2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Human-Robot Coexistence and Contact Handling with Redundant Robots

Emanuele Magrini Alessandro De Luca

Abstract— **We present further computational tools and control results in the framework of human-robot coexistence and collaboration. A GPU parallel processing algorithm is introduced for real-time monitoring of dynamic distances between a robot and generic obstacles moving in its environment, taking advantage of the handling of RGB-D data directly in the depth space of the sensor. Combined with the use of model-based residual signals, this approach allows efficient detection of contact points on the robot with simultaneous estimation of the exchanged contact forces. When the robot is kinematically redundant for the original task and undergoes a physical contact, a control scheme accommodates collaboration trying to preserve task execution, or reacts by abandoning the task if the estimated contact forces exceed some safety threshold. Experimental results are reported for a KUKA LWR.**

I. Introduction

Taking advantage of recent exciting research results, the domain of physical Human-Robot Interaction (pHRI) is booming. Friendly lightweight robots with the capability of safely sharing their workspace with humans are appearing in service and industrial environment. However, the realization of truly collaborative applications, merging the adaptive skills of humans with the high precision of robots, remains still a big challenge. Within the European research project SAPHARI [1], we have proposed a control architecture devoted to pHRI, based on a hierarchy of consistent robot behaviors organized in three layers addressing, respectively, human-robot safety, coexistence, and collaboration [2]. Safety issues in physical interaction being of primary concern, we implemented strategies for collision detection, isolation, and reflex reaction based on the residuals [3]–[5] in the *safety* layer. To limit as much as possible undesired collisions, the *coexistence* layer includes algorithms for monitoring the shared workspace with external sensors (cameras, RGB-D, laser) and for preventing and avoiding collisions in real time [6]–[8]. A real-time collision avoidance algorithm based on 3D Point Clouds has been presented in [9]. The data are processed taking advantage the Graphics Processing Unit (GPU) and the Compute Unified Device Architecture (CUDA) implemented on NVIDIA graphics cards.

The upper layer is devoted to physical *collaboration* tasks, in which a continuous and intentional contact may (or needs to) take place with a controlled exchange of forces/torques. In many applications, contacts are assumed to happen only at the robot end-effector, usually equipped with a 6D force/torque sensor to measure the interaction. In order to extend the robot collaborative capability, contacts on the entire robot body should also be considered. This raises the additional issue of reconstructing the exchanged forces at generic contact points along the robot structure, either by measuring them, e.g., by placing patches of conformable tactile skin in different locations [10], or by estimating them in an indirect way, e.g., combining model-based methods with a force/torque sensor mounted at the robot base [11] or with machine learning techniques [12]. In [13], a first example of a method that estimates contact forces occurring at a generic point (a priori unknown) of the robot surface in contact with the human hand(s) was given. This was obtained by a *virtual* force sensor, which combines the proprioceptive information of the residual signals with the localization of the human hands provided by Microsoft Kinect. By removing the assumption of contacts limited to human hands (thus the dependency on Kinect built-in skeleton tracking algorithm), we present a novel approach for localizing in real-time contact points between a robot and generic obstacles moving in its workspace. The algorithm is based on distance computation in the depth space [7], taking advantage of the CUDA framework for mass parallel GPU programming.

In our recent works, the estimated contact force was used in a series of human-robot interaction experiments based on admittance, impedance, direct force, or hybrid force/velocity control schemes in order to assign a desired robot behavior at or around the current *contact point*, where the exchange of forces is taking place [13]–[15]. In this paper, we add another element to the portfolio of control laws developed for physical human-robot collaboration. Following the ideas presented in [16], we propose a control scheme that uses robot redundancy for preserving a Cartesian task despite the possible occurrence of (un)desired contacts. By exploiting redundancy, the robot can preserve the execution of an end-effector motion task, while still reacting to a detected contact so as to keep the contact forces below a defined threshold. As soon as the contact force exceeds the threshold, the robot should abandon the task and realize an admittance control scheme at the contact point. Note that, thanks to the *collaboration* layer, contacts with robot are allowed and can be dynamically controlled using an estimate/measure of the contact force. On the contrary, at the *safety* layer, contacts are not allowed and only evasive reactions are provided.

