Constraint-based Whole-Body-Control of Mobile Manipulators in Human-Centered Environments

Matthias Stueben, Alwin Hoffmann, Wolfgang Reif
Institute for Software & Systems Engineering, University of Augsburg, Augsburg, Germany
{stueben, hoffmann, reif}@isse.de

***Abstract*—In this work, we describe a ROS-based method for whole-body control (WBC) of mobile manipulators in the context of safe human-robot interaction. Our method is based on cyclic quadratic programming (QP) with a set of simultaneously active tasks that define constraints. The importance of different tasks is captured through priorities and weights. Robot behavior can be changed at run-time by re-configuring the active tasks through ROS interfaces. We evaluate the suitability of our method for safe human-robot collaboration in a Gazebo simulation. We show that our method lets the mobile manipulator perform evasive motions while staying consistent with other tasks if possible. At the same time, self-collisions and static obstacles are avoided. If a given safety threshold is crossed, the robot comes to a safe stop. Operation continues once the distance is high enough again.**

***Index Terms*—Collision Avoidance, Real-time Robot Control, Human-Robot Interaction, Whole-Body Control**

I. Introduction

Mobile manipulators are an increasingly popular and promising form of robot system, as they combine the capabilities of mobile platforms and manipulators. Their potential to move within the environment as well as manipulate it makes them a suitable candidate for a wide range of applications, including elderly care [1]–[3], as personal robot [4], and as an assistant in hospitals [5] or in production [6]–[8]. These applications have in common that the robot has to operate in human-centered environments. Hence, the robot often has to navigate in less structured environments and has to evade humans in a safe and expected manner.

Traditional safety concepts such as strictly separated workspaces are not applicable anymore. New approaches to ensure safety are needed. In addition to preventing collisions, robots should also allow for naturally appearing interactions which need to follow the users' expectations. Thus, simply stopping the robot abruptly as soon as an obstacle comes close is not practicable. The redundant kinematics typically used in such applications allow more complex evasion maneuvers.

We developed a WBC framework for mobile manipulators in ROS [9]. This framework allows for interaction tasks which exploits the robot's redundant kinematics. Tasks can be defined to check for and to avoid collisions and, thus, to ensure a safe operation. These tasks form an optimization problem which is solved in every control cycle to obtain a fast and appropriate reaction.

Hence, the contribution of this paper is a constraint-based robot control framework in ROS which (1) enables WBC for redundant mobile manipulators, which (2) allows for specifying naturally appearing and fluent interaction tasks, and which (3) is designed especially for safe human-robot-interactions. The benefits of the framework were evaluated in simulation with Gazebo.

The structure of this paper is as follows: Related work is discussed in Section II. The concept of our robot control framework and its available tasks are introduced in Section III, the implementation is described in Section IV. An evaluation with respect to safe human-robot-interaction is given in Section V. Section VI concludes the paper and gives an outlook to future work.

II. State of the Art and Related Work

Safe control strategies have been created for safe human-robot collaboration, e.g. [10], [11]. However, these typically consider a single, static control strategy for fixed-base robots. In contrast, we consider mobile robots where the appropriate safety strategy might change depending on context.

Control of redundant mobile manipulators has been an active area of research over the past years. Many approaches consider navigation and manipulation as separate problems, so that only the arm or only the platform moves at any time, e.g. [12], [13]. This however limits the possibilities to make use of the robot's redundancy.

Therefore, WBC methods have been developed, for mobile manipulators and other types of redundant robots, which control all degrees of freedom at the same time. Several works plan and execute trajectories in this manner, typically based on specialized inverse kinematics, but do not consider reactive behavior [14]–[16]. The factors influencing the choice of robot position are also statically encoded in these works and can not be easily changed at run-time to adapt the robot behavior.

A prominent work which controls redundant robots reactively by considering a set of sub-tasks is the iTaSC framework [17], [18], with a Jacobian pseudoinverse-based control scheme. Other works have adopted methods inspired by this, e.g. [19].

The Stack-of-Tasks [20] and eTaSL/eTC [21] use QP to find control signals instead. Our method of calculating joint velocities is based on these works. The novelty of the work described here lies in the incorporation of dynamic obstacles described by geometric primitives, the application to safe human-robot interaction with mobile manipulators as well as its integration into the ROS ecosystem.

III. CONCEPT

Before any robot motions can be calculated, the intended behavior has to be formalized which can be complex in itself. To simplify this, the control problem is broken down into small, independent tasks. Here, the word *task* is used to refer to any requirement on the robot motion. This includes, for example, moving the end-effector to a target position, but also staying away from joint limits and avoiding obstacles.

Tasks often have conflicting requirements, which is why priorities and weights are required to resolve them. Using this information, a QP problem is formed and solved in every control cycle to find a control signal that best fulfills the tasks. In our case, the robot control signal consists of joint velocities $\dot{\mathbf{q}} \in \mathbb{R}^n$ with n being the number of controlled joints. Fig. 1 shows an overview of our control system which is described in further detail below.

All information about the current state of the world is encapsulated in the robot model and the environment model. The environment model contains a set of objects, each of which consists of a name, a type identifier, its pose in Cartesian space, and an optional *collision geometry*. The collision geometry is described through geometric primitives such as boxes, spheres, and cylinders. These objects can be created through various methods of perception and scene analysis. Our interface is independent of any specific perception method.

The robot model contains all relevant information about the robot itself. It encapsulates static information about the kinematic structure, as well as current state information, such as the current joint angles $\mathbf{q} \in \mathbb{R}^n$.

