsymbol{P}_d$ while correcting its heading from the initial heading error of $-180°$ to the desired heading of $360°$ at the end of the path.

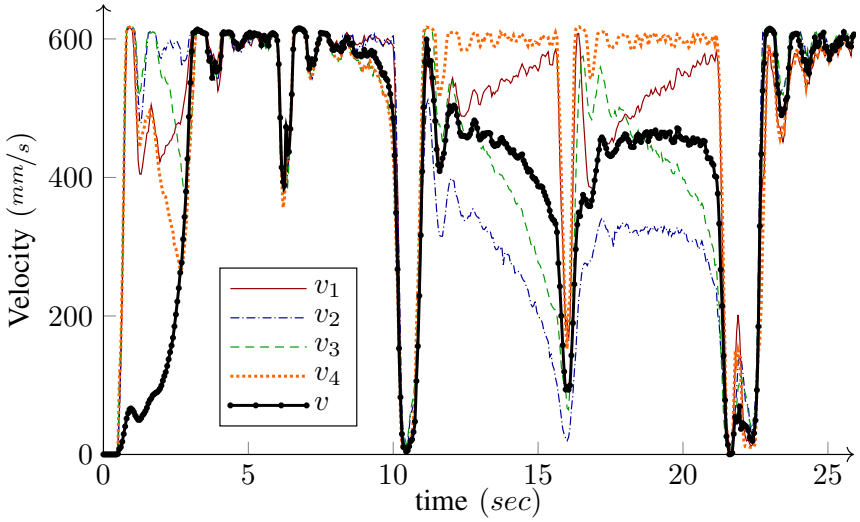


Figure 3.6: *Experiment*: Driving velocities v_i, and the base speed v, for the bounded velocity path-following of iMoro depicted in Fig. 3.5. The maximum driving velocity for all of the wheels ($v_i^{(\max)}$) is set to be $600^{mm}/_{s}$.

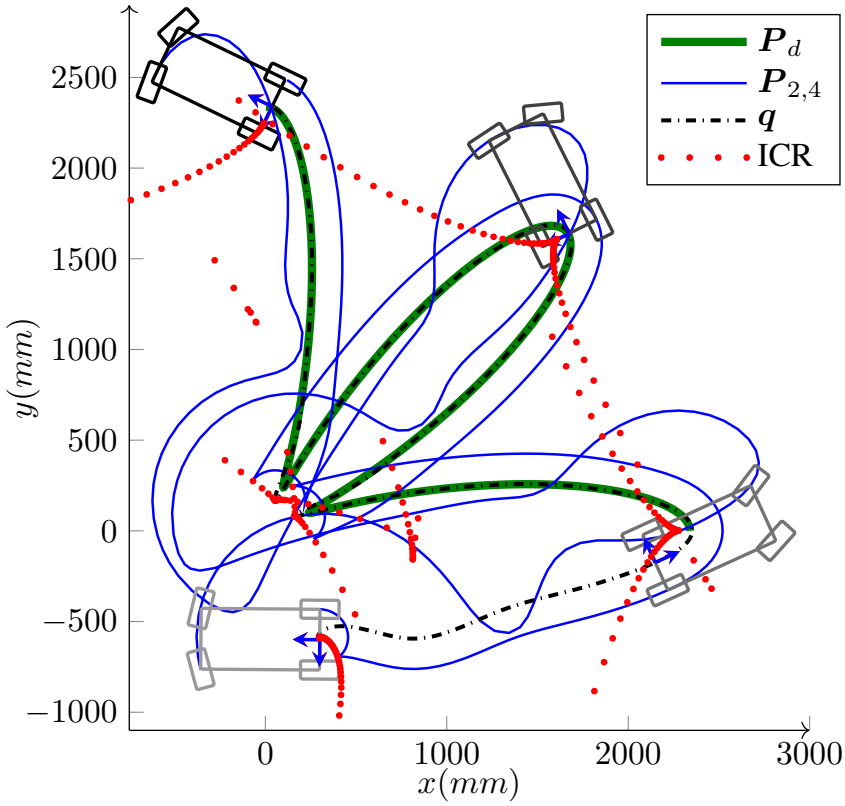


Figure 3.7: *Experiment*: Bounded velocity path-following with large initial errors for iMoro WMR in car-like mode ($\delta = (1, 1)$). It seeks and follows the path $\boldsymbol{P}_d$ while correcting its heading from the initial heading error of $-180°$ toward the path tangent angle $\psi_t(s)$.

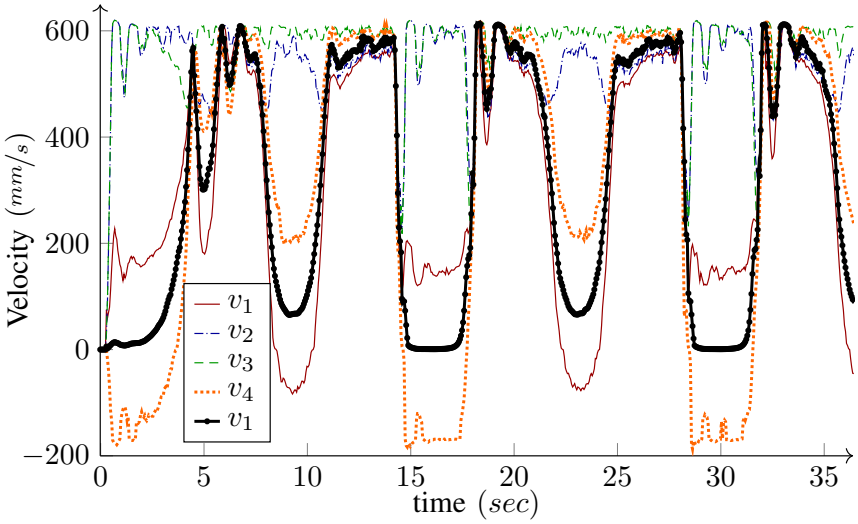


Figure 3.8: *Experiment*: Driving velocities v_i, and the base speed v, for the bounded velocity path-following of iMoro in car-like mode depicted in Fig. 3.7. The maximum driving velocities ($v_i^{(\max)}$) are set to be $600^{mm}/_{s}$.

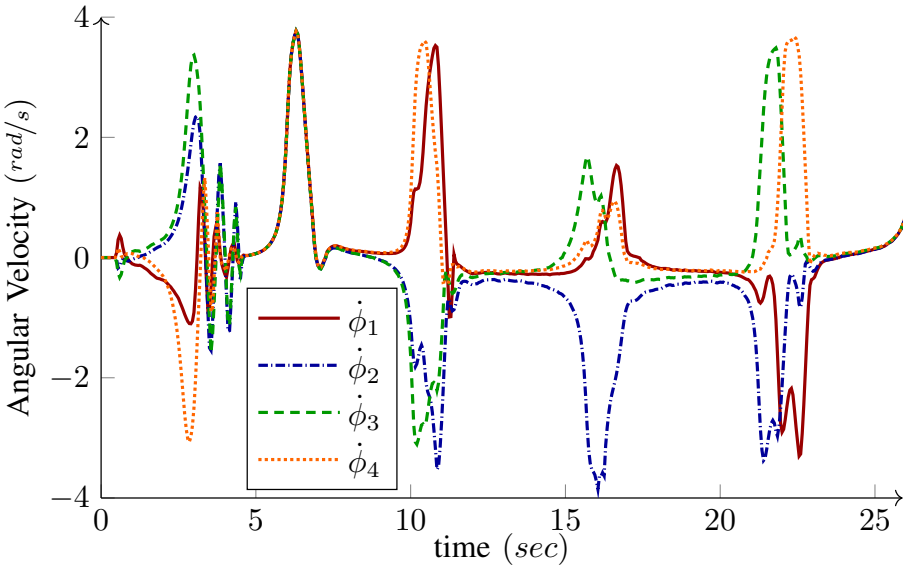


Figure 3.9: *Experiment*: Steering velocities $\dot{\phi}_i$ for the bounded velocity path-following of iMoro depicted in Fig .3.5. The maximum steering velocity for all the wheels ($\dot{\phi}_i^{(\max)}$) are set to be $3.84^{rad}/_s(220^{\text{deg}}/_s)$.

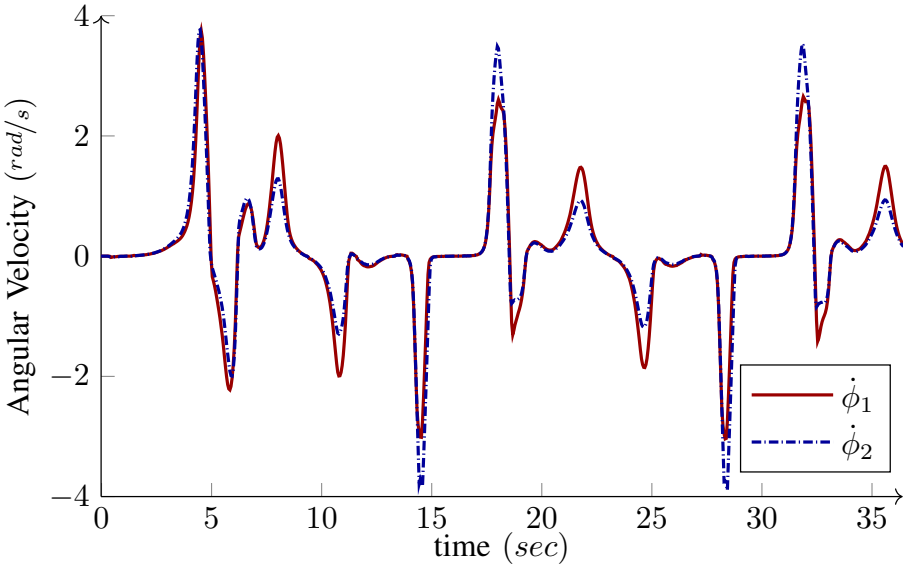


Figure 3.10: *Experiment*: Steering velocities $\dot{\phi}_1$ and $\dot{\phi}_2$ of the front wheels of iMoro for the bounded velocity path-following scenario depicted in Fig. 3.7. The maximum steering velocities are set to be $3.84^{rad}/_s(220^{\text{deg}}/_s)$.

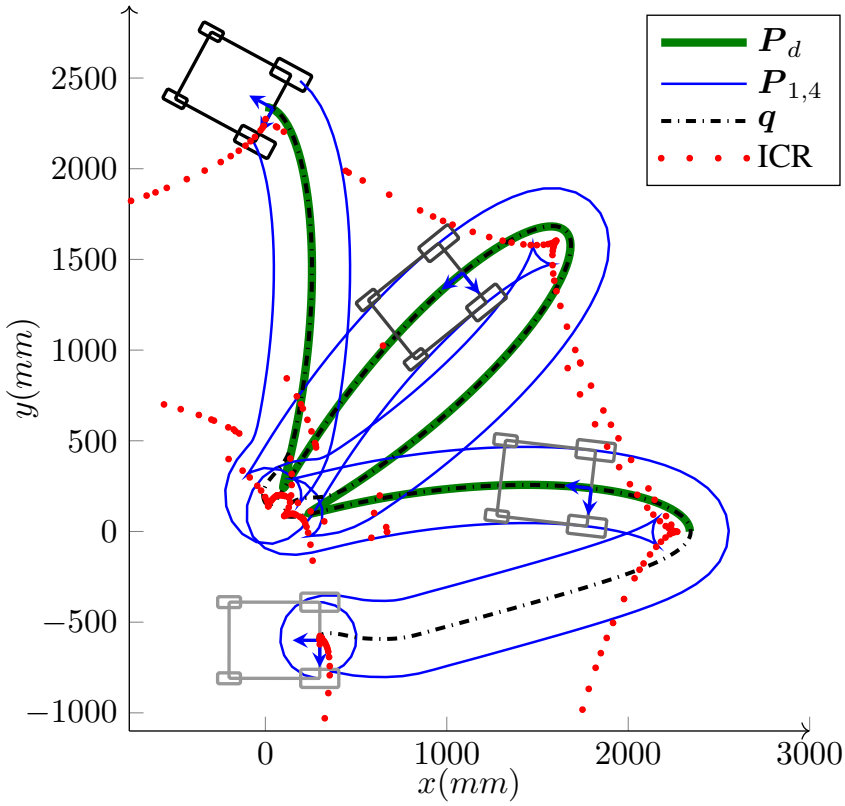


Figure 3.11: *Experiment*: Bounded velocity path-following with large initial errors for LabRat WMR ($\delta = (2, 0)$). It seeks and follows the path $\boldsymbol{P}_d$ while correcting its heading from the initial heading error of -180° toward the path tangent angle $\psi_t(s)$.

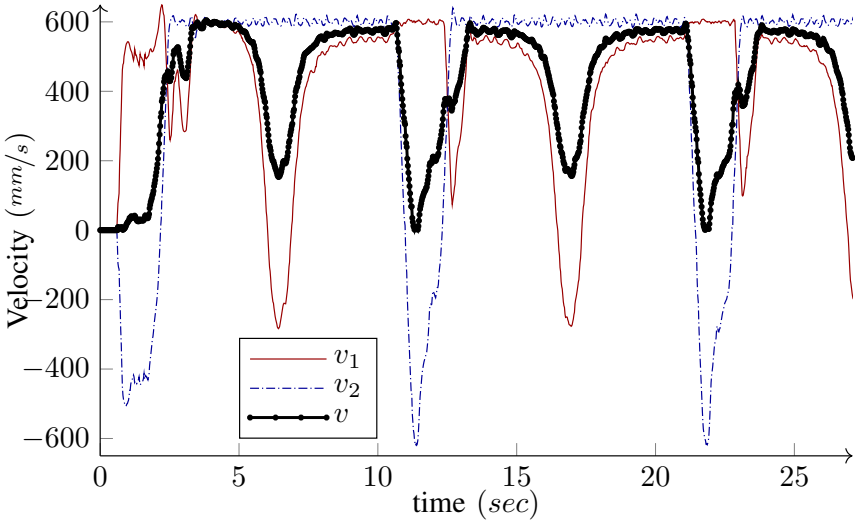


Figure 3.12: *Experiment*: Driving velocities v_1 and v_2, and the base speed v, for the bounded velocity path-following of LabRat ($\delta = (2, 0)$) depicted in Fig. 3.11. The maximum driving velocities ($v_i^{(\max)}$) is set to be $600^{mm}/_s$.

4 Summary of Publications

This chapter provides a brief summery of each of the listed publications. The first section gives a summery of the first four publications, which are related to the theoretical development of the bounded velocity controller for WMRs, while the second section summarizes the last three, which mainly cover the implementation phase, including mechatronic integration and real-time software design.

4.1 Theoretical Development

4.1.1 PI: Bounded-Velocity Motion Control of Four Wheel Steered Mobile Robots

This paper proposes a path-following controller for 4WS mobile robots. We provide non-linear motion control laws for the WMR's base to follow a given desired smooth path and heading profile. Having the orientation of the linear velocity vector and the angular velocity of the base as control signals, control laws are derived to solve the path- and heading profile-following problem. Therefore, the proposed controller provides instantaneous asymptotic trajectories that guide the robot toward and onto the desired pose.

Previous related studies incorporate a specific representation of the robot's ICR as the WMR's configuration space and utilize that to analyze and prevent singularities. However, in this paper, the base asymptotic trajectory is directly incorporated into the wheels' kinematic constraints, which yields to a set of expressions for the curvature of the wheels' trajectories. Those expressions are then utilized to derive a bounded velocity solution for the actuators.

Our solution takes full advantage of the steerability of all wheels and provides the capability for independent control of translation and rotation of the robot. However, in order for this solution to practically bound the velocities, the underlying path of the asymptotic trajectories has to be relatively close to the desired path. This has been enforced in the controller via the incorporation of saturation functions in the control laws that virtually bound the errors; therefore, in case of large errors, the convergence to the desired path is relatively slow. This issue has been addressed in the next paper by proposing a different set of control laws that result in instantaneous asymptotic paths rather than trajectories, which allows us to derive a closed-form bounded velocity solution with exponential convergence, regardless of the size of the errors.

4.1.2 PII: A Novel Time Optimal Path Following Controller with Bounded Velocities for Mobile Robots with Independently Steerable Wheels

This paper proposes a novel solution to the path-following problem of independently steered mobile robots. Specifically, it overcomes some of the shortcomings of the previous paper by proposing a set of new control laws for the WMR's base. We show that similar to the previous solution, utilizing this controller allows the robot to smoothly follow the desired path while it changes its heading based on a given desired function. Moreover, this design leaves the speed of the robot base as an arbitrary variable. However, in contrast with the previous controller, the new control laws significantly simplify the noholonomic constraints of the wheels into explicit proportional relations between the base speed (magnitude of the base linear velocity vector) and the driving and steering velocities of the wheels. This feature is then utilized to present a close-form bounded velocity trajectory generation that globally bounds the driving and steering velocities under their corresponding prespecified bounds.

Many proposed approaches strive to keep the robot out of bulky regions around the singular points by relying on numerical solutions, and hence, lose some of the robot's maneuverability. However, the proposed method allows the robot to arbitrarily approach its singular configuration by duly regulating the base speed, which in turn provides a solution that fully exploits the high maneuverability of the platform. The experiments show that even without considering any model-based approach or dynamic analysis, the designed path follower and related kinematics formulations are capable of bounding both steering and driving velocities in practice with some tolerances. Since this approach gives an analytical solution, it is capable of real-time implementation with low process costs.

The major shortcoming of the proposed solution is that it imposes high accelerations and decelerations on the actuators to bound the velocities, especially as the robot approaches one of its singular configurations. In other words this solution shifts the high steering velocity in the vicinity of singularity from the velocity space to the acceleration space. Therefore, this shortcoming prevents having high velocity bounds, since at some points, actuators would not be able to provide the required torque (acceleration). This shortcoming has been remedied in the next paper by extending this solution to bound the wheels accelerations as well.

4.1.3 PIII: Time Optimal Path Following with Bounded Velocities and Accelerations for Mobile Robots with Independently Steerable Wheels

This paper takes up where the previous one left off and shows that for the proposed controller, the closed-loop system provides a set of differential equations for an asymptotic path that guides the platform toward the desired path, which can be derived analytically and independent of the platform velocity profile. Based on this result, this study extends the trajectory generation part of the previous solution to bound the accelerations as well as velocities. To this end, we present an online trajectory generation scheme based on a modified version of the Phase Plane Approach (PPA). The PPA is a class of minimum time trajectory generation methods based on a bang-bang switching algorithm; it was first introduced by [8] and [67]. For serial manipulators, the method is commonly used *offline* to derive the time-optimal trajectory for the robot's *desired path*; so far, this has been mainly applied to serial manipulators.

To this end, we show that the simplified closed-loop nonholonomic constraints derived in the previous paper, which are in the form of a set of differential kinematic constraints, can serve as a *Dynamic Model* for the PPA and enable us to develop an efficient *online* version of PPA that acts on the *asymptotic path* instead of the desired path and bounds the velocities and accelerations of the actuators. Moreover, we show that the algorithm efficiently regulates the velocity and acceleration of the robot around the singular configurations, which allows the robot to realize wide ranges of complex maneuvers. In other words, by setting the acceleration bounds for the driving actuators, the trajectory generator ensures that the velocities are regulated well in advance and obviates the need for high accelerations and decelerations in order to bound the velocities.

4.1.4 PIV: A Time-Optimal Bounded Velocity Path-Following Controller for Generic Wheeled Mobile Robots

This paper generalizes the bounded-velocity path-following controller proposed in **PII** for 4WS mobile robots and presents a universal solution for the path-following and trajectory generation of WMRs, which abide by the rolling/no-skidding constraint. Therefore, contrary to many other path-following controllers that focus only on one or two kinematic structures, we solve the path-following problem for all common categories of WMRs, such as car-like, differential, omnidirectional, all wheels steerable, among others. To the best knowledge of the authors, this is the first work that coherently solves the path-following problem for all these popular types of WMRs. Moreover, unlike many other path-following controllers in the literature, in this design, the control laws and the resultant closed-loop equations of motion are *explicit* in terms of the WMR's speed. We further develop the results of **PII** and show that selecting control laws transforms closed-loop equations of motion to a geometrical representation of an asymptotic pose that steers the robot toward the desired pose. In other words, the controller becomes a feedback path planner with the corresponding cost function being a Lyapunov function of errors.

The generality of this method is the consequence of a two-stage approach that tackles the platform's path-following and wheels' kinematic constraints separately. In the first stage, for a WMR divested of the wheels' constraints, we develop the controller with the aforementioned features. The second stage takes into account the kinematic constraints imposed by the wheels. In this stage, we present an abstract generalized wheel along with a set of parameters that encompass the features of all those types. All types of wheels are representable by configuring the parameters of the generalized wheel. We utilize this generalization to obtain a unified form for the kinematic constraints that facilitates the integration of the path-following controller for all types of WMRs. We demonstrate that the designed controller simplifies the otherwise impenetrable kinematic constraints for the generalized wheel into explicit proportional functions between the velocity of the platform and that of the wheels. This result enables us to extend the closed-form trajectory generation scheme for the asymptotic path to all categories of WMRs, in which the controller constantly keeps the wheels' steering and driving velocities within their corresponding prespecified bounds.

4.2 Implementation and Integration

4.2.1 PV: Unified Framework for Rapid Prototyping of Linux based Real-Time Controllers with Matlab and Simulink

This paper presents a coherent framework for the design, interfacing, and rapid prototyping of service-oriented real-time software modules for mechatronic systems. It follows the formal principle of host-target architecture commonly used for embedded system development, in which the design and development is performed on a host system and the developed module is then transmitted, built, and executed on a remote target platform. To this end, we outline a formal but easy-to-use automated procedure for the design and rapid prototyping of real-time services that encompass several roles, such as control, device driver, and hardware interfacing, among others. The host system consists of a normal PC with Matlab/Simulink, and the target is an embedded PC with a Portable Operating System Interface (POSIX) compliant real-time Linux Operating System. Therefore, the aim is to introduce a coherent framework for remote development of independent, hard, real-time services based on Matlab and Simulink products for the rapid prototyping of software for PC-based automation.

In the proposed framework, the software modules and controllers are developed in a Simulink project on a PC with the Windows operating system, while the target is one or several embedded PCs with a POSIX-based real-time Linux. Then a standalone and hard real-time C\C++ code is generated for the real-time Linux Operating System (OS) by using the Code Generators toolbox. The code is automatically transmitted via Secure SHell (SSH) to the target, compiled, and built there. The module will then be ready to be debugged, executed, or verified on the target. This method could be used simultaneously for multiple Simulink models to develop different parts of the whole software. Then, each model can be turned into a module or service on the Target PC that collaborates with other modules by means of different internal communication services, such as programming queues. Hence, developers are able to benefit from Service Oriented Architecture (SOA) principles [66] in developing their software.

This work addresses some of the issues and difficulties commonly associated with remote development using Matlab and Simulink products. One of these is the integration of device drivers and hardware interfaces into the Simulink model. Technically, those interfaces should be turned into reusable Simulink block libraries. However, these drivers and their related API libraries are installed on the remote target system (Real-time Linux), which is different from the operating system of the development environment (Windows). Hence, deriving such blocks demands more advanced treatment. Addressing this issue requires detailed knowledge of S-Function programming, as well as the development of *Target Language Compiler (TLC)* files that tell code generators how to generate C code out of the S-Functions.

4.2.2 PVI: Mechatronic Design of a Four Wheel Steering Mobile Robot with Fault-Tolerant Odometry Feedback

In this paper, the mechatronics design and hardware architecture of the iMoro mobile manipulator is presented in detail. The goal is to design a mobile manipulator that can be utilized in hazardous environments of research facilities to perform inspection and manipulation tasks. We start by introducing several key requirements that should be considered and presenting how those requirements have been met in the final design.

Consequently, the mechanical structure and electrical interfaces are described and a low-level software architecture based on an embedded PC-based control is designed that enables the WMR to operate all of its actuators synchronously. The design is based on a modular principle, in which each wheel is a separate module consisting of a driving and a steering actuator.

In the second part of the paper, we derive the kinematic model of WMRs with active steering wheels and utilize it to derive the forward and inverse kinematic relations between the velocities of the base and that of the wheels. We provide a solution for synchronization of the actuators based on the forward and inverse kinematic model. Moreover, a similar formulation to estimate the robot's pose from the wheel odometry is given. The solution detects faulty measurements to achieve a better pose estimation based on wheel odometry. We also address some of the problems caused by slippage and inaccuracy in synchronization of the actuators and provide a fault-tolerant solution for the robust mapping of the velocities, that is, the kinematic constraints. The solution provided in this paper paves the road for higher-level control modules that navigate the robot around the obstacles and perform mobile manipulation.

4.2.3 PVII: System Integration for Real-Time Mobile Manipulation

This paper is the synergistic conclusion of our previous papers **PV** and **PVI**. It outlines the successful implementation of the framework proposed in **PV** for rapid prototyping of real-time controllers on the iMoro mobile manipulator. The paper discusses the challenges in designing a modular architecture for a mobile manipulator and the selected communication interfaces between the components. It details the design of real-time software services and interfacing the software and hardware components for mobile manipulation.

5 Discussion

In this research, we developed and implemented a unified framework for a high-performance motion controller for wheeled mobile robots that encompasses the criteria listed in Section 1.2. The overall concept of the controller is given in Section 3.2, the empirical evaluation of the system is presented in Section 3.3, and the mathematical details are provided in the relevant publications. Next, we summarize how we have addressed all the research questions presented in Section 1.2.

5.1 Reusability

The motion controller should be easily reconfigured and adapted for different kinematic structures, such as a differential drive, Ackerman steering, and independently steerable wheels.

As discussed in the introduction, specifically in Definition 1, the generic WMR considered in this study has a rigid base carried by n wheels. Each wheel belongs to the following types: *fixed wheels, standard steerable wheels, off-centered steerable wheels (Caster wheels), and Swedish wheels*, and it is assumed that all of these roll without any side slippage. Some of the driving wheels and the steering of steerable wheels are actuated, and the actuators provide velocity and position control.

To methodologically analyze the feasibility of the controller for various types of WMRs, we incorporated the seminal work of Campion et al. [11], which formally defines and classifies the mobility, controllability, and nonholonomy of WMRs equipped with various types of the aforementioned wheels. Their work provides a general kinematic form, which arise from all wheels abiding by the rolling/no-skidding constraints, which leads to the introduction of maneuverability by the pair: $\delta = (\delta_m, \delta_s)$, in which δ_m is the degree of mobility and δ_s is the degree of the WMR's steerability. The number of wheels, their types, and their arrangements determine δ_m and δ_s, and therefore the WMR's category. Here, δ_m represents the dimension of the tangent space of the configuration space, while δ_s corresponds to the ability to change (steer) the basis of the tangent space by means of the steering wheels. The degree of maneuverability defined as $\delta_M = \delta_m + \delta_s$, which is analogous to degrees of freedom for the mechanism, determines the total mobility of the WMR.

We have shown meticulously in **PIV** that the motion controller can be easily reconfigured and adapted for all five kinematically feasible types of WMRs, namely differential drive ($\delta = (\delta_m = 2, \delta_s = 0)$), Ackerman steering ($\delta = (\delta_m = 1, \delta_s = 1)$), independently steerable wheels ($\delta = (\delta_m = 1, \delta_s = 2)$), and so on. This is the main novelty of the proposed solution. While motion controllers vary significantly from one structure to another [26], we solve the motion control for wheeled robots in its most general form, allowing it to

be easily adapted for different types of structures. The only tunable parameters are geometrical dimensions and actuator limits. Therefore, the scheme allows the successful implementation of the hypothesis in Section 1.2. In other words, the controller has been designed by incorporating a dual-stage approach that tackles the base path-following and kinematic and nonholonomic constraints separately. In the first stage, for a mobile platform divested of the wheels' constraints, we demonstrated a generic path-following controller that plans asymptotic paths from the WMRs to the desired path. The second stage took into account the kinematic constraints imposed by the wheels. In this stage, we demonstrated that the designed controller can be appropriately mapped onto the wheels' control signals so that it will uphold the wheels' kinematic constraints.

5.2 High Performance

The controller should provide high performance while abiding by the velocity and acceleration bounds of all of the actuators.

Another focus of this study has been on enhancing the robot's performance while keeping the WMR's maneuverability intact. Generally, during the course of a complex maneuver, the required velocity for the wheels can be highly different than the base velocity. This is due to the nonlinearity of the kinematic constraints and can become highly problematic in designing highly maneuverable controllers. Path-following controllers that have the base speed as an exogenous input are specifically susceptible to this issue. In other words, the WMR may easily approach a configuration where the realization of the given base speed would dictate a velocity for one or more actuators that is simply out of their reach.

Another major contribution of this study is the development of a closed-form trajectory generation method that guarantees minimum time path following or achieving a desired pose while abiding by the velocity bounds of all the actuators. Unlike the approaches given only for unicycle robots [42, 43], bounded actuation can be achieved for all different kinematic structures. Again, the solution follows the hypothesis given in Section 1.2, in which the base path-following laws designed in the first stage simplifies the wheels' otherwise impenetrable kinematic and nonholonomic constraints in the second stage and transfer them into explicit proportional functions between the velocity of the platform and that of the wheels. This result enabled us to derive a closed-form trajectory generation scheme for the asymptotic path that constantly keeps the wheels' steering and driving velocities within their corresponding pre-specified bounds.

As showed in **PII**, the bounded velocity feature is highly valuable in treating steering singularities in the case of nonholonomic omnidirectional WMRs ($\delta = (\delta_m = 1, \delta_s = 2)$). As previously discussed, during certain maneuvers, as the base ICR approaches one of the wheel's steering axes, the steering velocity unboundedly increases while the driving velocity approaches zero. The previous solutions for this issue have mostly been based on numerical approaches that strive to keep the robot out of bulky regions around the singular points, which may extensively limit the robot's dexterity in performing complex motions. However, the closed-form trajectory generation in this study allows the robot to arbitrarily approaches its singular configuration by tuning the reference base speed, which in turn provides a solution that permits the robot to perform complex motions. The provided empirical analysis demonstrates the efficacy of the designed path follower and related kinematics formulations in bounding both steering and driving velocities in practice with some tolerances. Moreover, the closed-form characteristic of the solution makes it suitable for hard, real-time implementation with low process cost.

Moreover, we enhanced the Phase Plane Approach (PPA), commonly used offline for trajectory generation of serial manipulators and derived an online trajectory generation scheme that bounds the driving accelerations of the WMR. To this end, we showed in **PIII** that the closed-loop system provides a set of differential equations, independent of the base speed, for an asymptotic path that navigates the robot toward the desired path. This feature is incorporated into the trajectory generation part, as a dynamic model, and an online prediction scheme is derived that constantly evaluates the appropriate velocity profile for the asymptotic path to bound the wheels' driving accelerations and velocities.

5.3 Real-time Capabilities

The solution should be computationally deterministic and fast enough so that its implementation abides by the hard, real-time deadline.

The closed-form nature of the proposed solution allows for fast execution of the implemented algorithm. In order to properly test, verify, and configure this implementation we developed a unified framework for rapid prototyping of real-time controllers, as presented in **PV** and **PVII**. This method enables the engineers to implement the motion controller in Matlab/Simulink and generate real-time C\C++ code for real-time Linux and automatically send, compile, and execute the controller software on the platform's target PC. With the above features, this method provides a modular motion controller for wheeled mobile robots that reduces the corresponding R&D costs and time to market of developing mobile bases for service robots.

5.4 Verification and Testing

The controller should also be practically implemented and tested on various types of WMRs.

In Section 3.3, we demonstrated the successful implementation of the universal path-follower with the aforementioned features on TUT's four wheel independently steerable mobile manipulator (iMoro), depicted in Fig. 1.2, and TUT's differential drive WMR (LabRat). We used iMoro to show the successful implementation of the controller on an all wheels independently steered wheeled mobile robot. Moreover, iMoro can be configured to simulate other types of WMRs, such as Ackerman steering (car-like), and this reconfiguration has been used to demonstrate the successful implementation of the proposed method on platforms with this configuration as well.

Finally, this study aimed to demonstrate that re-configurable modular robots assisted by general and advanced motion control techniques are capable of performing a variety of high-performance operations. This capability is due to the generation of a unified framework to combine the scalability, flexibility, and modularity of mobile modules with the reliability and configurability of the proposed motion controller. It is expected that such deployment in the chosen industrial settings/domains/applications will address the limitations related to the lack of a generic motion controller for diverse modular mobile platform morphologies.

6 Conclusion and Future Works

This study presents a universal bounded-velocity path-following controller for WMRs under the assumption of pure rolling without skidding. A multistage solution, a generic form of the base control laws as proportional functions between the differentials of the states with respect to a geometric variable, and the base speed were proposed. It was shown that this format transforms the kinematic and nonholonomic constraints of the wheels into the same linear proportional structure. This result was then employed to derive a closed-form time scaling solution for the base speed that keeps the velocities of the actuators within a set of pre-specified limits.

The contributions of this study has been listed in Section 1.4, which has been expounded over the course of the seven publications listed in Chapter 4. To this end, paper **PI** provides a preliminary solution to the bounded-velocity path-following problem of WMRs with independently steering wheels. In the solution provided in this paper, bounded velocity was only achievable for small errors; in the face of large errors, the performance of the controller had to be reduced to guarantee the velocity bounds. This shortcoming has been resolved in paper **PII**, which provides a closed-form bounded velocity solution for this type of robot, regardless of the magnitude of errors, which accounts for the fifth contribution listed in Section 1.4. Then, the results of **PII** were extended in paper **PIII** to provide a bounded acceleration solution for the independently steering WMRs, representing the sixth listed contribution. Moreover, these results also extended in **PIV** to cover all categories of WMRs, realizing the first four contributions in the list. Finally, regarding the last listed contribution, which is related to hardware and software design and implementation, paper **PVI** delineates the mechatronic design of the iMoro mobile manipulator, and **PV** provides a rapid-prototyping framework for design and customization of its real-time controller software based on a service-oriented scheme.

The methods presented in this thesis are promise to be expendable to two main areas beyond the realm of WMRs. First, several path-following controllers in the same class of the proposed controller have already been extended to Three-dimensional (3D) space, representing the extension of the proposed bounded velocity path-following controller to the three dimensional field of Unmanned Aerial Vehicles(UAVs) is viable. Second, the proposed methods are expandable to the more general area of redundant nonholonomic systems, particularly Wheeled Mobile Manipulator (WMM)s. A main open area in the redundancy resolution and motion control of WMMs is the derivation of steering velocity commands. For WMMs, the redundancy is normally solved in the velocity space, but steering velocities are coupled with the accelerations of the base variables; thus, the pseudo-inverse-based redundancy resolution methods fail to provide a solution for the steering velocities [23]. However, the solution provided for the steering wheels of WMRs for the closed-loop system, which links the steering velocities to the base speed and not its acceleration, is currently being utilized to solve the open problem of mobile manipulation

for WMMs with active steering wheels.

Bibliography

[1] S. Abourida and J. Belanger. Real-time platform for the control prototyping and simulation of power electronics and motor drives. In *Proceedings of the Third International Conference on Modeling, Simulation and Applied Optimization*, pages 1 – 6, 2009.

[2] Michele Aicardi, Giuseppe Casalino, Antonio Bicchi, and Aldo Balestrino. Closed loop steering of unicycle like vehicles via lyapunov techniques. *Robotics & Automation Magazine, IEEE*, 2(1):27 – 35, 1995.

[3] Adeel Akhtar, Christopher Nielsen, and Steven L Waslander. Path following using dynamic transverse feedback linearization for car-like robots. *Robotics, IEEE Transactions on*, 31(2):269 – 279, 2015.

[4] Giuseppe Ambrosino, Marco Ariola, Umberto Ciniglio, Federico Corraro, Ettore De Lellis, and Alfredo Pironti. Path generation and tracking in 3-d for uavs. *Control Systems Technology, IEEE Transactions on*, 17(4):980 – 988, 2009.

[5] Mohammad M. Aref, R. Ghabcheloo, Antti Kolu, M. Hyvönen, K. Huhtala, and Jouni Mattila. Position-based visual servoing for pallet picking by an articulated-frame-steering hydraulic mobile machine. *IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, 2013.

[6] Thomas Bak and Hans Jakobsen. Agricultural robotic platform with four wheel steering for weed detection. *Biosystems Engineering*, 87(2):125 – 136, 2004.

[7] Marco Bibuli, Gabriele Bruzzone, Massimo Caccia, and Lionel Lapierre. Path-following algorithms and experiments for an unmanned surface vehicle. *Journal of Field Robotics*, 26(8):669 – 688, 2009.

[8] James E Bobrow, Steven Dubowsky, and JS Gibson. Time-optimal control of robotic manipulators along specified paths. *The International Journal of Robotics Research*, 4(3):3 – 17, 1985.

[9] Johann Borenstein. Control and kinematic design of multi-degree-of freedom mobile robots with compliant linkage. *Robotics and Automation, IEEE Transactions on*, 11 (1):21 – 35, 1995.

[10] Roberto Bucher and Silvano Balemi. Rapid controller prototyping with Matlab/Simulink and linux. *Control Engineering Practice*, 14(2):185 – 192, 2006. ISSN 09670661. Compendex.

[11] Guy Campion, Georges Bastin, and Brigitte D'Andréa-Novel. Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 12(1):47, 1996.

[12] J.M.P. Cardoso, P. Diniz, M.P. Monteiro, J.M. Fernandes, and J. Saraiva. A domain-specific aspect language for transforming matlab programs. In *Domain-Specific Aspect Language Workshop (DSAL'2010), part of AOSD*, pages 15–19, 2010.

[13] Christophe Cariou, Roland Lenain, Benoit Thuilot, and Michel Berducat. Automatic guidance of a four-wheel-steering mobile robot for accurate field operations. *Journal of Field Robotics*, 26(6-7):504–518, 2009.

[14] Anton Cervin. *Integrated Control and Real-Time Scheduling*. Department of Automatic Control, 2003.

[15] Sachin Chitta, Ioan Sucan, and Steve Cousins. Moveit![ros topics]. *IEEE Robotics & Automation Magazine*, 1(19):18–19, 2012.

[16] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[17] Christian Connette, Martin Hagele, and Alexander Verl. Singularity-free state-space representation for non-holonomic, omnidirectional undercarriages by means of coordinate switching. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4959–4965. IEEE, 2012.

[18] Christian Pascal Connette, Andreas Pott, M Hagele, and Alexander Verl. Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an icm representation in spherical coordinates. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 4976–4983. IEEE, 2008.

[19] Christian Pascal Connette, Christopher Parlitz, M Hagele, and Alexander Verl. Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 4124–4130. IEEE, 2009.

[20] C.P. Connette, A. Pott, M. Hägele, and A. Verl. Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4775–4781. IEEE, 2010.

[21] Steve Cousins. Ros on the pr2 [ros topics]. *Robotics & Automation Magazine, IEEE*, 17(3):23–25, 2010.

[22] EA Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5):233–249, 2000.

[23] Alessandro De Luca, Giuseppe Oriolo, and Paolo Robuffo Giordano. Kinematic control of nonholonomic mobile manipulators in the presence of steering wheels. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1792–1798. IEEE, 2010.

[24] Clarence W De Silva. *Mechatronics: an integrated approach*. CRC press, 2004.

