A Constraint-Based Approach for Human-Robot Collision Avoidance

Dennis Mronga[a], Tobias Knobloch[b], José de Gea Fernández[a], Frank Kirchner[ac]

[a]Robotics Innovation Center at German Research Center for Artificial Intelligence (DFKI), Bremen, Germany; [b]Hochschule Aschaffenburg, Aschaffenburg, Germany; [c]Robotics Group of the University of Bremen, Bremen, Germany

ARTICLE HISTORY
Compiled February 10, 2020

ABSTRACT
In this paper we present a software-based approach for collision avoidance that can be applied in human-robot collaboration scenarios. One of the contributions is a method for converting clustered 3D sensor data into computationally efficient convex hull representations used for robot-obstacle distance computation. Based on the computed distance vectors, we generate collision avoidance motions using a potential field approach and integrate them with other simultaneously running robot tasks in a constraint-based control framework. In order to improve control performance, we apply evolutionary techniques for parameter optimization within this framework based on selected quality criteria. Experiments are performed on a dual-arm robotic system equipped with several depth cameras. The approach is able to generate task-compliant avoidance motions in dynamic environments with high performance.

KEYWORDS
Collision Avoidance; Human-Robot Collaboration; Real-Time Robot Control; Parameter Optimization

1. Introduction

Human-robot collaboration is currently of booming interest not only in research but also in the industrial domain. The appearance of the first lightweight collaborative robots ([1–3]) allowed to some extent the removal of the usual fences behind which industrial robots were confined so far. The appearance of safety-rated sensors (e.g. [4]) enabled external monitoring of shared spaces and the assurance of safety by keeping a minimum distance between robot and human. These approaches usually stop the robot motion as soon as the sensors detect a violation of the minimum distance separation. Despite being perfectly suited solutions for some applications, other scenarios require a tighter human-robot cooperation (e.g. in shared assembly) and stopping the robot would not be an efficient and practical solution. Thus, more elaborated algorithms are required that enable robots to avoid unintended collisions by either finding escape trajectories whenever possible or by briefly pausing the motion if a collision cannot be prevented.

The focus of this paper is collision distance computation and avoidance control. Regarding the former,this work is based on the Kinematic Continuous Collision Detection

CONTACT A. N. Author. Email: dennis.mronga@dfki.de

Library (KCCD) for robotic self-collision avoidance [5]. We extend the library to also allow distance computations between robots and unknown external objects. Furthermore, we provide a method for converting 3D point cloud data into computationally efficient rigid body representations used by KCCD. Given the robot-obstacle distance vectors computed by KCCD, we generate suitable escape trajectories for the robot using repulsive potential fields. A particularly appealing way of integrating avoidance behaviors with the actual robot task into a coherent control signal is constraint-based robot control [6]. The principle here is that the robot is supposed to avoid collisions with external objects (e.g. a human entering its workspace), while at the same time executing its main task (e.g. following a trajectory) and other secondary tasks. While constraint-based robot control offers a flexible and modular way of describing complex robot tasks, the control solution is usually governed by a large number of parameters. Since manual tuning of those parameter is time-consuming and error-prone, we investigate automatized methods to identify them. In particular, we apply evolutionary techniques for parameter optimization based on a suitable set of performance indicators. Experiments showing the feasibility of our approach are performed on a dual-arm robot system.

In summary, this work presents the integration of the following original research contributions:

- An extension of an existing collision distance computation method (KCCD) by introducing (a) the possibility of taking into account external objects and (b) a method to efficiently convert raw sensor data in form of point clouds into the required format for the collision distance computation so that it can be used in real-time applications.
- An approach for generation of optimal control parameters within a constraint-based control framework based on an evolutionary algorithm, which provides improved control performance compared to manual tuning

This paper is organized as follows. In Section 2.3, an overview of related research on the topic is given. Section 3 illustrates the main contributions including the algorithm for converting 3D point cloud data into convex hulls for collision distance computation (section 3.2), the framework for constraint-based collision avoidance control (3.3) and the parameter optimization methods based on evolutionary computation (3.4). The experimental setup and results are described and discussed in Section 4, while Section 5 contains a short conclusion and outlook.

2. Related Work

2.1. Robot-Obstacle Distance Computation

Due to its fundamental importance in robotics, collision detection and avoidance has been a long studied topic. One of the most interesting recent works is [7]. In the presented approach, distances between obstacles and control points on the robot are computed in depth image space, which is much faster than other distance evaluation methods. Compared to this, we evaluate robot-obstacle distances in 3D space, which is usually slower. However, it allows integration of multiple, heterogeneous sensors that provide point cloud data. Furthermore, our approach is able to integrate multiple additional constraints apart from collision avoidance and position control. Another approach that uses convex hull presentations has been recently presented in [8]. How-

ever, the approach rather focuses on motion planning than on reactive, task oriented control as our work does.

The collision distance computation method in our work is based on the KCCD library [5,9]. KCCD uses a compact and computationally efficient convex hull representation of rigid bodies (see section 3.2 for details). In this work we adapt KCCD in order to allow the inclusion of arbitrary external objects observed by sensor data, which is not possible with the original approach.

2.2. Multi-Objective and Collision Avoidance Control

Regarding the creation of avoidance motions, a lot of methods are based on the concept of artificial potential fields [10]. The work in [11], for example, combines a method for real-time path modification and task-consistent dynamic control based on repulsive potential fields. While the approach may generate task consistent avoidance motions, it is based on 2D laser scanner information. Our approach on the other hand uses 3D point cloud data, which allows more complex collision avoidance scenarios, e.g. between a human and robot.

Managing multiple tasks in redundant robots has been widely studied and first approaches came up in the early nineties [12]. When first humanoid robots were physically available, the topic recurred under the name *whole-body control* and lots of approaches have emerged within the last decade. The work in [13] generates torque-level motions for complex robotic systems based on the concept of operational space control [14]. Another important approach, which forms the basis of our work for task consistent collision avoidance, is the iTaSC-framework presented in [6]. This framework is implemented on velocity level and can be used for sensor-based, reactive control of complex robot motions. We adopted this approach in our work, because it is extremely flexible and easier to implement than torque-based approaches as in [13], which require quite precise knowledge of the robot dynamics. Apart from the aforementioned methods, many approaches based on quadratic programming (QP) came up recently. In these methods, task objectives are formulated as quadratic programs and a common solution is computed using numerical QP-solvers. In [15] such a multi-objective controller is presented. The tasks are implemented in a strict hierarchy of quadratic programs allowing equality and inequality constraints. The approach in [16] on the other hand allows the implementation of both, strict and soft task priorities. Many more works on this topic exists, a review can be found in [17]. Also, integrating collision avoidance motions by using constraint-based robot control has been dealt with before, e.g. by applying the stack of tasks method on a humanoid robot [18], using a sequence of quadratic programs [19] or by adopting a torque-based approach for self-collision avoidance [20].

The novelty of our work with respect to these approaches is that we extend the concept of constraint-based robot control with machine learning techniques in order to find optimal controller parameters, which are often difficult to obtain by hand.

2.3. Automatic Parametrization of Multi-Objective Controllers

Automatic parametrization of multi-objective controllers by using machine learning has been considered before. In [21] a learning method for humanoid whole-body control is presented. In particular, the approach tries to overcome task discrepancies using reinforcement learning on the simulated iCub robot. In [22] on the other hand a method

for learning soft task priority functions using evolutionary techniques is presented. The approach is evaluated using a collision avoidance task on a single-arm industrial manipulator. In [23] the authors present a method to learn strict task priorities based on user demonstrations for bi-manual robot tasks like dual-arm reaching. Finally, in [24] a framework for automatic derivation of mixture coefficients of a multi-objective controller is described. The approach is applied for tasks like single-arm reaching in a simulation of a humanoid robot.

All approaches mentioned above are either applied in a very restricted context, only in simulation or for very simple robot tasks. Compared to that we are striving for methods that can be applied to more general robot control problems, are able to deal with large number of parameters and contexts and can be applied on real robots.

3. Real-Time Collision Avoidance based on Constraints

In this section we illustrate our approach for real-time human-robot collision avoidance using constraints. Ideally, in a dynamic environment, a robotic system is supposed to avoid obstacles without pausing its main task, but continue executing it if possible. By using a constraint-based control approach a smooth integration of avoidance behaviors and task goals can be achieved, as will be shown in the following sections.

3.1. 3D Sensor Processing

Our approach for obstacle detection is based on raw point cloud data. Since we use RGB-D cameras as input, additional pre-processing is required to convert the raw depth images into point clouds. In summary, the following sensor processing steps are performed in advance to collision distance computation: (1) Background subtraction, (2) Robot self-filtering, (3) Depth image to point cloud conversion and (4) Spatial clustering. While we only show an overview in Fig. 1, a detailed description can be found in [25]. The result of these processing steps are a number of 3D point clusters, which describe the external objects in the vicinity of the robot. Each cluster typically comprises a couple of hundred up to a couple of thousand 3D points, depending on the size of the object. In the following section we describe a method for converting these 3D point clusters into a convex hull representation understood by KCCD.

3.2. Real-Time Robot-Obstacle Distance Computation

For robot-obstacle distance computation, we use an extension of the KCCD library [5]. KCCD is able to evaluate distances between rigid bodies in real-time. Each body is thereby represented by a convex hull enclosing a finite set of supporting points, extended by a buffer radius r in all directions, see Fig. 2 for examples. For the sake of simplicity the examples are in 2D, even though KCCD can generally deal with 3D volumes. Using this volume representation, arbitrary shapes can be approximated as long as the number of support points is large enough. For example, a sphere can be approximated with one support point and a buffer radius. More complex objects require more points, which increases the computational complexity.

