Human-aware motion reshaping using dynamical systems

Matteo Saveriano[a,**], Fabian Hirt[a], Dongheui Lee[a]

[a] *Human-centered Assistive Robotics, Technical University of Munich, Karlstrasse 45, 80333 Munich, Germany*

ABSTRACT

In this work, we present a real-time approach for human-aware motion replanning using a two-level hierarchical architecture. The lower level leverages stable dynamical systems to generate motor commands and to online reshape the robot trajectories. The reshaping strategy modifies the velocity of the robot to match three requirements: *i)* to increase the human safety in case of close interaction with the robot, *ii)* to guarantee the correct task execution in case of unforeseen obstacles (including the human), and *iii)* to replan online the current task taking into account the human behavior. The lower level has to execute all needed computations in real-time. To this end, we also propose a novel approach that leverages depth space structure and parallel programming to rapidly and accurately estimate the robot-obstacle distances. The higher level of the architecture monitors the human and provides to the lower level information about the human status. The proposed approach has been tested in a human robot interaction scenario, showing promising results in terms of safe human-robot coexistence.

1. Introduction

Future robots will be required to actively cooperate with humans to accomplish daily-life activities. The possibility of removing barriers between humans and robots will significantly increase the applicability of robotics co-workers both in industrial and daily-life scenarios. An effective human-robot cooperation has to fulfill several requirements. Human safety during close or physical interaction with the robot is probably the most important requirement to satisfy (De Luca and Flacco, 2012). In current industrial scenarios robots are usually enclosed into cages and stopped if the human enters in the cage. Several studies have been conducted to understand the risk of injuries in case of human-robot accidental collisions (Haddadin et al., 2012), and effective control approaches have been proposed to minimize the severity of possible injuries (Haddadin et al., 2008). Researchers have also investigated the possibility of preventing possible collisions. In (Ikuta et al., 2003; Kulić and Croft, 2006; Lacevic et al., 2013) various danger indexes were suggested to determine how dangerous is the current robot posture for humans or obstacles in the scene. The danger index is then incorporated in the robot control strategy to produce safe trajectories. Reactive collision avoidance approaches (Flacco et al., 2014; Khansari-Zadeh and Billard, 2012; Saveriano and Lee, 2013b, 2014a) can be also applied to limit the risk of collisions with human co-workers.

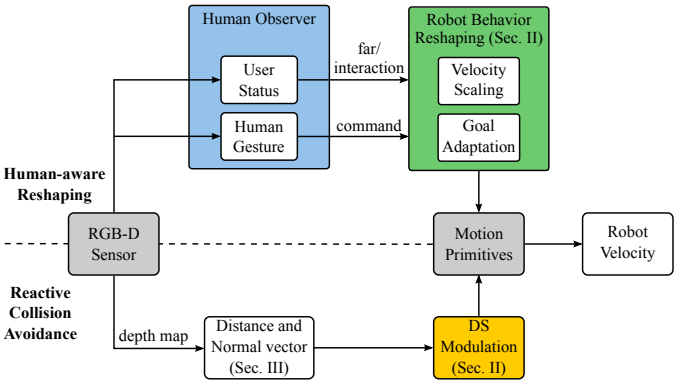


Fig. 1. The two-level hierarchical architecture used for human-aware motion reshaping. An RGB-D sensor is adopted to track a human user and estimate his status. RGB-D data are also used to avoid possible collisions.

A recent study (Lasota and Shah, 2015) has shown that, in a human-robot co-working scenario, human-aware motion adaptation is beneficial for the human partner, improving both the working speed and the human satisfaction. Hence, beyond the safety, an effective human-robot collaboration requires also the correct task execution, and the adaptation of the predefined task according to the human intentions. This aspect of the inter-

**Corresponding author: Tel.: +49-89-289-268840; fax: +49-89-289-26901; *e-mail:* matteo.saveriano@tum.de (Matteo Saveriano)

action, usually neglected in the aforementioned reactive safety strategies, is explicitly considered by geometric planning approaches (Sisbot et al., 2007; Cirillo et al., 2010; Sisbot and Alami, 2012). Geometric planners formulate the human-robot interaction as an optimization problem, where the cost to minimize depends on various aspects like human and robot kinematics, objects in the scene, human needs and preferences. A grid based representation of the world, which also includes the human and the robot, is usually adopted and the robotic task is computed by finding the minimum-cost path in the grid (Sisbot et al., 2007). Geometric planners are able to generate safe (collision-free), feasible (robot physical limitations) and socially acceptable (human preferences and needs) plans, but the computational cost required for the replanning limits the applicability of geometric planners in dynamic environments.

We aim at developing a motion adaptation approach for human-robot coexistence which takes into account: *i)* human safety, *ii)* correct task execution, and *iii)* real-time task replanning according to the human behavior. To this end, we propose the two-level hierarchical architecture in Fig. 1 for real-time human-aware motion generation. The higher level of the architecture is used to monitor the human. In the current implementation, the *Human Observer* monitors the user status to determine if the human is outside the robot workspace (far) or if the human and the robot share the same working area (interaction). During the interaction phase, the user has the possibility to modify the robot behavior via gestures, forcing the robot to replan the current task. According to the *Human Observer*, the *Robot Behavior Reshaping* module (the lower level) selects the right behavior from a database of Motion Primitives, scales down the velocity and modifies the desired goal position. The main focus of this work is to develop a flexible lower layer that quickly replans the current task. It leverages stable dynamical systems for online task generation and replanning, and it implements a reactive collision-avoidance approach (DS Modulation block) to produce collision-free paths. Depth data from a RGB-D sensor are used to track the human and measure the distance of the robot from eventual obstacles. The Distance and Normal vector block can execute all the computations in real-time using the novel approach presented in Sec. 3.

The main contribution of this work is a unified framework which can handle both human action monitoring and reactive motion generation. In particular, we focus on developing a flexible motion generation layer that quickly replans the robot task in order to avoid possible collisions and execute eventual human commands. Preliminary versions of this work were presented in (Saveriano and Lee, 2013b, 2014a,b). In this paper, we extend our previous work as follows: *1)* we describe in detail our framework, especially concerning the collision avoidance with the robot body; *2)* we propose an approach for real-time distance estimation; *3)* we present novel experiments in a dynamic environment; *4)* we compare our distance estimation algorithm with a state-of-the-art one; *5)* we extend the *Human Observer* with command interface based on human gesture recognition.

The rest of the paper is organized as follows. Section 2 presents our approach for online motion replanning using dynamical systems. Section 3 describes a novel approach to compute robot-obstacle distances in real-time. Results in a human-robot collaborative scenario are presented in Sec. 4. Section 5 states the conclusions and proposes further extensions.

2. Online motion replanning using dynamical systems

This section describes the proposed approach for real-time motion generation and adaptation using Dynamical Systems (DS). The proposed reshaping strategy generates a robot velocity command in the form

$$\dot{\boldsymbol{p}} = \alpha_h \boldsymbol{M}(\boldsymbol{p}) \boldsymbol{f}(\boldsymbol{p} - \boldsymbol{g}) \tag{1}$$

where α_h is a velocity scaling term, $\boldsymbol{M}(\boldsymbol{p})$ is the so-called modulation matrix which generates collision-free paths, and $\boldsymbol{g}$ is the target position to reach. A detailed description of the reshaped velocity in (1) is given in the rest of the section.