[25] Frederik Debrouwere, Wannes Van Loock, Goele Pipeleers, Quoc Tran Dinh, Moritz Diehl, Joris De Schutter, and Jan Swevers. Time-optimal path following for robots with convex–concave constraints using sequential convex programming. *Robotics, IEEE Transactions on*, 29(6):1485–1495, 2013.

[26] M Deng, A Inoue, K Sekiguchi, and L Jiang. Two-wheeled mobile robot motion control in dynamic environments. *Robotics and Computer-Integrated Manufacturing*, 26(3):268–272, 2010.

[27] Markus Deppe, M Zanella, Michael Robrecht, and Wolfram Hardt. Rapid prototyping of real-time control laws for complex mechatronic systems: a case study. *Journal of Systems and Software*, 70(3):263–274, 2004.

[28] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger. Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 6136–6142. IEEE, 2011.

[29] Alexander Dietrich, Thomas Wimbock, Alin Albu-Schaffer, and Gerd Hirzinger. Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom. *Robotics & Automation Magazine, IEEE*, 19(2):20–33, 2012.

[30] P Encarnaçao and A Pascoal. 3d path following for autonomous underwater vehicle. In *Proc. 39 th IEEE Conference on Decision and Control*. Citeseer, 2000.

[31] Thor Fossen, Kristin Y Pettersen, Roberto Galeazzi, et al. Line-of-sight path following for dubins paths with adaptive sideslip compensation of drift forces. *Control Systems Technology, IEEE Transactions on*, 23(2):820–827, 2015.

[32] Julien Frémy, François Ferland, Michel Lauria, and François Michaud. Force-guidance of a compliant omnidirectional non-holonomic platform. *Robotics and Autonomous Systems*, 2014.

[33] Manfred Glesner, Andreas Kirschbaum, Frank-Michael Renner, and Burkart Voss. State-of-the-art in rapid prototyping for mechatronic systems. *Mechatronics*, 12(8): 987–998, 2002.

[34] Birgit Graf, Ulrich Reiser, M Hagele, Kathrin Mauz, and Peter Klein. Robotic home assistant care-o-bot® 3-product vision and innovation platform. In *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*, pages 139–144. IEEE, 2009.

[35] Brad Hamner, Seth Koterba, Jane Shi, Reid Simmons, and Sanjiv Singh. An autonomous mobile manipulator for assembly tasks. *Autonomous Robots*, 28(1): 131–149, 2010.

[36] Uwe D Hanebeck, Nihad Saldic, and GK Schmidt. A modular wheel system for mobile robot applications. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 1, pages 17–22. IEEE, 1999.

[37] D. Henriksson, A. Cervin, and K.E. Årzén. Truetime: Real-time control system simulation with matlab/simulink. In *Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark*, 2003.

[38] Robert Holmberg and Oussama Khatib. Development and control of a holonomic mobile robot for mobile manipulation tasks. *The International Journal of Robotics Research*, 19(11):1066 – 1074, 2000.

[39] Chih-Lyang Hwang and Hsiu-Ming Wu. Trajectory tracking of a mobile robot with frictions and uncertainties using hierarchical sliding-mode under-actuated control. *IET Control Theory & Applications*, 7(7):952 – 965, 2013.

[40] Isaac Kaminer, António Pascoal, Enric Xargay, Naira Hovakimyan, Chengyu Cao, and Vladimir Dobrokhodov. Path following for small unmanned aerial vehicles using l1 adaptive augmentation of commercial autopilots. *Journal of guidance, control, and dynamics*, 33(2):550 – 564, 2010.

[41] Kiattisin Kanjanawanishkul and Andreas Zell. Path following for an omnidirectional mobile robot based on model predictive control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3341 – 3346. IEEE, 2009.

[42] L Lapierre and G Indiverri. Path-following control of a wheeled robot under actuation saturation constraints. In *Proceedings of the IAV'07 conference*, 2007.

[43] Lionel Lapierre and Rene Zapata. A guaranteed obstacle avoidance guidance system. *Autonomous Robots*, 32(3):177 – 187, 2012.

[44] Lionel Lapierre, D Soetanto, and Antonio Pascoal. Nonsingular path following control of a unicycle in the presence of parametric modelling uncertainties. *International Journal of Robust and Nonlinear Control*, 16(10):485 – 503, 2006.

[45] Lionel Lapierre, Rene Zapata, and Pascal Lepinay. Combined path-following and obstacle avoidance control of a wheeled robot. *The International Journal of Robotics Research*, 26(4):361 – 375, 2007.

[46] Ciprian Lapusan, Vistrian Maties, Radu Balan, and Olimpiu Hancu. Rapid control prototyping in design process of mechatronic systems. *Solid State Phenomena*, 166-167:247 – 252, September 2010. ISSN 1662-9779. doi: 10.4028/www.scientific.net/SSP.166-167.247.

[47] Edward M Marszal and Eric William Scharpf. *Safety Integrity Level Selection: Systematic Methods Including Layer of Protection Analysis*. ISA, 2002.

[48] Alain Micaelli and Claude Samson. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. Technical Report 2097, INRIA, 1993.

[49] Kevin L Moore and Nicholas S Flann. A six-wheeled omnidirectional autonomous mobile robot. *Control Systems, IEEE*, 20(6):53 – 66, 2000.

[50] Kevin L Moore, Morgan Davidson, Vikas Bahl, Shayne Rich, and Stephan Jirgal. Modelling and control of a six-wheeled autonomous robot. In *American Control Conference, 2000. Proceedings of the 2000*, volume 3, pages 1483 – 1490. IEEE, 2000.

[51] Hadi Moradi, Kei Kawamura, Erwin Prassler, Giovanni Muscato, Paolo Fiorini, Takao Sato, and Rares Rusu. Service robotics (the rise and bloom of service robots)[tc spotlight]. *Robotics & Automation Magazine, IEEE*, 20(3):22 – 24, 2013.

[52] Yoshikazu Mori, Eiji Nakano, and Takayuki Takahashi. Mechanism, control and design methodology of the nonholonomic quasi-omnidirectional vehicle “odv9". *The International Journal of Robotics Research*, 21(5-6):511 – 525, 2002.

[53] Pieter J. Mosterman, Sameer Prabhu, Andrew Dowd, John Glass, Tom Erkkinen, John Kluza, and Rohit Shenoy. Embedded real-time control via matlab, simulink, and xpc target. In Dimitrios Hristu-Varsakelis, William S. Levine, and William S. Levine, editors, *Handbook of Networked and Embedded Control Systems*, Control Engineering, pages 419 – 446. Birkhuser Boston, 2005. ISBN 978-0-8176-4404-8.

[54] Paul Moubarak and Pinhas Ben-Tzvi. Modular and reconfigurable mobile robotics. *Robotics and Autonomous Systems*, 60(12):1648 – 1663, 2012.

[55] Patrick F Muir and Charles P Neuman. Kinematic modeling of wheeled mobile robots. *Journal of robotic systems*, 4(2):281 – 340, 1987.

[56] Derek R Nelson, D Blake Barber, Timothy W McLain, and Randal W Beard. Vector field path following for miniature air vehicles. *Robotics, IEEE Transactions on*, 23 (3):519 – 529, 2007.

[57] KyuCheol Park, Hakyoung Chung, and Jang Gyu Lee. Point stabilization of mobile robots via state-space exact feedback linearization. *Robotics and Computer-Integrated Manufacturing*, 16(5):353 – 363, 2000.

[58] L Peterson, David Austin, and Danica Kragic. High-level control of a mobile manipulator for door opening. In *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, volume 3, pages 2333 – 2338. IEEE, 2000.

[59] G. Quaranta and P. Mantegazza. Using matlab-simulink rtw to build real time control applications in user space with rtailxrt. In *Realtime Linux Workshop. Milano*. Citeseer, 2001.

[60] 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*, volume 3, page 5, 2009.

[61] David B Reister and MA Unseren. Position and constraint force control of a vehicle with two or more steerable drive wheels. *Robotics and Automation, IEEE Transactions on*, 9(6):723 – 731, 1993.

[62] Claude Samson. Time-varying feedback stabilization of car-like wheeled mobile robots. *The International journal of robotics research*, 12(1):55 – 64, 1993.

[63] Claude Samson. Motion control of wheeled mobile robots. In *Springer Handbook of Robotics*, pages 799 – 826. Springer, 2008.

[64] Ulrich Schwesinger, Cedric Pradalier, and Roland Siegwart. A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5360 – 5366. IEEE, 2012.

[65] Majura F Selekwa and Jonathan R Nistler. Path tracking control of four wheel independently steered ground robotic vehicles. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 6355–6360. IEEE, 2011.

[66] W. Shen, Q. Hao, S. Wang, Y. Li, and H. Ghenniwa. An agent-based service-oriented integration architecture for collaborative intelligent manufacturing. *Robotics and Computer-Integrated Manufacturing*, 23(3):315–325, 2007.

[67] Kang Shin and N McKay. Minimum-time control of robotic manipulators with geometric path constraints. *Automatic Control, IEEE Transactions on*, 30(6):531–541, 1985.

[68] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[69] D. Soetanto, L. Lapierre, and A. Pascoal. Adaptive, non-singular path-following control of dynamic wheeled robots. In *Proceedings of 42nd IEEE Conference on Decision and Control (CDC)*, volume 2, pages 1765–1770. IEEE, 2003.

[70] Jae-Bok Song and Kyung-Seok Byun. Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels. *Journal of mechanical science and technology*, 23(10):2747–2756, 2009.

[71] PB Sujit, Srikanth Saripalli, and Joao Borges Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *Control Systems, IEEE*, 34(1):42–59, 2014.

[72] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[73] B. Thuilot, B. d'Aandrea Novel, and A. Micaelli. Modeling and feedback control of mobile robots equipped with several steering wheels. *Robotics and Automation, IEEE Transactions on*, 12(3):375–390, 1996.

[74] James P Trevelyan, Sung-Chul Kang, and William R Hamel. Robotics in hazardous applications. In *Springer Handbook of Robotics*. Springer, 2008.

[75] J. van Amerongen. Mechatronic design. *Mechatronics*, 13(10):1045–1066, 2003.

[76] Wannes Van Loock, Goele Pipeleers, Moritz Diehl, Joris De Schutter, and Jan Swevers. Optimal path following for differentially flat robotic systems through a geometric problem formulation. *Robotics, IEEE Transactions on*, 30(4):980–985, 2014.

[77] Stefano Vitturi. Pc-based automation systems: an example of application for the real-time control of blowing machines. *Computer Standards & Interfaces*, 26(2): 145–155, 2004.

[78] Dong Xu, Dongbin Zhao, and Jianqiang Yi. Dynamic model and control for an omnidirectional mobile manipulator. In *Robotic Welding, Intelligence and Automation*, pages 21–30. Springer, 2007.

[79] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum. *Building embedded Linux systems*. O'Reilly Media, 2008.

[80] Takeshi Yamasaki and SN Balakrishnan. Sliding mode-based pure pursuit guidance for unmanned aerial vehicle rendezvous and chase with a cooperative aircraft. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(10):1057 – 1067, 2010.

[81] Kamal Youcef-Toumi. Modeling, design, and control integration: a necessary step in mechatronics. *Mechatronics, IEEE/ASME Transactions on*, 1(1):29 – 38, 1996.

[82] Yuan-Fang Zheng. *Recent trends in mobile robots*, volume 11. World scientific, 1993.

Publications

Publication I

Oftadeh, Reza, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Bounded-velocity motion control of four wheel steered mobile robots." In *Advanced Intelligent Mechatronics (AIM), 2013 IEEE/ASME International Conference on*, pp. 255-260. IEEE.

Bounded-Velocity Motion Control of Four Wheel Steered Mobile Robots

Reza Oftadeh, Mohammad M. Aref, Reza Ghabcheloo, Jouni Mattila

***Abstract*— In this paper, we address the problem of motion control for a mobile robot with four independently steer and drive wheels. Our solution fully takes advantage of steerability of all wheels and provide capability of independent control of translation and rotation of the robot. Using non-linear control techniques, we provide a motion control law that makes the base follow a given desired smooth path and heading profile. Derivation of motion control is three folded. Assuming velocity vector orientation and angular velocity of the base as control signals, control laws are derived to solve the path and heading profile following problem. Then these control signals are mapped to eight actuator signals (driving and steering). In a later stage, robot base speed magnitude is controlled in such a way to keep the control signals under predefined limits. Simulations as well as experiment on a real robot show the efficacy of the proposed method.**

I. Introduction

Four Wheel Steerable (4WS) mobile robot has again recently gained attention of many researchers due to increased demands for high maneuverability and flexibility in uneven floor. Our 4WS mobile robot called *iMoro* is shown in Fig. 1. It has become known that motion control of such vehicles with eight degrees of freedom is a challenging task [1]. The difficulty stems from the kinematic constraints among the degrees of freedom. Thus, a proper motion control demands for a close coordinated motion of 8 actuators, while considering actuators saturation and limited bandwidth in both steering and driving mechanisms. In order to simplify motion control and synchronization of actuators, steering angles have been constrained by their steering mechanism [2], [3], or control strategies has been employed that keep the wheels parallel to each other [4] suitable for crab steering. These solutions can solve conventional over-steering or under-steering problems of car-like robots but it decreases robot maneuverability. However, an efficient solution that enables the robot to maneuver in limited space require independent steering of each wheel leg [4], [5].

Other simplifying but limiting solution is the use of traditional bicycle model [6] or other mappings determining motion of the robot with respect to its *Instantaneous Center of Rotation* (ICR) [7]. In these solutions, position of ICR and angular velocity of the body describe robots behavior during path following. Many comprehensive research works were performed during recent years on motion planning and control of 4WS using ICR [8], [9]. In these works, first motion is calculated for ICR and using kinematic relations, motion of ICR is then mapped to joint space to command steering and driving actuators. This way, coordination among degrees of freedom is achieved and wheel slippage is significantly reduced.

In this paper, it is shown that the robot is capable of performing independent translation and heading motion, which is widely ignored in the literature, and a motion control solution is devised to exploit this capability. This motion control is used for example in [10] for sensor planning. Only few researches have considered it as an option for enhancement of robot motion [11]. Utilizing ICR based calculations makes linear and angular velocities mathematically coupled to each other. Therefore, even with recent accomplishments in this range, for instance [4], motion planning and control for independent heading and linear velocity of the robot main body is unheeded. Alternatively, mapping of the velocities can be done by kinematic relations between velocity twists in work space and joint space [12]. We use this method to extract consistent twist of driving velocity and steering angle for each leg.

In contrast to [9], [13], where ICR together with a predictive control is used to address actuator saturation, we employ path curvature analysis to determine velocity boundaries for each leg. Using nonlinear control strategies, we propose a solution to motion control of a 4WS, which addresses aforementioned limitations on all eight degrees of freedom. Our approach is based on a path following method where rate of progress on the path becomes an free variable [14]. This free variable is then used to control evolution of the

Fig. 1. **iMoro** mobile platform

Authors are with the Department of Intelligent Hydraulics and Automation(IHA) , Tampere University of Technology, 33101, Finland.
Corresponding author's email: oftadeh.reza@ieee.org.

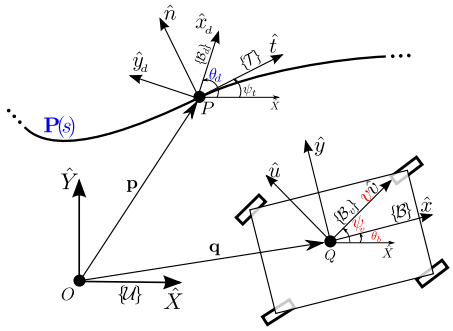


Fig. 2. Desired path and the required coordinate frames

states such that the actuators satisfy velocity bounds. We also provide an analytical formula for velocity bounds as functions of actuators limitations and path curvature of each wheel-ground contact point.

The paper is organized as follows. In Section II, we describe the general architecture of the robot and introduce the notations and symbols. Next, we define the problem in hand and the error space to which the path following and heading control are mapped. In section IV, we describe the solution, and the pertinent kinematic constrains. We address actuator limitations and velocity bounds in Section V-B. In the last section, we show efficacy of the proposed solution through simulations and real experiments with iMoro.[15].

II. General Descriptions

A. Robot's Architecture

iMoro, shown in Fig.1, consists of a rigid base and four legs, each of which has two degrees of freedom(DOF). In fact, each leg is equipped with two independent servo drives for steering and driving. The steering actuator rotates the whole leg along its vertical axis, hence determining the heading of that wheel. The driving actuator rotates the respective wheel along its normal axis. Furthermore, for each leg, the wheel's contact point with the ground coincides with the leg's vertical axis. The robot is equipped with off-the-shelf control drivers that provide velocity and position control options.

B. Notations and Symbols

We denote the vectors in **bold** with hat-sign($\hat{}$) for unit vectors. A left superscript to a vector denotes the frame in which it is expressed, except for the inertial frame, which is omitted for the sake of brevity. Moreover, for two arbitrary vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a}.\boldsymbol{b}$ and $\boldsymbol{a} \times \boldsymbol{b}$ are the inner and cross products of the two vectors, respectively.

Fig. 2 depicts a schematic view of the platform together with a desired path. The coordinate frame $\boldsymbol{\mathcal{U}}\{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\}$ is the inertial frame. Frame $\boldsymbol{\mathcal{B}}\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ is a fixed-body frame defining the heading of the robot, and $\boldsymbol{\mathcal{B}}_v\{\hat{\boldsymbol{v}}, \hat{\boldsymbol{u}}\}$ is the velocity frame. Unit vector $\hat{\boldsymbol{v}}$ determines the direction of the robot's base linear velocity vector with scalar v being its magnitude. Both $\boldsymbol{\mathcal{B}}$ and $\boldsymbol{\mathcal{B}}_v$ are attached to the robot's base at point $\boldsymbol{Q}$ which can be chosen at will. Angles ψ_v and θ_b are the angle of $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{x}}$, respectively, in $\boldsymbol{\mathcal{U}}$. The tangent frame $\boldsymbol{\mathcal{T}}\{\hat{\boldsymbol{t}}, \hat{\boldsymbol{n}}\}$ is the Serret-Frenet frame at point $\boldsymbol{P}$ attached to the desired path $\boldsymbol{P}_d(s)$ and the angle of $\hat{\boldsymbol{t}}$ in $\boldsymbol{\mathcal{U}}$ is denoted as ψ_t. To determine the desired heading, we define frame $\boldsymbol{\mathcal{B}}_d\{\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{y}}_d\}$ attached to $\boldsymbol{P}$ such that the angle of $\hat{\boldsymbol{x}}_d$ in $\boldsymbol{\mathcal{U}}$ which is θ_d, equals the desired heading function $\theta_d(s)$. Note that both desired path $\boldsymbol{P}_d(s)$ and desired heading $\theta_d(s)$ are parametrized with the same parameter s. Moreover, vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ define points $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively, in $\boldsymbol{\mathcal{U}}$.

III. Problem Definition

The desired path $\boldsymbol{P}_d(s)$ is assumed to be a regular curve with bounded curvature on a horizontal plane. It is defined by a vector-valued function $\boldsymbol{P}_d : [0, L_P] \rightarrow \boldsymbol{R}^2$, where s and L_P are natural parametrization and length of the path, respectively. The curvature of $\boldsymbol{P}_d(s)$ which is a function of s is denoted as $C_c(s)$. Moreover, we assume that the desired heading function $\theta_d(s) : [0, L_P] \rightarrow \boldsymbol{R}$ is two times differentiable function of s.

Let us define following error signals,

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = {}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}^{-1}(\boldsymbol{q} - \boldsymbol{p}) \tag{1a}$$

$$\theta_e = \theta_d - \theta_b \tag{1b}$$

$$\psi_e = \psi_t - \psi_v, \tag{1c}$$

where ${}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}$ is the rotation matrix $\boldsymbol{R}(\psi_t)$ that defines the rotation from frame $\boldsymbol{\mathcal{T}}$ to the frame $\boldsymbol{\mathcal{U}}$. In this case the error signals x_e and y_e are position errors measured along $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{n}}$, respectively.

Time derivative of eqs. (1a) and (1b) results in,

$$\dot{x}_e = \dot{s}(C_c(s)y_e - 1) + v\cos(\psi_e) \tag{2a}$$

$$\dot{y}_e = -\dot{s}C_c(s)x_e - v\sin(\psi_e) \tag{2b}$$

$$\dot{\theta}_e = \frac{\partial \theta_d}{\partial s}\dot{s} - \omega_b \tag{2c}$$

in which, $\omega_b = \dot{\theta}_b$ is the angular velocity of $\boldsymbol{\mathcal{B}}_v$. Note that $\dot{\psi}_T = C_c\dot{s}$.

Problem 1: Given desired path $\boldsymbol{P}_d(s)$ and heading profile $\theta_d(s)$, derive *feedback control laws* for the speed of each wheel and the steering angle of each leg such that,

1) Path following: frame $\boldsymbol{\mathcal{B}}_v$ converges and follows frame $\boldsymbol{\mathcal{T}}$, that is, error signals x_e, y_e and ψ_e remain bounded and converge to zero. See eqs. (1a) and (1c).
2) Heading control: frame $\boldsymbol{\mathcal{B}}$ converges and follows frame $\boldsymbol{\mathcal{B}}_d$, that is, error signal θ_e remain bounded and converge to zero. See (1b).
3) Bounded control signal: rate of steering and speed of the wheels do not exceed predefined actuator bounds.

We derive our solution to this problem based on the following assumptions. The robot moves on a flat and horizontal plane. The base and the legs are rigid and the wheels are non deformable. Furthermore, the condition of pure rolling without any side slippage is assumed for the wheels. Notice that desired path $\boldsymbol{P}_d(s)$ and desired heading $\theta_d(s)$ are for example generated by a path planner to achieve certain objectives as described in the introduction.

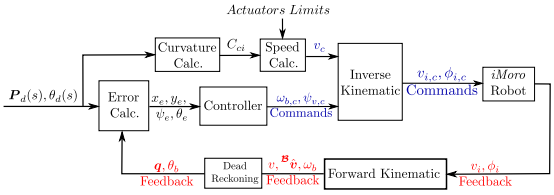


Fig. 3. Schematic block diagram of the whole system

IV. Derivation of Control Laws

In this section, angular velocity ω_b and base linear velocity direction $\hat{\boldsymbol{v}}$ are considered to be control inputs to satisfy the first two conditions of Problem 1 which are the *Path* and *Heading Following*. To satisfy the third requirement, in the next section, the base linear speed v is selected to constrain the velocity of the actuators. Finally, forward and inverse kinematics relations in velocity space are used to relate base linear and angular velocities to the wheels speeds and steering angles. Fig. 3 schematically depicts the block diagram of the whole procedure. We explain those blocks in following sections.

A. Controller Design

Consider the error states eqs. (1a) to (1c). The control objective is to derive feedback control laws for ψ_v, and $\dot{s}$ and ω_b such that $x_e, y_e, \theta_e, \psi_e$ asymptotically converge to zero. First, define functions $\mathrm{sat}(x)$, σ and a simple saturation function $\mathrm{sat}(x)$ as,

$$\sigma(y_e) = -\operatorname{sign}(v)\sin^{-1}\frac{k_2 y_e}{|y_e| + \epsilon} \tag{3a}$$

$$\mathrm{sat}(x, x^{sat}) = \begin{cases} \operatorname{sign}(x)x^{sat} & |x| > x^{sat} \\ x & \text{otherwise} \end{cases} \tag{3b}$$

where, $0 < k2 \leq 1$ and $\epsilon > 0$. For the sack of brevity, $\mathrm{sat}(x, x^{sat})$ is referred to by $\mathrm{sat}(x)$ throughout the paper.

Proposition 1: The feedback control laws given by,

$$\dot{s} = k_1 \,\mathrm{sat}(x_e) + v\cos\sigma(y_e) \tag{4a}$$

$$\omega_{b,c} = k_3 \,\mathrm{sat}(\theta_e) + \frac{\partial\theta_d}{\partial s}\dot{s} \tag{4b}$$

$$\psi_{v,c} = \psi_t - \sigma(y_e) \tag{4c}$$

along with the condition $|v(t)| \geq v_m > 0$ solves the first two conditions of Problem 1.

Proof: Lyapunov base techniques similar to [16] and [17] are used to derive the control laws. Consider the following Lyapunov function,

$$V_P = \frac{1}{2}x_e^2 + \frac{1}{2}y_e^2 + \frac{1}{2}\theta_e^2 \tag{5}$$

which is positive definite and radially unbounded. The derivative of V_P along eqs. (2a) to (2c) result in,

$$\dot{V}_P = -k_1 x_e \,\mathrm{sat}(x_e) - k_2|v(t)|\frac{y_e}{|y_e| + \epsilon} - k_3\theta_e \,\mathrm{sat}(\theta_e) \tag{6}$$

where we use eqs. (4a) to (4c). Under condition; $|v(t)| \geq v_m > 0$, $\dot{V}_P$ is negative definite. Thus, the origin is globally asymptotically stable and it is locally exponentially stable

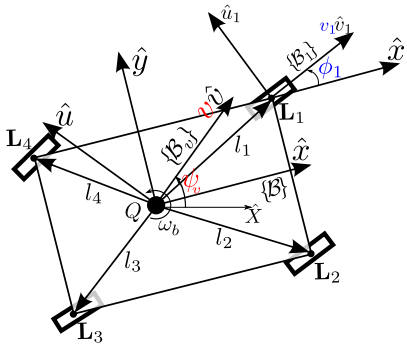


Fig. 4. Denoted kinematic parameters

when the error signals are inside saturation bounds. More details on this can be found in [18]. Note that substituting ψ_v from (4c) into (1c) results in $\psi_e = \sigma(y_e)$. Hence, ψ_e is bounded and when y_e converges to zero ψ_e would converge to zero. Functions $\mathrm{sat}(x_e)$ and $\mathrm{sat}(\theta_e)$ saturate the error signals in ranges defined with $[-x_e^{sat}, x_e^{sat}]$ and $[-\theta_e^{sat}, \theta_e^{sat}]$, respectively. In the next section, these saturating limits along with independent speed variable v are utilized to properly bound the actuator velocities. ■

Note that in this approach the natural parameter s is an auxiliary state of the system and the solution of (4a).

B. Kinematics Constrains

In Fig. 4 the necessary variables are defined. Base on the figure, ${}^{\mathcal{B}}\boldsymbol{\ell}_i$, $i \in \{1, 2, 3, 4\}$ are constant vectors related to the geometry of the robot. ϕ_i is the heading angle of the i^{th} leg while $v_i\hat{\boldsymbol{v}}_i$ is the velocity vector of the attachment point $\boldsymbol{L}_i$. The vector is ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ can be written in the form of $[\cos(\phi_i)\ \ \sin(\phi_i)\ \ 0]^T$.

As described in Section II, the input value for steering and driving actuators of the leg i are ϕ_i and v_i, respectively. The following kinematic relations map the base velocity space to the legs velocities.

$${}^{\mathcal{B}}\hat{\boldsymbol{v}} = \boldsymbol{R}(\psi_v - \theta_b)[1\ 0\ 0]^T \tag{7a}$$

$$v_i\ {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = v\ {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) \tag{7b}$$

in which, $\hat{\boldsymbol{z}} = [0\ 0\ 1]^T$ and $\boldsymbol{R}(\psi_v - \theta_b)$ is the rotation matrix with angle $\psi_v - \theta_b$ around z-axis, that is, frame $\boldsymbol{\mathcal{B}}_v$ in $\boldsymbol{\mathcal{B}}$. Equation (7b) is the simplest relation between the base and legs parameters in which no time derivative of steering angle ϕ_i is present in the equation. This fact justifies having steering actuators to be controlled in position mode. Note that here it is assumed that the leg's steering is *multi-turn* and $v_i > 0, \forall i$.

Based on Fig. 3, given base commands $\psi_{v,c}$, $\omega_{b,c}$, and v_c, equation (7b) is utilized to derive wheels commands $\phi_{i,c}$ and $v_{i,c}$. Furthermore, the equation can be used to solve forward velocity kinematics which falls in to the category of *Localization based on Wheel Odometry*. The details are given by authors in [15].

V. Bounded Velocity Solution

In this section, we will derive upper-bounds for v_c, the command signal for v, to limit the control signals to pre-specified actuators velocity bounds. We will address this issue using curvature of nominal paths associated to ground contact points or points $\boldsymbol{L}_i, i = 1, ..., 4$ shown in Fig. 4. In iMoro, points $\boldsymbol{L}_i$ coincide with leg's vertical rotation axis.

Based on constraint imposed by kinematics, it can be shown that when the ICR approaches a leg's vertical axis, speed of that wheel necessarily decreases and its steering velocity increases. When ICR and one of $\boldsymbol{L}_i$ coincide, that leg becomes *singular*, that is, wheel speed becomes zero and its angle becomes irrelevant to the movement of the base [19]. Next, we will study this phenomena using curvature of the path taken by points $\boldsymbol{L}_i$.

A. Curvature of ground contact point trajectories

Let us denote the nominal path associated to $\boldsymbol{L}_i$ by $\boldsymbol{\gamma}_i$, its first and second derivative with respect to s by $\boldsymbol{\gamma'}_i$ and $\boldsymbol{\gamma}''_i$, respectively, and its curvature by C_{ci}. The following relations hold and given s, they can be computed analytically.

$$\boldsymbol{\gamma}_i(s) = \boldsymbol{P}_d(s) + \boldsymbol{R}(\theta_d(s))\,{}^{\boldsymbol{B}}\boldsymbol{\ell}_i \tag{8a}$$

$$\boldsymbol{\gamma'}_i(s) = \boldsymbol{P'}_d(s) + \theta'_d(s)(\hat{\boldsymbol{z}} \times \boldsymbol{R}(\theta_d(s))\,{}^{\boldsymbol{B}}\boldsymbol{\ell}_i) \tag{8b}$$

$$C_{ci} = \frac{\det\left(\boldsymbol{\gamma}'_i(s), \boldsymbol{\gamma}''_i(s)\right)}{\|\boldsymbol{\gamma}'_i(s)\|^3}, \tag{8c}$$

Assuming that frame $\boldsymbol{\mathcal{B}}$ closely follows $\boldsymbol{P}_d(s)$ and $\theta_d(s)$, point $\boldsymbol{L}_i$ will follow $\boldsymbol{\gamma}_i$ defined in (8a), with curvature (8c). For a given curvature C_{ci}, wheel velocity v_i and steering velocity $\dot{\phi}_i$ of leg i are related by

$$C_{ci} = \frac{\dot{\phi}_i}{v_i}. \tag{9}$$

From (8a) and (8b), we conclude that because $\boldsymbol{P}_d(s)$ and $\theta_d(s)$ and their respective derivatives are bounded, $\boldsymbol{\gamma}_i(s)$ and its derivatives are bounded. Furthermore, from (8c) and (9), we conclude that if $\boldsymbol{\gamma'}_i(s)$ tends to zero (this happens when ICR approaches $\boldsymbol{L}_i$), C_{ci} becomes unboundedly large, and even for small values of speed v_i steering velocity becomes unbounded.

B. Derivation of Command Velocity

At a point on the path with a given curvature, only one of the drive or steering velocities of each leg may assume maximum value. Let $\dot{\Phi}^S_{max}$ and V^D_{max} be two constant scalars defining the maximum velocities for the steering and driving, respectively. Furthermore, speed of the base v is related to the speeds of each wheel v_i by (7b). Thus, at each point on the path there is a maximum value for v at which one of the wheel speeds reaches its maximum. Following procedure exploits these ideas to produce a command speed v_c that lead to steering and drive commands that are bounded.

- For each $i = 1, 2, 3, 4$:
  1) Evaluate $V^S_{i,max}$ from:
  $$V^S_{i,max} = \frac{\dot{\Phi}^S_{max}}{|C_{ci}|} \tag{10}$$
  2) Evaluate $V_{i,max}$ from:
  $$V_{i,max} = \min(V^S_{i,max}, V^D_{max}) \tag{11}$$
  3) Evaluate the i_{th} candidate for v_c namely v^i_c, by selecting the positive solution of
  $$v^{i\,2}_c + 2\,v^i_c\,\omega_b({}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}.(\hat{z} \times {}^{\boldsymbol{\mathcal{B}}}\boldsymbol{\ell}_i)) - V^2_{i,max} + \omega^2_b\,l^2_i = 0 \tag{12}$$
  where l_i is length ${}^{\boldsymbol{\mathcal{B}}}\boldsymbol{\ell}_i$.
- Evaluate $v_{c,max}$ from:
$$v_{c,max} = \min_i(v^i_c), i \in \{1, 2, 3, 4\} \tag{13}$$

Variables $V^S_{i,max}$ define maximum allowable speed for points $\boldsymbol{L}_i$. Using $V^S_{i,max}$, maximum speed $v_{c,max}$ of the body is then calculated. Based on (10) and (11), for curvatures $|C_{ci}| < \dot{\Phi}^S_{max}/V^D_{max}$, $V^S_{i,max}$ exceeds V^D_{max} and driving actuator sets the limit. However, as the curvature increases, $V^S_{i,max}$ reduces and steering velocity becomes more critical.

Equation (12) is derived by evaluating (7b) for $v_i = V_{i,max}$. We seek a positive solution for (12), which is a quadratic equation with unknown v^i_c. Notice that (12) has exactly one positive solution if

$$|\omega_b| < \min_i\left(\frac{|V_{i,max}|}{l_i}\right) \tag{14}$$

From (4a), (4b), ω_b can be written as

$$\omega_b = k_3\,\mathrm{sat}(\theta_e) + \frac{\partial\theta_d}{\partial s}k_1\,\mathrm{sat}(x_e) + \frac{\partial\theta_d}{\partial s}v\cos\sigma(y_e) \tag{15}$$

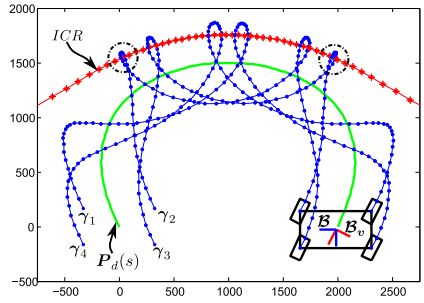
Now it is clear why we introduce the saturation function in the control laws. Moreover, in the planning phase one must make sure desired heading $\theta_d(s)$ and it derivative are designed such that w_b does not exceed bound (14).

VI. Simulation and Experimental Results

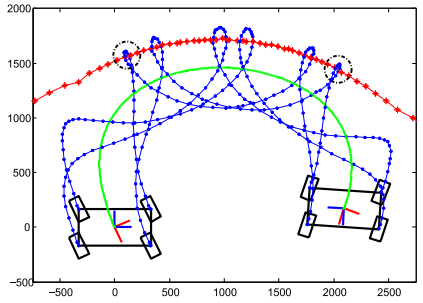
In this section, simulation and experimental results of the proposed control system are presented. Experiments are done on *iMoro*, Fig. 1, with parameters shown in Table I. The experiments were performed to show the efficacy of the algorithm and the limits mentioned in the table are considerably lower than real values due to safely and limited experimental area. Controller software is implemented in an embedded PC with real-time Linux [20].

Figures in Fig. 5 show the simulation results while Figures in Fig. 6 illustrate the corresponding experimental results for the same desired path and heading profile. Solid green line in Fig. 5(a) shows the desired path, a *Bezier* spline. The desired heading changes along the path linearly from 0 (at the beginning of the path) to 3π (at the end of the path). The footprint of points $\boldsymbol{L}_i$ (indicated by paths $\gamma_\mathbf{i}$) as well as base ICR are also portrayed in Fig. 5(a).

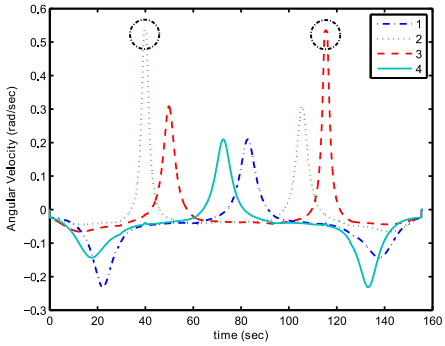
It can be seen from Fig. 5(a) that when a leg approaches the base ICR, its respective path $\boldsymbol{\gamma}_i$ shows a quick turn thus rapid increase of curvature. The two most critical cases occurs to the legs two and three. The instances are highlighted by dotted circle. Fig. 6(a) shows the actual path taken by the robot in the experiment based on wheel odometry. The figure shows that the robot properly follows the desired path and the heading profile.



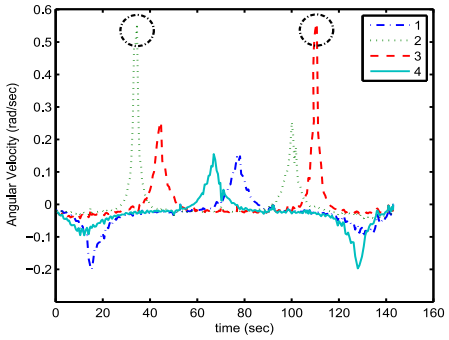
(a) Robot desired footprint



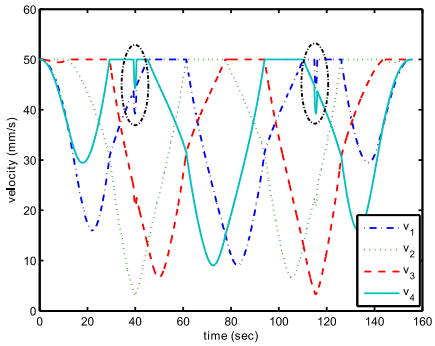
(a) Robot footprint in the experiment



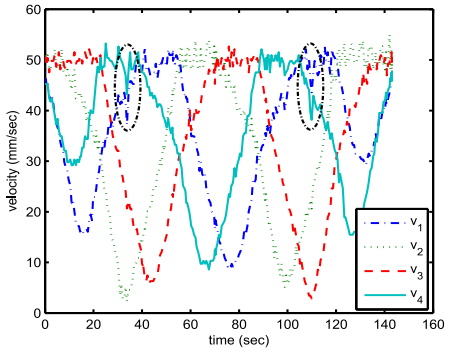
(b) Steering velocities $\dot{\phi}_i$, extracted from feedback



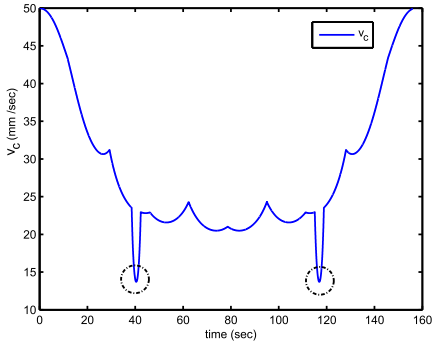
(b) Steering velocities $\dot{\phi}_i$, extracted from feedback



(c) Driving velocities feedback v_i

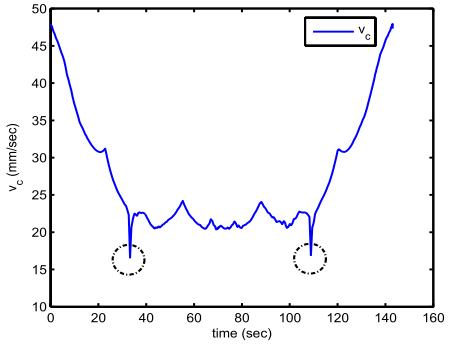


(c) Driving velocities feedback v_i



(d) Command signal for base speed v_c

Fig. 5. Simulation Results



(d) Command signal for base speed v_c

Fig. 6. Experimental Results

TABLE I
SPECIFICATIONS OF iMORO ROBOT FOR THE EXPERIMENTAL STUDIES

Description	Quantity
Main Body Length (x direction)	$655\ mm$
Main Body Width (y direction)	$335\ mm$
Maximum Velocity of Steering Servo Motors	$0.6\ ^{rad}/_{sec}$
Maximum Velocity of Driving Servo Motors	$50\ ^{mm}/_{sec}$
Wheel Diameter	$210\ mm$
Approximate Overall Mass	$120\ kg$

Fig.5(b) and Fig.5(c) illustrates the simulated steering and driving velocities of the legs, respectively. The corresponding results of the experiment are shown in Fig. 6(b) and Fig. 6(c). Notice that except some fluctuations due to robot dynamic forces, the velocity boundaries are kept in comply with the simulation results. Fig.5(b) shows that the steering velocities remain below the bound, while Fig.5(c) shows that the driving velocities are saturated to the bound. Extreme steering instances are highlighted by circles which correspond to the points where one of the legs approaches the base ICR, see Fig. 5(b).