The main component of the robot model is the kinematic tree of the robot's structure. Robot links can be given a collision geometry in the form of geometric primitives. The robot model provides the geometric Jacobian $\mathbf{J_x(q)}$ for any point $\mathbf{x}$ on the robot structure. A further part of the robot model are the limits for joint positions $\mathbf{q_{min}}, \mathbf{q_{max}} \in \mathbb{R}^n$ and joint velocities $\dot{\mathbf{q}}_{\mathbf{min}}, \dot{\mathbf{q}}_{\mathbf{max}} \in \mathbb{R}^n$. The vector $\boldsymbol{\omega}_\mathbf{q} \in \mathbb{R}^n$ defines a weight for each controlled robot joint. Motions of joints with higher weights cause higher cost in the optimization. We use $\boldsymbol{\omega}_\mathbf{q} = 1$, effectively not using the weights. They could however be used, for example, to introduce a preference of arm motions over platform motions.

A. Tasks

The intended robot behavior is described using a set of tasks $T = \{t_1, \ldots, t_s\}$. Each task t_i defines the following components:

- $d_i \in \mathbb{N}^+$ dimension of the task.
- $\mathbf{J_i(q)} \in \mathbb{R}^{d_i \times n}$: the task Jacobian which defines the influence of joint movements on the task.
- A controller calculating lower and upper bounds on the task. We refer to the current output as $\mathbf{b_i} \in \mathbb{R}^{d_i}$ for the lower bounds and $\mathbf{B_i} \in \mathbb{R}^{d_i}$ for the upper bounds.
- $p_i \in \mathbb{N}^+$ priority level of the task.
- $\boldsymbol{\omega}_\mathbf{i} \in \mathbb{R}^{d_i}$: task weights, which define the relative importance of individual task components. A weight of zero disables the task component.

Examples of tasks include moving the robot towards a target joint position, moving a robot link to a Cartesian pose, or avoiding obstacles. Individual task types are described in further detail in section III-C. In the following subsection, we describe how joint velocities are calculated from the defined tasks.

B. Formulation of the Optimization Problem

In its most basic form, the QP problem for a set of s tasks $T = \{t_1, \ldots, t_s\}$ can be formulated as follows:

$$\begin{aligned} \min_{\dot{\mathbf{q}}} \quad & \dot{\mathbf{q}}^T \mathbf{A} \dot{\mathbf{q}} \\ \text{s. t.} \quad & \mathbf{b} \leq \mathbf{J}\dot{\mathbf{q}} \leq \mathbf{B} \\ & \dot{\mathbf{q}}_{\mathbf{min}} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\mathbf{max}} \end{aligned} \tag{1}$$

where $\mathbf{J} = [\mathbf{J_1}^T \ldots \mathbf{J_s}^T]^T$ consists of the vertically stacked task Jacobians of all tasks, $\mathbf{A} = \mathrm{diag}(\boldsymbol{\omega}_\mathbf{q})$ is the diagonal matrix of the axis weights, $\mathbf{b} = (\mathbf{b_1}^T, \ldots, \mathbf{b_s}^T)^T$ and $\mathbf{B} = (\mathbf{B_1}^T, \ldots, \mathbf{B_s}^T)^T$ are the stacked vectors of lower and upper bounds of all tasks.

All tasks are considered equally with this formulation which is not always desirable. The optimization fails in cases where no solution fulfilling all tasks can be found. This is unacceptable for robot control. Thus, we extend the formulation to allow for tasks to deviate from their bounds. This deviation should only occur when necessary, and for some tasks it is more acceptable to deviate from their bounds than for others. For example, joint limits must usually not be violated under any circumstance, while it can be more acceptable for the Cartesian end-effector position to deviate from its target. Of course, this always depends on the specific application. To define which tasks need to observe their bounds more strictly than others, we introduce priorities and weights to the QP problem. The full formulation then is as follows:

$$\begin{aligned} \min_{\gamma} \quad & \gamma^T \boldsymbol{\eta} \gamma \\ \text{s. t.} \quad & \mathbf{b} - \boldsymbol{\chi} \leq \mathbf{J}\dot{\mathbf{q}} \leq \mathbf{B} + \boldsymbol{\chi}, \\ & \dot{\mathbf{q}}_{\mathbf{min}} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\mathbf{max}} \end{aligned} \tag{2}$$

The optimization variable $\gamma = (\dot{\mathbf{q}}^T, \boldsymbol{\chi}^T)^T$ is extended to include slack variables $\boldsymbol{\chi} = (\boldsymbol{\chi_1}^T, \ldots, \boldsymbol{\chi_s}^T)^T$ with $\boldsymbol{\chi_i} \in \mathbb{R}^{d_i}$ for each task which captures the deviation of the task from its bounds. In this way, solutions which do not fulfill all tasks completely become feasible. The matrix $\boldsymbol{\eta} = \mathrm{diag}((\boldsymbol{\omega}_\mathbf{q}{}^T, \boldsymbol{\omega}^T)^T)$ is a diagonal matrix containing the axis weights $\boldsymbol{\omega}_\mathbf{q}$ and the task weights $\boldsymbol{\omega} = (\epsilon^{p_1}\boldsymbol{\omega_1}^T, \ldots, \epsilon^{p_s}\boldsymbol{\omega_s}^T)^T$. It is used to specify how much each axis motion and a deviation of a task from its bounds is penalized. In some situations, such as tasks with variables of different orders of magnitude, hand-tuning the weights may be required to maintain the correct priorities.

We implement task priorities based on the observation that giving a task lower priority is equivalent to taking the limit of the task weight towards zero [18]. In practice, choosing a "very small" weight ϵ is a sufficient approximation of this limit.