2.1. Motion generation

The proposed reshaping strategy assumes that the robot trajectory is generated using a first-order, globally stable DS

$$\dot{\boldsymbol{p}} = \boldsymbol{f}(\boldsymbol{p}) \tag{2}$$

where $\boldsymbol{f}(\cdot)$ is a continuous function and $\boldsymbol{p} \in \mathbb{R}^3$ and $\dot{\boldsymbol{p}} \in \mathbb{R}^3$ represent the robot end-effector position and velocity respectively. Recall that the global asymptotic stability of a DS implies that $\boldsymbol{p} \longrightarrow \boldsymbol{p}^*$ for $t \longrightarrow \infty$, where $\boldsymbol{f}(\boldsymbol{p}^*) = \boldsymbol{0}$ (Slotine and Li, 1991). Stable DS are well suited to represent point-to-point motions, since they are guaranteed to convergence to a specified target. Driving robots with DS has also several advantages in terms of robustness to external perturbations, such as unexpected contacts or changes in the goal/initial position. Moreover, DS can be combined to generate more complicated tasks, e.g., assembly tasks. A stable DS in the form (2) can be designed by an expert user, or, more conveniently, learned from human demonstrations (Ijspeert et al., 2013; Neumann and Steil, 2015).

2.2. Real-time goal adaptation

Stable dynamical systems can be combined to execute complex robotic tasks consisting of several point-to-point motions. A typical example is an assembly task, where the robot has to pick several items and to put them together. In an assembly task, the DS equilibrium position represents the position of a particular item, or the position where the item has to be placed. In a realistic co-working scenario, item positions are not fixed across several executions and the robot has to be able to suddenly adapt to changes in the desired target position. DS are robust to changes in the equilibrium position. Indeed, convergence towards a different goal is achieved by modifying (2) as

$$\dot{\boldsymbol{p}} = \boldsymbol{f}(\boldsymbol{p} - \boldsymbol{g}) \tag{3}$$

where $\boldsymbol{g}$ is the novel goal (equilibrium) position. It is worth noticing that, if the DS in (2) is asymptotically stable, the DS in (3) converges to $\boldsymbol{g}$.

2.3. Velocity scaling

The severity of possible injuries due to an accidental collision between the human and the robot depends on the velocity of the robot during the impact (Haddadin et al., 2012). It is clear that impacts at high speed have higher probability to generate serious injuries. Hence, it is of importance for a safe human-robot interaction to scale down the velocity of the robot to a safe value (Haddadin et al., 2012), without affecting the execution of the task (i.e., reaching a goal). A human-aware velocity scaling algorithm is implemented by modifying (2) as

$$\dot{\boldsymbol{p}} = \alpha_h \boldsymbol{f}(\boldsymbol{p}) \tag{4}$$

where $0 < \alpha_{min} \leq \alpha_h \leq 1$ is a scalar function proportional to the distance between a human and a robot. If the user enters in the workspace of the robot, $\alpha_h = \alpha_{min}$ and the velocity of the robot is scaled to a safe value. When the user is outside the workspace, $\alpha_h = \alpha_{max}$ and the task is executed at maximum speed. Being α_h strictly positive, the DS in (4) converges to the goal and the task is always correctly executed.

2.4. DS modulation for reactive collision avoidance

Modulate a DS means, in this work, finding a certain state dependent matrix $\boldsymbol{M}(\boldsymbol{p})$ such that the modulated DS

$$\dot{\boldsymbol{p}} = \boldsymbol{M}(\boldsymbol{p})\boldsymbol{f}(\boldsymbol{p}) \tag{5}$$

satisfies certain properties. In particular, we are interested in modifying the robot velocity to avoid possible collisions with obstacles in the scene without affecting the goal of the modulated DS. An approach to find such a matrix $\boldsymbol{M}(\boldsymbol{p})$ is proposed in (Khansari-Zadeh and Billard, 2012), leveraging an analytical representation of the obstacle surfaces. The work in (Khansari-Zadeh and Billard, 2012) is extended in (Saveriano and Lee, 2013b, 2014a) to consider a more realistic scenario in which objects are represented via point clouds and an analytical representation of their surfaces is not available. In particular, the modulation matrix proposed in (Saveriano and Lee, 2013b) uses the Euclidean distance between pairs of points instead of the analytical representation of the obstacle surface.

2.4.1. Modulation with a point-mass robot

The key idea of the DS modulation for collision avoidance is to reduce the velocity of the robot in the direction normal to the obstacle surface and to project the motion into a plane tangent to the obstacle (tangential hyperplane). In this way, the modulated DS in (5) generates an avoiding motion which locally deforms the DS trajectories and prevents possible collisions (Khansari-Zadeh and Billard, 2012; Saveriano and Lee, 2013b).

Let us first focus on collision avoidance for the end-effector with a single obstacle. The case of multiple obstacles, as well as the possibility to avoid collisions with other links of the robot, will be discussed later in this section. Consider that one d-dimensional obstacle is present in the workspace of the robot, e.g., the obstacle close to the end-effector of the robot in Fig. 2. Assume also that the normal vector to the obstacle surface

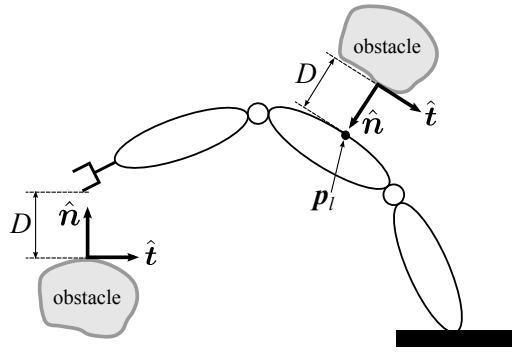


Fig. 2. Representation of two obstacles in the workspace of the robot. The DS modulation requires the distance D between a point on the robot and a point on the obstacle, and the vector $\hat{\boldsymbol{n}}$ normal to the surface of the obstacle. In the 2D case, the tangential hyperplane is the vector $\hat{\boldsymbol{t}}$ tangent to the obstacle surface and orthogonal to $\hat{\boldsymbol{n}}$.

$\hat{\boldsymbol{n}}(\bar{\boldsymbol{p}}) = [\hat{n}_1(\bar{\boldsymbol{p}}) \ \cdots \ \hat{n}_d(\bar{\boldsymbol{p}})]^T$ is defined for all $\bar{\boldsymbol{p}}$. The tangential hyperplane can be defined at each point on the surface as

$$\hat{t}_i^j(\bar{\boldsymbol{p}}) = \begin{cases} -\hat{n}_{i+1}(\bar{\boldsymbol{p}}) & j = 1 \\ \hat{n}_1(\bar{\boldsymbol{p}}) & j = i+1 \ \ i = 1..d-1, j = 1..d \\ 0 & j \neq 1, j \neq i+1 \end{cases} \tag{6}$$

where $\hat{t}_i^j$ corresponds to the j-th component of the i-th basis vector. The matrix $\boldsymbol{V}(\bar{\boldsymbol{p}}) = \left[\hat{\boldsymbol{n}}(\bar{\boldsymbol{p}}) \ \hat{\boldsymbol{t}}_1(\bar{\boldsymbol{p}}) \ \cdots \ \hat{\boldsymbol{t}}_{d-1}(\bar{\boldsymbol{p}})\right]$ represents an orthonormal basis of the d-dimensional space.