Fig.5(d) and Fig.6(d) show the maximum allowable speed command in the simulation and the experiment, respectively. The figures show how the propose algorithm reduces the command speed (particularly sharply at critical areas) and thus retain the steering velocities below the prespecified limit. It can be noticed from the figures that at each point in time, one actuator is performing at its limit. When the legs are far from the base ICR, the limit is determined by driving actuators. When a leg gets close to the base ICR, the limit is set by the steering actuator.

VII. CONCLUSION

This paper addresses a stable and practical solution for the challenging problem of four wheel steerable (4WS) robots' motion control. In contrast with previous works, instead of using instantaneous center of rotation (ICR) in the calculations, path curvature of each wheel is used to bound the velocity of the actuators. Effectiveness of this method examined through simulation and experimental evaluation. The experiments show that even without considering any model-based approach or dynamic analysis, the designed path follower and related kinematics formulations are capable of bounding both steering and driving velocities in practice with some tolerances. The Since this approach gives an analytical solution, its computational costs are considerably low in real-time implementations.

VIII. ACKNOWLEDGMENTS

This work, supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network, was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA grant agreement number 264336.

REFERENCES

[1] K. Berns, K. Kuhnert, and C. Armbrust, "Off-road robotics-an overview," *KI-Kunstliche Intelligenz*, vol. 25, no. 2, pp. 109–116, 2011.
[2] P. Santana, J. Barata, and L. Correia, "Sustainable robots for humanitarian demining," *International Journal of Advanced Robotic Systems*, vol. 4, no. 2, pp. 207–218, 2007.
[3] M. Vilaplana, O. Mason, D. Leith, and W. Leithead, "Control of yaw rate and side slip in 4-wheel steering cars with actuator constraints," vol. 3355, pp. 201–222, 2005.
[4] M. Selekwa and J. Nistler, "Path tracking control of four wheel independently steered ground robotic vehicles," in *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pp. 6355–6360, IEEE, 2011.
[5] C. Connette, A. Pott, M. Hägele, and A. Verl, "Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4775–4781, IEEE, 2010.
[6] D. Wang and F. Qi, "Trajectory planning for a four-wheel-steering vehicle," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 4, pp. 3320–3325, IEEE, 2001.
[7] B. Thuilot, B. d'Aandrea Novel, and A. Micaelli, "Modeling and feedback control of mobile robots equipped with several steering wheels," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 375–390, 1996.
[8] E. Godoy, G. Tangerino, R. Tabile, R. Inamasu, and A. Porto, "Networked control system for the guidance of a four-wheel steering agricultural robotic platform," *Journal of Control Science and Engineering*, vol. 2012, p. 4, 2012.
[9] P. Falcone, F. Borrelli, J. Asgari, H. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.
[10] M. Lauri and R. Ritala, "Receding horizon control for selection of focus of attention," in *Proceedings of The 20th IMEKO World Congress, Busan, Republic of Korea*, 2012.
[11] C. Cariou, R. Lenain, B. Thuilot, and M. Berducat, "Automatic guidance of a four-wheel-steering mobile robot for accurate field operations," *Journal of Field Robotics*, vol. 26, no. 6-7, pp. 504–518, 2009.
[12] T. Lam, H. Qian, Y. Xu, and G. Xu, "Omni-directional steer-by-wire interface for four wheel independent steering vehicle," in *IEEE International Conference on Robotics and Automation, ICRA'09.*, pp. 1383–1388, IEEE, 2009.
[13] T. Oksanen, "Path following algorithm for four wheel independent steered tractor," in *CIGRAgEng Conference Proceedings*, (Valencia, Spain), 8-12 July 2012.
[14] R. Ghabcheloo, A. Aguiar, A. Pascoal, C. Silvestre, I. Kaminer, and J. Hespanha, "Coordinated path-following in the presence of communication losses and time delays," *SIAM Journal on Control and Optimization*, vol. 48, no. 1, pp. 234–265, 2009.
[15] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback," in *6th IFAC Symposium on Mechatronic Systems*, (Hangzhou, China), pp. 663–669, IFAC, 10-12 April 2013.
[16] D. Soetanto, L. Lapierre, and A. Pascoal, "Adaptive, non-singular path-following control of dynamic wheeled robots," in *Proceedings of 42nd IEEE Conference on Decision and Control (CDC)*, vol. 2, pp. 1765–1770, IEEE, 2003.
[17] I. Kaminer, A. Pascoal, and O. Yakimenko, "Nonlinear path following control of fully actuated marine vehicles with parameter uncertainty," in *16th IFAC World Congress*, (Prague, Czech Republic), 2005.
[18] R. Ghabcheloo, *Coordinated Path Following*. PhD thesis, Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, 2007.
[19] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6136–6142, IEEE, 2011.
[20] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM),Kaohsiung, Taiwan*, pp. 274 –279, July 2012.

Publication II

Oftadeh, Reza, Reza Ghabcheloo, and Jouni Mattila. "A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels." In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pp. 4845-4851. IEEE.

A Novel Time Optimal Path Following Controller with Bounded Velocities for Mobile Robots with Independently Steerable Wheels

Reza Oftadeh, Reza Ghabcheloo, Jouni Mattila

***Abstract*— Mobile robots with independently steerable wheels possess many high maneuverability features of omnidirectional robots while benefiting from better performance and capability of moving on rough terrains. However, motion control of such robots is a challenging task due to presence of singular configurations and unboundedly large steering velocities in the neighborhood of those singularities. Many proposed approaches rely on numerical solutions that keep the robot out of bulky regions around the singular points and hence lose some of the robot maneuverability. Based on a class of traditional path followers we design a new globally stable path following controller that exploits the high maneuverability of the platform. This design allows us to derive a set of closed-form analytical functions that describe the robot base velocity as a function of the wheels driving and steering velocities while abide to the robot non-holonomic constraints. Those functions are then utilized to find the maximum instantaneous velocity of the body that keeps the wheels velocities under the pre-specified bounds no matter how much the robot gets close or far from its singular configurations. The control algorithms developed in this paper have been evaluated on *iMoro*, a four wheel independently steered mobile manipulator designed and developed at IHA/TUT. Experimental data is also shown that show efficacy of the method.**

I. Introduction

The interest in mobile robots with active steering wheels has been increasing over time. This is due to their high maneuverability and flexibility while being able to operate on rough terrains and carry higher payloads with better efficiency compared with the other types of omnidirectional mobile robots. Such platforms are now being used in developing advanced mobile robots in many practical fields such as service robotics [1], [2], agricultural tasks [3] and space applications [4]. We as a part of the PURESAFE [1] project that aims to prevent human intervention in radioactive environments have designed and developed a four wheeled independently steerable mobile manipulator (iMoro) that is shown in Fig. 1. Our goal is to take advantage of such platforms maneuverability to perform manipulation and remote inspection in the confined spaces of CERN LHC tunnel.

The platform high maneuverability comes from its omnidirectional nature. The mobile platform is able to realize any arbitrary, independent set of linear and angular velocities but only after it has initially reoriented its wheels to a predetermined corresponding configuration. Hence, while those platforms have three Degrees of Freedom (DoF) [5], they are over-actuated [6] with all the actuators should abide the non-holonomic constraints of the robot. Hence, such robots are sometimes called non-holonomic omnidirectional robots[7] or pseudo-omnidirectional robots[8], [4].

The early researches on such robots have focused more on over-constrained nature [9] and correction of wheel odometry errors for deadreconing localization [10], [11]. Advances in sensors and actuators along with sensor fusion based algorithms for localization have solved many of those early issues. The interest is now shifted to analyze and develop more sophisticated control schemes for those robots [12], [13], [14].

However, presence of singularities both inherently [15] and in the presentation of the configuration space [16] makes design and realization of motion controllers a challenging task. One of the most popular ways to describe the platform configuration space and those singularities, is with the notion of Instantaneous Center of Rotation (ICR) [15]. The ICR of the robot body is defined on the horizontal Cartesian plane and with respect to the coordinate frame attached to the body. Each wheel is following an instantaneous circle with ICR at its center. As the ICR gets close to the wheel axis the radius of that circle becomes smaller. Hence, the driving velocity of that wheel decreases while its steering velocity unboundedly increases. When the ICR coincides with the wheel steering axis the driving velocity of the wheel becomes zero and its steering angle becomes undefinable. Hence, the 2D position vector of the ICR along with the angular velocity around it can serve as the state space for

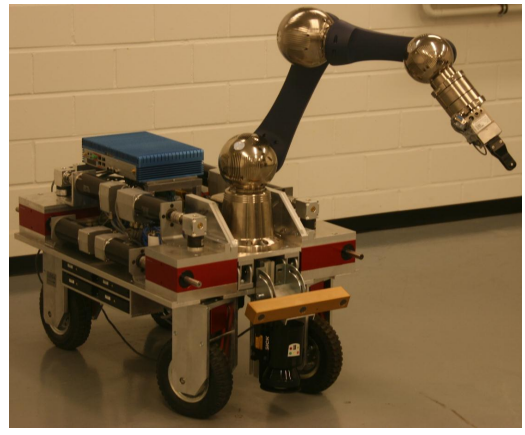


Fig. 1. The iMoro Mobile Manipulator

Authors are with Intelligent Hydraulics and Automation Department, Tampere University of Technology, 33101, Finland.
Corresponding author's email: oftadeh.reza@ieee.org.

[1] www.iha.tut.fi/puresafe/

the platform except when a wheel is singular and should be treated separately [15]. Moreover, when traversing along a straight line, ICR remains at infinity which can be regarded as a presentation singularity for that state space. Authors in [16] proposed an alternative ICR representation to avoid such singularities. Results in [16] are later extended in [17], where a singular-free switching state space has been proposed that addresses both inherent and presentation singularities. In another approach by [18], to avoid singularity, each wheel has been given an extra degree of freedom which makes the wheels footprint variable. However, the problem of operating in the close neighborhood of the singular configurations still remains unsolved.

Most of the solutions proposed so far try to plan ICR trajectories in singular free regions of platform velocity space [15]. Other solutions [8], [19] treat the singular configurations and their neighborhood as obstacles and solve a navigation problem based on potential field and/or model predictive control methods. However, in all of those methods considerable portions of the configuration space are avoided, thus reducing maneuverability of the platform. Even when realizing some simple maneuvers, at some points of the operation ICR will necessarily get relatively close to at least one wheel. One of the simple cases is when the platform is moving on a straight line while changing its heading by 180 degrees.

Furthermore, when the robot is required to follow a desired path and heading profile, ICR position has already been determined. Hence none of those approaches are suitable for path following problems. To clarify this matter, lets call the footprint of a wheel steering axis, the wheel path. Consider the platform body frame is moving on a regular curve called the body path with its natural parameter denoted $'s'$. Moreover, its heading, changes as a smooth function of s namely the heading function. Regardless of the velocities, the wheel path can be determined as a function of the body path and the heading function. Clearly, relative distance between the platform ICR and a wheel axis is the reciprocal of the wheel path curvature. Hence, a wheel is singular when the ICR coincides with its steering axis and equivalently when wheel path curvature is infinity.

In this paper, following ideas from classical path followers [20], we extend our earlier work[21] and develop a novel path following controller that guides the platform on a desired path and correct the heading accordingly. Consider the magnitude of the platform velocity is v, we show that stability of the origin of the error space is achieved regardless of v. Moreover, the control signals simplify the non-holonomic constraints and makes the curvature of the wheels paths independent of v and only as functions of desired body path, heading function and error signals. Hence, the steering and driving velocities of the wheels become linear proportions of v. Having v as an independent variable, this solution allows us to analytically find a maximum v at each sample time that keeps the actuators equal or less than their pre-specified bounds. The solution for a given path and heading function is time optimal since at each time step at least one actuator operates at its maximum velocity.

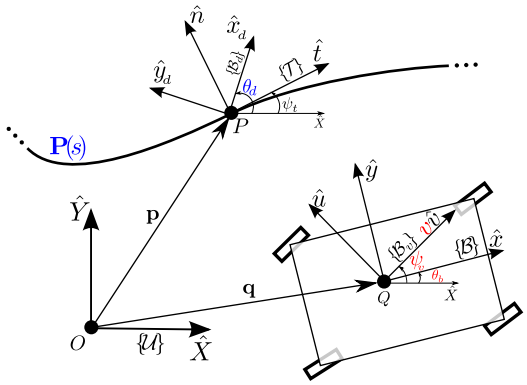


Fig. 2. Desired path and the required coordinate frames

The paper is organized as follows. In Section II, we describe the general architecture of the robot. Next, we define the problem at hand and the error space. In section IV, we describe the controller. Next, in Section V, by means of the proposed control signals we derive the analytical velocity constraints and based on that the optimal velocity of the platform is derived in VI. In the last section, we show the efficacy of the proposed solution through experiments done with the *iMoro* mobile platform.

II. General Descriptions

A. Robot's Architecture

iMoro, shown in Fig.1, consists of a rigid base and four wheels, each of which has two DOF. In fact, each wheel is equipped with two independent servo drives for steering and driving. The steering actuator rotates the whole wheel along its vertical axis, hence determining the heading of that wheel. The driving actuator drives the wheel and its contact point with the ground coincides with the wheel's vertical axis. The robot is equipped with off-the-shelf control drivers that provide velocity and position control options. Because of control strategy adopted in this paper, it is desirable to control the position of the steering and velocity of driving actuators. Hereafter, we will assume that control signals are four steering angles and four wheel speeds.

B. Notations and Symbols

We denote the vectors in bold with hat-sign($\hat{}$) for unit vectors. A left superscript to a vector denotes the frame in which it is expressed, except for the inertial frame, which is omitted for the sake of brevity. Moreover, for two arbitrary vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a}.\boldsymbol{b}$ and $\boldsymbol{a} \times \boldsymbol{b}$ are the inner and cross products of the two vectors, respectively.

Fig. 2 depicts a schematic view of the platform together with a desired path. The coordinate frame $\boldsymbol{\mathcal{U}}\{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\}$ is the inertial frame with unit vectors $\hat{\boldsymbol{X}}$, and $\hat{\boldsymbol{Y}}$. Frame $\boldsymbol{\mathcal{B}}\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ is a fixed-body frame defining the heading of the robot, and $\boldsymbol{\mathcal{B}}_v\{\hat{\boldsymbol{v}}, \hat{\boldsymbol{u}}\}$ is the velocity frame, that is, unit vector $\hat{\boldsymbol{v}}$

determines the direction of the robot's base linear velocity vector and scalar v its magnitude. Both $\boldsymbol{\mathcal{B}}$ and $\boldsymbol{\mathcal{B}}_v$ are attached to the robot's base at point $\boldsymbol{Q}$ which can be chosen at will. Angles ψ_v and θ_b are the angle of $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{x}}$, respectively, in $\boldsymbol{\mathcal{U}}$. The tangent frame $\boldsymbol{\mathcal{T}}\{\hat{\boldsymbol{t}}, \hat{\boldsymbol{n}}\}$ is the Serret-Frenet frame at point $\boldsymbol{P}$ attached to the desired path $\boldsymbol{P}_d(s)$ and the angle between $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{X}}$ is denoted as ψ_t. To determine the desired heading, we define frame $\boldsymbol{\mathcal{B}}_d\{\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{y}}_d\}$ attached to $\boldsymbol{P}$ such that the angle of $\hat{\boldsymbol{x}}_d$ in $\boldsymbol{\mathcal{U}}$ which is θ_d, equals the desired heading function $\theta_d(s)$. Note that both desired path $\boldsymbol{P}_d(s)$ and desired heading $\theta_d(s)$ are parametrized with the same parameter s. Moreover, vectors $\boldsymbol{p}$ and $\boldsymbol{q}$ define points $\boldsymbol{P}$ and $\boldsymbol{Q}$, respectively, in $\boldsymbol{\mathcal{U}}$.

III. Problem Definition

The desired path $\boldsymbol{P}_d(s)$ is assumed to be a regular curve with bounded curvature on a horizontal plane. It is defined by a vector-valued function $\boldsymbol{P}_d : [0, L_P] \rightarrow \boldsymbol{R}^2$, where s and L_P are natural parametrization and length of the path, respectively. The curvature of $\boldsymbol{P}_d(s)$ which is a function of s is denoted as $C_c(s)$. Moreover, we assume that the desired heading function $\theta_d(s) : [0, L_P] \rightarrow \boldsymbol{R}$ is two times differentiable function of s.

As long as $\boldsymbol{P}_d(s)$ and $\theta_d(s)$ with mentioned conditions are defined, the scalar s determines the pos of the platform base. Moreover, the wheels configurations are uniquely determined since the wheels steering are tangent to the wheels paths. Hence, as long as the curvature of a wheel path is less than infinity the configuration of the whole platform is determinable by s. In case of errors, three values for errors in x, y directions and one for the heading in addition to s are needed to determine the platform configuration. Hence, along s we derive the following error signals to serve as state variables,

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = {}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}^{-1}(\boldsymbol{q} - \boldsymbol{p}) \tag{1a}$$

$$\theta_e = \theta_d - \theta_b \tag{1b}$$

$$\psi_e = \psi_t - \psi_v, \tag{1c}$$

where ${}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}$ is the rotation matrix $\boldsymbol{R}(\psi_t)$ that defines the rotation from frame $\boldsymbol{\mathcal{T}}$ to frame $\boldsymbol{\mathcal{U}}$. In this case the error signals x_e and y_e are position errors measured along $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{n}}$, respectively.

Time derivative of eqs. (1a) and (1b) results in,

$$\dot{x}_e = \dot{s}(C_c(s)y_e - 1) + v\cos(\psi_e) \tag{2a}$$

$$\dot{y}_e = -\dot{s}C_c(s)x_e - v\sin(\psi_e) \tag{2b}$$

$$\dot{\theta}_e = \frac{\partial\theta_d}{\partial s}\dot{s} - \omega_b \tag{2c}$$

in which, $\omega_b = \dot{\theta}_b$ is the angular velocity of $\boldsymbol{\mathcal{B}}$.

Problem 1: Given desired path $\boldsymbol{P}_d(s)$ and heading profile $\theta_d(s)$, derive *feedback control laws* for the speed and the steering angle of each wheel such that,

1) Path following: frame $\boldsymbol{\mathcal{B}}_v$ converges and follows frame $\boldsymbol{\mathcal{T}}$, that is, error signals x_e, y_e and ψ_e remain bounded and converge to zero. See eqs. (1a) and (1c).

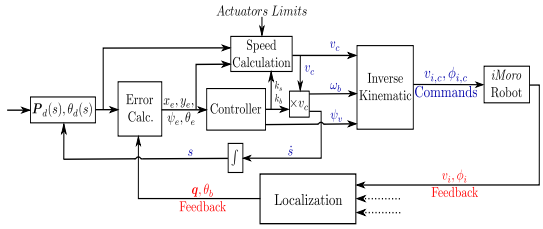


Fig. 3. Schematic block diagram of the whole system

2) Heading control: frame $\boldsymbol{\mathcal{B}}$ converges and follows frame $\boldsymbol{\mathcal{B}}_d$, that is, error signal θ_e remain bounded and converge to zero. See (1b).
3) Bounded control signal: rate of steering and speed of the wheels do not exceed predefined actuator bounds.

We consider the following assumptions to derive our solution. There is no mechanical constraint for wheels steering. The wheels are free-turn. The robot moves on a flat and horizontal plane. The base and the wheels are rigid and the wheels are non deformable. Furthermore, the condition of pure rolling without any side slippage is assumed for the wheels.

We will solve Problem 1 in two stages. First, assuming speed v a free variable, we use angular velocity ω_b and base linear velocity direction $\hat{\boldsymbol{v}}$ as control inputs and propose control laws to address subproblems 1 and 2 of Problem 1. This is presented next in Section IV. These control laws are then mapped to actuator signals (wheels steering and speed commands) using inverse kinematic relations. In the second stage, we use velocity magnitude v to address actuator bounds. This is addressed in Section VI. Fig. 3 schematically depicts the block diagram of control architecture. In this paper we do not discuss the localization block. We have described Inverse and Forward kinematic blocks in [22]. Many standard localization algorithms for mobile robots along with more specific approaches such as the one given in [23] are applicable. However, one of the key requirements of our approach is that the error signals should be continuous and differentiable which imposes the localization block to provide smooth signals for the platform pose.

IV. Derivation of Control Laws

Next we derive the control laws for ω_b and ψ_v (direction of $\hat{\boldsymbol{v}}$). Consider the error states eqs. (1a) to (1c). The control objective is to derive feedback control laws for ψ_v, ω_b, and $\dot{s}$ such that $x_e, y_e, \theta_e, \psi_e$ asymptotically converge to zero. Notice that $\dot{s}$ becomes an auxiliary control signal. First, define functions σ as,

$$\sigma(y_e) = -\operatorname{sgn}(v)\sin^{-1}\frac{k_2 y_e}{|y_e| + \epsilon} \tag{3}$$

where, $0 < k_2 \leq 1$ and $\epsilon > 0$. $\sigma(y_e)$ is a function that generates an appropriate approach angle from the robot to $\boldsymbol{P}_d(s)$. We assume that $v \geq 0$, so $\operatorname{sgn} v = 1$ and $\sigma(y_e)$

becomes independent of v. Since steering angle can take any value from $[-\pi, \pi]$, choice $v \geq 0$ puts no constraint on the configuration space.

Proposition 1: The feedback control laws given by,

$$\dot{s} = (k_1 x_e + \cos\sigma(y_e))v = k_s v \tag{4a}$$

$$\omega_b = (k_3\theta_e + \frac{\partial\theta_d}{\partial s}k_s)v = k_b v \tag{4b}$$

$$\psi_v = \psi_t - \sigma(y_e) \tag{4c}$$

where $k_1, k_3 > 0$ solves sub-problems 1 and 2 of Problem 1. In particular, the origin of the error space is stable and is semi-globally exponentially stable if $v(t) \geq v_m > 0$.

Proof: Here we use similar Lyapunov functions as in [24] and [25], while new control laws are chosen to make curvatures independent of speed v, importance of which will be clear later. Consider the following Lyapunov function,

$$V_P = \frac{1}{2}x_e^2 + \frac{1}{2}y_e^2 + \frac{1}{2}\theta_e^2 \tag{5}$$

which is positive definite and radially unbounded. The derivation of V_P along solution of eqs. (2a) to (2c) result in,

$$\dot{V}_P = -(k_1 x_e^2 + k_2 \frac{y_e^2}{|y_e| + \epsilon} + k_3\theta_e^2)\, v(t) \tag{6}$$

which is negative, thus the origin is stable. For a given $d_1 > 0$, if $v(t) \geq v_m > 0$ and initially $|y_e(t_0)| < d_1$, it is easy to show that $\dot{V}_P < -\lambda V_P$. Thus, the origin is semi-globally exponentially stable [26]. Detail of the proof follows similar steps as in [20]. Now, note that substituting ψ_v from (4c) into (1c) results in $\psi_e = \sigma(y_e)$. Hence, ψ_e is bounded and when y_e converges to zero, error ψ_e will converge to zero. ■

It is worth noticing that gains k_s and k_b defined in (4a) and (4b), respectively, are independent of v. Thus rate of progress $\dot{x}_e$, $\dot{y}_e$, and $\dot{\theta}_e$ become proportional to scaler v. This can be easily shown by investigating $\dot{V}_P$ or substituting the control laws in (2a), (2b), and (2c) and they are given in Appendix. This means that as the platform goes faster it compensates the errors faster. As we show in the next section this choice simplifies the non-holonomic constraints to great extents. One way to derive k_1 and k_3 is to assume a maximum for v namely v_{max} and design constant error gains $k_{1,max}$ and $k_{3,max}$ based on v_{max} and then select k_1 and k_3 such as,

$$k_1 = \frac{k_{1,max}}{v_{max}}\ ,\ k_3 = \frac{k_{3,max}}{v_{max}} \tag{7}$$

Last but not least, while the controller require the $\boldsymbol{P}_d(s)$ to be a regular curve, the actual desired path may consist of multiple regular curves that are non-smoothly connected together and a higher level state-machine can be designed to feed the regular segments individually to the controller. When the platform reaches the end of a segment the state machine stops the platform and rotates the steering wheels to comply with the start of the next segment and then feeds the new segment to the controller. Moreover, while a segment could be infinitesimally small in order to emulate a spot turn for the platform, that higher level state machine can also be incorporated to provide the platform with exact spot turn at the end of any desired segment.

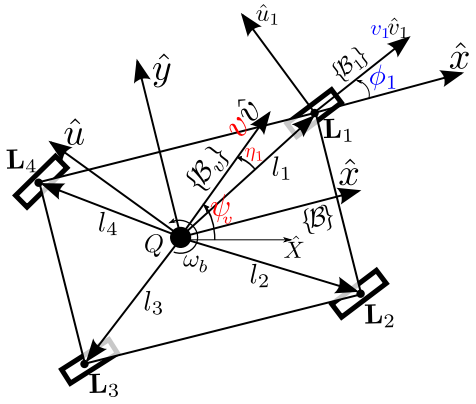


Fig. 4. Denoted kinematic parameters

V. Derivation of Kinematics Constrains

In Fig. 4 the necessary variables are defined. Based on the figure, ${}^{\mathcal{B}}\boldsymbol{\ell}_i$, $i \in \{1, 2, 3, 4\}$ are constant vectors related to the geometry of the robot with their magnitude being l_i. ϕ_i is the steering angle of the i^{th} wheel and the vector ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ can be written in the form of $[\cos(\phi_i)\ \ \sin(\phi_i)\ \ 0]^T$. $v_i\hat{\boldsymbol{v}}_i$ is the velocity vector of the attachment point $\boldsymbol{L}_i$ which also coincides with the wheel steering axis. η_i is the angle between $\hat{\boldsymbol{v}}$ and $\boldsymbol{\ell}_i$.

The following kinematic constraint maps the base velocity space to the wheels velocities.

$${}^{\mathcal{B}}\hat{\boldsymbol{v}} = \boldsymbol{R}(\psi_v - \theta_b)[1\ 0\ 0]^T \tag{8a}$$

$$v_i\ {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = v\ {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) \tag{8b}$$

in which, $\hat{\boldsymbol{z}} = [0\ 0\ 1]^T$ and $\boldsymbol{R}(\psi_v - \theta_b)$ is the rotation matrix with angle $\psi_v - \theta_b$ around z-axis, that is, frame $\boldsymbol{\mathcal{B}}_v$ in $\boldsymbol{\mathcal{B}}$. The norm of (8b) and also its time differentiation can be simplified to,

$$v_i = \sqrt{v^2 + \omega_b^2 l_i^2 + 2v\omega_b l_i \sin\eta_i} \tag{9a}$$

$$v_i^2\ \dot{\phi}_i = l_i\,(\dot{\omega}_b v - \dot{v}\omega_b)\cos\eta_i + v\,(\omega_v - \omega_b)\,(v + l_i\omega_b \sin\eta_i) \tag{9b}$$

in which, $\omega_v = \dot{\psi}_v$. Note that deriving (9b) requires some tedious but elementary algebraic manipulation. Notice the presence of acceleration terms $\dot{v}$ and $\dot{\omega}_b$ in the above equation and also its non-linearity with respect to v. In what follows we show that the choice of control signals in previous section cancel out those accelerations. It is virtually impossible to derive the velocity v based on the other variables. Moreover, the singular configuration is when $|\omega_b|$ becomes v/l_i, $\sin\eta_i$ becomes $-\operatorname{sgn}(\omega_b)$ and consequently $\cos\eta_i$ becomes zero. In this case v_i becomes zero and $\dot{\phi}_i$ is undefined. Notice

that as the robot gets close to its singular configuration $\dot{\phi}_i$ becomes unboundedly large.

Here, we simplify above equations using the control signals presented in previous section. Based on (4b) and (4c), $\dot{\omega}_b$ and ω_v can be written as,

$$\dot{\omega}_b = k'_b \; v^2 + k_b \; \dot{v} \tag{10a}$$

$$\omega_v = k_v \; v \tag{10b}$$

in which, k'_b and k_v are independent of the platform speed v and are given in Appendix. Substitute $\dot{\omega}_b$, ω_b and ω_v from eqs. (4b), (10a) and (10b) into eqs. (8b), (9a) and (9b),

$$^{\mathcal{B}}\hat{\boldsymbol{v}}_i = \frac{^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b(\hat{z} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)}{\sqrt{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i}} \tag{11}$$

$$v = \frac{v_i}{\sqrt{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i}} \tag{12a}$$

$$v = \dot{\phi}_i \frac{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i}{k'_b l_i \cos\eta_i + (k_v - k_b)\,(1 + l_i k_b \sin\eta_i)} \tag{12b}$$

As described in Section II, the input values for steering and driving actuators of the wheel i are ϕ_i and v_i, respectively. Based on (11), the wheel steering angle is independent of v. Hence, even if the robot is stopped, the wheel path and so the steering angle can be determined. Moreover, k'_b, k_b and k_v are functions of errors signals x_e, y_e, θ_e and desired variables $\boldsymbol{P}_d(s)$, $\theta_d(s)$ and their partial differentiations. Hence, for a given v_i, (12a) gives the velocity v that realizes that velocity. Correspondingly, given a $\dot{\phi}_i$, (12b) gives the velocity v that realizes that steering velocity. The curvature of the wheel path namely κ_i is $\dot{\phi}_i / v_i$ which is independent of v. Hence, as the platform moves toward its singular position and κ_i goes to infinity, (12b) reduces v with an appropriate rate to retain the given steering velocity.

Notice that right at the singular configuration, (12b) gives zero for v and the platform stops. Hence, if the robot were to be entrapped exactly in a singular configuration it would stop indefinitely. However, we state without proof that such configuration is an unstable equilibrium point of the system and practically it cannot become deadlock for the platform.

VI. Bounded Velocity Solution

In this section, we present our strategy to find the upper-bounds for v_c; the command signal for v, to limit the driving and steering velocities of wheels to prespecified bounds.

For a robot that has n independently steered wheels there are $2n$ velocity constraints to fulfill which are n deriving and n steering constraints. Consider the maximum driving velocity of a wheel is $V_{d,max}$ and the maximum steering velocity is $\dot{\phi}_{max}$. At each step of time, k'_b, k_b and k_v are calculated so as the control signals ω_b and ψ_v. Next, substituting v_i by $V_{d,max}$ in (12a) and evaluate it for n wheels result in n candidates for the platform velocity. Equivalently, substituting $\dot{\phi}$ by $\dot{\phi}_{max}$ in (12b) and again evaluate it for all the wheels result in another set of n velocity candidates. Hence, there are $2n$ candidates for v namely $v_{c,j}$, we select v as,

$$v_c = \min(|v_{c,j}|), j \in \{1, 2, .., 2n\} \tag{13}$$

Note that functions (12a) and (12b) are strictly monotonic with respect to v_i and $\dot{\phi}_i$ respectively and so is their inverse with respect to v. Hence, the minimum of those eight candidates result in driving and steering velocities less than or equal to the given bounds. Moreover, based on this method at each point in time, at least one actuator is performing at its limit. When the wheels are far from the body ICR, the limit is determined by driving bond. When a wheel gets close to the body ICR, the limit is set by the steering bond. Hence, at each step the given velocity is the maximum feasible velocity to satisfy the actuators constraints and the solution is time optimal. In real robot, command speed v_c differs from v. It can be shown that the system is input-to-state stable with deviation $v - v_c$ as input. See [26] for more details.

VII. Experimental Results

In this section, experimental results of the proposed control system are presented. Experiments are done on *iMoro*, Fig. 1, with parameters shown in Table I. The limit for steering velocity mentioned in the table is considerably lower than feasible value to show the efficacy of the algorithm. Controller software is implemented in an embedded PC with real-time Linux [27].

Fig. 5 shows the actual path taken by the robot in the experiment based on wheel odometry. The desired path is a Bezier curve and the heading function is a simple polynomial of the path variable that changes from zero to 2π and hence require the robot to turn around itself while it follows the path. There is a relatively large intentional initial error between the start of the path and the initial position of the platform to test the performance of the controller. The figure shows that the robot properly follows the desired path and the heading profile.

Notice that even with some fluctuations due to the robot's unmodeled dynamics, the velocity boundaries are kept in comply with the velocity requirements. Fig.6 shows that the steering velocities remain below the bound, while Fig.7 shows that the driving velocities are saturated to the bound. Fig.8 show the maximum allowable speed command in the experiment. The figures show how the propose algorithm reduces the command speed (particularly sharply at critical areas) and thus retain the steering velocities below the prespecified limit.

TABLE I

Specifications of *i*Moro robot for the experimental studies

Description	Quantity
Main Body Length (x direction)	655 mm
Main Body Width (y direction)	335 mm
Maximum Velocity of Steering Servo Motors	0.4 $^{rad}/_{sec}$
Maximum Velocity of Driving Servo Motors	50 $^{mm}/_{sec}$
Wheel Diameter	210 mm
Approximate Overall Mass	120 kg

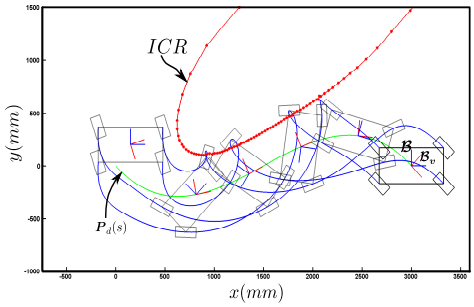


Fig. 5. Robot footprint in the experiment

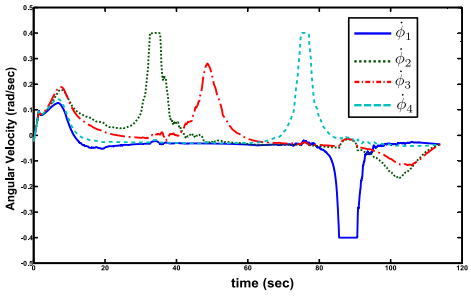


Fig. 6. Steering velocities $\dot{\phi}_i$

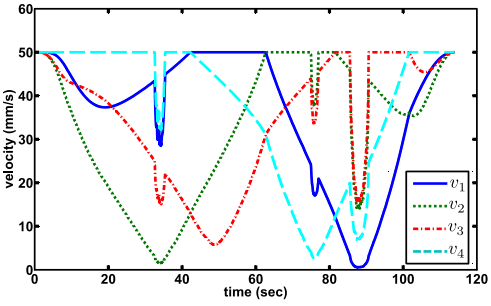


Fig. 7. Driving velocities v_i

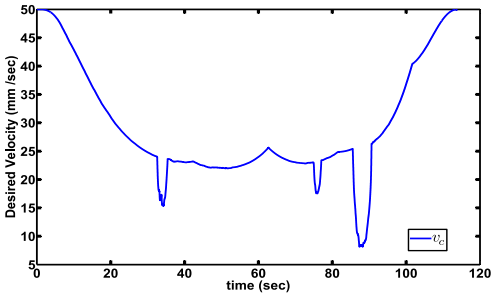


Fig. 8. Command signal for base speed v_c

VIII. Conclusion

This paper proposes a new solution to the path following problem of independently steered mobile robots. We show that utilizing this controller, the robot smoothly follow the desired path while it changes its heading based on a given desired function. Moreover, this design leaves the speed of the robot base as an arbitrary variable. We show that the proposed control signals significantly simplifies the non-holonomic kinematic constraints of the robot. Hence, the speed of the base can be determined analytically to keep the steering and driving velocities of the wheels under pre-specified values. Therefore, unlike many previous methods, our approach allows the robot to get close to its singular configurations and hence exploit its inherent maneuverability. The experiments show that even without considering any model-based approach or dynamic analysis, the designed path follower and related kinematics formulations are capable of bounding both steering and driving velocities in practice with some tolerances. Since this approach gives an analytical solution; it is capable of real-time implementation with low process costs. Our future work will target performance improvement of the controller for mobile manipulation purposes and its fault tolerance.

Appendix

Substituting the control signals (4b) and (4c) in the error states (2a), (2b), and (2c), they can be written as,

$$\dot{x}_e = \left(k_s \left(C_c y_e - 1\right) + \cos\left(\psi_e\right)\right) v = k_x v \tag{14a}$$

$$\dot{y}_e = -\left(k_s C_c x_e + \sin\left(\psi_e\right)\right) v = k_y v \tag{14b}$$

$$\dot{\theta}_e = \left(\frac{\partial \theta_d}{\partial s} k_s - k_b\right) v = k_\theta v \tag{14c}$$

Differentiating from (4c) and (4b) respectively yields to,

$$\omega_v = \left(\left(1 + \frac{\partial \sigma\left(y_e\right)}{\partial y_e} x_e\right) C_c k_s + \frac{\partial \sigma\left(y_e\right)}{\partial y_e} \sin(\psi_e)\right) v = k_v v \tag{15a}$$

$$\dot{\omega}_b = k_b' \, v^2 + k_b \dot{v} \tag{15b}$$

in which,

$$k_b' = k_2 k_\theta + \frac{\partial^2 \theta_d}{\partial s^2} k_s^2 + \frac{\partial \theta_d}{\partial s} k_1 k_x - \frac{\partial \theta_d}{\partial s} \frac{\partial \sigma\left(y_e\right)}{\partial y_e} k_y \sin \sigma(y_e) \tag{16}$$

Acknowledgments

This work, supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network, was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA grant agreement number 264336.

REFERENCES

[1] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 2, pp. 20–33, 2012.

[2] B. Graf, U. Reiser, M. Hagele, K. Mauz, and P. Klein, "Robotic home assistant care-o-bot® 3-product vision and innovation platform," in *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*. IEEE, 2009, pp. 139–144.

[3] C. Cariou, R. Lenain, B. Thuilot, and M. Berducat, "Automatic guidance of a four-wheel-steering mobile robot for accurate field operations," *Journal of Field Robotics*, vol. 26, no. 6-7, pp. 504–518, 2009.

[4] U. Schwesinger, C. Pradalier, and R. Siegwart, "A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5360–5366.

[5] G. Campion, G. Bastin, and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 1, pp. 47–62, 1996.

[6] P. F. Muir and C. P. Neuman, "Kinematic modeling of wheeled mobile robots," *Journal of robotic systems*, vol. 4, no. 2, pp. 281–340, 1987.

[7] J.-B. Song and K.-S. Byun, "Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels," *Journal of mechanical science and technology*, vol. 23, no. 10, pp. 2747–2756, 2009.