The paper is organized as follows. Section II recalls the contact force detection and estimation method, including the generalized admittance control [13]. Section III details how

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Via Ariosto 25, 00185 Roma, Italy ({magrini,deluca}@diag.uniroma1.it). This work is supported by the European Commission, within the H2020 projects FoF-637080 SYMPLEXITY (www.symplexity.eu) and ICT-645097 COMANOID (www.comanoid.eu).

the proposed contact point estimation algorithm is able to localize contacts between obstacles and robot, combining the information provided by a depth sensor and the use of model-based residual signals. The control scheme for reacting to the estimated contact forces while preserving the robot task is presented in Sec. IV. Section V reports on experimental results obtained with the proposed method, using a 7-dof KUKA LWR robot and a Kinect sensor. Conclusions are drawn in Sec. VI.

II. Preliminaries

We consider robot manipulators as open kinematic chains of $n+1$ rigid links, connected by n joints and with associated generalized coordinates $\boldsymbol{q} \in \mathbb{R}^n$. Let $\boldsymbol{F}_c \in \mathbb{R}^3$ be an interaction force acting on robot surface in $\boldsymbol{x}_c$. Its dynamic model is given by

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} + \boldsymbol{\tau}_c, \qquad (1)$$

where $\boldsymbol{M}(\boldsymbol{q}) > 0$ is the robot inertia matrix, the Coriolis and centrifugal terms are factorized using the matrix $\boldsymbol{C}(\boldsymbol{q},\dot{\boldsymbol{q}})$ of Christoffel symbols, $\boldsymbol{g}(\boldsymbol{q})$ includes gravitational terms, $\boldsymbol{\tau} \in \mathbb{R}^n$ is the control torque, and $\boldsymbol{\tau}_c = \boldsymbol{J}_c^T(\boldsymbol{q})\boldsymbol{F}_c$ is the joint torque associated to the interaction force $\boldsymbol{F}_c$. Using external sensing, the location of the contact point $\boldsymbol{x}_c$ can be determined, and the associated $3 \times n$ contact Jacobian $\boldsymbol{J}_c$ can be computed.

A. Collision detection and contact force estimation

Following [3], for a robot with dynamics (1), the residual vector $\boldsymbol{r} \in \mathbb{R}^n$ is defined as

$$\boldsymbol{r}(t) = \boldsymbol{K}_I \left(\boldsymbol{p} - \int_0^t \left(\boldsymbol{\tau} + \boldsymbol{C}^T(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} - \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{r} \right) ds \right), \qquad (2)$$

where $\boldsymbol{p} = \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}}$ is the generalized momentum of the robot and $\boldsymbol{K}_I > 0$ is a diagonal gain matrix. The dynamic evolution of $\boldsymbol{r}$ is that of a stable, first-order low-pass filter, $\dot{\boldsymbol{r}} = \boldsymbol{K}_I(\boldsymbol{\tau}_c - \boldsymbol{r})$. Therefore, for sufficiently large gains, we can assume that

$$\boldsymbol{r} \simeq \boldsymbol{\tau}_c = \boldsymbol{J}_c^T(\boldsymbol{q})\boldsymbol{F}_c. \qquad (3)$$

Equation (3) shows that $\boldsymbol{r}$ is observing the joint torque resulting from a contact or a collision. In fact, during free motion, $\boldsymbol{r} = \boldsymbol{0}$ up to unmodeled disturbances and measurement noise. To avoid false detection due to spurious spikes in noisy signals, collisions are detected if there exists at least an index k, with $k \in \{1, \dots, n\}$, for which $|r_k| > r_{low,k} > 0$, namely when

$$\mathrm{CD} = \max\left\{ \frac{|r_1|}{r_{low,1}}, \cdots, \frac{|r_n|}{r_{low,n}} \right\} > 1. \qquad (4)$$

The linear system (3) forms the basis for the estimation of the unknown contact force $\boldsymbol{F}_c$. Depending on which link of the serial kinematic chain is involved in the contact, this may have a square, under-, or over-determined coefficient matrix $\boldsymbol{J}_c^T$. In any case, by pseudoinverting (3) the contact force is estimated as

$$\widehat{\boldsymbol{F}}_c = \left(\boldsymbol{J}_c^T(\boldsymbol{q}) \right)^{\#} \boldsymbol{r}. \qquad (5)$$

Indeed, the estimate $\widehat{\boldsymbol{F}}_c$ will be limited only to those components of $\boldsymbol{F}_c$ that can be detected by the residual $\boldsymbol{r}$, namely those contact forces that do not belong to the null space $\mathcal{N}(\boldsymbol{J}_c^T(\boldsymbol{q}))$. For further details and for the analysis of cases when the contact Jacobian is not full rank, see [13].