Given the distance $D(\bar{\boldsymbol{p}}_m)$ between the manipulator and the surface of the obstacle, where $\bar{\boldsymbol{p}}_m$ is the point of minimum distance, it is possible to define $\boldsymbol{E}(\bar{\boldsymbol{p}}_m) = \text{diag}(\lambda_1(\bar{\boldsymbol{p}}_m), \ldots, \lambda_n(\bar{\boldsymbol{p}}_m))$, where

$$\begin{cases} \lambda_1(\bar{\boldsymbol{p}}_m) = 1 - \dfrac{1-\epsilon}{D(\bar{\boldsymbol{p}}_m)+1} \\ \lambda_i(\bar{\boldsymbol{p}}_m) = 1 + \dfrac{1}{D(\bar{\boldsymbol{p}}_m)+1} \quad i = 2, 3, \ldots d \end{cases} \tag{7}$$

and $\epsilon > 0$ is an arbitrary small constant ($\epsilon = 10^{-5}$ in this work), used to avoid that $\lambda_1(\bar{\boldsymbol{p}}_m) = 0$. The modulation matrix in (5) is

$$\boldsymbol{M}(\bar{\boldsymbol{p}}_m) = \boldsymbol{V}(\bar{\boldsymbol{p}}_m)\boldsymbol{E}(\bar{\boldsymbol{p}}_m)\boldsymbol{V}(\bar{\boldsymbol{p}}_m)^{-1} \tag{8}$$

The modulation matrix in (8) is symmetric and positive definite by construction. This implies that the equilibrium points of the modulated DS remain unchanged, i.e., $\boldsymbol{M}(\boldsymbol{p})\boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{0} \leftrightarrow \boldsymbol{f}(\boldsymbol{p}) = \boldsymbol{0}, \forall \boldsymbol{p}$. The eigenvalue $\lambda_1(\bar{\boldsymbol{p}}_m)$ in (7) tends to ϵ for $D(\bar{\boldsymbol{p}}_m) \to 0$. Being $\lambda_1(\bar{\boldsymbol{p}}_m)$ the velocity component along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{p}})$, the robot does not generate motions towards the obstacle if[1] $D(\bar{\boldsymbol{p}}_m) = 0$. On the contrary, the robot is free to move in the tangential hyperplane ($\lambda_i(\bar{\boldsymbol{p}}_m) \geq 1$). In this way, the DS trajectories are locally deformed to generate collision-free paths.

The described modulation approach directly applies in case of multiple obstacles. Being the modulation matrix updated at each time step, one has to simply avoid the closest obstacle at the current time (Saveriano and Lee, 2013b).

[1]The velocity of the robot along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{p}})$ is zero if $\epsilon = 0$. Having chosen $\epsilon = 10^{-5}$ the velocity component along $\hat{\boldsymbol{n}}(\bar{\boldsymbol{p}})$ is, in practice, negligible.

2.4.2. Modulation considering the robot body

The described DS modulation approach considers the robot as a point-mass object. In order to apply DS modulation in real scenarios, two problems have to be considered. First, the end-effector of the robot is not a point but a rigid body. Second, collisions may occur with other links apart from the end-effector.

There are mainly two possibilities to take into account the physical dimensions of the robot links when computing robot-obstacle distances. The first is to cover the robot body with basic primitives like spheres and to compute the distance of the obstacles from the surface of the sphere. Alternatively, one can compute the distance of the obstacles form the surface of each link using a triangular mesh model. This second possibility produces a more accurate estimation of the distance, but it is computationally expensive. A novel approach for real-time distance computation using mesh models is presented in Sec. 3.

Avoiding collisions with the end-effector does not prevent the robot to hit an obstacle with one of its links. To avoid collisions with the robot body, an approach for link collision avoidance has to be implemented. To this end, we combine multi-priority inverse kinematics (Siciliano et al., 2009) with the DS modulation. Recall that the general solution for the multi-priority inverse kinematics problem is given by

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{p}} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\dot{\boldsymbol{q}}_N \tag{9}$$

where $\dot{\boldsymbol{q}}$ is the joint velocity, $\dot{\boldsymbol{p}}$ is end-effector velocity, $(\cdot)^{\dagger}$ denotes the pseudo-inverse of a matrix and $\boldsymbol{J}(\boldsymbol{q})$ is the manipulator Jacobian. The second term in (9) projects an arbitrary velocity $\dot{\boldsymbol{q}}_N$ into the robot null-space to generate joint motions that do not affect the end-effector velocity. Null-space motions are exploited in this work to avoid collisions with the robot links.

The end-effector velocity $\dot{\boldsymbol{p}}$ in (9) is the reshaped velocity in (1). As already mentioned, this guarantees a collision-free path for the end-effector. For the null-space avoidance, we propose to modulate the velocity of the closest point to the robot body $\boldsymbol{p}_l$ (see Fig. 2). Hence, we choose for the null-space motion the avoiding velocity $\boldsymbol{M}(\boldsymbol{p}_l)\dot{\boldsymbol{p}}_l$, which becomes the joint velocity $\dot{\boldsymbol{q}}_N = \boldsymbol{J}_l^{\dagger}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{p}_l)\dot{\boldsymbol{p}}_l$. The resulting joint velocity command is

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\dagger}(\boldsymbol{q})\dot{\boldsymbol{p}} + (\boldsymbol{I} - \boldsymbol{J}^{\dagger}(\boldsymbol{q})\boldsymbol{J}(\boldsymbol{q}))\boldsymbol{J}_l^{\dagger}(\boldsymbol{q})\boldsymbol{M}(\boldsymbol{p}_l)\dot{\boldsymbol{p}}_l \tag{10}$$

where $\boldsymbol{J}_l(\boldsymbol{q})$ is the Jacobian of $\boldsymbol{p}_l$. The velocity command in (10) considers the link collision avoidance as a secondary task which does not affect the end-effector motions. This approach guarantees the correct end-effector task execution (i.e., convergence to the target and collision avoidance) and, in many situations, it prevents collisions with the robot body. If collision avoidance is the most important task for a certain application, task priorities can be changed to guarantee that the collision avoidance has always priority (Falco and Natale, 2014; An and Lee, 2015).

3. Distance computation in the depth space

An usual approach to use the spatial scene information for distance evaluation algorithms is to transform the data given by the visual sensors into 3D point clouds (Bascetta et al., 2010; Pan et al., 2013). However, operating directly on the depth image provided by a depth sensor reduces the computation time (Flacco et al., 2012, 2014). This performance improvement mainly depends on structure of the depth space, which is efficiently exploited to speed-up the computation. This section discusses the problem of robot-obstacle distance calculation in the depth space. We briefly describe the structure of the depth space and how distances can be directly computed in the depth space, and then we present a novel approach for fast Distance Evaluation in the Depth space (fDED).

3.1. Depth space

Depth sensors represent each observed Cartesian point $\boldsymbol{p}$ with three values. Two values represent the coordinates of the projection of $\boldsymbol{p}$ in the depth sensor plane. In this work, these coordinates are indicated with the subscripts x and y. The third value represents the distance (depth) of $\boldsymbol{p}$ from the sensor plane. Hence, the depth space is a non-homogeneous 2.5-dimensional space. Let us consider a generic Cartesian point expressed in a world frame $\boldsymbol{p}_w = [x_w, y_w, z_w]^T$. $\boldsymbol{p}_w$ can be expressed in the depth sensor frame by applying the homogeneous transformation $\boldsymbol{p}_c = \boldsymbol{R}\boldsymbol{p}_w + \boldsymbol{t}$. Assuming a pin-hole camera model (Ma et al., 2003), $\boldsymbol{p}_c = [x_c, y_c, z_c]^T$ is mapped into a depth point by

$$\boldsymbol{p}_d = \begin{bmatrix} p_x \\ p_y \\ d_p \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{11}$$

where p_x and p_y are the pixel coordinates of the projected point, f_x and f_y are the focal lengths of the camera in pixel units, and c_x and c_y are the pixel coordinates of the camera's optical center.