[8] C. Connette, C. Parlitz, M. Hägele, and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots," in *IEEE International Conference on Robotics and Automation, (ICRA)*. IEEE, 2009, pp. 4124–4130.

[9] F. G. Pin and S. M. Killough, "A new family of omnidirectional and holonomic wheeled platforms for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 10, no. 4, pp. 480–489, 1994.

[10] D. B. Reister and M. Unseren, "Position and constraint force control of a vehicle with two or more steerable drive wheels," *Robotics and Automation, IEEE Transactions on*, vol. 9, no. 6, pp. 723–731, 1993.

[11] J. Borenstein, "Control and kinematic design of multi-degree-of freedom mobile robots with compliant linkage," *Robotics and Automation, IEEE Transactions on*, vol. 11, no. 1, pp. 21–35, 1995.

[12] Y. Mori, E. Nakano, and T. Takahashi, "Mechanism, control and design methodology of the nonholonomic quasi-omnidirectional vehicle odv9," *The International Journal of Robotics Research*, vol. 21, no. 5-6, pp. 511–525, 2002.

[13] M. Selekwa and J. Nistler, "Path tracking control of four wheel independently steered ground robotic vehicles," in *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*. IEEE, 2011, pp. 6355–6360.

[14] A. K. Singh, D. Ghose, and K. M. Krishna, "Optimum steering input determination and path-tracking of all-wheel steer vehicles on uneven terrains based on constrained optimization," in *American Control Conference (ACC), 2012*. IEEE, 2012, pp. 3611–3616.

[15] B. Thuilot, B. d'Aandrea Novel, and A. Micaelli, "Modeling and feedback control of mobile robots equipped with several steering wheels," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 3, pp. 375–390, 1996.

[16] C. P. Connette, A. Pott, M. Hagele, and A. Verl, "Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an icm representation in spherical coordinates," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 4976–4983.

[17] C. Connette, M. Hagele, and A. Verl, "Singularity-free state-space representation for non-holonomic, omnidirectional undercarriages by means of coordinate switching," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4959–4965.

[18] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011, pp. 6136–6142.

[19] C. Connette, A. Pott, M. Hägele, and A. Verl, "Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 4775–4781.

[20] R. Ghabcheloo, "Coordinated Path Following," Ph.D. dissertation, Instituto Superior Técnico, Technical University of Lisbon, Lisbon, Portugal, 2007.

[21] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Bounded-velocity motion control of four wheel steered mobile robots," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Wollongong, Australia*, July 2013.

[22] ——, "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback," in *The 6th IFAC Symposium on Mechatronic Systems*, Hangzhou, China, 10-12 April 2013.

[23] L. Clavien, M. Lauria, and F. Michaud, "Instantaneous centre of rotation estimation of an omnidirectional mobile robot," in *IEEE International Conference on Robotics and Automation (ICRA), Anchorage, Alaska, USA*. IEEE, 2010, pp. 5435–5440.

[24] D. Soetanto, L. Lapierre, and A. Pascoal, "Adaptive, non-singular path-following control of dynamic wheeled robots," in *Proceedings of 42nd IEEE Conference on Decision and Control (CDC)*, vol. 2. IEEE, 2003, pp. 1765–1770.

[25] I. Kaminer, A. Pascoal, and O. Yakimenko, "Nonlinear path following control of fully actuated marine vehicles with parameter uncertainty," in *16th IFAC World Congress*, Prague, Czech Republic, 2005.

[26] H. K. Khalil and J. Grizzle, *Nonlinear systems*. Prentice hall New Jersey, 1996, vol. 3.

[27] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM),Kaohsiung, Taiwan*, July 2012, pp. 274 –279.

Publication III

Oftadeh, Reza, Reza Ghabcheloo, and Jouni Mattila. "Time optimal path following with bounded velocities and accelerations for mobile robots with independently steerable wheels." In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2925-2931. IEEE.

Time Optimal Path Following with Bounded Velocities and Accelerations for Mobile Robots with Independently Steerable Wheels

Reza Oftadeh, Reza Ghabcheloo, Jouni Mattila

***Abstract*— Mobile robots with independently steerable wheels provide better robustness and efficiency compared to the other types of omnidirectional mobile robots. However, the non-holonomic constraints and singular configurations give rise to several challenging issues in exploiting the high maneuverability features of the robot. Many proposed motion controllers for such robots force the robot to stay outside of bulky regions around its singular points, which in turn limits the robot's dexterity. In this paper, which extends our previous works, we present an online trajectory generation along with a globally stable path following controller that enables the robot to follow any given smooth path and heading function. We show that the control signals extensively simplify the kinematic constraints and are utilized to develop an efficient online "Phase Plane" switching algorithm that bounds the velocities and accelerations of the actuators. Moreover, we show that the algorithm efficiently regulates the velocity of the robot around the singular configurations which allows the robot to realize wide ranges of complex maneuvers. The proposed control algorithm has been tested on iMoro(our four-wheeled independently steerable mobile manipulator), and the presented results show the efficacy of our method.**

I. Introduction

Interest in mobile robots with active steering wheels has been increasing over time. This is due to their high maneuverability and flexibility while being able to operate on rough terrains and carry high payloads with more efficiency. Such platforms are now being used in developing advanced mobile robots for many practical tasks such as service [1], [2], agricultural [3] and space applications [4]. We as a part of the PURESAFE[1] project that aims to prevent human intervention in radioactive environments have designed and developed a four-wheeled independently steerable mobile manipulator (iMoro), which is shown in Fig. 1. Our goal is to take advantage of such platforms' maneuverability to perform manipulation and remote inspection in the confined spaces of the CERN LHC tunnel.

The mobile platform's high maneuverability comes from its omnidirectional nature. The mobile platform is able to realize any arbitrary, independent set of linear and angular velocities but only after it has initially reoriented its wheels to a predetermined corresponding configuration. Hence, while such platforms have three Degrees of Freedom (DoF) [5], they are over-actuated [6], with all the actuators abiding by the non-holonomic constraints of the robot. Hence, such robots are sometimes called non-holonomic omnidirectional robots [7] or pseudo-omnidirectional robots [8], [4].

Authors are with the Intelligent Hydraulics and Automation Department, Tampere University of Technology, 33101, Finland.
Corresponding author's email: oftadeh.reza@ieee.org.

[1]www.iha.tut.fi/puresafe/

However, the presence of singularities both inherently [9] and in the presentation of the configuration space [10] makes the design and realization of motion controllers a challenging task. One of the most popular ways to describe the platform configuration space and those singularities is with the notion of Instantaneous Center of Rotation (ICR) [9], [11]. As the ICR gets close to a wheel axis, the driving velocity of that wheel decreases while its steering velocity unboundedly increases. When the ICR coincides with the wheel steering axis, the driving velocity of the wheel becomes zero and its steering angle undefinable.

Most of the solutions proposed so far try to plan ICR trajectories in singular free regions of platform velocity space [9]. Other solutions [8], [12] treat the singular configurations and their neighborhoods as obstacles and solve a navigation problem based on potential field and/or model predictive control methods. However, in all of those methods, considerable portions of the configuration space are avoided, thus reducing maneuverability of the platform. Even when realizing certain simple maneuvers, at some points of the operation ICR will necessarily get relatively close to at least one wheel at some point of its operation. Furthermore, when the robot is required to follow a desired path and heading profile, the ICR position that corresponds with the curvature of the wheels' paths has already been determined, and a wheel is singular when the ICR coincides with its steering axis and equivalently, when the curvature of the wheel path is infinity. Hence, no existing approaches are suitable for path-following problems.

Fig. 1. The iMoro Mobile Manipulator

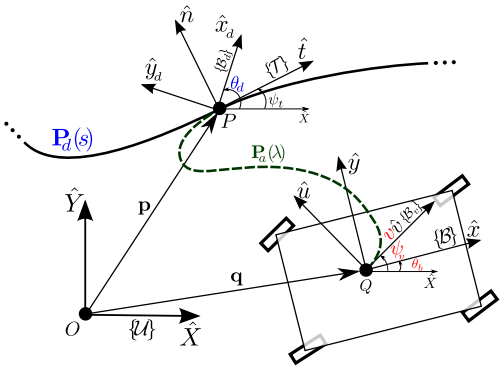


Fig. 2. Desired path and the required coordinate frames

In [13] and [14], we have presented a globally stable nonlinear controller for the path-following problem of the robot that simplifies the kinematic constraints to develop a bounded velocity trajectory. However, the evaluated maximum velocity changes very rapidly, especially in the vicinity of singular configurations, which requires the platform to accelerate or decelerate very rapidly.

In this paper, we give new insights and explanations for our previous controller that allows us to derive acceleration bounds for the robot platform. Furthermore, we show that, at each instant, the robot's actual path, which guides the platform toward the desired path, can be derived analytically and independent of the platform velocity profile. Taking advantage of these results, we incorporate a class of minimum time trajectory generation methods commonly known as the Phase-Plane Approach (PPA), first introduced by [15] and [16]. For serial manipulators, the method is commonly used *offline* to derive time-optimal trajectory for the robot's *desired path* (see [17] for a review on this topic). As the major contribution of this paper, we show that our path-following controller and the resultant differential kinematic constraints can serve as a *dynamic model* for the PPA. Moreover, we present an *online* version of the PPA to generate a minimum time trajectory for the robot's *actual path* that switches between the maximum and minimum allowable accelerations and bounds the velocities of both driving and steering actuators.

II. General Descriptions

A. Robot's Architecture

iMoro, shown in Fig.1, consists of a rigid base and four wheels. Each wheel is equipped with two independent, off-the-shelf servo actuators for steering and driving. The steering actuator rotates the whole wheel along its vertical axis. The driving actuator drives the wheel. Additionally, the wheel's contact point with the ground coincides with its vertical axis. Because of the control strategy adopted in this paper, it is desirable to control the position of the steering and the velocity of the driving actuators. Hereafter, we will assume that the control signals are four steering angles and four wheel speeds.

B. Notations and Symbols

We denote the vectors in bold, using the hat-sign($\hat{}$) for unit vectors. A left superscript on a vector denotes the frame in which it is expressed, except for the inertial frame, which is omitted for the sake of brevity. Moreover, for the two arbitrary vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a}.\boldsymbol{b}$ and $\boldsymbol{a} \times \boldsymbol{b}$ are the inner and cross products of the two vectors, respectively.

Fig. 2 depicts a schematic view of the platform, together with a desired path. The coordinate frame $\boldsymbol{\mathcal{U}}\{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\}$ is the inertial frame. Frame $\boldsymbol{\mathcal{B}}\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ is a fixed-body frame that defines the platform heading: θ_b. $\boldsymbol{\mathcal{B}}_v\{\hat{\boldsymbol{v}}, \hat{\boldsymbol{u}}\}$ is the velocity frame, that is, the unit vector $\hat{\boldsymbol{v}}$, defined by the velocity angle ψ_v, determines the direction of the robot's base linear velocity vector, and the scalar v, represents its magnitude. Both $\boldsymbol{\mathcal{B}}$ and $\boldsymbol{\mathcal{B}}_v$ are attached to the robot's base at point $\boldsymbol{Q}$, defined by the position vector $\boldsymbol{q}$, and can be chosen at will. The position vector $\boldsymbol{p}$ defines the point $\boldsymbol{P}$ on the desired path $\boldsymbol{P}_d(s)$. The tangent frame $\boldsymbol{\mathcal{T}}\{\hat{\boldsymbol{t}}, \hat{\boldsymbol{n}}\}$ is the path's Serret-Frenet frame at point $\boldsymbol{P}$, and the angle between $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{X}}$, denoted as ψ_t, is its tangent angle. To determine the desired heading, we define another frame $\boldsymbol{\mathcal{B}}_d\{\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{y}}_d\}$ attached to $\boldsymbol{P}$ such that the angle of $\hat{\boldsymbol{x}}_d$ in $\boldsymbol{\mathcal{U}}$, which is θ_d, equals the desired heading function $\theta_d(s)$.

III. Problem Definition

The desired path $\boldsymbol{P}_d(s)$ is assumed to be a regular curve with a bounded curvature on a horizontal plane. It is defined by the vector-valued function $\boldsymbol{P}_d : [0, L_d] \rightarrow \boldsymbol{R}^2$, where s and L_d are the natural parametrization and length of the path, respectively. The curvature of $\boldsymbol{P}_d(s)$, which is a function of s, is denoted as $C_c(s)$. We assume that the desired heading function $\theta_d(s) : [0, L_d] \rightarrow \boldsymbol{R}$ is a two-times differentiable function of s. Furthermore, the path $\boldsymbol{P}_a(\lambda)$ is the output of the path-following controller and is defined by $\boldsymbol{P}_a : [0, L_a] \rightarrow \boldsymbol{R}^2$, where λ and L_a are the natural parametrization and length of the path, accordingly. $\boldsymbol{P}_a(\lambda)$ consists of two parts. The first part starts at the robot and ends tangentially to $\boldsymbol{P}_d(s)$, and it guides the robot toward the path. The second part starts at the point P and exactly follows $\boldsymbol{P}_d(s)$ to the end of the path. We will show that the closed-loop error states derived from the designed controller result in a set of differential equations for $\boldsymbol{P}_a(\lambda)$.

Based on Fig. 2, the platform configuration can be determined by s and three values for errors, two in x, y directions, and one for the heading. Those errors are defined as,

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = {}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}^{-1}(\boldsymbol{q} - \boldsymbol{p}) \tag{1a}$$

$$\theta_e = \theta_d - \theta_b \tag{1b}$$

$$\psi_e = \psi_t - \psi_v, \tag{1c}$$

where ${}^{\boldsymbol{\mathcal{U}}}\boldsymbol{R}_{\boldsymbol{\mathcal{T}}}$ is the rotation matrix $\boldsymbol{R}(\psi_t)$ that defines the rotation from frame $\boldsymbol{\mathcal{T}}$ to frame $\boldsymbol{\mathcal{U}}$. In this case, the error signals x_e and y_e are position errors measured along $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{n}}$, respectively.

A time derivative of eqs. (1a) and (1b) results in the open-loop equations of motion in the form of,

$$\dot{x}_e = \dot{s}(C_c(s)y_e - 1) + v\cos(\psi_e) \tag{2a}$$
$$\dot{y}_e = -\dot{s}C_c(s)x_e - v\sin(\psi_e) \tag{2b}$$
$$\dot{\theta}_e = \frac{\partial\theta_d}{\partial s}\dot{s} - \omega_b \tag{2c}$$

in which $\omega_b \triangleq \dot{\theta}_b$ is the angular velocity of the frame $\boldsymbol{\mathcal{B}}$.

Problem 1: Given the desired path $\boldsymbol{P}_d(s)$ and heading profile $\theta_d(s)$, derive *feedback control laws* for the speed and the steering angle of each wheel such that:

1) Path Following: frame $\boldsymbol{\mathcal{B}}_v$ converges and follows frame $\boldsymbol{\mathcal{T}}$; that is, error signals x_e, y_e and ψ_e remain bounded and converge to zero. See eqs. (1a) and (1c).
2) Heading Control: frame $\boldsymbol{\mathcal{B}}$ converges and follows frame $\boldsymbol{\mathcal{B}}_d$; that is, error signal θ_e remains bounded and converges to zero. See (1b).
3) Bounded Velocity: The rate of steering and the speed of the wheels do not exceed predefined actuator bounds.
4) Bounded Acceleration: The acceleration of driving actuators does not exceed predefined actuator bounds.

We consider the following assumptions to derive our solution: There is no mechanical constraint for wheel steering. The wheels are free-turn. The robot moves on a flat and horizontal plane. The base and the wheels are rigid, and the wheels are non-deformable.

We will solve Problem 1 in two stages. First, assuming speed v to be a free variable, we use angular velocity ω_b and base linear velocity direction $\hat{\boldsymbol{v}}$ as control inputs and propose control laws to address subproblems 1 and 2 of Problem 1 and to derive the closed-loop state space. This is presented next in SectionIV. In sectionV, we derive and simplify kinematic constraints and utilize them to evaluate the platform's maximum velocity and acceleration bounds. Finally, in sectionVI, we present an online prediction algorithm based on PPA to solve subproblems 3 and 4.

Fig. 3 schematically depicts the block diagram of the proposed control architecture. We do not discuss the localization block in this paper. However, we have described the inverse and forward kinematic blocks in [18]. Many standard localization algorithms for mobile robots are applicable. However, as with many other controllers, while some level of noise is acceptable the performance of our algorithm is directly related to the smoothness of the localization feedback.

IV. Path-Following Controller

A. Derivation of Control Laws

Next, we derive the control laws for ω_b and ψ_v (direction of $\hat{\boldsymbol{v}}$). Consider the error state eqs. (1a) to (1c). The control objective is to derive feedback control laws for ψ_v, ω_b, and $\dot{s}$ such that $x_e, y_e, \theta_e, \psi_e$ asymptotically converge to zero. Notice that $\dot{s}$ becomes an *auxiliary* control signal. First, we define functions σ as:

$$\sigma(y_e) = -\operatorname{sgn}(v)\sin^{-1}\frac{k_2 y_e}{|y_e| + \epsilon} \tag{3}$$

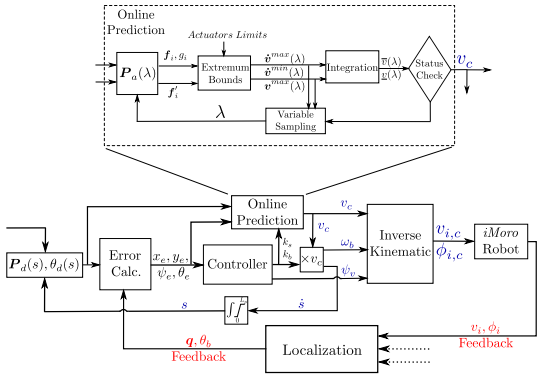


Fig. 3. Schematic block diagram of the whole system

where $0 < k_2 \leq 1$ and $\epsilon > 0$. $\sigma(y_e)$ is a function that generates an appropriate approach angle from the robot to $\boldsymbol{P}_d(s)$. We assume that $v \geq 0$, so $\operatorname{sgn} v = 1$ and $\sigma(y_e)$ becomes independent of v. Since the steering angle can take any value from $[-\pi, \pi]$, choice $v \geq 0$ puts no constraint on the configuration space.

Proposition 1: The feedback control laws given by

$$\dot{s} = (k_1 x_e + \cos\sigma(y_e))v \triangleq k_s v \tag{4a}$$
$$\omega_b = (k_3\theta_e + \frac{\partial\theta_d}{\partial s}k_s)v \triangleq k_b v \tag{4b}$$
$$\psi_v = \psi_t - \sigma(y_e) \tag{4c}$$

where $k_1, k_3 > 0$, solve sub-problems 1 and 2 of Problem 1. In particular, the origin of the error space is stable and is semi-globally exponentially stable if $v(t) \geq v_m > 0$.

Proof: Here, we use similar Lyapunov functions as in [19], [20], [21] but choose new control laws to make curvatures independent of speed v, the importance of which will be clear later. Consider the following Lyapunov function:

$$V_P = \frac{1}{2}x_e^2 + \frac{1}{2}y_e^2 + \frac{1}{2}\theta_e^2 \tag{5}$$

which is positive definite and radially unbounded. The derivation of V_P along the solution of eqs. (2a) to (2c) results in:

$$\dot{V}_P = -(k_1 x_e^2 + k_2\frac{y_e^2}{|y_e| + \epsilon} + k_3\theta_e^2)\, v(t) \tag{6}$$

which is negative, and thus, the origin is stable. For a given $d_1 > 0$, if $v(t) \geq v_m > 0$ and initially $|y_e(t_0)| < d_1$, it is easy to show that $\dot{V}_P < -\lambda V_P$. Thus, the origin is semi-globally exponentially stable. Now, note that substituting ψ_v from (4c) into (1c) results in $\psi_e = \sigma(y_e)$. Hence, ψ_e is bounded, and when y_e converges to zero, the error ψ_e will converge to zero. ∎

B. Closed-Loop Equations of Motion

The robot moves along the path $P_a(\lambda)$. Since λ is the natural parameterization of the robot's actual path, we have $||\dot{\mathbf{q}}|| = v = \dot{\lambda}$. Substituting the control laws(eqs. (4a) to (4c))

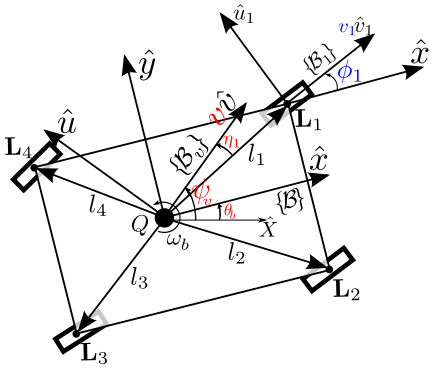


Fig. 4. Denoted kinematic parameters

into the open-loop eqs. (1a) to (1c) results in the closed-loop equations of motion in the form of

$$\dot{s} = k_s v \quad \dot{x}_e = k_x v \quad \dot{y}_e = k_y v \quad \dot{\theta}_e = k_\theta v \tag{7}$$

in which,

$$k_s \triangleq \frac{\partial s}{\partial \lambda} = k_1\ x_e + \cos\sigma\,(y_e) \tag{8a}$$

$$k_x \triangleq \frac{\partial x_e}{\partial \lambda} = k_s\,(C_c y_e - 1) + \cos\sigma\,(y_e) \tag{8b}$$

$$k_y \triangleq \frac{\partial y_e}{\partial \lambda} = -\,(k_s C_c x_e + \sin\sigma\,(y_e)) \tag{8c}$$

$$k_\theta \triangleq \frac{\partial \theta_e}{\partial \lambda} = -k_3\theta_e \tag{8d}$$

From the above equations, it can be concluded that, contrary to many other path-following controllers[19], [20], this design makes the robot progress independently of the future velocity commands. Hence, at each time step, all of the whole kinematic parameters of the platform and the path $P_a(\lambda)$ can be determined by integrating from the above equations.

V. Kinematics Constraints

A. Derivation of Kinematics Constraints

The necessary variables are defined in Fig. 4 . Consider that the robot has $n \geq 2$ independently steered wheels. Based on the figure, ${}^{\mathcal{B}}\boldsymbol{\ell}_i$, $i \in \{1, 2, .., n\}$, are constant vectors related to the geometry of the robot with a magnitude of l_i. ϕ_i is the steering angle of the i^{th} wheel and the vector ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ can be written in the form of $[\cos(\phi_i)\ \ \sin(\phi_i)\ \ 0]^T$. $v_i\hat{\boldsymbol{v}}_i$ is the velocity vector of the attachment point $\boldsymbol{L}_i$, which also coincides with the wheel steering axis. ϕ_i and v_i are the input values for wheel i's steering and driving actuators, respectively. η_i is the angle between $\hat{\boldsymbol{v}}$ and $\boldsymbol{\ell}_i$. The following kinematic constraints map between the velocities and accelerations of the body and the wheels.

$$v_i\ {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = v\ {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) \tag{9a}$$

$$\begin{aligned}\dot{v}_i\,{}^{\mathcal{B}}\hat{\boldsymbol{v}}_i + v_i\dot{\phi}_i\,(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i) =& \dot{v}\ {}^{\mathcal{B}}\hat{\boldsymbol{v}} \\ &+ v\,(\omega_{\boldsymbol{v}} - \omega_b)\,(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\hat{\boldsymbol{v}}) \\ &+ \dot{\omega}_b\,(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)\end{aligned} \tag{9b}$$

in which $\hat{\boldsymbol{z}} = [0\ 0\ 1]^T$ and $\omega_v \triangleq \dot{\psi}_v$. Note that deriving (9b) requires some tedious but elementary algebraic manipulation. Notice the presence of acceleration terms $\dot{v}$ and $\dot{\omega}_b$ in the above equation and also its non-linearity with respect to v. Here, we simplify above equations using the control signals presented in the previous section. Substitute $\dot{\omega}_b$, ω_b, and ω_v from eq. (19) and eqs. (4b) and (18a) from Appendix into the norm of eqs. (9a) and (9b),

$${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = \frac{{}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b(\hat{z} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)}{\sqrt{f_i}} \tag{10a}$$

$$v_i = \sqrt{f_i}\,v \tag{10b}$$

$$\dot{\phi}_i = \frac{g_i}{f_i}v \tag{10c}$$

where,

$$f_i = 1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i \tag{11a}$$

$$g_i = k_b' l_i \cos\eta_i\ + (k_v - k_b)\,(1 + l_i k_b \sin\eta_i\ ) \tag{11b}$$

and $k_b' \triangleq \frac{\partial k_b}{\partial \lambda}$ and $k_v \triangleq \frac{\partial \psi_v}{\partial \lambda}$ are given in Appendix.

Based on (10a), the wheel steering angle is independent of v. Hence, even if the robot is stopped, the wheel path and, thus the steering angle can be determined. Moreover, k_b', k_b and k_v are functions of errors signals x_e, y_e, θ_e and desired variables $\boldsymbol{P}_d(s)$, $\theta_d(s)$ and their partial differentiations. Hence, for a given v_i, (10b) gives the velocity v that realizes that driving velocity. Correspondingly, given a $\dot{\phi}_i$, (10c) gives the velocity v that realizes that steering velocity. The curvature of the wheel path, namely, κ_i, is $\dot{\phi}_i/v_i$, which is independent of v. Hence, as the platform moves toward its singular position and κ_i goes to infinity, (10c) reduces v with an appropriate rate to retain the given steering velocity. We have given a thorough explanation regarding the above equations in [14].

B. Derivation of Velocity and Acceleration Bounds

In order to derive acceleration constraints for driving actuators, time differentiation from (10b) leads to:

$$\dot{v}_i = \sqrt{f_i}\ \dot{v} + \frac{f_i'}{2\sqrt{f_i}}v^2 \tag{12a}$$

$$f_i' \triangleq \frac{\partial f_i}{\partial \lambda} = 2k_b k_b' l_i^2 + 2 l_i k_b' \sin\eta_i\ + 2\,(k_v - k_b)\,l_i k_b \cos\eta_i \tag{12b}$$

Considering that v_d^{max} and $\dot{\phi}^{max}$ are the given maximum velocity bounds for the driving and steering actuators, and $\dot{v}_d^{max}$ is the maximum driving acceleration, we may write,

$$0 < v_i \leq v_d^{max} \quad \left|\dot{\phi}_i\right| \leq \dot{\phi}^{max} \quad |\dot{v}_i| \leq \dot{v}_d^{max} \tag{13}$$

and from (12a), the acceleration bounds for $\dot{v}$ can be written as,

$$\dot{v}^{min}\,(v) \leq \dot{v} \leq \dot{v}^{max}\,(v) \tag{14}$$

where,

$$\dot{v}^{max} = \min_i \alpha_i^{max} \quad \dot{v}^{min} = \max_i \alpha_i^{min} \tag{15a}$$

$$\alpha_i^{max} = \frac{\dot{v}_d^{max} 2\sqrt{f_i} - f_i' v^2}{2 f_i} \quad \alpha_i^{min} = \frac{-\dot{v}_d^{max} 2\sqrt{f_i} - f_i' v^2}{2 f_i} \tag{15b}$$

Now we have to evaluate the maximum allowable velocity for v, namely v^{max}. Substituting v_d^{max} and $\dot{\phi}^{max}$ in equations (10b) and (10c) results in $2n$ limits for v. Another bound comes from the solution of equation $\dot{v}^{min}(v) = \dot{v}^{max}(v)$, which we call it v_{acc}^{max}. If v becomes equal or larger than v_{acc}^{max}, the acceleration bounds collapse and there is no solution for $\dot{v}$. The presentation of v_{acc}^{max} in the phase plane $v - \lambda$ is a curve commonly known in the literature as the velocity limit curve. However, in our case, given the fact that v also needs to satisfy actuator velocity constraints v_d^{max} and $\dot{\phi}^{max}$, we define our velocity limit curve v^{max} as

$$v^{max} = \min_i \left(v_{acc}^{max}, \frac{f_i \dot{\phi}^{max}}{|g_i|}, \frac{v_d^{max}}{\sqrt{f_i}} \right) \tag{16}$$

VI. Online Prediction Algorithm

In this section, we present our online prediction scheme that evaluates v_c, the command signal for the platform velocity v. Applying the evaluated v_c along with the other control signals given in eqs. (4a) to (4c), results in the time optimal path-following of the robot that also respects the given constraints in 13. The solution for v_c is the so called *bang-bang* trajectory, in which at least one of the actuators is always at its maximum velocity or acceleration [22]. Hence, for the given path and the heading profile, the solution for the derived trajectory is time optimal.

At each time step, based on the localization feedback, the path $\boldsymbol{P}_a(\lambda)$ is derived numerically by integrating from closed-loop eqs. (8a) to (8d). Then, the velocity and acceleration constraints are evaluated which are used to find accelerating and decelerating velocity predictions, namely, $\overline{v}(\lambda)$ and $\underline{v}(\lambda)$, respectively. Judging by the behaviour of those velocities, we determine whether v_c should be evaluated based on the maximum acceleration $\dot{v}^{max}$ or whether it *switches* to the deceleration phase and should be derived based on $\dot{v}^{min}$. It should be noted that the robot may not exactly follow $\boldsymbol{P}_a(\lambda)$ and that $\boldsymbol{P}_a$ may therefore change over time. Hence, this algorithm must be executed at each sample time. However, it is assumed that the rate of the change in $\boldsymbol{P}_a(\lambda)$ is relatively slower than the sampling time and that the algorithm is thus able to compensate for the changes in $\boldsymbol{P}_a$. Given that our algorithm and its offline version share some similar parts, which were thoroughly elaborated in [15], [16], [17], here we shall omit further explanations of said parts here.

1) At the start, set $s = 0$ and, considering the initial errors, derive $\dot{v}^{max}$ and set $\dot{v}_0 = \dot{v}^{max}$ and go to 3.
2) At each sample time, consider that the previous v_c is v_{c0}, set $\lambda = 0$ and $\underline{v}(0) = \overline{v}(0) = v_{c0}$. At each sample for λ, consider λ_0 as the previous λ and:
   a) Integrate from eqs. (8a) to (8d) and derive $\dot{v}^{max}(\lambda)$,$\dot{v}^{min}(\lambda)$ and $v^{max}(\lambda)$. Let $\dot{v}_m^{max}(\lambda)$ be the differentiation of $v^{max}(\lambda)$ with respect to λ. If $\overline{v}(\lambda_0) >= v^{max}(\lambda_0)$ then set $\dot{v}^{max}(\lambda_0) = min(\dot{v}^{max}(\lambda_0), \dot{v}_m^{max}(\lambda_0))$. Derive $\underline{v}(\lambda)$ and $\overline{v}(\lambda)$ by substituting $\dot{v}^{min}(\lambda_0)$, $v^{max}(\lambda_0)$ in to the following chain rule relation and integrate from it:

$$\dot{v} = \frac{\partial v(\lambda)}{\partial \lambda} v(\lambda) \tag{17}$$

   b) If $\underline{v}(\lambda) \leq 0$ and $s(\lambda) \geq L_d$ set $\dot{v}_0 = \dot{v}^{min}(0)$ and go to 3.
   c) If $\underline{v}(\lambda) \leq 0$ and $\overline{v}(\lambda) \leq v^{max}(\lambda)$ set $\dot{v}_0 = \dot{v}^{max}(0)$ and go to 3.
   d) If $\underline{v}(\lambda) \geq v^{max}(\lambda)$ or $\overline{v}(\lambda) > v^{max}(\lambda)$ set $\dot{v}_0 = \dot{v}^{min}(0)$ and go to 3.
   e) Go back to 2a.
3) Evaluate v_c by integrating from $\dot{v}_0$ and set v_{c0} as its initial value. The value for v_c is then sent to the main controller to derive other control signals, as shown in the Fig. 3.

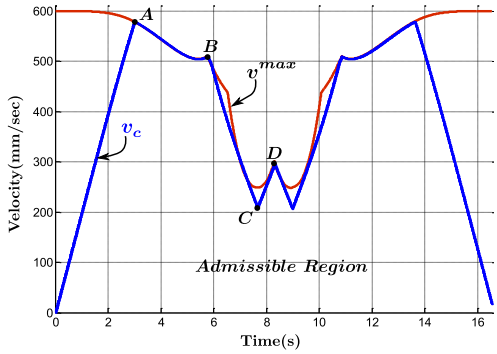


Fig. 5. An example of evaluated platform speed command v_c by the proposed algorithm.

Fig. 5 shows the results of the algorithm when the robot moves along a straight line and changes its heading by 180 degrees. The v^{max} curve in the figure is $v^{max}(\lambda = 0)$ at each instant. As is clear in the figure, the robot starts to accelerate from the zero velocity. At each instant during this time, the prediction is run on λ and sequentially increases it to drive $\underline{v}(\lambda)$ and $\overline{v}(\lambda)$. During this phase, $\underline{v}(\lambda)$ hits the zero velocity axis sooner than $\overline{v}(\lambda)$ hits the velocity limit curve. Hence, during this phase, the condition 2c activates and sets the robot in acceleration phase. This is continuous until the v_c hits the v^{max} curve at the point $\boldsymbol{A}$. Since v^{max} is derived based on the velocity constraints, it is a *critical arc* (see [22]), and applying the $\dot{v}^{max}$ acceleration violates the velocity constraints, while applying the $\dot{v}^{min}$ violates the time optimality. The condition we put in the algorithm- $\dot{v}^{max}(\lambda_0) = min(\dot{v}^{max}(\lambda_0), \dot{v}_m^{max}(\lambda_0))$ when $\overline{v}(\lambda_0) >= v^{max}(\lambda_0)$- makes the robot to follow the v^{max} profile until it reaches the point $\boldsymbol{B}$. At the point $\boldsymbol{B}$, the prediction for $\underline{v}(\lambda)$ hits $v^{max}(\lambda)$, the condition 2d activates and, hence, the platform switches to the deceleration phase. The robot stays in this phase until it reaches to the point $\boldsymbol{C}$ in which $\underline{v}(\lambda)$ hits the zero velocity line sooner than the $\overline{v}(\lambda)$ hits the $v^{max}(\lambda)$. In this case, the robot switches back to the acceleration phase and the process continues as shown in the figure.

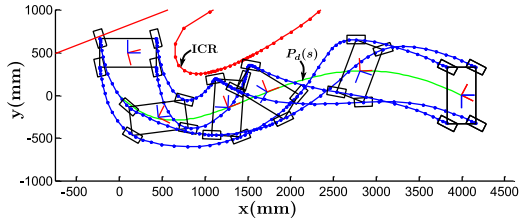


Fig. 6. Robot footprint in the experiment

VII. Experimental Results

This section presents the experimental results of the proposed control system. It should be noted that the figures depict the *command* signals and their differentiations with respect to time. Experiments were done on *iMoro*, Fig. 1, with parameters shown in Table I. Controller software was implemented on an embedded PC with real-time Linux based on the ideas given in [23]. We ran the online prediction algorithm at each 0.005 seconds and incorporated an adaptive variable sampling for λ with "$0.005mm \leq \Delta\lambda \leq 1mm$". Running on a dedicated core of a 2.6Ghz Intel Core™i7 CPU, the implemented algorithm was fast enough to keep the hard real-time deadline.

Fig. 6 shows the actual path taken by the robot in the experiment based on wheel odometry. The desired path is a sine-like Bezier curve, and the heading function is a simple polynomial of the path variable that changes from zero to 1.5π and hence requires the robot to turn around itself while following the path. There is a relatively large intentional initial error between the start of the path and the initial position of the platform to test the performance of the controller. The figure shows that the robot properly follows the desired path and the heading profile. Fig.7 depicts the v_c command evaluated by the online predication algorithm, and the v^{max} curve in the figure is $v^{max}(\lambda = 0)$ at each instant. As shown in the Fig. 6, errors are selected in a way such that the robot ICR changes very rapidly right at the start of the path and hence the curvature of $\boldsymbol{P}_a(\lambda)$ becomes very large. This situation imposes a very low v^{max} at the start of the path, as shown in Fig.7. Fig. 10 shows that the steering velocities. Fig. 8 shows the velocity commands for the wheels' driving actuators and their time differentiation are depicted in Fig. 9. It is clear form these figures that the proposed controller keeps the desired command signals bounded while guiding the robot on the desired path.

TABLE I

Specifications of *i*Moro robot for the experimental studies

Description	Quantity
Main Body Length (x direction)	655 mm
Main Body Width (y direction)	335 mm
Maximum Velocity of the Steering	1 $^{rad}/_{sec}$
Maximum Velocity of the Driving	600 $^{mm}/_{sec}$
Maximum Acceleration of the Driving	200 $^{mm}/_{sec^2}$
Wheel Diameter	210 mm
Approximate Overall Mass	120 kg

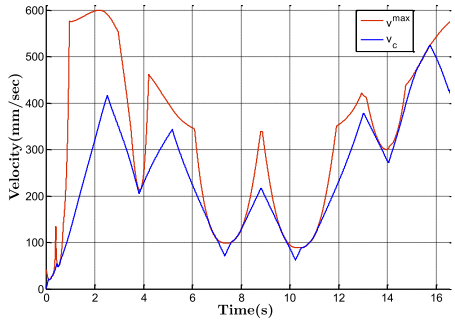


Fig. 7. Command signal v_c and the velocity limit curve over *time*

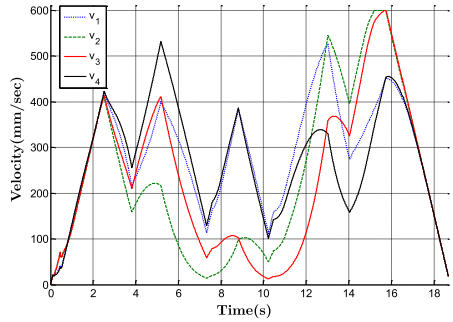


Fig. 8. The velocity command for the driving actuators

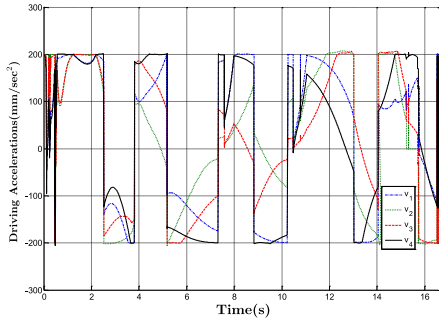


Fig. 9. Time differentiation of driving commands($\dot{v}_i$)

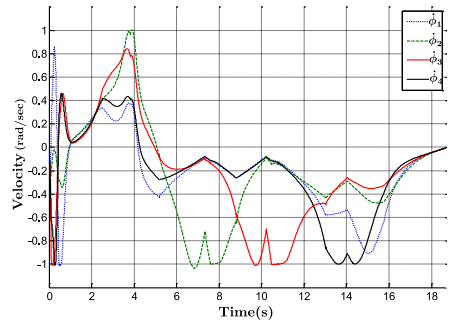


Fig. 10. Time differentiation of steering commands($\dot{\phi}_i$)

VIII. Conclusion

In this paper, which extends our previous works, we present a globally stable path-following controller for independently steerable mobile robots. We show that the control signals lead to closed-loop equations of motion directly propositional to the platform velocity and express a correcting path that smoothly guides the robot toward the reference path and heading function. Based on these results, we simplify the kinematic constraints of the robot in both velocity and acceleration space. This design leaves the speed of the robot base as an arbitrary variable. We exploit this feature to develop an online prediction scheme that keeps the steering velocities and the driving velocities and accelerations of the wheels below a set of given bounds. Moreover, we show that the algorithm efficiently regulates the velocity of the robot around the singular configurations, which allows the robot to realize wide ranges of complex maneuvers. The proposed control algorithm has been tested on iMoro, our four-wheeled, independently steered mobile manipulator, and presented results show the efficacy of our method.