KCCD requires a formal model of all robots and collision bodies in the scene. For this reason, it is unable to deal with unknown external objects that enter the workspace

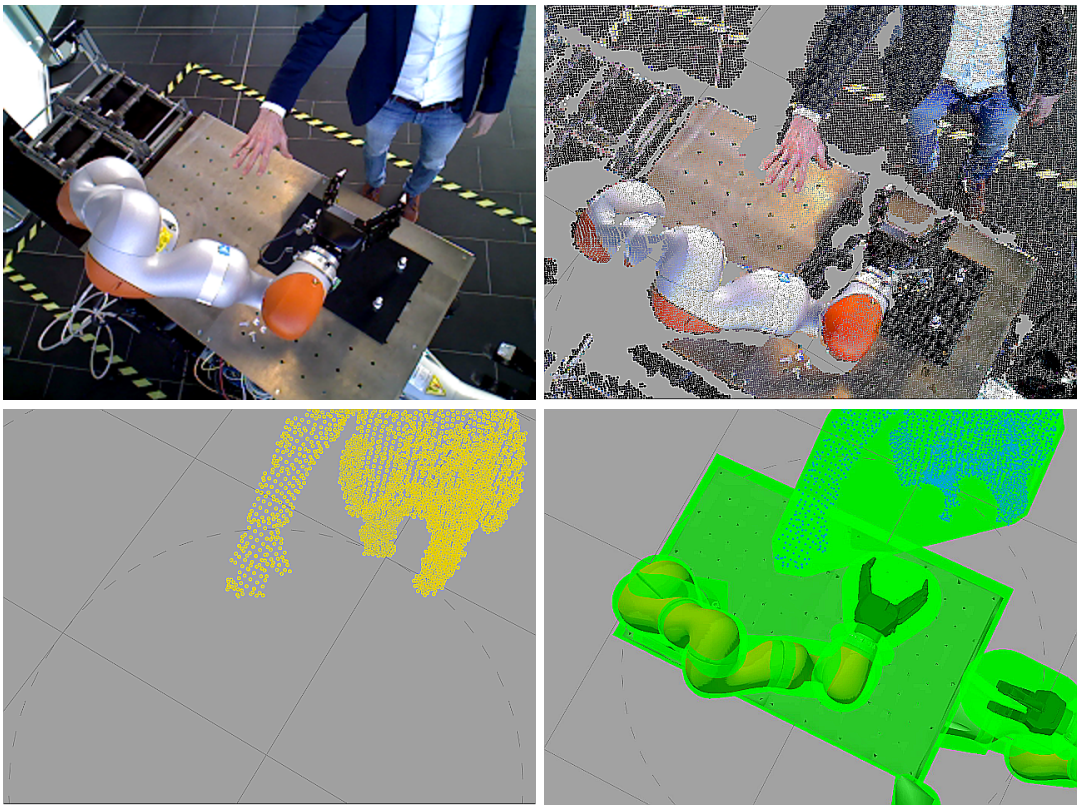
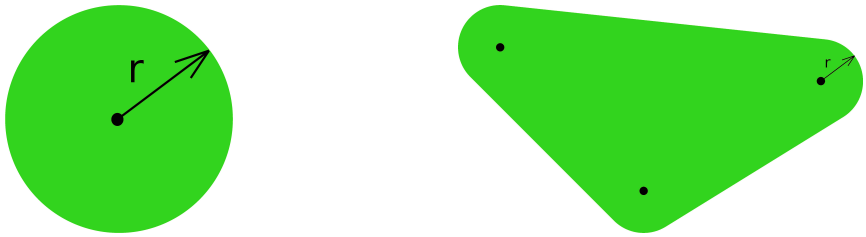


Figure 1.: Overview of the most important sensor processing steps: *Top Left*: Original scene, *Top Right*: 3D point cloud, *Bottom Left*: Point cluster representing the external object, *Bottom Right*: Convex hulls for collision distance computation.

of the robot. Here, we extend the KCCD software to allow inserting arbitrary KCCD volumes at runtime and provide a method to efficiently convert 3D sensor data into these data structures. As a starting point, we assume that sensor data is given as a set of 3D points, which have been grouped using spatial clustering as described in section 3.1. The goal is to find an optimal KCCD volume (supporting points and buffer radius) for each external object. Optimal means here, that the volume covers all initial 3D points from the given cluster, while comprising the minimal number of KCCD supporting points.

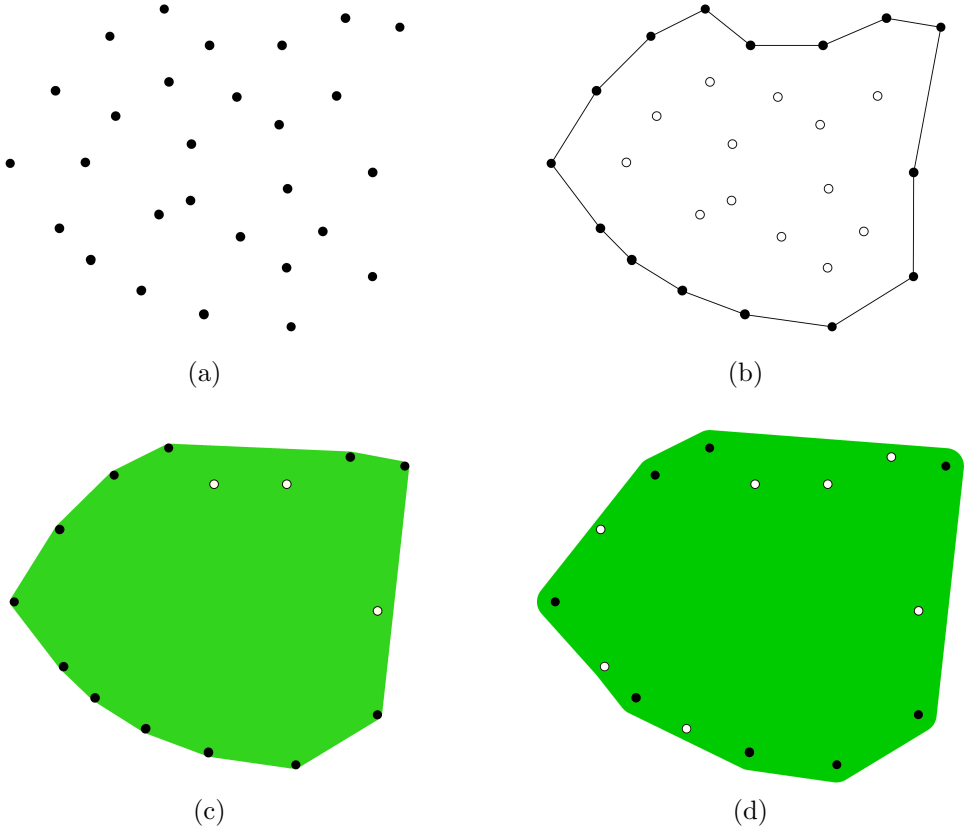
At first we compute a convex hull to remove the inner points using the Qhull algorithm [26]. Figure 3b shows the convex hull of a sample object. To reduce the number of the remaining points, we developed Algorithm 1. First, the buffer radius r is set to an initial value r_{init}, selected by the user. In our work we used $r_{init} = 1e-3$. Now we iterate through each point of the volume and create a simpler volume without this point and the buffer radius. Afterwards we calculate the distance between this point and the simpler volume, using the KCCD library. If the distance is zero or smaller, the point is still covered by the simpler Volume. If, after one iteration loop, too many points are left ($N > N_{max}$), we increase the buffer radius and repeat the process. Figure 3c shows the intermediate result after one iteration and Fig. 3d shows the final result.

We track the collision objects over multiple sensor frames and use the collision volume from the previous frame as starting point in order to deal with sensor noise.



(a) Sphere with one point and buffer radius (b) Volume with three points and buffer radius

Figure 2.: 2D examples for KCCD volumes



(a) (b)

(c) (d)

Figure 3.: Illustration of the algorithm for point cloud to convex hull conversion: (a) Clustered point cloud from a depth sensor (b) Convex hull (c) Intermediate result after one iteration of Algorithm 1 (d) Final result.

Fur this purpose, we check which points are not covered by the collision volume of the previous sensor sample and extend the volume respectively. Furthermore, we check which points of the previous volume are too far away from the current volume and remove them. The resulting collision volume is then used as a basis for Algorithm 1. However, the algorithm considers only the currently visible 3D sensor data and does not "interpolate" missing visual information since this would require knowledge about the original object shape. If parts of an object, which were invisible before, become suddenly visible in the camera image, the algorithm will re-optimize the KCCD volume to fit the new object's shape.

In general KCCD can represent any shape accurately given that the number of support points is large enough. Nevertheless, there is a trade-off between accuracy and computational effort. Since the conversion algorithm stops when (a) all points of the point cloud are covered by the volume and (b) the number of support points is less than the allowed maximum number of points, it sometimes tends to overestimate the volume of objects, while saving computation time compared to a high-accuracy approximation (see for example Fig. 9). Other methods like for example [27] can compute distances between triangle meshes, which are a more accurate representation of the original object shape. However, this accuracy comes with the expense of a much higher computation time.

A possible improvement of the conversion algorithm could be to add another optimization step which adapts the shape of the hull if the volume is strongly non-convex. However, we found it in our case computationally too demanding when considering the real-time requirements of the system. Also we consider the over-estimation of the object volumes as non-critical in the case of human-robot collision avoidance since the robot should maintain a considerable safety distance to any human in the workspace. Especially with regard to the acceptance of collaborative robots in industrial environments a rather large safety margin will be beneficial.

