EU Horizon 2020 Research & Innovation Program
Advanced Robot Capabilities & Take-Up
ICT-25-2016-2017
Grant Agreement No. 779942

Safe Robot Navigation in Dense Crowds

http://www.crowdbot.org

Technical Report
D 3.4: Reactive Motion Planning

Work Package 3 (WP 3)
Navigation

Task Lead: École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
WP Lead: Swiss Federal Institute of Technology (ETHZ), Switzerland

Table of Contents

Executive Summary

This report details the reactive navigation techniques developed for the CrowdBot project between months M1 to M30. In this sense, we have investigated three main technical components for achieving reactivity in different types of mobile or service robots when navigating in crowded environments.
Each of the three technical components are designed to complement high-level planning techniques and focus on exploiting different sensing capabilities of the different robots. In this report, we present tests in both simulation and real environments showcasing the applicability of the proposed solutions for dynamic environments..

In particular, the three key challenges that we addressed are:

- Fast reactive navigation based on proximity sensing that adapts quickly to dynamic environments with pedestrians and guarantees obstacle avoidance.
- Executing low-latency local collision avoidance considering the actual physical constraints of the robots, from shape to kinematic constraints.
- Post-collision response in case of pedestrians unpredicted or untrackable behaviour by controlling the direction of motion of the robot to minimize the contact forces exerted.

For proximity based solutions to the reactive navigation problem, we present two approaches. The first solution is based on a dynamical system (DS) representation of the robot's control input and its environment (obstacles) offering a continuous solution in closed-form equations of harmonic potential flow around multiple moving obstacles, as such, enables the robot to react immediately to dynamic obstacles. This approach was implemented both in simulation and on a wheeled omnidirectional robot and validated for holonomic robot control in 2D space, effectively controlling the robot direction and velocity in a plane. Further details are described in section 3, and the open source code is available at: https://github.com/epfl-lasa/dynamic_obstacle_avoidance_linear .

The second approach focuses on non-holonomic (limited motion capabilities as wheelchairs) and non-circular robots, providing a solution to complement high-level planners and navigate tight environments with accurate robot shape descriptions. This would be an advantage for robots whose shape would be over increased if a point-mass approach were to be used. The proposed method formulates a convex optimization problem for representing the robot with accurate shapes, offering a reactive response that assumes a coupling with a higher level planner (navigation algorithm or human driver); extending the method of velocity obstacles (VO). This approach is validated in both simulation environments and real robots with the CrowdBot robot Qolo which has similar kinematics to standard powered wheelchairs. Further details are described in section 4, and the open source code is available at: https://github.com/epfl-lasa/rds.

Finally, post-collision reactivity was investigated through the concept of compliance control for achieving a directional compliance to unexpected collisions that would minimize the collision force without forcing the robot to freeze. We present a method for estimating contact forces at the robot's surface through learning nonlinear stiffness and compliance to internal force torque sensors. Then estimated collision location and magnitude is used in a continuous compliant controller that cancels the contact forces in a closed-loop system. Further information is detailed in section 5.

1. Introduction

CrowdBot project deals with robot motion, control, and decision-making processes within dense crowd environments, therefore, a fundamental component is to have reactive motion planning that would deal with unexpected changes in the dynamics of the surrounding crowd that could not be predicted fast enough for full-motion replanning. Herewith, the methods presented in this report focus in complementing high-level motion planners as described in D3.1 and D3.3.

In this report, we detail two main components developed for reactive navigation that could be coupled with any high-level path planning, including direct human input in shared-control schemes. First, we investigate proximity based reactivity through the usage of the robots' available LIDAR and RGBD sensing, which aims to guarantee obstacle avoidance based on the models of the pedestrian motions and the state of the robot. Second, we investigate the post-collision reactions in case of pedestrians' unpredictable abrupt motions causing contact with the robot, in this case, we make use of contact sensing such as force and torque (F/T) sensors, for achieving compliance in the direction of the impact.

The algorithms reported here have been tested in the CrowdBot version of the Qolo robot [Paez-Granados D, et al, 2018], adapted with the recommendations from D6.2. Both in simulation and real life experiments within laboratory settings and natural crowds. For such experimental validations we have integrated the CrowdBot detection and tracking system documented in D2.2, with our array of LIDAR and RGBD sensors. In parallel simulated evaluations have been developed in the CrowdBot simulator documented in D4.2 where the Qolo robot has been integrated.

In the following section, we describe the state of the art in collision avoidance, describing the contribution of the developed algorithms for dynamic environments for both holonomic, and non-holonomic robots. Section 3 details the first approach and experiments validating on an omnidirectional robot. Section 4, describes in detail the robot Qolo adapted for crowd navigation. Finally, section 5, addresses the second approach focused on non-circular and non-holonomic robots and assistive driving testing with the robot Qolo.

2. Related Work

Robots that work in real-life environments often have to deal with dynamic obstacles like a pedestrian running in front of an autonomous car or a bird flying in front of a drone. In such cases, the robot cannot follow its initial path anymore and has to compute a new path quickly enough to avoid a collision. Such robots also have to face situations in which the state of the environment differs from their inner knowledge as static obstacles change to a new configuration. The problem to adapt and control the robot's motion suitably in such situations is in the focus of several related research fields, known as reactive control, reactive motion planning, collision avoidance and obstacle avoidance. Recent reviews are given e.g. by [Kamil et al., 2015] and [Hoy et al., 2015].

2.1. Obstacle avoidance

Methods for obstacle avoidance typically emphasize convergence in addition to preventing collisions, i.e. they provide navigation laws that move the robot through free space and guarantee to lead it to a particular destination. They share these two goals with path planning methods [Choset et al., 2005], which have a stronger focus on static environments. Addressing mostly the static case, numerous techniques have been proposed, such as artificial potentials [Khatib, 1986], navigation functions [Rimon and Koditschek, 1992], [Rimon and Koditschek, 1991], and probabilistic exploration by random trees. On the other hand, velocity obstacles [Fiorini and Shiller, 1998] are a traditional framework for planning trajectories among moving objects.

Planning a trajectory that leads the robot from its current position to the target location can be computationally expensive, especially in complex environments. Furthermore, the whole trajectory may have to be re-planned if it is made invalid by a sudden change in the environment.

Other methods try to avoid path planning, e.g. recent developments use power diagrams to identify a collision-free convex neighbourhood around the robot. A continuous flow is then generated by solving the associated convex optimization problem [Arslan and Koditschek, 2016] and convergence is ensured for convex obstacles. Machine learning algorithms have been directly applied to sensor data to obtain data-driven control laws [Michels et al., 2005] but they cannot ensure impenetrability. Other similar navigation functions transform star-shaped obstacles and trees of stars into simpler environments in which obstacles are reduced to spheres and convergence to the global minima can be ensured for almost all trajectories [Paternain et al., 2017].

Ensuring both convergence to the target location and impenetrability is not a trivial task. Some path planning oriented methods ensure global convergence for quasi static environments at the expense of a high computational cost which precludes online applications [LaValle and Kuffner, 2001]. On the software side, re-planning only a part of the trajectory [Ferguson et al., 2006] or deforming it locally [Brock and Khatib, 2002] are ways to reduce the workload, but do not guarantee global convergence. Other approaches focus on improving the hardware for faster processing [Murray et al., 2016]. Such advances in software and hardware have made possible the real-time application in dynamic environments of optimization algorithms such as model predictive control [Rasekhipour et al., 2016].

Control using dynamical systems (DS) offers an alternative to address such situations. The control law can ensure the impenetrability of obstacles and does not require re-planning as it is closed form [Feder and Slotine, 1997]. A DS-based solution guarantees stability and convergence while offering on-the-fly reactivity [Khansari-Zadeh and Billard, 2011]. This approach has been extended to a Modulated Dynamical System

(MDS) which represents obstacles analytically as star-shaped level sets of a distance function and considers the robot as a point moving in cartesian space (in the case of mobile robot navigation). It guarantees to lead the robot to its goal by exploiting the assumptions that the robot is holonomic, and it offers a closed-form solution, as such, it enables the robot to react immediately to obstacles. This is particularly useful when the robot has only a partial view of the environment, and that obstacle may suddenly appear following an update on on-board sensing [Huber et al., 2019].

2.2. Reactive control and collision avoidance

Reactive control in a broad sense is concerned with making robots react appropriately to unforeseen events, e.g. avoid a collision with an appearing object or person. Works from this branch thus mostly aim to affect the robot's short-term behaviour [Mansard and Chaumette, 2007], [Dietrich et al., 2012]. Disregarding long-term trajectory planning and restricting the focus on collision avoidance itself simplifies the problem and favours computationally lightweight approaches which are applicable on complex robotic systems with high degree of freedom and kinematic constraints [Stasse et al., 2008].

Among such approaches, quadratic programs are popular as they can handle arbitrarily many constraints (in principle) for achieving collision avoidance between multiple limbs [Escande et al., 2014]. The method [Rauscher et al., 2016] treats robotic arms and uses constraints for collision avoidance within a QP wherein the nominal velocity commands set the (unconstrained) optimum in the objective function.

2.3. Non-holonomic and non-circular robot navigation

Many mobile robots violate the assumptions of some obstacle avoidance methods that the robot's shape is circular and that its lateral and longitudinal velocity can be chosen freely. While purely reactive control methods (as those discussed in the previous section) typically incorporate these aspects in their robot description, they do not provide guarantees beyond collision avoidance, e.g. regarding optimal avoidance trajectories. On the other hand, some navigation approaches specifically take these complications into account and optimize trajectories over a finite time horizon.

The dynamic window approach [Fox et al., 1997] is a popular method, which treats robots with non-holonomic constraints, which occur in almost any vehicle with wheels. It optimizes over a set of circular arc trajectories, where each trajectory is characterized by a constant linear (longitudinal) and angular velocity. It solves at every time step an optimization problem subject to constraints that represent potential collisions in the near future and dynamic constraints (due to acceleration bounds). However, it does not represent obstacle velocities and assumes a circular robot shape. It has also been applied for semi-autonomous wheelchairs [Carlson and Demiris, 2012]. A related approach is given by [Schlegel, 1998], which can handle non-circular shapes, too, but does not take into account object velocities either.

We propose the RDS method to take into account object velocities in addition to non-holonomic kinematics and a non-circular shape for the robot. The method employs the well-known velocity obstacle (VO) concept [Fiorini and Shiller, 1998], which originally considers circular obstacles that move with constant velocities and defines for the circular agent the corresponding cone-shaped sets of constant velocities that will lead to a collision in the future. The proposed method formulates a convex optimization problem similar to the Optimal Reciprocal Collision Avoidance (ORCA) method [Van Den Berg et al., 2011]. ORCA limits the time horizon for considering collisions in the future to a finite value τ and thus truncates each VO cone. Disregarding long term interactions, ORCA enables agents to avoid collisions in the near future in an optimal manner, deviating minimally from their preferred velocities.

In contrast to ORCA, our method treats non-circular robot shapes and thus takes into account rotation and not only translation. Still, the differential degree of freedom remains two due to the non-holonomic kinematics. More specifically, the robot's instantaneous center of rotation must be on the infinite line which contains the wheel axle. While many prior works extend VO-based methods such as ORCA to non-circular or non-holonomic robots, they mostly consider only one of these two aspects (e.g. [Alonso-Mora et al., 2013] considers the non-holonomic case). When considering non-circular holonomic robots, it is common to replace in the classical VO the robot's shape by the area which it sweeps for a given amount of rotation (e.g. in [Best et al., 2016], [Ma et al., 2018], [Giese et al., 2014]), which allows to separate the rotational and translational velocity computation. They rely on pre-computed look-up tables that store such swept surfaces for different amounts of rotation. In contrast, we employ the VO concept in a more simple and lightweight framework addressing non-holonomic and non-circular robots.

We introduce a novel technique, considering individual collisions between surrounding agents and the closest corresponding incircle in the robot's capsule shape and constructing the relative VO. The approach linearly approximates these individual VOs and relates them via the robot's nonholonomic kinematic mapping to yield a single problem for the velocity of a specific robot-fixed reference point. This problem's solution prescribes the reference point velocity and thereby also implicitly both the robot's linear and angular velocity due to the nonholonomic kinematic relations.

3. Modulated Dynamical System

The main objective of the reactive control module is to provide a layer for local navigation assistance and immediate responsiveness to unmodeled or unpredicted object occurrences or motions. Therefore, this algorithm assumes a continuous dynamical system guiding the robot's motion to be existing and given by a high-level algorithm that plans for optimality towards its intended goal.

The Modulated Dynamical System (MDS), represents obstacles analytically as star-shaped level sets of a distance function that absorb the robot's footprint, thus, allowing the robot to be represented as a point moving in the cartesian space, as shown in Figure 3.1. It guarantees to lead the robot to its goal (hereafter called attractor) by exploiting the assumptions that the robot is holonomic, deriving its virtual boundary as a circle-shaped and that the objects are star-shaped [Huber L., et al 2019].

In this section, we explain this concept and its evaluation on an omnidirectional wheeled platform using onboard sensing.

3.1. System Representation

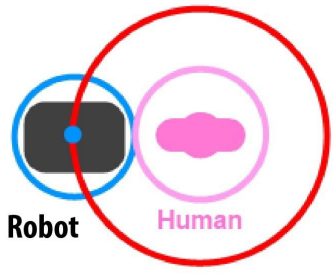


Figure 3.1. Example of robot-human representation for the approach on reactive motion planning.

To understand this approach one first needs to understand how the system will approximate its world. In the case of robot obstacle avoidance, this means, the robot and the obstacles. This method represents each obstacle by a corresponding continuous distance function $\Gamma()$ with three regions, at the boundary ($\Gamma()=1$) means the robot is in a position which implies contact with the obstacle; at the exterior of the obstacle with a distance greater than one ($\Gamma()>1$) meaning positions at a distance greater than zero from the obstacle boundary; or within the obstacle ($\Gamma()<1$), as described in Figure 3.2.

From this definition, we can extract the normal direction pointing away from the object's surface as the gradient of the distance function as $[n_1, n_2] = \frac{\partial\Gamma}{\partial x} / |\frac{\partial\Gamma}{\partial x}|$, which allows constructing a tangential unit vector $[e_1, e_2] = [-n_2, n_1]$

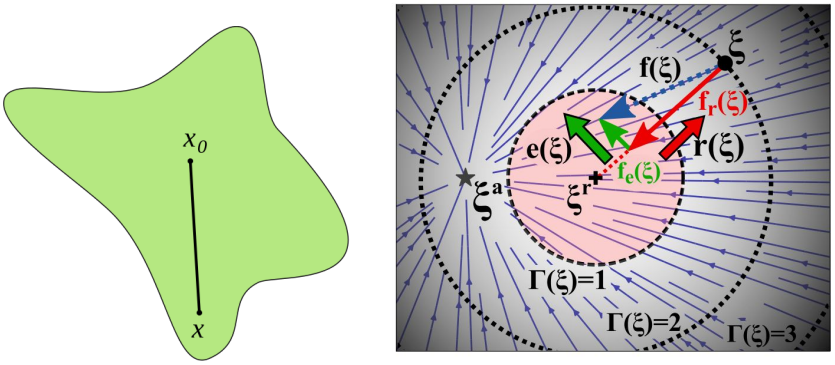


Figure 3.2: Left: Star-shape definition for obstacles, right: robot's definition of the distance function used to characterize the obstacle [Huber L., et al 2019].

The method guarantees impenetrability of the obstacles as long as we can define them as star-shape, which is any concave or convex shape where there exists a reference point (x_o) inside its boundary that could connect in a straight line with any point in the surface (x) without crossing the boundary (as shown in Figure 3.2, left).

3.2. Obstacle Avoidance Formulation

For a given goal location x_{goal}, the method constructs a nominal velocity field $f(x)$ which guides the robot from every possible position in space straight to the goal according to $f(x) = -k(x - x_{goal})$. Multiplication of this nominal velocity with the modulation matrix M gives the actual velocity command for the robot to avoid collisions with the objects in its way. The modulation matrix M has the form $M := EDE^{-1}$, with diagonal matrix D and with the matrix E having the vectors r and e as the first and second column, respectively. The entries on the diagonal of D have therefore the effect to rescale the nominal velocity's components in the direction towards the reference point and the tangential direction. Herewith, the method lets the first entry on the diagonal decay to zero as Γ approaches one and the second entry increases by the respective amount, which absorbs the nominal velocity's normal component (as the reference point is within the object and the object is assumed to be star-shaped) and amplifies the tangential component. Thereby, it guarantees both collision avoidance and reaching the goal.

The following example in Figure 3.3, displays a vector field directed to the attractor (green dot) at $x_{goal} = (2, 0)$ () being modulated by a single obstacle and 2 obstacles. The red dots in the left represent initial states of the robot.

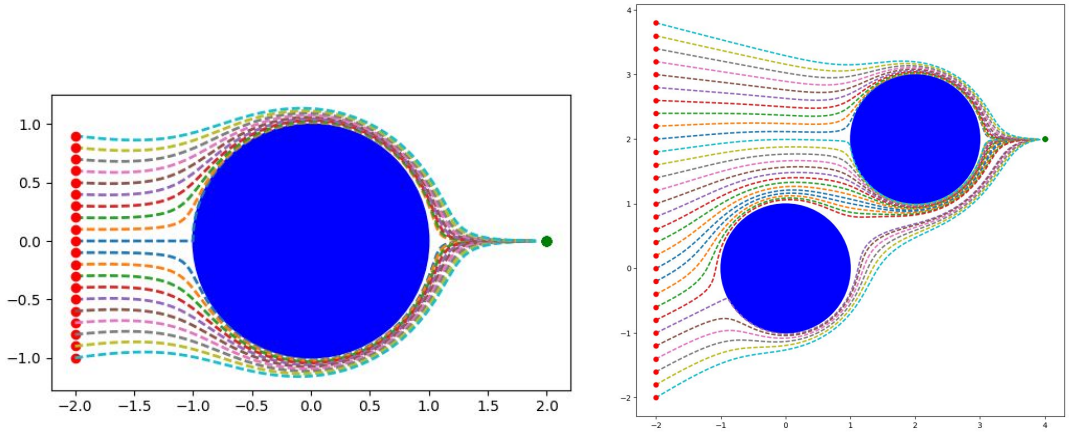


Figure 3.3. Simple obstacles as circles in a linear dynamical system modulated by the current approach.

When navigating in crowds, avoiding obstacles very rapidly using low-level command is crucial as these obstacles may appear rapidly and time to react is very limited. Therefore, this approach would fit into a low-level controller guaranteeing impenetrability of obstacles given an autonomous dynamical system that guides the robot towards its desired goal.

Multiple obstacles as it is expected in crowd navigation can be equally handled by the current method as long as they pose a star-shape. For this purpose a weighted sum of the modulated dynamical system for each obstacle's distance function $\Gamma(x)$ is used, separating magnitude and directional control.

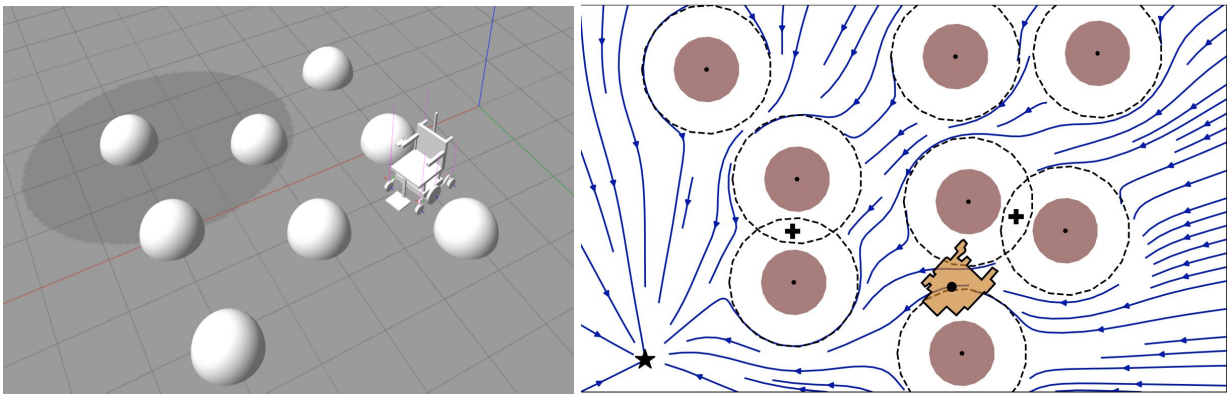


Figure 3.4: Simulation of a 2D navigation with a wheelchair represented as a point-mass and all pedestrians as circles. On the left, the 3D simulation, and on the right, the modulated velocity field followed by the robot [Huber L., et al 2019].

In Figure 3.4, an example for a point-mass control of a wheelchair is used around multiple sphere obstacles (representing humans), in the right side the modulated dynamical system is presented.

3.3. Extended Representation of Obstacles

The modulated dynamical system (MDS) approach for performing obstacle avoidance [Huber L., et al 2019], allows to handle continuous obstacle avoidance as long as the obstacles could be defined as a star-shape. An initial solution for handling non-star shape obstacles in the dynamical system (velocity field) modulation has been devised with three steps for each obstacle (as described in Figure 3.5).

First, a coordinate transformation maps the position of the robot and the obstacle at hand into space where the obstacle is defined as a circle. There, the robot's angular position is such that it preserves across both spaces the arc length along the obstacle circumference between the projections of the robot and the attractor on the obstacle boundary. The robot's distance to the obstacle boundary in the new space is a monotonic function of the original distance.

Second, the MDS computes the velocity command in the new space, and third, the command is transformed back into the actual space by multiplication with the Jacobian of the inverse transformation. The final velocity command results as the distance-weighted average of the velocity commands computed in this manner for each individual obstacle.

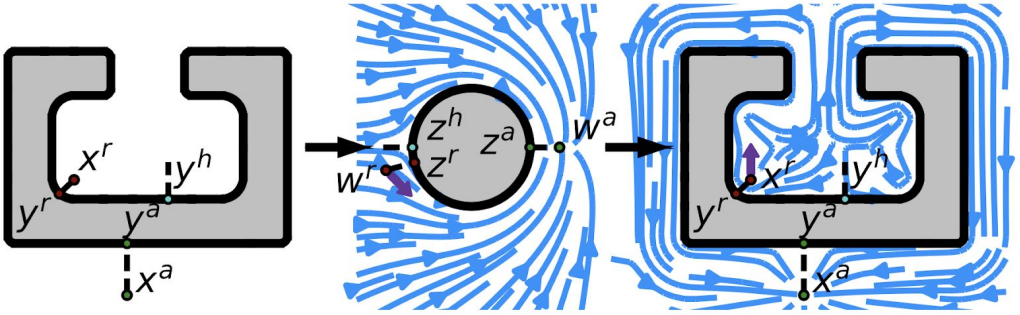


Figure 3.5: The dynamical system for obstacle avoidance is computed in three steps per obstacle.

Impenetrability is still guaranteed for multiple star-shaped or non-star-shaped obstacles regardless of their position. Several obstacles can be considered at the same time in areas where their influence overlap by using a weighted mean of the velocity commands generated by all obstacles independently.

To limit the influence of each obstacle to a small range, the weight of any contribution becomes zero if the distance to the respective obstacle is larger than that range. Convergence to the attractor is guaranteed for any trajectory that does not enter a region which falls into the range of two or more obstacles. The velocity field modulation result for multiple obstacles is shown in Figure 3.7.

Subsequently, an initial solution for applying this algorithm in real-time navigation compatible with any current navigation system is given here through a method for transforming an occupancy grid (common mapping method in SLAM and navigation) representing the environment into a continuous shape (including concave) obstacle. As our projection method works with the obstacle boundaries, an enclosing surface is needed to be constructed for each group of occupied cells.

An interpolation with cubic Bezier splines is performed to get a smooth surface even if the edge of a group of cells is not aligned with the grid (staircase effect) as shown in Figure 3.6. The approach obtains a smooth boundary for each obstacle by fitting a Bézier spline to the occupied cells.

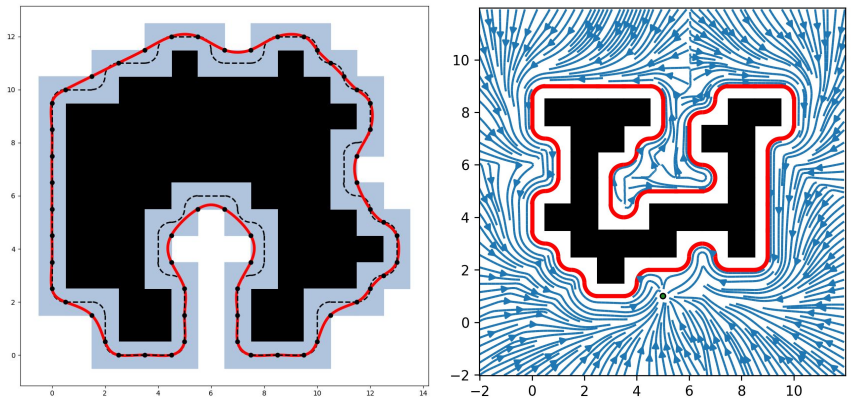


Figure 3.6: Obstacle transformation from a grid map to a continuous shape through Bezier curves

Finally, we present an example of map transformation including concave and convex objects being transformed from a standard grid map into a continuous shape, using the modulation and space transformation described above.

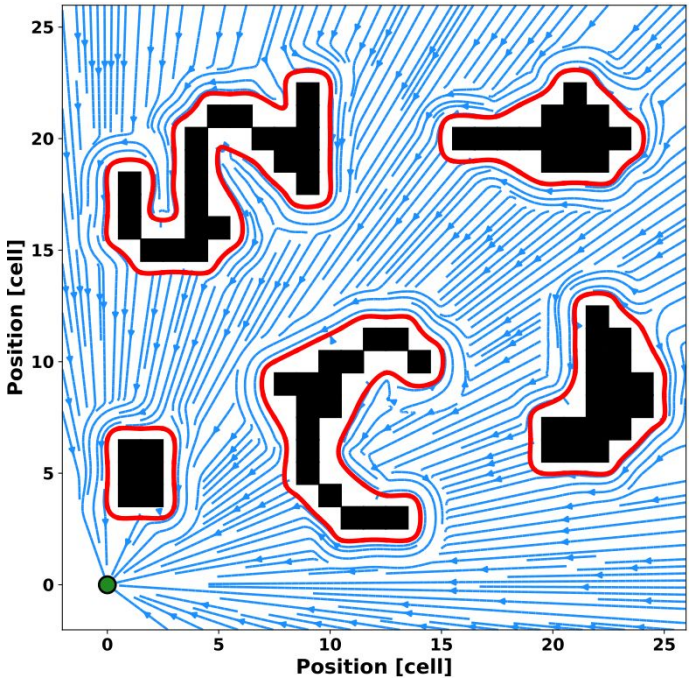


Figure 3.7: The example shows the velocity field for an arrangement of several concave obstacles with guaranteed convergence to the attractor.

3.4. MDS applied to Non-holonomic Robots

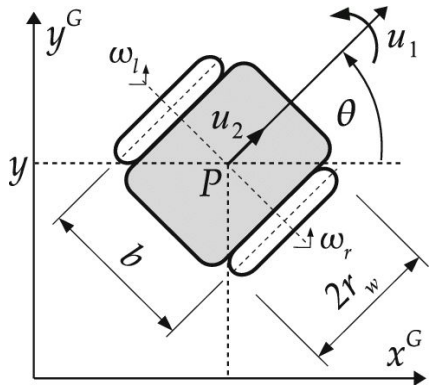


Figure 3.8: An example of a non-holonomic robot, this unicycle system has a set of wheels over the same axis.