Appendix

Differentiating from (4c) and (4b) respectively yields to

$$\omega_v = \left(\left(1 + \frac{\partial\sigma\left(y_e\right)}{\partial y_e} x_e\right) C_c k_s + \frac{\partial\sigma\left(y_e\right)}{\partial y_e} \sin\psi_e\right) v \triangleq k_v v \tag{18a}$$

$$\dot{\omega}_b = k_b' \, v^2 + k_b \dot{v} \tag{18b}$$

in which,

$$k_b' = -k_3^2\theta_e + \frac{\partial^2\theta_d}{\partial s^2} k_s^2 + \frac{\partial\theta_d}{\partial s} k_1 k_x - \frac{\partial\theta_d}{\partial s} \frac{\partial\sigma\left(y_e\right)}{\partial y_e} k_y \sin\sigma(y_e) \tag{19}$$

Acknowledgments

This work, supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network, was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA grant agreement number 264336.

References

[1] A. Dietrich, T. Wimbock, A. Albu-Schaffer, and G. Hirzinger, "Reactive whole-body control: Dynamic mobile manipulation using a large number of actuated degrees of freedom," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 2, pp. 20–33, 2012.

[2] B. Graf, U. Reiser, M. Hagele, K. Mauz, and P. Klein, "Robotic home assistant care-o-bot® 3-product vision and innovation platform," in *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*. IEEE, 2009, pp. 139–144.

[3] C. Cariou, R. Lenain, B. Thuilot, and M. Berducat, "Automatic guidance of a four-wheel-steering mobile robot for accurate field operations," *Journal of Field Robotics*, vol. 26, no. 6-7, pp. 504–518, 2009.

[4] U. Schwesinger, C. Pradalier, and R. Siegwart, "A novel approach for steering wheel synchronization with velocity/acceleration limits and mechanical constraints," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5360–5366.

[5] G. Campion, G. Bastin, and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 1, pp. 47–62, 1996.

[6] P. F. Muir and C. P. Neuman, "Kinematic modeling of wheeled mobile robots," *Journal of robotic systems*, vol. 4, no. 2, pp. 281–340, 1987.

[7] J.-B. Song and K.-S. Byun, "Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels," *Journal of mechanical science and technology*, vol. 23, no. 10, pp. 2747–2756, 2009.

[8] C. Connette, C. Parlitz, M. Hägele, and A. Verl, "Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots," in *IEEE International Conference on Robotics and Automation, (ICRA)*. IEEE, 2009, pp. 4124–4130.

[9] B. Thuilot, B. d'Aandrea Novel, and A. Micaelli, "Modeling and feedback control of mobile robots equipped with several steering wheels," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 3, pp. 375–390, 1996.

[10] C. P. Connette, A. Pott, M. Hagele, and A. Verl, "Control of an pseudo-omnidirectional, non-holonomic, mobile robot based on an icm representation in spherical coordinates," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*. IEEE, 2008, pp. 4976–4983.

[11] C. Connette, M. Hagele, and A. Verl, "Singularity-free state-space representation for non-holonomic, omnidirectional undercarriages by means of coordinate switching," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 4959–4965.

[12] C. Connette, A. Pott, M. Hägele, and A. Verl, "Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 4775–4781.

[13] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Bounded-velocity motion control of four wheel steered mobile robots," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM), Wollongong, Australia*, July 2013.

[14] R. Oftadeh, R. Ghabcheloo, and J. Mattila, "A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4845–4851.

[15] J. E. Bobrow, S. Dubowsky, and J. Gibson, "Time-optimal control of robotic manipulators along specified paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[16] K. Shin and N. McKay, "Minimum-time control of robotic manipulators with geometric path constraints," *Automatic Control, IEEE Transactions on*, vol. 30, no. 6, pp. 531–541, 1985.

[17] D. Costantinescu and E. Croft, "Smooth and time-optimal trajectory planning for industrial manipulators along specified paths," *Journal of Robotic Systems*, vol. 17, no. 5, pp. 233–249, 2000.

[18] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback," in *The 6th IFAC Symposium on Mechatronic Systems*, Hangzhou, China, 10-12 April 2013.

[19] D. Soetanto, L. Lapierre, and A. Pascoal, "Adaptive, non-singular path-following control of dynamic wheeled robots," in *Proceedings of 42nd IEEE Conference on Decision and Control (CDC)*, vol. 2. IEEE, 2003, pp. 1765–1770.

[20] L. Lapierre, R. Zapata, and P. Lepinay, "Combined path-following and obstacle avoidance control of a wheeled robot," *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 361–375, 2007.

[21] ——, "Simulatneous path following and obstacle avoidance control of a unicycle-type robot," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 2617–2622.

[22] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[23] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM),Kaohsiung, Taiwan*, July 2012, pp. 274 –279.

Publication IV

Oftadeh, Reza, Reza Ghabcheloo, and Jouni Mattila. "A time-optimal bounded velocity path-following controller for generic Wheeled Mobile Robots." In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pp. 676-683. IEEE.

A Time-Optimal Bounded Velocity Path-Following Controller for Generic Wheeled Mobile Robots

Reza Oftadeh, Reza Ghabcheloo, Jouni Mattila

***Abstract*— This paper, as a generalization of our previous works, presents a unified time-optimal path-following controller for Wheeled Mobile Robots (WMRs). Unlike other path-following controllers, we solve the path-following problem for all common categories of WMRs such as car-like, differential, omnidirectional, all wheels steerable and others. We show that the insertion of our path-following controller into the kinematic and non-holonomic constraints of the wheels, simplifies the otherwise impenetrable constraints, resulting in explicit monotonic functions between the velocity of the base and that of the wheels. Based on this foundation, we present a closed-form solution that keeps all the wheels' steering and driving velocities within their corresponding pre-specified bounds. Simulation data and experimental results from executing the controller in a real-time environment demonstrate the efficacy of the method.**

I. Introduction

Wheeled Mobile Robots (WMRs) are the most predominant type of mobile robots. They have been prevalent in research, exploration, and industrial set-ups, and advances in service robotics are bringing them even closer to humans' lives. One of the main areas of research in the field of WMRs is their motion control. The continuing trend in developing autonomous and intelligible robots demands more reliable and higher performance motion controllers for WMRs.

However, the mechanism complexity and the intrinsic constraints of wheels make motion control design a challenging task. The multitude of motion controllers proposed for mobile robots, especially in the presence of non-holonomic constraints, are roughly classified into three branches [1]: point stabilization [2], trajectory tracking [3], and path-following [4]. The path-following approach, which this paper explores, maneuvers a robot toward a desired path and steers it so that it follows the path indefinitely.

In the path-following approach, the motion controller is decoupled into two phases: planning a path and following it. In that sense, the advances of path planners for mobile robots in recent years have helped with the popularity of path-following controllers. Early notable works in this field has been done by [5], [6]. There have been various extensions of the original problem, such as an extension to marine vehicles [7], [8], covering dynamics [9], [10], uncertainties [11] and actuator saturations [12]. However, the majority of the literature on this topic considers only a special case of WMR, which is mostly the unicycle type with some exceptions such as [13]. Nevertheless, there is no unified solution for the path-following of all types of WMRs.

Authors are with the Intelligent Hydraulics and Automation Department, Tampere University of Technology, 33101, Finland.
Corresponding author's email: oftadeh.reza@ieee.org.

WMRs are classified based on the eminent work of [14]; this classification has become the formal way of studying various WMRs and their kinematic and dynamic properties [15]. Four types of wheels are considered in this classification: *fixed wheels, standard steerable wheels, off-centered steerable wheels (Caster wheels), and Swedish wheels*. The number of wheels, their types, and their arrangements determine the degree of mobility and steerability of a WMR as well as its category. In this paper, we present an abstract generalized wheel encompassing the features of all those types. We use this generalization to obtain a unified form for the kinematic constraints that facilitate the integration of the path-following controller for all types of WMRs.

The contributions of this paper are as follows. We generalize our previous works on the path-following of four wheeled steering mobile robots [16], [17] to include all of the categories of WMRs in which their wheels roll without skidding. To the best knowledge of the authors, this is the first study that coherently solves the path-following problem for all the popular types of WMRs. Unlike other path-following controllers in the literature, in our design, the control laws and the resultant closed-loop equations of motion are *explicit* in terms of the WMR's speed. As a major peripheral result of this approach, we show that the closed-loop equations of motion represent a geometric path that steers the robot toward the desired path. As mentioned before, the kinematic constraints of all types of wheels are rigorously derived, and we show that the insertion of the path-following controller into the constraints yields to an explicit relationship between the velocity of the base and that of the wheels. Based on this set-up, we present a *closed-form* solution for the speed of the WMR so that all the wheels steering and driving velocities remain within their respective bounds. The solution is time-optimal, because at each time step at least one of the wheels runs at its maximum speed.

In Section II, we give some general descriptions of the employed notations and assumptions. We formally define the problem at hand in Section III and in Section IV, we present the path-following controller and derive the closed-loop equations of motion. The kinematic constraints of various types of wheels are derived in Section V and the path-following laws obtained in the former section are used to simplify the constraints into explicit functions of WMR velocity. In Section VI, we present the bounded velocity solution by utilizing the results obtained in previous sections. Finally, Section VII provides simulation and experimental results for the proposed controller.

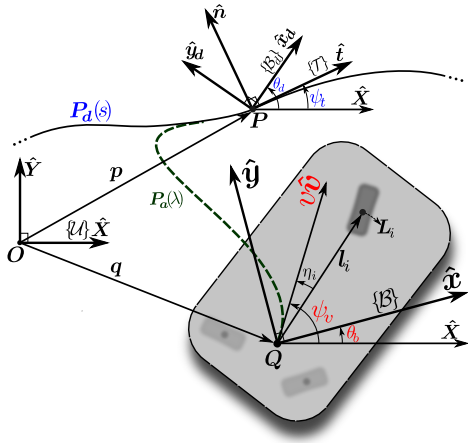


Fig. 1. The desired path and the required coordinate frames

II. Notations and Assumptions

In this paper, all vectors and matrices are in bold, with a hat-sign($\hat{}$) representing unit vectors. The coordinate frame that a vector is expressed in, is shown as a left superscript above the vector, which is omitted for vectors that are in the inertial frame. For the two arbitrary vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, $\boldsymbol{v}_1.\boldsymbol{v}_2$ and $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ are the inner and cross products of the two vectors, respectively.

The WMR considered here is equipped with n wheels attached to a main body called the *base*. Each wheel belongs to one of the following categories: *fixed wheels, standard steerable wheels, off-centered steerable wheels (Caster wheels), Swedish wheels*. We presume that the robot belongs to and possesses the minimum actuated wheels of one of the five kinematically feasible categories of WMRs classified in [14] and the actuators provide velocity and position control. The WMR traverses on a flat and horizontal plane. The base and the wheels are rigid, the tires are non-deformable, their contact surface with the ground can be approximated with a point and their spinning is pure rolling without any side slippage. Moreover, there is no mechanical constraint for the steering of the steerable wheels, hence they are free-turn.

Fig. 1, represents a schematic view of a WMR, the desired path, and the corresponding parameters. The desired path $\boldsymbol{P}_d$ is a 2D and bounded-curvature regular curve on the horizontal plane. It is defined by the vector-valued function $\boldsymbol{P}_d(s) : [0, L_d] \rightarrow \mathbb{R}^2$, where s and L_d are the natural parametrization and the length of $\boldsymbol{P}_d$, respectively and $C_c(s)$ represents its curvature as a function of s. The position vector $\boldsymbol{p}$ defines the point $\boldsymbol{P}$ on $\boldsymbol{P}_d$ as the instantaneous desired position. The tangent frame $\mathcal{T}\{\hat{\boldsymbol{t}}, \hat{\boldsymbol{n}}\}$ is the FrenetSerret frame of $\boldsymbol{P}_d$ at the point $\boldsymbol{P}$ with the tangent angle ψ_t expressing its rotation with respect to the inertial coordinate frame $\mathcal{U}\{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\}$. The desired heading function $\theta_d(s) : [0, L_d] \rightarrow \mathbb{R}$ of class C^2 determines the base's desired heading θ_d, which in turn represents the frame $\mathcal{B}_d\{\hat{\boldsymbol{x}}_d, \hat{\boldsymbol{y}}_d\}$ located at $\boldsymbol{P}$.

As for the WMR, the vector $\boldsymbol{q}$ defines the base position at the point $\boldsymbol{Q}$ with respect to the inertial frame. The body-fixed frame $\mathcal{B}\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ defines the base heading angle: θ_b. The frame $\mathcal{B}_v\{\hat{\boldsymbol{v}}, \hat{\boldsymbol{u}}\}$ at $\boldsymbol{Q}$ is the base velocity frame; that is, the unit vector $\hat{\boldsymbol{v}}$, defined by the velocity angle ψ_v, determines the direction of the robot's base linear velocity vector, and the scalar v represents its magnitude. Moreover, the i^{th} wheel, which belongs to one of the aforementioned categories of wheels, is attached to the base at the point $\boldsymbol{L}_i$, and the vector $\boldsymbol{\ell}_i$(with a magnitude of l_i) determines its position with respect to the base's frame origin: $\boldsymbol{Q}$.

Finally, the path $\boldsymbol{P}_a(\lambda)$ is the asymptotic path of the closed-loop system that guides the WMR toward and along the desired path $\boldsymbol{P}_d$. It is defined by $\boldsymbol{P}_a(\lambda) : [0, L_a] \rightarrow \mathbb{R}^2$, where λ and L_a are the natural parametrization and length of the path, respectively. In the absence of disturbances and uncertainties, when the WMR perfectly follows the path-following control signals, $\boldsymbol{P}_a(\lambda)$ becomes the actual footprint of the base of the WMR at the point $\boldsymbol{Q}$ that asymptotically converges to $\boldsymbol{P}_d$. We will show that the closed-loop error states derived from the designed controller result in a set of differential equations for $\boldsymbol{P}_a(\lambda)$.

III. Problem Definition

As shown in Fig. 1, the base's absolute and relative poses with respect to the desired path and heading are expressible with four variables: the instantaneous desired point $\boldsymbol{P}$, which is determined by s, along with three error signals, which are x_e and y_e for the position, and θ_e for the heading. Hence, the error signals are

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = {}^{\mathcal{U}}\boldsymbol{R}_{\mathcal{T}}^{-1}(\boldsymbol{q} - \boldsymbol{p}) \tag{1a}$$

$$\theta_e = \theta_d - \theta_b, \tag{1b}$$

where ${}^{\mathcal{U}}\boldsymbol{R}_{\mathcal{T}}$, equivalent to $\boldsymbol{R}(\psi_t)$, is the rotation matrix from frame $\mathcal{T}$ to frame $\mathcal{U}$. In other words, x_e and y_e are measured along $\hat{\boldsymbol{t}}$ and $\hat{\boldsymbol{n}}$, respectively. The time derivation of eqs. (1a) and (1b) yields to the open-loop equations of motion

$$\dot{x}_e = \dot{s}(C_c(s)y_e - 1) + v\cos(\psi_t - \psi_v) \tag{2a}$$

$$\dot{y}_e = -\dot{s}C_c(s)x_e - v\sin(\psi_t - \psi_v) \tag{2b}$$

$$\dot{\theta}_e = \frac{\partial \theta_d}{\partial s}\dot{s} - \omega_b, \tag{2c}$$

in which the angular velocity of the frame $\mathcal{B}$ is $\omega_b \triangleq \dot{\theta}_b$.

Problem 1: Given the desired path $\boldsymbol{P}_d(s)$ and heading profile $\theta_d(s)$, derive *feedback control laws* for the wheels' driving and steering commands such that:

1) Path-Following: The velocity frame $\mathcal{B}_v$ converges and follows the tangent frame $\mathcal{T}$; that is, error signals x_e, y_e remain bounded and converge at zero. See eq. (1a).
2) Heading Control: The body frame $\mathcal{B}$ converges and follows $\mathcal{B}_d$; that is, the error signal θ_e remains bounded and converges to zero. See (1b).
3) Bounded Velocity: The driving velocities of the wheels and steering velocities of the steerable wheels do not exceed the predefined limits.

In the following, we solve the abovementioned problem in two stages. In the first stage, the base speed v is left as a

free variable and the control laws for the base linear velocity direction, ψ_v or its rate, $\dot{\psi}_v$, and the base angular velocity, ω_b, are derived, resulting in an asymptotically stable solution for the path-following and the heading control. In the second stage, we take into account the kinematic constraints imposed by the WMR's wheels.

We solve the path-following problem with an independent heading for a base divested of all the wheels' constraints. At first glance, this approach is solely applicable to holonomic omnidirectional WMRs ($\delta_m = 3$). However, this treatment is a rather general case, and we will show that, for each category of WMRs, a proper subset of the derived control signals along with a pertinent choice of body origin $\boldsymbol{Q}$ results in a feasible solution that abides by the kinematic limitations of that category.

IV. The Path-Following Controller

A. Derivation of Control Laws

In this section, we solve sub-problems 1 and 2 of Problem 1. Given the error state equations; eqs. (1a) and (1b), the objective is to design feedback control laws for ψ_v, ω_b, and $\dot{s}$. Here, s is an *auxiliary* state and $\dot{s}$ is its corresponding control signal.

First, we define ψ_d, the desired input for ψ_v, as

$$\psi_d = \psi_t - \sigma(y_e), \tag{3}$$

in which, $\sigma(y_e)$ is a function that generates a suitable approach angle from the base to $\boldsymbol{P}_d(s)$ and has the following features: $\sigma(0) = 0$ and $y_e\sigma(y_e) > 0 \;\; \forall y_e \neq 0$. One candidate for $\sigma(y_e)$ is

$$\sigma(y_e) \triangleq \sin^{-1}\frac{k_2 y_e}{|y_e| + \epsilon}, \tag{4}$$

where $0 < k_2 \leq 1$ and $\epsilon > 0$.

Proposition 1: The feedback control laws $\omega_{b,c}$, and $\psi_{v,c}$, which are control commands for ω_b and ψ_v, respectively, and $\dot{s}$ are given by

$$\dot{s} = (k_1 x_e + \cos\sigma(y_e))v \triangleq k_s v \tag{5a}$$

$$\omega_{b,c} = (k_3\theta_e + \frac{\partial\theta_d}{\partial s}k_s)v \triangleq k_{b,c}v \tag{5b}$$

$$\psi_{v,c} = \psi_d, \tag{5c}$$

where $k_1, k_3 > 0$, resulting in x_e, y_e and θ_e to asymptotically converge to zero. Consequently, the origin of the error space is stable and it becomes semi-globally exponentially stable by setting $v(t) \geq v_m > 0 \;\; \forall t$. The stability proof is given in Appendix A.

In this design, the linear velocity direction ψ_v is open-loop and is set directly. As we will show, this approach is convenient when the WMR is equipped with a steerable wheel and the steering angle is being directly controlled. However, when the kinematic constraints impose velocity control on the steering angles or there is no steerable wheel, it is suitable to have ψ_v as a state and to incorporate the error state

$$\psi_e = \psi_d - \psi_v \tag{6}$$

into the system.

Proposition 2: Defining $\omega_v = \dot{\psi}_v$, the control signals in eq. (5a) and eq. (5b) and $\omega_{v,c}$, which is the control signal for ω_v, are given by

$$\omega_{v,c} = C_c\dot{s} - \sigma'\dot{y}_e - v y_e\Delta + k_4 v\psi_e \tag{7a}$$

$$\Delta = \begin{cases} \frac{\sin(\psi_t - \psi_v) - \sin\sigma(y_e)}{\psi_e} & y_e \neq 0 \\ 1 & y_e = 0 \end{cases}, \tag{7b}$$

in which, $\sigma'(y_e) = \frac{\partial\sigma(y_e)}{\partial y_e}$, and $k_4 > 0$, result in x_e, y_e,θ_e and ψ_e to asymptotically converge to zero. The proof is given in AppendixA.

B. Closed-Loop Equations of Motion

The actual path taken by the base is denoted as $P_a(\lambda)$. λ is the natural parameterization of the robot's actual path, and consequently $||\dot{\mathbf{q}}|| = v = \dot{\lambda}$. Substituting the control laws (eqs. (5a) to (5c)) into the open-loop error states (eqs. (2a) to (2c)) results in

$$\dot{s} = k_s v \quad \dot{x}_e = k_x v \quad \dot{y}_e = k_y v \quad \dot{\theta}_e = k_\theta v, \tag{8}$$

which are the closed-loop equations of motion, and,

$$k_s \triangleq \frac{\partial s}{\partial\lambda} = k_1\; x_e + \cos\sigma(y_e) \tag{9a}$$

$$k_x \triangleq \frac{\partial x_e}{\partial\lambda} = k_s\,(C_c y_e - 1) + \cos\sigma(y_e) \tag{9b}$$

$$k_y \triangleq \frac{\partial y_e}{\partial\lambda} = -\left(k_s C_c x_e + \sin\sigma(y_e)\right) \tag{9c}$$

$$k_\theta \triangleq \frac{\partial\theta_e}{\partial\lambda} = -k_3\theta_e. \tag{9d}$$

Hence, at each instant of time, t, $\boldsymbol{P}_a(\lambda)$ and other kinematic variables of the base are determinable by integrating the abovementioned equations. In other words, the path-following controller acts as a feedback path-planner that plans asymptotic paths from the WMR toward and onto the desired path. Therefore, similar to other path-following controllers [4], [10] for any base velocity profile $v \geq v_m > 0$, the WMR asymptotically follows the desired pose. However, in those controllers while v does not have a direct role in the stability, an asymptotic path, $\boldsymbol{P}_a$, cannot be determined for any arbitrary profile of $v(t)$, and it is only after the assignment of the velocity profile that one can derive the asymptotic trajectory $\boldsymbol{P}_a(\lambda(t))$. On the contrary, the choice of control signals in the previous section enables us to determine $\boldsymbol{P}_a(\lambda)$ independent of the future velocity commands by directly integrating the closed-loop equations.

C. Reformulation of the control laws

As a direct consequence of the control design obtained in the previous section, it is feasible to rearrange the control laws and their time differentiations into explicit functions of the base speed v, which in turn facilitates the simplification of the kinematic constraints in the next section. With the help of eqs. (9a) and (9c), we may rewrite $\dot{\psi}_d$ and $\omega_{v,c}$, given in eq. (7a), as

$$\dot{\psi}_d = (C_c(s)k_s - \sigma' k_y)v \triangleq k_d v \tag{10a}$$

$$\omega_{v,c} = (C_c(s)k_s - \sigma' k_y - y_e\Delta + k_4\psi_e)v \triangleq k_{v,c}v. \tag{10b}$$

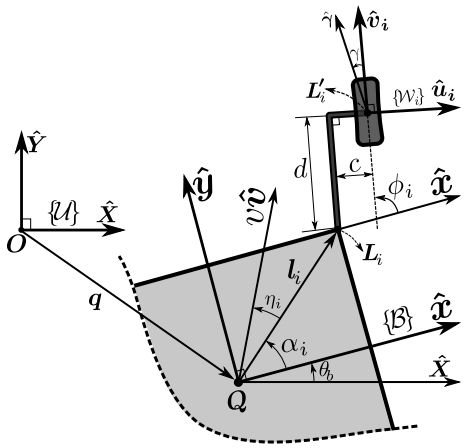


Fig. 2. A generalized wheel and its corresponding parameters

Similarly, time derivatives of k_d, $k_{v,c}$, $k_{b,c}$ are again in the form of

$$\dot{k}_d = k'_d v, \quad \dot{k}_{v,c} = k'_{v,c} v, \quad \dot{k}_{b,c} = k'_{b,c} v, \tag{11}$$

and, hence, for the closed-loop system, the time derivatives of the control laws are

$$\ddot{\psi}_d = k'_d v^2 + k_d \dot{v} \tag{12a}$$
$$\dot{\omega}_{v,c} = k'_{v,c} v^2 + k_{v,c} \dot{v} \tag{12b}$$
$$\dot{\omega}_{b,c} = k'_{b,c} v^2 + k_{b,c} \dot{v}, \tag{12c}$$

where the corresponding expressions for all of the above variables are given in Appendix B.

Collectively, we conclude that, for any subset of the control laws, the angular velocity of the base, ω_b and the angular rate of its linear velocity, ω_v, along with their accelerations are expressible as:

$$\omega_v = k_v v, \qquad \omega_b = k_b v \tag{13a}$$
$$\dot{\omega}_v = k'_v v^2 + k_v \dot{v}, \qquad \dot{\omega}_b = k'_b v^2 + k_b \dot{v}. \tag{13b}$$

V. The kinematic constraints of the wheels

In this section, we derive the kinematic constraints for all the typical types of wheels that are commonly used in WMRs. Incorporating the results of the control laws obtained in previous sections, the kinematic constraints of each type are analyzed and reduced to explicit functions between the wheels' velocities and the base speed v. This approach enables us to solve sub-problem 3 of Problem 1 by finding the right instantaneous values for v that keep the velocities of all the actuators bounded.

Fig. 2 depicts a schematic view of an abstract Generalized Wheel(GW) as the i^{th} wheel of the WMR and its corresponding parameters. The GW represents both types of Swedish wheels and normal wheels. In this sense, r_{sr} and γ define the radius and the direction of the small rollers' axis, respectively, hence, for a functional Swedish wheel: $\gamma \neq \frac{\pi}{2}$ and $r_{sr} \neq 0$. For a normal non-Swedish wheel, we simply set $\gamma = \frac{\pi}{2}$ and $r_{sr} = 0$. The wheel is mounted on an L-shaped rod parametrized by off-center values: (d, c), at the point $\boldsymbol{L}'_i$. The rod is connected to the base at the attachment point $\boldsymbol{L}_i$ by a revolute joint. As shown in the figure, ϕ_i represents the steering angle of the wheel, and $v_i \hat{\boldsymbol{v}}_i$ represents its *driving velocity vector*, generated by the wheel's actuator; thus, it is generally different than the velocity of $\boldsymbol{L}'_i$.

For the GW, the velocity constraint between the wheel and the base is

$$\begin{aligned} v\, {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) =& (v_i - c(\omega_b + \dot{\phi}_i))\, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i \\ &+ d(\omega_b + \dot{\phi}_i)\, {}^{\mathcal{B}}\hat{\boldsymbol{u}}_i \\ &- r_{sr}\dot{\phi}_{sr}(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\hat{\boldsymbol{\gamma}}), \end{aligned} \tag{14}$$

in which, $\dot{\phi}_{sr}$ is the angular velocity of the GW's small rollers. For a non-Swedish GW($\gamma = \frac{\pi}{2}, r_{sr} = 0$), we must differentiate from the above equation with respect to time. After some algebraic manipulations, the scalar equation

$$\begin{aligned} v_i d(\dot{\omega}_b + \ddot{\phi}_i) - \dot{v}_i d(\omega_b + \dot{\phi}_i) =& \dot{\phi}_i(v^2 + \omega_b^2 l_i^2 + 2 l_i v \omega_b \sin\eta_i) \\ &+ v(\omega_b - \omega_v)(v + \omega_b \sin\eta_i) \\ &+ (\dot{v}\omega_b - \dot{\omega}_b v)\cos\eta_i. \end{aligned} \tag{15}$$

represents the steering velocity $\dot{\phi}_i$.

Next, we derive the kinematic constraints for each type of wheel from the GW constraints; see eqs. (14) and (15). For each type, we investigate the conditions for the validity of the constraints and simplify them based on the general control laws; see eqs. (13a) and (13b).

A. Fixed Wheel

For a standard fixed wheel, there is no off-center rod, $\boldsymbol{L}'_i$ coincides with $\boldsymbol{L}_i$, and there is no steering, $\phi_i = \text{cte}$. Hence, GW is a standard fixed wheel with: $\{d = 0, c = 0, \dot{\phi}_i = 0, \gamma = \frac{pi}{2}, r_{sr} = 0\}$. Setting these values in eqs. (14) and (15) results in

$$v_i\, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = v\, {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) \tag{16a}$$
$$0 = v(\omega_b - \omega_v)(v + \omega_b \sin\eta_i) + (\dot{v}\omega_b - \dot{\omega}_b v)\cos\eta_i. \tag{16b}$$

Based on eq. (16b), v is chosen arbitrarily if $\omega_b = \omega_v$ and $\cos\eta_i = 0$. These conditions imply that the body frame $\boldsymbol{\mathcal{B}}$ must be placed on the wheel's axis. Note that, all the fixed wheels of a WMR must be coaxial and consequently the point $\boldsymbol{Q}$, origin of $\boldsymbol{\mathcal{B}}$, should be placed on that common axis. Using eq. (13a), eq. (16a) can be rearranged as

$$v_i = v\sqrt{1 + l_i^2 k_b^2 + 2 l_i k_b \sin\eta_i} \tag{17a}$$
$${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = \frac{{}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b(\hat{z} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)}{v_i}. \tag{17b}$$

Based on the aforementioned conditions, for a WMR equipped with fixed wheels, ${}^{\mathcal{B}}\hat{\boldsymbol{v}}$ is the direction of the WMR's heading, and we determine the right choice for k_b in the next section based on the category of the WMR that incorporates the wheel. ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ is always normal to the wheel's axis, however, we assume that $v_i > 0 \quad \forall t$ and eq. (17b) determines the driving direction(backward or forward).

B. Swedish Wheel

The standard Swedish wheel is a GW with $\{d = 0, c = 0, \dot{\phi}_i = 0, \gamma \neq \frac{pi}{2}, r_{sr} \neq 0\}$. Hence, $\boldsymbol{L}'_i$ coincides with $\boldsymbol{L}_i$, and eq. (14) reduces to

$$v_i \, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i - r_{sr}\dot{\phi}_{sr}(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\hat{\boldsymbol{\gamma}}) = v \, {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i). \quad (18)$$

Dot multiplying the above equation with $\hat{\boldsymbol{\gamma}}$, the Swedish kinematic constraint for the closed-loop system becomes

$$v_i = \frac{v \, {}^{\mathcal{B}}\hat{\boldsymbol{\gamma}}.({}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i))}{{}^{\mathcal{B}}\hat{\boldsymbol{\gamma}} \, . \, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i}. \quad (19)$$

Note that, contrary to the fixed wheel, we choose an arbitrary direction for ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ but normal to the wheel's axis, and consequently the sign of v_i determines the right direction(backward or forward) of the driving velocity.

C. Standard Steerable Wheel

The standard steerable wheel is a GW with $\{d = 0, c, \dot{\phi}_i(t), \gamma = \frac{pi}{2}, r_{sr} = 0\}$. For this type of wheels, c may be zero [18] or non-zero [19]. The wheel steering angle ϕ_i is actuated and determines the wheel heading with respect to the base. For this type of wheel, eqs. (14) and (15) reduce to

$$v \, {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) = (v_i - c(\omega_b + \dot{\phi}_i)) \, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i \quad (20a)$$

$$\dot{\phi}_i(v^2 + \omega_b^2 l_i^2 + 2 l_i v \omega_b \sin\eta_i) = v(\omega_v - \omega_b)(v + \omega_b \sin\eta_i) + (\dot{\omega}_b v - \dot{v}\omega_b)\cos\eta_i. \quad (20b)$$

Again, incorporating eqs. (13a) and (13b), the above kinematic constraints are simplified to

$${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i = \frac{{}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b(\hat{z} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)}{\sqrt{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i}} \quad (21a)$$

$$\dot{\phi}_i = \frac{k'_b l_i \cos\eta_i + (k_v - k_b)\,(1 + l_i k_b \sin\eta_i)}{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i} v \triangleq \phi'_i v \quad (21b)$$

$$v_i = \left(c(k_b + \phi'_i) + \sqrt{1 + k_b^2 l_i^2 + 2 l_i k_b \sin\eta_i} \right) v. \quad (21c)$$

In the these equations, ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ determines the steering angle ϕ_i independent of the base speed v, and the derived expressions for driving and steering velocities, v_i, and $\dot{\phi}$, are explicit in terms of v.

D. Caster Wheel

The caster wheel, or off-centered steerable wheel, is an omnidirectional wheel and a GW with $\{d \neq 0, c = 0, \dot{\phi}_i(t), \gamma = \frac{pi}{2}, r_{sr} = 0\}$. Hence, eq. (14) reduces to

$$v \, {}^{\mathcal{B}}\hat{\boldsymbol{v}} + \omega_b(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) = v_i \, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i + d(\omega_b + \dot{\phi}_i) \, {}^{\mathcal{B}}\hat{\boldsymbol{u}}_i. \quad (22)$$

Given the omnidirectional nature of the wheel, and contrary to standard orientable wheels, the steering angle assumes any instantaneous arbitrary value. In this design, the steering angle and therefore ${}^{\mathcal{B}}\hat{\boldsymbol{v}}_i$ and ${}^{\mathcal{B}}\hat{\boldsymbol{u}}_i$, are measured from feedback and the steering velocity $\dot{\phi}_i$ is controlled. Hence, for the closed-loop system, eq. (22) simplifies to

$$v_i = v({}^{\mathcal{B}}\hat{\boldsymbol{v}}_i.{}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b \, {}^{\mathcal{B}}\hat{\boldsymbol{v}}_i.(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i)) \quad (23a)$$

$$\dot{\phi}_i = \frac{v}{d}({}^{\mathcal{B}}\hat{\boldsymbol{u}}_i.{}^{\mathcal{B}}\hat{\boldsymbol{v}} + k_b \, {}^{\mathcal{B}}\hat{\boldsymbol{u}}_i.(\hat{\boldsymbol{z}} \times {}^{\mathcal{B}}\boldsymbol{\ell}_i) - d\,k_b). \quad (23b)$$

Note that although here we set $c = 0$, similar but slightly more complex results are achievable with $c \neq 0$.

VI. Bounded velocity controller for WMRs

In the previous section, we showed that the sets $\mathcal{C}_{im} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}, \omega_b, \dot{\omega}_b, \omega_v \, \dot{\omega}_v\}$ or $\mathcal{C}_{ex} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}, k_b, k'_b, k_v, k'_v\}$ determine the velocities and directions of the wheels as implicit or *explicit* functions of v, respectively. In this section, first, for each category of WMRs, we determine the right subset of control laws obtained in section IV and *map* them to $\mathcal{C}_{im}$ and $\mathcal{C}_{ex}$. Second, we incorporate the explicit kinematic functions and present a closed-form solution for instantaneously finding the base speed v such that all the actuator's velocities remain within the pre-specified bounds.

A. The selection of control signals

WMRs are classified based on the ordered pair $\boldsymbol{\delta} = (\delta_m, \delta_s)$ into five different categories. Three of these categories possess the degree of maneuverability $\delta_M = \delta_m + \delta_s = 3$, and, for the other two $\delta_M = 2$. In the following, we select the right control signals and explain the accompanying details based on the WMR's degree of maneuverability δ_M.

1) WMRs with $\delta_M = 3$: These types of WMRs are omnidirectional in nature, which means they realize independent heading and linear movements. However, the holonomic type with $\boldsymbol{\delta} = (3, 0)$ provides full mobility and hence, instantaneous velocity in any direction. The other two categories($\boldsymbol{\delta} = (2, 1)$ and $\boldsymbol{\delta} = (1, 2)$) are steerable and non-holonomic. They are capable of providing movement in any arbitrary direction but only after they have steered their wheels to the corresponding configuration. For the problem at hand, the difference between holonomic and non-holonomic types is only at the beginning of the path in which the holonomic type may start the path-following instantly, but the non-holonomic types have to rearrange their steerable wheels. Other than this, on a smooth path and heading profile, both types provide the same functionality. Therefore, for this type of WMR:

- The body frame $\mathcal{B}$ is chosen arbitrarily.
- The heading and linear movements are independent. Hence, the controller's inputs are both $\boldsymbol{P}_d$ and θ_d.
- There are two options for $\mathcal{C}_{im}$ and $\mathcal{C}_{ex}$. If ψ_v and, consequently, ${}^{\mathcal{B}}\hat{\boldsymbol{v}}(\psi_v)$ are set to be a state(${}^{\mathcal{B}}\hat{\boldsymbol{v}}(\mathcal{S})$) and

$$\mathcal{C}_{im} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}(\mathcal{S}), \omega_{b,c}, \dot{\omega}_{b,c}, \omega_{v,c} \, \dot{\omega}_{v,c}\}$$
$$\mathcal{C}_{ex} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}(\mathcal{S}), k_{b,c}, k'_{b,c}, k_{v,c}, k'_{v,c}\}. \quad (24)$$

Conversely, if the control for ψ_v and ${}^{\mathcal{B}}\hat{\boldsymbol{v}}(\psi_v)$ is set to be open-loop,

$$\mathcal{C}_{im} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}(\psi_d), \omega_{b,c}, \dot{\omega}_{b,c}, \dot{\psi}_{v,c} \, \ddot{\psi}_{v,c}\}$$
$$\mathcal{C}_{ex} = \{{}^{\mathcal{B}}\hat{\boldsymbol{v}}(\psi_d), k_{b,c}, k'_{b,c}, k_{d,c}, k'_{d,c}\}. \quad (25)$$

2) WMRs with $\delta_M = 2$: These types of WMRs have limited mobility in their working plane and the heading and linear movements are dependent. They are either *differential* with $\boldsymbol{\delta} = (2, 0)$ or *carlike* with $\boldsymbol{\delta} = (1, 1)$. Both categories have a set of coaxial fixed wheels. Again, the only difference between these categories is at the beginning of the path-following. The *differential* type starts the path-following instantly, but the *carlike* type has to steer its steerable wheel according to the start of the path. Aside from this difference, both types provide the same functionality on a smooth path. Therefore, for this type of WMR:

- The body frame $\boldsymbol{\mathcal{B}}$ is chosen on the common axis of the fixed wheels.
- The heading and linear movements are dependent. Hence, the input for the controller is only $\boldsymbol{P}_d$.
- There is only one viable option for $\mathcal{C}_{im}$ and $\mathcal{C}_{ex}$. $\psi_v = \theta_b$, and consequently ${}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}(\psi_v) = \hat{\boldsymbol{x}}$, is the base heading and a state(${}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}(\mathcal{S})$). Hence, the selection for control signals are

$$\mathcal{C}_{im} = \{{}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}(\mathcal{S}), \omega_{v,c}, \dot{\omega}_{v,c}, \omega_{v,c}\,\dot{\omega}_{v,c}\}$$
$$\mathcal{C}_{ex} = \{{}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}(\mathcal{S}), k_{v,c}, k'_{v,c}, k_{v,c}, k'_{v,c}\}. \qquad (26)$$

B. The bounded velocity solution

In this section, we present the bounded velocity solution for the base speed v. For a WMR with a specific $\boldsymbol{\delta} = (\delta_m, \delta_s)$, the base controller is designed based on the classification provided in the previous section and once the proper $\mathcal{C}_{ex}$ is selected. Then, the $\mathcal{C}_{ex}$ is used to derive the explicit kinematic constraints given in Section V for each wheel of the WMR. For a given v, those explicit constraints are used to derive the actuators' commands for the velocities and directions of the wheels.