The robot obstacle distance vectors evaluated by KCCD can be used to compute suitable avoidance motions, which is described in the following section.

3.3. Task-Compliant Collision Avoidance Control

Robot control based on constrained optimization is a method that allows to control complex robotic tasks based on constraints. The idea is to break down the overall control problem into (prioritized) tasks and describe them as constraints to an online optimization problem, whose solution represents the control signal for the robot. With this approach avoidance behaviors for different parts of the robot, as well as multiple other robot tasks can be controlled simultaneously and merged into a coherent control signal.

In our constraint-based control framework we use a similar approach as in [6]. We assign a motion generator, a controller and a constraint to each task. A motion generator defines the desired behavior of a task by providing for example a trajectory, distance information or a set of waypoints. The controllers attempt to regulate this behavior in task space, while the constraints describe the tasks as part of an optimization problem. The solution of the optimization problem is a velocity-based robot

Algorithm 1 Point cloud to KCCD volume conversion

```
r ← r_init
while N > N_max do
  for all points ∈ volume do
    simplerVolume ← volume − point
    d ← computeDistance(point, simplerVolume)
    if d ≤ 0 then
      volume ← simplerVolume
    end if
  end for
  r ← r + r_step
end while
```

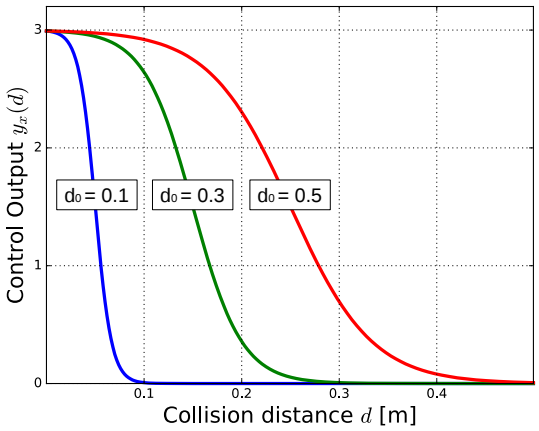


Figure 4.: Control output $y_x(d)$ of the collision avoidance controller (only x-axis) with respect to the collision distance d for $k_{p,x} = 3$ and different values of the maximum influence distance d_0

control signal, which is updated in every control cycle. Figure 5 shows an example configuration of our framework, which can be easily extended by additional constraints like e.g. joint position limits.

For controlling the pose of the robot in Cartesian space, we use a proportional controller with the control law

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_t \\ \mathbf{y}_\phi \end{pmatrix} = \mathbf{K}_p \begin{pmatrix} \mathbf{x}_r - \mathbf{x}_a \\ \mathbf{R}_a \cdot \theta \hat{\boldsymbol{\omega}}_r^a \end{pmatrix}, \qquad \mathbf{y} \leq \bar{\mathbf{y}} \tag{1}$$

where $\mathbf{y} \in \mathbb{R}^{6\times 1}$ is a Cartesian twist (linear and angular velocity $\mathbf{y}_t$ and $\mathbf{y}_\phi$) representing the Cartesian control output, $\mathbf{K}_p \in \mathbb{R}^{6\times 6}$ is a diagonal matrix containing the 6 feedback gain constants and $\bar{\mathbf{y}} \in \mathbb{R}^{6\times 1}$ is a vector containing the maximum control output for each dimension of the controller. The vectors $\mathbf{x}_r \in \mathbb{R}^{3\times 1}$ and $\mathbf{x}_a \in \mathbb{R}^{3\times 1}$ denote the reference and actual position of the controlled robot frame, the matrix $\mathbf{R}_a \in \mathbb{R}^{3\times 3}$ its actual orientation. The term $\theta \hat{\boldsymbol{\omega}}_r^a \in \mathbb{R}^{3\times 1}$ is the angle-axis representation of the rotation between actual and reference orientation of the controlled robot frame[1]. This vector is transformed to the robot base frame by multiplying with the actual orientation of the controlled robot frame $\mathbf{R}_a$. Thus, the angular part of $\mathbf{y}$ rotates the controlled robot frame from its current orientation to the given reference orientation around an axis defined by $\hat{\boldsymbol{\omega}}_r^a$.

For avoidance motions we use repulsive potential fields with the control law

$$\mathbf{y} = \begin{pmatrix} \mathbf{y}_t \\ \mathbf{0}^{3\times 1} \end{pmatrix} \quad \mathbf{y} \leq \bar{\mathbf{y}} \tag{2}$$

with

$$\mathbf{y}_t = \begin{cases} \mathbf{K}_p \frac{\mathbf{x}_a - \mathbf{x}_0}{d} S(d), & d < d_0 \\ \mathbf{0}, & \text{else} \end{cases} \tag{3}$$

[1]The angle-axis notation represents a rotation between two coordinate frames by a single angle $\theta \in \mathbb{R}$ and a unit vector $\hat{\boldsymbol{\omega}} \in \mathbb{R}^{3\times 1}$ [28].

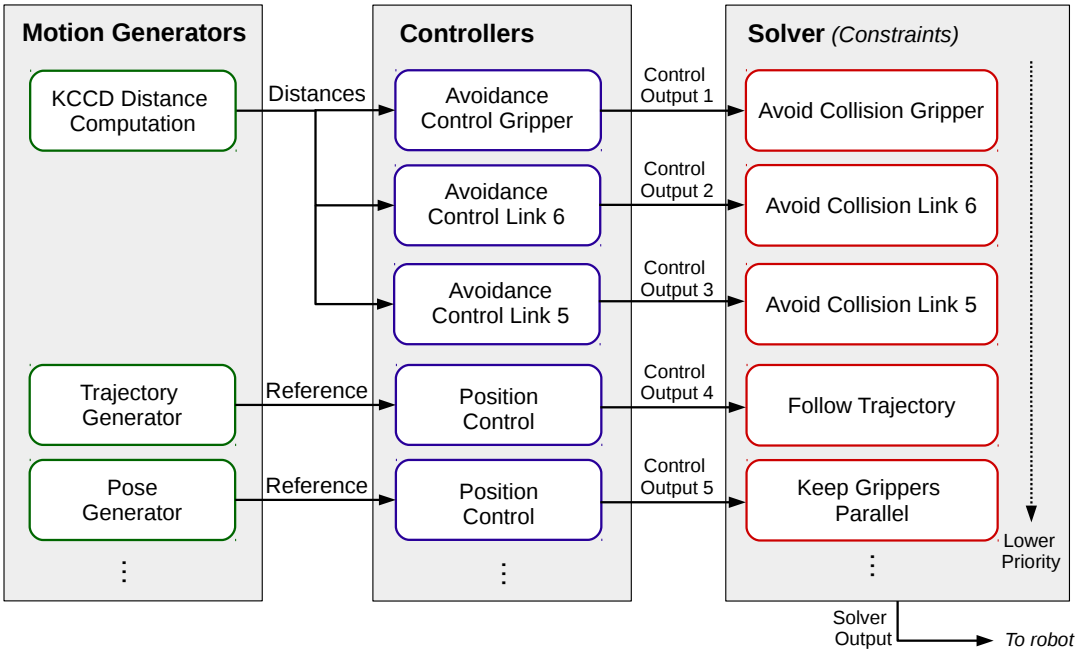


Figure 5.: Overview of the constrained-based approach for collision avoidance.

where $\mathbf{x}_a \in \mathbb{R}^{3\times1}$ and $\mathbf{x}_0 \in \mathbb{R}^{3\times1}$ are the actual robot position and the position of the potential field center, $d_0 \in \mathbb{R}$ the maximum influence distance of the potential field and $\mathbf{K}_p \in \mathbb{R}^{3\times3}$ is a diagonal matrix containing the 3 feedback gain constants. The term $S(d)$ is a sigmoid function of the distance $d = \|\mathbf{x}_a - \mathbf{x}_0\|$ between the robot and the potential field center:

$$S(d) = \left(1 + e^{\alpha(1-2\frac{d_0-d}{d_0})}\right)^{-1} \tag{4}$$

with $\alpha = 6$ having been chosen empirically. If the collision distance d becomes small ($d \approx 0$), the sigmoid term will be $S(d) \approx 1$, since the exponential term in the sigmoid function becomes small (e^{-6}). Thus, for a small collision distance the entries in $\mathbf{y}$ approach the proportional gains in $\mathbf{K}_p$ multiplied by a direction vector: $\mathbf{y} \approx \mathbf{K}_p \frac{\mathbf{x}_a - \mathbf{x}_0}{d}$, where $\|\frac{\mathbf{x}_a - \mathbf{x}_0}{d}\| = 1$. On the other hand, if the collision distance d is close to the maximum influence distance d_0 of the potential field ($d \approx d_0$), the exponential term becomes large (e^6) and $S(d) \approx 0$. Thus, for $d \approx d_0$, the control output $\mathbf{y} \approx \mathbf{0}$. Figure 4 illustrates the development of the control output with respect to the computed collision distance. We chose a Sigmoid function to generate the avoidance motions, because it implements a smooth transition if the robot enters the influence sphere of a potential field (in contrast to e.g. piecewise linear functions). Also, the value range of Sigmoids is finite, which ensures a finite repulsive velocity and thus a more stable robot behavior compared to having a reciprocal (or even squared reciprocal) dependency of the repulsive velocity with respect to the robot-obstacle distance. Finally, the shape of a Sigmoid function can be nicely adapted by adjusting the function parameters, in this case d_0 and α.