Non-holonomic motion implies that the number of degrees of freedom (DOF) of the robot is less than the space of work in which it will operate.Therefore, there are constraints in the achievable direction of motion, i.e., guaranteeing that the robot would follow the desired direction of motion for avoiding unexpected obstacles becomes more challenging.

In this section we address the problem through a nonlinear model predictive control (nMPC), so that all the kinematic constraints of the robot motion capabilities are taken into account, and solved in real-time for reactive navigation.
In the case of mobile robots, a common configuration is that of a unicycle as in Figure 3.8. In this case, the robot has only 2 DOF [ω_l ω_r], for configuration space in 3D [X, Y, θ]. The kinematics of the robot can be defined as,

$$\begin{aligned} \dot{x} &= \tfrac{\omega_R+\omega_L}{2} \cdot r \cdot cos(\theta) \\ \dot{y} &= \tfrac{\omega_R+\omega_L}{2} \cdot r \cdot sin(\theta) \\ \dot{\theta} &= \tfrac{\omega_R-\omega_L}{b} \cdot r \end{aligned} \quad \Rightarrow \quad \begin{cases} \dot{x} = v \cdot cos(\theta) \\ \dot{y} = v \cdot sin(\theta) \\ \dot{\theta} = \omega \end{cases}$$

where x, y, and θ are the position and the orientation of the robot in a fixed reference frame; r is the radius of the wheels, d is the distance between the two wheels and ω_r and ω_l are the angular velocity of the right and left wheels. $v = r(\omega_r + \omega_l)/2$ and $\omega = (\omega_r + \omega_l)/b$. In Figure 3.8, u_2 and u_1 respectively, which are the control inputs for the system. As a matter of fact it is possible to rewrite the dynamics of the system as

$$\dot{\xi} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = G(\xi) \cdot u = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

Given the non-holonomic motion of the robot, this implies a set of constraints as not all possible configurations are achievable from a given initial state. Therefore, we make use of an optimization at each control step through a nMPC.

The nMPC's strength is the optimization considering a finite time-horizon, the nMPC optimizes N steps ahead in the future based on a given model of the system, thus, anticipating the events and taking proper control actions; of the N control inputs computed we only make use of the control input for the current time-step. Then, the process is repeated at each modulation step after the generation of the desired motion.

The nMPC allows us to obtain better performances than a linear MPC, since it takes into account the full dynamics of the system. This implementation exploits the potential of nMPC to generate control inputs that are bounded but not saturated.

The reference trajectory is modeled as a virtual unicycle leader and described by a reference state vector $\xi_r(t)$ and a reference control signal $u_r(t)$.

In order to implement the nMPC we used the optimization software Casadi [Andersson J.A.E. et al 2019]. The main characteristics for setting the controller are the variables to be optimized, the state trajectory *X*, which has dimension 3 × N + 1, and the control inputs U , with dimension 2 × N .
For the current implementation we further define:

- Cost function *J*
- Continuous model of the system
- Discretization of the system function (using Runge-Kutta4)
- Sampling time *h*: time step used to discretize
- Limits of the system (soft and hard constraints)
- Gains of the function to minimize
- Number of prediction steps N

The cost function *J* to minimize is:

$$\sum_{i=1}^{N+1}(X_i - X_{ref})^T G_x (X_i - X_{ref}) + \sum_{i=1}^{N} U_i^T G_u U_i$$

with Gx and Gu diagonal matrices composed by the gains that we have to set, depending on the importance of the variable. X_{ref}, where X_{ref} is defined as the desired state of the robot given by the modulated DS.
Our system, which can be modelled as a unicycle robot, with state defined by $?\xi = [x, y, \theta]^T$ as described above by ξ^* . The discretization of the system is done following Runge-Kutta4 method for a *h* time step.
Then we defined the constraints on the state, on the inputs and the ones derived by the system's dynamics. We do not have any constraints on the state X, while we have limitations on the linear speed and acceleration and on the angular speed and acceleration. For the linear and angular acceleration we have to approximate: assuming a general acceleration constraint as $|u^*| < acc_{constraint}$, approximated as $\left|u_k - u_{k-1}\right| < h\, acc_{constraint}$, for a set of discrete steps (k), with a given time step (h).

In order to take account of the non-holonomic constraints, we consider the head position of the unicycle, which is the point that lies at a distance d along the perpendicular bisector of the wheel axis ahead of the robot. As well, we have to include the constraints given by the dynamics of the system, using the discrete function computed at each prediction step. We have to constrain the optimization variable X to X_0, which is the initial pose of the robot from the point of view of the nMPC. i.e., the pose of the robot at the time-step at which the nMPC is executed.
The constraints are summarized here to be:

$$-Inf < X < Inf \;\; and \;\; -speed\_constraints < U < speed\_constraints$$
$$-h \cdot acc\_constraints < U_1 - U_{prev} < h \cdot acc\_constraints$$
$$-h \cdot acc\_constraints < U_{i+1} - U_i < h \cdot acc\_constraints$$

The outputs of the nMPC are N control inputs. Generally, only the first inputs are used and then the optimization is computed again. In our case, we will use a number of control inputs equal to $N_{input} = F_{control} / F_{DS}$. Where $F_{control}$ is the control frequency, i.e. the frequency at which we control the motors of the robot, while F_{DS} is the frequency of the high level controller that gives us the reference pose for the nMPC. The overall controller architecture is shown in Figure 3.9 with the flow of sensing data, state of the robot, to the detailed modulation of the dynamical system and the optimization step with the nMPC.

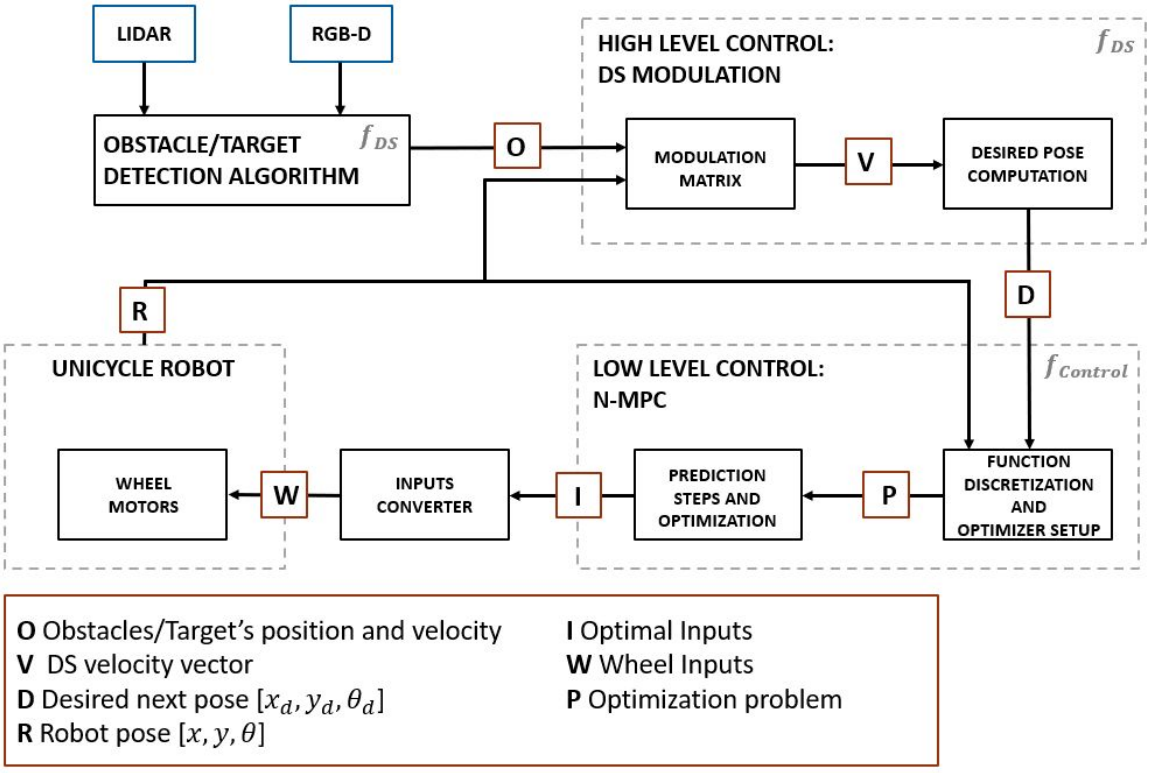


Figure 3.9: Control Diagram for MDS with nMPC for a non-holonomic robot.

Figure 3.10, shows an example of the MDS and the design output for the orientation of the robot given the online optimization through the nMPC. In the left side is a set of obstacles modulated as ellipses through the method presented in the previous section, and in the right side the output commands to the robot given the output of the nMPC controller.

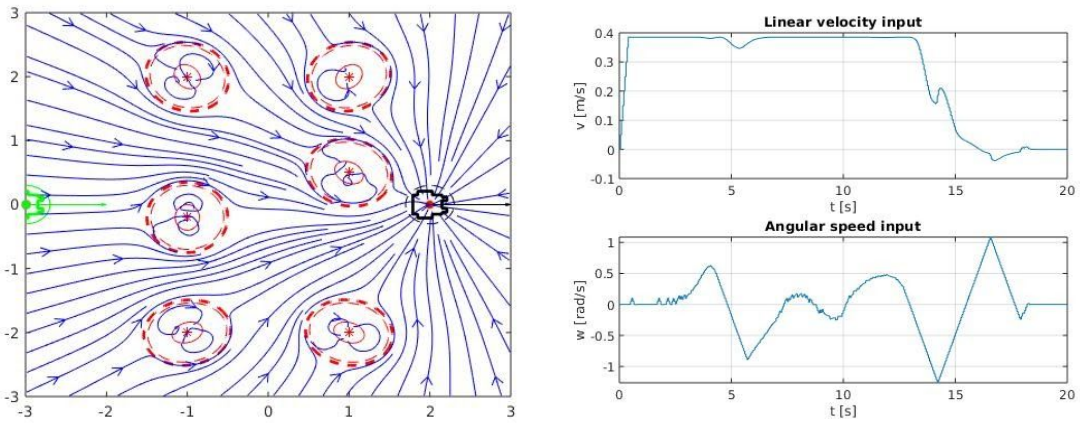


Figure 3.10: Modulation of a linear dynamical system considering multiple obstacles and the desired final state of the robot including orientation.

4. Redirecting Driver Support for Non-holonomic and Non-circular Robots

In case of a crowded environment (density of 1 person/m² or higher) and for a platform with an elongated shape (like Qolo), a circle is a rather conservative simplification of the robot's true shape. Further, the wheels induce a non-holonomic motion constraint. One can abstract from this constraint by controlling the bounding circle's center (if it is not on the wheel axle). However, the location of the controlled point determines which orientation the robot approaches when following a straight motion: it will move forward if the controlled point is in front of the wheel axle and backwards otherwise. Since forward motion is desirable, the controlled point needs to be in front of the wheel axle, and as it also represents the center of the bounding circle, this requires a greatly enlarged radius if the robot's shape extends mostly behind the wheel axle (as for Qolo).

Thus, a supplementary approach, the method termed Redirecting Driver Support (RDS) has been developed, which replicates the behaviour of the MDS locally. While it sacrifices the guarantee to reach the global goal (by itself, i.e. without an additional path planner), it allows representing more accurately the robot shape detail and non-holonomic constraints of robot kinematics.
The RDS approach has been designed to be functional when using raw laserscan measurements of objects and/or estimates from a tracking system (containing position and velocity information about objects). The control law computation requires geometric constructions and the solution of an optimization problem with characteristics like a linear program. We provide an efficient implementation which can be executed with a high control frequency on standard processors, as detailed in section 6.3, experimental analysis showed the current algorithm executed at 150Hz for a set of 900 constraints.
Conceptually, this development extends the Velocity Obstacle framework by applying it locally for each object and the closest corresponding part of the robot's non-circular shape. Thereby, it derives Cartesian velocity constraints for the robot's local parts, and transforms the constraints via the kinematic relations into a single optimization space. The optimization then finds within the admissible velocity set the command which is closest to the driver's command.
The resulting behaviour locally deflects the nominal robot velocity to slide along or around objects rather than colliding with them, and it can therefore complement a high-level motion planner or assist a human driver or operator. Preliminary experimental results and practical demonstrations have been presented in the deliverables D1.4 and D3.1. The following sections describe the technical details and theoretical formulation, as well as subsequent experimental evaluations.

4.1. Shapes Representation

The method uses a capsule for the robot and circles for surrounding objects to model their collision shapes, as shown in Figure. 4.1. It further constructs for each object the corresponding closest incircle in the robot's shape. As Figure. 4.2 shows, capsules are a shape class which can capture different types of intelligent vehicles.

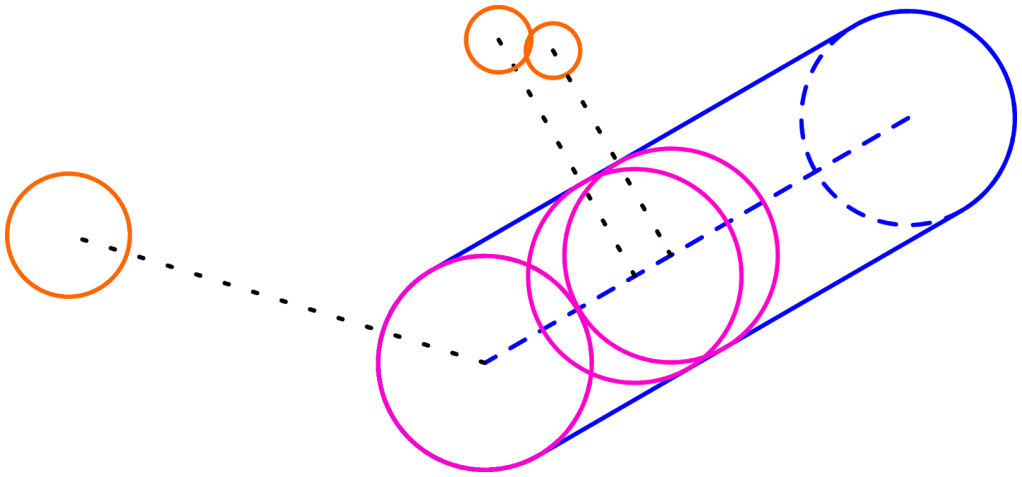


Figure 4.1: The RDS method represents the robot by a capsule (blue) and the surrounding objects by circles (orange) and constructs corresponding closest local in-circles on the robot (pink) for collision avoidance.

Figure 4.2: The top left picture shows the capsule approximation we use for Qolo, which is chosen rather conservative in order to provide a safety margin. The top right picture shows an example for an electric scooter. The bottom row shows all the research platforms for the CrowdBot project, whose footprint can also be described by a capsule.

4.2. Robot Kinematics

The method assumes that the robot's degree of freedom is reduced to two by the fact that its wheels prevent their axle from translating laterally, which imposes a non-holonomic motion constraint. Therefore, the velocity of any point that does not belong to the line through the wheel axle determines the entire body's velocity distribution. This relation between point velocities allows us to choose one point's velocity to represent the state of motion, and we term this point as the reference point. The method uses the reference

point's velocity as the optimization variable in the control computation. Into this optimization space, it maps linear velocity constraints that arise for other points' velocities due to velocity obstacles for the purpose of collision avoidance. It also maps into this space the nominal command, which is given by the nominal linear forward velocity and the nominal angular velocity and uses it as the target for the optimization.

4.3. Velocity Constraints

First, each object's velocity obstacle (VO) is constructed with respect to the corresponding closest incircle in the robot's shape. Figure 4.3 shows the construction for an example incircle and object with a non-zero velocity.

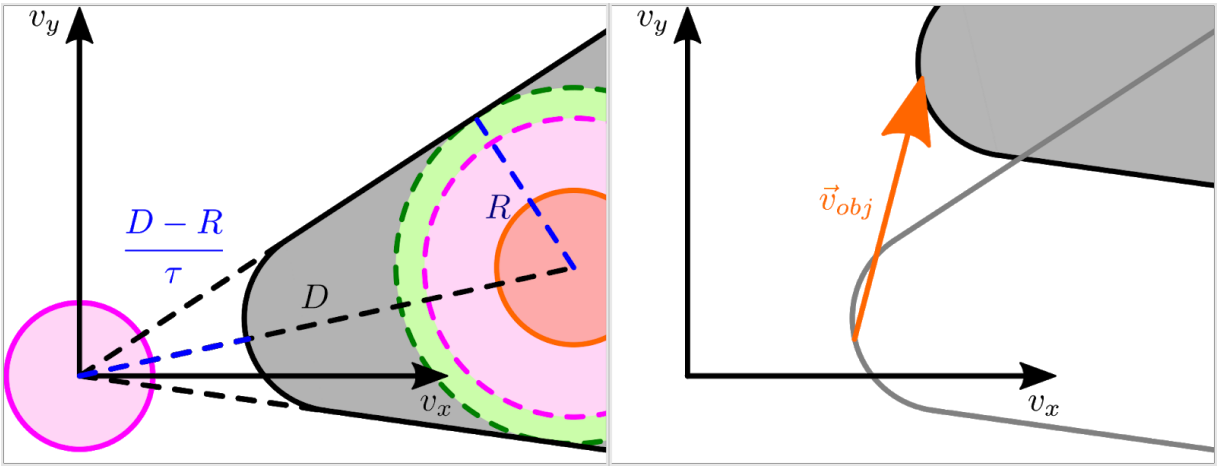


Figure 4.3: Constructing the velocity obstacle which an object (orange disk) induces for a robot part (pink disk). On the left, the object is enlarged by adding the robot radius and an additional margin (yielding the combined radius R). Tangent to the enlarged circle, a cone is drawn from the robot circle's center, truncated along the circle boundary, and scaled by the inverse time horizon τ to yield the relative velocity obstacle (grey). On the right, this is shifted by the object's velocity to yield the absolute velocity obstacle (grey filled).

As a second step, the method derives a linear velocity constraint from every VO for the corresponding local incircle's center point, as Figure 4.3 shows.

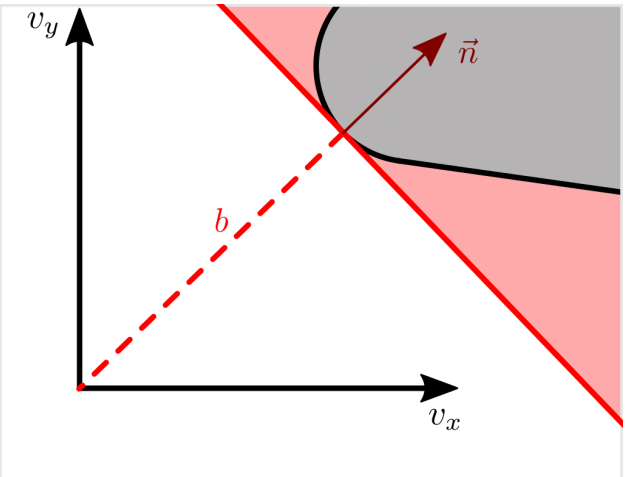


Figure 4.4: The linear approximation (red) for a velocity obstacle (grey) is derived by drawing the tangent at the point which is closest to the origin (in velocity space).

Next, only if the linear constraint renders the origin infeasible, its boundary is shifted to go through the origin. Finally, the linear velocity constraint that arises for each locally closest point per object is transformed into the reference point's velocity space.

4.4. Optimization

The method computes the linear and angular velocity command (v^*, ω^*) which the robot will execute by minimizing its deviation from a given nominal command $(\bar{v}, \bar{\omega})$ coming from the driver or any higher-level module. It achieves collision avoidance by imposing the above constraints on this optimization. The metric during the optimization is the Euclidean norm of the difference between the nominal velocity and the chosen velocity of the reference point. The nominal reference velocity $\bar{v}_x(\bar{v}, \bar{\omega})$, $\bar{v}_y(\bar{v}, \bar{\omega})$ is computed via the aforementioned kinematic relations. The optimization problem is formally written as

$$(v_x{}^*, v_y{}^*) = arg\ min_{v_x, v_y}(v_x - \bar{v}_x)^2 + (v_y - \bar{v}_y)^2$$

$$\text{s.t.} \qquad n_x^i v_x y^{R,i}/y^P + n_y^i v_x (x^P - x^{R,i})/y^P + n_y^i v_y \leq b_i, \quad \forall \quad i \in \{1, ..., N\}$$

where n_x^i, n_y^i denote for the i-th object the local linear constraint's normal vector, b_i denotes the corresponding offset from the origin (as shown in Figure 4.4). Further, $x^{R,i}, y^{R,i}$ denote the corresponding closest point on the robot, and x^P, y^P denote the reference point (both in local coordinates). From the solution, the actual velocity command for the robot is then computed as $v^*(v_x{}^*, v_y{}^*)$, $\omega^*(v_x{}^*, v_y{}^*)$ again according to the kinematic relations.

5. Post Contact Compliant Control

Considering post-collision handling of the robot control, we have as main objective to avoid that the robot freezes within a crowd, therefore, we investigate here a compliant control architecture based on an admittance behavior [Keemink, A. 2018] at the contact point, so that, we could react to collisions without the need to take the robot to a complete hold, which could lead to collisions with other moving agents. Initially, we are assuming a single collision per force/torque (F/T) sensor .

In Figure 5.1, we present the controller architecture considering that the robot is driven by a high-level controller that takes into account the current state of the robot and plans the desired trajectory, thus, given a set of desired velocities. In parallel we have a F/T sensor input with a low-pass filter removing potential noise (tuned to the robot motion profile and hardware), subsequently the signal is processed through a learnt model of the surface damping-stiffness profile which handles the nonlinearities of the transformation between the external contact surface and the point of sensing at the F/T sensor.

In order to control for the contact force at the collision location we transform both the collision force and the current state of the robot, then we introduce the controller dynamics at this point as:

$$
\begin{aligned}
M\ddot{\xi} + D\dot{\xi} &= F_c + F_u \\
\ddot{\xi} &= \frac{F_c - D\dot{\xi}}{M} + \frac{F_u}{M} \\
&= \frac{F_c - D\dot{\xi}}{M} + \ddot{\xi}_u
\end{aligned}
$$

where M is the virtual mass of the robot, D represents the damping coefficient, ξ^* the robot's velocity, F_u corresponds to the desired control input and F_c the external contact force.

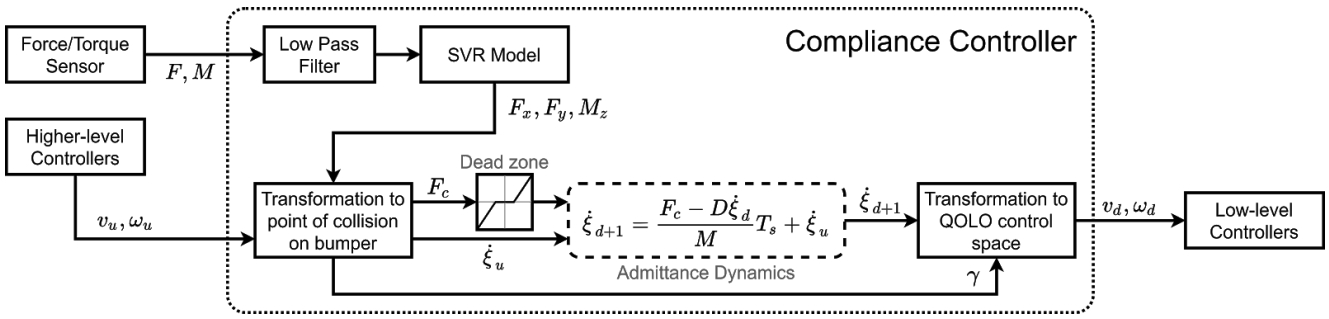


Figure 5.1: Control diagram post-collision compliance control on a mobile robot.

We consider the system to be controlled in the velocity space of the robot, therefore an integration step is required to control the output to the robot (first order discretization for a robot)

$$
\dot{\xi}_{d+1} = \frac{F_c - D\dot{\xi}_d}{M} T_s + \ddot{\xi}_u T_s + \dot{\xi}_d
$$

where T_s is a time constant for the discretization, then $\xi^{**}{}_u$ can be replaced as

$$\ddot{\xi}_u = \frac{\dot{\xi}_u - \dot{\xi}_d}{T_s}$$

Finally, the controller dynamics comes to be:

$$\dot{\xi}_{d+1} = \frac{F_c - D\dot{\xi}_d}{M} T_s + \dot{\xi}_u$$

where $\xi_{d+1}{}^*$ represents the output desired state, for a current state $\xi_d{}^*$ of the robot with a given input command $\xi_u{}^*$. The virtual mass M has been chosen as a function of the acceptable maximum contact ($F_{max} > 0$), so that, we can control directly for the desired limits in the interaction as,

$$M(\dot{\xi}_u) = \begin{cases} \frac{F_{max} T_s}{|\dot{\xi}_u|} & \text{if } |\dot{\xi}_u| > v_{min} \\ M_{max} & \text{otherwise} \end{cases}$$

which is state dependant of the current velocity at the collision point $(\xi_u{}^*)$, but limited to a maximum value of virtual mass for velocities below a set threshold (v_{min}) of the robot, which could be set freely based on the desired response in each robot.

Space transformation for non-holonomic constraints:

The transformation between the robot control space and the contact point (assuming a single contact) at a given point c over the surface of the bumper, at an angle (γ) from the center of the bumper (as displayed in the example Figure 5.2. We could control the effective velocity (ξ^*) at the point of collision perpendicular to the bumper surface by transforming this to the control space of the robot as follows,

$$O = \sqrt{(o \sin\gamma)^2 + (l + o\cos\gamma)^2}$$
$$\beta = \tan^{-1}\left(\frac{o\sin\gamma}{l + o\cos\gamma}\right)$$
$$\dot{\xi}_u = v_u \cos\gamma + \omega_u O \sin(\gamma - \beta)$$

where the distances o, l, O, angles β, and γ are depicted in Figure 5.2. And the velocities (v, ω) correspond to the control point of the robot at the center of the axis.

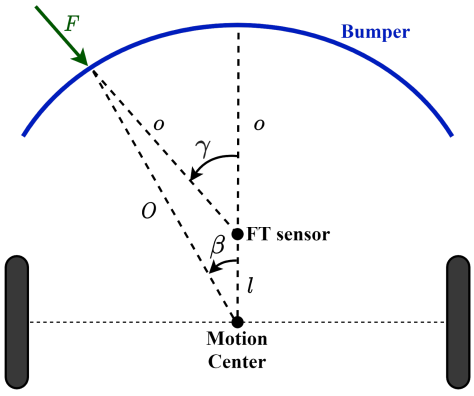


Figure 5.2: Contact force transformation for estimating contact location and magnitude with reference to the robot's control point.

Now, considering the compliant control scheme as explained above at the bumper surface, we propose a mapping that can control the non-holonomic constraint limits of the direction of motion of the robot for minimizing the contact force. First, the output is scaled with the robot's linear and angular velocity constraints (v_{max}, ω_{max}), using the following,

$$\dot{\xi}_d = v_d + \sin(\theta - \beta)\frac{\omega_d}{\omega_{MAX}} v_{MAX}$$

Here, we have an infinite number of solutions over the line that would satisfy this equation as depicted in Figure 5.3, given that we control a one dimensional contact force in the 2 dimensional control space of the robot, any point over the line segment would satisfy the required velocity command.