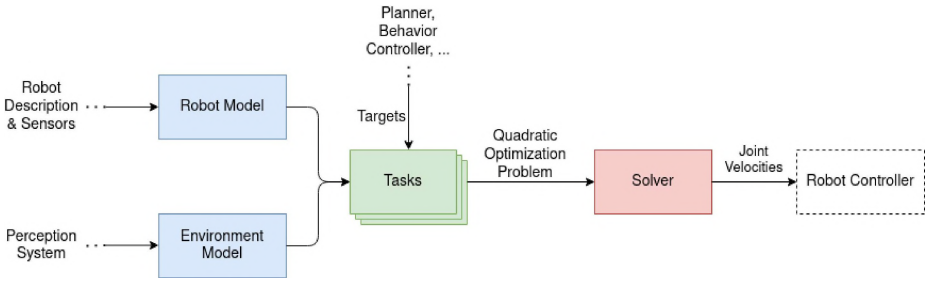


Fig. 1: Architectural overview of the control system.

This generalizes to multiple levels of priority by weighting a task t_i of priority p_i with ϵ^{p_i}. In our implementation, we use $\epsilon = 1e^{-6}$. Alternatively, exact solutions can be found by using specialized solvers [22]. However, it has also been shown that the approximation through weights can perform very well in practice [23].

C. Implemented Task Types

The task types we have implemented are described here. Additional types can be implemented by defining the task Jacobian and the controllers of upper and lower bounds.

1) Cartesian Pose Task: This task moves a control point $\mathbf{c}$ on the robot structure to a given Cartesian target pose $\mathbf{t}$. The task Jacobian $\mathbf{J_i}$ is the geometric Jacobian of the control point. This task has $d_i = 6$ dimensions, consisting of the 3 translational and 3 rotational degrees of freedom of Cartesian space. Since this task is supposed to generate the Cartesian twist which leads the control point to the target pose, and not a range of permissible motions, upper and lower bounds of this task are set to the same value. They are calculated according to the control law

$$\mathbf{b_i} = \mathbf{B_i} = K \begin{pmatrix} \mathbf{x_t} - \mathbf{x_a} \\ \mathbf{R_a} \theta \hat{\boldsymbol{\omega}}_\mathbf{t}^\mathbf{a} \end{pmatrix}$$

where $\theta \hat{\omega}_t^a$ is the rotation between target and actual orientation expressed in axis-angle format, with θ being the angle between the two orientations and $\hat{\boldsymbol{\omega}}_\mathbf{t}^\mathbf{a}$ the unit vector defining the rotation axis. $\mathbf{R_a}$ denotes the rotation matrix of the control point. $\mathbf{x_t}$ and $\mathbf{x_a}$ are the target and actual position vectors of the control point. K is a gain parameter set by the user.

2) Joint Position Task: This task type moves the robot axes towards a target position $\mathbf{q_t}$. Since this task completely operates in joint space, the task Jacobian is the Identity matrix $\mathbf{J_i} = \mathbf{I_n}$. As in the *Cartesian Pose Task*, upper and lower bounds are identical. They are calculated according to the proportional control law

$$\mathbf{b_i} = \mathbf{B_i} = K(\mathbf{q_t} - \mathbf{q})$$

where K is a user-provided gain parameter.

3) Joint Limit Avoidance: The purpose of this task type is to ensure the robot axes do not move beyond their limits given by $\mathbf{q_{min}}$ and $\mathbf{q_{max}}$. Therefore, joint movements will be restricted if they approach those limits.

A plot of the bounds generated by this task can be seen in Fig. 2. The task Jacobian is the identity matrix $\mathbf{J_i} = \mathbf{I_n}$. The task dimension is equal to the number of controlled joints $d_i = n$.

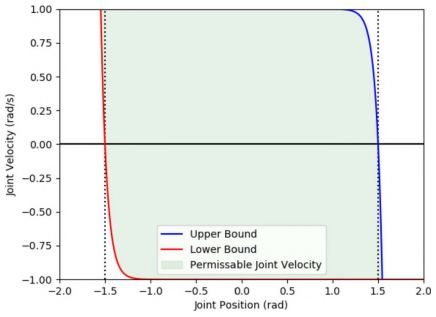


Fig. 2: Example plot of bounds calculated by the *Joint Limit Avoidance* task with $\alpha = 3$, $q_{max} = 1.5$, $q_{min} = -1.5$, $\dot{q}_{max} = 1$, $\dot{q}_{min} = -1$, $D = 0.2$.

Upper and lower bounds are calculated for each joint $j \in \{1, \ldots, n\}$ by first calculating a scaling factor f_j depending the current joint position $\mathbf{q}_j$:

$$f_j = 1 - \frac{1}{e^{\alpha(1 - \frac{D - (\mathbf{q_{max,j}} - \mathbf{q_j})}{D})}} \tag{3}$$

where D and α are parameters influencing the shape of the curve. Bounds for the joint velocities are then calculated by scaling the maximum joint velocity with this factor:

$$\mathbf{B_{i,j}} = f_j \dot{\mathbf{q}}_\mathbf{max,j} \qquad \mathbf{b_{i,j}} = f_j \dot{\mathbf{q}}_\mathbf{min,j} \tag{4}$$

4) Obstacle Evasion: This task aims to actively increase the distance from a robot control point to obstacles. This is based on the calculation of a repulsive vector for the robot control point $\mathbf{c}$ from the set of obstacle points O from the environment model. The repulsive vector generates a maximum velocity of v_{max}. The vector reaches zero when a target distance d_t is kept to all obstacles. Our method of generating the repulsive vector is based on the work of Flacco et al. [24].

The Jacobian for this task type is the geometric Jacobian of the controlled robot link. For each obstacle point $\mathbf{o} \in O$, a repulsive vector $\mathbf{R}$ is calculated for the robot control point $\mathbf{c}$:

$$\begin{aligned} \mathbf{R}(\mathbf{o}, \mathbf{c}) &= v(\mathbf{o}, \mathbf{c}) \frac{\mathbf{c} - \mathbf{o}}{||\mathbf{c} - \mathbf{o}||} \\ v(\mathbf{o}, \mathbf{c}) &= \frac{v_{max}}{1 + e^{(((2||\mathbf{c}-\mathbf{o}||/d_t)-1)\alpha))}} \end{aligned} \tag{5}$$

The function v controls the magnitude of the vector. α is a parameter influencing the slope of the curve. The generated magnitude of the repulsive vector is illustrated in Fig. 3.