The output of each controller $\mathbf{y}_1 \dots \mathbf{y}_N$ is represented in an equality constraint of

the optimization problem

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \|\mathbf{u}\| \\ \text{subject to} \quad & \begin{pmatrix} \mathbf{A}_{1,w} \\ \vdots \\ \mathbf{A}_{N,w} \end{pmatrix} \mathbf{u} = \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} \end{aligned} \tag{5}$$

where $\mathbf{u} \in \mathbb{R}^{n \times 1}$ is the robot control signal[2], n the number of robot joints, N is the number of tasks and $\mathbf{A}_{i,w} = \mathbf{W}\mathbf{A}_i \in \mathbb{R}^{6 \times n}$ is the weighted task Jacobian related to the i-th task. The task Jacobian describes the relation between joint velocities and task space velocities for the controlled robot frame (e.g. the robot's end effector). The term $\mathbf{W} \in \mathbb{R}^{6 \times 6}$ is a diagonal matrix containing the *task weights* $\mathbf{w} = (w_1 \dots w_6)$. The weights can be used to balance the importance of the constraint variables. For example, when controlling the pose of the robot in Cartesian space and the orientation is not relevant, the corresponding task weights can be set to zero.

There are different ways to compute the solution of Eq. 5. In general, if the number of linear independent constraint equations is less than the robot's degrees of freedom, the robot is said to be redundant with respect to the given task(s) and the redundancy can be utilized to fulfill additional tasks or optimization criteria [28]. The work in [13] for example resolves the robot redundancy by implementing a strict control hierarchy using iterative Nullspace projections of physical, task and postural constraints. However, if the number of constraints is higher than the degrees of freedom of the robot, such a method potentially fails, because some constraints cannot be considered in the solution. A widely used method to overcome this problem is to use a weighting scheme (task weights) to balance the importance of constraints. We prefer task weights here, since they are more suitable for numerical optimization (see section 4.4) than strict task priorities. Regarding the solution of Eq. 5 we first stack the weighted task Jacobians $\mathbf{A}_{1,w}, \dots, \mathbf{A}_{N,w}$ into a single matrix $\mathbf{A} \in \mathbb{R}^{6N \times n}$ and the control output vectors $\mathbf{y}_1, \dots \mathbf{y}_N$ into a single vector $\mathbf{y} \in \mathbb{R}^{6N \times 1}$ as follows:

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{1,w} \\ \vdots \\ \mathbf{A}_{N,w} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{pmatrix} \tag{6}$$

The solution of Eq. 5 can now be computed as $\mathbf{u} = \mathbf{A}^+\mathbf{y}$, where $\mathbf{A}^+$ denotes the Pseudo Inverse of the matrix $\mathbf{A}$. In the under-constrained case ($n > 6N$) the Pseudo Inverse provides the solution that minimizes $\|\mathbf{u}\|$, while in the over-constrained case ($n < 6N$) it provides an approximate solution that minimizes the error $\|\mathbf{A}\mathbf{u} - \mathbf{y}\|$. A general method for computing the Pseudo inverse can be obtained by using singular value decomposition in the following way. Let $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ be the singular value decomposition of the real-valued matrix $\mathbf{A}$, where $\mathbf{U}$ and $\mathbf{V}$ are the matrices with left- and right-singular vectors, respectively. The matrix $\mathbf{\Sigma}$ is the singular value matrix, whose main diagonal entries σ_i are the singular values of $\mathbf{A}$. The Pseudo inverse of $\mathbf{A}$ can now be computed as $\mathbf{A}^+ = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^T$. The inverse of the singular value matrix $\mathbf{\Sigma}$ can be computed by taking its transpose and inverting the singular values. However, if $\mathbf{A}$ is not of full rank, some singular values may approach zero and the resulting joint

[2]In our implementation, the control signal is a joint velocity

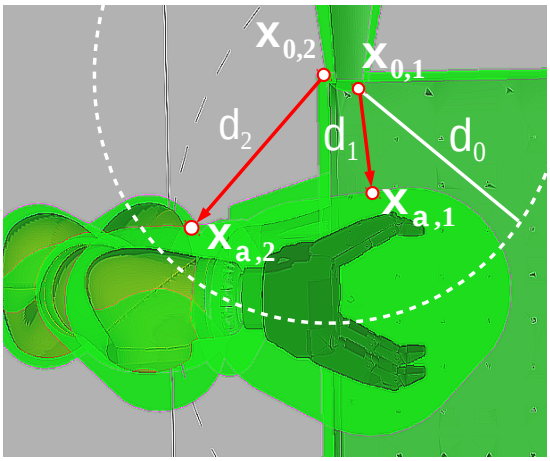


Figure 6.: Computation of avoidance vectors according to Eq. 3. The potential field center $\mathbf{x}_0$ and the corresponding control point $\mathbf{x}$ on the robot are assigned to the two closest points on the collision volumes of obstacle and robot link, respectively. The control vector $\mathbf{y}$ of the potential field controller points from the potential field center to control point on the robot, pushing the corresponding link away from the obstacle. The avoidance controllers are integrated as equality constraints of the form $\mathbf{Au} = \mathbf{y}$, where $\mathbf{u}$ is the robot control signal in joint space and $\mathbf{A}$ is the Jacobian from the robot base to the robot link containing the control point $\mathbf{x}$.

velocities $\mathbf{u}$ approach infinity. To avoid such numerical instabilities we compute the entries of the inverted singular value matrix as follows:

$$(\mathbf{\Sigma})^{+}_{ij} = \begin{cases} \frac{\sigma_i}{\sigma_i^2+\delta^2} & \text{if} \quad i = j \\ 0 & \text{else} \end{cases} \tag{7}$$

The damping factor δ is determined automatically. It is set to zero when the smallest singular value of $\mathbf{A}$ is higher than an empirically determined threshold. If one or more singular values approach zero, the damping factor is set to a value $\lambda > 0$ as described in [29].

The task weights introduce "soft" task priorities, which means that the tasks are not hierarchically organized as in [13], but the solution is computed as a weighted combination of the tasks governed by the value of the corresponding task weights. In the over-constrained case, this means that constraints with lower weight will be represented less in the solution. As a simple example one might consider the case where a single 6-DoF arm is commanded to two different poses $\mathbf{X}_a$ and $\mathbf{X}_b$ at the same time. The problem is obviously over-constrained, an accurate solution does not exist. Depending on the choice of the task weights the above method would provide an approximate solution, where none of the two poses is reached, but a "weighted average" solution in between is assumed.

Now, we integrate robotic collision avoidance and task control into a continuous control signal using artificial potential fields. We assign a constraint and an avoidance controller to each relevant link of the robot. The potential field center $\mathbf{x}_0$ and the corresponding control point $\mathbf{x}$ on the robot are assigned to the two closest points on the collision volumes of obstacle and robot link, respectively. This choice is illustrated in

Fig. 6. Using Eq. 3 we convert the safety distances, which actually represent inequality constraints, into equality constraints of the form $\mathbf{Au} = \mathbf{y}$. The task Jacobians $\mathbf{A}$ are computed from the robot base to the robot link, which is located within the influence distance of an obstacle. The weights for the avoidance tasks are computed as follows

$$w_i = \begin{cases} (d_0 - d)/d_0, & d < d_0 \\ 0, & \text{else} \end{cases} \quad \forall i \tag{8}$$

The weights are zero if the robot is located far from any obstacle, so that the avoidance behaviors do not interfere with the main task. When a robot link comes into the influence distance of an obstacle, the weights gradually increase, implementing a smooth transition between unconstrained and constrained robot motion. This way we can smoothly integrate avoidance motions with other simultaneously running tasks.

However, the solution is governed by a rather large set of parameters, control gains $\mathbf{k}_p$, saturation terms $\bar{\mathbf{y}}$, maximum influence distance d_0 of the potential field controllers, as well as the task's weights $\mathbf{w}$. Manual selection and fine-tuning of these parameters can be time-consuming, error prone and may lead to a sub-optimal solutions in the end. To overcome these issues we developed a parameter optimization approach, which is described in the following section.

3.4. Optimization of Control Parameters

We propose to apply evolutionary techniques to generate an optimal parameter set for our constraint-based control framework. The advantage of using evolutionary techniques for parameter optimization is that they nicely explore fitness functions with local maxima, which is commonly the case when dealing with conflicting subtasks. The disadvantage is that they usually require many evaluations of the fitness function. Thus, we rely on simulation for generating training data.

Since constraint-based control tasks usually have multiple, possibly conflicting objectives, defining a global fitness function for optimization is not a straightforward task[3]. In our case, avoidance behaviors might prevent reaching the reference position of the manipulator. Typically, trading off the performance of the individual constraints according to the overall task goal is required. In order to find a suitable fitness measure, we evaluate combinations of simple criteria:

$$\begin{array}{lrcl} \textit{Steady State Error} & f_{ss} & = & S\left\{\|\mathbf{e}(t_{ss})\|\right\} \\ \textit{Percentage Overshoot} & f_{po} & = & S\left\{\|\mathbf{X}_{max} - \mathbf{X}_{min})\|\right\} \\ \textit{Settling Time} & f_{st} & = & S\left\{t_{ss} - t_0\right\} \\ \textit{RMS Control Error} & f_{rms} & = & S\left\{\sum_{t=0}^{t_{ss}} \|\mathbf{e}(t)\|\right\} \\ \textit{Min. Col. Distance} & f_{cd} & = & S\left\{d_{min}\right\} \end{array} \tag{9}$$

where $\mathbf{e} = \|\mathbf{X}_r - \mathbf{X}\|$ is the position control error, t_{ss} and t_0 the settling and starting time of the control action, and d_{min} the minimum distance to a collision object during the control action. The term S is a Sigmoid function, which is used to confine the possible values of the performance criteria to the interval [0..1].