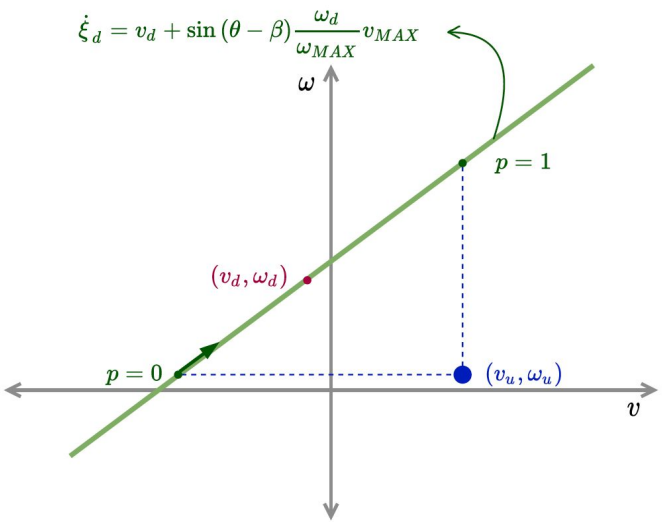


Figure 5.3: Transformation from the collision point space to the constrained nonholonomic output velocity for controlling the impact force response.

The parameter p is then defined as a function proportional to the contact angle γ,

$$p = |\gamma| \left(\frac{p_{max} - p_{min}}{\pi/2} \right) + p_{min}$$

where the p_{min} and p_{max} values could be in the range $[0\ 1]$ or it could be used as the intersection over the actual feasible space of solutions with acceleration constraints. Herewith, imposing a behaviour in the robot that would align the perpendicular axis of the wheels to the collision vector in order to cancel it, given that this would ensure the best controllability of such force.

The final transformation to the robot control space is given by,

$$v_d = p v_u + (1 - p)\left(\frac{\dot{\xi}_d - b\omega_u}{a} \right)$$

$$\omega_d = p\left(\frac{\dot{\xi}_d - a v_u}{b} \right) + (1 - p)\omega_u$$

where $a = 1$, and $b = sin(\theta - \beta)(v_{max}/\omega_{max})$ obtained from the kinematic distribution of the bumper and control point for the robot.

In summary, we find the possible transformations to the control space of the robot as a state dependent line, depicted in Figure 5.4. Four examples of the output behavior in the robot based on four initial states from a $\xi^* = [0, 0]$, $\xi^* = [0.5, 0]$, $\xi^* = [1.0, 0]$, and $\xi^* = [1.5, 0]$ are depicted with the output behavior based on the force of collision e.g., for the 50% of the maximum force the solution is found in the grey line. In these cases, the location on that line is defined by the parameter *p*, which is directly proportional to the impact angle over the bumper surface of the robot.

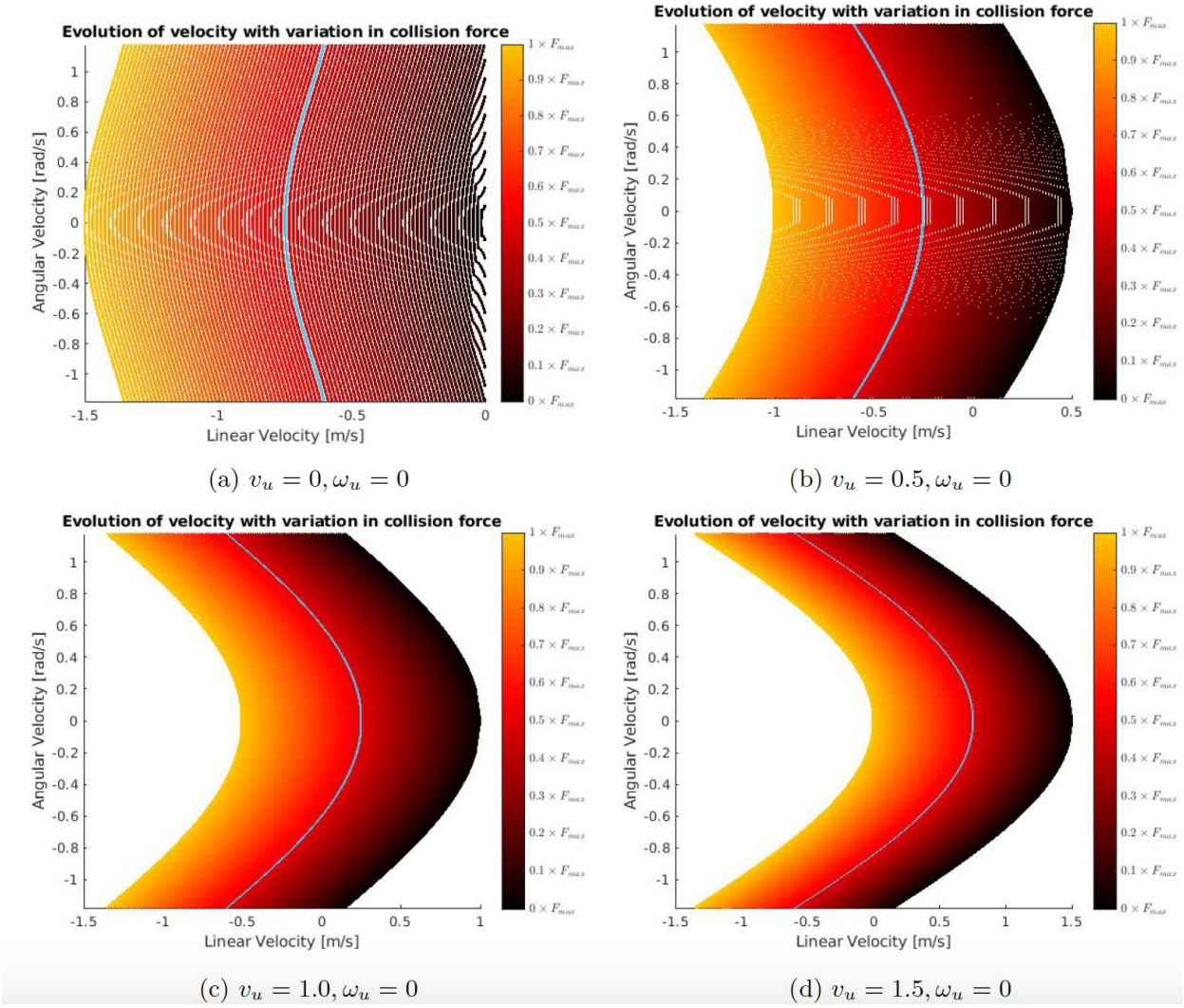


Figure 5.4: Control space transformation from a given impact force at a known location and its corresponding mapping to the non-holonomic space of the robot. e.g., the grey line in the center demarks the trade-off between linear and angular velocities where a collision with 50% of the maximum force would be found, for the four different cases of zero velocity to the maximum velocity.

Calibrating Bumper stiffness-damping effect:

Given the design of the bumper, its weight, structural constitution, and attachment to the rigid body of the robot it is expected that significant forces are absorbed by its flexible construction. However, accurate

measurement of the contact force becomes more challenging. To solve this, we proposed a training of the nonlinearities in the rigid body transformation for the bumper. To this end, actual collision forces were captured with an external F/T sensor as depicted below.

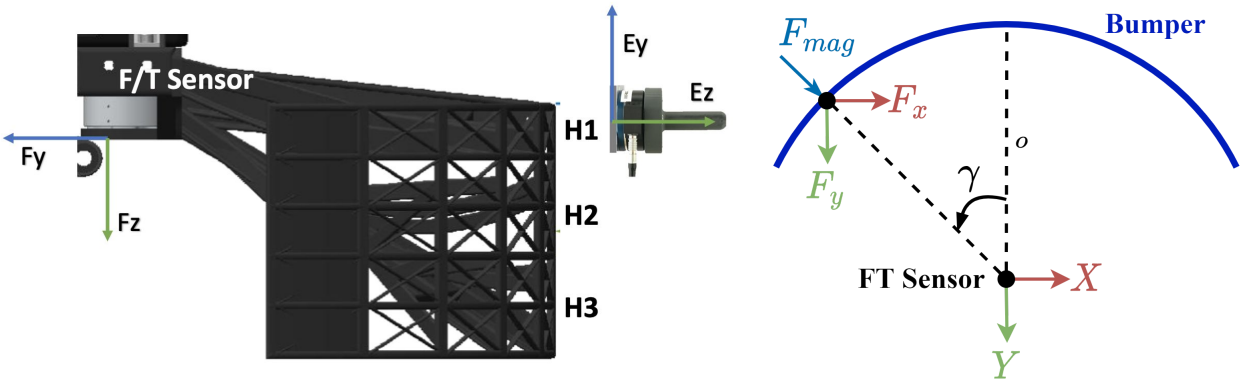


Figure 5.5: Bumper stiffness-damping calibration process. Left: lateral view of the bumper with the location of the F/T sensor in reference to the bumper. Right: top view diagram of contact forces.

Using a rigid body transformation one would obtain a significant error based on the actual location of the contact and the attachment of the bumper onboard, as depicted in Figure 5.5, where for a point (P1) with a small angle from the center of the axis ($\gamma = 15^{\circ}$) the error is small (bottom figure), whereas for a a further away point in the bumper (P3 at $\gamma = 45^{\circ}$) the nonlinearities render the error over 100%.

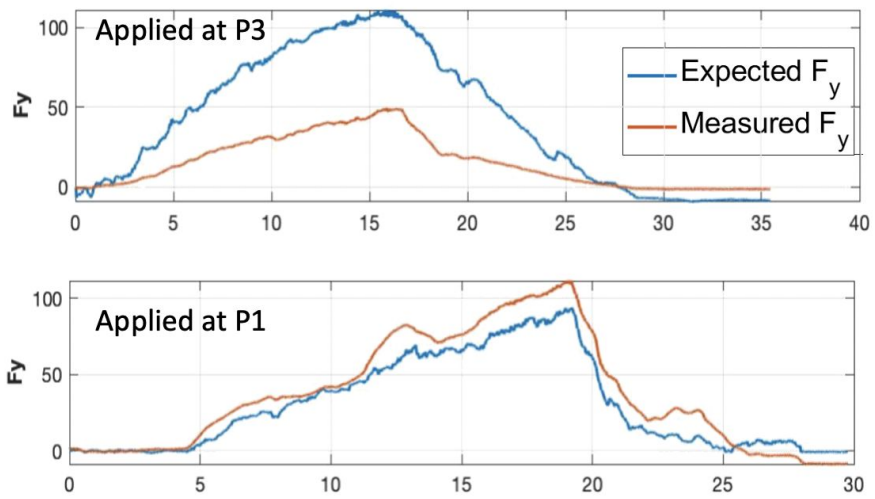


Figure 3.6.6: Sample data for contact force at the bumper perimeter for P1 which stands for γ = 15 □, and point P3 which stands for γ=45□, with a height H1.

To remove the effects of non-rigidity, and partial force transfer of the bumper from the measured values, a prediction model was developed based on a support vector regression (SVR) [Fan R.E. et al 2006], with the regression function $f(x)$ defined as,

$$f(x) = \langle w, \phi(x) \rangle + b = \sum_{i=1}^{m} (\alpha_i^* - \alpha_i) k(x_i, x) + b$$

where the linear function of w, b is a projection of the original input data into the feature space $\varphi(x)$ for dealing with the nonlinearity of the data, while keeping a simpler optimization problem of fitting the data. The prediction model is then dependent only on a set of support vectors $(\alpha_i^* - \alpha_i)$ for the given training set of m samples, a bias (b), and a kernel function. For this application we have chosen a Gaussian kernel function $k = e^{-|x_i - x_j|/2\sigma^2}$ with σ as a hyperparameter.

To train the SVR model, the set of known forces is applied to the bumper at various locations as shown in Figure 5.5, using a second FT sensor. Then, a one dimensional SVR is trained over the measured FT values from the onboard sensor for each of the needed data for estimating the contact force at the bumper surface Fx , Fy , and Mz. Using as predictors five axes of the F/T sensor $x = [Fx,\ Fy,\ Tx,\ Ty,\ Tz]'$, in here, we neglected Fz as the force parallel to gravity in the current setting is minimally affected by external contacts on the bumper.

***Assumptions for real time execution*:**

- The SVR model removes all non-rigidity effects from the sensor measurements, so we can consider the bumper as a rigid body.
- Only pure force is applied at the point of contact.

With the above assumptions, we can estimate γ from Fx , Fy , and Mz . The solution for force magnitude and contact angle is taken as,

$$F_{mag} = F_x \sin\gamma + F_y \cos\gamma$$

$$\begin{aligned} M_z &= F_x r \cos\gamma - F_y r \sin\gamma \\ \frac{M_z}{r} &= F_x \left(\frac{e^{i\gamma} + e^{-i\gamma}}{2} \right) + iF_y \left(\frac{e^{i\gamma} - e^{-i\gamma}}{2} \right) \\ \gamma &= -i \log \left(\frac{\frac{M_z}{r} + i\sqrt{F_x^2 + F_y^2 - \left(\frac{M_z}{r}\right)^2}}{F_x + iF_y} \right) \end{aligned}$$

where the $F_x,\ F_y$, M_z are the estimated values from the output of the trained SVR. Note that Fx and Fy at the contact point and at the sensor are the same.

By discretizing the bumper into 27 contact points, we gathered contact data in 2 sets with different maximum collision forces (0~100N) and (0~200N). Each collision sample contains 30 seconds of data recorded at 400Hz. The total dataset contains 129.6K samples for each dimension in both sensors (Fx, Fy, Fz, Mx, My, Mz).

The resulting SVR function for each of the dimensions of interests (Fx, Fy, Tz) contains 30.8K,14.7K, and 29.9K support vectors, respectively.

Figure 5.6, shows a set of errors over the surface of the bumper for each of the three trained models, for a set of examples of the training dataset with the mean and standard deviation for each of the contact points.

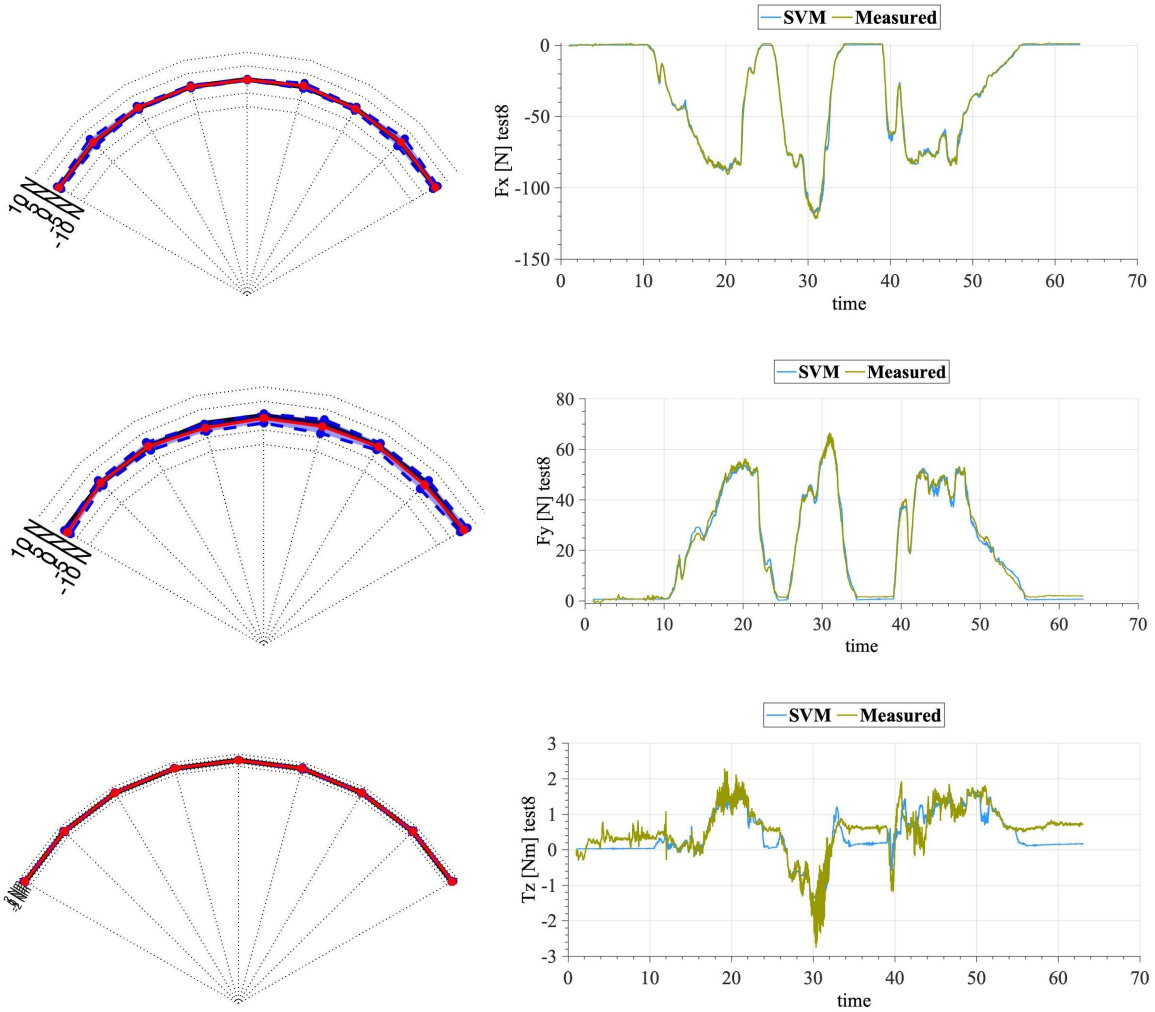


Figure 5.6: Error plot of estimated forces distributed over the bumper. From top down: Fx, Fy, and Mz, on the left with zero error at the black line, and standard deviation in blue. On the right, examples of the temporal response for the three trained models.

6. Experimental Evaluation

6.1. Robots used for Implementation:

6.1.1. Omnidirectional Robot: Ridgeback

The Ridgeback is an omnidirectional robotic platform produced and sold by Clearpath Robotics. Thanks to its dimensions (960 x 793 x 296 mm) it can be used in classic indoor environments (houses, offices, industrial and commercial buildings) as it can travel through standard 80 cm wide doors. Since the Ridgeback is holonomic thanks to its four Mecanum wheels with two controllable degrees of freedom of translation in the horizontal plane and one controllable DOF of rotation over the vertical axis, there is no problem to follow directly a (x^*, y^*, θ^*) velocity command that would be generated by the MDS obstacle avoidance algorithm.

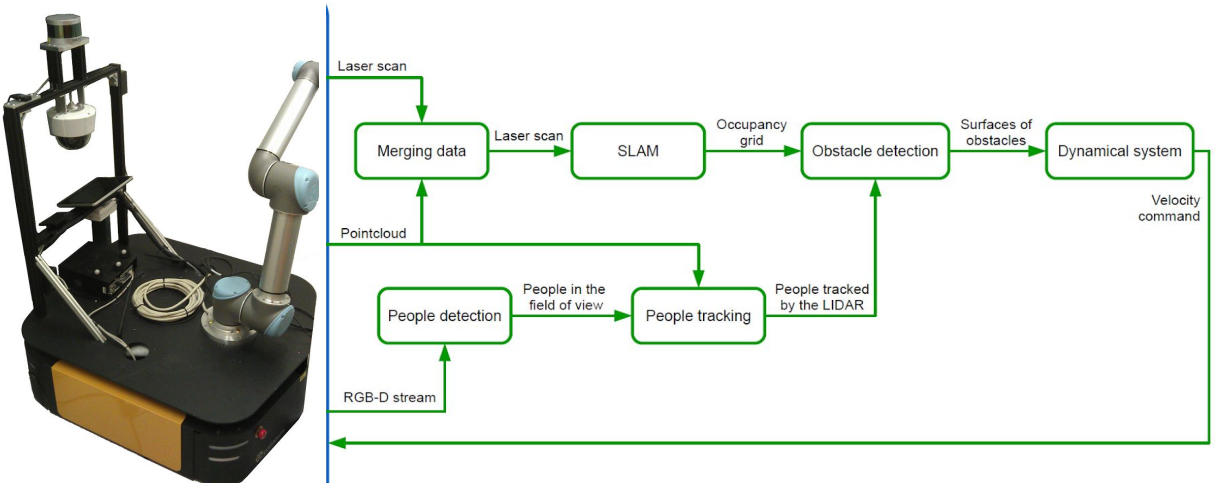


Figure 6.1.1 Ridgeback robot [Clearpath robotics] used for omnidirectional validation of the modulated dynamical system, and the sensing-control loop implemented.

In terms of sensors, a Velodyne LIDAR and two Hokuyo laser range finders are installed on the Ridgeback platform. An additional Realsense camera was installed for people tracking. Each Hokuyo sensor provides a planar 270-wide scan of the surroundings. As those sensors are set up back-to-back, there is no dead angle and a 360 scan can be obtained by merging data emitted by the two sensors, although they only scan in a single horizontal plane. Contrary to the Velodyne LIDAR which provides a 3D point cloud of the surroundings (with two dead areas respectively above and under the sensor). In Figure 6.1.1, the process for creating obstacle representation with this robot is depicted.

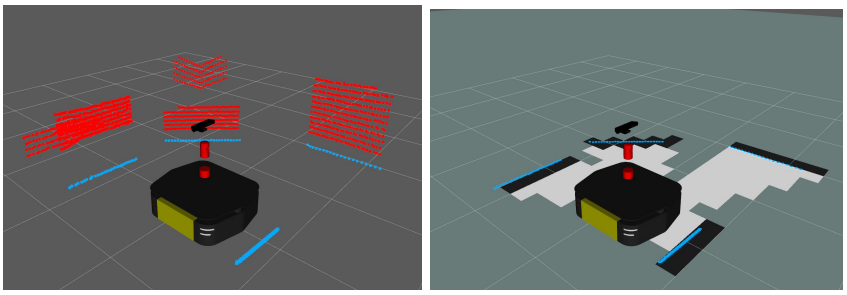


Figure 6.1.2: Mapping process for obstacle definition from onboard sensing data on the ridgeback.

A summary of the collected sensors and algorithm execution is presented in Figure 6.1.3, which has developed for online evaluation and data recording.

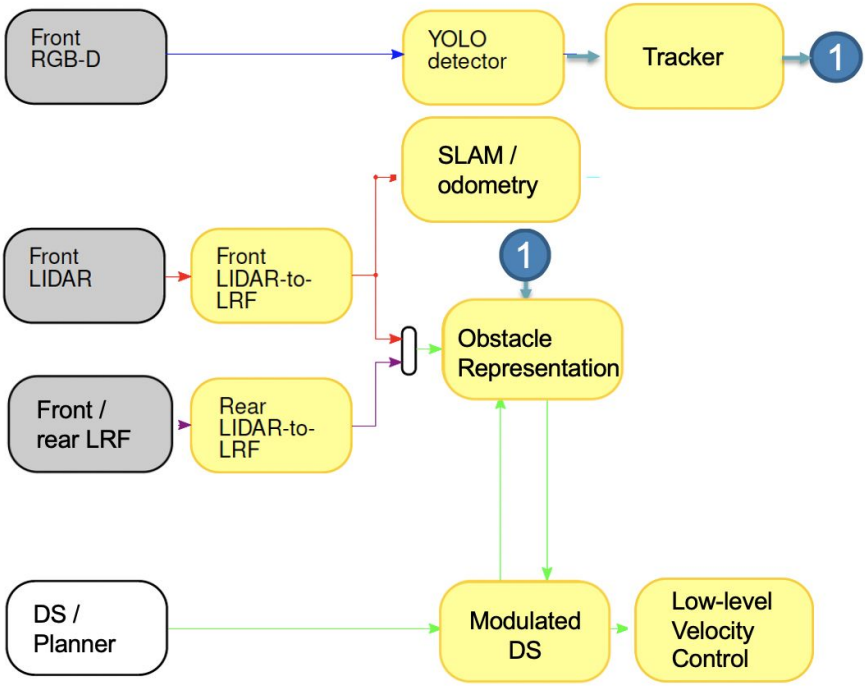


Figure 6.1.3 Data pipeline for the tests on the ridgeback robot.

6.1.2. Qolo Robot and Assistive Control

In this section we introduce the robot Qolo, as the platform chosen for evaluation in real-crowd motions. This robot's kinematics is a common 2 wheeled powered device, as any powered wheelchair, or 4 wheel mobile device, or scooter. This design presents new challenges for real-time navigation around crowds as it is non-holonomic, therefore, it has less freedom to move in the space.

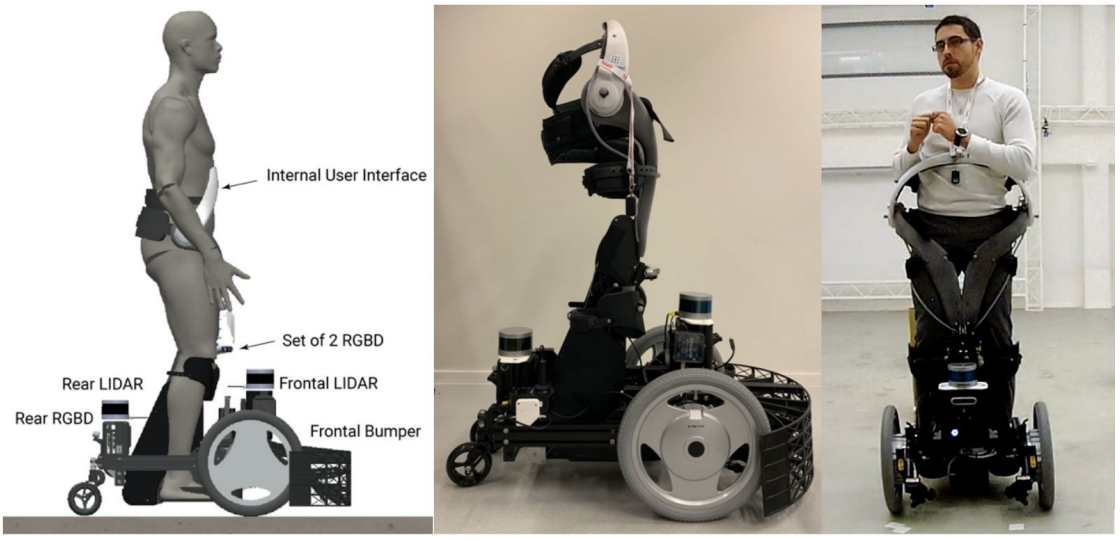


Figure 6.1.4. Qolo robot used in CrowdBot project. Left: CAD of the robot with sensors, middle: robot profile with a full set of sensors installed, right: user driving the robot.

The Qolo robot (depicted in Figure 6.1.4) developed by the University of Tsukuba is very much like a powered wheelchair, as it accomplishes the same task of transporting a person with lower-body impairment. such as spinal cord injury (SCI), while allowing the usage of their remaining motion capabilities in the upper body. In contrast to standing wheelchairs, no external power source is required for achieving the sit-to-stand or stand-to-sit transition [Paez-Granados D. et al 2018].

Another important consideration for this robot is motion constraint which plays a significant role where acceleration limits are strict for ensuring the safety of the user (preventing tip-over), similar to that of the robot Pepper. Moreover, the user interface and potentially the shared-control strategy are different in type and level of autonomy required. Some examples of these types of vehicles which are becoming popular worldwide as a last-mile mobility solution and short-mid distance commute proposals are standing scooters or smart scooters.