The total repulsive vector $\mathbf{R_{all}}$ is formed by taking the sum $\mathbf{R_T}$ of all individual repulsive vectors as direction and scaling it to the magnitude generated by the closest obstacle $\mathbf{o_{min}}$:

$$\mathbf{R_T}(O,\mathbf{c}) = \sum_{\mathbf{o}\in O} \mathbf{R}(\mathbf{o},\mathbf{c})$$
$$\mathbf{R_{all}}(O,\mathbf{c}) = v(\mathbf{o_{min}},\mathbf{c})\frac{\mathbf{R_T}(O,\mathbf{c})}{||\mathbf{R_T}(O,\mathbf{c})||} \tag{6}$$

Finally, each component of the task bounds $\mathbf{b}_{i,j}, \mathbf{B}_{i,j}, j \in \{1,\dots,6\}$ is calculated from $\mathbf{R_{all}}$ as follows:

$$\mathbf{b}_{i,j} = \begin{cases} -\infty, & \mathbf{R_{all,j}} \leq 0 \\ \mathbf{R_{all,j}}, & \text{else} \end{cases}$$
$$\mathbf{B}_{i,j} = \begin{cases} +\infty, & \mathbf{R_{all,j}} \geq 0 \\ \mathbf{R_{all,j}}, & \text{else} \end{cases} \tag{7}$$

5) Obstacle Avoidance: This task type creates bounds that prevent the chosen robot link from moving too close to any obstacle. The difference to the *Obstacle Evasion Task* is that this type only prevents moving too close to obstacles, but does not attempt to increase the distance.

The set of obstacles O is taken from the environment model. The task Jacobian is the geometric Jacobian of the controlled robot link. The geometries of both the controlled robot link and the obstacles are represented as geometric primitives. For each obstacle $\mathbf{o} \in O$ we calculate the closest point on the surface of the obstacle $\mathbf{c_o}$ and on the surface of the robot link $\mathbf{c_r}$ using the Gilbert-Johnson-Keerthi algorithm [25].

To find which directions to put bounds on, we calculate the distance gradient

$$\nabla d(\mathbf{c_o},\mathbf{c_r}) = \begin{pmatrix} \nabla_{lin} d(\mathbf{c_o},\mathbf{c_r}) \\ \nabla_{rot} d(\mathbf{c_o},\mathbf{c_r}) \end{pmatrix}$$

The linear part of the gradient is given by

$$\nabla_{lin} d(\mathbf{c_o},\mathbf{c_r}) = \frac{\mathbf{c_r} - \mathbf{c_o}}{||\mathbf{c_r} - \mathbf{c_o}||}$$

and the gradient for rotational motion is given by

$$\nabla_{rot} d(\mathbf{c_o},\mathbf{c_r}) = \begin{pmatrix} (\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T \times \mathbf{r}) \circ \nabla_{lin} d(\mathbf{c_o},\mathbf{c_r}) \\ (\begin{pmatrix}0 & 1 & 0\end{pmatrix}^T \times \mathbf{r}) \circ \nabla_{lin} d(\mathbf{c_o},\mathbf{c_r}) \\ (\begin{pmatrix}0 & 0 & 1\end{pmatrix}^T \times \mathbf{r}) \circ \nabla_{lin} d(\mathbf{c_o},\mathbf{c_r}) \end{pmatrix}$$

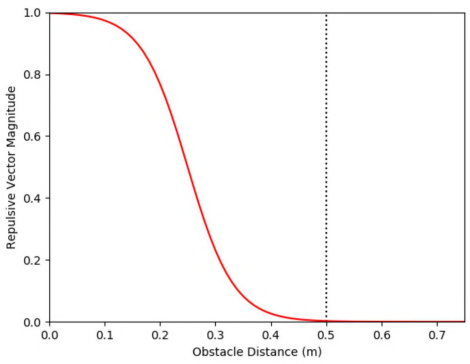


Fig. 3: Magnitude v of a repulsive vector $\mathbf{R}(\mathbf{o},\mathbf{c})$ for the *Obstacle Evasion* task with $\alpha = 6$, $v_{max} = 1$, $d_t = 0.5$.

If the robot link rotates around its center $\mathbf{x_r}$, the closest surface point $\mathbf{c_r}$ rotates with radius vector $\mathbf{r} = \mathbf{x_r} - \mathbf{c_r}$. We then calculate the tangential velocities caused by rotations around the three principal axes, and use them to calculate the distance gradient for rotational motions.

It must be noted that this gradient only models the distance between the chosen points on the object surface $\mathbf{c_r}$ and $\mathbf{c_o}$, i.e. the closest points in the current configuration. This is not necessarily equal to the total minimum distance between the two collision geometries, as these points may no longer be the closest points when the objects move. However, accurate distance gradient computation between geometric bodies is a much more complex problem and this approach performs well in practice. Should problems arise due to this simplification, specialized methods can be used, e.g. [26].