[3]Note that we refer to offline optimization of the control parameters here, not to be mistaken for the online optimization problem for generating a robot control signal as in Eq. 5

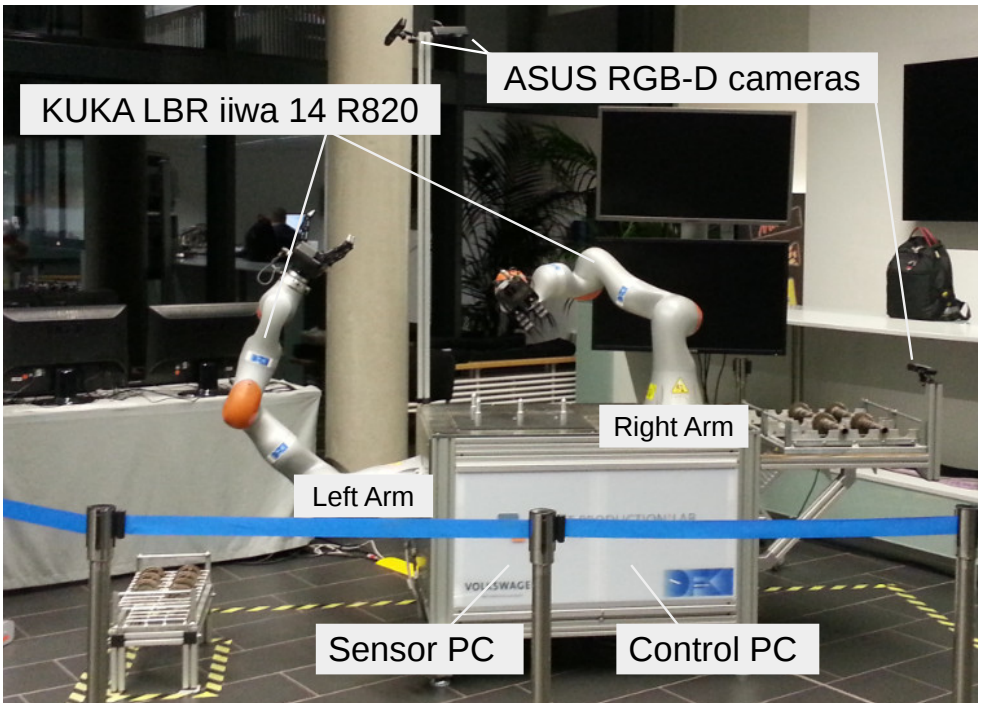


Figure 7.: Dual arm industrial robot system used as experimental platform

We assign a fitness function to each subtask. The global fitness function is defined as the weighted sum of the individual fitness functions:

$$f = \xi \sum_{i=1}^{N} \alpha^i f^i \tag{10}$$

where α_i is the weighting factor for the i-th fitness measure and $\xi = exp\left(-\frac{1-min(f^i)}{min(f^i)}\right)$ is a regularization term that punishes results with low values of individual fitness functions. By using this term, solutions that represent a trade-off between different subtasks will be favored compared to solutions where one subtask is fully achieved while another subtask is completely ignored. As a result, the performance of the individual objectives may degrade, which is however a desired effect since we are looking for a trade-off between the different objectives.

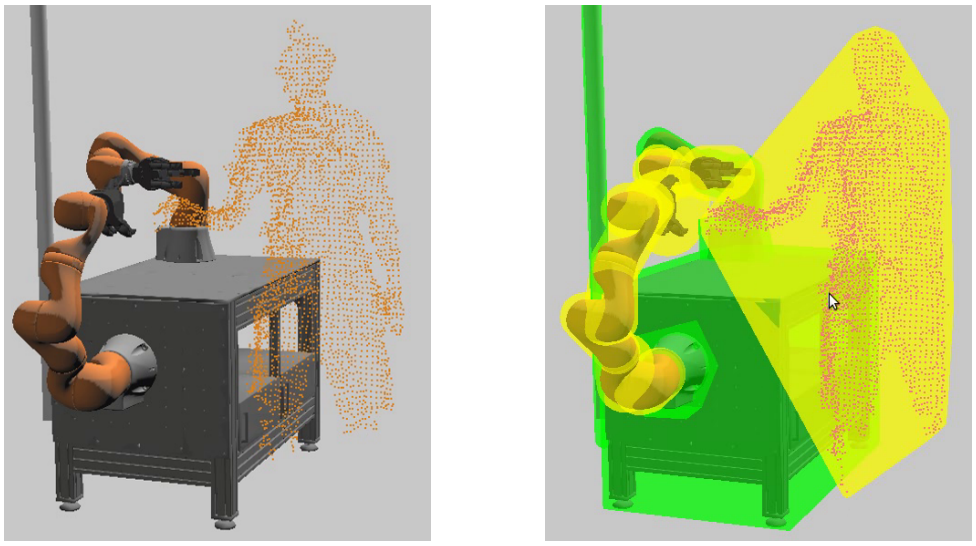
4. Results

4.1. System Description

For evaluation of robotic skills required in human-robot collaboration scenarios, we developed the system shown in Fig. 7, which is equipped with two KUKA LBR iiwa 14 R820 lightweight robots [1]. We attached 3 ASUS Xtion Pro Live RGB-D cameras [30], which are used for detection of external objects in the proximity of the robot. The control of the arms is performed on an industrial PC with Intel Core i7 4790K 4 x 4.00 GHz, while sensor processing runs on a standard desktop PC with Intel Core i7-2600 CPU 4 x 3.40 GHz. The overall system including the complete software framework is

(a) Coffee Mug (b) Box (c) Hand

Figure 8.: Enclosing KCCD volumes for different objects.



(a) Clustered point cloud (b) KCCD collision volume

Figure 9.: Human in the workspace scenario used for performance evaluation

described in [25] in more detail.

4.2. Real-Time Robot-Obstacle Distance Computation

4.2.1. Point Cloud to KCCD Conversion

Figure 8 shows point clouds of different objects and the enclosing KCCD collision volumes computed by Algorithm 1. The algorithm is tuned for efficiency and not supposed to perfectly match the exact shape of the underlying object, as particularly well shown in Fig. 8c. However, for collision avoidance it is usually acceptable to provide only a rough approximation of the true external object shape.

4.2.2. Computational Performance

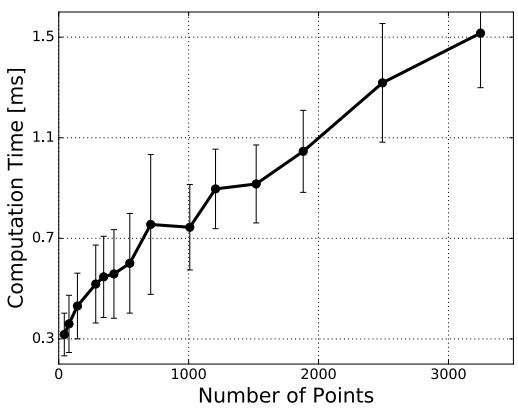
We further evaluate the performance of our method for computing robot-obstacle distance vectors in a "human-in-the-workspace" scenario as shown in Fig. 9. In order

Processing Step	Single Camera (640 x 480 px)	3 Cameras (320 x 240 px)
Background Subtraction	0.8870 ± 0.2032	0.3062 ± 0.0885
Self Filtering	7.5755 ± 0.9532	2.6589 ± 0.8263
Point Cloud Conversion	2.4813 ± 0.2823	1.4424 ± 0.6162
Clustering	9.6727 ± 1.5174	25.2576 ± 1.8769
Total (Preprocessing)	20.6165 ± 2.9561	29.6651 ± 3.4079
PC to KCCD Conversion	0.8846 ± 0.1773	1.7431 ± 0.3647
Distance Computation	0.2868 ± 0.3310	0.2742 ± 0.3315
Total (Distance Computation)	1.1714 ± 0.5083	2.0173 ± 0.6962

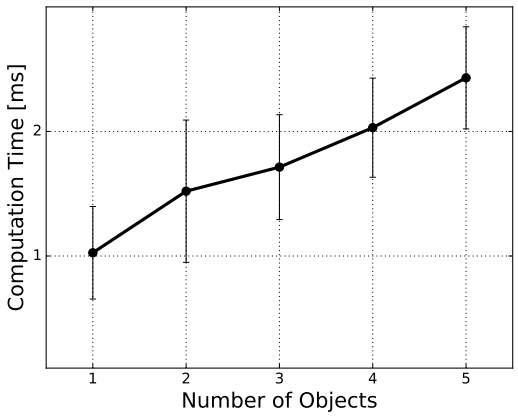
Table 1.: Mean computation time in ms $\pm$ single standard deviation for the respective processing steps.

to compare with the work in [7] we select a single-camera setup with the same camera resolution. Table 1 shows the mean computation time for the respective processing steps. It can be seen that considering the computation of collision distance vectors, we nearly achieve $1ms$ cycle time, which is comparable to the performance of the depth space approach in [7]. On the negative side our method usually requires some preprocessing in order to cluster the raw point cloud. This may take considerable amount of time, although it has to be mentioned that the pre-processing steps used here are not optimized for performance and can be considered optional to some extent. For example self filtering is only required if the robot is actually visible in the camera image. Furthermore, the performance loss is outweighed by the fact that our method allows to easily integrate multiple, heterogeneous sensors that provide point cloud data, whereas the approach in [7] operates on a single camera. The second column of Table 1 shows the performance of the setup as illustrated in section 4.1. We use 3 RGB-D cameras in order to cover the complete workspace of the robot, while reducing each camera's resolution to limit processing time. Even with the three-camera setup, the approach can be applied for real-time control.