In the subsequent section, we describe the hardware, and control architecture developed for the CrowdBot project.

6.1.2.1. Hardware Architecture:

The Qolo robot for the project follows the sensor recommendations given in CrowdBot Deliverable D6.2 Robot Design Recommendations.

There are three Intel RealSense D435, as well as, two 3D Lidar sensors Velodyne VLP16 are installed in the robot. Location of the RGBD sensors could be changed accordingly to the desired testing scenario.

Currently, two cameras are set forward and one looking backwards at the rear of the robot, as depicted in Figure 6.1.4.

Qolo as a standing mobility device for spinal cord injury carries a person in the middle of the robot, therefore, it is necessary to use 2 lidars mounted in front and rear of the robot, as proposed in D6.2, for gaining full 360 view of the surroundings. The two Intel RealSense D435 cameras located at the front are placed 65cm above ground to give a combined field of view (FoV) of the sensors' RGB (cameras) of approximately 138 degrees of the frontal area, as depicted in Figure 3.2, left side (blue FoV). While point cloud information from infrared sensors could provide up to 170 degrees of frontal FoV, as shown in the right side (red FoV) in Figure 6.1.5. As well the rear camera set at 35 cm above the ground can be used for detecting incoming people or for using the point cloud proximity data directly behind the robot, with 68 degrees for camera FoV, and 89 degrees for proximity data.

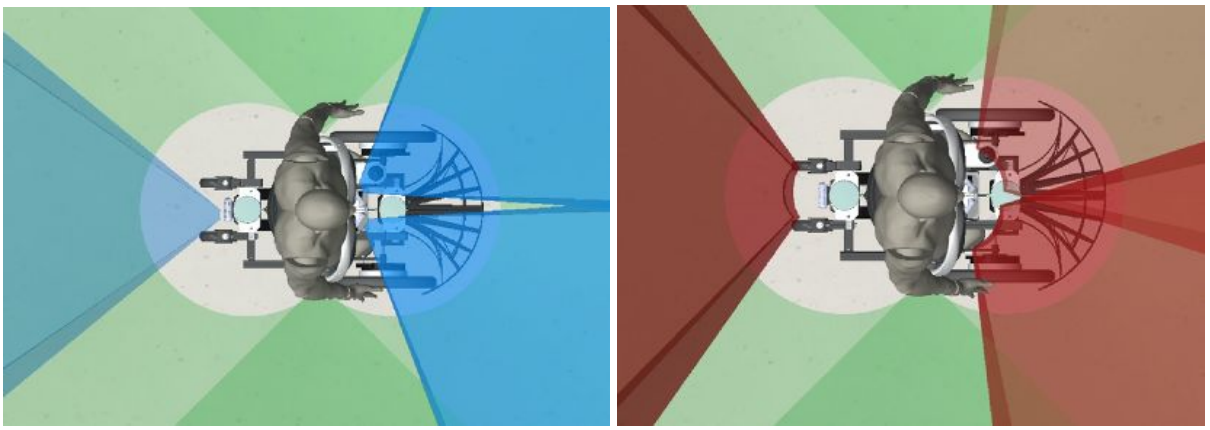


Figure 6.1.5. Sensors location in the robot Qolo, with frontal and rear RealSense D435 (blue and red) field of view, and front and rear Lidar in green.

Moreover, we have included a frontal FT sensor (Botasys Rokubi 2.1) for contact detection attached to a bumper, which could be used for detecting small contact with feet or other surfaces, and potentially controlling for compliance in such scenarios. As a passenger carrier robot Qolo's control interface is designed for forward motion, therefore, the sensor is placed in front of the robot with a protective bumper attached to it. The overall bumper weight is 1 Kg. Thus, for avoiding overload of the FT sensor (maximum measurement of impact force is 1KN, and torques up to 10 Nm), the bumper is placed with partial support to the main frame of the robot, as depicted in Figure 6.1.6

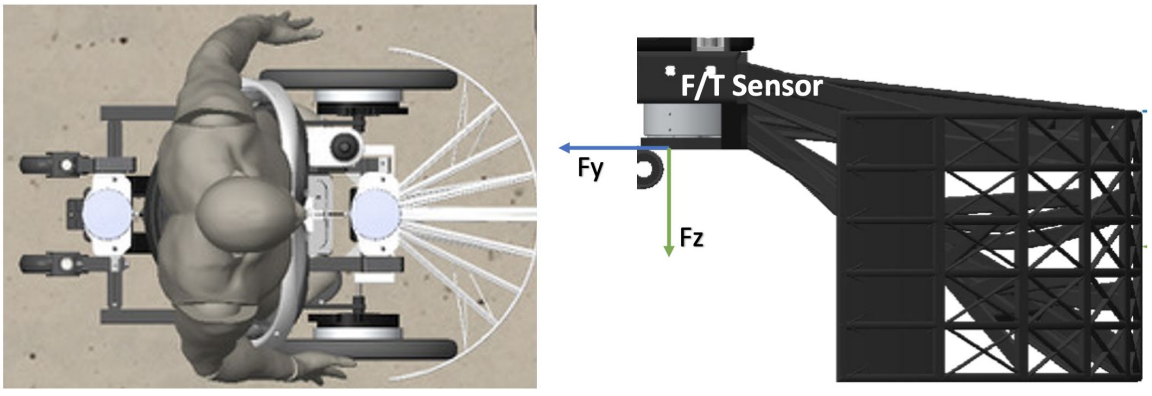


Figure 6.1.6. FT sensor mounting and protective bumper around the robot with 33 cm in diameter and 20 cm in height, placed at 5 cm from the ground.

6.1.2.2. Control Architecture:

We have followed a modular approach for distributing the computing power as proposed for the CrowdBot project robots where all the components of the control have been constructed with a ROS communication architecture so that it would be compatible with all the partners' developed modules.

As depicted in Figure 6.7, the controller of the robot is executed from an embedded computer UpBoard Squared (intel Celeron 2.4Ghz), integrated within the exoskeleton structure, which handles low-level control and user interface inputs. Internal velocity control is handled between an embedded microcontroller and in-wheel motor driver.

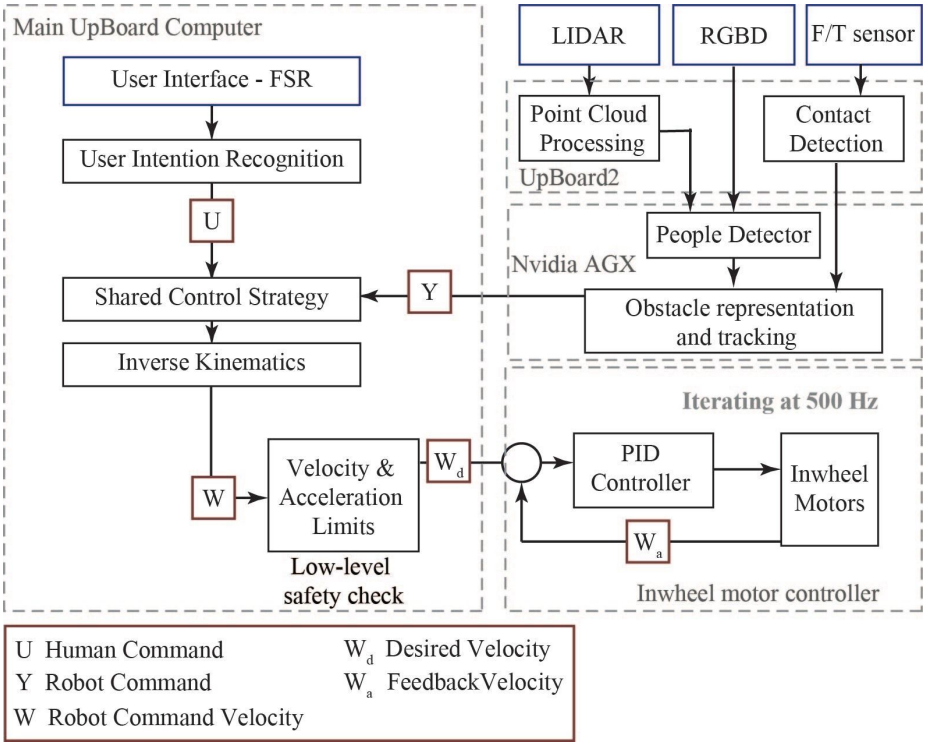


Figure 6.1.7: Overall control distribution with sensors, computing units and actuators.

For the CrowdBot project, we have modified the standard version of Qolo by integrating sensing and computing capabilities for onboard autonomous control of the robot. For the integration of the lidars, we have set a second UpBoard computer which can run independently of the low-level controller for a distributed control architecture. This second computer is embedded in the lidar's base for a compact configuration, equally an Nvidia Jetson AGX is installed in the rear of the robot with its own compact power source, for processing up to 2 RGBD sensing information and people tracking.

6.1.2.3. Shared Control:

Qolo as a standing mobility device aims to give hands-free control for people with mobility in their upper limbs (spinal cord injury up to the neurological level T4), to this end, we have developed an internal array of pressure sensors as a torso control interface. From this control interface, we use an intention recognition function that fits the input from the set of pressure sensors in the T-bar of the device and infers a desired linear [v] and angular velocity [w] of the user as control input [U], which is limited to the velocity of standard wheelchairs 1.5 m/s [Chen Y, et al. 2020].

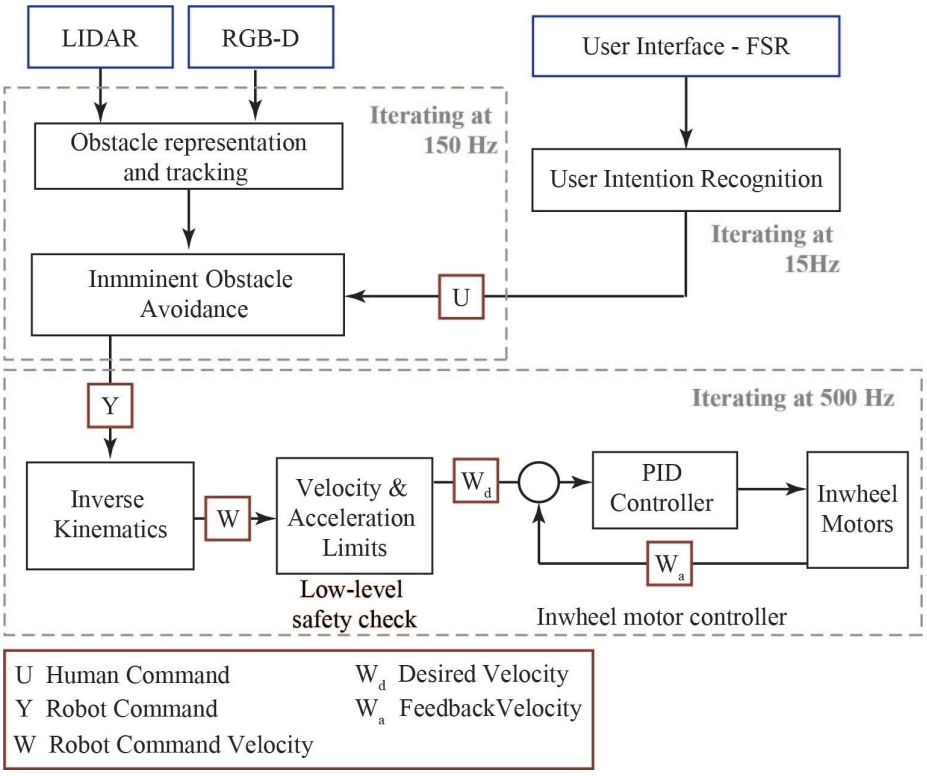


Figure 6.1.8: Diagram of control for the Qolo robot.

Figure 6.1.8 summarizes the shared control architecture, where the user input is taken from the intention recognition algorithm and sets a desired motion [U]. The reactive navigation generates an output that would be consistent with avoiding obstacles [Y] while keeping the closest direction of motion to the one given by the user .

In the low-level control the robot command [Y] is taken and transformed by the inverse kinematics of the robot to the motors command [W], which is then checked for acceleration and velocity limitations of the actuators with a sigmoid function in case of exceeding limits. Finally the output [Wd] sets a desired velocity for the internal velocity control of the in-wheel motors.

6.1.3. Data Collection:

For the current experiments the whole pipeline of sensing, detection, tracking, control and actuation is described in Figure 6.1.9. As well, the same architecture reflects the information recorded, as each yellow box represents a communication node in the ROS architecture.

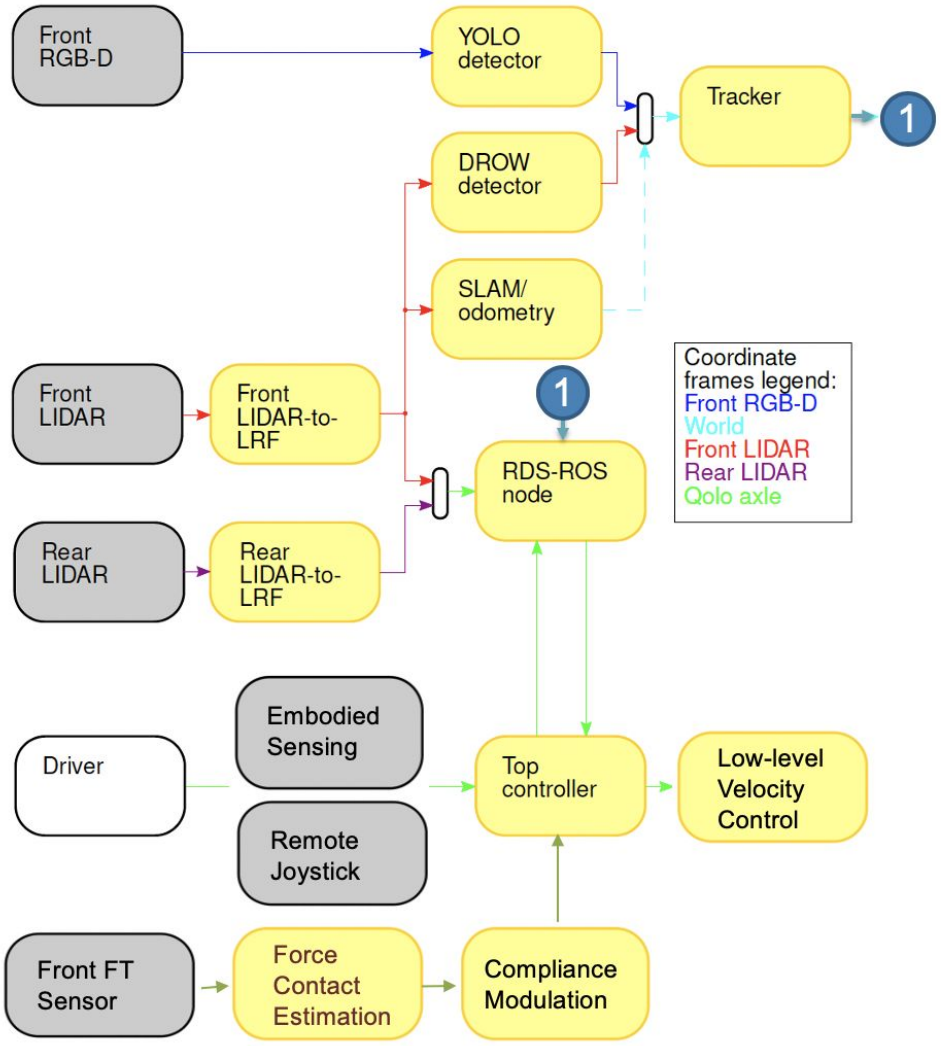


Figure 6.1.9: Communication flow of the robot's sensing, perception, control and actuation signals for the entire pipeline of processes in the robot.

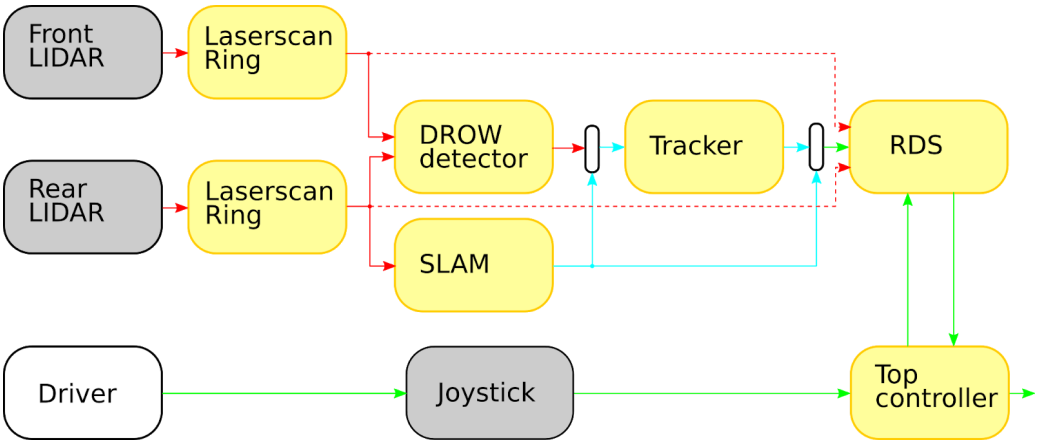


Figure 6.1.10 Example of the pipeline used for algorithm evaluation, reducing to the required minimum components.

For each independent evaluation of the system specific sensor inputs and detection layers are considered or not for control purposes according to the objective of the experiment. Herewith, validating different applications of the algorithm and demonstrating its flexibility in terms of hardware, and sensing capabilities.

6.2. MDS Simulation Experiments

6.2.1. Simulation for a Non-holonomic robot with non-linear Model Predictive Control

For this assessment we chose a set of moving obstacles (pedestrians) randomly located in a linear dynamical system pointing towards an attractor located at [4,5] from the robot's initial location, as shown in Figure 6.2.3. The blue lines denote the vector field output from all possible locations within the simulated space, the modulated dynamical system of a linear DS pointing towards the attractor at (0,2).

For testing we have selected the actual robot parameters of Qolo, so that we could observe what could be the output behavior in a real situation. The parameters are as follows:

- Velocity constraints: $U_{max} = [1.5\ ,\ 3.0]$
- Acceleration Constraints: $A_{max} = [2.0\ ,\ 7.0]$
- Control frequency: $F_{control} = 20\ Hz$
- DS frequency: $F_{DS} = 20\ Hz$

A first evaluation is shown in Figure 6.2.3, with 4 dynamic obstacles emulating pedestrians, and setting a minimum proximity to the boundary of the pedestrians to be 0.2 m, thus effectively enlarging their representation.

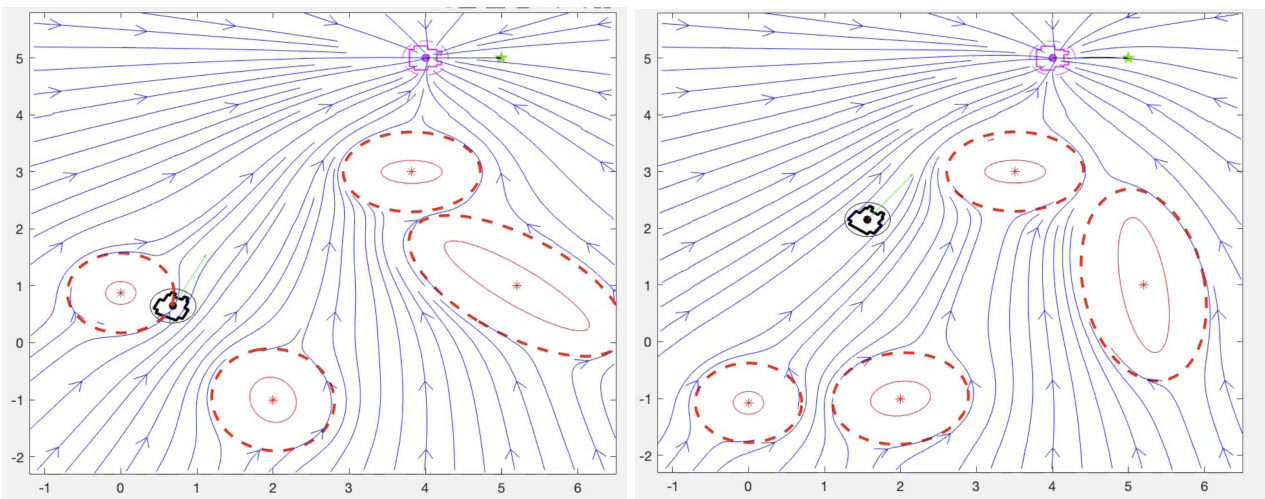


Figure 6.2.3: Modulation of a linear dynamical system considering multiple obstacles and the desired final state of the robot including orientation.

The results of the modulated velocity of the robot applied in simulation as in Figure 6.2.4 displays the trajectory followed by the robot and the difference in angular velocity from the desired one, which we conclude to be a consequence of the simulation parameters which try to follow real robot simulation with limits in acceleration and control delay thus such error would be induced.

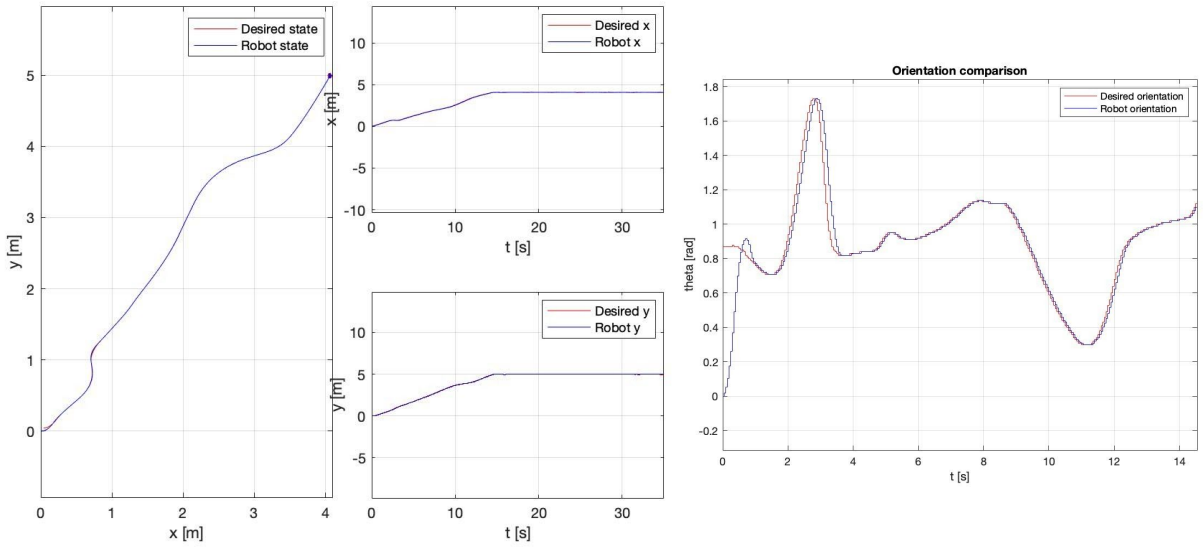


Figure 6.2.4: Example of MDS-nMPC control for a non-holonomic robot given the previous scenario. Left: the trajectory performed, right: desired vs. executed orientation.

The results in this simulation of the real system implementation with constraints to the real values for controlling a nonholonomic robot display clearly that the method can handle dynamic obstacles without complete knowledge of their motion, rather by using the update given in position. However, some error in the orientation control is still found as some situations might find it infeasible to steer the robot as desired, which is true for any robot in real implementations. This result clearly states the real limitations of the robot to execute some trajectories, thus, help to design and understand its applicability.

6.3. MDS testing on Omnidirectional Robot

The performance of the proposed framework is evaluated on a mobile robotic platform (Ridgeback by Clearpath Robotics). Thanks to its Mecanum wheels, this platform is omnidirectional and can therefore follow any velocity command that would be generated by the algorithm regardless of its current orientation. As the motion planner does not work directly with 2D shapes but only with 2D points, it requires a [x,y] position as its input. The position of the center of the platform is used as a reference to locate the robot in the horizontal plane. As the Ridgeback has a rectangular shape with a half-diagonal of roughly 56 cm and to have a small security margin, the platform can be enclosed in a circle with a radius of 60 cm. To avoid collisions with obstacles even if the robot is only considered as a point, obstacles are therefore expanded by 60 cm in all directions, as shown in Figure 6.3.1.

In all scenarios, the robot starts with no prior knowledge about its environment. To get information about its surroundings, it uses two Hokuyo laser rangefinders (one at the front, one at the rear) and a Velodyne VLP-16 LIDAR. Data coming from these sensors is merged into a single planar scan of the surroundings. Sensor handling and processing is done with the Robot Operating System (ROS) that runs on the on-board computer [quigley 2009]. These scans are used by a Simultaneous Localisation and Mapping (SLAM) algorithm which outputs a two-dimensional occupancy grid of the environment.

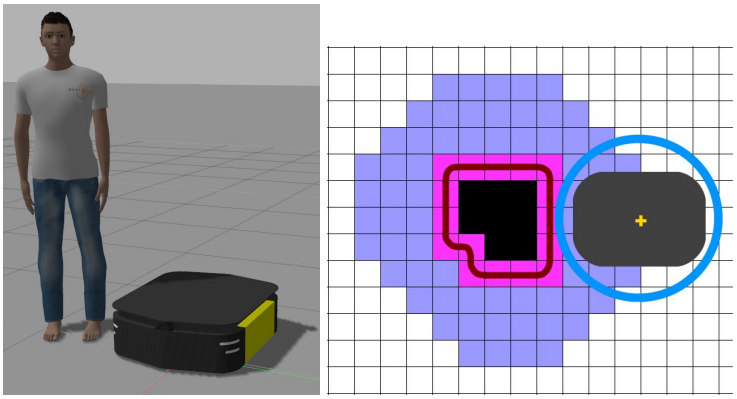
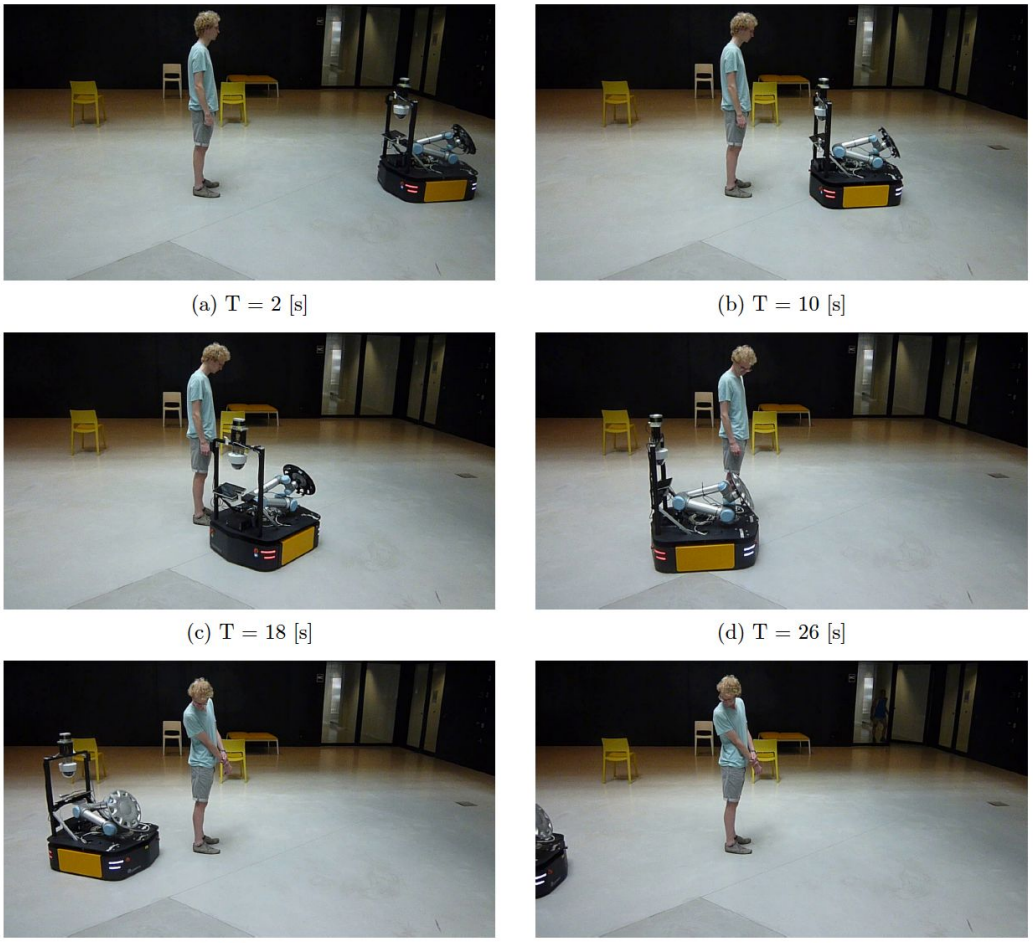


Figure 6.3.1: Implementation for the omnidirectional robot Ridgeback through coupling with standard grid maps and the robot representation.