B. Admittance control scheme at contact point

In [13], we have used the estimated contact forces within an admittance control in presence of one or two contacts. For simplicity, we consider from now on only single contact situations. Let $\boldsymbol{q}_{c,d} = \boldsymbol{q}(t_c) \in \mathbb{R}^7$ an d $\boldsymbol{x}_{c,d} = \boldsymbol{x}_c(t_c) \in \mathbb{R}^3$ be respectively the robot configuration and the position of the contact point when the interaction with the environment begins, namely at $t = t_c$. In admittance control, the desired velocity $\dot{\boldsymbol{x}}_c$ of the contact point is assigned to be proportional to the (real and/or virtual) force $\boldsymbol{F}_a$ acting on that point, or

$$\dot{\boldsymbol{x}}_c = \boldsymbol{K}_a \boldsymbol{F}_a, \qquad \boldsymbol{F}_a = \widehat{\boldsymbol{F}}_c + \boldsymbol{K}_e(\boldsymbol{x}_{c,d} - \boldsymbol{x}_c), \qquad (6)$$

where $\boldsymbol{K}_a > 0$ is the admittance gain matrix, $\boldsymbol{K}_e > 0$ is the stiffness gain matrix, and $\boldsymbol{F}_a$ is the total active force on the contact point $\boldsymbol{x}_c$. In this way, an equilibrium may arise ($\boldsymbol{F}_a = \boldsymbol{0}$) between an obstacle pushing continuously on the robot and the virtual spring pulling the contact point back to its initial position $\boldsymbol{x}_{c,d}$.

Since we are only considering positional motion tasks, in a dual fashion the robot will be redundant for contact force tasks that occur on link $i \geq 4$, so that an extra null-space motion contribution can be considered. Thus, the joint velocity command to the robot will be defined in general as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}_c^{\#}\dot{\boldsymbol{x}}_c + \boldsymbol{P}_c\dot{\boldsymbol{q}}_n \qquad (7)$$

with the projector $\boldsymbol{P}_c = \boldsymbol{I} - \boldsymbol{J}_c^{\#}\boldsymbol{J}_c$ in the null-space of the contact Jacobian $\boldsymbol{J}_c$. The additional joint velocity in (7) is specified as $\dot{\boldsymbol{q}}_n = \boldsymbol{K}_n(\boldsymbol{q}_{c,d} - \boldsymbol{q})$, with $\boldsymbol{K}_n > 0$, and helps driving the robot back to its initial configuration $\boldsymbol{q}_{c,d}$.

III. Contact Point Estimation Algorithm

In order to estimate the force (5) applied by the environment to the robot, we need to localize the point on the robot surface where a contact happens. In [13], Cartesian distances between all vertices of geometric surfaces of a robot CAD model and the human hand(s) were computed. As soon as a contact was detected, the point at minimum distance was taken as the contact point. The method is effective but has some limitations: *i)* it relies on the Kinect skeleton-tracking algorithm which is quite noisy, especially if more than one human appears in the scene; *ii)* it is only able to localize contacts with human hands and joints; *iii)* it is not optimized from a computational point of view, since vertices that are not visible by the camera should not be included in the distance computation.

To overcome these limitations, we propose a contact point estimation algorithm based on the visual feedback provided by a RGB-D sensor. In this kind of sensors, each image pixel contains also the information about the distance from the camera image plane to the 3D point in the Cartesian space associated to the pixel along a given ray. Moreover, such sensors are able to localize contacts with any kind of obstacle (possibly, a human), taking into account only those parts of the robot which are visible to the camera, maximizing thus the computational performance.

A. Overview

The proposed contact localization algorithm has been implemented as GPU program and it is able to exploit the parallelism of a graphic board. In particular, it is based on the OpenGL library that provides hardware accelerated rendering functions, and on the CUDA framework for mass parallel programming within the NVIDIA architecture. With respect to a common CPU, in a graphics processing unit each core has the capability to execute at the same time hundreds of processes. This high degree of parallelism gives to any GPU-based application huge performance improvements, thanks also to a high-speed memory closely interconnected to the GPU's cores. The CUDA architecture provides to developers the access facilities to the GPU resource, with the possibility of writing programs similarly to the CPU.