Projections in the depth space generate occlusions, the so-called gray area. In fact, a point $\boldsymbol{p}_c$ with depth d_p occludes all the points having depth $d \geq d_p$ and lying on the virtual line connecting the camera center and $\boldsymbol{p}_c$ (see Fig. 3). The gray area is thus a region of uncertainty, since no information on this region can be stored in the depth space representation.

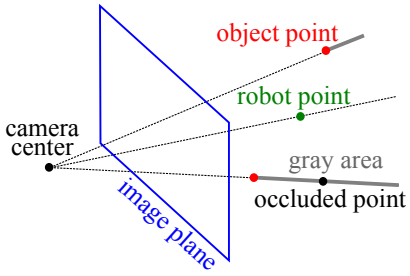


Fig. 3. Representation of the gray area.

3.2. Computing the Cartesian distance in the depth space

Given the depth representations of a point on the robot $\boldsymbol{p}_{R_d} = [r_x, r_y, d_r]^T$ and a point in the scene $\boldsymbol{p}_{O_d} = [o_x, o_y, d_o]^T$, the distance between $\boldsymbol{p}_{R_d}$ and $\boldsymbol{p}_{O_d}$ is (Flacco et al., 2012)

$$d(\boldsymbol{p}_{R_d}, \boldsymbol{p}_{O_d}) = \sqrt{(v_x)^2 + (v_y)^2 + (v_z)^2} \tag{12}$$

where $v_x = [(o_x - c_x)d_o - (r_x - c_x)d_r]/f_x$, $v_y = [(o_y - c_y)d_o - (r_y - c_y)d_r]/f_y$, and $v_z = d_o - d_r$.

To take into account the gray area, the depth value d_o of the object point $\boldsymbol{p}_{O_d}$ is changed to

$$d_o = \begin{cases} d_r & \text{if } d_r \geq d_o, \\ d_o & \text{otherwise} \end{cases} \tag{13}$$

In other words, if the robot point is behind the scene point ($d_r > d_o$), the depth of the object point is $d_o = d_r$ to consider possible occluded points (see Fig. 3).

3.3. Fast distance evaluation in the depth space

Equation (12) can be used to evaluate the distance of each object point from each link of the robot. To reduce the computation time, the robot body can be approximated using simple geometric primitives, i.e., cylinders or spheres, as in (Flacco et al., 2012, 2014). Instead of approximating the robot, we use a triangular mesh (CAD) model of the robot, providing more accurate and realistic distance estimations. The main idea of the proposed fDED is to reduce the computation time by reducing the number of considered points using a lattice of robot and a lattice of objects points, directly computed in the depth space. The resulting algorithm is highly parallelizable, which allows the parallel execution of the needed computations on the Graphics Processing Unit (GPU). An overview of the proposed approach is shown in Fig. 4.

3.3.1. Removing the robot from the depth image

The raw depth image from the sensor (real depth) contains also points belonging to the robot surface. Obviously, points on the manipulator should not be treated as obstacles to be avoided. The robot can be efficiently removed from the depth image given its mesh model and its current joint configuration (see the top panel in Fig. 4). First, the mesh model of the robot is projected into the depth space to create a depth map which contains only robot points (virtual depth). The projection is computed by applying the transformation (11) on each vertex of the robot model. Being the robot model composed by one mesh model for each link, we attach an index to each point to determine to which link a certain point belongs to.

After this step, each point on the robot $\boldsymbol{p}_R$ has been transformed into a depth point $\boldsymbol{p}_{R_d} = [r_x, r_y, d_r]^T$. As detailed in the previous sections, the position of $\boldsymbol{p}_{R_d}$ in the depth plane is determined by the pixel coordinates r_x and r_y. Given the virtual depth, the robot is removed from the real depth map via a pixel-wise comparison. In more detail, we extract from the real depth map the depth value d of the point located at (r_x, r_x). If $|d - d_r| \leq \delta$, the point belongs to the robot and it is removed from the real depth map. In this work, we use $\delta = 5\,\text{cm}$. Repeating the pixel-wise comparison for all the points in the virtual depth map, we obtain the filtered depth map in Fig. 4 with the robot removed. Described operations can be executed separately for each point and can be executed in real-time ($< 1\,\text{ms}$) on the GPU[2].

[2]The code for robot filtering is available online: github.com/blodow/ real-time_urdf_filter

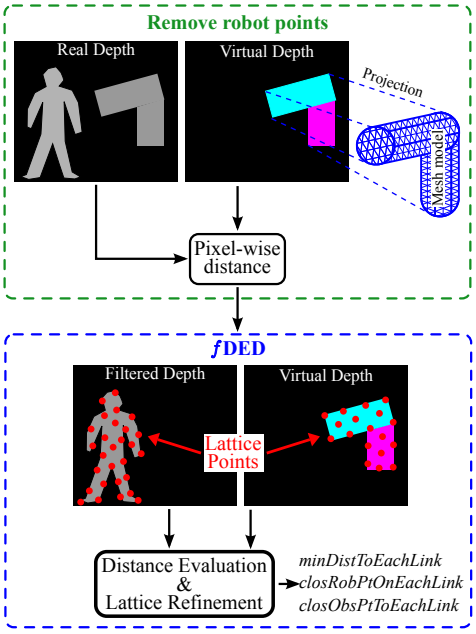


Fig. 4. Overview of the fDED approach.

3.3.2. Lattice of robot points

Computing the pair-wise distance of every point in the virtual depth from every point in the filtered depth is computationally unfeasible in real-time ($\approx 1\,\text{ms}$). For this reason, we propose an approach to drastically reduce the number of considered points without significantly affecting the accurate distance evaluation. Instead of considering all the robot points, we subsample them building a lattice of robot points. The lattice of robot points has to meet, simultaneously, two requirements: *1)* the lattice has to cover the entire robot, and *2)* the lattice has to be fine enough to produce accurate results. To fulfill both requirements and reduce the robot points, the following procedure is proposed:

I. A (quasi-) equally spaced, coarse lattice of robot points is created (see Fig. 5). The lattice covers the entire robot.

II. The distances of the robot lattice points to all objects in the scene are calculated.

III. For each link, the closest robot lattice point $\boldsymbol{p}^*_{R_l}$ to an object is determined.

The robot lattice points is created by dividing the virtual depth into squared tiles (see Fig. 5). The initial side length T_s of the tiles is a tunable parameter. In each tile, a different thread checks if there are robot points. The robot point closest to the center of the tile is selected as a lattice point. Each tile can contain at most one lattice point for each link. Tiles that do not contain any robot point are simply dropped out. After the closest robot lattice point $\boldsymbol{p}^*_{R_l}$ to an object is determined (step III), steps II and III are repeated for all the points in the tile around $\boldsymbol{p}^*_{R_l}$. This refinement step is needed to increase the accuracy, otherwise bounded by the size of the tile. The described procedure can be executed in parallel for each lattice point, which allows a GPU implementation.

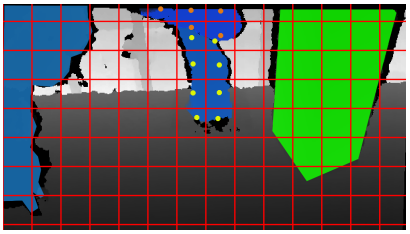


Fig. 5. Lattice of robot points creation on the GPU. The blue region represents the robot, the green region the object (a box). Both the robot and the box are standing on a table, which is not considered as an obstacle. Yellow bullets represent the lattice points of the first link, orange bullets represent the lattice points of the second link.