The experiments test the real-time avoidance of static and dynamic obstacles. We get closer to a real world application by introducing several obstacles in the environment of the robot as well as moving pedestrians. Pedestrians are detected with an Intel Realsense Depth Camera D435 installed at the front of the Ridgeback and informing a YOLO object detector [Redmon et al., 2016]. To ensure continuous tracking and position estimation once people have been detected, they are tracked with LIDAR data and a Kalman filter if they get out of the field of view of the camera. This detection runs at 7 Hz with a Tiny YOLO v1 object detector and an $Intel\ Movidius^{TM}\ \ Myriad^{TM}$ 2 2450 VPU, an accelerator for neural network workloads.

From the SLAM algorithm the occupancy grid is updated and as described in section 3.4 a continuous shape of the obstacles is generated for usage in the modulation of the dynamical system.

In the current experiment, the objective was to assess how well the proposed framework can handle moving obstacles even if their velocity is not taken into account. At each time-step all obstacles are considered uniquely in the control loop, herewith, evaluating the reactivity of the algorithm to the changes in the environment perception.



(a) T = 2 [s]

(b) T = 10 [s]

(c) T = 18 [s]

(d) T = 26 [s]

(e) T = 34 [s]

(f) T = 42 [s]

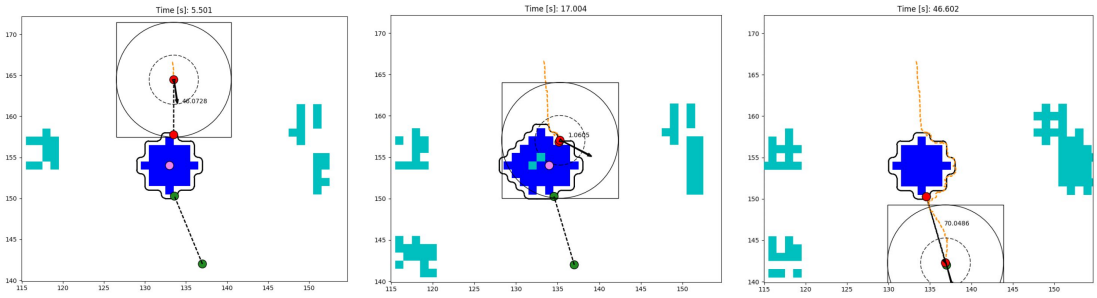


Figure 6.3.2: Snapshots of the experiment with a single pedestrian detection and avoidance.

Figure 6.3.2, shows an initial test with a single pedestrian for demonstration of the proper functioning of the entire pipeline of detection, tracking, and control. In this experiment, the robot is driven be a simple dynamical system towards the attractor (green dot demarcated in the bottom of figure, behind the pedestrian), as a proof of concept of the reactive behavior, the robot simply reacts to the obstacles in its path accordingly to a set safety margin. i.e., the only given knowledge to the robot is its surroundings within 5 m.

In a subsequent test we evaluate the fusion of obstacles and the robot's response to abrupt changes in the scene, with two pedestrians. While the original flow leads between the obstacles (Figure 6.3.3, top left), after this gap closes (Figure 6.3.3, bottom left), the newly computed flow leads out of and around the concave

obstacle formed by the two pedestrians and the static obstacles (Figure 6.3.3 bottom left), however, as the pedestrians move away (Figure 6.3.3 bottom right), the the open path is immediately taken between the pedestrians.

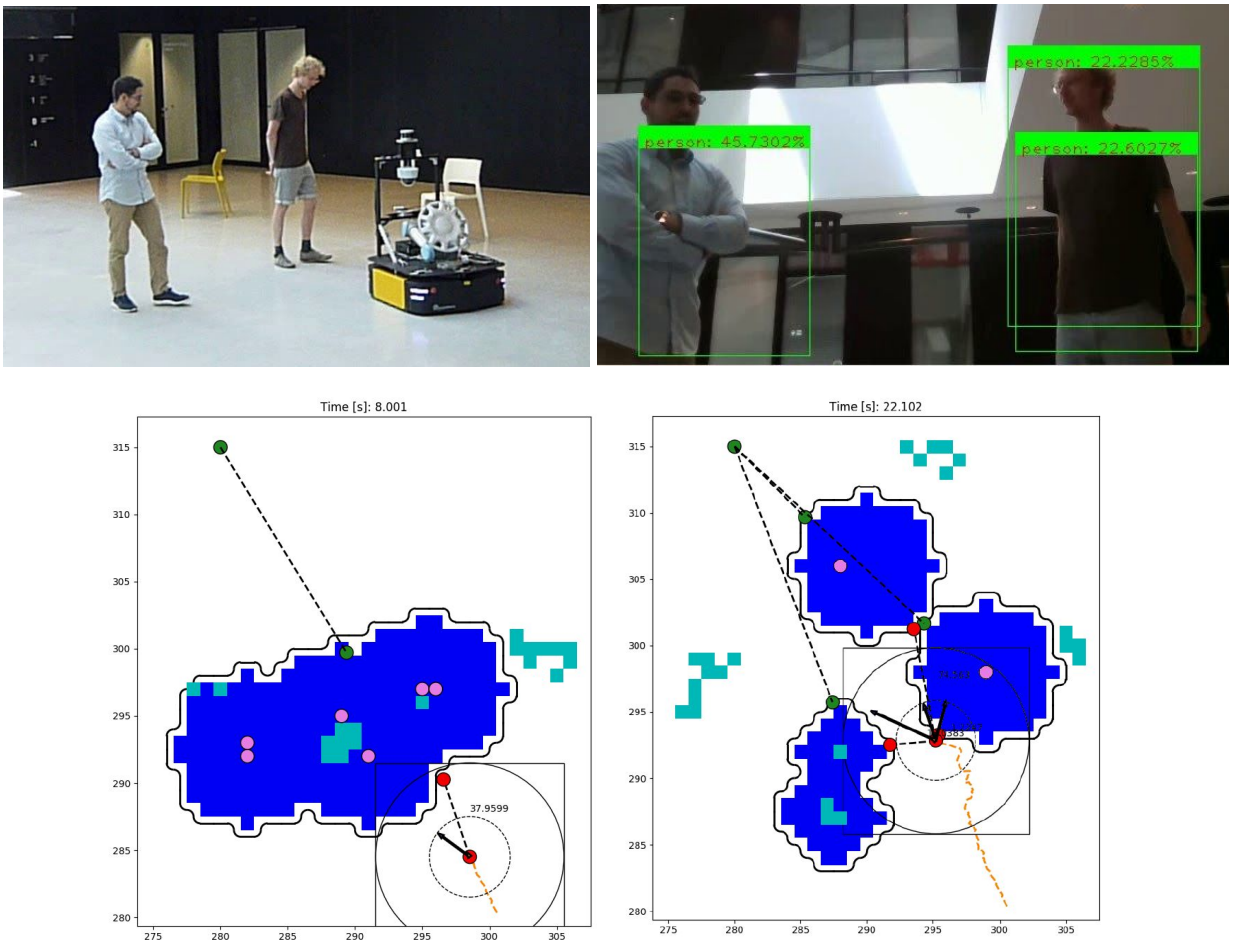


Figure 6.3.3: left: external view of the experiment setup with 2 pedestrians only, right: view from the robot's camera, and bottom: online representation of the current perception from the robot's local frame.

As the frequency of the control loop is limited, in real-time experiments with sensing and actuation limits the robot may cross the boundary of obstacles if they move faster than the robot's sensing and actuation latency. It could also happen by the uncertainty of a SLAM algorithm which could render sudden changes in the map in unknown areas. To solve any possible entry of the virtual space of an obstacle, a practical solution was implemented as a repulsive velocity field which is not part of the main algorithm (green stream lines).

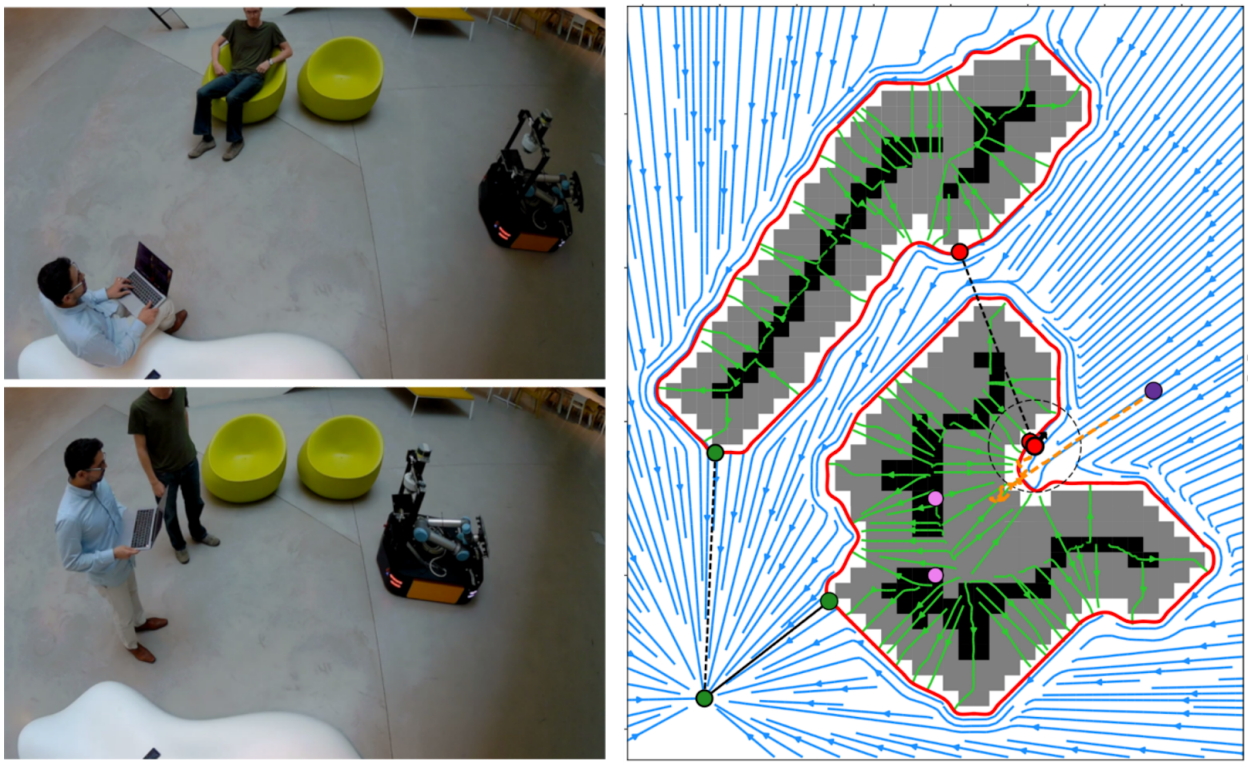


Figure 6.3.4: In this experiment, the robot adapts to dynamic obstacles, namely two pedestrians who block its path and follows a new flow leading around the composite obstacle (on the right).

Figure 6.3.4 shows the experimental setup map and resulting velocity field leading towards an attractor, the robot (red dot with an arrow) starts inside the non-star-shaped obstacle (purple dots) and gradually converges to the attractor (green dot at the bottom left) by following the modulated velocity field (blue stream lines). The experiments demonstrate that the implementation can respond to and avoid such dynamic obstacles (in addition to static obstacles) in real-time and converge to the desired position.

Through this experiment we validate the work of the algorithm in real-time robot operation following a higher-level set of commands created in this case by a simple linear dynamical system moving towards an attractor, but which could be a path planner. In here, we validated that random motions from the pedestrians would immediately be accounted for by the modulation of the entire vector field. Moreover, the space transformation for a concave (non-star shape) obstacle was shown to work as designed.

As well, the results showed at some points the robot could enter the virtual space of the obstacles especially in very close space where the reactivity is limited to the sensor's frequency. This is an important consideration for real operation of mobile robots in crowded environments, as it calls for known operation conditions of the surroundings in order to understand what sensing capabilities the robot must have to operate on it. For instance, the sensing and actuation latency of the robot to operate in a human environment should be lower than the human responsiveness frequency, and the operational velocity could be limited in accordance.

6.4. Redirecting Driver Support Experiments in the CrowdBot-simulator

The RDS method has been implemented on the model of Qolo in the CrowdBot-simulator, to enable Qolo to react to and avoid collisions with surrounding pedestrians. This section presents results of 1D-flow experiments with different densities and desired robot speeds. Example snapshots from these experiments in the simulator are shown in the figure below.

Figure 6.4.1 Snapshots from the 1D-flow experiments with Qolo using the RDS method. The pictures in the top row show the lowest density (0.2 p/m²) and the pictures in the bottom row show the highest density (1 p/m²) which have been tested.

The experimental evaluation in the crowd simulator uses the architecture presented in section 6.2, based on the simulated lidar measurements of the virtual robot. In this set of scenarios we have evaluated the response in the robot with different densities in a 1D flow crowd. The plot in the following figure shows the robot's time-averaged speed for each tested combination of crowd density and desired (nominal) speed commanded for the robot.

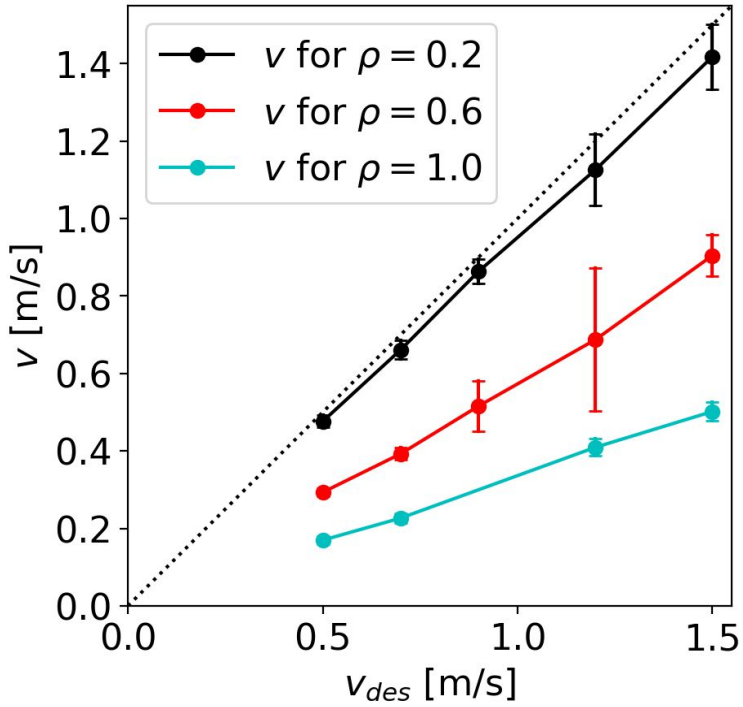


Figure 6.4.2 The robot's actual speed as a function of the robot's desired speed in the 1D-flow tests with different densities ρ (in persons per square meter). The plot shows the average values and standard deviation computed over several short trajectories per density and desired speed.

It is visible that higher densities reduce the robot speed, which can be attributed to two effects. First, the entire crowd moves at lower average speeds when the density is higher. Second, the RDS method is forced to reduce the speed at higher densities since it happens more frequently that a pedestrian occupies at some moment in time a part of the area which the robot's shape will traverse in the near future.

6.5. Redirecting Driver Support evaluation in a sparse crowd simulation from real data

In this section's experiments, the agents' nominal trajectories match real pedestrians' tracked motions from the "Crowds-by-Example" dataset [Lerner et al., 2007b] (which served as the basis for the data-driven crowd simulation technique in [Lerner et al., 2007a]). Our simulation here thereby reenacts the original dataset's crowd movement. Namely, every simulation agent's nominal velocity is generated as the nominal trajectory's velocity at the given time (yielding a feedforward term) plus a term which is proportional to the difference between the current position and the nominal trajectory's position at the given time (i.e. a feedback term). Details follow below on the agent's behaviour and control and how we configure the sample simulations underlying the evaluation.

For the simulation experiments in this section, the robot uses the proposed method RDS for reactive collision avoidance inside a simulated crowd. The crowd performs reciprocal collision avoidance (also considering the robot) using ORCA [Van Den Berg et al., 2011] via the open-source library RVO2 [Van Den Berg et al., 2016], whereas the robot reacts to the moving agents using the proposed method. For using ORCA, agents

perceive the robot as a finite collection of circles whose union's outline tightly bounds the robot's true capsule shape. Since agents need to know other agents' most recent velocities to construct their own new velocity constraints according to ORCA, they compute the robot circles' most recent velocities by transforming the robot's most recent forward and angular velocity to the circle centers' corresponding cartesian velocities.
Additionally, as a baseline for the experiments, we evaluate ORCA for controlling the robot, herewith, comparing the effects on crowd motion and robot response between the two methods.

We use ORCA as the baseline method for controlling the robot, even though it is non-holonomic and non-circular, whereas the original ORCA formulation only considers holonomic circular robots. Following the popular principle in [Snape et al., 2010] a solution to this problem is implemented by enclosing the robot in a single bounding circle and using ORCA to control this circle's center, whose velocity has degree of freedom two (if it does not lie on the wheel axle). This control point coincides for simplicity with the reference point for RDS. It could be chosen differently, but in any case it needs to be in front of the wheel axle because otherwise the robot aligns backwards to straight nominal trajectories. This is of course not permissible when there is a driver on board. For the Qolo-robot, this leads to a particularly conservative bounding circle, since its shape extends mostly in the rear direction.

The simulation uses ORCA for the agents and either ORCA or RDS for the robot to augment their behaviours due to the nominal trajectories with local reactive collision avoidance. This framework allows agents to handle perturbations to the original movements without colliding and then to return to their nominal trajectory due to the combination of feedforward and feedback control in the nominal command generation. Also, the nominal trajectories sometimes imply collisions, e.g. because the agents' nominal radius (0.3 m) does not match every real pedestrian or because the circular shape cannot model shoulder turning in close interactions.

The idea behind the experiments in this section is illustrated by Figure 6.5.1 The robot replaces one pedestrian and follows its trajectory, either using RDS or ORCA. For the quantitative evaluation in Table 6.2, a variety of configurations is generated by replacing in each configuration a different pedestrian by the robot. We identify each sample configuration by the pedestrian to be replaced by the robot, termed as the representative pedestrian. Each sample configuration gives rise to three simulation cases which together provide the data to evaluate metrics for the sample configuration. The first case does not include the robot, whereas the second case replaces the representative pedestrian by the robot using the RDS representation and control law, and the third case does the same replacement but with the ORCA representation and control law.

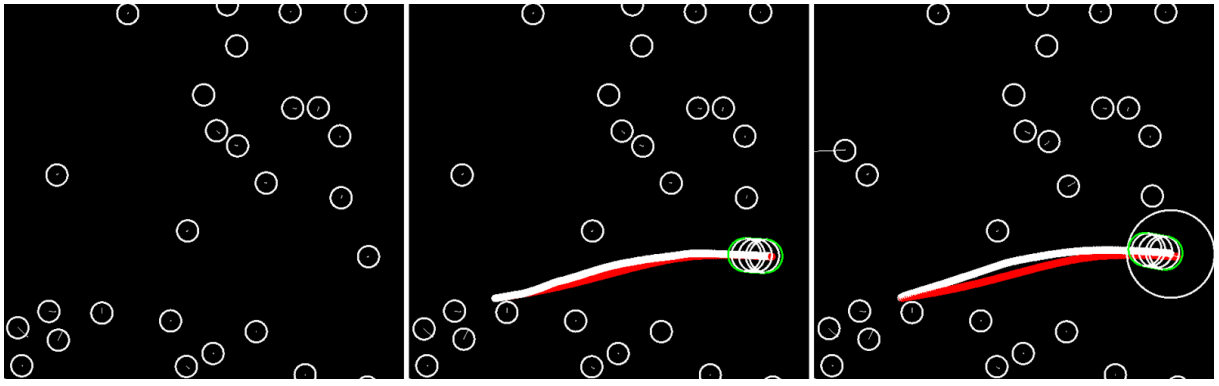


Figure 6.5.1 The tests framework incorporates real crowd recordings into the simulation. Each sample configuration has three corresponding cases, one without a robot (left), and two with the robot replacing a particular pedestrian, either with the RDS (middle) representation (green capsule) and control or the ORCA

(right) representation (big circle) and control. The ORCA representation's circle must have its center in front of the wheel axle, which leads to an overly conservative size.

The original recording's data points that describe the representative pedestrian's motion define the time window for the sample configuration's three simulation cases. After this time window elapses, a simulation holds the nominal trajectories constant and terminates after an additional time window (20% of the first time window) in which agents have time to converge to their now constant nominal goal positions.

The evaluation metrics follow below (in accordance with the consortium's metric definitions).

- *Mean deviation* $\Delta_0^{T_{final}}$. Quantifies the robot's ability to track the nominal trajectory by the temporal average of the robot's distance from its nominal position.
- *Crowd time to goal ratio* $E_t = \frac{T_{crowd\ alone}}{T_{crowd+robot}}$. Quantifies how much the robot's presence affects the pedestrians' time to converge to their goals.
- *Effect on velocity* $E_v = \frac{V_{crowd\ alone}}{V_{crowd+robot}}$. Quantifies how much the robot's presence affects the pedestrians' velocities.
- *Neighbours' time to goal ratio* $N_{ttg} = \frac{T_{robot\ neighbors}}{T_{crowd}}$. Quantifies how much the robot affects the reaching time for pedestrians that move in its vicinity.
- *Neighbours' velocity* $N_v = \frac{V_{robot\ neighbors}}{V_{crowd}}$. Quantifies how much the robot affects the velocity for pedestrians that move in its vicinity.
- *Total number of collisions* C. Counts the number of collisions between the robot's approximate capsule shape and the pedestrians' circle shapes.

While the metrics E_t and E_v require a case with the robot and the same case without the robot as a reference, the metrics $\Delta_0^{T_{final}}$, N_{ttg}, and N_v are evaluated only and directly for a case with the robot. The Table 6.1 reports the evaluation of the above metrics in terms of average and standard deviation (except for the total number of collisions) computed from 430 sample configurations, comparing RDS and ORCA as the robot controller. A two sample t-test was used to compare the difference in the output behaviour of both methods, finding significant differences at the level (p<0,05) for the metrics of mean deviation, crowd time to goal ratio, effect on crowd velocity E_v and effect on neighbours reaching time N_{ttg}.

Metric	*Mean deviation* $\Delta_0^{T_{final}}$ [m]	*Crowd time to goal ratio* E_t [-]	*Effect on velocity* E_v [-]	*Neighbours' time to goal ratio* N_{ttg} [-]	*Neighbours' velocity* N_v [-]	*Total number of collisions* C [-]
ORCA	3.6 σ : 2.6	0.999 σ : 0.005	**1.0034** * σ : 0.015	**1.001** * σ : 0.014	1.06 σ : 0.24	192
RDS	**2.9** * σ : 2.2	0.998 σ : 0.005	1.0077 σ : 0.014	1.007 σ : 0.019	1.05 σ : 0.24	**0**

Table 6.1 The metrics evaluation compares ORCA and RDS as the robot controllers. Superior performance values are marked in bold.

In the first case, favoring RDS with a significant improvement of *19.7% (*in average) smaller deviations from the original path.Whereas perturbations to other agents were consistently smaller with ORCA, the difference in velocity and time to goal of the other agents is affected less than *0.5%*. Moreover, the total number of tests that contained collisions with other agents was zero for the proposed RDS against 192 for ORCA for the whole set of 430 samples. Other metrics of effect on the robot and neighbours velocity showed no significant difference.

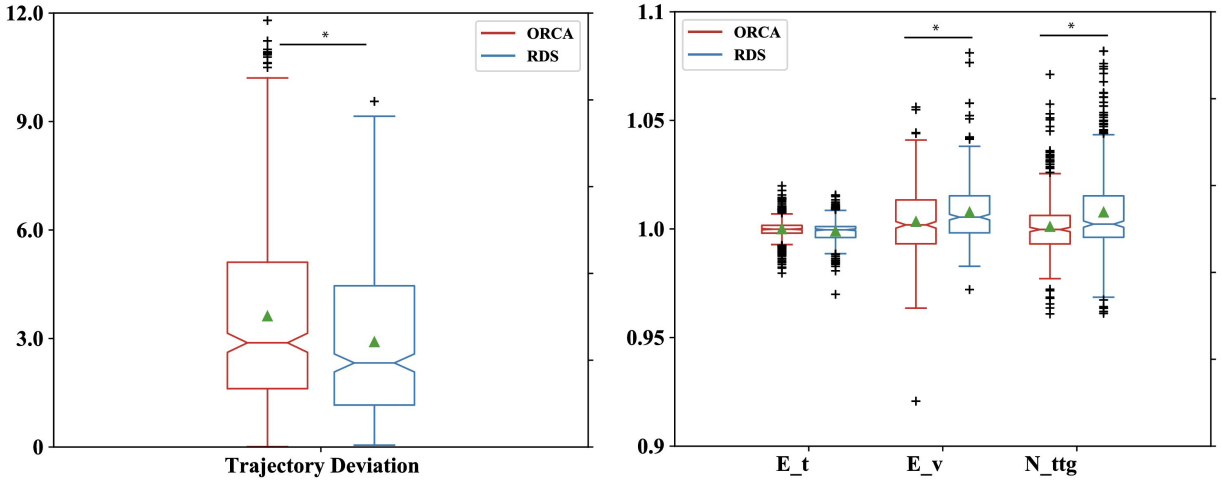


Figure 6.5.2: On the left, box plot for the deviation from the original path (m) and on the right, the effects on the crowd measured by crowd time to goal (E_t), crowd velocity (E_v) neighbours-time-to-goal ratio (N_ttg), with significant differences demarcated (*) at the level p<0.05.

6.6. RDS Experiments with Qolo

Considering that Qolo is a person carrier robot, thus, the user would have its input directly to the robot for motion control. Therefore, it is necessary to introduce a set of metrics to assess the level of assistance a reactive navigation would provide to the driver. In here, we consider the driver to be a high-level planner who makes the decisions for the long-term plan (such as, desired velocity, general direction of motion, and goal), and compare how the designed algorithm performs for his/her assistance.