Next, the individual bounds for each obstacle $\mathbf{o} \in O$ are calculated. Each component of the 6-dimensional bounds vectors $\mathbf{b}_i(\mathbf{o})$ and $\mathbf{B}_i(\mathbf{o})$ is calculated as follows, for $j \in \{1,\dots,6\}$:

$$f(\mathbf{o}) = 1 - \frac{1}{e^{-\alpha \frac{d_{min} - ||\mathbf{c_r} - \mathbf{c_o}||}{d_{min}}}}$$
$$\mathbf{b}_{i,j}(\mathbf{o}) = \begin{cases} -\infty, & \nabla_j d(\mathbf{c_o},\mathbf{c_r}) \leq 0 \\ -f(\mathbf{o})(\nabla_j d(\mathbf{c_o},\mathbf{c_r}))^{-1}, & \text{else} \end{cases}$$
$$\mathbf{B}_{i,j}(\mathbf{o}) = \begin{cases} \infty, & \nabla_j d(\mathbf{c_o},\mathbf{c_r}) \geq 0 \\ -f(\mathbf{o})(\nabla_j d(\mathbf{c_o},\mathbf{c_r}))^{-1}, & \text{else} \end{cases} \tag{8}$$

α is a parameter that allows the user to influence the shape of the curve. Finally, the total bounds of this task for the entire set of obstacles O are then chosen by taking the component-wise strictest bounds of all obstacles:

$$\mathbf{b}_{i,j} = \max_{\mathbf{o}\in O} \mathbf{b}_{i,j}(\mathbf{o}) \qquad \mathbf{B}_{i,j} = \min_{\mathbf{o}\in O} \mathbf{B}_{i,j}(\mathbf{o}) \tag{9}$$

6) Self-Collision Avoidance: This task type prevents motions that would cause the robot to collide with itself. Each instance of this task considers the distance between the collision geometries of two robot control points.

The task Jacobian expresses the motion of the control point relative to the point to be avoided. Since the point to be avoided is also part of the kinematic chain, $\mathbf{J_i} = \mathbf{J_{CP}} - \mathbf{J_B}$ is used, where $\mathbf{J_{CP}}$ and $\mathbf{J_B}$ are the geometric Jacobians of the controlled and the avoided robot control points, respectively.

The calculations of the upper and lower bounds uses the same method as the *Obstacle Avoidance Task*. The only difference is that *Self-collision Avoidance Task* only considers one object to avoid, but the same method is still applicable. If self-collisions between multiple robot links have to be avoided, multiple task instances have to be created (which is fully supported by our implementation).

7) Velocity Limiting: This task limits all robot motions if an obstacle of a given type comes too close. This can be used, for example, to stop the robot if a person is in the immediate proximity of the robot and even evasive motions can not be considered safe anymore. The task Jacobian is the identity matrix. Bounds on axis motions are calculated depending on the minimum obstacle distance d_{min} using the following formulas:

$$
\begin{aligned}
f(d_{min}) &= \frac{1}{e^{-\alpha \max(d_{min}-D,0)}} - 1 \\
\mathbf{b}_i &= f(d_{min})\dot{\mathbf{q}}_{min} \\
\mathbf{B}_i &= f(d_{min})\dot{\mathbf{q}}_{max}
\end{aligned}
\tag{10}
$$

The parameter α defines the slope of the curve and D is the distance at which the robot has to stop all movements.

Other task types which may prove useful include torque and acceleration constraints. At the moment, we rely on the low-level robot controller to track the desired reference velocity appropriately. If acceleration or torque spikes become a problem, these can be added as separate task types. For a torque-task, the robot model would have to be extended to contain joint torque data, then the constraints can easily be expressed using the task bounds. Acceleration constraints can be realized by creating bounds that keep the target velocity within a given range of the current velocity.

IV. Implementation

Our approach is implemented in a combination of ROS1 and ROS2 components [9]. Our QP-based controller is implemented as a ROS2 node, while the robot hardware controllers currently necessitate the use of ROS1.

All interaction with our system is available through standard ROS interfaces. Tasks can be activated and deactivated through ROS services. Targets for tasks can be updated through ROS topics, and each task can publish feedback information in the same way. The robot description is read from URDF files. Initial task configuration is described in YAML files.

Discrete behavior control can be implemented by updating or switching the active tasks according to the feedback provided by the tasks or outside information. The ROS interfaces make it easy to implement discrete behavior control. We use Python scripts, but many behavior control frameworks, e.g. state machines or behavior trees, can be integrated easily.

The QP described above is performed by the solver qpOASES [27]. Specifically, we use the extended variant of the online active set strategy for varying matrices [28]. We use the Flexible Collision Library [29] for geometric calculations and the Kinematics and Dynamics Library [30] for kinematic calculations.

We implemented our approach on a mobile manipulator consisting of a holonomic Neobotix MPO-700 [31] mobile base equipped with a Schunk LWA 4P [32] arm. The robot is depicted in Fig. 4. The holonomic base is modeled by two virtual translational joints for x and y position, and a virtual rotational joint for the heading. Combined with the 6 joints of the arm, we have 9 controlled joints in total. For efficiency reasons, evaluation was performed in a Gazebo [33] simulation.

V. Evaluation

To evaluate the approach, we have created a simulation based on Gazebo. The robot responses to dynamic obstacles and the required calculation time are described below. Table I shows the tasks used for the evaluation and their priorities. Task parameters were chosen empirically.

Fig. 4: The robot platform, using a Neobotix MPO-700 platform and Schunk LWA arm. The arm is shown here with a custom sensor casing unrelated to the described research.

A. Reactions to dynamic obstacles

The robot attempts to follow a "figure eight"-like shape with its gripper in an environment constrained by static obstacles, i.e. several pieces of furniture, shown in Fig. 5a. Two types of human interference are simulated: a walking person (Fig. 5b); and a hand, approximated as a sphere, moving above the table surface (Fig. 5c).

The person walks with a random speed of up to 1.48 m/s, an average convenient indoor walking speed [34]. The direction of approach and the turning point are also chosen randomly. The person may also randomly choose to wait for a random amount of time up to 3 seconds. We have divided the evaluations with walking people into four categories: One where a person approaches the robot from behind (i.e. the top of the image in Fig. 5a) and then turns around, and three others where a person walks by the robot from left to right or the reverse with varying distances. The distances we have used are 1.2 m, 2.2 m and 3 m from the center of the table.