3.3.3. Lattice of object points

The algorithm proposed to create the lattice of robot points assumes that the pixel coordinates of each robot point are known a priory. This assumption is not valid for the filtered depth map, since we do not know a priory which pixels represent the obstacles. Moreover, in order to apply the refinement procedure to object points we should create a lattice of object points for each robot link. Instead of inspecting all the points in the filtered depth map, we propose to skip object points to improve the execution time. The lattice of object points is simply created by skipping pixels in the filtered depth. The spacing between the object lattice points, namely the step size S_x and S_y, is a tunable parameter and it can vary to produce more or less accurate results. Given an object lattice point with pixel (px) coordinates (o_x, o_y), the next lattice point lies outside the region $[o_x + S_x, o_y + S_y]$. $S_x = S_y = 1$ px means that all the points in the filtered depth map are lattice points. As for the lattice of robot points, the algorithm to create a lattice of objects points can be parallelized and executed on a GPU.

3.4. Computing the normal vector

The dynamical system modulation approach presented in Sec. 2 requires the distance of the obstacles from the robot and the normal vectors at the points of minimum distance. In this work, the normal vector is estimated using a parallel implementation of the algorithm in (Marton et al., 2009). Given the point of minimum distance $\boldsymbol{O}_m$ with pixel coordinates (o_x, o_y), a plane π_o tangent to $\boldsymbol{O}_m$ is created using weighted least squares. L points[3] in the neighborhood of (o_x, o_y) are transformed into Cartesian points and used to generate π_o. The normal vector to the plane π_o represents the normal vector at the point $\hat{\boldsymbol{n}}(\boldsymbol{O}_m)$. This algorithm is robust to noise and it requires neither a smooth surface nor a certain density of points.

3.5. Evaluation

The goal of this evaluation is two-fold. First, it shows the real-time capabilities (up to 1 ms) of the proposed approach. Second, it provides a guideline to tune the parameters used to create the point lattices. The evaluation is conducted on the static scene (fixed robots and obstacles) shown in Fig. 5. We preferred a static scene in order to test the algorithm always in the same conditions. A dynamic environment is considered in the case study in Sec. 4. The computer used in this evaluation has an Intel® Core™ *i*7 – 4790*K* - 4 Cores CPU, a GeForce GTX 660 - 960 Cores GPU and 16 GB of memory.

3.5.1. A Guideline for tuning parameters selection

To provide a guideline for the parameters selection, the effects of different parameter sets on the computation time and the accuracy of the *f*DED have to be understood. Recall that the parameters for *f*DED are the initial tile size T_s and the step size (S_x and S_y). We want to verify the following hypothesis:

H_1. Increasing (decreasing) T_s reduces (increases) the computation time and the accuracy.

H_2. Increasing (decreasing) S_x and S_y reduces (increases) the computation time and the accuracy.

In order to consider the effects of noisy data while keeping the same (light) conditions, we recorded 100 depth images and we provide averaged results. The robot is a 7 dof KUKA LWR IV+. The triangular mesh of the robot has 4861 vertexes, while the depth images have 640×480 pixels (px).

Several analysis are conducted with different values of the tuning parameters. The main results are shown in Fig. 6. For comparison consider that a brute force approach which considers all the points takes about $128s$ on the CPU and about $0.9s$ on the GPU. Indeed, in this scenario, 1.5×10^9 distances have to be computed. Being the most accurate approach (all points are considered), results of the brute force algorithm serves as ground truth for the accuracy. To give more compact results, average errors over the 7 links are used in Fig. 6.

As expected, reducing S_x and S_y increases the computation time (note that times are in *ms*) and the accuracy (Fig. 6(a) and 6(c)), since more points have to be considered. Values of S_x and S_y bigger than 32 px penalize too much the accuracy without significantly reducing the computation time. The initial raster tile value T_s affects (slightly) the computation time and the accuracy (Fig. 6(b) and 6(d)). To summarize, hypothesis H_1 and H_2 are verified. Taking into account the conducted analyses, a parameter set that guarantees an execution time of about 1 ms and an accuracy[4] of about 5 mm is shown in Tab. 1.

Table 1. *f*DED - parameter set to have $T \approx 1$ ms and $A \approx 5$ mm ("T" computation time, "A" accuracy).

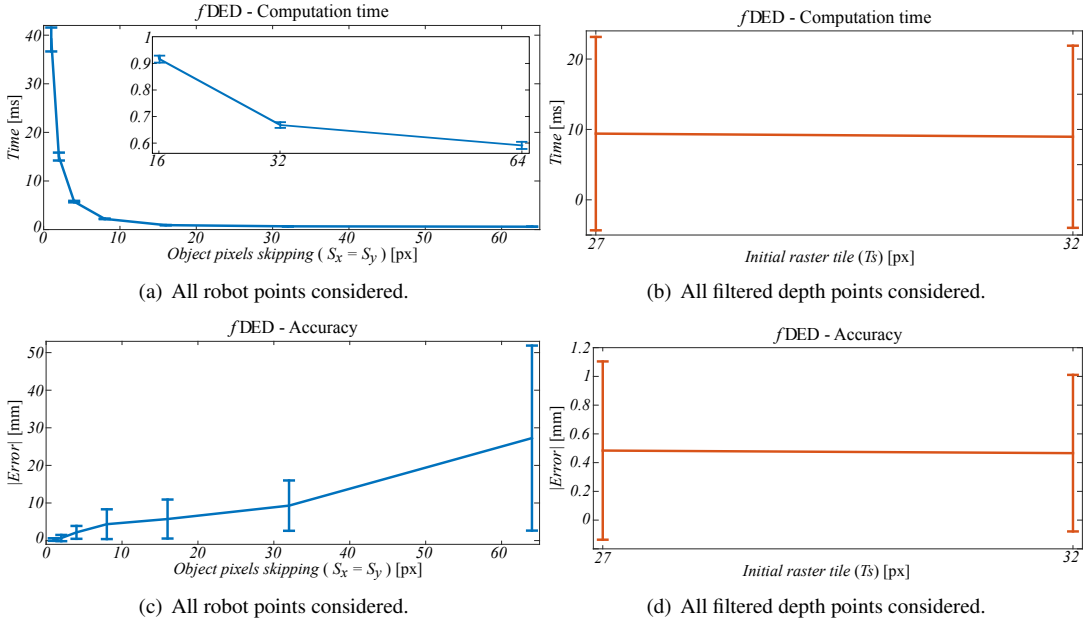
T_s [px]	S_x/S_y [px]	T [ms]	A [mm]
32	16/16	0.9	5

3.5.2. Computation time in the worst case

The worst case scenario for the computation time is obtained by creating a real depth image in which all the 640×480 pixels consist of object points. The robot is located behind those

[3] In experiments we use $L = 1\%$ of the number of points as suggested in (Saveriano and Lee, 2013b).

[4] In our set-up 5 mm of error is widely tolerable considering errors introduced by depth sensors (Khoshelham and Oude Elberink, 2012).



(a) All robot points considered.

(b) All filtered depth points considered.

(c) All robot points considered.

(d) All filtered depth points considered.