6.6.1. Evaluation metrics for shared control:

All metrics have been defined for the CrowdBot testings in D1.4, however, we define here the selected metrics that could be applied for the current tests

Agreement: We defined agreement in terms of the deviation of the direction of user's commands from the direction of the final shared control's velocity, herewith, considering only the directional agreement and not magnitude, defined as, $agreement = \frac{\sum_{i=0}^{N} a_i \cdot \Delta t_i}{\sum_{i=0}^{N} \Delta t_i}$, where the normalized agreement at each time step is defined as $a_i = 1 - \frac{|\theta(u_h^i) \ominus \theta(u_r^i)|}{\pi}$, for $\theta(u) = tan^{-1}(\frac{v}{w})$.

where v and w are the translational and rotational velocities $\tilde{u}[v\ w]$, a_i is the normalised agreement at time step, u_r^i is the final output of the shared control or reactive navigation. N is the number of samples available in which data from both the measured input of the user, u_h^i coincide in time with u_r^i. Δt_i is the duration of user's input command u_h^i.

Overall Disagreement: The normalized disagreement between the user command and the robot's execution is computed as the mean of the L2-norm of the command differences D =1/N $\sum_{i=0}^{N} \|u_r^i - u_h^i\|$ for the test interval. This measurement gives an insight into how the algorithm performs compared to the set desired motion of the user. Both normalized to the maximum linear and angular velocities. In here, || x || represents the L2-norm of x.

Assistance Contribution: The contribution C is calculated as C =1/N $\sum_{i=0}^{N} \|u_r^i - u_h^i\| / \|u_h^i\|$, over a finite number of discrete samples N. Where the u_r^i is the user command in the robot command space (linear and angular speed), u_r^i is the user input.

Fluency: We observe the fluency of commands of the user (temporal continuity) given that the reactive navigation is intervening with its desired motion, as a second reference measurement of the disagreement with the decisions of the assistive navigation. $F = \frac{1}{N} \sum_{t=t_0}^{t_N} 1 - |u_h^t - u_h^{t-1}|$, where u_h^t represents the user given command at time t, and N the number of samples taken in the interval t_0 *to* t_N.

Risk: We first define the notion of distance to closest approach (dca), which is, at a given time, the minimum distance in meters between the robot and an agent on the crowd if their current velocity were constantly projected in a time window. If they will eventually collide if they do not try any avoidance maneuver, the time of closest approach (ttca) is the corresponding time in seconds, at which the collision happens. The risk is defined as the sum of the minimal ttca if dca is 0 at each time step.

$$R = \sum_{0}^{T_{robot}} (1/ttca_{min}\ if\ dca = 0,\ 0\ otherwise)$$

Expected Severity : We define the severity derived from the previous risk definition including the current relative velocities of the agents: $S_R = \sum_{0}^{T_{robot}} (v_{relative}^2 \times 1/\ ttca_{min}\ /if\ dca = 0,\ 0\ otherwise)$

Number of collisions: This metric simply counts collisions that occured in a scenario. Note that a collision is not necessarily dangerous, as a touch is already a collision.

6.6.2. Interactions with one pedestrian and static environment

This section presents experiments with Qolo using the RDS method as described in the respective theoretical section in this document. The experiments here demonstrate how the RDS method avoids collisions in basic interactions with one pedestrian and a static environment. First, a description of each experiment with motion and command plots as well as their interpretation are given. Finally, the Table 6.2 gives a quantitative evaluation of the metrics from the previous section for the four described tests.

Case 1: Overtaking a pedestrian

The Figure 6.6.1 shows an overtaking maneuver which represents the basic idea behind this test, in which a pedestrian walks straight forward and the driver with Qolo approaches from behind and overtakes the pedestrian.

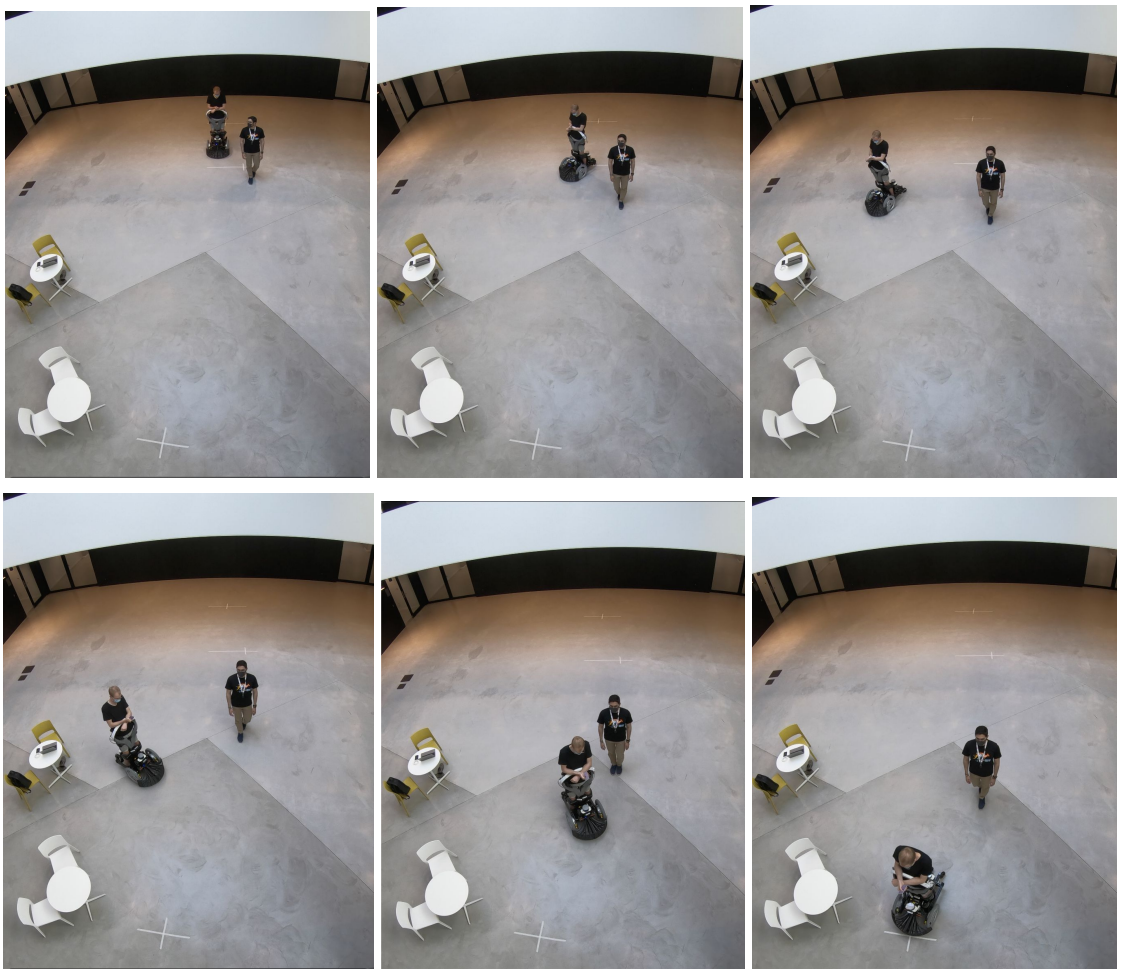


Figure 6.6.1: Snapshots of one of the experimental evaluations for overcoming a pedestrian.

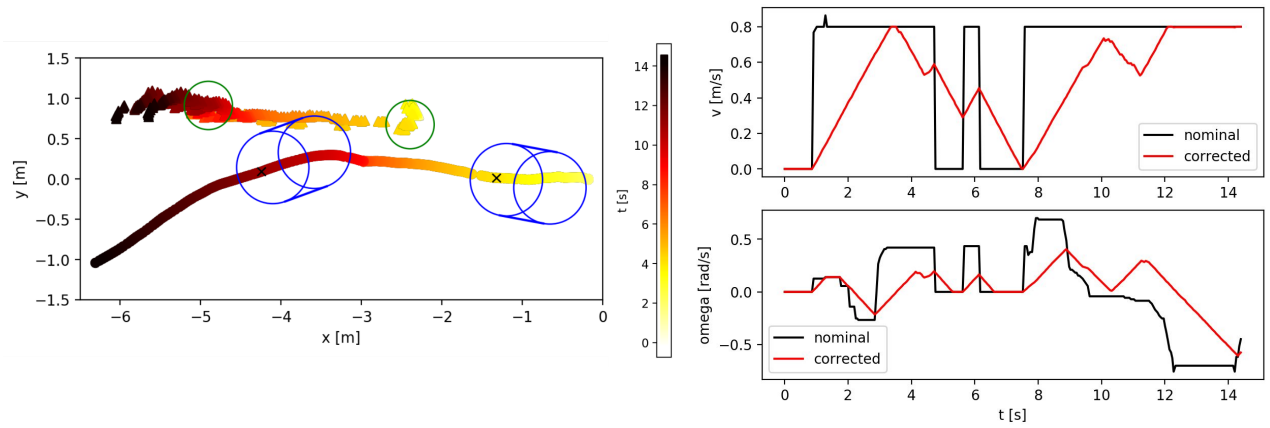


Figure 6.6.2 Test overcoming a pedestrian. On the left trajectories with color indicating time, becoming darker over time (as the scale on the right indicates). For two selected instants in time, the robot's shape (blue) and the pedestrian's shape (green) are drawn. On the right, the nominal linear and angular velocity commands over time (top and bottom, respectively, black) are transformed into corrected commands by RDS

(red).

The Figure 6.6.2 shows the estimated trajectories for this test (where the direction of motion in the plot on the left is from right to left). It can be seen in the top right plot in Figure 6.6.2 that RDS decelerates when Qolo approaches the pedestrian while the nominal linear velocity command remains constant. Subsequently, the incoming nominal commands step between zero and constant high values. This period, wherein the acceleration constraints govern the output command, is followed by a period where again the constraints for collision avoidance become active (around t=10s). It can be seen from Figure 6.6.2 (left) that Qolo is at this point again closer to and moving towards the pedestrian. Then, RDS initiates a left turning motion, as the bottom plot in Figure 6.6.2 (right) shows, since the corrected angular velocity becomes positive while the nominal angular command is zero, which would keep moving the robot straight forward. Thus it can be seen also in Figure 6.6.2 (right) that the method assists the driver to avoid the collision and successfully overtake the pedestrian towards the end of the time window.

Case 2: Backward crossing a pedestrian

As Figure 6.6.3 shows, in this test Qolo is driven backwards into a crossing pedestrian.

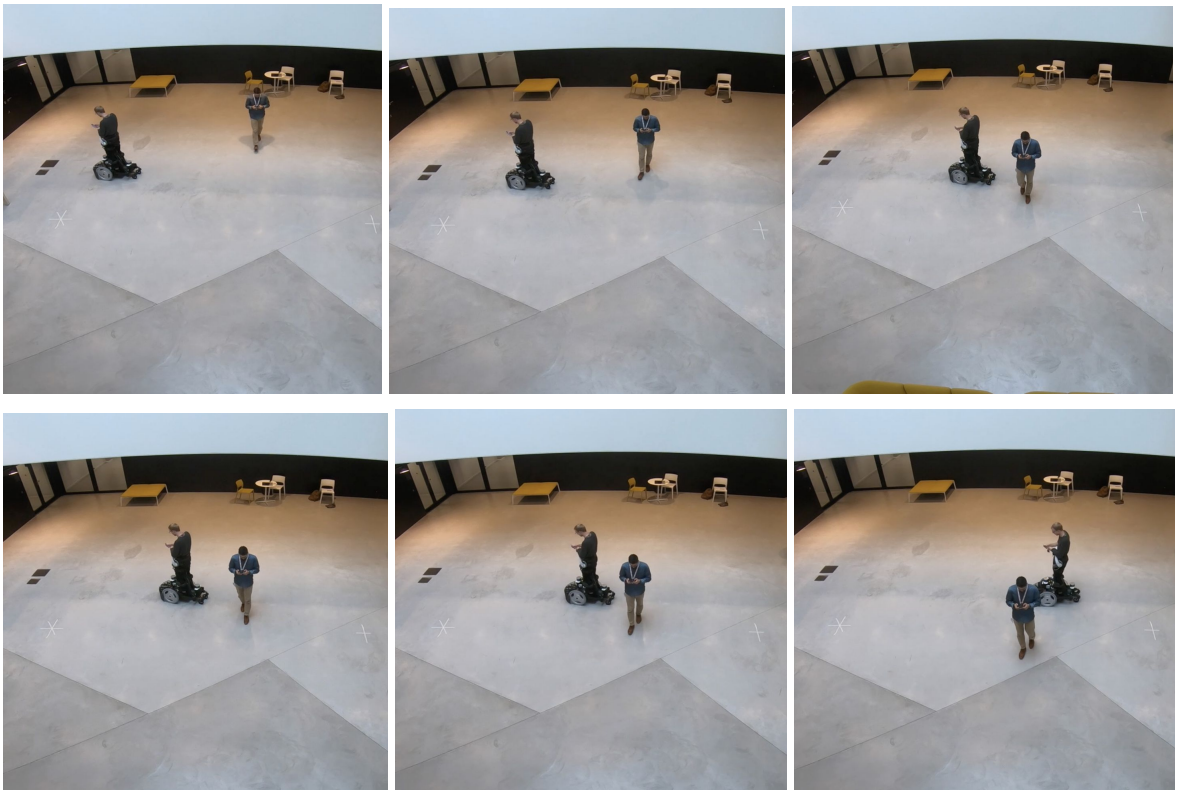


Figure 6.6.3: Snapshots of one of the experimental evaluations of backwards driving and intersecting a pedestrian.

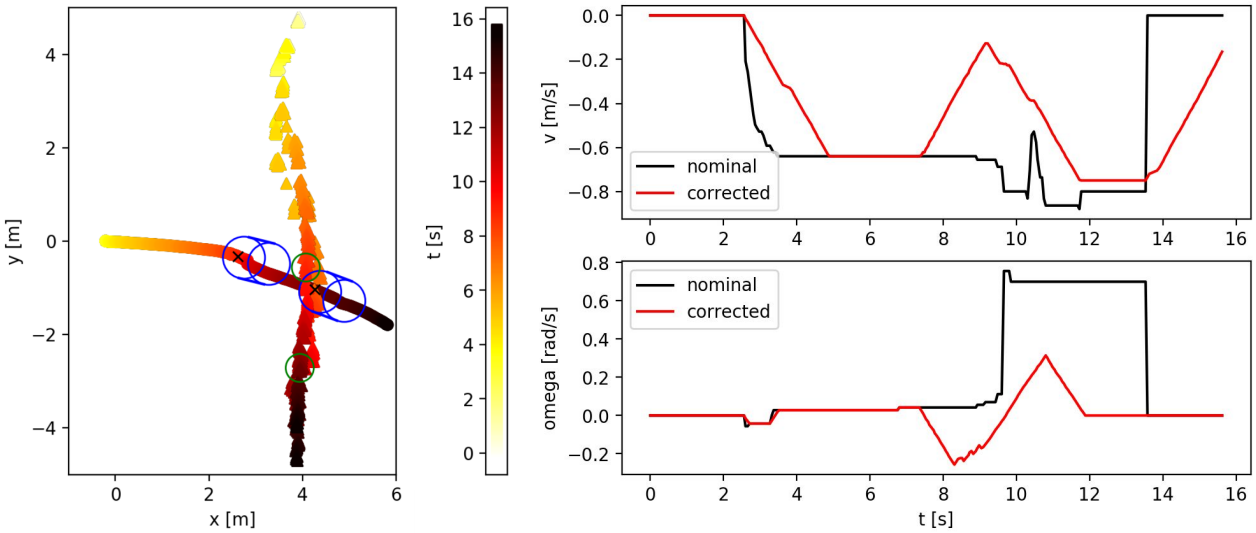


Figure 6.6.4 Test driving backwards towards a pedestrian. On the left, their shapes (blue capsule, green circle) are shown for two exemplary time instances. On the right, the nominal linear and angular velocity commands over time (top and bottom, respectively, black) are transformed into corrected commands by RDS (red).

It is visible in Figure 6.6.4 (left) that Qolo and the pedestrian's trajectories meet such that it is rather ambiguous who will pass first (i.e. the crossing order). Namely, the plot displays their shapes around the time instance when they meet, where the pedestrian is only slightly ahead of Qolo. The top right plot in Figure 6.6.4 shows that RDS subsequently corrects the nominal linear velocity command to decelerate Qolo and consequently Qolo gives way to the pedestrian. At the same time, the corrected angular velocity initiates a right turn such that the capsule's cap rotates away from the pedestrian and leaves more space to pass. After crossing the pedestrian, Qolo resumes the straight motion in the original direction.

Case 3: Forward crossing a pedestrian

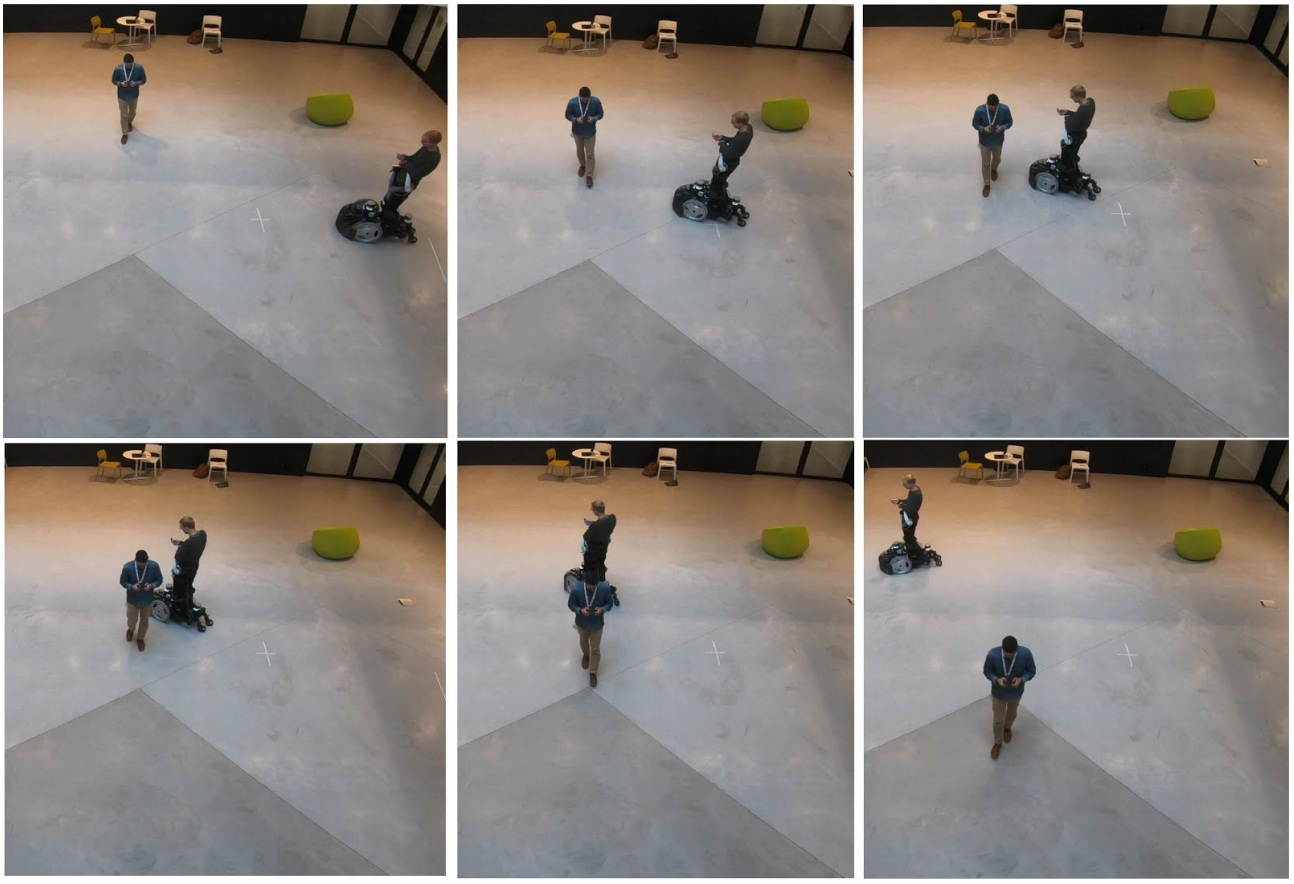


Figure 6.6.5: Snapshots of experimental evaluations for forward driving and intersecting a pedestrian.

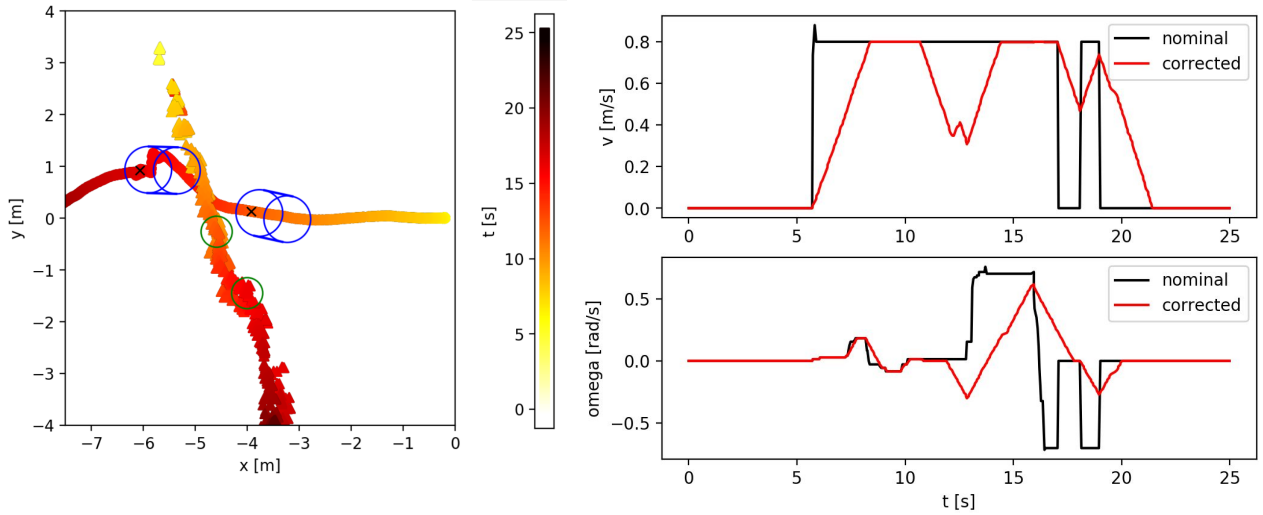


Figure 6.6.6 Qolo test driving forward into a pedestrian. Their shapes (blue capsule, green circle) are shown for two exemplary time instances.

As Figure 6.6.5 shows, in this test Qolo is driven forward into a crossing pedestrian. As in the backward crossing test which was described earlier, Qolo gives way to the pedestrian, whereas here, the anticipated crossing order is less ambiguous, as the first snapshot of their shapes in Figure 6.6.6 (left) shows. The top

right plot in Figure 6.6.6 indicates how RDS makes Qolo decelerate and the bottom plot shows at the same time a deflection by a right turning maneuver. Qolo's trajectory also indicates this rotation which can facilitate passing behind the pedestrian and reduce the linear deceleration necessary to avoid the collision. The direction of the rotation is different from the backward crossing case above and corresponds to a more time-efficient maneuver, as it does not require to stop during the maneuver. The superior navigation performance by RDS in this example can be attributed to the fact that the anticipated crossing order is more clearly pre-determined in this example. After passing behind the pedestrian, the driver brings Qolo again on the original course.

Case 4: Navigating clutter corridor

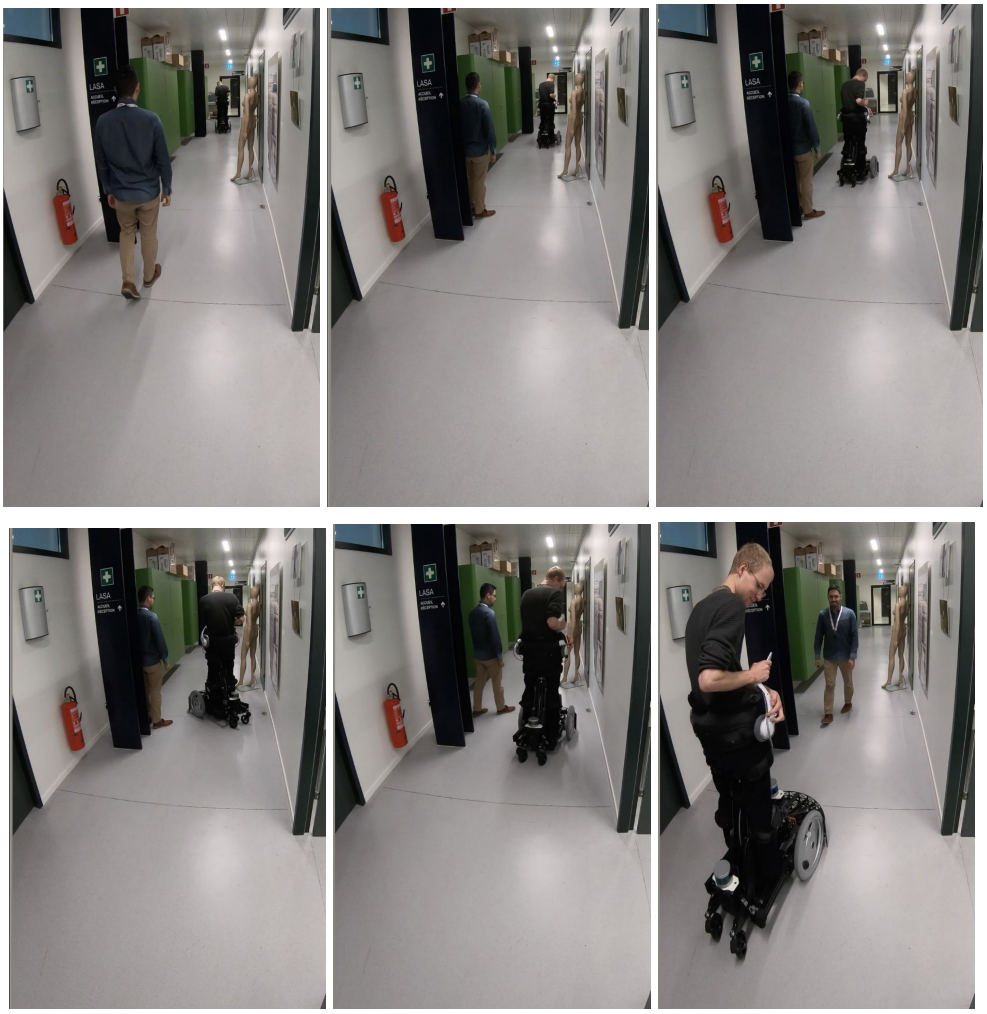


Figure 6.6.7 Qolo is shown, driving in the corridor in which the related test is performed.