The number of the WMR's actuators is denoted as $m <= 2n$, of which k of them are driving actuators and $m - k$ of them are steering actuators. If the j^{th} actuator is driving, the maximum driving velocity of the actuator is denoted as $V_{j,max}$. Otherwise, if the j^{th} actuator is steering, the maximum steering velocity is denoted as $\dot{\phi}_{j,max}$. The command for the base speed v, namely v_c, is derived using the following procedure:

1) At each step of time, evaluate $\mathcal{C}_{ex}$.
2) For the j^{th} actuator mounted on the i^{th} wheel: Based on the type of wheel, determine the corresponding explicit equations for the wheel's driving and steering. If the actuator is steering, substitute $\dot{\phi}_i$ with $\dot{\phi}_{j,max}$ in the steering velocity equation. If the actuator is driving, substitute v_i with $V_{j,max}$ in the driving velocity equation. Each case results in a v that is the j^{th} candidate for the base speed command v_c, denoted as $v_{c,j}$.
3) Evaluate 2 for all the m actuators.
4) There are m candidates for v, and we select v as

$$v_c = \min(|v_{c,j}|), j \in \{1, 2, .., m\}. \qquad (27)$$

5) Use v_c and $\mathcal{C}_{ex}$ to evaluate the actuators' commands.

Note that the explicit constraints of Section V, which are used in the second step of the above procedure to derive the velocity candidates, are strictly monotonic with respect to v_i, and $\dot{\phi}_i$ and so is their inverse with respect to v. Hence, applying the minimum of those m velocity candidates results in driving and steering velocities less than or equal to the given velocity bounds. In other words, at each instant, at least one of the actuators is being driven at its maximum velocity, which renders the solution as a bang-bang control [20] for the velocity v and, therefore, for a given desired path, heading, and control gains, the solution is time-optimal.

VII. Simulation and Experimental Results

In this section, we present some experimental and simulation data for the path-following controller of three types of WMRs. We performed the experiments using our four wheel steering mobile manipulator called iMoro [18], which is shown in Fig. 3. iMoro is a non-holonomic omnidirectional WMR($\delta_m = 1, \delta_s = 2$) also known as two-steer [15]. After fixing the rear wheels, it can also emulate the car-like type ($\delta_m = 1, \delta_s = 1$). We have implemented the controller for both modes on iMoro in a real-time Linux environment based on [21]. The iMoro footprint in two-steer and car-like modes are shown in Figures 4 and 5, respectively. As depicted in the figures, we have intentionally set large initial errors (the WMR is facing away from the desired path) to demonstrate the performance of the controller. The implemented modes are typical cases of the two major categories of WMRs ($\delta_M = 3$ and $\delta_M = 2$) that we discussed in Section VI-B. Figures 7 and 8 depict the angular and driving velocity commands, generated by the controller in the two-steer mode and Fig. 9 depicts the same control signals for the driving velocities of the wheels in the car-like mode. The figures show that the controller efficiently keeps the velocities within their set limits.

For further validation, we have also simulated a holonomic omnidirectional WMR with Swedish wheels. The simulated robot has the same architecture as iMoro, but the only difference is that the steering wheels are replaced by Swedish wheels of the same radius. Fig. 6 shows its footprint under the same initial errors and the desired path and 10 presents its bounded driving velocities.

Note that, for the two-steer mode implemented with iMoro and the simulated holonomic omnidirectional($\delta_M = 3$) case, the given desired heading is for the robot to rotate 2π while performing the path-following. Furthermore, as discussed in Section V, the wheels' driving velocities are always positive in cases of having fixed or steering wheels while the actual direction of the wheels are determined by the vector ${}^{\boldsymbol{\mathcal{B}}}\hat{\boldsymbol{v}}_i$. Conversely, for the Swedish wheels, the sign of v_i derived from (19) determines the velocity direction. This arrangement is also clear in Figures 8, 9, and 10.

As shown in the figures, the proposed generic method navigates the WMRs while remaining within the velocity boundaries. However, in order to achieve higher velocity limits while performing tight maneuvers, bounding velocities are not enough; the accelerations should be bounded, too. We have presented a bounded acceleration solution for two-steer WMRs, such as iMoro, in [17].

Fig. 3. iMoro mobile Manipulator

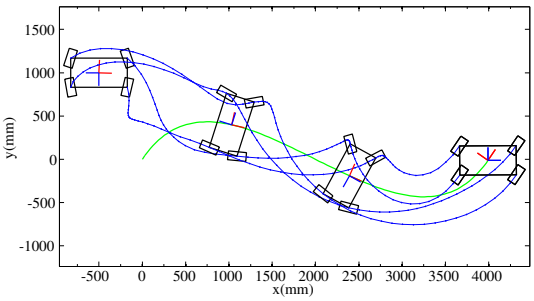


Fig. 4. Footprint of iMoro in two-steer mode

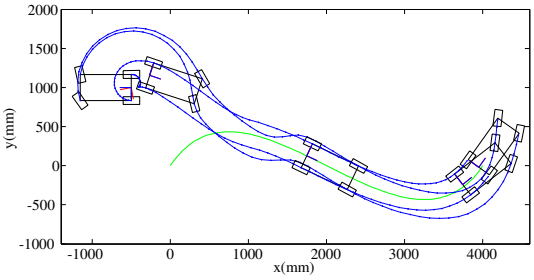


Fig. 5. Footprint of iMoro in car-like mode

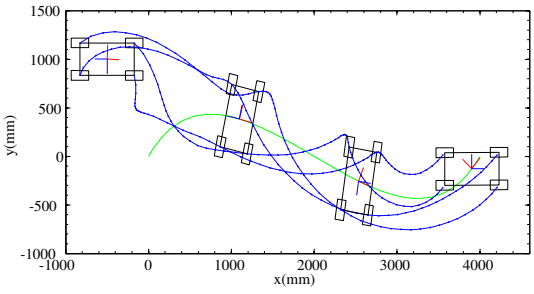


Fig. 6. Simulation of an omnidirectional WMR with Swedish wheels

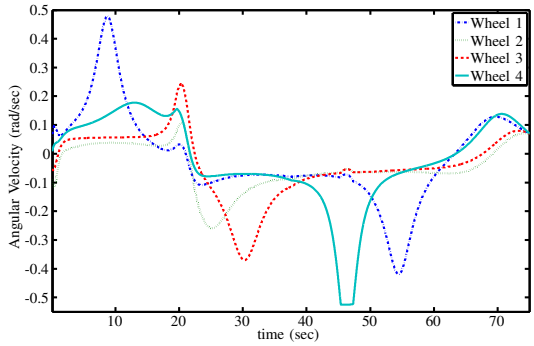


Fig. 7. iMoro in two-steer mode: Control Signals for steering

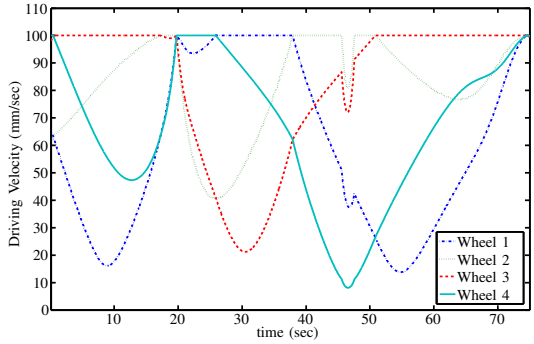


Fig. 8. iMoro in two-steer mode: Control signals for driving

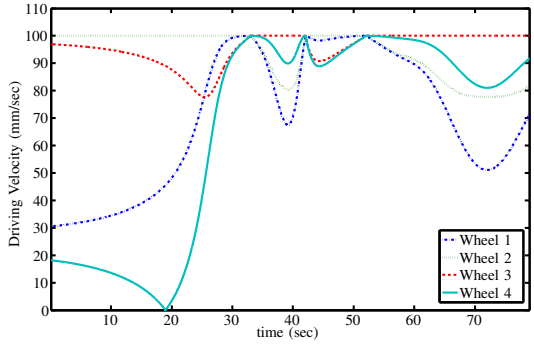


Fig. 9. iMoro in car-like mode: Control signals for driving

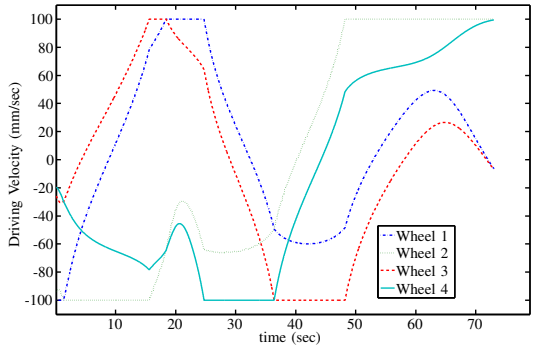


Fig. 10. Holonomic WMR simulation: Control signals for the Swedish wheels

VIII. Conclusion

In this paper, we have presented a bounded velocity path-following controller for Wheeled Mobile Robots(WMRs) operating under the condition of pure rolling without skidding. The solution is applicable to various types of WMRs such as car-like, differential drive, and omnidirectional, among others. The major advantage of this solution is that the path-following control laws for the base of the WMR extensively simplify the kinematic and non-holonomic constraints of the wheels. In this paper, we have used this advantage to derive a time-optimal solution for the speed of a WMR's base that keeps the velocities of the actuators within pre-specified bounds. We are currently working to extend the solution to cover bounded accelerations and dynamic uncertainties.

Appendix

A. The proofs of propositions 1 and 2

Proof: Here, we use similar Lyapunov functions as in [4], [10] but with modified control laws to make curvatures independent of speed v. Consider the following Lyapunov function:

$$V_1 = \frac{1}{2}x_e^2 + \frac{1}{2}y_e^2 + \frac{1}{2}\theta_e^2, \tag{28}$$

which is positive definite and radially unbounded. The time differentiation of V_1 along with the solution of eqs. (2a) to (2c) results in:

$$\dot{V}_1 = -(k_1 x_e^2 + k_2 \frac{y_e^2}{|y_e| + \epsilon} + k_3 \theta_e^2)\, v(t)\,, \tag{29}$$

which is negative; thus, the origin is stable. For a given $d_1 > 0$, if $v(t) \geq v_m > 0$ and initially $|y_e(t_0)| < d_1$, it is easy to show that $\dot{V}_1 < -\lambda V_1$. Thus, the origin is semi-globally exponentially stable.

In order to prove proposition 2, consider the following Lyapunov function:

$$V_2 = V_1 + \frac{1}{2}\psi_e^2. \tag{30}$$

The time differentiation of V_2 along the solution of eqs. (2a) to (2c) and (6) results in:

$$\dot{V}_2 = \dot{V}_1 - (k_4 \psi_e^2)\, v(t)\,, \tag{31}$$

which again is negative, and therefore the origin of the error state is stable. ∎

B. The closed-loop kinematic variables

Differentiating from (3) and (5b) respectively yields to

$$k_d = Cc(s)k_s - \frac{\partial \sigma\,(y_e)}{\partial y_e} k_y \tag{32a}$$

$$k'_{b,c} = -k_3^2\theta_e + \frac{\partial^2 \theta_d}{\partial s^2}k_s^2 + \frac{\partial \theta_d}{\partial s}k_1 k_x - \frac{\partial \theta_d}{\partial s}\frac{\partial \sigma\,(y_e)}{\partial y_e}k_y \sin\sigma(y_e). \tag{32b}$$

By differentiating from eqs. (10a) and (10b) and extracting v, similar results are acquirable for k'_d and $k'_{v,c}$.

References

[1] C. Samson, "Motion control of wheeled mobile robots," in *Springer Handbook of Robotics*. Springer, 2008, pp. 799–826.

[2] K. Park, H. Chung, and J. G. Lee, "Point stabilization of mobile robots via state-space exact feedback linearization," *Robotics and Computer-Integrated Manufacturing*, vol. 16, no. 5, pp. 353–363, 2000.

[3] J.-M. Yang and J.-H. Kim, "Sliding mode control for trajectory tracking of nonholonomic wheeled mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 15, no. 3, pp. 578–587, 1999.

[4] L. Lapierre, R. Zapata, and P. Lepinay, "Combined path-following and obstacle avoidance control of a wheeled robot," *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 361–375, 2007.

[5] A. Micaelli and C. Samson, "Trajectory tracking for unicycle-type and two-steering-wheels mobile robots," INRIA, Tech. Rep. 2097, 1993.

[6] M. Aicardi, G. Casalino, A. Bicchi, and A. Balestrino, "Closed loop steering of unicycle like vehicles via lyapunov techniques," *Robotics & Automation Magazine, IEEE*, vol. 2, no. 1, pp. 27–35, 1995.

[7] M. Aicardi, G. Casalino, G. Indiveri, A. Aguiar, P. Encarnaçao, and A. Pascoal, "A planar path following controller for underactuated marine vehicles," in *Proc. 9th Mediterranean Conference on Control and Automation*, 2001.

[8] L. Lapierre, D. Soetanto, and A. Pascoal, "Nonlinear path following with applications to the control of autonomous underwater vehicles," in *Decision and Control, 2003. Proceedings. 42nd IEEE Conference on*, vol. 2. IEEE, 2003, pp. 1256–1261.

[9] R. Fierro and F. L. Lewis, "Control of a nonholonomic mobile robot: backstepping kinematics into dynamics," in *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, vol. 4. IEEE, 1995, pp. 3805–3810.

[10] D. Soetanto, L. Lapierre, and A. Pascoal, "Adaptive, non-singular path-following control of dynamic wheeled robots," in *Proceedings of 42nd IEEE Conference on Decision and Control (CDC)*, vol. 2. IEEE, 2003, pp. 1765–1770.

[11] L. Lapierre, D. Soetanto, and A. Pascoal, "Nonsingular path following control of a unicycle in the presence of parametric modelling uncertainties," *International Journal of Robust and Nonlinear Control*, vol. 16, no. 10, pp. 485–503, 2006.

[12] L. Lapierre and G. Indiverri, "Path-following control of a wheeled robot under actuation saturation constraints," in *IAV07 conference, 2007*, 2007.

[13] K. Kanjanawanishkul and A. Zell, "Path following for an omnidirectional mobile robot based on model predictive control," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3341–3346.

[14] G. Campion, G. Bastin, and B. Dandrea-Novel, "Structural properties and classification of kinematic and dynamic models of wheeled mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 1, pp. 47–62, Feb 1996.

[15] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[16] R. Oftadeh, R. Ghabcheloo, and J. Mattila, "A novel time optimal path following controller with bounded velocities for mobile robots with independently steerable wheels," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 4845–4851.

[17] ——, "Time optimal path following with bounded velocities and accelerations for mobile robots with independently steerable wheels," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014.

[18] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Real-time system integration for mobile manipulation," *International Journal of Advanced Robotic Systems*, vol. 11, no. 15, 2013.

[19] B. Graf, U. Reiser, M. Hagele, K. Mauz, and P. Klein, "Robotic home assistant care-o-bot® 3-product vision and innovation platform," in *Advanced Robotics and its Social Impacts (ARSO), 2009 IEEE Workshop on*. IEEE, 2009, pp. 139–144.

[20] N. Osmolovskii and H. Maurer, *Applications to Regular and Bang-Bang Control*, ser. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2012.

[21] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila, "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink," in *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on*. IEEE, 2012, pp. 274–279.

Publication V

Oftadeh, Reza, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Unified framework for rapid prototyping of linux based real-time controllers with matlab and simulink." In *Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on*, pp. 274-279. IEEE.

Unified Framework for Rapid Prototyping of Linux based Real-Time Controllers with Matlab and Simulink

Reza Oftadeh, Mohammad M. Aref, Reza Ghabcheloo, Jouni Mattila

***Abstract*— We propose a systematic solution for real-time software development for safety critical mechatronic systems. The solution is based on Matlab/Simulink toolboxes and off-the-shelf drivers provided by hardware manufacturers, to address software development challenges in the area of PC based automation. In many cases, developers especially control systems designers found themselves immersed in technical difficulties of real-time programming and hardware interfacing. The remote development environment described here is used to develop real-time software based on Linux operating systems. Unlike other solutions that supports only limited interfaces, it demonstrates systematic methodology to develop reusable Simulink blocks for communicating with wide variety of device drivers and services. Some examples are given based on Xenomai real-time Linux. As a case study, the software development for a mobile robot based on this methodology is presented. The models and blocks developed for this study are available to interested developers for download and test.**

I. Introduction

A. Background study

Inherent complexity of a mechatronic system as a multidisciplinary field necessitates efforts in various aspects. Such a system should be a common language between software engineers and control engineers at many design and implementation levels correlating with the development of such systems. In many cases, these systems are basis for iterative verification and validation of different research projects, commercial products or educational purposes.

In either cases, certain parts of the system require focus of designers and developers while for other parts utilizing already available subsystems is considered a fair deal. The mechatronic system considered here consist of one or more embedded PC with real-time Linux as the main processing unit connected to different hardware components via distinctive mediums with appropriate interfaces. Hence, these systems fall into the category of PC automation solutions.

There are different issues associated with the development of such mechatronic systems especially in the low level that covers device drivers and system controllers. One example of such difficulties is interfacing with hardware devices which includes different communication mediums and protocols. Moreover, developing such systems requires numerous tests in order to verify and validate functionality and integrity of the developed algorithms and codes. Ultimately, the process should lead to optimized implementation of controllers that could receive sensors data and produce control signals in guaranteed intervals of time. In this regard, while software engineers prefer direct C\C++ programming, control engineers and system developers prefer Matlab and Simulink or similar software or toolboxes because of their ready to use advanced mathematical functions and algorithms that could be utilized directly. However, this high-level programming makes the models developed in Matlab not suitable for real-time tasks.

Several research works have addressed real time implementation of algorithms in Matlab [1] and [2] or in Matlab integration with Linux [3]. Moreover, there are available software packages utilizing selected functionalities of Matlab in their real-time execution environment such as xPC Targeting toolbox [4], RT-Lab [5], RTAI-Lab [6]. Mathworks has addressed this problem by embedding code generator products which automatically generate optimized, independent and royalty free C\C++ code from Matlab codes and Simulink models. Comparing generated C\C++ codes with their original Matlab code is the subject of reserach in [7].

Without doubt, one of the main challenging issues in a mechatronic system development is to make interfaces between the control software and the sensors and actuators. This issue has been addressed in dSpace® products by means of the complementary hardware in its package for rapid prototyping or code generation [8]. Another comprehensive work has been done by RTAI Linux team [9] by integration of RTAI-Lab into Simulink and translation of the code into the RTAI framework by means of *Real-Time-Workshop* (RTW). These systems are popular for a certain range of control and software requirements. However, their solutions enforce limitations compared to traditional C\C++ software development in reusability of existing codes. This problem is negligible until research and development team uses only compatible products to their development environment. In case of any unforeseen necessity of instruments, adaptation of a third-party driver to those environments is full of difficulties. On the other hand, a C\C++ application can easily use the drivers provided by the manufactures which are normally in C\C++.

On one hand, during design procedures of a complex mechatronic system, several teams of researchers work simultaneously on design processes and modeling of hardware and software platforms. Thus, even conceptual design and development of the product hardware can be subject to changes while the software is being developed. On the other hand, validation of the developed design models, methods and tools are done with the prototype experiments. It is desirable to benefit from modules and patterns of the already developed software components by reusing them. To address these challenges, we introduce a coherent framework for remote development of independent hard real-time software controllers based on Matlab and Simulink for the rapid prototyping of PC based mechatronic systems. In the proposed methodolody, we consider other criteria such as modularity, flexibility, reusability, compatibility with commercially off-the-shelf components, having dependable basis for future

Authors are with Intelligent Hydraulics and Automation (IHA) Department, Tampere University of Technology, 33101, Finland.
Corresponding authur email: reza.oftadeh@tut.fi

advances in safety measures such as *Safety Integrity Level* (SIL).

B. Proposed Method in a Nutshell

The method proposed here consists of developing software modules and controllers in a Simulink project on a PC with Windows operating system while the target is real-time Linux of one or several embedded PCs. Then a standalone and hard real-time C\C++ code is derived for real-time Linux operating system using code generation toolboxes. The resultant code is automatically transferred to the embedded target, and then compiled and built there. The final software would be ready to debug, run, or verify on the target.

For a complex mechatronic systems that software and controllers cannot be put in one executable, this method could be used simultaneously for multiple Simulink models to develop different parts of the whole software. Then each model can be turned into a module or service on the target PC that collaborates with other modules by means of different internal communication services such as programming queues. Hence, developers are able to benefit from Service Oriented Architecture (SOA) principles [10] in developing their software. Furthermore, the code can be generated for Windows as well as wide variety of operating systems that support *Portable Operating System Interface* (POSIX) [11].

Moreover this framework addresses some of the issues and difficulties commonly associated with remote development and code generation with Matlab and Simulink products. One of the main issues is how to integrate device drivers and hardware interfaces into Simulink model. Technically, those interfaces should be turned into reusable Simulink block libraries. However, these drivers and their related API libraries are installed on the remote target system (Real-time Linux) different than the operating system of development environment(Windows). Hence, deriving such blocks demand more advanced treatment. Addressing this issue requires detailed knowledge of S-Function programming as well as developing *Target Language Compiler* (TLC) files that tells code generators how to generate C code out of the S-Functions.

In this study, it is described how a programmer can take an available interface for a specific device in the target operating system and use Matlab Legacy Code Tool to systematically derive the associate S-Function and TLC files. The derived S-Function can be represented as a block in the Simulink model of the project. Now still in windows, code generation tools automatically extract the useful code out of the S-Function and appropriately connect the interface to the rest of the code generated for the model. Hence in this approach a general knowledge of S-Functions would be enough and there is no need for the developer to know TLC programming. This approach is not only useful for device divers but is also for constructing Simulink blocks for different Linux programming components like queues, Interrupt services.

As a case study, the proposed procedure is implemented on a mobile robot called IHA mobile Robot. The robot consists of three actuators and different sensors along with an embedded PC with Xenomai real-time Linux. The modules of the generated software is described and real-time performance of the code is analyzed. Different interfacing blocks was developed for this study. One block uses Linux realtime CAN library to produce CAN Open packets for controlling a set of Maxon EPOS2 Motor drivers. The other block reads and writes CAN Open packets from CAN bus connected to the embedded PC using Xenomai queue service. Furthermore, in those blocks, complex signals such as signals carrying an array of data packets between different blocks is implemented. This approach not only makes the model more succinct and readable but leads to more coherent connection of software modules generated by code generator. The detail instructions of developing such blocks is given as technical report accessible through the website of the project [12].

The contents of this paper are as follows. First a detailed description of the development environment and mechatronic system under the study is given. Then the procedure of generating real-time code for Xenomai Linux and its properties is discussed. Section III concerns with interfacing and development of device drivers block sets. Detail description is given on how to design the drivers and embed them in the code generation process. Next, implementation of the proposed method on the IHA mobile robot is presented and some results for validation of generated real-time software are demonstrated.

II. Architecture

The proposed architecture is divided into three distinct environments which is schematically shown in Figure 1. The *Development Environment* consists of tools that is used to develop the controller and generates the pertinent C code for the control software. The *Embedded Real-Time Target Environment* part consists of the actual mechatronic system which receives the generated code and compiles it into the executable real-time software. The *Supervisory System* is a part that sends runtime commands to the mechatronic system and generally receives monitoring data and information. Both the *Development Environment* and the *Supervisory System* can be on the same machine. The detailed description of each part is given in the following sections.

A. Remote Development Environment

The Development Environment consists of a normal PC with Matlab and Simulink installed on it. Here, it will be called Host PC. It is connected to the target PC using LAN. The model for the control software is developed on this system which may consist of one or more Simulink models. SSH connection is used for transferring data between Host PC and target PC. Hence, the target PC should be configured to accept SSH connections. Moreover, the *PuTTY*[13] SSH software should be installed on the Host PC. Matlab uses this software to establish a RSH(Remote Shell) connection to the target PC and transferring the generated code. Note that prior to Matlab 2011a the code generation packages were called *Real-Time Workshop* and *Embedded Real-Time Workshop*. Since Matlab 2011a these packages have been combined by some other related packages and now presented as three products called *Matlab Coder*, *Simulink Coder* and *Embedded Coder*. The reader is referred to Matlab documentation for further information. The terminologies used in this paper are based on new versions of Matlab.

Once the design evolved to an acceptable level of maturity, generated blocks for device drivers will be added to the model. Note that some of the blocks in Simulink are platform

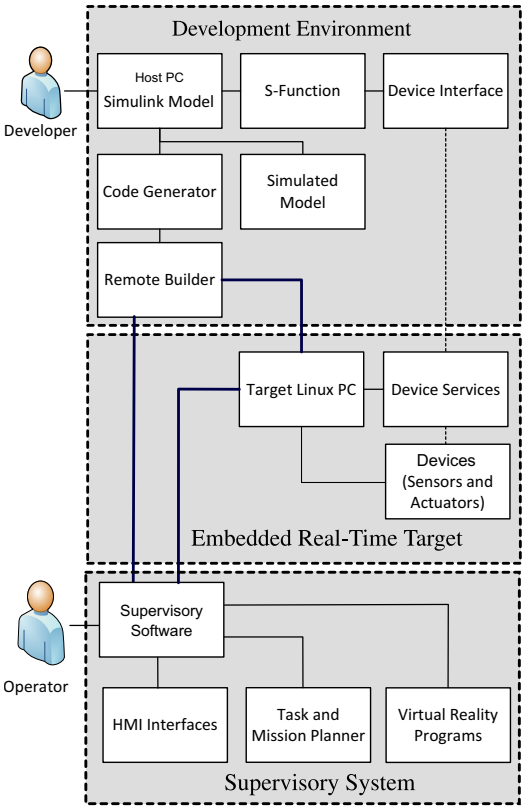


Fig. 1: Schematic Diagram of the Rapid Prototyping Framework

dependant. Although the development environment is Windows, we need to use Linux version of the blocks since the compilation is being done in Linux. For example, networking blocks such as UDP have both Windows and Linux versions however for a Linux target the Linux counterparts should be used in the model.

The Model needs some specific configurations to allow the code generator produce appropriate code for the Linux target. Once those configurations are made, *Embedded Coder* is used to generate C code from the whole Simulink model. Along with generating code, Simulink Coder is configured to generate a Makefile for the produced code with appropriate paths and flags. The Makefile is made based on set of configurable templates. However, it is made for the Host PC which means although libraries used in the generated code are for Linux, the paths and flags associated with them are related to Matlab installation folder in the Host PC. Hence, the Makefile is useless in the Target PC.

Nevertheless, a useful Matlab function called *remoteBuild* is used to solve this issue. *remoteBuild* receives as input an object called *buildInfo* which is a byproduct of the code generation process along with the Target PC information such as IP address and user's credentials. First, the function reads the produced Makefile from which it gathers all the headers and libraries based on their paths and adds them all into an archived folder along with the generated code. *remoteBuild* modifies the Makefile and produces another Makefile with appropriate paths and flags and adds it to the archive as well. Then, it establishes a RSH connection with the target PC and transfers the archive there. Finally, *make* command is used to compile and build the software executable on the Target PC. Note that this whole process is done automatically. Further notes on configurations that should be made in Simulink model along with using *remoteBuild* is given in Section III. Note that the executable that is made on the Target PC is fully independent of the Matlab, Host or any other part of development environment. Hence the development environment can be disconnected from the Embbeded real-time target.

B. Embedded Real-Time Target

Real-Time Target consists of an embedded PC with a real-time Linux installed. It is connected to hardware devices by means of different communication networks and protocols such as Serial, USB, CAN. It is assumed that device drivers and software libraries associated by those devices are properly installed on the Target PC. As mentioned before by executing *remoteBuild* function, the generated code is copied to a specified path in the real-time Linux along with the Makefile and other dependencies. *remoteBuild* uses GNU compiler to compile and build the code into an executable software.

C. Supervisory System

As mentioned before, Supervisory System is in fact the operator station and is used to send run-time commands and receive monitoring data from the Target PC. Physically, it can be either the Host PC itself or the Target PC or a separate system that is connected to the target, for example with an UDP connection. The Supervisory system may also contain HMI interfaces, Virtual Reality programs and Task Planners.

III. Real-Time Software

A. Simulink Model configuration

In order for Simulink and *Embedded Coder* to generate proper code, several configurations should be made in the Model. First, the *Target Performance* block should be added to the model. The choice for the *IDE/Tool Chain* should be set to *Eclipse* which basically means integration with Eclipse IDE[14]. Additionally, CPU properties of the Target PC are put in *Board Properties* section and Linux should be chosen as the operating system. Moreover, *Scheduling Mode* for Linux should be *real-time*.

Table I shows values for different *Configuration Parameters* of the Simulink Model. The first two rows are related to Solver pane, the next two rows are in *Code Generation* pane

TABLE I: Values for *Configuration Parameters*

Parameter	Value
Solver Type	Fixed-step
Solver	discrete
System target file	idelink_ert.tlc
Language	C
Build format	Makefile
Build action	Create_makefile
Configuration	Custom

and the the last three are in *IDE Link* section. Note that in all those configurations the values are set as if the target is the *Eclipse IDE*. Basically, it means that the generated code and makefile are compatible with the Eclipse IDE. Hence, The IDE can be used for further development of the code.

Along with mentioned set up, some other data should be provided for the Model as well. First, in order for the final software to run properly under hard real-time environment some extra C commands must be added to the generated code. Basically, these commands are not added by Matlab in the generated code. List 1 shows two examples of these commands. These commands should be added in *System Start* block that is part of custom code block sets. This implementation puts those commands in their proper position in initialization part of the generated code. The first line which is a memory management command is mandatory for a real-time process in Linux. It locks all pages mapped into the address space of the calling process. The second line is in *rtdk* real-time printing library. It should be used if the user intends to use real-time *printf* functions such as *rt_printf* in C-Mex SFunctions.

List 1: Custom Commands

```
1 mlockall(MCL_CURRENT|MCL_FUTURE);
2 rt_print_auto_init(1);
```

Next, the path for any pre-written C code along with the related header files should be added to the list of Include directories in *custom code* pane of *Configuration Parameters*. This includes C files written for implementation of device drivers which is explained in the next section. At last, in the *IDE Link* pane, there are two important parameters called *Complier options string* and *Linker options string*. Those parameters should be used to set the compiler and linker flags in the generated Makefile. These flags should be set based on the Linux operating system used in the Target PC. They provide useful information for the builder such as the optimization level for the compiler and library paths for code dependencies. Comprehensive explanations of these options can be found in GNU Make documentation. As an example, for our case study which is based on Xenomai Linux, The options shown in two lines of List 2 are some of the necessary options for the Compiler and Linker, respectively. These options contain a list of directives to substitute calls to Linux POSIX services with calls to Xenomai POSIX services [15].

List 2: Build Options

```
1 \$(shell xeno-config --posix-cflags)
2 \$(shell xeno-config --posix-ldflags)
```

B. remoteBuild *Function*

The *Embedded Coder* generates the code in a folder inside the working path. The name of the folder starts with the name of the model following by *_eclipseide* extension. Moreover, it produces a file called *buildInfo.mat* which contains information about the generated code, libraries used and many other parameters. *remoteBuild* uses the data provided in this file to gather the Matlab headers and libraries(in C languages) used in the code and manipulate the generated makefile. Hence, the makefile could be built in the Traget PC. Then it gathers the whole code along with the dependencies into an archived folder and sends them over to the target. There it extracts the archived folder and runs *make* command to build the code into an executable. At last, it runs the executable and logs the output.

List 3: remoteBuild Function

```
1 s=load('<Model Name>_eclipseide\buildInfo.mat');
2 remoteBuild(s.buildInfo,<Target Path>,<IP>,...
3  <user name>,<password>,<Putty Path>);
```

In List 3 the first line simply loads the build information and assign it to variable *s*. As for the second line, the inputs to *remoteBuild* are as follows. First is the *s.buildInfo* variable. Second is a path on the Target PC from which the files should be extracted. Third input is the IP address of the Target PC while the fourth and fifth inputs are the username and password valid on the remote system. The last input is the installation path to the PuTTY software. It uses the last four inputs to establish a RSH connection to the Target PC.

IV. Device Driver Block Sets

Generally speaking, API libraries provided by a device manufacturer or a service provider are used to develop interfacing modules that communicate with such systems. Once those modules are programmed, the main task is how to combine them with the generated code for the controller. One way is to import the generated code and interface functions into an external IDE such as Eclipse and manually program such connections. In this approach, the developer should be quite familiar with the generated code as well as those interface functions. Moreover, it demands considerable amount of coding. This could be quite troublesome especially in the rapid prototyping phase, where the model is going through many fundamental changes.

On the other hand, putting Matlab in charge of integrating those modules requires developers to program complicated S-Functions along with other necessities such as Target Language Compiler(TLC) files. A more suitable approach to tackle such complexities is using a Matlab tool called *Legacy Code Tool*. This section presents how this method could be easily utilized to develop general and reusable Simulink blocks out of those modules. The blocks and interfaces behind them are meaningless in the Matlab and Windows environment of the Host PC. However, the generated codes for those blocks are integrated gracefully with controller's and other parts of the model. With the presence of underlying API libraries in the target environment, the whole code is built without any difficulties.

A. *Developing Interfaces*

The requirements and properties associated with the interfaces are as follows. First, the device drivers or services should be installed properly on Linux Target along with their related API libraries and other pertinent dependencies. Second, appropriate flags that identify those libraries with the compiler and linker should be defined. Next step is to add those flags to *Complier options string* and *Linker options string* similar to ones given in List 2. As described before, doing so makes the code generator add those flags to the generated Makefile.

Next, the interface modules based on the API libraries should be developed in C\C++. In many cases, those are provided by the manufacturer as complete libraries, templates or examples. Basically, those interfaces can be categorized and mapped into four different functions. First, the *Start* Function gathers all procedures needed to initialize a connection with the device or the service. Acquiring needed memory, defining variables, checking device validity and other related tasks are in this category. Second is the *Initialize* Function which is similar to Start. However, Start is executed only once at the beginning of connection establishment. While, Initialize is used every time the connection, states or the device or the service is being restarted. Third, the *Main* or *Step* Function contains all the procedures to communicate with the device. Usually, this function is executed at each sample time of real-time software. It writes data and commands to the device and reads information from it. At last, *Termination* function is responsible for closing the connection, setting acquired memory free as well as ending other related tasks.

Besides those functions, an interface requires other sets of structures. The mentioned functions should be able to communicate with each other through different variables defined globally in the scope of the interface. Those variables contain socket features, connection states and memory addresses and other related information. In the area of S-Functions these variables are called *D-Works*. If the interfaces are to be integrated as S-Functions and Simulink blocks, the way those variables should be implemented is different to the traditional approach. Commonly, those are defined as class members or global variables and are accessible by aforementioned functions. However, for integrating into S-Functions, D-Works should always be defined as inputs to the functions. Hence, for being changeable inside the procedures, they should be passed by reference. In this case Matlab takes care of defining, memory handling and passing those variables to the appropriate functions in the generated S-Function and consequently in the generated code. In addition to D-Works, there are another set of variables called *Parameters*. Contrary to D-Works, they are constant and are used to parameterize and hence generalize the interfaces. Again, similar to D-Works, they should be defined as inputs to the interface functions. Once the S-Function and the related Simulink block were generated for the Interface, the parameters can be set and change by the developer before the code generation process. Hence the interface can be customized easily without changing the underlying code.

B. Generating S-Functions and Blocks

Once an interface is made compatible with the specifications given in the previous section, the Legacy Code Tool is used to generate the S-Function, the related TLC file, and the Simulink Block. Note that the code generator uses the TLC file to appropriately integrate and inline the interface code into the generated code and eliminate unnecessary S-Function routines and internally related Matlab functions. The detail instruction on utilizing the Legacy Code Tool is given in Matlab documentation. Here, useful key points are given for interface development in the Host PC while the generated code is to be used in a remote Linux target. More detailed instructions are also given in [12].

Consider an interface implemented in files called *interface.c* and *interface.h*. The files contain declarations and definitions for four aforementioned functions along with the inclusion of any relevant library. Depending on the type of function, it has none or all sets of inputs, outputs, D-Works and parameters. Programmatically, all those variables are defined as inputs. On the other hand, real outputs and D-Works should be passed by reference in order to be writable in the function domain. Moreover, the inputs, D-Works and parameters can be defined as variable size arrays of structures. Note that global variables should not be defined in *interface.c* or *interface.h* and should be always defined as D-Works. Multiple instances of a block interface in a model share any global variable defined in those files.

Once the related S-Function is generated by Legacy Code Tool, it has to be compiled and built into a Mex S-Function. This part is mandatory for having a Simulink block from the S-Function as well initiating the code generation process. Since this compilation occurs in the host PC, Matlab compiler does not have any reference to most of the libraries and methods used in the interface. In order to go around this issues one can put libraries and methods that depend on Linux between conditional compilation commands. In this case, host compiler only sees empty functions and generates an empty Mex file but the code generator, generates the code for the whole model and the whole interface files will be sent to the target. In Linux target, all the underlying libraries are present. Hence, the Linux compiler can easily build the software. An example of using such compilation commands for Xenomai Linux is shown in List 4.

List 4: Conditional Compilation Commands

```
#ifdef __XENO__
  #include<native/queue.h>
  #include<rtdk.h>
  // Linux Dependent Code
#endif

#ifdef MATLAB_MEX_FILE
  // Matlab Dependent Code
#endif
```

V. Case Study

Implementation of the proposed method was done on the *IHA mobile robot* (iMoro) as a mechatronic system. The robot's uncovered view along with hardware components is shown in Figure 2. It's a differential drive Mobile robot with an extendable boom carrying a camera system. It has three degree of freedom. Two of them are for the wheels and the third one extends the boom. Several computational systems have been tested as controllers. Due to lack of compatibility and flexibility, its embedded PC104 evolved into Janz® emPC-MN270 Xenomai Linux. The embedded PC has Intel Atom™ N270–1.6 GHz processor. A Hokuyo Rapid URG scanning laser rangefinder is connected to the embedded PC via USB connection. Moreover, three Maxon® Epos2 motor drivers are connected to the target PC over a CAN network and communicate based on CiA CANOpen protocol. Each driver drives a Maxon EC servo motor and transmits the position of an incremental encoder connected to the motor. The target PC connects to a wireless intranet of the department via Wi-Fi modem. It is accessible through the school network. Over this network, the embedded PC with Xenomai Linux is connected to a Windows host PC.