Finally, we evaluate the relationship between the computation time for the robot-obstacle distance calculations and the object size (indicated by the number of points in the cluster), as well as the number of objects in the workspace. In our three-camera setup external objects typically consist of a couple of hundred up to a couple of thousand 3D points, while usually no more than 4 or 5 objects are located within the visible workspace. Figure 10 shows that the computation time is approximately linear with respect to the number of object points and also linear with respect to the number of visible objects. It also shows that the algorithm can perform distance computations



(a) Computation time vs. number of points in point cloud cluster (single object)



(b) Computation time vs. number of objects (equally-sized, box-shaped objects)

Figure 10.: Illustration of computation time (in ms) of point cloud to KCCD conversion + collision distance computation. The error bars illustrate the double standard deviation

with respect to multiple external objects with high performance.

4.3. Task-Compliant Collision Avoidance Control

In this section, we evaluate the principal functionality of the constraint-based framework to generate task-compliant collision avoidance behaviors. We select a scenario, where a single arm tries to follow a circular trajectory, while an obstacle is randomly placed into the path of the end effector. For the trajectory following subtask, we use the controller described by Eq. 1 and select the parameters for all degrees of freedom as follows: Control gain $k_p = 1.5$, maximum control output $\bar{y} = 0.5$, subtask weights $w = 1$. For collision avoidance we use the potential field controller from Eq. 3. We select a control gain of $k_p = 0.05$, a maximum control output of $\bar{y} = 0.5$ and a maximum influence distance $d_0 = 0.5m$. The weights of the collision avoidance subtask are computed automatically according to Eq. 8. Results are illustrated in Fig. 11. It can be seen that the end effector trajectory is followed until the robot approaches the obstacle. In this case the weight of the collision avoidance subtask (red dotted line) is increased, which makes the robot deviate from the target trajectory. However, the transient behavior when entering the influence distance of the obstacle is not smooth, because too aggressive control parameters have been chosen. This leads to repeated activation and deactivation of the collision avoidance subtask in the transient zone, a process that we refer to as "recurrence behavior". As discussed in section 3.4, parameter selection for constraint-based control systems is commonly a manual process, which can be time-consuming and, as in this case, may lead to suboptimal results. We therefore propose to apply automatized methods based on evolutionary optimization to select proper control parameters.

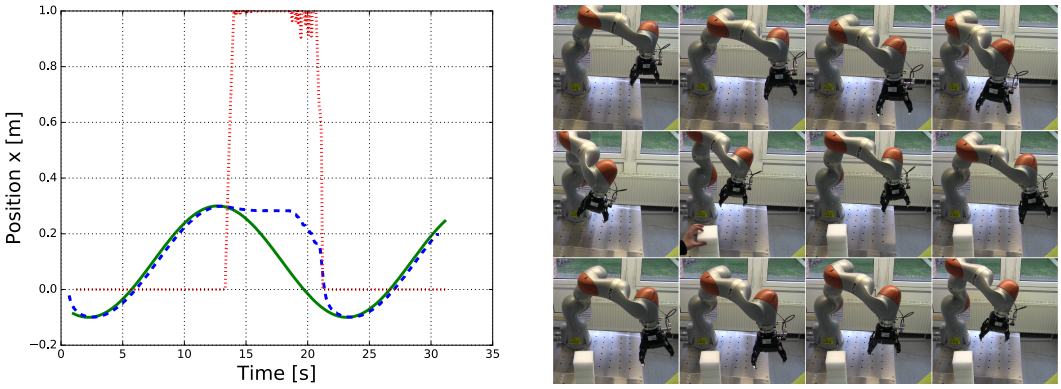
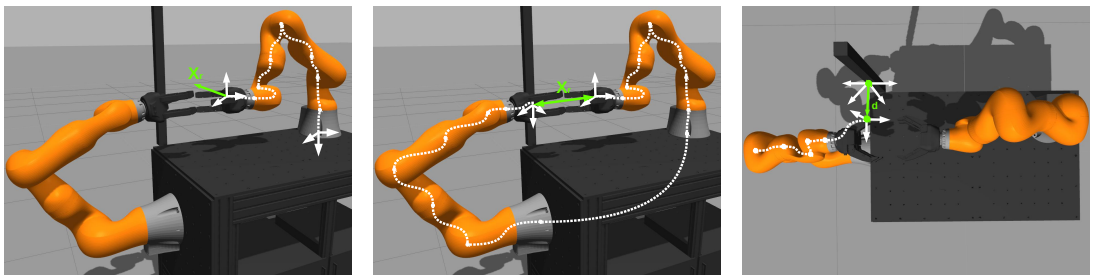


Figure 11.: Results on collision avoidance control. *Left:* Trajectory following behavior. –: Desired end effector x-position, – –: Actual position, · · ·: Weight value of the collision avoidance constraint (Eq. 8). *Right:* Snapshots taken from a video of the experiment.



(a) *Step Response* subtask (b) *Keep Parallel* subtask (c) *Avoid Collisions* subtask

Figure 12.: Illustration of subtasks used in the experimental setup. The dotted white lines indicate which parts of the robot are used for a subtask.

4.4. Optimization of Control Parameters

For parameter optimization within our constraint-based control framework we use a genetic algorithm (GA) from the DEAP evolutionary computation framework [31]. The choice was motivated by the fact that the fitness function comprises multiple extrema. Finding the global maximum in such a fitness landscape can be achieved by using evolutionary optimization techniques like GA.

Given that GA's typically require many computations of the fitness value, we use the GAZEBO 7.0 simulator [32] for fitness evaluation. As experimental setup, we select the scenario illustrated in Fig. 12. We apply a step input to the end effector of the right arm (*Step Response* subtask). The end effector of the left arm shall move parallel to the right arm (*Keep Parallel* subtask) and additionally avoid collisions with obstacles (*Avoid Collisions* subtask)[4]. An obstacle is given by the vertical pole, which is in the path of the left gripper. Its position is fixed and known in this case, since we do not want perception errors to influence optimization. This experimental setup includes many of the commonly encountered problems in constraint-based robot control, like conflicting constraints, dual arm coordination and $n > 6$ degrees of freedom. Each

[4]We only consider collisions of the left gripper in the optimization. However, the solution can be extended to include all links of the robot.

individual encoded in the GA comprises the following parameters:

$$\begin{Bmatrix} k_p^s & w^s & \bar{y}^s \\ k_p^k & w^k & \bar{y}^k \\ k_p^a & d_0^a & \bar{y}^a \end{Bmatrix} \tag{11}$$

The superscripts denote the three subtasks (**s**tep response, **k**eep parallel and **a**void collisions), respectively. The parameters include the proportional control gains k_p, the subtask weights w, the maximum control outputs $\bar{y}$, as well as the maximum influence distance d_0 of the potential field controller (see Eq. 1 to 5 for explanation of the parameters). The task weights of the avoidance controller are computed automatically according to Eq. (8), which is why they will not be optimized. Since Cartesian positioning and avoidance tasks have 6 and 3 degrees of freedom (DoF), respectively, we actually have $6 \cdot 3 + 6 \cdot 3 + 3 \cdot 3 = 45$ free parameters. For simplification, we set the parameters of all DoF of a subtask to the same value, which reduces this number to 9. We use 100 individuals per generation and initialize them with random values within sane bounds. The global fitness function for each experiment is computed according to Eq. 10 as follows:

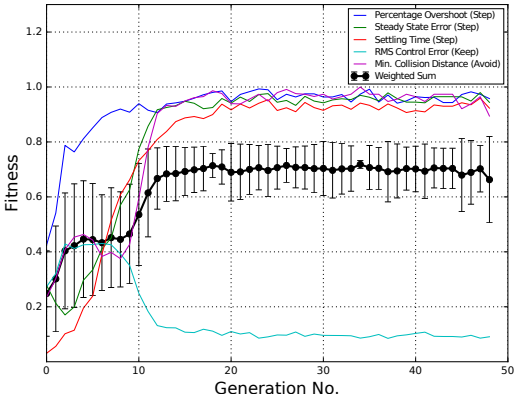
$$f = \xi \cdot (\alpha^s \cdot f^s + \alpha^k \cdot f^k + \alpha^a \cdot f^a) \tag{12}$$

where ξ is the regularization term and f^i, $i \in \{s, k, a\}$ the fitness measures for the individual subtasks, which are chosen as follows

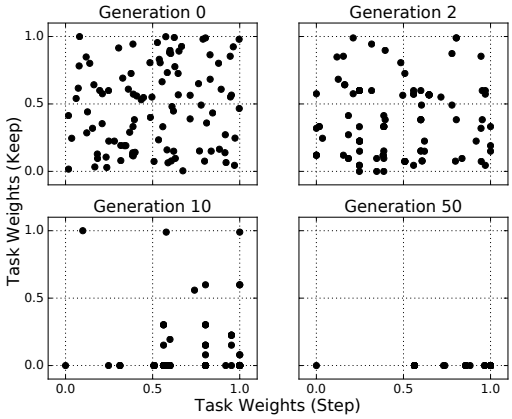
$$\begin{aligned} f^s &= (f_{ss} + f_{po} + f_{st})/3 \quad \alpha^s = 0.2 \\ f^k &= f_{RMS} \quad \alpha^k = 0.3 \\ f^a &= f_{cd} \quad \alpha^a = 0.5 \end{aligned} \tag{13}$$

We compute the fitness of the *Step Response* subtask using the mean of three different performance indicators, namely the steady state error f_{ss}, percentage overshoot f_{po} and settling time f_{st} of the position controller (see section 3.4). For the *Keep parallel* subtask, we choose the root mean square control error, since the left arm should follow the right arm during the complete motion as precisely as possible. For the *Avoid Collisions* subtask, we select the minimum collision distance as fitness measure. As it can be seen, we emphasize the *Avoid Collisions* (highest weight $\alpha^a = 0.5$) and the *Keep Parallel* (medium weight $\alpha^k = 0.3$) subtasks. The latter makes sense when e.g. carrying an object with both arms, which shall obviously not fall down.