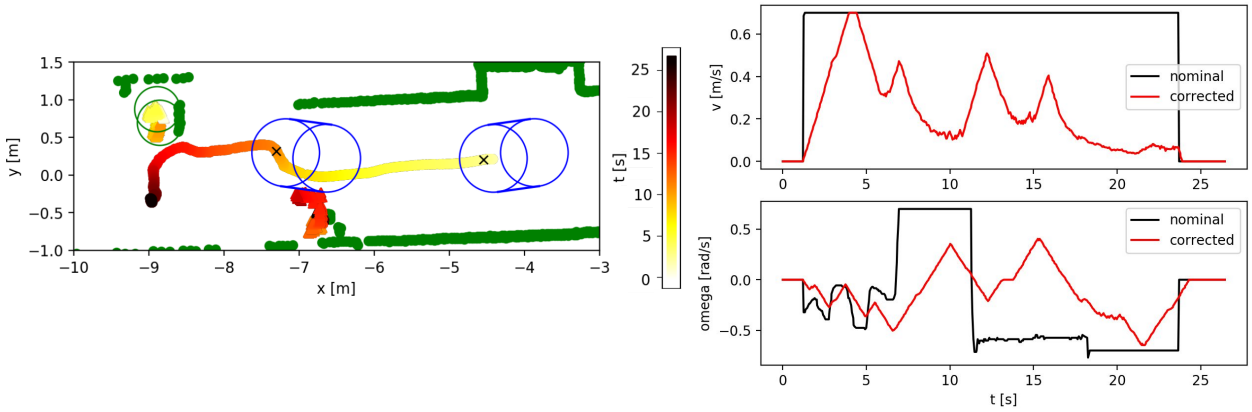


Figure 6.6.8 Qolo drives through a narrow corridor and avoids obstacles using the RDS collision avoidance system. On the left, its trajectory is shown by the swept circle whose color encodes time (yellow-red, as the right scale shows).

As presented in Figure 6.6.8 (left), pedestrian detections occur at two (static) locations (indicated by triangles encoding time similarly). The later pedestrian detection and Qolo's shapes (green circle and blue capsule, respectively) as used by RDS are shown for two exemplary time instances. Additionally, RDS retrieves the raw scan points (from the two onboard LIDAR sensors) and treats them as static obstacles. They are shown for two exemplary time instances (green points).

The constraints for collision avoidance become active as Qolo gets closer to the first obstacle (which is also a pedestrian detection, visible in Figure 6.6.7 close to the wall on Qolo's left in the direction of travel). First, as the top right plot in Figure 6.6.8 shows, RDS decreases the output linear velocity (around t=4 s) and then deflects the motion to the right as the bottom plot shows (until t=7 s). Then the angular acceleration constraint becomes active and slightly delays the rotation in the opposite sense which would align Qolo again with the corridor. Therefore, Qolo moves towards the wall on its right and RDS decelerates the linear velocity consequently until the alignment is complete. At this point (around t = 10 s) the linear velocity can increase again (adhering to the acceleration constraint) whereas counter-clockwise rotation is now constrained to prevent Qolo's tail from hitting the wall to its right.

As Qolo approaches the second obstacle (a pedestrian detection, visible in Figure 6.6.8 (left) close to the wall on Qolo's right), the linear velocity is constrained again and RDS also initiates a left turn deflecting the robot to avoid the obstacle (around t=12 s). The subsequent navigation is then similar again as after avoiding the first obstacle, i.e. the angular acceleration constraint becomes active again and it thus requires some time until Qolo is aligned again with the direction of travel along the corridor.

Quantitative Evaluation

For the tests described above, Table 6.2 presents a quantitative metrics evaluation, where the definitions for the metrics are given in Section 6.6.1.

Metrics	Case 1: Overtaking a pedestrian	Case 2: Backward crossing a pedestrian	Case 3: Forward crossing a pedestrian	Case 4: Navigating Clutter corridor
Linear Velocity Difference [%]	40.78±34	25.72 ±25.1	22.4±25.4	62.81 ± 20.77
Directional Agreement [%]	91.11±6.4	89.19±9.8	87±9.9	79.87 ± 10.07
Disagreement [%]	65.57±30.6	48.61±37.6	57.82±34.2	88.42 ± 28.9
User Fluency [%]	95.83	98.25	97.95	97.42
Autonomy Contribution [%]	55.28	45.44	48.85	77.86
Minimal Distance Obstacles [m]	0.32	-0.08	0.011	0.027
Risk	0.23	60.01	126	13.34
Expected Severity	0.067	16.915	154	0.095
Number of Contacts	0	1	0	0

Table 6.2 The displayed metrics provide a quantitative evaluation of the RDS collision avoidance method for the four basic tests with one pedestrian and a static environment.

Conclusion

For the above basic interactions between Qolo and a pedestrian in a static environment, the functionality of the RDS approach to collision avoidance has been demonstrated. In all cases, the collision avoidance system decelerated or deflected the robot and thereby ensured collision-free motion. In all the cases except in Case 2, even the robot's capsule shape approximation maintained a positive distance to other shapes (as the minimal distance values in Table 6.2. indicate). However, as the capsule shape is rather conservative, also the Case 2 did not exhibit any physical collisions, rather a virtual intersection between the representative shape of the robot and obstacles, in this case, due to the implemented acceleration constraints and a small latency in the sensing pipeline. In terms of shared control for the current set of tests, the responses in user fluency *(over 95%)*, shows that the driver was consistently leaving the maneuvering to the autonomous system (validated with over *45%* contribution in all tests and disagreement between *48%* and *88%*). These results validate the intention of the experiment of leaving the reactive control to the proposed RDS method, while a high-level planner (driver) only gives a general intention of motion without worrying about the environment, as it is desired for a crowd situation.

6.6.3. Crowd Evaluation with preliminary RDS version

To use object information directly from sensor data without introducing any latency or potential for missing out object parts during further processing, the current implementation creates a circle and corresponding constraints for each measured point provided by a planar laserscanner.
In our case, the shape of the robot, Qolo, is approximated by two circles to take into account the elongation from the back to the front. as shown in Figure 6.6.9.

The implementation uses a geometric algorithm to solve the quadratic program at the core of the RDS method, which is also being used in the ORCA method [Van Den Berg et al., 2011]. Besides being efficient on small-dimensional problems like this one (which is in 2D), its advantages include that it converges in a finite number of iterations, which is desirable to ensure real-time execution, and that it is simple and rather easy to implement.

Our implementation of the method on Qolo uses two circles (as depicted in Figure 5.4), which are centered at the LIDAR sensors, to approximate the robot's shape. This allows the robot to move closer to an object than the distance where the LIDAR sensors can still perceive it reliably.
The parameters which we have tuned in the tests are:

- Robot shape (circles' radius): based on sensor's accuracy (0.3 m ~ 0.6 m)
- Minimum clearance to obstacles imposed via the constraints: based on the robot's control latency (0.03 ~ 0.1 m)
- Maximum velocity towards obstacles as a function of their distance imposed via the constraints: (0.0 m/s ~ 0.3 m/s) .

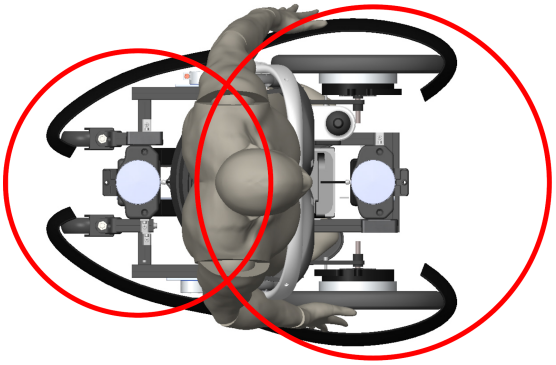


Figure 6.6.9: Implemented representation of the robot Qolo in the 2D spaces for the preliminary assessments in real crowds.

We have conducted two sets of pilot tests to assess the shared control and reactive collision avoidance system. The first set of tests were conducted in an enclosed lab space, with a set of circuits. The second tests were done in a public corridor (140 m long), highly frequented by students, at EPFL (estimated 0.2 to 0.8 person per square meter at peak hours).
Tests conducted in an enclosed space were meant to allow us to tune the control system's parameters such as to maximally prevent collisions before deploying the robot in the real pedestrian environment. While obstacle avoidance is our priority, we retained tuned parameters to maintain the robot's ability to maneuver between objects according to the driver's commands.

6.6.3.1 Laboratory Assessment

Indoor testing was performed for analysis of the behavior of the sensor and robot projection to a single plane and tuning parameters in the reactive navigation, such as, permitted proximity to the obstacles, uncertainty error to sensor's proximity, acceleration constraints, and user preference. To this end, a set of 2 static scenarios in the laboratory were used as shown in the figure below.

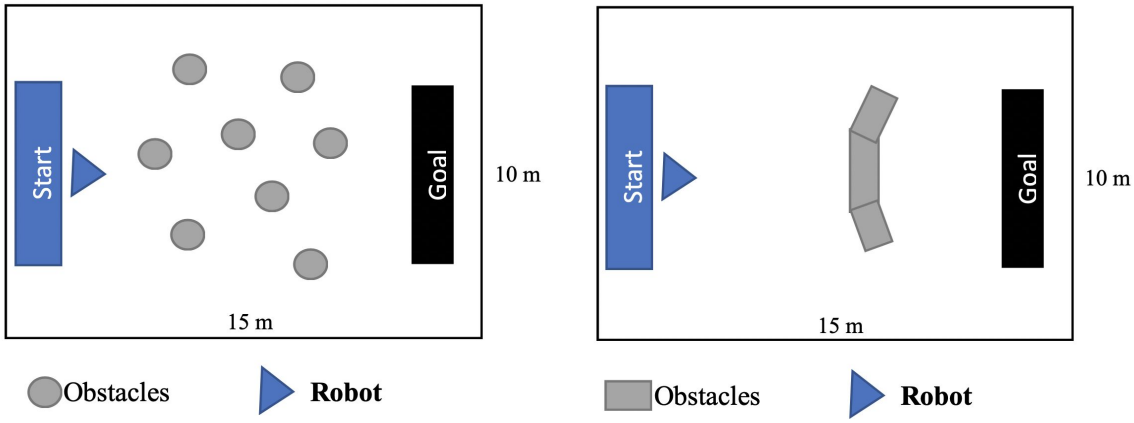


Figure 6.6.10: Diagram of evaluation scenarios for the lab environment.

Case 1: Wall Test

In the situation depicted by Figure 6.6.11, the collision avoidance system maintains a minimum clearance between the laserscan points (yellow) and the two circles, which approximate the robot's shape. The driver's command points forward as indicated by the velocity vector for a fixed point (blue, drawn from the fixed point) and is modified by the collision avoidance system to yield a vanishing actual velocity command vector (green).

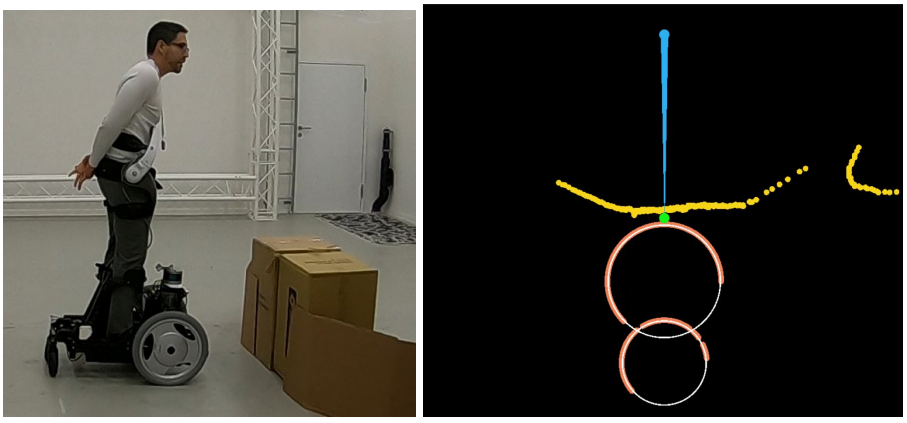


Figure 6.6.11: Obstacle clearance test in the enclosed space.

To ensure collision avoidance, we have first tuned the front shape's radius and the minimum clearance from obstacles as imposed via the constraints. We have set the parameters for the front circle's radius to 0.4 m (and for the rear circle's radius to 0.3 m) and the minimum clearance to obstacles to 0.1 m. With these values, the collision avoidance system makes the robot stop at a distance of around 0.4 m from a wall in front (as Figure 6.6.11 shows), where the distance is measured between the front of the wheels and the wall.

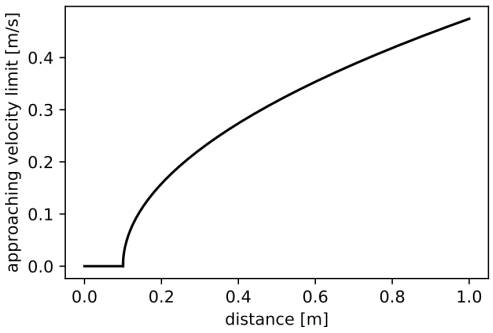


Figure 6.6.12: The chosen (shifted and scaled) square root function which limits the velocity towards objects dependent on their distance.

Further, we have set the function which bounds the velocity via the corresponding constraint to be the square root function (scaled by a factor of 0.5 m^(0.5)/s and shifted by the found clearance value of 0.1 m), as this achieves a constant deceleration profile (with a deceleration of 0.125 m/s^2) when approaching a wall. Figure 6.6.12 shows the resulting function for the permitted velocity towards an object dependent on its distance to the robot. In the framework of the velocity obstacle formulation, this feature corresponds to linearly increasing the value for the time horizon, which is also called the (minimum) time-to-collision, dependent on the proximity to the object when constructing its velocity obstacle.

Case 2: Static obstacles circuit

Figure 6.6.13: Circuit test in the enclosed space.

In this test the system's ability to pass through narrow passages was evaluated. The collision avoidance system assists the driver to navigate the test circuit as depicted in Figure 6.6.13. Here, we have reduced the parameter for minimum clearance to obstacles to 0.05 m as this helps with navigating in narrow spaces, herewith allowing the driver to control contact with obstacles at very low speeds (< 0.11 m/s).
We have observed that the driver can pass among the obstacles through the targeted passages while the system deflects and turns the robot away from obstacles.

The normalized disagreement between the user command and the robot's execution was measured to be 30.5% ± 30.11. This measurement gives an insight into how the algorithm performs compared to the set

desired motion of the user. Detailing the difference in commands for heading angle and linear velocity we found that angular velocity observes an influence of 14.95% ±19.61 compared to a 23.75% ±25.78 for the linear velocity. This shows that the algorithm corrects more the linear velocity over the heading angle, which is the desired behaviour for following the user intended motion. Finally, the overall contribution of the reactive navigation for performing the circuit has been computed to be C = 33.83% ± 30.67. This result highlights the assistance provided by the reactive navigation in the driving task during the circuit.

Based on the tests, we found that the best parameters were those described in the table below, for the reactive navigation algorithm usage for crowd navigation. These are the parameters which we plan to use for further shared navigation user-study to be performed by volunteer participants at EPFL open crowd of students in the corridors on campus, as approved in the experimental protocol EPFL HREC-032-2019.

Constraint type	**Value**
Clearance to obstacle	0.05 [m]
Absolute Clearance	0.1 [m]
Obstacles uncertainty (based on sensor's error)	0.05 [m]
Velocity (linear and angular)	1.0 [m/s], 1.03 [rad/s]
Acceleration (linear and angular)	1.0 [m/s^2], 1.5 [rad/s^2]

Table 6.3: Constraints set for shared control tests in real crowds.

6.6.3.2 Crowd Testing

The second set of tests were conducted in a corridor of EPFL which is used on a regular basis by students to move from one classroom to the other. In the current test, the robot was tasked to travel in the corridor while students would also walk (a real crowd).
We picked a time period where the corridor was moderately crowded, it was observed from 0.2 to 0.8 person per square meter (ppm), with *sparse crowds* moving in *one dimensional flow*. During these tests, we encountered two situations: first, the robot was *moving in the same direction as the flow* (scenario 1.3 from D1.1), and second, the robot was *moving opposite to the flow in a 2D crowd* as in scenario 2.1 from D1.1.

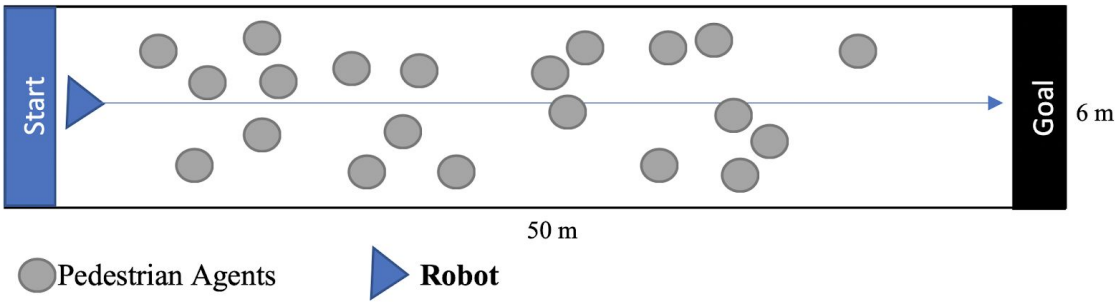


Figure 6.6.14: Plan of the corridor of 140 m where crowd tests were performed at EPFL (highlighted in green in the top picture). In the bottom a simple diagram of the task.

This experiment followed the guidelines of the approved experiments by the ethical committee at EPFL, which established a designated area as shown in Figure 6.6.14 (a long corridor where crowds of students are common during the break in-between two lectures). Signs displays were placed in the corridor to indicate that an experiment was performed and to inform passers-by that all data collected from the robot would be stored on EPFL's designated computer and follow Data Protection Law of Switzerland. From this test, data from on-board RGB-D camera and laser scanner as well as an external static camera were recorded for performing the post-analysis.
The current tests aimed to estimate the level of assistance that is provided by the navigation algorithm to the driver, and the level of agreement (or disagreement) that the driver would have from the executed motion by the robot. Therefore, we evaluate the shared control metrics proposed in D1.4 and presented in section 6.6.1. Within the crowd test we have taken 3 cases to highlight the differences in the algorithm contribution to the navigation.

- Case 0: No crowd, moving parallel to the wall.
- Case 1: Qolo moving with a 1D flow.
- Case 2: Qolo moving opposite to a flow, in a 2D flow.

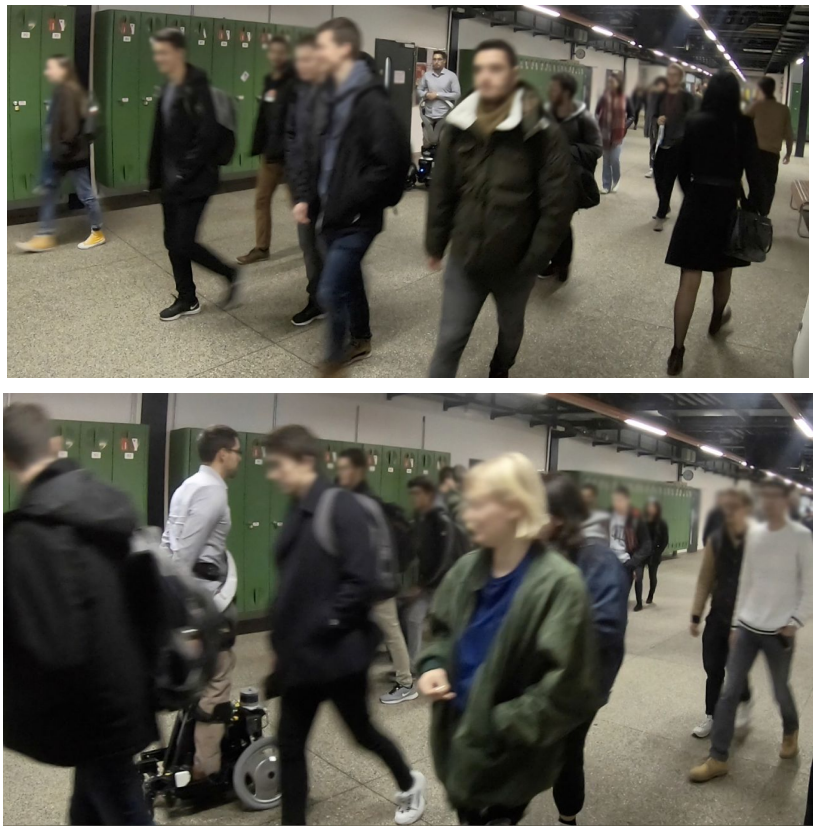


Figure 6.6.15: Scenarios taken from a static camera. From top to bottom: cases 1, 2, and 3

In each of these scenarios (depicted in Figure 6.6.15) a sample of 20 seconds of data was isolated out of the whole recording (30 minutes of data) for matching the proposed evaluation scenarios in D1.1, namely 1D flow motions and 2D flow motions. Out of these, we computed the measurements of assistance provided by the reactive navigation algorithm, and the level of agreement to the user input.

The first baseline scenario without a crowd (Figure 6.6.16) shows single pedestrians and static obstacles (walls) as the only constraints to the motion of the robot, thus, nominal commands by the driver are followed closely (Figure 6.6.16 bottom).

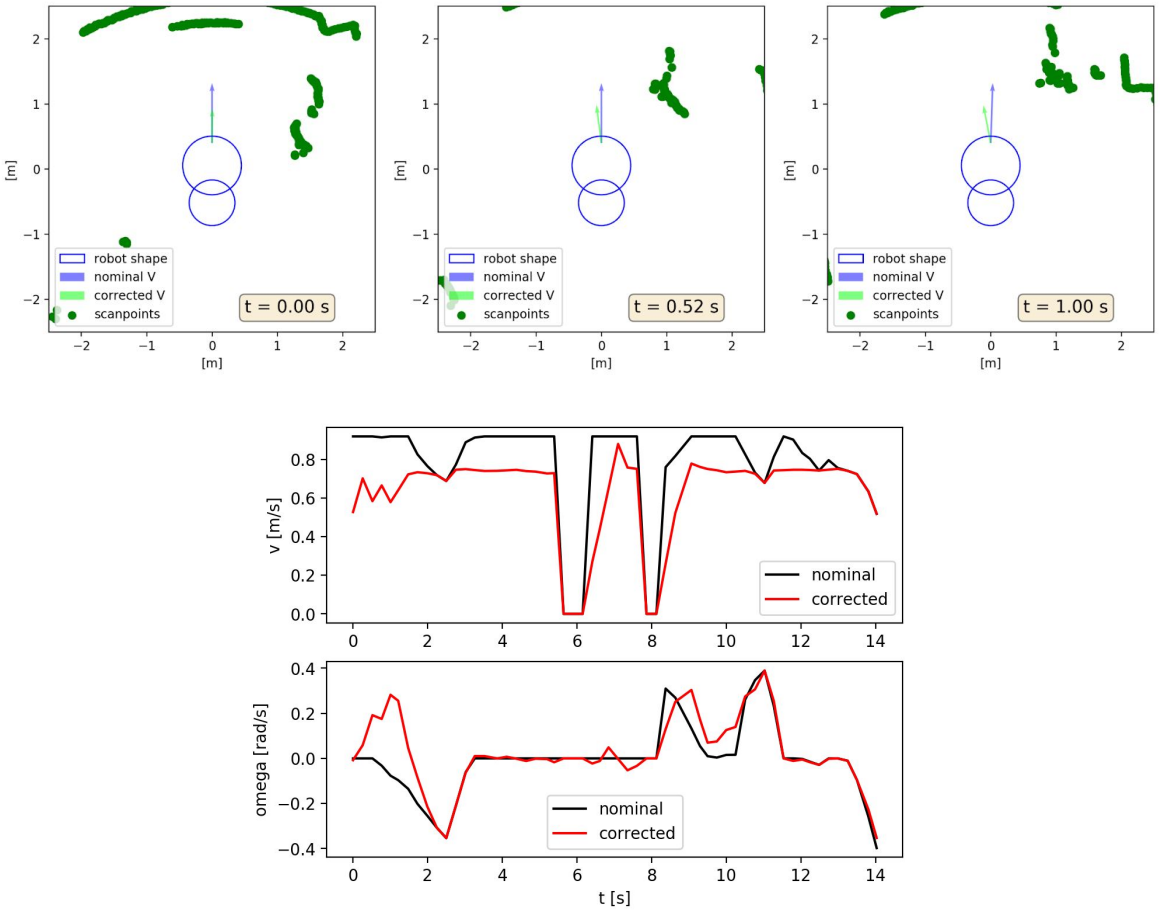
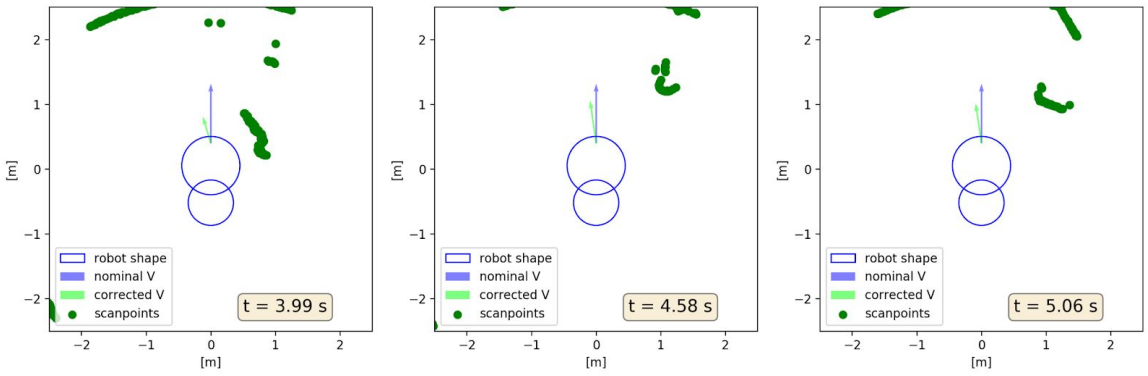


Figure 6.6.16: For the Case 0 (no crowd, only a few single pedestrians), the top plots are snapshots showing the laser-scan points, the nominal and the corrected velocity (mapped on an example point on the robot), and the robot's approximate shape. The bottom plots show the nominal and corrected linear and angular velocity commands over time.

The second case of evaluation was within a one-dimensional crowd following the flow, in this case, the commands (as shown inFigure 6.6.17), are significantly deviating from the driver's in consequence of the crowd moving around the robot.



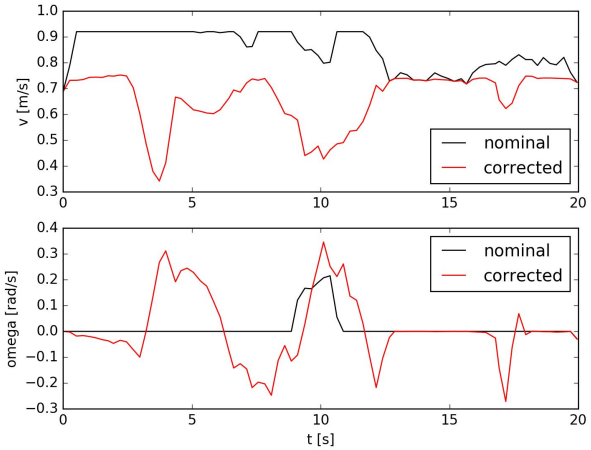


Fig 6.6.17: For the Case1 (Qolo moving with a 1D flow), the top plots are snapshots showing the laser-scan points, the nominal and the corrected velocity (mapped on an example point on the robot), and the robot's approximate shape. The bottom plots show the nominal and corrected linear and angular velocity commands over time.

Finally, the third case of 2D flow of the crowd, leads to a set of corrections (Figure 6.6.18) with higher variability than the previous cases, as the pedestrians in the crowd appear more suddenly in the robot's way.