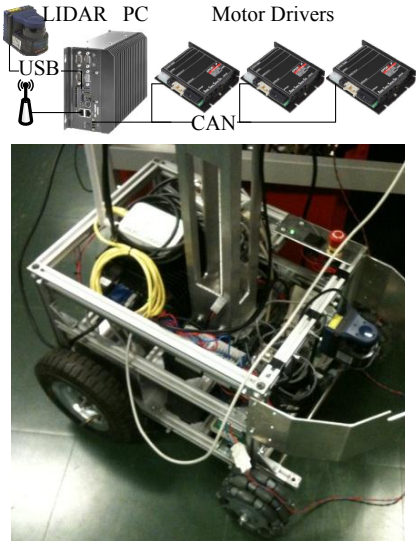


Fig. 2: The target system, its photo and related components

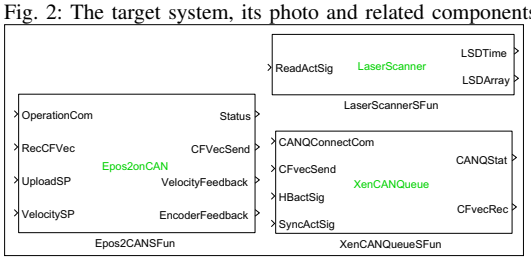


Fig. 3: Set of Constructed Blocks

The framework presented in this paper was used to develop a software controller for the robot. Several controller modules was developed in Simulink. The modules consist of a system for receiving teleoperation commands, obstacle detection using Laser Scanner, differential driving and etc. In order to interface with hardware components, the approach discussed in Section IV was implemented. Some of the developed interfacing blocks is shown in Figure 3. The *LaserScanner* block communicates with the Hokuyo range finder, when triggered by *ReadActSig* signals provides range data and a time stamp at its output ports. The *Epos2onCAN* block communicates with Epos2 motor drivers. It processes an array of CAN frames sent by the driver. The block also outputs array of CAN frames that contains control instructions for the driver. The other block that is called *XenCANQueue* that communicates with a Linux queue service that is responsible for exchanging data with the CAN network.

The software has two real-time periodic tasks. The periods for tasks are 0.1 and 0.01 seconds. The jitter for the real-time tasks are less than 100 micro seconds which is quite sufficient for the purpose of the project. Control engineers can now change their control algorithms and the parameters quickly and easily, and automatically generate the real-time software.

VI. Conclusion

In this paper, a simplified solution for rapid prototyping of control software in hard real-time Linux environment is given. The methodology discussed here enables developers to automatically implement their algorithms by using Matlab code generation products. It gives systematic approach to facilitate the development of interfaces connecting computational controllers with device drivers and Linux services. The proposed method was used to develop a real-time control software for a mobile robot. Independent of any Matlab related runtime library, the software successfully runs in real-time environment of Xenomai Linux. A set of general Simulink blocks was developed for the model. They can be used by other developers having Xenomai as their target operating system. The blocks are available for download and test. Taking advantage of this method leads to producing ready to use Simulink blocksets and model templates for other types of devices, services and Linux operating systems.

VII. Acknowledgments

This work has received funding by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network. It was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA grant agreement number 264336.

References

[1] D. Henriksson, A. Cervin, and K. Årzén, "Truetime: Real-time control system simulation with matlab/simulink," in *Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark*, 2003.
[2] A. Cervin, *Integrated Control and Real-Time Scheduling*. Department of Automatic Control, 2003.
[3] G. Quaranta and P. Mantegazza, "Using matlab-simulink rtw to build real time control applications in user space with rtailxrt," in *Realtime Linux Workshop. Milano*. Citeseer, 2001.
[4] P. J. Mosterman, S. Prabhu, A. Dowd, J. Glass, T. Erkkinen, J. Kluza, and R. Shenoy, "Embedded real-time control via matlab, simulink, and xpc target," in *Handbook of Networked and Embedded Control Systems*, ser. Control Engineering, D. Hristu-Varsakelis, W. S. Levine, and W. S. Levine, Eds. Birkhuser Boston, 2005, pp. 419–446.
[5] S. Abourida and J. Belanger, "Real-time platform for the control prototyping and simulation of power electronics and motor drives," in *Proceedings of the Third International Conference on Modeling, Simulation and Applied Optimization*, 2009, pp. 1–6.
[6] R. Bucher and S. Balemi, "Rapid controller prototyping with matlab/simulink and linux," *Control Engineering Practice*, vol. 14, no. 2, pp. 185 – 192, 2006.
[7] J. Cardoso, P. Diniz, M. Monteiro, J. Fernandes, and J. Saraiva, "A domain-specific aspect language for transforming matlab programs," in *Domain-Specific Aspect Language Workshop (DSAL'2010), part of AOSD*, 2010, pp. 15–19.
[8] C. Lpusan, V. Maties, R. Balan, and O. Hancu, "Rapid control prototyping in design process of mechatronic systems," *Solid State Phenomena*, vol. 166-167, pp. 247–252, Sep. 2010.
[9] R. Bucher and S. Balemi, "Rapid controller prototyping with Matlab/Simulink and linux," *Control Engineering Practice*, vol. 14, no. 2, pp. 185–192, 2006, compendex.
[10] W. Shen, Q. Hao, S. Wang, Y. Li, and H. Ghenniwa, "An agent-based service-oriented integration architecture for collaborative intelligent manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 23, no. 3, pp. 315–325, 2007.
[11] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum, *Building embedded Linux systems*. O'Reilly Media, 2008.
[12] (2012, May) Code generation project website. [Online]. Available: http://www.iha.tut.fi/puresafe/codegen.shtml
[13] Putty download page. [Online]. Available: http://www.putty.org/
[14] The eclipse foundation open source community website. [Online]. Available: http://www.eclipse.org/
[15] X. real-time platform users. (2012, January) Porting posix applications to xenomai. [Online]. Available: http://www.xenomai.org/index.php/Porting_POSIX_applications_to_Xenomai

Publication VI

Oftadeh, Reza, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback." In *6th IFAC Symposium on Mechatronic Systems*, pp. 663-669. IFAC, 2013.

Mechatronic Design of a Four Wheel Steering Mobile Robot with Fault-Tolerant Odometry Feedback

Reza Oftadeh [*,†] **Mohammad M. Aref** [*] **Reza Ghabcheloo** [*] **Jouni Mattila** [*]

[*] *Department of Intelligent Hydraulics and Automation (IHA), Tampere University of Technology, 33101, Finland.*
[†] *email: reza.oftadeh@tut.fi*

Abstract: In this paper, the mechatronics design of a four wheel steered mobile robot is discussed in detail. Mechanical structure and electrical interfaces are presented. Low-level software architecture based on embedded pc-based control is designed that enables the robot to operate its eight independent actuators synchronously. Kinematics models are elaborated, and it is shown that how mechanical structure of the robot affects kinematics and the feedback. Based on kinematics models, a fault tolerant wheel odometry is proposed to make the feedback robust to practical wheel odometry faults during the solution of forward kinematics. Real-time implementation of presented to support the the efficacy of proposed methods.

Keywords: Mobile robot, Four-wheel drive, Four-wheel steering, Autonomous vehicle, Fault tolerant feedback, Mechatronic design.

1. INTRODUCTION

A mobile robot is a complex mechatronics system which synergistically blends multidisciplinary solutions and methods from mechanics and embedded control system, see van Amerongen (2003). In this paper we describe design, implementation and kinematics of a four wheeled independently steering mobile robot from a mechatronics viewpoint. Wheeled mobile robots with different drive-steering mechanisms have been researched and engineered. Most common of all are differential drive robots. They have minimal actuated degrees of freedom and no moving parts for steering, see Chen et al. (2009).

Second most popular mechanisms, particularly for office environments, use omni-directional wheels like Mechanum or Swedish wheels. See for example Loh et al. (2003) and Salih et al. (2006) . Such mechanisms provide freedom of controlling heading independent of linear translation, thus increased maneuverability. They also use differential drive for steering. However, as mentioned by Diegel et al. (2002), large portion of the tangent force is lost by the rolling parts. Therefore, these types of robots can have high maneuverability but less efficiency which is an important factor for a battery powered mobile platform, see Song and Byun (2009).

For years differential type mechanism have been popular due to simplicity of their mechanics , however *four wheel steered* (4WS) mechanisms are receiving attention as a suitable platform for mobile manipulators. They enhance functionality of normal wheels for smooth motion, efficiency, and maneuverability for indoor applications, see Connette et al. (2010). Off-road application of 4WS robots have been reviewed by Berns et al. (2011). The drawback of 4WS design is its increased complexity in mechanics, kinematics, and control both in autonomous and manual modes. See for example Lam et al. (2009), and Parlitz et al. (2008). In many researches, in order to simplify motion control and synchronization of actuators, steering angles have been constrained mechanically by means of implementing steering mechanisms, such as in Santana et al. (2007); Vilaplana et al. (2005), or control strategies have been employed that keep the wheels parallel to each other, for instance see Selekwa and Nistler (2011). These solutions can solve conventional over-steering or under-steering problems of car-like robots but they decrease robot maneuverability. However, an efficient solution that enables the robot to maneuver in limited space require independent steering of each wheel, see Selekwa and Nistler (2011), and Connette et al. (2010).

Fig. 1. Mobile platform of *i*Moro.

Independent steering of each leg for a 4WS robot requires having eight drive and steering actuators synchronized. Here, we describe current state-of-the-art researches in the field of independently steered 4WS robots. One of the issues for the steered wheels is their power transmission for steering and driving shafts simultaneously, as addressed by Jacobs et al. (2012).

Many studies, such as Thuilot et al. (1996); Connette et al. (2010); Clavien et al. (2010), focus on studying main body motion properties in relation to its *instantaneous center of rotation* (ICR) and then mapping into wheels' coordinates to guarantee wheels synchronized movements, at least theoretically. However, these mappings are singular at certain configurations see Dietrich et al. (2011); Connette et al. (2009) for more details. Motion in neighborhood of these singular points disturbs kinematic formulations of the robot, thus velocity of certain wheels may fail to follow motions which are compatible with estimated ICR of the robot body, see Clavien et al. (2010). This will cause error in estimation of the ICR which instantly leads to two problems: slippage of the wheels by applying incorrect inverse kinematics, and also fault in the feedback by inaccurate forward kinematics solution and dead-reckoning. Faulty dead-reckoning can be propagated in each iteration, see Cooney et al. (2004). In this paper, instead of considering ICR, we map twist of body velocities directly to wheels' coordinate. The proposed mapping method is tolerant to single point failures and outliers in one leg.

This paper is organized as follows. Section 2 explains how to design modular independently steerable wheels, how to command its actuators by a real-time embedded controller via *control area network* (CAN), and the software components and interconnections among them. In Section 3, we derive the kinematic formulations for the robot and its velocity analysis. We provide a solution for synchronization of the actuators based on forward and inverse kinematic formulation of robot body movement. Moreover, the similar formulation to estimate robots pose from motor encoders is given in section 3.2. The solution detects faulty measurements to achieve a better pose estimate based on wheel odometry. We also address problems caused by slippage and inaccuracy in synchronization of the actuators, and provide a fault tolerant solution for robust mapping of the velocities, that is, the kinematics constraints. The solution provided in this paper paves the road for higher level control modules that navigate the robot around the obstacles and perform mobile manipulation. Finally, in Section 4 the experimental results from *i*Moro mobile platform are presented.

*i*Moro, shown in Figure 1, is a 4WS platform for mobile manipulation which is under investigation for high maneuverability and flexibility in high speed motions under the PURESAFE project. The PURESAFE project, *"Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation"*, was initiated to advance theoretical and experimental knowledge on semi-autonomous mobile manipulation within the ionizing radiation and contaminated environment of accelerators, in particular tunnels inside European Organization for Nuclear Research (CERN).

2. ROBOT ARCHITECTURE DESIGN

In this section, some of the main design requirements are listed. We have documented a comprehensive list of the required requirements in M. Aref et al. (2012). Based on those requirements the *iMoro* mobile manipulator is designed. The 3D model of the robot is illustrated in Figure 2. In this section, different components of the mobile platform are presented and their physical and logical architecture and designed interfaces are explained. Moreover, the implemented embedded controller that facilitates rapid-prototyping of the controller software is described. Finally, we explain different implemented control modules designed for performing tele-operation and autonomous tasks. A high level state-machine controller is devised that governs the execution order of the controller modules.

2.1 Design Requirements

As mentioned before, the main aim of PURESAFE project is to design a mobile manipulator that can be utilized in hazardous environments of research facilities to perform inspection and manipulation tasks. Here, the main focus is to design a mobile platform for such a robot. Below key requirements which are considered in the design of the mobile platform are listed,

- The mobile platform should be able to carry 30 kg of extra load.
- The mobile platform should be able to pass through normal doors.
- The mobile platform should have high maneuverability that enables it to rotate and change its heading in limited spaces.
- The whole system should be able to perform operations for at least several hours and travel two to three kilometers without recharging.
- The hardware and software of the whole system should be component-wise and modular that allow simultaneous development of different components.
- The software platform for the controller should support real-time capabilities while provides simple and straightforward means to develop and test different software modules.
- The designed control modules should provide both autonomous and tele-operated capabilities for the mobile platform.

In order to maximize the stability of the platform especially during manipulation tasks performing by the manipulator it is desired to keep the size of the platform as large as possible. However, the second and third requirements limit the size of the platform. The maximum dimensions that satisfy those requirements are considered for the overall size of the base. Table 1 lists specific parameters of the the mobile platform.

The third requirement presented in 2.1 requires the platform to perform pure rotations. More generally, in order to achieve high maneuverability the robot should have a decoupled heading and linear movement or in other words three degrees of freedom for the main body. As mentioned before, among different architectures of mobile robots that hold such characteristics, two types are widely popular. The first type are differential drive mobile robots with at least three omnidirectional wheels such as the

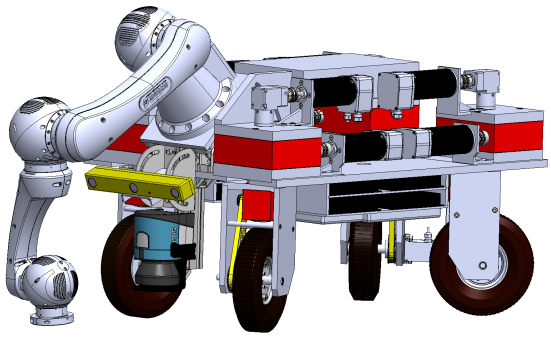
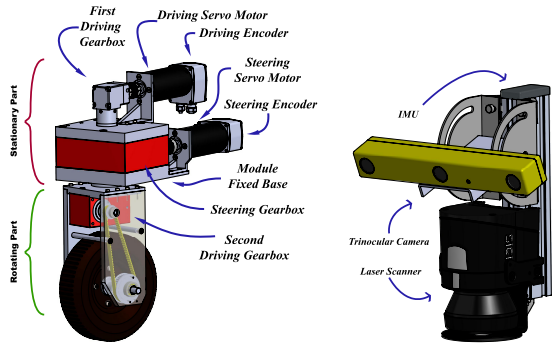


Fig. 2. 3D model of the mobile robot



(a) Wheel Module (b) Sensor Rack

Fig. 3. Designed Modules

one presented in Loh et al. (2003). The second types are pseudo-omnidirectional mobile robots with at least two wheels with independent steering and driving actuators. The efficiency of omnidirectional wheels is quite small hence as the general efficiency of the robot. Moreover, such robots can carry relatively smaller payloads. Based on the payload and efficiency requirements, we designed the mobile platform based on the second type.

2.2 General Design

The designed platform consists of four wheel modules (legs). Each legs has two degrees of freedom for independent drive and steer. Figure 3(a) portrays the designed leg. Based on the figure it consists of two parts: a moving part and a fixed part. The fix part contains the actuators and their directly connected gearboxes. The steering motor rotates the whole moving part along the vertical axis of the leg. Since there is no mechanical constrains, the moving part is able to rotate multiple turns. The first driving gearbox is connected to the second one with a shaft passing through the steering gearbox. The second driving gearbox then rotates a pulley-belt system connected to the wheel.

All the actuators are drived by a same type of motors (Brushless servo motor 400 Watt) . Based on the selected motors and other mechanical components the nominal efficiency of each MW at the speed of $1.7 m/sec$ is 70%. The motors are derived using Maxon EPOS2 drivers. The drivers support different control schemes such as position, velocity, and torque control based on CiA CANopen® specifications (LSB (2007)). The drivers for steering actuators are configured to work in position mode and return the position as feedback. On the other hand, the drivers for driving actuators receive and execute velocity commands while their desired feedback is configured to be the speed of the motor.

The main sensors that are used for localization and navigation (except actuators' encoders) are assembled on an external body called *sensor rack*. It contains a laser range finder, a stereo camera and an Inertial Measurement Unit(IMU). Figure 3(b) illustrates the designed sensor rack. As shown in the figure, field of view of the camera and vertical position of the laser scanner can be adjusted at will during development phase. The main advantage of such design is that it makes the whole system an individual module for localization and navigation that can be attached or detached from the base easily. Such modular design facilitates also the calibration of the sensors with respect to each other. Moreover, during the development it allows the developers to work with the module separate from the mobile platform itself while the mobile robot can undergo independent developments before the integration phase.

2.3 Communication Interfaces

As it is described in the next subsection, a fanless embedded PC is used as the controller hardware. Sensors and actuator drivers are connected to the controller via various communication interfaces. The schematics are shown in Figure 4.

The main communication interface is a CANopen network that connects IMU and all the motor drivers (including the encoders) to the embedded PC. The communication speed of the network is 1 Mbps. The CANopen *Heartbeat* protocol is implemented on the network to continuously monitor the status of the devices. All the devices expect to receive a *Heartbeat* message from the embedded PC every 50ms and send *Heartbeat* messages to the embedded PC every 100ms. The controller sends desired set points using Process Data Objects(PDO) to the drivers every 10ms. In order to synchronize the actuators, the CANopen *Sync* protocol is implemented.

2.4 Embedded Controller

The robot is equipped with an embedded pc acting as a central embedded controller for the whole mobile platform. Since the robot is required to operate in contaminated areas, a fanless embedded pc is selected with passive cooling. The quad-core CPU of the embedded PC allows

Table 1. General specifications of the robot

Description	Quantity
Main Body Length	655 mm
Main Body Width	335 mm
Max. Velocity of Leg's Steering	0.74 $^{rad}/_{sec}$
Max. Velocity of the Platform(30kg load)	2.1 $^{m}/_{sec}$
Wheel Diameter	200 mm
Approximate Mass of Each Leg	20 kg
Approximate Overall Mass	120 kg

for running multiple software modules such as vision and motion control in parallel. Hence, it provides a powerful computational base for executing complex algorithms. For some communications, it is mandatory to guarantee a prompt respond to the received data from a sensor or send the commands at exact time intervals. Hence, the software platform should provide hard real-time capabilities for some interfacing software modules. We use Xenomai real-time Linux as the *Operating System*(OS) for the embedded PC in order to provide such real-time capabilities. However, it is impossible to have all the software modules executed in the hard real-time environment. Specifically, due to the non-realtime nature of some of the functions in the software. This is the case especially for some interfacing modules such as *User Datagram Protocol* (UDP) packets that are not generally supported in real-time environment.

Here, a multi-thread programming approach is used. Different software modules of the controller are executed in different executing threads. Hence, depend on the module, they can be run in the hard real-time scheduler of the OS or the non real-time scheduler. Real-time programming *queues* are used to communicate between different software modules.

The communication interface for CANopen network is implemented using Xenomai *RTDM* library to be executed in hard real-time environment. This is also the case for gathering data from the laser range finder through LAN using *RTnet* real-time library. However, capturing images from the stereo camera or UDP communication through the wireless network is executed in a non real-time environment. However, implementation of such strategy could be quite tedious and time consuming. Especially in the development phase where the software modules and algorithms are subjected to lots of changes and improvements. In order to facilitate rapid-prototyping of different algorithms and software modules especially in real-time environment, a unified framework which is proposed in Oftadeh et al. (2012) is implemented. The framework allows the developers to develop their control software in Simulink® software and use Matlab® code generation products to automatically generate and build real-time code for their control modules.

2.5 Controller Architecture

Figure 5 illustrates the schematic diagram of different control modules. Two low level motion controllers are implemented for the mobile platform. The manual motion controller is designed based on Ackerman steering. The two steering actuators of the rear wheel modules are fixed and front steering actuators rotates to comply with Ackerman steering principal.

For the autonomous module a controller strategy by Oftadeh et al. (2013) is used that enables the robot to follow a desired path and a given heading profiles independently. The path is generated based on the planning method presented by Agha-mohammadi et al. (2011). Moreover, the *Simultaneous Localization and Mapping* (SLAM) method provides environment map for planning and location of the robot by means of a laser range finder (Tamjidi et al., 2009).

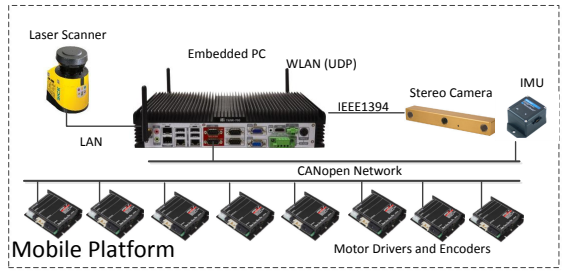


Fig. 4. Communication Interfaces for different components

In the architecture shown in Figure 5, different motion control modules either manual or automatic receive the configuration of the main body (pose and velocity) as inputs and generates the desire linear and angular velocities of the body. Then the inverse kinematic relation is used to map those velocities to the speed and position angle of the driving and steering actuators of the legs, respectively. Those values are then sent to the drivers as desired set points.

One of the main information used in localization algorithms is dead-reckoning using wheels speeds and angles. Here, the feedback gathered from motor encoders is mapped to the linear and angular velocity of the body and, in the localization module, it is combined by the data gathered by other sensors in order to provide the accurate estimation of the body position and velocity. We show, in the next section, how utilizing the redundancy in odometry data can be utilized to derive more accurate and fault-tolerant estimation of the linear and angular velocity of the body.

3. KINEMATIC MODEL

In this section, we present kinematics formulation in velocity space for the 4WS mobile platform . Although the equation is quite straightforward, it embeds substantial information pertinent to the relative velocities of the body and wheel modules. A general solution is then given for estimating body linear and angular velocity using kinematic constraints and encoders data. The solution uses a least square method to measure relative consistency of the legs' velocities,thus detecting faulty encoder data. The mobile robot can be fully functional even after loosing the functionality of one of its wheel modules. Hence, this approach can be used to single out a faulty leg during the robots operation.

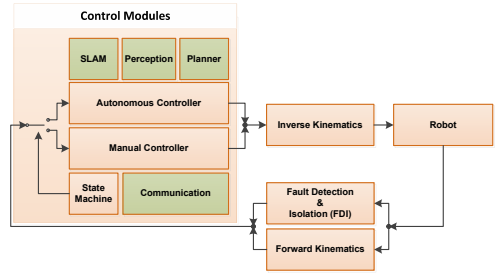


Fig. 5. Control Topology

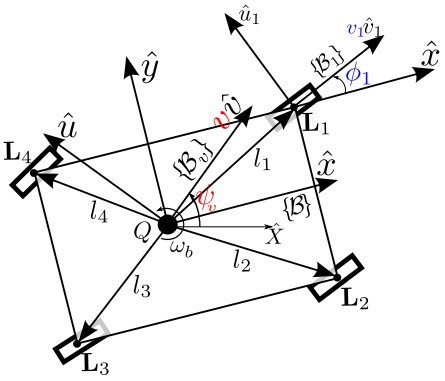


Fig. 6. Denoted kinematic parameters

3.1 Inverse Kinematics

Figure 6, shows a schematic view of a 4WS mobile robot. The denoted parameters for the configuration of the body and the leg modules is presented in the figure. The coordinate frame $\boldsymbol{U}\{\hat{\boldsymbol{X}}, \hat{\boldsymbol{Y}}\}$ is the inertial frame. Frame $\boldsymbol{B}\{\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}\}$ is fixed-body frame defining the heading of the robot, and $\boldsymbol{B}_v\{\hat{\boldsymbol{v}}, \hat{\boldsymbol{u}}\}$ is the velocity frame, that is, unit vector $\hat{\boldsymbol{v}}$ determines the direction of the robot's base linear velocity vector. Both $\boldsymbol{B}$ and $\boldsymbol{B}_v$ are attached to the robot's base at point $\boldsymbol{Q}$ which can be chosen at will. Angles ψ_v and θ_B are the angles $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{x}}$, respectively, in $\boldsymbol{U}$. Scalar value v is the magnitude of linear velocity of point Q. Variables $\omega_B = \dot{\theta}_B$ and $\omega_v = \dot{\psi}_v$ are the angular velocity of the base and that of $\hat{\boldsymbol{v}}$, respectively. Constant vectors $\overrightarrow{\boldsymbol{QL}_i}$ presented in frame $\boldsymbol{B}$ are denoted by ${}^{\boldsymbol{B}}\boldsymbol{\ell}_i, i = 1..4$. Angle ϕ_i is the heading angle of the i^{th} leg while $v_i\hat{\boldsymbol{v}}_i$ is the velocity vector of the attachment point $\boldsymbol{L}_i$. Steering and speed control commands for leg i are calculated using ϕ_i and v_i, respectively. The following kinematic relations hold

$$ {}^{\boldsymbol{B}}\hat{\boldsymbol{v}} = \boldsymbol{R}(\psi_v - \theta_B)[1\ 0\ 0]^T \tag{1a} $$

$$ v_i\ {}^{\boldsymbol{B}}\hat{\boldsymbol{v}}_i = v\ {}^{\boldsymbol{B}}\hat{\boldsymbol{v}} + \omega_B(\hat{z} \times {}^{\boldsymbol{B}}\boldsymbol{\ell}_i) \tag{1b} $$

in which, $\hat{z} = [0\ 0\ 1]^T$ and $\boldsymbol{R}(\psi_v - \theta_B)$ is the rotation matrix with angle $\psi_v - \theta_B$ around z-axis, that is, frame $\boldsymbol{B}_v$ in $\boldsymbol{B}$. For motion control purposes, once the control signals ψ_v, ω_B and v are calculated for the body frame, desired signals ϕ_i and v_i can be derived using above kinematics equaitons. Notice that only angle ϕ_i and not its derivatives appear in (1a-1b), as angle of vector ${}^{\boldsymbol{B}}\hat{\boldsymbol{v}}_i$. This fact justifies having steering actuators to be controlled in position mode.

3.2 Forward Kinematics

As mentioned before, speed of the drive actuators and positions of the steering actuators are measured , that is, positions ϕ_i for steering actuators and speeds v_i for driving actuators . In this subsection, the goal is to derive ψ_v, ω_B and v based on theencoders measurement data, that is,sets $\{v_i, \phi_i\}$ for $i = 1..4$. Clearly, the relative velocity of the wheel modules with respect to each other must be zero. In other words, the set $\{v_i, \phi_i\}$ applied to a leg should be consistent with other sets. Although the set points for steering and drive commands are derived based on the equation (1b) and hence consistent, the actual value of the actuators encoders will not comply exactly the kinematics relations due to different loads and different servo dynamics. Since the legs are connected to a rigid base this inconsistency makes the legs to slip with respect to each other. This slippage is different than the overall slippage of the robot that can not be observed through odometry data. In order to measure the consistency of the actuators actual values, we define the following measure:

$$ e_i = \frac{1}{4}\sqrt{\sum_{j=1, j\neq i}^{4} e_{ij}^2} \quad , i \in \{1, 2, 3, 4\} \tag{2} $$

where,

$$ e_{ij} = (v_i\ {}^{\boldsymbol{B}}\hat{\boldsymbol{v}}_i - v_j\ {}^{\boldsymbol{B}}\hat{\boldsymbol{v}}_j).\boldsymbol{\ell}_{ij} \tag{3a} $$

$$ \boldsymbol{\ell}_{ij} = \frac{{}^{\boldsymbol{B}}\boldsymbol{\ell}_i - {}^{\boldsymbol{B}}\boldsymbol{\ell}_j}{||{}^{\boldsymbol{B}}\boldsymbol{\ell}_i - {}^{\boldsymbol{B}}\boldsymbol{\ell}_j||_2} \tag{3b} $$

In above equations e_{ij} is the relative velocity line L_i- to L_j. Clearly, if both wheels are synchronized perfectly and no slippage occur e_{ij} is zero. However, as explained before practically this variables are not zero and value e_{ij} indicates relative slippage of the wheel modules i and j. The variable e_i is the measure of relative slippage of one wheel with respect to other three wheels. Practically for the mobile robot to move properly, all e_is must be small. The value of e_i determine the overall consistency of wheel i with respect to the other modules. Several cases can be considered for different values of e_i,

- In the case that,
$$ \min(e_i) > e_{max} \tag{4} $$
it means that legs are quite faulty and they are not consistent with each other at all. Observing such condition for an pre-specified interval of time should make the controller to issue a fault signal to stop the robot.
- If e_i is large only for one wheel, it can be interpreted that the wheel is not consistent with other wheels and it is being pushed by the others. In this case different strategies can be considered such as stopping the robot or singling out the faulty leg from the control loop and set the actuators of the faulty wheel to rotate freely.
- Otherwise, more complex strategies can be designed based on the robot's assigned task and application.

After calculating e_i for all legs and evaluating their values, the forward kinematic problem can be solved as follow. Let ${}^{\boldsymbol{B}}\hat{\boldsymbol{v}}$ and ${}^{\boldsymbol{B}}\hat{\boldsymbol{v}}_i$ be denoted by their components by $[v_x\ v_y]^T$ and $[v_{ix},\ v_{iy}]^T$. Then (1b) for all the legs can be rearranged into the following equation,

$$ \boldsymbol{A}\begin{bmatrix} vv_x \\ vv_y \\ \omega_B \end{bmatrix}_{3\times 1} = \boldsymbol{B} \tag{5} $$

in which,

$$ \boldsymbol{A}^T = \begin{bmatrix} 1 & 0 & \dots & 1 & 0 \\ 0 & 1 & \dots & 0 & 1 \\ -l_{1y} & l_{1x} & \dots & -l_{4y} & l_{4x} \end{bmatrix}_{3\times 8} \tag{6a} $$

$$ \boldsymbol{B}^T = [\, v_{1x}\ v_{1y}\ \dots\ v_{4x}\ v_{4y}\,]_{1\times 8} \tag{6b} $$

Obviously, in general case the matrix equation (5) consists of eight equations and three unknowns. A simple least square solution is given by

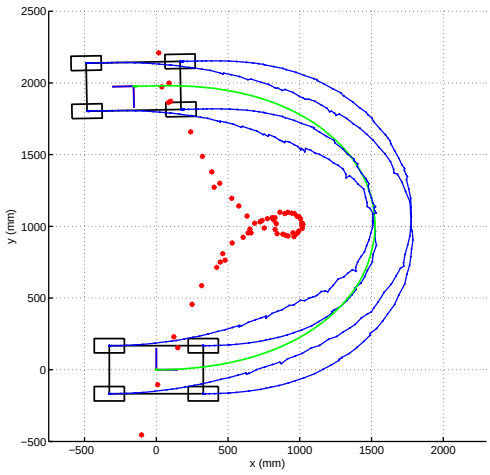


Fig. 7. Experimental Results: Robot's trajectories, 180° rotation

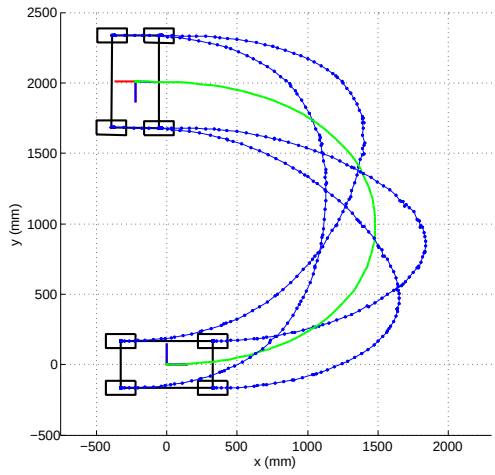


Fig. 8. Experimental Results: Robot's trajectories, -90° rotation

$$\begin{bmatrix} vv_x \\ vv_y \\ \omega_B \end{bmatrix}_{3\times 1} = \boldsymbol{A}^{\dagger}\boldsymbol{B} \tag{7}$$

Note that the elements of $\boldsymbol{A}$ are only function of the constant vectors ${}^{\boldsymbol{B}}\boldsymbol{\ell}_i$. Hence, $\boldsymbol{A}^{\dagger} = (\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}^T$ is needed to be calculated once. In practice, we need at least three rows of matrices A and B to solve the forward kinematics. Thus, a way to improve the solution is to eliminate the inconsistent wheel module and eliminate the corresponding rows from (5). Our strategy is based on isolating the leg with maximum of e_i. Then, the rows with respect to that leg is eliminated from matrix A in 6a and the least square is calculated based on this new matrix. Using experimental data, we show next the efficacy of the above strategy.

4. EXPERIMENTAL RESULTS

In this section the experimental results of implementing the autonomous controller described in Section 2.5 are shown. The localization is obtained by dead-reckoning of the odometry data as described in previous section. The *i*Moro mobile platform shown in Figure 1 is used to run the experiments.

The desired path that is followed by the mobile platform is a cubic *Bezier* curve with its control points located at $\{(0,0),(2m,0),(2m,2m),(0,2m)\}$. The path is used with two desired heading profiles. Figures 7 shows the body and legs trajectories for following the path with 180° rotation of the main body. While, Figure 8 shows the same trajectories, this time for $-90°$ rotation of the body. Figures 7 also depicts the estimated location of the body's ICR during its movement as red dots.

The forward kinematic solution is solved with and without eliminating the leg with maximum e_i. Figure 9 depicts the pseudo-inverse error before and after the elimination. From the figure, its clear that the error is reduced considerably by eliminating the leg with maximum e_i from the forward kinematic solution.

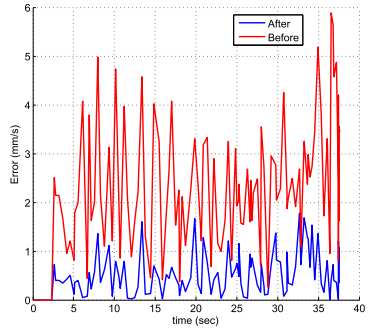


Fig. 9. Experimental Results: Least Square error before and after elimination of the outlier leg

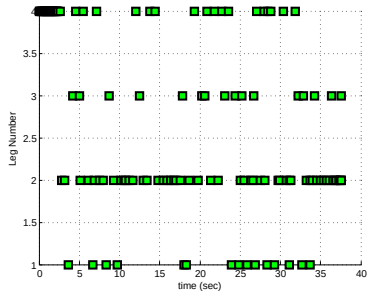


Fig. 10. Experimental Results: Leg number for the most inconsistent leg

Figure 10 shows the leg number with biggest e_i at each sample time. From the figure it is clear that at each period, a different leg has the maximum e_i. This is due to the change in the dynamic loads acting on each leg during the movement. Having one specific leg with maximum e_i all the time could indicate a defect in that leg.

5. CONCLUSION

Firstly, we described mechatronics design of a mobile platform to fulfill the requirements of a robot for scientific facilities. We detailed mechanical design and electronics. We also showed that synchronization of eight actuators is essential and provided software control architecture to make it possible. We derived and explored the kinematics relations which are used both in calculation of motion control signals and in fault tolerant velocity estimates of the main body. Performance of the proposed system is verified by experiments on IHA Mobile Robot (*i*Moro).

ACKNOWLEDGEMENTS

This work, supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network, was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA grant agreement number 264336.

REFERENCES

Agha-mohammadi, A., Chakravorty, S., and Amato, N. (2011). FIRM: Feedback controller-based Information-state RoadMap -a framework for motion planning under uncertainty-. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4284–4291.

Berns, K., Kuhnert, K., and Armbrust, C. (2011). Off-road robotics-an overview. *KI-Kunstliche Intelligenz*, 25(2), 109–116.

Chen, C., Li, T., Yeh, Y., and Chang, C. (2009). Design and implementation of an adaptive sliding-mode dynamic controller for wheeled mobile robots. *Mechatronics*, 19(2), 156–166.

Clavien, L., Lauria, M., and Michaud, F. (2010). Instantaneous centre of rotation estimation of an omnidirectional mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA), USA.*, 5435–5440.

Connette, C., Parlitz, C., Hägele, M., and Verl, A. (2009). Singularity avoidance for over-actuated, pseudo-omnidirectional, wheeled mobile robots. In *IEEE International Conference on Robotics and Automation, (ICRA)*, 4124–4130.

Connette, C., Pott, A., Hägele, M., and Verl, A. (2010). Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4775–4781.

Cooney, J., Xu, W., and Bright, G. (2004). Visual dead-reckoning for motion control of a mecanum-wheeled mobile robot. *Mechatronics*, 14(6), 623–637.

Diegel, O., Badve, A., Bright, G., Potgieter, J., and Tlale, S. (2002). Improved mecanum wheel design for omnidirectional robots. In *Proc. 2002 Australasian Conference on Robotics and Automation, Auckland*, 117–121.

Dietrich, A., Wimbock, T., Albu-Schaffer, A., and Hirzinger, G. (2011). Singularity avoidance for nonholonomic, omnidirectional wheeled mobile platforms with variable footprint. In *IEEE International Conference on Robotics and Automation (ICRA)*, 6136–6142.

Jacobs, T., Connette, C., Haegele, M., and Verl, A. (2012). Design of wheel modules for non-holonomic, omnidirectional mobile robots in context of the emerging control problems. In *Proceedings of 7th German Conference on Robotics (ROBOTIK)*, 1–4. VDE.

Lam, T., Qian, H., Xu, Y., and Xu, G. (2009). Omnidirectional steer-by-wire interface for four wheel independent steering vehicle. In *IEEE International Conference on Robotics and Automation, ICRA*, 1383–1388.

Loh, W., Low, K., and Leow, Y. (2003). Mechatronics design and kinematic modelling of a singularityless omnidirectional wheeled mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, 3237–3242.

LSB, M. (2007). 6 general object definitions. *IEC 61800-7-200: Adjustable speed electrical power drives systems–Part 7-200: Generic interface and use of profiles for power drive systems–Profile specifications*, 3577.

M. Aref, M., Oftadeh, R., Koivumäki, J., Ghabcheloo, R., and Mattila, J. (2012). System requirement document for modular mobile manipulator system. *PURESAFE System Engineering Documents*, WP3(RP10-RP14).

Oftadeh, R., M. Aref, M., Ghabcheloo, R., and Mattila, J. (2012). Unified framework for rapid prototyping of Linux based real-time controllers with Matlab and Simulink. In *IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM), Taiwan*, 274 –279.

Oftadeh, R., M. Aref, M., Ghabcheloo, R., and Mattila, J. (2013). Bounded-velocity motion control of four wheel steered mobile robots. *Submitted for IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM).*

Parlitz, C., Hägele, M., Klein, P., Seifert, J., and Dautenhahn, K. (2008). Care-o-bot 3-rationale for human-robot interaction design. In *Proceedings of 39th International Symposium on Robotics (ISR), Seul, Korea*, 275–280.

Salih, J., Rizon, M., Yaacob, S., Adom, A., and Mamat, M. (2006). Designing omni-directional mobile robot with mecanum wheel. *American Journal of Applied Sciences*, 3(5), 1831–1835.

Santana, P., Barata, J., and Correia, L. (2007). Sustainable robots for humanitarian demining. *International Journal of Advanced Robotic Systems*, 4(2), 207–218.

Selekwa, M. and Nistler, J. (2011). Path tracking control of four wheel independently steered ground robotic vehicles. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, 6355–6360.

Song, J. and Byun, K. (2009). Steering control algorithm for efficient drive of a mobile robot with steerable omnidirectional wheels. *Journal of mechanical science and technology*, 23(10), 2747–2756.

Tamjidi, A., Taghirad, H., and Aghamohammadi, A. (2009). On the consistency of EKF-SLAM: Focusing on the observation models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2083–2088.