Our method consists in processing three 2.5D images, having the same resolution:

- *Real depth image* is an image of the environment as captured by the depth sensor, see Fig. 1(a).
- *Virtual depth image* is an image containing only a projection of the robot in a virtual environment. The image is created using OpenGL to load a CAD model of the robot. Once the CAD has been loaded, by using direct kinematics, we move the virtual model to match the real robot configuration, see Fig. 1(b).
- *Filtered depth image* is an image of the environment containing only the obstacles. It is obtained subtracting to the *real depth image* the *virtual depth image* of the robot. It is a common practice to load a slightly expanded CAD model of the robot in the virtual depth image, as in Fig. 1(c).

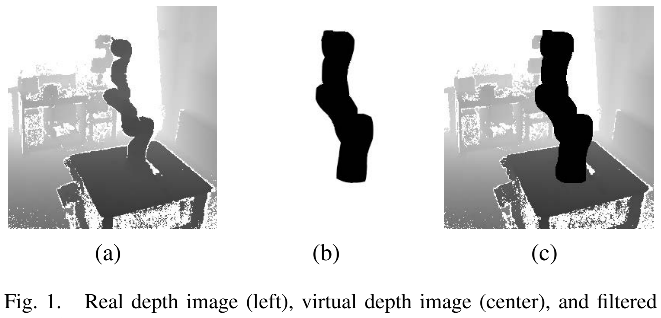

Fig. 1. Real depth image (left), virtual depth image (center), and filtered depth image (right).

B. Robot filtering by CUDA

The robot filtering process is needed to remove the robot from the depth sensor image so as to obtain a filtered image in which there are only obstacles. The entire processing scheme is shown in Figure 2.

The depth sensor provides a new frame (here, at 30 Hz) of the environment and loads the data into the GPU memory. In the meantime, a CAD model of the robot has been loaded in a virtual environment, combining the information of the direct kinematics and the capabilities of OpenGL library. At this stage, a sequence of matrix transformation have been applied in order to obtain a virtual environment point of view that is the same of the depth sensor point of view. In this way, the virtual robot will overlap to the real one.

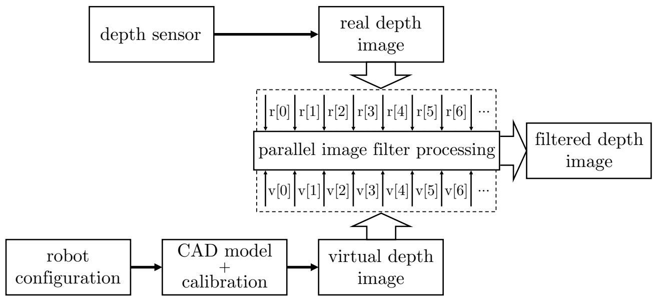

Fig. 2. Robot filtering processing scheme.

The first transformation $\boldsymbol{T}_{world}$ maps the coordinates of a point $\boldsymbol{p}_{CAD} = \begin{pmatrix} p_x & p_y & p_z \end{pmatrix}^T$ from a local reference frame (defined in the CAD model) to a world reference frame (often placed at the robot base). Next, a calibration matrix $\boldsymbol{T}_{camera}$ between the world and the camera sensor provides a second transformation to express the world coordinates in the camera frame. A perspective transformation matrix $\boldsymbol{T}_{clip}$ projects then these coordinates into clip-space coordinates. In particular, this transformation determines whether an object is too close to the camera or too far away to be handled. The last transformation $\boldsymbol{T}_{depth}$ determines the depth space coordinates of the point.

Summarizing, the coordinates of a point in the virtual depth image can be determined applying the above transformations to the 3D CAD model as

$$\boldsymbol{p}_v = \begin{pmatrix} p_{v,x} \\ p_{v,y} \\ d_v \end{pmatrix} = \boldsymbol{T}_{depth} \cdot \boldsymbol{T}_{clip} \cdot \boldsymbol{T}_{camera} \cdot \boldsymbol{T}_{world} \cdot \boldsymbol{p}_{CAD}, \tag{8}$$

where $p_{v,x}$ and $p_{v,y}$ are the pixel coordinates in the image plane, d_v is the corresponding depth.