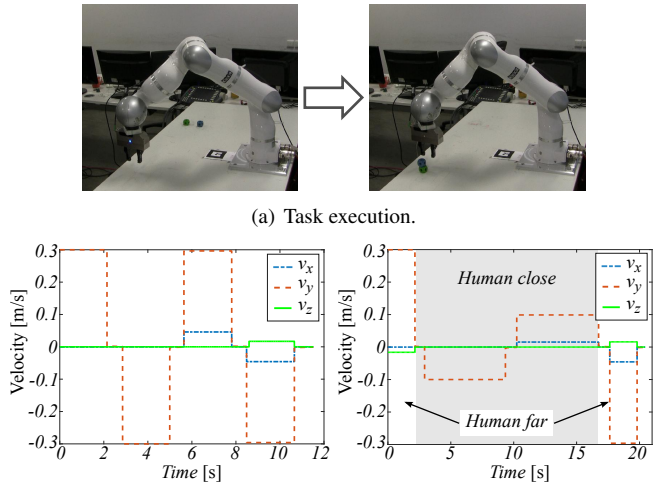
Fig. 6. *f*DED - Execution time and accuracy (mean and standard deviation over 100 iterations). (a) Dependency of the execution time on the step size. (b) Dependency of the execution time on the initial raster tile. (c) Dependency of the accuracy on the step size. (d) Dependency of the accuracy on the initial raster tile.

points, in order not to filter any values out of the real depth image. Moreover, the robot covers the entire virtual depth image. Hence, both depth images have to be entirely filled with a lattice of points and the number of distance computations is maximal. In this case, the brute force algorithm computes $(640 \times 480)^2 \approx 0.9 \times 10^{12}$ distances. Time results for different parameter sets are shown in Tab. 2. An average time (over 100 iterations) of 2.8 ms is obtained with $S_x = S_y = 32$ px.

Table 2. *f*DED - Computation time in the worst case scenario.

T_s [px]	S_x/S_y [px]	Robot Pts (%)	Time (mean/std) [ms]
32	8/8	100	6.3/0.19
32	16/16	100	3.7/0.16
32	32/32	100	**2.8/0.16**
32	32/32	50	**1.9/0.16**
32	32/32	30	**1.3/0.16**

The proposed worst case represents a theoretical upper bound for the computation time, but it is unrealistic in real scenarios. Indeed, having a depth map completely covered by objects is possible in extremely cluttered environments, while assuming that the robot covers all the virtual depth is unrealistic. Camera sensors, in fact, are used to monitor the robot's workspace. If the robot covers all the scene, this means the sensor is not correctly placed (Flacco and De Luca, 2010). More realistic upper bounds on the computation time are given in Tab. 2 by considering that the robot occupies 50% and 30% of the total points. As for the worst case, the robot is located behind the objects, in order not to filter any values out of the depth image.



(a) Task execution.

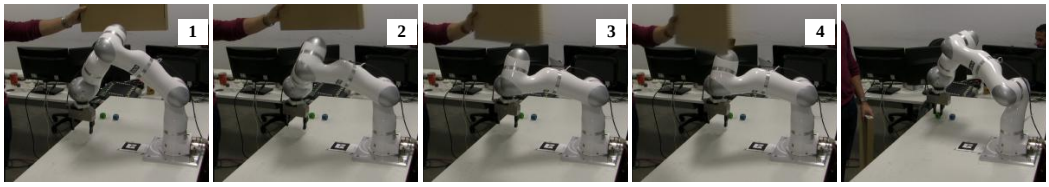
(b) Nominal velocity ($\alpha_h = 0.3$ m/s).

(c) Velocity scaling.

Fig. 7. Execution of the stacking task. (a) Snapshots of the task execution. (b) Robot velocity when the human is not in the workspace. (c) Velocity scaling when the human enters/leaves the workspace.

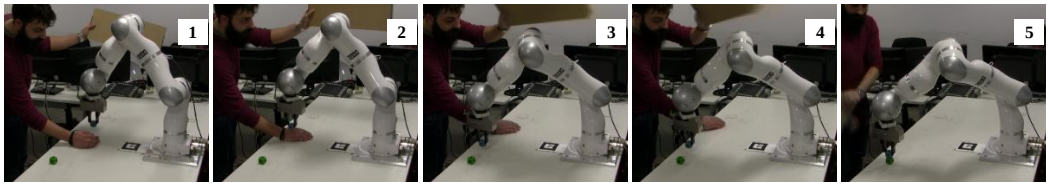
4. Case study

The proposed architecture for online motion replanning is tested in a human-robot interaction scenario. The human is tracked at 30 Hz using a Microsoft Kinect RGB-D camera. The same camera is used to monitor the scene and compute the distance from eventual obstacles. The robot is a 7 dof KUKA LWRIV+ (Bischoff et al., 2010), controlled at 500 Hz. As shown in Fig. 7(a), the robot task consists in stacking two blocks. The blocks are initially placed at $\boldsymbol{g}_1 = [-0.55, 0.35, 0.01]$ and $\boldsymbol{g}_2 = [-0.45, 0.35, 0.01]$ respectively. The stacking positions are $\boldsymbol{g}_3 = [-0.55, -0.3, 0.01]$

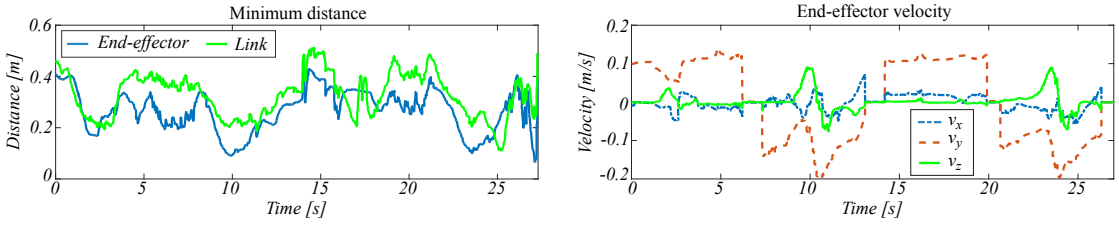


(a) Avoiding collisions between the robot body and a moving obstacle.

(b) Avoiding collisions between the robot end-effector and a moving obstacle (user's hand).



(c) Avoiding multiple collisions with moving obstacles.



(d) Typical minimum distance and velocity profiles during the interaction with the user.

Fig. 8. The robot avoids several collisions with dynamic obstacles while preserving the task execution.

and $\boldsymbol{g}_4 = [-0.55, -0.3, 0.035]$ respectively. The stacking task consists of four point-to-point motions generated by combining four stable DS in the form

$$\dot{\boldsymbol{p}} = \alpha_h \frac{\boldsymbol{g}_i - \boldsymbol{p}}{\|\boldsymbol{g}_i - \boldsymbol{p}\|}, \; i = 1, .., 4$$

where $\alpha_h > 0$ is the desired (constant) speed. During the motion, the robot orientation is kept fixed. In this way, the robot can exploit 1 dof for null-space collision avoidance.

4.1. Velocity scaling and collision avoidance

The user can enter the robot workspace at any time and start the interaction. As shown in Fig. 7(c), when the human is close to the robot, the task is executed at the reduced speed of $\alpha_h = 0.1$ m/s. When the human is far from the robot, the task is executed at the nominal speed of $\alpha_h = 0.3$ m/s (see Fig. 7(b)).

In realistic scenarios the robot needs to avoid possible collisions with dynamic obstacles, preferably without compromising the task execution. Figures 8(a)-(c) show the robot avoiding collisions with several moving objects, while correctly executing its task. The robot is able to prevent multiple collisions with its entire body, using the combination of multi-priority inverse kinematics and DS modulation presented in Sec. 2.4.2.