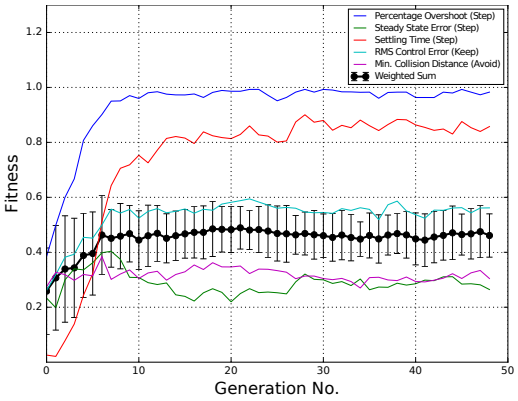
Figure 13c shows the development of the individual fitness functions and the global fitness during genetic optimization. The algorithm converges after approximately 10 generations. Given the fact that the constraints in our experimental setup are conflicting (positioning vs. collision avoidance), an optimal achievement of all subtasks is not possible. As a result, the optimizer may sometimes favor results where one subtask is fully achieved, while the performance of another subtask goes to zero. This problem is illustrated in Fig. 13a. Here, the optimization algorithm converges to a solution, where the task weights of the *Keep Parallel* subtask tend towards zero (see Fig. 13b). As a consequence, the right arm does not move at all, which leads to high fitness for



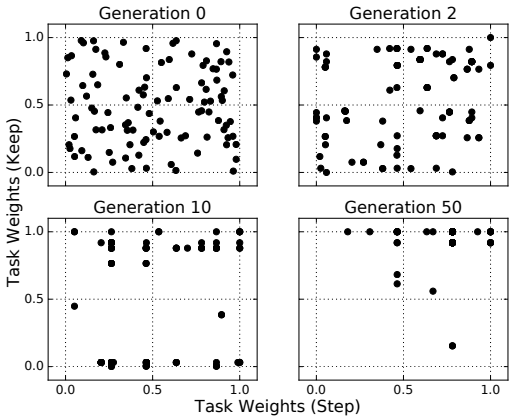
(a) Fitness measures without regularization term



(b) Task weights without regularization term



(c) Fitness measures with regularization term



(d) Task weights with regularization term

Figure 13.: Optimization results after 50 generations of the genetic algorithm. Development of individual fitness measures and the global fitness. Development of subtask weights of the *Keep Parallel* and the *Step Response* subtask

the *Avoid Collisions* subtask, zero fitness of *Keep Parallel* subtask and an average overall fitness. The introduction of the regularization term in Eq. 10 avoids such kind of situations by punishing results with low fitness values of the individual subtasks. Figure 13c shows that a trade off between the values of the individual fitness values is achieved and none of the performance functions converges to zero. The resulting parameter set after genetic optimization is shown in Table 2. We compare this parameter set with our previously used parameters that have been found by manually tuning the individual subtasks one at a time. The most prominent difference between the parameter sets is the proportional gain value of the *Avoid Collisions* subtask, which ends up with a very low value after optimization. This results in different behavior, especially when entering the influence distance of an obstacle. Figure 14a shows the step response of the left end effector in the proximity of the obstacle. Obviously the control gains for the manually selected parameter set have been chosen too high for this specific scenario, which results in a suboptimal transient behavior. In our case, the parameter optimization strategy automatically avoids this problem by minimizing the overshoot and settling time and thus proposing a lower proportional gain for the

Subtask	Optimized Parameter Set		
Step	$k_p^s = 2.651$	$w^s = 1.000$	$\bar{y}^s = 0.361$
Keep	$k_p^k = 1.578$	$w^k = 0.919$	$\bar{y}^k = 0.371$
Avoid	$\mathbf{k_p^a = 0.001}$	$d_0^a = 0.654$	$\bar{y}^a = 0.768$
Subtask	Manually Tuned Parameter Set		
Step	$k_p^s = 1.500$	$w^s = 1.000$	$\bar{y}^s = 0.200$
Keep	$k_p^k = 2.500$	$w^k = 1.000$	$\bar{y}^k = 0.200$
Avoid	$\mathbf{k_p^a = 0.100}$	$d_0^a = 0.500$	$\bar{y}^a = 0.500$

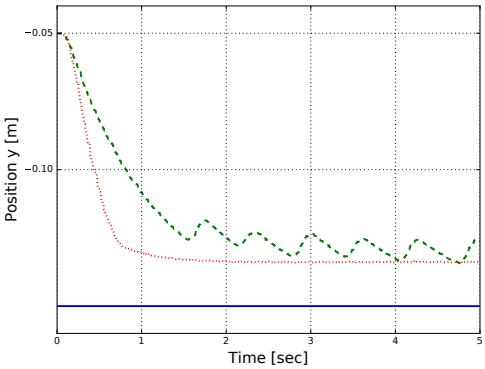
Fitness	f^s	f^k	f^a	f
Optimized	**0.956**	0.600	0.611	0.347
Manual	**0.618**	0.600	0.615	0.314

Table 2.: Optimization results: Optimized and manually tuned parameter set. Individual fitness measures f^s, f^k, f^a and global fitness f for the respective parameter sets

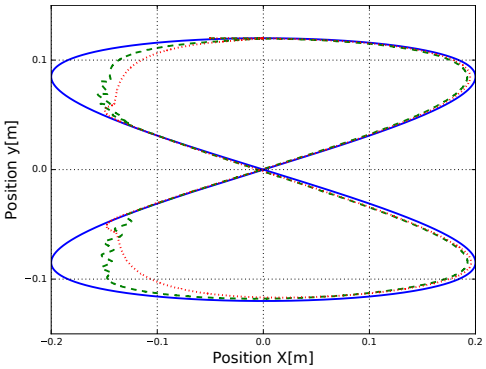
potential field controller. The fitness values for the individual subtasks and the global fitness for both parameter sets are also shown in Table 2. As it can be seen, the *Keep Parallel* and *Avoid Collisions* subtasks perform similarly for both parameter sets. The *Step Response* subtask on the other hand performs much worse for the manually tuned parameter set, which is due to the long settling time of the controller, as can also be seen in Fig. 14a. Having found an optimal control parameter set in an automatized manner like this, we will evaluate its quality in a human-robot coexistence scenario in the following section.

4.5. Human-Robot Coexistence Scenario

We transfer the optimal parameter set found in simulation to the real robotic system and evaluate it in a human-robot coexistence scenario. The left robot arm is supposed to follow a trajectory, while a human enters the workspace, blocking the path of the robot. Again, the right arm should imitate the motion of the left arm. Figure 15 shows snapshots from a video of the human-robot collision avoidance. Figure 14b compares the reference and actual trajectory of the left end effector using the optimized and the manually tuned parameter set. Again, it can be seen that the optimized parameter set provides a smoother transient behavior when the collision avoidance is activated. Far away from any collision, the end effector trajectory is followed accurately. This shows that the approach is generally able to create a suitable parameter set for such kind of complex robotic task in an automatized manner. In this case, the parameter set we found provides safe and jerk-free reactions in a human-robot coexistence scenario.



(a) Step response with vertical pole as obstacle (simulation)



(b) Trajectory tracking with human in the workspace (real robot)

Figure 14.: Comparing the robot behavior for the optimized and manually tuned parameter sets. –: Reference trajectory, – –: Trajectory with manually tuned parameters, · · ·: Trajectory with optimized parameters

4.6. Discussion

The combination of constraint-based control and automatized parameter learning techniques seems promising for two reasons. On the one hand, constraints represent low-dimensional descriptors of particular aspects of the control problem that might be easier to learn than the overall task at once. On the other hand, constraint-based robot control provides the flexibility to learn particular subtasks, while manually tuning others.

Naturally, the choice of the fitness function has a large influence on the result of optimization and thus especially the weighting factors α and normalization functions S have to be chosen carefully, a process that may be time-consuming. The five criteria in Eq. 9 have been chosen based on experience and literature review, thus they might not be optimal and finding a single, generic quality criterion for a wide range of tasks is rather difficult. Here, the idea is that each subtask is assigned a certain optimality criterion. Now, when combining subtasks to a new overall task, the corresponding combination of criteria, again, provides an optimality criterion for the new task. Thus, the optimality criterion must not be found from scratch every time, but only the weights of the respective criteria have to be tuned again. Also, once a suitable set of weights is found, transferring them to other robot tasks might be feasible.