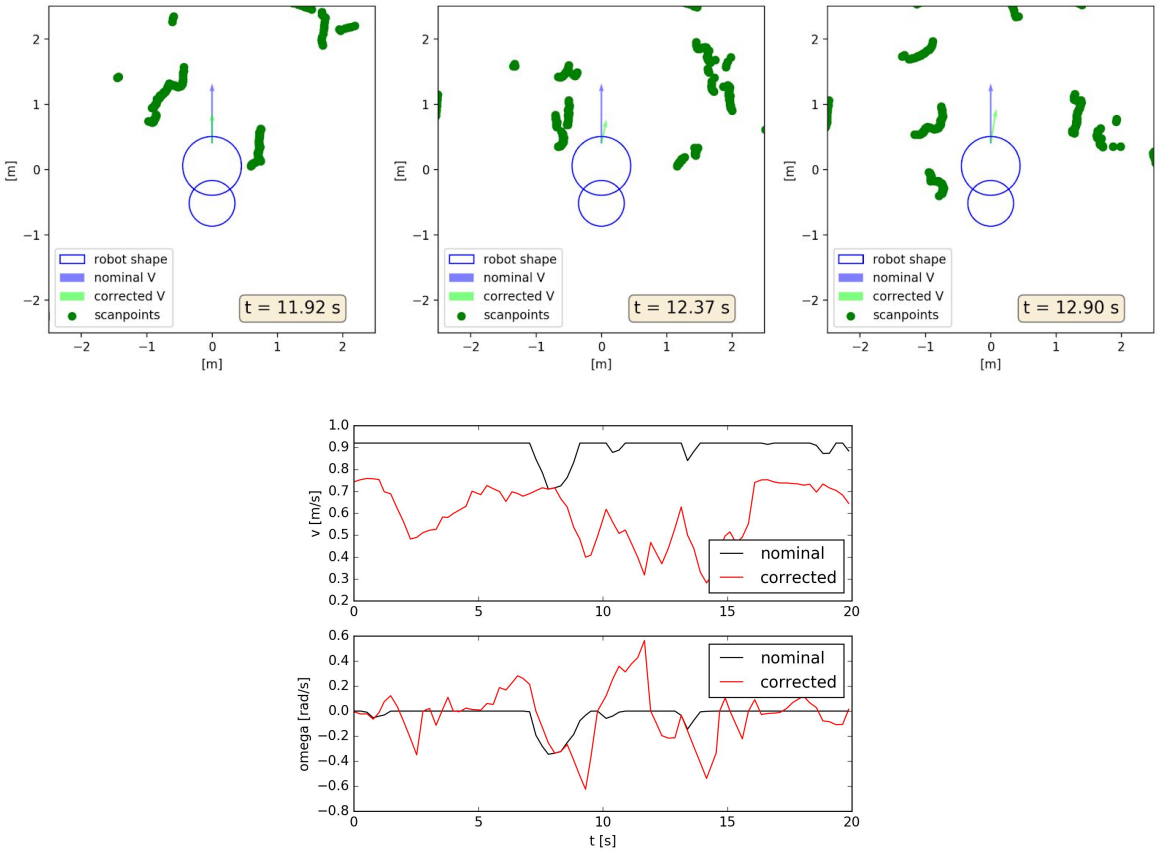


Fig 6.6.18: For the Case2 (Qolo moving opposite to a flow, in a 2D flow), the top plots are snapshots showing the laser-scan points, the nominal and the corrected velocity (mapped on an example point on the robot), and the robot's approximate shape. The bottom plots show the nominal and corrected linear and angular velocity commands over time.

The results as described in the table below, highlight an increment in the autonomy contribution to the navigation from the algorithm between the 3 scenarios, from 16.56% at case 0 to 35.57% in the bidirectional flow. As well, we can highlight the fact that linear difference was higher than angular difference for all three cases, meaning that the algorithm performs as designed, following as closely as possible to user heading angle (the desired direction of motion given by the drive).
Finally, the disagreement of the velocity vector with the user input increases from 17.9% to 34.58% from case 0 to case 3. Nonetheless, the fluency of the user's commands is not affected, remaining between 93% to 98% in all cases. This highlights the fact that the user effort in navigation is not increased although the robot's autonomy is increasing.

Measurements	Case 0: No Pedestrians	Case 1: 1D Flow:	Case 2: 1D Flow - contrary
Linear Velocity Difference [%]	16.07 % ± 14.04	18.8 % ± 14.9	30.34% ± 14.5
Directional Agreement [%]	5.28% ± 8.2%	7.9% ± 8.6%	13.28% ± 14.37%
Disagreement [%]	17.9 % ± 15.15	21.17 % ± 16.3	34.58 % ± 17.79
Autonomy Contribution [%]	16.56%	21.26%	35.57%
User Fluency [%]	93.3%	98.3%	98.8%
Minimal Distance Obstacles [m]	0.72	0.29	0.11
Number of Contacts	0	0	0

Table 6.4: Results of the 3 testing scenarios in real sparse crowds for shared control metrics.

In Figure 6.6.19, we depict the closest distance to the robot for each of the scenarios above. Showcasing the tests in the public corridor where the driver could navigate through the crowd, with the collision avoidance system preventing running into pedestrians, with a zero collision count in all scenarios.
Compared the the single pedestrian scenarios in the tests on the previous section, we observe a clear decrease in the autonomy contribution, which reflects that the driver was actually giving instructions in the trajectory to follow during the experiment, however, there is not an observable decrease in the fluency which suggest a good compatibility with the driver intentions
Over all no contact or collision was registered either virtually (robo's representation to obstacles) neither physical in any of the tests, although, pedestrians obviously contribute to collision avoidance themselves.

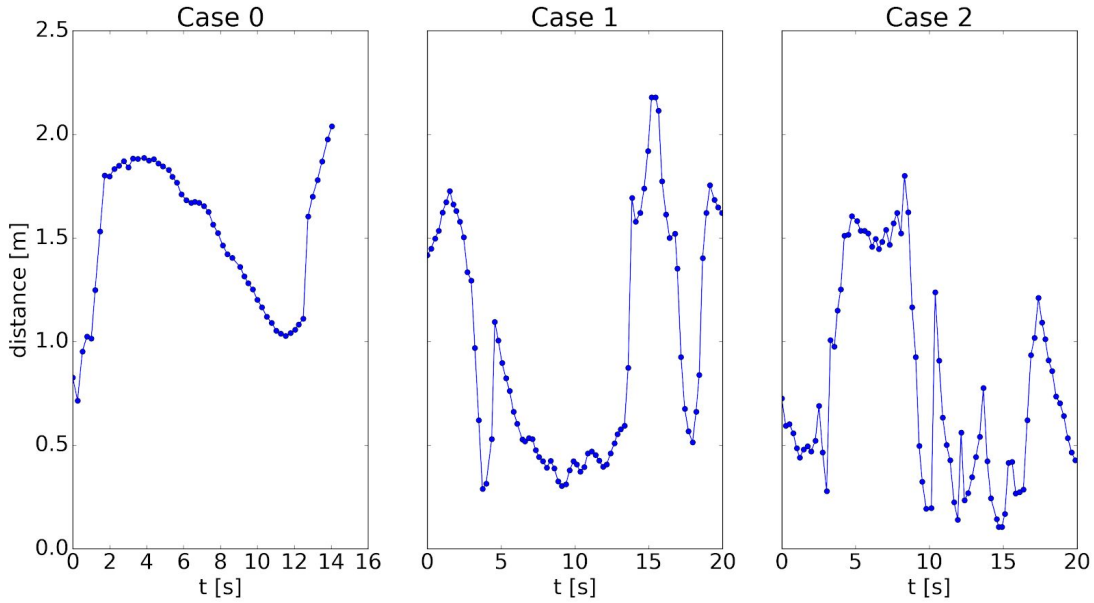


Figure 6.6.19: The closest distance between the robot's bounding shape and scanned points (due to pedestrians and static obstacles) is shown for the three cases.

6.7. Compliance Control Experiments with Qolo

Evaluation of the proposed compliance modulation of the collision forces are presented here in two folds. First assessment of the accurate online estimation of the collision forces is presented. Subsequently, a simple scenario of collision was tested with multiple initial states of the robot with an unexpected pedestrian (occluded by an obstacle) coming in the way of the robot, as shown in the figure below.

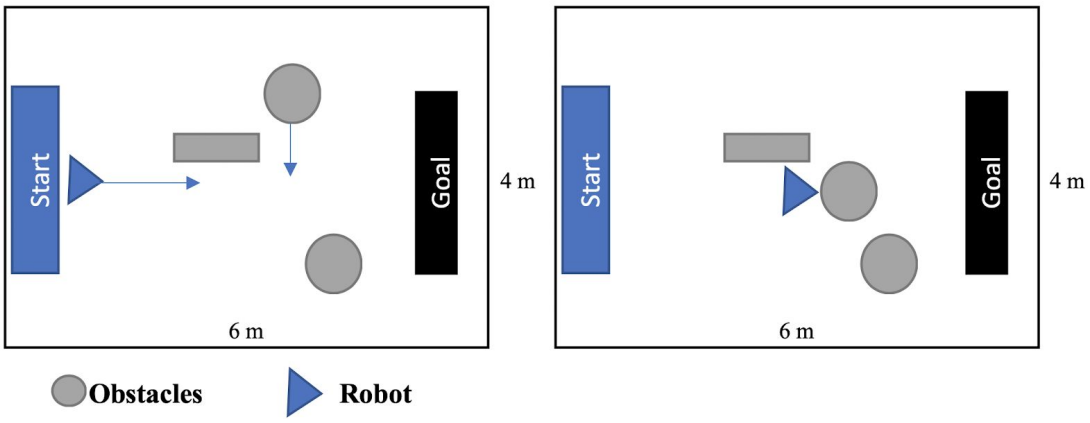
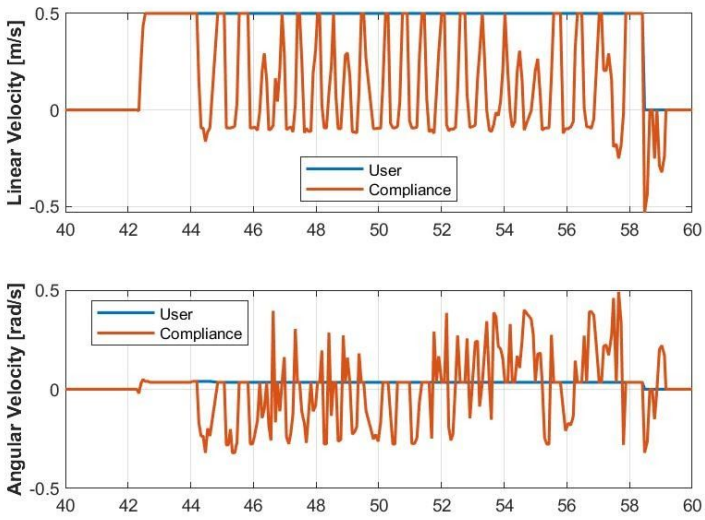


Figure 6.7.1: Diagram of evaluation scenario for collision and compliant response in the mobile robot. In the left, the initial state, and in the right the expected collision, at a set velocity.

Initially the results shown in Figure 6.7.2, present an example of the scenario where the robot is driven at a constant speed of $v = 0.5\ m/s$, $\omega = 0\ rad/s$, and a pedestrian abruptly appears in its way, creating a collision between the feet and the bumper of the robot. For this test the maximum acceptable collision force is set to be 45 N, minimum known pain threshold for the legs (shin) in human adults [Park, M.Y., 2019]). And intentionally set the robot's desired velocity constant during 16 s, thus, forcing multiple collision samples.



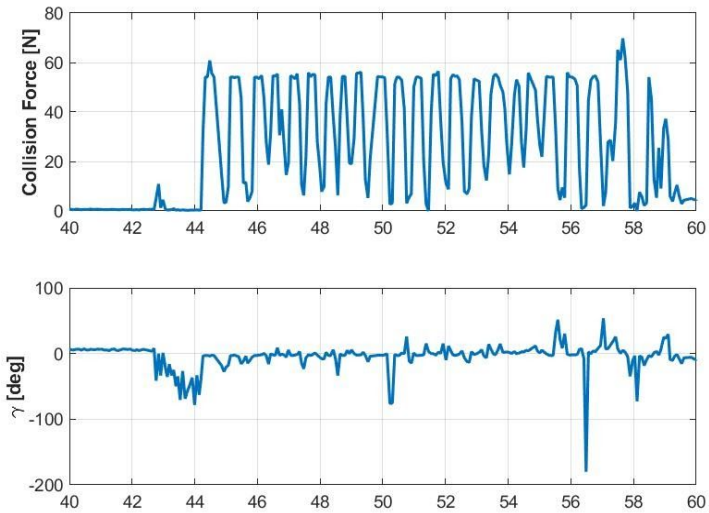


Figure 6.7.2: Experimental result for collision at 0.5 m/s with the robot and a pedestrian in laboratory settings.

The results in Figure 6.7.2 present a mean peak force for the trials of (56 N ± 3.8 N), which is approximately 11 N higher than the set limited value. In Figure 6.7.3 we zoom into the reaction of the controller at a single collision, depicting only 4 s of the interaction. Here, full velocity achieved at t=1s, and collision started at t=1.5s the maximum force observed is of 58N, which exceeds the set value by 13 N. We consider this result as part of an offset in the maximum contact force from the control loop delays and sensing delays.

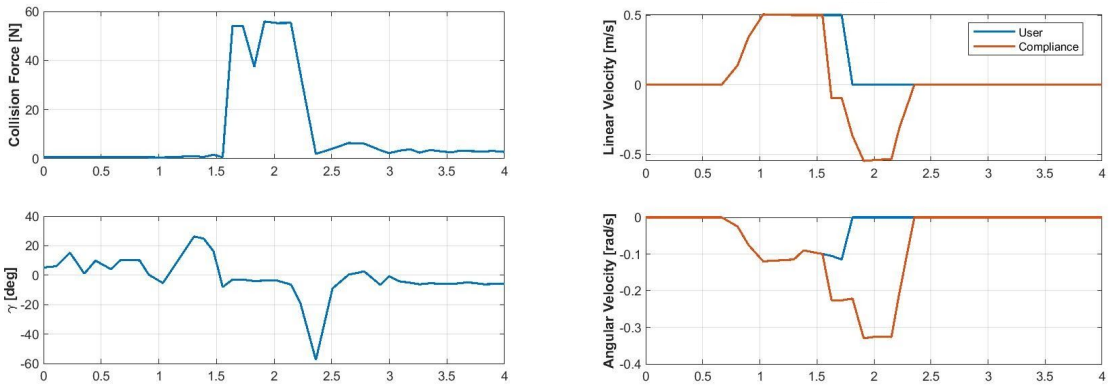


Figure 6.7.3: Single contact collision data with force magnitude, collision angle and the input commands and output resulting velocity on the robot control space (v, ω)

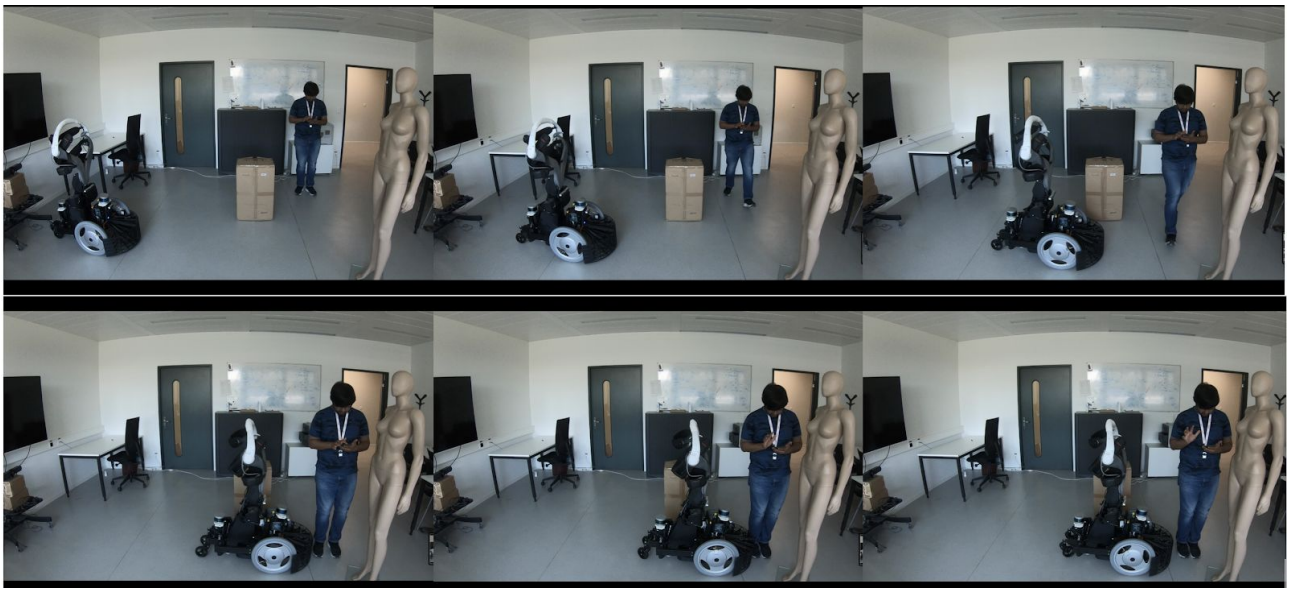


Figure 6.7.4: Timelapse of the collision with the robot and its reaction

The second experimental evaluation was recorded to observe the collision force variation at different robot speeds: $v = 0.25,\ 0.5,\ 0.75,\ 1.0\ m/s$, with $\omega = 0\ rad/s$. A set of 3 tests per initial state (desired velocity in the robot), was used to get statistical data on the test.
The results are depicted in Figure 6.7.4, where we found that collision forces remain in the same range for all initial states, between 53N and 56N for a set value of 45 N limiting collision force. Herewith, we can conclude that the compliance behavior performs as designed, limiting the output force during collisions with little variation from the velocity of the robot at the moment of collision.
The results as depicted in Figure 6.7.5 shows an error in the exerted forces with a steady state error between *9 N* upto *15 N,* which could be derived from control delays, and wheels slippery contact.

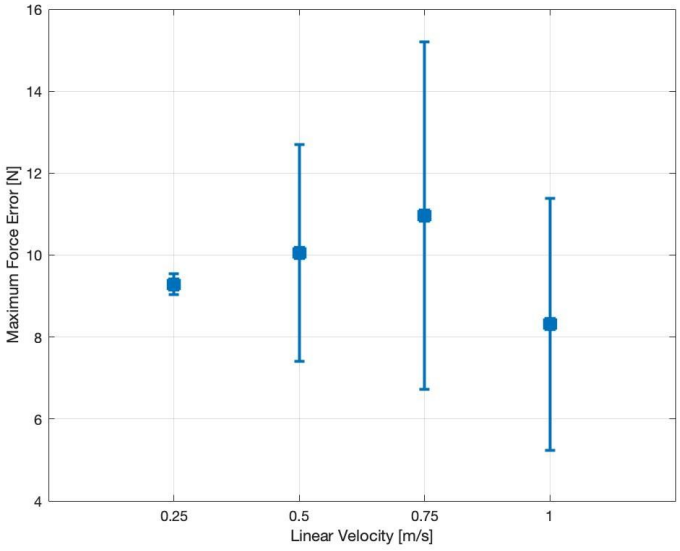


Figure 6.7.5: Mean collision forces error with respect to the set limit contact force for a set of initial state conditions in the robot varying the desired velocity prior to collisions.

References

[Andersson J.A.E., et al 2019] Joel A E Andersson, Joris Gillis, Greg Horn,James B Rawlings and Moritz Diehl{CasADi} -- {A} software framework for nonlinear optimization and optimal control. Mathematical Programming Computation, pp. 1-36, 2019.

[Alonso-Mora et al., 2013] Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P., and Siegwart, R. (2013). Optimal reciprocal collision avoidance for multiple non-holonomic robots. In Distributed autonomous robotic systems, pages 203–216. Springer.

[Arslan and Koditschek, 2016] Arslan, O. and Koditschek, D. E. (2016). Exact robot navigation using power diagrams. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 1–8. IEEE.

[Best et al., 2016] Best, A., Narang, S., and Manocha, D. (2016). Real-time reciprocal collision avoidance with elliptical agents. In 2016 IEEE International Conference on Robotics and Automation (ICRA), pages 298–305.

[Brock and Khatib, 2002] Brock, O. and Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. The International Journal of Robotics Research, 21(12):1031–1052.

[Carlson and Demiris, 2012] Carlson, T. and Demiris, Y. (2012). Collaborative control for a robotic wheelchair: evaluation of performance, attention, and workload. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 42(3):876–888.

[Chen, Y., et al, 2020] Chen, Y., Paez-Granados, D., Kadone, H., and Suzuki, K. 2020. Design of a Hands-free Navigation Interface based on Upper-Body Movements for Standing Mobility Vehicles. In *IEEE International Conference on Intelligent Robots and Systems (IROS*).

[Choset et al., 2005] Choset, H. M., Hutchinson, S., Lynch, K. M., Kantor, G., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). Principles of robot motion: theory, algorithms, and implementation. MIT press.

[Dietrich et al., 2012] Dietrich, A., Wimbock, T., Albu-Schaffer, A., and Hirzinger, G. (2012). Integration of reactive, torque-based self-collision avoidance into a task hierarchy. IEEE Transactions on Robotics, 28(6):1278–1293.

[Escande et al., 2014] Escande, A., Mansard, N., and Wieber, P.-B. (2014). Hierarchical quadratic programming: Fast online humanoid-robot motion generation. The International Journal of Robotics Research, 33(7):1006–1028.

[Fan R.E., et al 2006] Fan, R.E. , P.H. Chen, and C.J. Lin. *A Study on SMO-Type Decomposition Methods for Support Vector Machines.* IEEE Transactions on Neural Networks, 17:893–908, 2006

[Feder and Slotine, 1997] Feder, H. J. S. and Slotine, J.-J. (1997). Real-time path planning using harmonic potentials in dynamic environments. In Proceedings of International Conference on Robotics and Automation, volume 1, pages 874–881. IEEE.

[Ferguson et al., 2006] Ferguson, D., Kalra, N., and Stentz, A. (2006). Replanning with rrts. In Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., pages 1243–1248. IEEE.

[Fiorini and Shiller, 1998] Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. The International Journal of Robotics Research, 17(7):760–772.

[Fox et al., 1997] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1):23–33.

[Giese et al., 2014] Giese, A., Latypov, D., and Amato, N. M. (2014). Reciprocally-rotating velocity obstacles. In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 3234–3241.

[Hoy et al., 2015] Hoy, M., Matveev, A. S., and Savkin, A. V. (2015). Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. Robotica, 33(3):463–497.

[Huber et al., 2019] Huber, L., Billard, A., and Slotine, J.-J. (2019). Avoidance of convex and concave obstacles with convergence ensured through contraction. IEEE Robotics and Automation Letters, 4(2):1462–1469.

[Kamil et al., 2015] Kamil, F., Tang, S., Khaksar, W., Zulkifli, N., and Ahmad, S. (2015). A review on motion planning and obstacle avoidance approaches in dynamic environments. Advances in Robotics & Automation, 4(2):134–142.

[Khansari-Zadeh and Billard, 2011] Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. IEEE Transactions on Robotics, 27(5):943–957.

[Khatib, 1986] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In Autonomous robot vehicles, pages 396–404. Springer.

[Keemink, A. 2018] Keemink, A. Q. L., van der Kooij, H., & Stienen, A. H. A. (2018). Admittance control for physical human–robot interaction. *International Journal of Robotics Research*, *37*(11), 1421–1444. https://doi.org/10.1177/0278364918768950

[LaValle and Kuffner, 2001] LaValle, S. M. and Kuffner, J. J. (2001). Rapidly-exploring random trees: Progress and prospects. Algorithmic and computational robotics: new directions, (5):293–308.

[Ma et al., 2018] Ma, Y., Manocha, D., and Wang, W. (2018). Efficient reciprocal collision avoidance between heterogeneous agents using ctmat. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pages 1044–1052. International Foundation for Autonomous Agents and Multiagent Systems.

[Mansard and Chaumette, 2007] Mansard, N. and Chaumette, F. (2007). Task sequencing for high-level sensor-based control. IEEE Transactions on Robotics, 23(1):60–72.

[Michels et al., 2005] Michels, J., Saxena, A., and Ng, A. Y. (2005). High speed obstacle avoidance using monocular vision and reinforcement learning.In Proceedings of the 22nd international conference on Machine learning, pages 593–600.

[Murray et al., 2016] Murray, S., Floyd-Jones, W., Qi, Y., Sorin, D. J., and Konidaris, G. D. (2016). Robot motion planning on a chip. In Robotics: Science and Systems.

[Paez-Granados et al, 2018] Paez-Granados, D. F., Kadone, H., & Suzuki, K. (2018). Unpowered Lower-Body Exoskeleton with Torso Lifting Mechanism for Supporting Sit-to-Stand Transitions. In *IEEE International Conference on Intelligent Robots and Systems (IROS),* pp. 2755–2761. Madrid

[Park, M.Y., et al 2019] Park, M. Y., Han, D., Lim, J. H., Shin, M. K., Han, Y. R., Kim, D. H., … Kim, K. S. (2019). Assessment of pressure pain thresholds in collisions with collaborative robots. *PLoS ONE*, *14*(5), 1–12. https://doi.org/10.1371/journal.pone.0215890

[Paternain et al., 2017] Paternain, S., Koditschek, D. E., and Ribeiro, A. (2017). Navigation functions for convex potentials in a space with convex obstacles. IEEE Transactions on Automatic Control, 63(9):2944–2959.

[Rasekhipour et al., 2016] Rasekhipour, Y., Khajepour, A., Chen, S.-K., and Litkouhi, B. (2016). A potential field-based model predictive path-planning controller for autonomous road vehicles. IEEE Transactions on Intelligent Transportation Systems, 18(5):1255–1267.

[Rauscher et al., 2016] Rauscher, M., Kimmel, M., and Hirche, S. (2016). Constrained robot control using control barrier functions. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 279–285. IEEE.

[Redmon et al., 2016] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 779-788).

[Rimon and Koditschek, 1991] Rimon, E. and Koditschek, D. E. (1991). The construction of analytic diffeomorphisms for exact robot navigation on star worlds. Transactions of the American Mathematical Society, 327(1):71–116.

[Rimon and Koditschek, 1992] Rimon, E. and Koditschek, D. E. (1992). Exact robot navigation using artificial potential functions. Departmental Papers (ESE), page 323.

[Schlegel, 1998] Schlegel, C. (1998). Fast local obstacle avoidance under kinematic and dynamic constraints for a mobile robot. In Proceedings. 1998 IEEE/RSJ International Conference on Intelligent Robots and Systems. Innovations in Theory, Practice and Applications (Cat. No. 98CH36190), volume 1, pages 594–599. IEEE.

[Stasse et al., 2008] Stasse, O., Escande, A., Mansard, N., Miossec, S., Evrard, P., and Kheddar, A. (2008). Real-time (self)-collision avoidance task on a hrp-2 humanoid robot. In 2008 ieee international conference on robotics and automation, pages 3200–3205. IEEE.

[Van Den Berg et al., 2011] Van Den Berg, J., Guy, S. J., Lin, M., and Manocha, D. (2011). Reciprocal n-body collision avoidance. In Robotics research, pages 3–19. Springer.