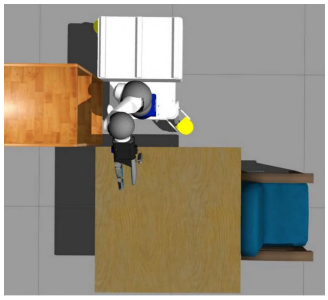
The simulated hand moves above the table at a random speed of up to 0.6 m/s, an average peak velocity for reaching motions of a seated person [35]. Motion follows either a straight or circular path. Minimum distance and direction are chosen at random.

Thus we have evaluated five scenarios, four with a walking person and one with a moving hand. Each of the scenarios has been recorded over 5 minutes. Results are presented below. A video showing examples is available at `https://video.isse.de/mobile-manipulator`.

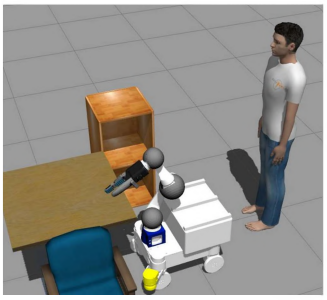
1) Reaction to a Hand: Fig. 6 shows the reaction to a straight motion. It can be seen how the robot attempts an evasive motion at first, but stops once the distance falls below the safety limit. Reaction to a circular motion of the hand is shown in Fig. 7.

TABLE I: Active tasks during the evaluation.

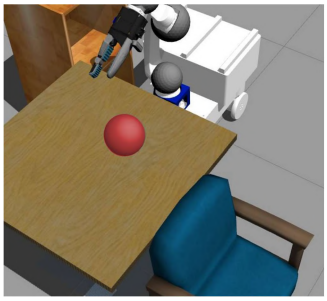
Prio.	Tasks
0	Joint Limits, Obstacle Avoidance (Base, Elbow, Gripper), Self Collision Avoidance (Gripper, Elbow)
1	Obstacle Evasion (Base, Gripper)
2	Cartesian Position (Gripper), Joint Position



(a) Seen from above.



(b) Person approaching.



(c) Simulated Hand, approximated as a sphere.

Fig. 5: Screenshots of the simulated scenario.

Here, the robot performs an evasive motion and no safety stop is needed since the distance does not fall below the safety stop distance. Fig. 8 shows that the gripper velocities scale down according to the *Velocity Limiting Task*, with a small delay.

2) Reaction to a Walking Person: Fig. 9a shows a heat map of the positions of gripper and platform during the evaluation where a person passes with a distance of 3 m, which is expected to have no influence on the robot. Accordingly, the gripper follows the prescribed shape without any distortions. The platform moves in almost the same pattern with only a small variation due to the static obstacles.

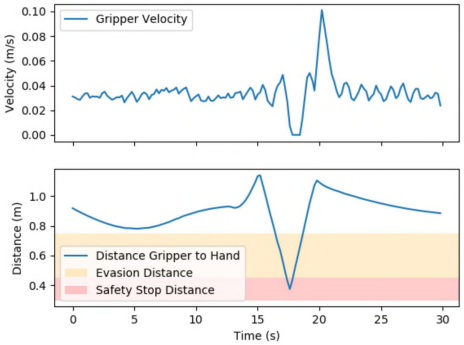


Fig. 6: Gripper velocity in response to a hand approaching in a straight line.

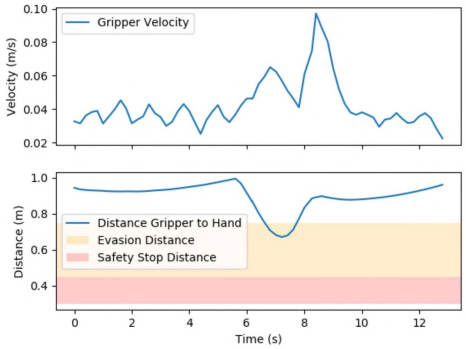


Fig. 7: Gripper velocity in response to a circular hand motion in proximity of the gripper.

Results from a person walking by with a distance of 2.2 m are shown in Fig. 9b. It can be seen that the platform tries to evade the person and thus moves less predictably. However, the gripper still follows its path with very little disturbance.

When the person moves even closer, as shown in Fig. 9c, the platform stops completely on several occasions. This is indicated by isolated bright spots in the heat map, meaning that the platform stayed in the same position over a longer period. The same isolated spots can be seen on the gripper heat map, but it still stays close to the prescribed path.

When the person approaches from behind (Fig. 9d), the platform heat map shows a similar pattern as before, but the gripper shows a larger deviation from its path. This can be explained by the fact that the robot has fewer possibilities to perform evasive motions towards the table without disturbing the gripper task, compared to the sideways motions required in the previous scenarios.

Fig. 10 shows the recorded velocities of the platform in relation to the distance from the person. Also shown is the scaling factor of the corresponding velocity limiting task. Ideally, velocities should become zero as soon as this factor becomes zero. It can be seen that, while the velocities generally follow the scaling factor and reach zero eventually, there is some delay before the robot comes to a stop.