The left panel in Fig. 8(d) shows the minimum distance between the robot and the closest obstacle. The minimum distance is always bigger than 6 cm, meaning that the robot is effectively able to avoid collisions. The right panel in Fig. 8(d) shows the end-effector velocity during the interaction. As detailed in Sec. 2.4, the velocity is reduced in the direction toward the obstacle, and increased in the collision-free directions. This generates the avoiding motion while minimizing the impact force (almost zero velocity) in case of unexpected collisions.

4.2. Comparison with state-of-the-art approach

In this experiment, we compare the performance of the proposed *f*DED approach (see Sec. 3) and the CPU-based approach for distance evaluation presented in (Flacco et al., 2012) and used in our previous work (Saveriano and Lee, 2014b). The approaches are compared considering the computation time and the minimum distance between obstacles and robot during the task execution. In our C++ implementation of the approach in (Flacco et al., 2012), we approximate the robot body with 10 spheres of radius $r_s = 0.12$ m, and compute the distance between the points in the depth map and each sphere. Moreover, we consider a region of surveillance with $\rho = 0.4$ m, i.e., we skip all the depth points outside a cube of side 2ρ around each sphere. We also skip points whose depth value is too close (< 1.8 m) and to far (> 3.5 m). If no points are found in the neighborhood of the robot, the distance is set to a default value

(0.5 m) which does not generate avoiding movements. As for the *f*DED approach, the robot is removed from the sensor depth using the approach described in Sec. 3.3.1.

Results in Fig. 9(a) show that *f*DED takes, on average, 1.9 ms to compute the distances, while the CPU-based approach requires 9.9 ms. The significantly smaller computation time of *f*DED is beneficial when avoiding collisions with moving obstacles. The robot, in fact, reacts quickly to the approaching obstacle and effectively avoids the collision. This result is illustrated in Fig. 9(b), which shows that the minimum distance between robot and obstacles is always bigger than 6.7 cm. On the contrary, the delay introduced by the CPU-based approach makes the robot less reactive while avoiding possible collisions. As shown in Fig. 9(c), in some cases the robot fails to avoid the collision (distance equal to zero). Presented results allow us to conclude that, in dynamic environments, a fast approach like *f*DED is beneficial to effectively avoid collisions. The drawback of *f*DED is that its implementation requires a dedicated hardware (GPU) for parallel processing.

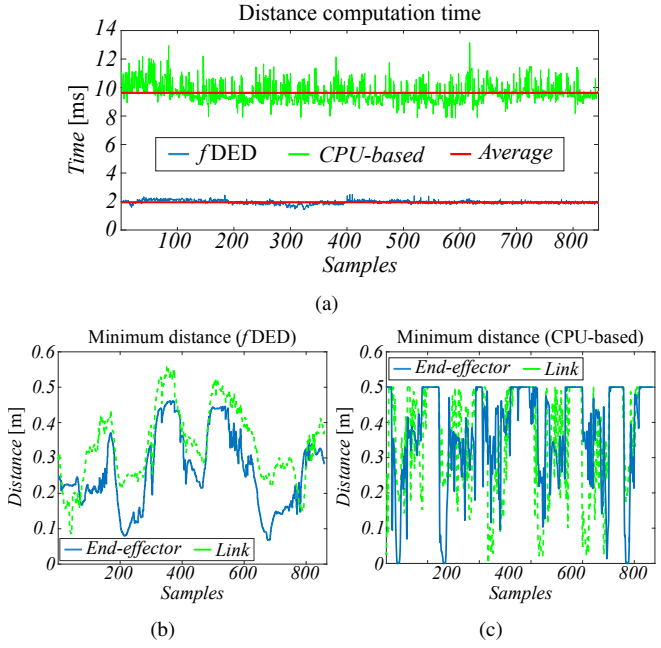
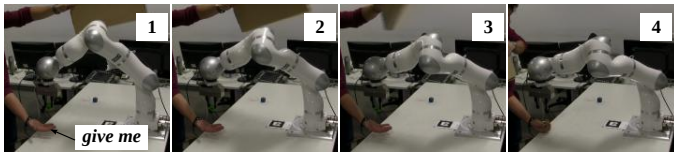


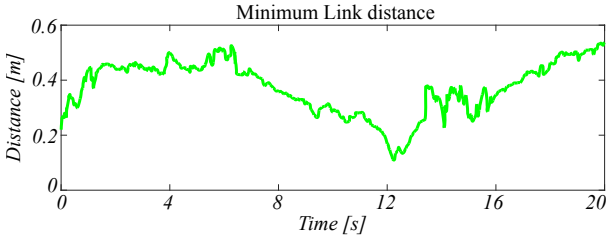
Fig. 9. Comparison between *f*DED and the CPU-based approach.

4.3. Cooperative task execution via human gestures

In this experiment, we exploit the proposed architecture to execute the stacking task in cooperation with the human. To this end, we integrate a gesture recognition module in the *Human Observer* (see Fig. 1) to modify the task execution via gesture commands. Human gestures are recognized using the motion descriptor proposed in (Saveriano and Lee, 2013a). This descriptor has the advantage of being invariant to roto-translations and scaling factors, which makes it well-suited for recognizing motions performed by different subjects or observed from different views. The approach works by transforming the trajectory of the human torso, right shoulder, elbow and hand into a sequence of invariant values. As already mentioned, the human is tracked at 30 Hz. Continuous gestures are recognized using a sliding widow of length 50 frames, and dynamic time warping (Sakoe and Chiba, 1978) for classification. We considered 3 gestures in this experiment: *1) come* to start the task execution, *2) stop* to suspend the execution, and *3) handover right* to command the robot to give the item to the user and continue the execution (take the next item). The 3 gestures are a subset of the 6 gestures used in (Saveriano and Lee, 2013a) for continuous gesture recognition. The recognition rate on the full set of 6 gestures is 91.67%.



(a) Snapshots of the handover task execution.



(b) Minimum robot-obstacle distance during the *handover* task.

Fig. 10. The robot gives the picked item to the user and avoids collisions with a dynamic obstacle.

In order to execute human commands, dynamical system are exploited to replan the motion in real time. In particular, if the robot receives a *stop* command, the desired speed is set to $\alpha_h = 0$ m/s, and the robot waits for other commands. If the robot receives a *come* command, the speed is set to 0.1 m/s (close user) or 0.3 m/s (far user), and the robot can continue the task execution. If the robot receives a *handover right* command, the goal is changed to reach the current hand position (plus an offset of 5 cm along the vertical direction) and leave the item. During the *handover* action, the end-effector avoidance is disabled since the robot has to reach the hand. The null-space avoidance, instead, is active to make the robot able to avoid collisions with its body. As shown in Figure 10, the robot successfully executes the *handover* task and, at the same time, avoids the collision with a moving obstacle.

4.4. Discussion

Presented results show the effectiveness of our approach in a realistic, dynamic scenario. Dynamical systems properties and real-time distance computation are effectively exploited to increase the human safety by reducing the robot's velocity in case of close interaction with the human and avoiding possible collisions. In order to make a step towards the interpretation of human intentions, we integrated a gesture based command interface in the high level of the architecture. Also in this case, dynamical systems are exploited to replan the current task online and generate the motor commands. Compared to geometric planners described in Sec. I, our approach generates suboptimal paths. Geometric planners consider human, robot

kinematics and objects in the scene, in order to search for the optimal (minimum-cost) path. However, these approaches are computationally expensive, thus the applicability of geometric planners is limited in dynamic scenarios. On the contrary, the path generated with our approach is not necessarily optimal in terms of traveled distance or human safety; but the robot trajectory is computed and replanned in significantly less time, which allows a more fluent human-robot interaction.