5. Conclusions

In this work we presented the integration of different original contributions. At first we described an extension of the KCCD algorithm for collision distance computation. This included (a) Adding the possibility to integrate external objects and (b) An algorithm for rapid conversion of raw point cloud data into convex hull representations understood by KCCD. We showed that the approach can be used for robot-obstacle distance computation in real-time control applications, even with multiple sensors. Secondly, we presented a constraint-based control framework, which is able to generate task-compliant avoidance motions based on potential fields. We showed that it can be used to smoothly integrate collision avoidance with other simultaneously running

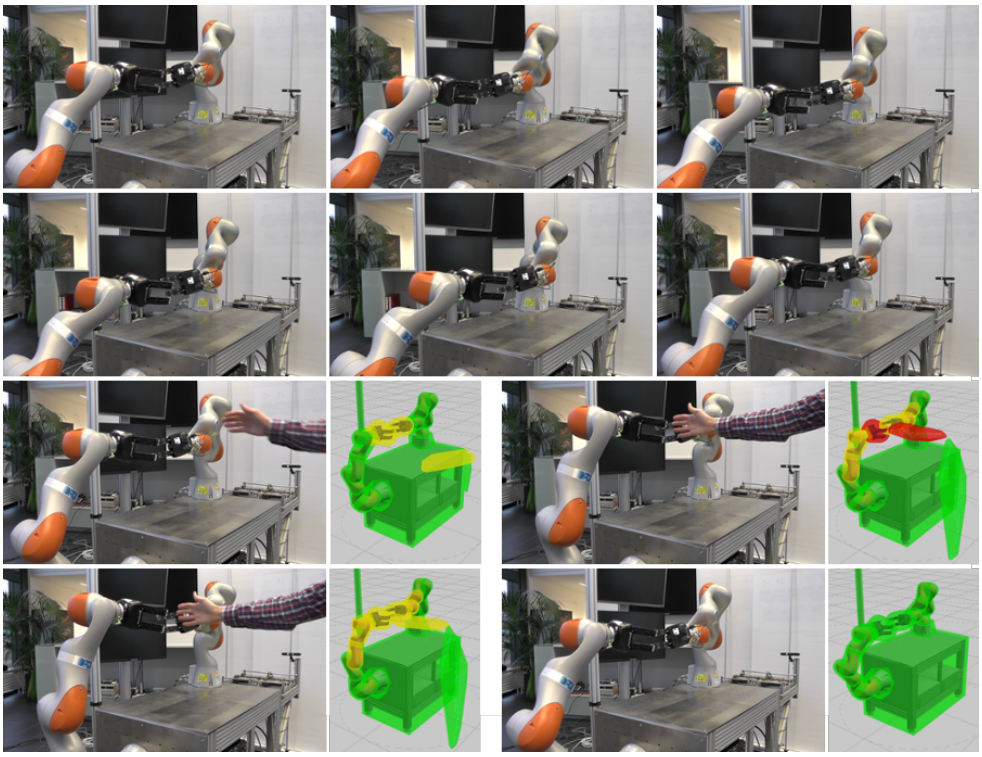


Figure 15.: Screenshots from a video showing trajectory tracking and collision avoidance with a human entering the workspace of the robot.

robot tasks, like trajectory tracking. Finally, we described a method for retrieval of optimal control parameters within this framework based on evolutionary techniques. We discussed different issues of parameter optimization for control tasks with conflicting goals and presented a fitness measure with regularization term to overcome these issues. The presented approach is applicable in human-robot coexistence scenarios in industrial settings.

As future work, we intend to use the collision avoidance approach as a baseline in safe human-robot collaboration scenarios, e.g. for handing over objects or for actual human-robot co-manipulation.

A possible extension of the KCCD convex hull conversion algorithm is a real-time capable solution that more accurately fits non-convex objects as explained in section 3.2. Also the implementation and comparison of other avoidance methods (here we only implemented potential fields) would be of interest.

Considering the automated selection of parameters for constraint-based robot controllers, we simplified the optimization problem in this work (only a few parallel tasks, same control parameters for each degree of freedom) since we wanted to show the principal feasibility of the approach. In future, we would like to apply our framework to more complex problems like humanoid walking or dual-arm, force-based manipulation. In this context a comparison of different learning methods would be useful.

Funding

This work was supported through a grant of the German Federal Ministry of Education and Research (BMBF, FKZ 01IS16026A).

References

[1] KUKA LBR iiwa [www.kuka-lbr-iiwa.com]; 2017. [Online; accessed 28-June-2017].
[2] Universal Robots [www.universal-robots.com]; 2017. [Online; accessed 28-June-2017].
[3] Rethink Robotics [www.rethinkrobotics.com]; 2017. [Online; accessed 28-June-2017].
[4] SafetyEye from PILZ [http://brochures.pilz.nl/bro_pdf/SafetyEYE_2014.pdf]; 2017. [Online; accessed 28-June-2017].
[5] Taeubig H, Frese U. A New Library for Real-time Continuous Collision Detection. In: Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on; May; 2012. p. 1–5.
[6] Smits R, De Laet T, Claes K, et al. iTASC: a tool for multi-sensor integration in robot manipulation. In: Multisensor Fusion and Integration for Intelligent Systems, 2008. MFI 2008. IEEE International Conference on; aug.; 2008. p. 426–433.
[7] Flacco F, Kroeger T, Luca AD, et al. A depth space approach to human-robot collision avoidance. In: Robotics and Automation (ICRA), 2012 IEEE International Conference on; May; 2012. p. 338–345.
[8] Han D, Nie H, Chen J, et al. Dynamic obstacle avoidance for manipulators using distance calculation and discrete detection. Robotics and Computer-Integrated Manufacturing. 2018;49(Supplement C):98 – 104. Available from: http://www.sciencedirect.com/science/article/pii/S0736584516303957.
[9] Taeubig H, Baeuml B, Frese U. Real-time Swept Volume and Distance Computation for Self Collision Detection. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, CA, USA; 9; 2011.
[10] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. In: Robotics and Automation. Proceedings. 1985 IEEE International Conference on; Vol. 2; Mar; 1985. p. 500–505.
[11] Brock O, Khatib O. Elastic strips: A framework for motion generation in human environments. International Journal of Robotics Research. 2002;21(12):1031–1052.
[12] Siciliano B, Slotine JJE. A general framework for managing multiple tasks in highly redundant robotic systems. In: Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on; June; 1991. p. 1211–1216 vol.2.
[13] Sentis L. Synthesis and Control of Whole-body Behaviors in Humanoid Systems [dissertation]. Stanford, CA, USA; 2007. AAI3281945.
[14] Khatib O. A unified approach for motion and force control of robot manipulators: The operational space formulation. IEEE Journal on Robotics and Automation. 1987 February; 3(1):43–53.
[15] Escande A, Mansard N, Wieber PB. Hierarchical quadratic programming: Fast online humanoid-robot motion generation. The International Journal of Robotics Research. 2014; 33(7):1006–1028.
[16] Liu M, Tan Y, Padois V. Generalized hierarchical control. Autonomous Robots. 2016 Jan; 40(1):17–31. Available from: https://doi.org/10.1007/s10514-015-9436-1.
[17] Del Prete A, Nori F, Metta G, et al. Prioritized motion-force control of constrained fully-actuated robots. Robot Auton Syst. 2015 Jan;63(P1):150–157. Available from: http://dx.doi.org/10.1016/j.robot.2014.08.016.
[18] Stasse O, Escande A, Mansard N, et al. Real-time (self)-collision avoidance task on a hrp-2 humanoid robot. In: 2008 IEEE International Conference on Robotics and Automation; May; 2008. p. 3200–3205.

[19] Kanoun O, Lamiraux F, Wieber PB. Kinematic control of redundant manipulators: Generalizing the task-priority framework to inequality task. IEEE Transactions on Robotics. 2011 Aug;27(4):785–792.

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

[21] Lober R, Padois V, Sigaud O. Efficient reinforcement learning for humanoid whole-body control. In: 2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids); Nov; 2016. p. 684–689.

[22] Modugno V, Neumann G, Rueckert E, et al. Learning soft task priorities for control of redundant robots. In: IEEE International Conference on Robotics and Automation (ICRA 2016); May; Stockholm, Sweden; 2016. Available from: https://hal.archives-ouvertes.fr/hal-01273409.

[23] Silvério J, Calinon S, Rozo LD, et al. Learning competing constraints and task priorities from demonstrations of bimanual skills. CoRR. 2017;abs/1707.06791. Available from: http://arxiv.org/abs/1707.06791.

[24] Dehio N, Reinhart RF, Steil JJ. Multiple task optimization with a mixture of controllers for motion generation. In: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); Sept; 2015. p. 6416–6421.

[25] Fernández JdG, Mronga D, Guenther M, et al. Multimodal sensor-based whole-body control for humanrobot collaboration in industrial settings. Robotics and Autonomous Systems. 2017;94:102119. Available from: http://www.sciencedirect.com/science/article/pii/S0921889016305127.

[26] Barber CB, Dobkin DP, Huhdanpaa H. The quickhull algorithm for convex hulls. ACM Trans Math Softw. 1996 Dec;22(4):469–483. Available from: http://doi.acm.org/10.1145/235815.235821.

[27] Pan J, Chitta S, Manocha D. Fcl: A general purpose library for collision and proximity queries. In: 2012 IEEE International Conference on Robotics and Automation; May; 2012. p. 3859–3866.

[28] Siciliano B, Khatib O. Springer handbook of robotics. Berlin, Heidelberg: Springer-Verlag; 2007.

[29] Maciejewski AA, Klein CA. Numerical filtering for the operation of robotic manipulators through kinematically singular configurations. J Field Robotics. 1988;5:527–552.

[30] ASUS Xtion PRO LIVE Product Specification [https://www.asus.com/3D-Sensor/Xtion_PRO_LIVE]; 2017. [Online; accessed 28-June-2017].

[31] Fortin FA, De Rainville FM, Gardner MA, et al. DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research. 2012 jul;13:2171–2175.

[32] GAZEBO - Robot Simulation Made Easy [http://gazebosim.org/]; 2017. [Online; accessed 28-June-2017].