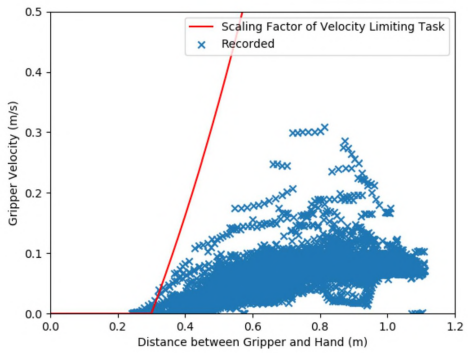
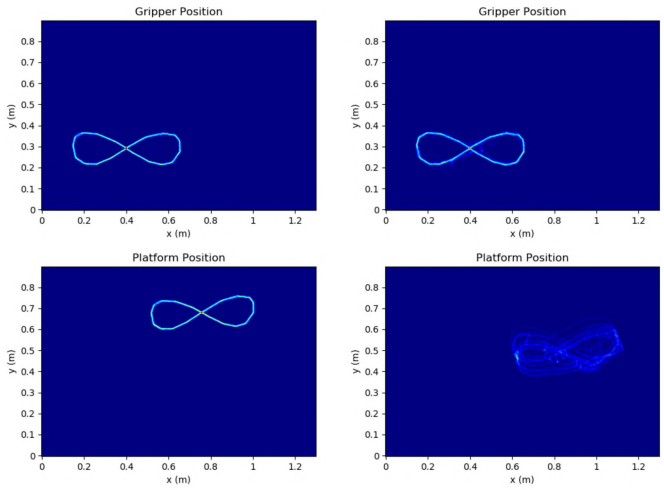
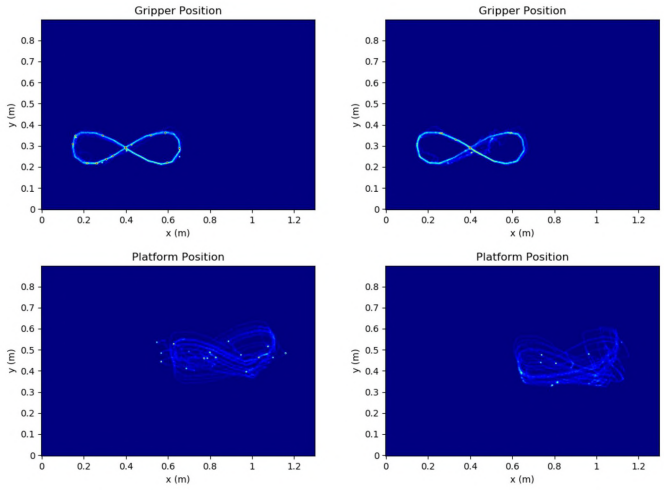


Fig. 8: Relation between gripper velocity and distance to a hand.



(a) Person walking by at $y = 3\ m$. (b) Person walking by at $y = 2.2\ m$.



(c) Person walking by at $y = 1.2\ m$. (d) Person approaching from behind with randomized distances.

Fig. 9: Heat maps of gripper and platform position with a person approaching at various distances.

This can be explained by the various delays inherent to the system, such as data transmission and hardware limitations. The conflict between obstacle evasion (i.e. increasing velocity) and velocity limiting in the presence of obstacles also contributes to this effect. Further experimentation, especially on real hardware, is needed here.

B. Timing

We have recorded the required calculation time over 2500 iterations on a PC with an Intel i5-4570 CPU, with the same 10 active tasks as above, 4 static obstacles, and the simulated hand described above. Fig. 11 shows the results. On average, solving takes 18 ms, with a maximum of 44 ms. The calculation of the bounds takes comparatively little time with only small deviations, while the QP solving introduces some spikes in irregular intervals. Their cause is unclear at present and will be inspected further in the future.

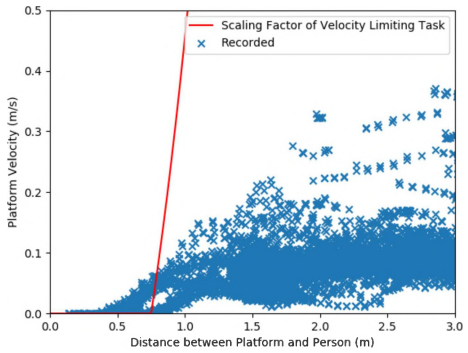


Fig. 10: Relation between platform velocity and distance to a person.

In real applications, the additional delay and inaccuracy caused by perception systems have to be considered. We plan to evaluate this in our future work.

VI. Conclusion and Future Work

We presented a method for safe WBC of redundant mobile manipulators in environments with static and dynamic obstacles. The intended robot behavior is described by a set of weighted and prioritized tasks. The control signal is found through QP with constraints defined by the active tasks. The configuration of tasks can be changed through ROS interfaces to adapt the robot behavior to the current situation. We describe the task types we have implemented and how to define new task types, namely by defining their Jacobian and bound controllers. In a simulation-based evaluation we have shown the capability of our method to react safely to human interference. The robot evades the human while utilizing its redundancy to perform its other tasks as well as possible and without coming into contact with static obstacles. If the human moves closer than a specified safety distance, the robot stops completely.

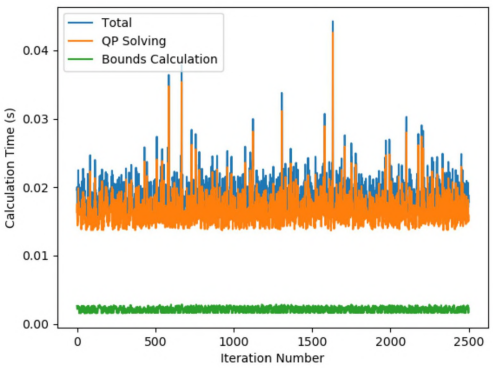


Fig. 11: Calculation times for finding the task bounds, solving the QP problem, and the total over 2500 cycles.

For the future, one obvious next step is the transferal of this approach to real hardware. This also enables further experimentation regarding physical interaction with the environment and with people, for example pick-and-place and object handover scenarios. We hope to develop the necessary methods to specify such interactions expressively and concisely. The different reactions required in different phases of an interaction and to different objects have to be considered. For example, physical contact with the object that is to be picked up must be possible, but only with specific parts of the robot and only during a picking attempt and not, for example, during an evasive motion. Once the object has been picked up, its geometry must also be considered for collision checking. Interaction with humans poses many similar challenges with a much higher level of complexity. We intend to work on the use of the approach described here for scenarios of this type. Besides that, we plan to make further use of the real-time capabilities of ROS2.

REFERENCES

[1] R. Kittmann, T. Fröhlich, J. Schäfer, U. Reiser, F. Weißhardt, and A. Haug, "Let me introduce myself: I am Care-O-bot 4, a gentleman robot," *Mensch und Computer 2015 – Proceedings*, 2015.