5. Conclusion and future work

In this work, we presented two-level hierarchical architecture for real-time, human-aware motion replanning. The lower level generates the motor commands for the robot and it has to match strict real-time requirements, being typical robot control frequencies between 0.1 and 1 kHz. Moreover, the lower level takes into account the human safety and the correct robotic task execution during the human-robot interaction. We leveraged stable dynamical systems to generate collision-free paths, without affecting the correct task execution. In case of close interaction with the human, the velocity of the robot is automatically reduced to a safe value. To match requirements of real-time performance in dynamic environments, we have proposed a novel approach capable to compute robot-obstacle distances in real-time. The proposed fDED algorithm performs all the computations directly in the depth space and it is highly parallelizable, which makes its implementation possible on the graphic processing unit. The higher level of the architecture is responsible to monitor the user status and to provide the needed information to the lower level for task replanning. The proposed approach has been tested in a human-robot interaction scenario and compared with a state-of-the-art approach. Obtained results have shown that the robot is effectively able to replan his motion online to execute human commands and avoid collisions. Moreover, the comparison has shown that the proposed framework is more effective in avoiding collisions with moving obstacles.

In its current version, the higher level of the architecture mainly considers the distance of the human from the robot to scale down the velocity and transforms human gestures into robot commands. Our future research will focus on extending the higher level to have a comprehensive description of the human task and to predict his/her future actions.

Acknowledgments

This work has been supported by the Technical Univ. of Munich, Int. Graduate School of Science and Engineering.

References

An, S., Lee, D., 2015. Prioritized inverse kinematics with multiple task definitions, in: Int. Conference on Robotics and Automation, pp. 1423–1430.

Bascetta, L., Magnani, G., Rocco, P., Migliorini, R., Pelagatti, M., 2010. Anti-collision systems for robotic applications based on laser time-of-flight sensors, in: Int. Conf. on Advanced Intelligent Mechatronics, pp. 278–284.

Bischoff, R., Kurth, J., Schreiber, G., Koeppe, R., Albu-Schäffer, A., Beyer, A., Eiberger, O., Haddadin, S., Stemmer, A., Grunwald, G., Hirzinger, G., 2010. The kuka-dlr lightweight robot arm - a new reference platform for robotics research and manufacturing, in: Int. Symposium on Robotics, pp. 1–8.

Cirillo, M., Karlsson, L., Saffiotti, A., 2010. Human-aware task planning: An application to mobile robots. Trans. on Intelligent Sys. and Tech. 1.

De Luca, A., Flacco, F., 2012. Integrated control for phri: Collision avoidance, detection, reaction and collaboration, in: International Conference on Biomedical Robotics and Biomechatronics, pp. 288–295.

Falco, P., Natale, C., 2014. Low-level flexible planning for mobile manipulators: a distributed perception approach. Advanced Robotics 28, 1431–1444.

Flacco, F., De Luca, A., 2010. Multiple depth/presence sensors: Integration and optimal placement for human/robot coexistence, in: International conference on Robotics and Automation, pp. 3916–3923.

Flacco, F., Kroeger, T., De Luca, A., Khatib, O., 2014. A depth space approach for evaluating distance to objects. Journal of Int. and Rob. Sys. 80, 1–16.

Flacco, F., Kroger, T., De Luca, A., Khatib, O., 2012. A depth space approach to human-robot collision avoidance, in: Int C. on Rob and Aut, pp. 338–345.

Haddadin, S., Albu-Schäffer, A., De Luca, A., Hirzinger, G., 2008. Collision detection and reaction: A contribution to safe physical human-robot interaction, in: Int.Conference on Intelligent Robots and Systems, pp. 3356–3363.

Haddadin, S., Haddadin, S., Khoury, A., Rokahr, T., Parusel, S., Burgkart, R., Bicchi, A., Albu-Schäffer, A., 2012. On making robots understand safety: Embedding injury knowledge into control. Int J of Rob Res 31, 1578–1602.

Ijspeert, A., Nakanishi, J., Pastor, P., Hoffmann, H., Schaal, S., 2013. Dynamical Movement Primitives: learning attractor models for motor behaviors. Neural Computation 25, 328–373.

Ikuta, K., Ishii, H., Nokatao, M., 2003. Safety evaluation method of design and control for human-care robot. Int. J. of Rob. Res. 22, 281–297.

Khansari-Zadeh, S.M., Billard, A., 2012. A dynamical system approach to realtime obstacle avoidance. Autonomous Robots 32, 433–454.

Khoshelham, K., Oude Elberink, S., 2012. Accuracy and resolution of kinect depth data for indoor mapping applications. Sensors 12, 1437–1454.

Kulić, D., Croft, E.A., 2006. Real-time safety for human–robot interaction. Robotics and Autonomous Systems 54, 1–12.

Lacevic, B., Rocco, P., Zanchettin, A.M., 2013. Safety assessment and control of robotic manipulators using danger field. Trans. on Rob. 29, 1257–1270.

Lasota, P.A., Shah, J.A., 2015. Analyzing the effects of human-aware motion planning on close-proximity human–robot collaboration. Human Factors: The Journal of the Human Factors and Ergonomics Society 57, 21–33.

Ma, Y., Soatto, S., Kosecka, J., Sastry, S.S., 2003. An Invitation to 3-D Vision: From Images to Geometric Models. Springer-Verlag.

Marton, Z.C., Rusu, R.B., Beetz, M., 2009. On fast surface reconstruction methods for large and noisy datasets, in: Proc. of the IEEE International Conference on Robotics and Automation, pp. 3218–3223.

Neumann, K., Steil, J., 2015. Learning robot motions with stable dynamical systems under diffeomorphic transformations. Rob. and Aut. Sys. 70, 1–15.

Pan, J., Sucan, I., Chitta, S., Manocha, D., 2013. Real-time collision detection and distance computation on point cloud sensor data, in: International Conference on Robotics and Automation, pp. 3593–3599.

Sakoe, H., Chiba, S., 1978. Dynamic programming algorithm optimization for spoken word recognition. Trans. on Ac., Sp., and Sig. Proc 26, 43–49.

Saveriano, M., Lee, D., 2013a. Invariant representation for user independent motion recognition, in: Int. Symposium on Robot and Human Interactive Communication, pp. 650–655.

Saveriano, M., Lee, D., 2013b. Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles, in: International Conference on Intelligent Robots and Systems, pp. 5380–5387.

Saveriano, M., Lee, D., 2014a. Distance based dynamical system modulation for reactive avoidance of moving obstacles, in: Int. Conf. on Rob. and Aut., pp. 5618–5623.

Saveriano, M., Lee, D., 2014b. Safe motion generation and online reshaping using dynamical systems, in: Int. Conf. on Ubiq. Rob. and Amb. Intel.

Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G., 2009. Robotics - Modelling, Planning and Control. Springer.

Sisbot, E.A., Alami, R., 2012. A human-aware manipulation planner. Transactions on Robotics 28, 1045–1057.

Sisbot, E.A., Marin-Urias, L.F., Alami, R., Simeon, T., 2007. A human aware mobile robot motion planner. Transactions on Robotics 23, 874–883.

Slotine, J., Li, W., 1991. Applied nonlinear control. Prentice-Hall.