Thuilot, B., d'Aandrea Novel, B., and Micaelli, A. (1996). Modeling and feedback control of mobile robots equipped with several steering wheels. *IEEE Transactions on Robotics and Automation*, 12(3), 375–390.

van Amerongen, J. (2003). Mechatronic design. *Mechatronics*, 13(10), 1045–1066.

Vilaplana, M., Mason, O., Leith, D., and Leithead, W. (2005). Control of yaw rate and sideslip in 4-wheel steering cars with actuator constraints. *Switching and Learning in Feedback Systems*, 3355, 201–222.

Publication VII

Oftadeh, Reza, Mohammad M. Aref, Reza Ghabcheloo, and Jouni Mattila. "System integration for real-time mobile manipulation." *International Journal of Advanced Robotic Systems.*

INTECH
open science | open minds

OPEN ACCESS
ARTICLE

International Journal of Advanced Robotic Systems

System Integration for Real-Time Mobile Manipulation

Regular paper

Reza Oftadeh[1,*], Mohammad M. Aref[1], Reza Ghabcheloo[1] and Jouni Mattila[1]

[1]Intelligent Hydraulics and Automation Department(IHA), Tampere University of Technology, Tampere, Finland.
* Corresponding author E-mail: oftadeh.reza@ieee.org

Originally published in Volume 1, Telerobotics and Systems Engineering for Scientific Facilities, Year 2013

Abstract Mobile manipulators are one of the most complicated types of mechatronics systems. The performance of these robots in performing complex manipulation tasks is highly correlated with the synchronization and integration of their low-level components. This paper discusses in detail the mechatronics design of a four wheel steered mobile manipulator. It presents the manipulator's mechanical structure and electrical interfaces, designs low-level software architecture based on embedded PC-based controls, and proposes a systematic solution based on code generation products of MATLAB and Simulink. The remote development environment described here is used to develop real-time controller software and modules for the mobile manipulator under a POSIX-compliant, real-time Linux operating system. Our approach enables developers to reliably design controller modules that meet the hard real-time constraints of the entire low-level system architecture. Moreover, it provides a systematic framework for the development and integration of hardware devices with various communication mediums and protocols, which facilitates the development and integration process of the software controller.

Keywords Mobile manipulators, Real-time systems, Autonomous vehicles, Mechatronic design.

1. Introduction

A mobile manipulator refers to a robotic manipulator arm mounted on a mobile base. Mobility provides the manipulator with an unlimited workspace at the expense of added challenges[1, 2]. *i*Moro, shown in Figure 1, is a four wheel independently steered (4WS) mobile manipulator that is under investigation for high maneuverability and flexibility in high-speed motions under the PURESAFE project. The PURESAFE project, *"Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation"*, was initiated to advance theoretical and experimental knowledge on semi-autonomous mobile manipulation within the ionizing radiation-filled and contaminated environments of accelerators, particularly within tunnels inside the European Organization for Nuclear Research (CERN).

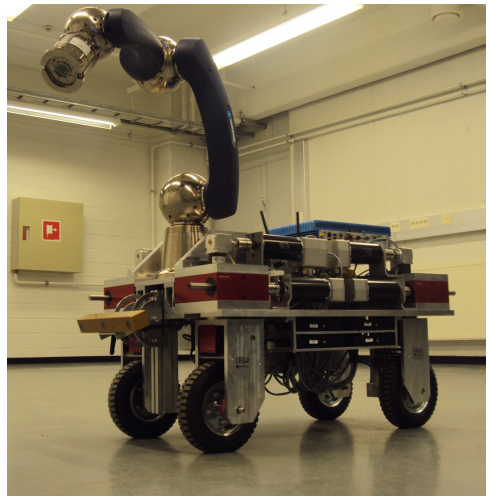


Figure 1. Mobile platform of *i*Moro.

A mobile manipulator is a complex mechatronics system that synergistically blends multidisciplinary solutions and methods from mechanics and embedded control systems [3, 4]. There are different issues associated with the development of such mechatronic systems, especially in the low level architectures involving device drivers and system controllers. One example of such an issue is that of interfacing with hardware devices, including different communication mediums and protocols. Moreover, developing mechatronic systems requires numerous and consecutive practical tests in order to verify and validate the functionality and integrity of the developed algorithms and codes. Ultimately, it should lead to an optimized implementation of controllers that can punctually receive sensor data and produce control signals over guaranteed intervals of time.

This paper is the synergistic conclusion of the authors' research on this topic, presented in [5, 6]. It demonstrates how the framework proposed in [5] for rapid prototyping of real-time controllers successfully provides a software development basis for the *i*Moro mobile manipulator, the design of which is presented in [6]. In the following, the authors present the background study for the hardware and software architecture and introduce the integration approach. Section 2 details the hardware design of the mobile manipulator and discusses the selected communication interfaces between the components. Section 3 explains the framework for the real-time software integration. The final section presents the experimental results of successful implementation of the method.

1.1. Hardware Architecture

The main challenge in designing a mobile manipulator is the selection of its mobile base and wheel configurations. Wheeled mobile robots with different drive-steering mechanisms have been researched and engineered [7]. The most common variation is that of the differential drive robot. These have minimal actuated degrees of freedom and no moving parts for steering [8]. The second most popular mechanisms, particularly for office environments, use omni-directional wheels such as the Mechanum or Swedish wheels [9, 10]. Such mechanisms provide freedom to control robot heading independent of linear translation, thus increasing maneuverability. They also use a differential drive for steering. However, as mentioned by [11], a large portion of the tangential force is lost by the rolling parts. Therefore, these types of robots often have high maneuverability but lower efficiency, which is an important factor in a battery-powered mobile platform [12].

For years, differential-type mechanisms have been popular due to the simplicity of their mechanics; however, 4WS mechanisms have recently received increasing attention as suitable platforms for mobile manipulators. 4WS mechanisms enhance the functionality of normal wheels for smooth motion, efficiency, and maneuverability for indoor applications [13]. Off-road applications of 4WS robots have been reviewed by [14]. The drawback of the 4WS design is the increased complexity of its mechanics, kinematics, and control in both autonomous and manual modes [15, 16].

Many studies have attempted to simplify motion control and synchronization of actuators by mechanically constraining steering angles (by implementing steering mechanisms, such as in [17, 18]) or by employing control strategies that keep the wheels parallel to one another (as in [19]). These solutions can solve conventional over-steering or under-steering problems of car-like robots, but they also decrease robot maneuverability. An efficient solution that enables the robot to maneuver in limited space requires independent steering of each wheel [13, 19].

1.2. Software Architecture

Another challenge in the process of designing a mobile manipulator is the selection of a suitable software development framework for the robot. While software engineers prefer direct C\C++ programming, control engineers and system developers prefer MATLAB and Simulink or similar software or toolboxes because of their ready-to-use advanced mathematical functions and algorithms. Additionally, in most cases, the simulation of the control algorithms is done in MATLAB/Simulink, and re-implementation in C/C++ is both time consuming and a potential source of errors. However, the high-level programming of this approach makes the models developed in MATLAB/Simulink unsuitable for real-time tasks.

On one hand, during the design procedures of a complex mechatronic system, several teams of researchers are working simultaneously on the design processes and modeling of both hardware and software platforms. Thus, even conceptual design and development of product hardware can be subject to change at the same time that software is being designed. On the other hand, validation of the developed design models, methods, and tools occurs through prototype experiments. The desirable outcome benefits the blocks, modules, and patterns of the implemented components and written software, which can be reused. Therefore, a real-time control rapid prototyping environment is necessary to overcome the control issues of a mechatronic system and simultaneously consider other criteria, such as modularity, flexibility, reusability, compatibility with commercial off-the-shelf components, and creating a dependable basis for future advances in safety measures (such as the *Safety Integrity Level* (SIL)).

Several studies have examined the real-time implementation of algorithms in MATLAB [20, 21] and its integration with Linux [22]. Moreover, there are available software package applications that utilize selected functionalities of MATLAB in their real-time execution environments, such as the xPC Targeting toolbox [23], RT-Lab [24], and RTAI-Lab [25]. Mathworks has addressed this problem by embedding code generator products that automatically generate optimized, independent, and royalty-free C\C++ code from MATLAB codes and Simulink models. Comparing generated C\C++ codes with their original MATLAB codes is the subject of research in [26].

Generally speaking, one of the challenges in a mechatronic system is making interfaces that are operational between real-world sensors and actuators and the software that develops computational controllers like MATLAB. This challenge has been solved in dSpace® products by means of the complementary hardware for rapid prototyping and code generation in its package, which has been used in several research studies [27]. The RTAI Linux team [28] has also produced a comprehensive work that integrates RTAI-Lab into Simulink and translates the code into the RTAI framework by means of Real-Time Workshop (RTW). These systems are popular for a certain range of control and software requirements. However, compared to traditional C\C++ software development, their solutions enforce additional limitations on the reusability of existing codes. This problem is negligible until research and development teams use products that are only compatible with their development environments. In the event of any unforeseen instrumental necessities, adaptation of a third-party driver to those environments is full of difficulties. On the other hand, a normal C\C++ application can easily use manufacturer-provided drivers for C\C++.

1.3. Integration

This paper discusses a software development environment for the rapid prototyping of the afore-mentioned mechatronic systems. Its aim is to introduce a coherent framework for remote development of independent hard, real-time software controllers based on MATLAB and Simulink products for the rapid prototyping of PC-based mechatronic systems. The authors provide the technical details of the method in [5]. This paper shows how the method is successfully applied to the *i*Moro. The proposed method consists of developing software modules and controllers in a Simulink project on a PC with a Windows operating system, with a target of one or several embedded PCs running real-time Linux (the "Target PC"). Then, the Code Generators toolbox is used to derive a stand-alone and hard real-time C\C++ code for the real-time Linux operating system. The resultant code is automatically transferred to the embedded target, where it is compiled and built. The final software is ready to debug, run, or verify on the Target PC.

Note that this methodology could easily be expanded to develop multiple software programs to run on the Target PC at once. For very complex mechatronic systems, in which software and controllers cannot be put into one executable application, this method could be used simultaneously for multiple Simulink models to develop different parts of the whole software. Then, each model could be turned into a module or service on the Target PC and collaborate with other modules by means of different internal communication services, such as programming queues. Hence, developers would be able to benefit from Service Oriented Architecture (SOA) principles [29] in developing their software. Furthermore, the code can be generated not just for Windows, but also for a wide variety of operating systems that support the *Portable Operating System Interface* (POSIX) [30].

Moreover, this paper addresses some of the issues and difficulties commonly associated with remote development and code generation with MATLAB and Simulink products. One of the main issues is how to integrate device drivers and hardware interfaces into Simulink models. Technically, these interfaces should be turned into re-usable Simulink block libraries. However, the drivers and their related API libraries are installed differently on the remote target system (real-time Linux) than on the operating system of the development environment (Windows). Hence, deriving such blocks demands more advanced treatment. Addressing this issue requires detailed knowledge of S-Function programming as well as the development of *Target Language Compiler* (TLC) files that tell code generators how to generate C code out of the S-Functions.

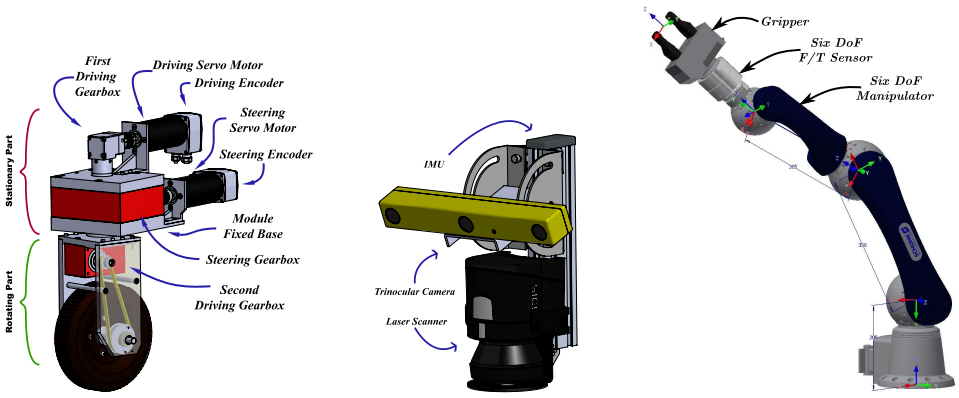
2. Robot Architecture Design

This section details the primary design requirements. We have documented a comprehensive list of the requirements in [31]. The *i*Moro mobile manipulator is designed based on this list. In this section, different components of the mobile platform are presented, and their physical and logical architecture and designed interfaces are explained. Moreover, the implemented embedded controller that facilitates rapid prototyping of the controller software is described. Finally, we explain different implemented control modules designed for performing tele-operation and autonomous tasks. A high-level state machine controller is devised that governs the execution order of the controller modules.

2.1. Design Requirements

As mentioned before, the main aim of the PURESAFE project is to design a mobile manipulator that can be utilized to perform inspection and manipulation tasks in hazardous environments of research facilities. Here, our main focus is to design a mobile base and manipulator for such a robot. Below are the key requirements considered in the design of the mobile manipulator:

1. The mobile platform should be able to carry 30 *kg* of extra load.
2. The mobile platform should be able to pass through normal doors.
3. The mobile platform should have high maneuverability that enables it to rotate and changes its heading in limited spaces.
4. The whole system should be able to perform operations for at least several hours and travel two to three kilometers without recharging.
5. The hardware and software of the whole system should be component-oriented and modular to allow the simultaneous development of different components.
6. The software platform for the controller should support real-time capabilities while providing simple and straightforward means to develop and test different software modules.



(a) Wheel Module (b) Sensor Rack (c) The Manipulator

Figure 2. Designed Modules

7. The designed control modules should provide both autonomous and tele-operated capabilities for the mobile platform.
8. The manipulator should be able to handle a weight of $10kg$
9. The manipulator should have a gripper capable of grasping a cube with a section size of $10cm \times 10cm$
10. The end-effector of the manipulator should have dexterous motion up to a height of $1.2m$ outside of the moving platform.

Some trade-offs among these requirements are necessary in order to maximize the stability of the platform, especially during manipulation tasks. For example, it is desirable to make the platform's size as large as possible. However, the second and third requirements limit the size of the platform. Thus, we consider the maximum dimensions that satisfy those requirements for the overall size of the base. Table 1 lists specific parameters of the mobile platform.

The third requirement presented in 2.1 requires the platform to perform pure rotations. More generally, in order to achieve high maneuverability, the robot should have a decoupled heading and linear movement or, in other words, three degrees of freedom for the main body. As mentioned before, there are two popular types of architectures for mobile robots that hold such characteristics. The first type is the differential drive mobile robot, which has at least three omnidirectional wheels, such as the one presented in [9]. The second type is the pseudo-omnidirectional mobile robot, which has at least two wheels with independent steering and driving actuators. The efficiency of omnidirectional wheels is minimal, hence, such wheels limit the general efficiency of the robot. Moreover, such robots must carry relatively smaller payloads. Based on the payload and efficiency requirements, we designed the mobile platform based on the second popular robot architecture.

Table 1. General specifications of the robot

Description	Quantity
Main Body Length	655 mm
Main Body Width	335 mm
Max. Velocity of Leg's Steering	0.74 $^{rad}/_{sec}$
Max. Velocity of the Platform(30kg load)	2.1 $^{m}/_{sec}$
Wheel Diameter	200 mm
Approximate Mass of Each Leg	20 kg
Approximate Overall Mass	120 kg

2.2. *General Design*

The designed platform consists of four wheel modules (legs). Each leg has two degrees of freedom for independent driving and steering. Figure 2(a) portrays the designed leg, which consists of two parts: a moving part and a fixed part. The fixed part contains the actuators and their directly connected gearboxes. The steering motor rotates the entire moving part along the vertical axis of the leg. Since there are no mechanical constraints, the moving part is able to rotate multiple turns. The first driving gearbox is connected to the second one via a shaft that passes through the steering gear box. The second driving gear box then rotates a pulley-belt system connected to the wheel.

All the actuators are driven by the same motor type (brushless servo motor, 400 watt). Based on the selected motors and other mechanical components, the nominal efficiency of each modular wheel at a speed of $1.7m/sec$ is 70%. The motors are derived using Maxon EPOS2 drivers, which support different control schemes such as position, velocity, and torque based on CiA CANopen® specifications [32]. The drivers for the steering actuators are configured to work in position mode and to return the position as feedback. On the other hand, the drivers for the driving actuators receive and execute velocity commands

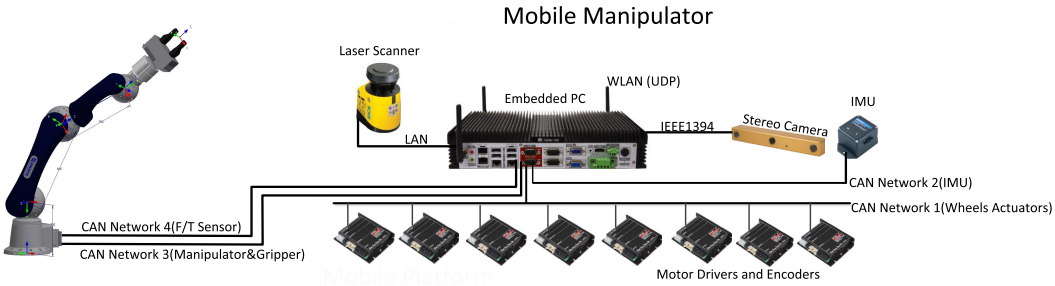


Figure 3. Communication Interfaces of various components

while their desired feedback is configured as the speed of the motor.

The main sensors that are used for localization and navigation (except actuators' encoders) are assembled on an external body called the *sensor rack*. It contains a laser range finder, a stereo camera, and an Inertial Measurement Unit (IMU). Figure 2(b) illustrates the designed sensor rack. As shown in the figure, the camera's field of view and the laser scanner's vertical position can be adjusted at will during the development phase. The main advantage of such a design is that it turns the whole system into an individual module for localization and navigation that can be easily attached or detached from the base. Such a modular design also facilitates the calibration of the sensors with respect to one other. Moreover, it allows the developers to work separately on the module and the mobile platform during development, while the mobile robot can undergo independent developments before the integration phase.

Last but not least, the schematic view of the selected manipulator for the *i*Moro is shown in Figure 2(c). The whole arm contains six DoF manipulators that are connected with six DOF Force/Torque sensors to a double finger gripper. These components are off-the-shelf products. The main challenge here is the synchronized interfacing of these components with the rest of the mobile platform devices, a topic that is thoroughly discussed in the next section.

2.3. Communication Interfaces

This section presents the selected communication interfaces for the *i*Moro. As described in the next subsection, a fanless embedded PC is used as the controller hardware. Sensors and actuator drivers are connected to the controller via various communication interfaces. The schematics are shown in Figure 3.

Figure 3 illustrates that there are four CAN networks in the system. The communication speed of each network is 1 Mbps. The devices connected to each network are selected in such a way as to minimize potential conflicts and maximize communication frequency. For example, the manipulator force feedback is the fastest loop in the system and should be closed with the frequency of $1KHz$. Hence, an entire network is dedicated to the force/torque sensor module to facilitate such a high sample rate. Moreover, since the drivers of the wheel actuators communicate via a similar CANopen[1] protocol, another CAN network is dedicated exclusively to them. The network implements the CANopen *Heartbeat* protocol to continuously monitor the status of the devices. All the devices expect to receive a *Heartbeat* message from the embedded PC every 50ms and to send *Heartbeat* messages to the embedded PC every 100ms. The controller sends desired set points to the drivers every 10 ms using Process Data Objects (PDO). The system implements the CANopen *Sync* protocol in order to synchronize the actuators.

2.4. Embedded Controller

The robot is equipped with an embedded PC that acts as a central controller for the whole mobile platform. Since the robot will need to operate in contaminated areas, the authors have selected a fanless embedded PC with powerful cooling. The PC's quad-core CPU is capable of running multiple software modules, such as vision and motion control, in parallel. Hence, it provides a powerful computational base for executing complex algorithms. Some communications require prompt responses to the data received from sensors or must send the commands at exact time intervals. Therefore, the software platform must provide hard real-time capabilities for some interfacing software modules.

The authors use Xenomai[2] real-time Linux as the *Operating System* (OS) for the embedded PC in order to provide such real-time capabilities. However, it is impossible to execute all the software modules in the hard real-time environment due to the non-real-time nature of some of the software functions. This is the case especially for some interfacing modules, such as *User Datagram Protocol* (UDP) packets and almost all other LAN network protocols that are not generally supported in real-time environment.

Here, the authors use a multi-thread programming approach. Different software modules of the controller are executed in different executing threads. Hence, modules can be run in the hard real-time scheduler of the OS or the non-real-time scheduler, depending on their characteristics. Real-time programming *queues* are used to communicate between different software modules.

[1] www.can-cia.org
[2] www.xenomai.org

The communication interface for the CANopen network is implemented using the Xenomai *RTDM* library, which is executed in hard real-time environment. This is also the case for gathering data from the laser range finder through LAN using the *RTnet* real-time library. However, the process of capturing images from the stereo camera or UDP communication through the wireless network is executed in a non-real-time environment. The implementation of such a strategy can be quite tedious and time consuming, especially in the development phase, when software modules and algorithms subject to constant change and improvement. In order to facilitate rapid prototyping of different algorithms and software modules, especially in real-time environments, the authors implement a unified framework that was initially proposed by the authors in [5]. This framework is presented in the next section.

3. Development Environment for the Controller Software

This section provides a general description of the proposed framework for the rapid prototyping of a PC-based mechatronic system such as a mobile manipulator. The framework allows developers to develop their control software in Simulink and to use MATLAB code generation products to automatically generate and build real-time code for their control modules. The whole system is divided into three distinct environments, which are shown schematically in Figure 4. The *Development Environment* consists of tools that are used to develop the controller and generates the pertinent C code for the whole software. The *Target Environment* consists of the actual mechatronic system, which receives the generated code and compiles it into the real-time software. The *Supervisory System* sends runtime commands to the mechatronic system and generally receives monitoring data and information. Both the *Development Environment* and the *Supervisory System* can be run on the same machine. Detailed descriptions of each part are given in the following sections.

3.1. Remote Development Environment

The *development environment* consists of a Windows PC with MATLAB and Simulink installed (the "Host PC"). The Host PC is connected to the Target PC responsible for controlling the mechatronic system by means of a LAN network. The model for the whole software controller is developed on this system as one or more Simulink models. An SSH connection is used to transfer data between the Host PC and the embedded real-time target. Hence, the Target PC should be configured to accept SSH connections. Moreover, the *PuTTY*[3] SSH software should be installed on the Host PC. MATLAB uses this software to establish a RSH connection to the Target PC and to transfer the generated code. Note that, prior to MATLAB 2011a, the code generation packages were called *Real-Time Workshop* and *Embedded Real-Time Workshop*. Since MATLAB 2011a, these packages have been combined with related packages and are now presented as three products called *Matlab Coder*, *Simulink Coder* and *Embedded Coder*. The reader is referred to MATLAB documentation for further information. The terminologies used in this paper are based on new versions of MATLAB. Those packages have their own licenses and should be installed with MATLAB.

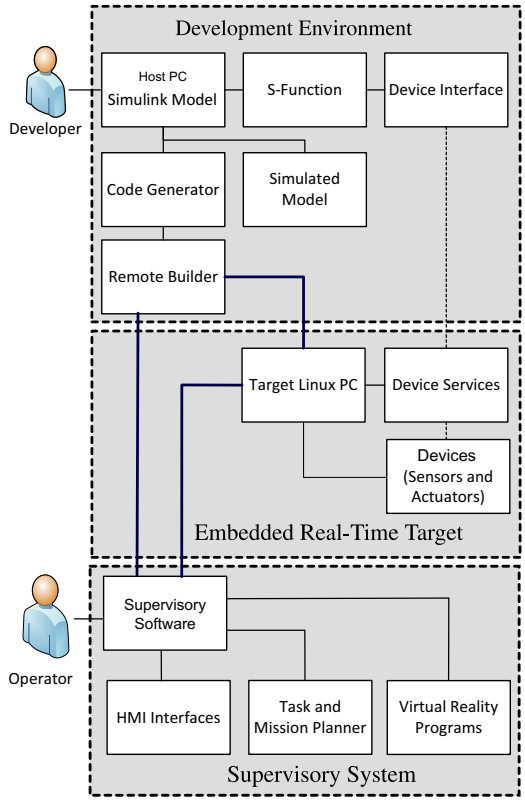


Figure 4. Schematic Diagram of the Rapid Prototyping Framework

The controller and computational algorithms for the mechatronic system are developed as a Simulink model and can be tested with simulation tools in Simulink before being sent to the actual system. Since the 2011 release, Simulink has added a new *Project* feature, which has useful properties such as Subversion® source control. *Project* can be utilized to manage a highly complex controller in a collaborative environment. Once the design evolves to an acceptable level of maturity, generated blocks for device drivers will be added to the model and appropriate signal connections will be set between those blocks and the developed software. Note that some of the blocks in Simulink support only a specific operating system. Although the Simulink model is being developed under Windows, networking blocks such as UDP have both Windows and Linux versions. Since the target system is Linux-based, the model should use the Linux versions of such blocks.

The model requires specific configurations to allow the code generator to produce appropriate code for the Linux

[3] www.chiark.greenend.org.uk/~sgtatham/putty/

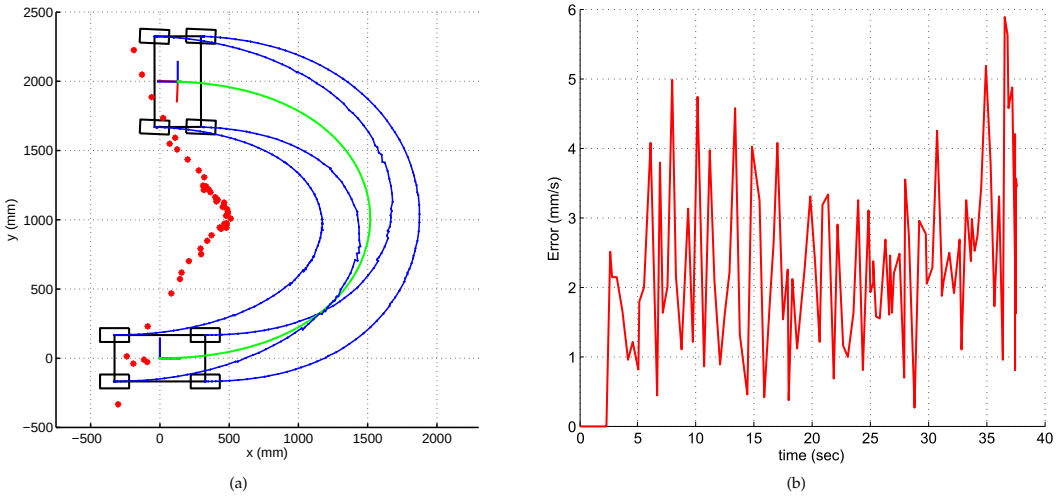


Figure 5. Experiment: (a)Mobile Platfrom Follows a half-circle trajectory and change its heading 180°. (b): The norm of synchronization error between the wheels' driving actuators

target. Once these configurations are made, the *Embedded Coder* is used to generate C code for the whole Simulink model. Along with generating code, the *Embedded Coder* is configured to generate the Makefile for the produced code with appropriate paths and flags. The Makefile is based on set of configurable templates. However, it is made for the Host PC which means that, although the libraries used in the generated code are designed for Linux, the paths and flags associated with them are related to the MATLAB installed folder in the Host PC. Hence, the Makefile is useless in the Target PC.

Nevertheless, a useful MATLAB function called *remoteBuild* can be used to solve this issue. *remoteBuild* receives as input an object called *buildInfo*, which is a by-product of the code generation process, along with information on the Target PC such as its IP address and user credentials. First, the *remoteBuild* function reads the produced Makefile, gathers all the headers and libraries based on their Makefile paths, and adds them into an archived folder along with the generated code. Then, the function modifies the Makefile and produces another with appropriate paths and flags, which it adds to the archive as well. Next, it establishes an RSH connection with the Target PC and extracts the archive there. Finally, the *make* command is used to compile and build the software executable on the Target PC. Note that this entire process is done automatically. Further notes on the configurations that should be made in the Simulink model for using *remoteBuild* are given in [5]. Note that the executable application made on the Target PC is fully independent of MATLAB, the Host PC, or any other part of development environment. Hence, the entire environment can be disconnected from the embedded real-time target.

3.2. Embedded Real-Time Target

The real-time target consists of an embedded PC running a real-time Linux operating system. It is connected to hardware devices by means of different communication networks and protocols, such as Serial, USB, CAN, etc. It is assumed that device drivers and software libraries associated with those devices are properly installed on the Target PC. As mentioned before, the execution of the *remoteBuild* function copies the generated code, along with the Makefile and other dependencies, to a specified path in the real-time Linux system. *remoteBuild* uses a GNU compiler to compile and build the code into executable software. Once the software is run, it can receive runtime commands from the supervisory system and send logging data and monitoring information in return.

3.3. Supervisory System

As mentioned before, the Supervisory System is in fact the operator station and is used to send runtime commands and receive monitoring data from the Target PC. Physically, it can be the Host PC, the Target PC, or a fully separate system that is connected to the Target by means of a computer network (more technically, through the sending and receiving of UDP packets). In advanced cases, it could contain HMI interfaces, virtual reality programs and task planners. Again, code generation products can be utilized for the development of such software.

4. Experimental Results

The proposed method was implemented on the *i*Moro mobile manipulator. The host PC connects to the department's wireless intranet via a Wi-Fi modem. It is accessible through the school network. Over this network, the embedded Target PC on the *i*Moro with Xenomai Linux is connected to the Host PC, which has MATLAB R2013a installed on it. The framework presented in this paper was used to develop a software controller for the robot. Several controller modules were developed in Simulink. The modules consist of systems for receiving

tele-operation commands, obstacle direction using the laser scanner, differential driving, etc. In order to interface with hardware components, the authors implemented the approach discussed in the previous sections with the technical details given in [5].

The entire real-time software for the robot based on the methodology discussed in Section 3. The complete model was developed in Simulink. Then, the Embedded Coder toolbox was used to generate the code from the model, with the real-time Linux system defined as the target. The code was sent automatically to the Target PC and built by the GNU compiler. The software has a base thread with a real-time periodic sample time of $1e-4$ seconds. No calculations are done in the base thread. The thread is used as a trigger for activating slower periodic tasks that implement the control modules and device communications. The faster periodic tasks have a sample time of $1e-3$ seconds and close the force/torque feedback loop of the manipulator's F/T sensor. The main periodic tasks have a sample time of $1e-2$ seconds and run the low-level control algorithms for both the wheels and the manipulator actuators. This arrangement guaranties synchronization between the actuators of the arm and the mobile base.

To show the accuracy of the synchronization between the actuators, we present some results of a sample experiment with the mobile platform. In this example, the platform is required to follow a desired curve and the synchronization error between the wheel-driving actuators is recorded. The path followed by the mobile platform is a cubic *Bezier* curve with its control points located at $\{(0,0),(2m,0),(2m,2m),(0,2m)\}$. The platform's desired heading profile changes from 0 to 180°. Figure 5(a) shows the body and leg trajectories following the given profile. It also depicts as red dots the estimated location of the body's Instantaneous Centre of Rotation (ICR) during its movement. Figure 5(b) renders the norm of the synchronization error between the wheel-driving actuators. From the figure, it is clear that, due to real-time synchronization of the set points, the velocity error between the actuators is far less than the medium of the actuators' speed, which is $100mm/sec$. In the event that the wheel odometer feedback is used to localize the base, this error can be further reduced by using the redundancy in the number of the mobile base actuators. We have presented the details for such treatment in [6].

5. Conclusion

In this paper, we present the mechatronics design of the *i*Moro mobile manipulator. We discuss the low-level software architecture based on embedded pcPC-based controls and present a systematic solution based on code generation products of MatlabMATLAB and Simulink. We The authors propose a simplified solution for rapid prototyping of the controller software in the hard real-time Linux environment that significantly reduces the complexities often arises in the design and implementation of low- level real-time controllers. The proposed framework allows the developers to easily synchronize the collection of various sensor data and actuator commands, that notably increases the performance of the mobile manipulator.

6. Acknowledgement

This work, supported by the European Union's Seventh Framework Program under the Marie Curie Initial Training Network, was carried out within the framework of the PURESAFE, *Preventing hUman intervention for incREased SAfety in inFrastructures Emitting ionizing radiation*, under REA Grant Agreement Number 264336.

7. References

[1] Brad Hamner, Seth Koterba, Jane Shi, Reid Simmons, and Sanjiv Singh. An autonomous mobile manipulator for assembly tasks. *Autonomous Robots*, 28(1):131–149, 2010.

[2] Robert Holmberg and Oussama Khatib. Development and control of a holonomic mobile robot for mobile manipulation tasks. *The International Journal of Robotics Research*, 19(11):1066–1074, 2000.

[3] J. van Amerongen. Mechatronic design. *Mechatronics*, 13(10):1045–1066, 2003.

[4] Mohammad M. Aref, R. Ghabcheloo, Antti Kolu, M. Hyvönen, K. Huhtala, and Jouni Mattila. Position-based visual servoing for pallet picking by an articulated-frame-steering hydraulic mobile machine. *IEEE International Conference on Robotics, Automation and Mechatronics (RAM)*, 2013.

[5] R. Oftadeh, M. M. Aref, R. Ghabcheloo, and J. Mattila. Unified framework for rapid prototyping of Linux based real-time controllers with Matlab and Simulink. In *IEEE/ASME Conference on Advanced Intelligent Mechatronics (AIM), Taiwan*, pages 274 –279, July 2012.

[6] Reza Oftadeh, Mohammad M Aref, Reza Ghabcheloo, and Jouni Mattila. Mechatronic design of a four wheel steering mobile robot with fault-tolerant odometry feedback. In *Mechatronic Systems*, number 1, pages 663–669, 2013.

[7] Roland Siegwart and Illah Reza Nourbakhsh. *Introduction to Autonomous Mobile Robotos*. The MIT press, 2004.

[8] C.Y. Chen, T.H.S. Li, Y.C. Yeh, and C.C. Chang. Design and implementation of an adaptive sliding-mode dynamic controller for wheeled mobile robots. *Mechatronics*, 19(2):156–166, 2009.

[9] WK Loh, K.H. Low, and YP Leow. Mechatronics design and kinematic modelling of a singularityless omni-directional wheeled mobile robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3237–3242, 2003.

[10] J.E.M. Salih, M. Rizon, S. Yaacob, A.H. Adom, and M.R. Mamat. Designing omni-directional mobile robot with mecanum wheel. *American Journal of Applied Sciences*, 3(5):1831–1835, 2006.

[11] O. Diegel, A. Badve, G. Bright, J. Potgieter, and S. Tlale. Improved mecanum wheel design for omni-directional robots. In *Proc. 2002 Australasian Conference on Robotics and Automation, Auckland*, pages 117–121, 2002.

[12] J.B. Song and K.S. Byun. Steering control algorithm for efficient drive of a mobile robot with steerable omni-directional wheels. *Journal of mechanical science and technology*, 23(10):2747–2756, 2009.

[13] C.P. Connette, A. Pott, M. Hägele, and A. Verl. Addressing input saturation and kinematic constraints of overactuated undercarriages by predictive potential fields. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4775–4781, 2010.

[14] K. Berns, K.D. Kuhnert, and C. Armbrust. Off-road robotics-an overview. *KI-Kunstliche Intelligenz*, 25(2):109–116, 2011.

[15] T.L. Lam, H. Qian, Y. Xu, and G. Xu. Omni-directional steer-by-wire interface for four wheel independent steering vehicle. In *IEEE International Conference on Robotics and Automation, ICRA*, pages 1383–1388, 2009.

[16] C. Parlitz, M. Hägele, P. Klein, J. Seifert, and K. Dautenhahn. Care-o-bot 3-rationale for human-robot interaction design. In *Proceedings of 39th International Symposium on Robotics (ISR), Seul, Korea*, pages 275–280, 2008.

[17] P.F. Santana, J. Barata, and L. Correia. Sustainable robots for humanitarian demining. *International Journal of Advanced Robotic Systems*, 4(2):207–218, 2007.

[18] Miguel Vilaplana, Oliver Mason, Douglas Leith, and William Leithead. Control of yaw rate and sideslip in 4-wheel steering cars with actuator constraints. *Switching and Learning in Feedback Systems*, 3355:201–222, 2005.

[19] M.F. Selekwa and J.R. Nistler. Path tracking control of four wheel independently steered ground robotic vehicles. In *50th IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 6355–6360, 2011.

[20] D. Henriksson, A. Cervin, and K.E. Årzén. Truetime: Real-time control system simulation with matlab/simulink. In *Proceedings of the Nordic MATLAB Conference, Copenhagen, Denmark*, 2003.

[21] Anton Cervin. *Integrated Control and Real-Time Scheduling*. Department of Automatic Control, 2003.

[22] G. Quaranta and P. Mantegazza. Using matlab-simulink rtw to build real time control applications in user space with rtailxrt. In *Realtime Linux Workshop. Milano*. Citeseer, 2001.

[23] Pieter J. Mosterman, Sameer Prabhu, Andrew Dowd, John Glass, Tom Erkkinen, John Kluza, and Rohit Shenoy. Embedded real-time control via matlab, simulink, and xpc target. In Dimitrios Hristu-Varsakelis, William S. Levine, and William S. Levine, editors, *Handbook of Networked and Embedded Control Systems*, Control Engineering, pages 419–446. Birkhauser Boston, 2005.

[24] S. Abourida and J. Belanger. Real-time platform for the control prototyping and simulation of power electronics and motor drives. In *Proceedings of the Third International Conference on Modeling, Simulation and Applied Optimization*, pages 1–6, 2009.

[25] Roberto Bucher and Silvano Balemi. Rapid controller prototyping with matlab/simulink and linux. *Control Engineering Practice*, 14(2):185 – 192, 2006.

[26] J.M.P. Cardoso, P. Diniz, M.P. Monteiro, J.M. Fernandes, and J. Saraiva. A domain-specific aspect language for transforming matlab programs. In *Domain-Specific Aspect Language Workshop (DSAL'2010), part of AOSD*, pages 15–19, 2010.

[27] Ciprian Lapusan, Vistrian Maties, Radu Balan, and Olimpiu Hancu. Rapid control prototyping in design process of mechatronic systems. *Solid State Phenomena*, 166-167:247–252, September 2010.

[28] Roberto Bucher and Silvano Balemi. Rapid controller prototyping with Matlab/Simulink and linux. *Control Engineering Practice*, 14(2):185–192, 2006.

[29] W. Shen, Q. Hao, S. Wang, Y. Li, and H. Ghenniwa. An agent-based service-oriented integration architecture for collaborative intelligent manufacturing. *Robotics and Computer-Integrated Manufacturing*, 23(3):315–325, 2007.

[30] K. Yaghmour, J. Masters, G. Ben-Yossef, and P. Gerum. *Building embedded Linux systems*. O'Reilly Media, 2008.

[31] M. M. Aref, R. Oftadeh, J. Koivumäki, R. Ghabcheloo, and J. Mattila. System requirement document for modular mobile manipulator system. *PURESAFE System Engineering Documents*, WP3(RP10-RP14), 2012.

[32] MSB LSB. 6 general object definitions. *IEC 61800-7-200: Adjustable speed electrical power drives systems–Part 7-200: Generic interface and use of profiles for power drive systems–Profile specifications*, page 3577, 2007.