[2] J. Miseikis, P. Caroni, P. Duchamp, A. Gasser, R. Marko, N. Miseikiene, F. Zwilling, C. de Castelbajac, L. Eicher, M. Früh, and H. Früh, "Lio – a personal robot assistant for human-robot interaction and care applications," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5339–5346, 2020.

[3] J. Pages, L. Marchionni, and F. Ferro, "Tiago: the modular robot that adapts to different research needs," in *International workshop on robot modularity, IROS*, 2016.

[4] J. Bohren, R. B. Rusu, E. G. Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mösenlechner, W. Meeussen, and S. Holzer, "Towards autonomous robotic butlers: Lessons learned with the pr2," in *2011 IEEE Itnl. Conf. on Robotics and Automation*. IEEE, 2011, pp. 5568–5575.

[5] E. Ackerman, "Moxi prototype from diligent robotics starts helping out in hospitals," *IEEE Spectrum*, 2018.

[6] M. Drust, T. Dietz, A. Pott, and A. Verl, "Production assistants: The rob@work family," in *IEEE International Symposium on Robotics 2013*, 2013, pp. 1–6.

[7] C. Wurll, T. Fritz, Y. Hermann, and D. Hollnaicher, "Production logistics with mobile robots," in *ISR 2018; 50th International Symposium on Robotics*, 2018, pp. 1–6.

[8] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich, "Fetch and freight: Standard platforms for service robot applications," in *Workshop on autonomous mobile service robots*, 2016.

[9] "ROS 2," www.ros2.org, 2021, [Online; accessed 05-March-2021].

[10] M. Faroni, M. Beschi, and N. Pedrocchi, "An mpc framework for online motion planning in human-robot collaborative tasks," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2019, pp. 1555–1558.

[11] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 882–893, 2016.

[12] A. Jain and C. Kemp, "EL-E: An assistive mobile manipulator that autonomously fetches objects from flat surfaces," *Autonomous Robots*, vol. 28, pp. 45–64, 09 2010.

[13] J. Carius, M. Wermelinger, B. Rajasekaran, K. Holtmann, and M. Hutter, "Deployment of an autonomous mobile manipulator at mbzirc," *Journal of Field Robotics*, vol. 35, 10 2018.

[15] R. Ancona, "Redundancy modelling and resolution for robotic mobile manipulators: a general approach," *Advanced Robotics*, vol. 31, no. 13, pp. 706–715, 2017.

[14] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of human support robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, 04 2019.

[16] F. Chen, M. Selvaggio, and D. G. Caldwell, "Dexterous grasping by manipulability selection for mobile manipulator with visual guidance," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1202–1210, 2019.

[17] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decré, R. Smits, E. Aertbeliën, K. Claes, and H. Bruyninckx, "Constraint-based task specification and estimation for sensor-based robot systems in the presence of geometric uncertainty," *The International Journal of Robotics Research*, vol. 26, no. 5, pp. 433–455, 2007.

[18] W. Decré, R. Smits, H. Bruyninckx, and J. De Schutter, "Extending iTaSC to support inequality constraints and non-instantaneous task specification," in *Proceedings of the 2009 IEEE International Conference on Robotics and Automation*, Kobe, Japan, 2009, pp. 964–971.

[19] D. Mronga, T. Knobloch, J. de Gea Fernández, and F. Kirchner, "A constraint-based approach for human-robot collision avoidance," *Advanced Robotics*, pp. 1–17, 2020.

[20] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar, "A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks," in *International Conference on Advanced Robotics (ICAR)*, June 2009, p. 119.

[21] E. Aertbeliën and J. De Schutter, "eTaSL/eTC: A constraint-based task specification language and robot controller using expression graphs," IEEE, 2014, pp. 1540–1546.

[22] A. Escande, N. Mansard, and P.-B. Wieber, "Hierarchical quadratic programming: Fast online humanoid-robot motion generation," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 1006–1028, 2014.

[23] M. Faroni, M. Beschi, N. Pedrocchi, and A. Visioli, "Predictive inverse kinematics for redundant manipulators with task scaling and kinematic constraints," *IEEE Transactions on Robotics*, vol. 35, no. 1, pp. 278–285, 2019.

[24] F. Flacco, T. Kroger, A. Luca, and O. Khatib, "Depth space approach to human-robot collision avoidance," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 338–345, 05 2012.

[25] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[26] A. Escande, S. Miossec, M. Benallegue, and A. Kheddar, "A strictly convex hull for computing proximity distances with continuous gradients," *Robotics, IEEE Transactions on*, vol. 30, pp. 666–678, 06 2014.

[27] H. Ferreau, C. Kirches, A. Potschka, H. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[28] H. J. Ferreau, P. Ortner, P. Langthaler, L. del Re, and M. Diehl, "Predictive control of a real-world diesel engine using an extended online active set strategy," *Annual Reviews in Control*, vol. 31, no. 2, pp. 293–301, 2007.

[29] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 3859–3866.

[30] R. Smits, "KDL: Kinematics and Dynamics Library," http://www.orocos.org/kdl, 2021, [Online; accessed 05-March-2021].

[31] "Neobotix MPO-700," www.neobotix-robots.com, 2021, [Online; accessed 02-March-2021].

[32] "Schunk LWA 4P," www.schunk.com, 2021, [Online; accessed 02-March-2021].

[33] "Gazebo," www.gazebosim.org, 2021, [Online; accessed 05-March-2021].

[34] C. Willén, "Walking speed indoors and outdoors in healthy persons and in persons with late effects of polio," *Journal of Neurology Research*, 01 2013.

[35] T. Honda, M. Hirashima, and D. Nozaki, "Adaptation to visual feedback delay influences visuomotor learning," *PloS one*, vol. 7, p. e37900, 05 2012.