Once the two 2.5D images are ready, they are loaded into the GPU memory as two row vectors with depth information. Components in two vectors having the same index correspond to the same pixels in depth images. A parallel comparison of the depth information for any pair of corresponding components is then performed to filter out the robot. In particular, if a pixel belonging to the robot has a shorter depth than its corresponding pixel in the real depth image, then a maximum depth value is assigned to the corresponding pixel in the filtered depth image. Thus, for each pair of pixel coordinates (x, y) we have

$$d_f(x,y) = \begin{cases} d_r(x,y) & \text{if } d_r(x,y) < d_v(x,y) \\ \text{max depth} & \text{if } d_r(x,y) \geq d_v(x,y), \end{cases}$$

where $d_f(x,y)$, $d_r(x,y)$, and $d_v(x,y)$ are the depth values in pixel coordinates (x, y) of the filtered, real, and virtual depth image, respectively. The resulting filtered image is shown in Fig. 1(c).

C. Contact localization in depth space

In our previous work, we used to localize the contact point by computing the minimum Cartesian distances between the robot CAD vertices and the human hand(s). We propose here a revised depth space approach for distance computation between obstacles and the robot. In [7], distances were computed between an obstacle point $\boldsymbol{O}$ and a set of 'control' points $\boldsymbol{P}_i$ distributed along the robot kinematic chain. Relying on this method and exploiting the parallel computation capabilities of CUDA architecture, we can now compute distances between *all* robot points $\boldsymbol{P}_D = \begin{pmatrix} p_{v,x} & p_{v,y} & d_v \end{pmatrix}^T$ projected in the virtual depth image and *all* obstacle points $\boldsymbol{O}_D = \begin{pmatrix} p_{f,x} & p_{f,y} & d_f \end{pmatrix}^T$ in the filtered depth image belonging to a *region of surveillance* centered in $\boldsymbol{P}$. Recalling the formulas in [7], we have

$$d(\boldsymbol{O}, \boldsymbol{P}) = \sqrt{v_x^2 + v_y^2 + v_z^2},$$

with

$$\begin{aligned} v_x &= \frac{(p_{f,x} - c_x)d_f - (p_{v,x} - c_x)d_v}{f\, s_x} \\ v_y &= \frac{(p_{f,y} - c_y)d_f - (p_{v,y} - c_y)d_v}{f\, s_y} \\ v_z &= d_f - d_v \end{aligned}$$

where $(p_{f,x}, p_{f,y})$ and $(p_{v,x}, p_{v,y})$ are the coordinates in the depth space of the points $\boldsymbol{O}$ and $\boldsymbol{P}$ respectively, d_f and d_v are their depth w.r.t. the camera, c_x and c_y are the pixel coordinates of the center of the image plane (on the focal axis), f is the focal length of the camera and s_x and s_y are the dimensions of a pixel in meters. The last five parameters constitute the intrinsic parameters of the camera and can be usually retrieved by the device manufacturer. Since we do not know how long an obstacle is, if the obstacle point has a depth smaller than the point of the robot ($d_f < d_v$) we assume the depth of the obstacle to be $d_f = d_v$.

The Cartesian surveillance region, constituted by a cube of side 2ρ centered in $\boldsymbol{P}$, will have dimensions in the image plane given by

$$x_s = \rho\,\frac{f s_x}{d_v - \rho}, \quad y_s = \rho\,\frac{f s_y}{d_v - \rho}.$$

Therefore, all pixel in the filtered depth image within the region $\boldsymbol{S} = \left[p_{f,x} - \frac{x_s}{2}, p_{f,x} + \frac{x_s}{2}\right] \times \left[p_{f,y} - \frac{y_s}{2}, p_{f,y} + \frac{y_s}{2}\right]$ will be considered for distance computation.

As soon as a contact is detected by the residual, the point of the visible robot surface which is at minimum distance from the obstacles is considered as contact point. Let $\boldsymbol{p}_c = \begin{pmatrix} p_{c,x} & p_{c,y} & d_c \end{pmatrix}^T$ be the contact point expressed in the 2.5D image plane. The corresponding contact point $\boldsymbol{x}_c$ in the robot frame is computed as

$$\boldsymbol{x}_c = (\boldsymbol{T}_{depth} \cdot \boldsymbol{T}_{clip} \cdot \boldsymbol{T}_{camera})^{-1} \boldsymbol{p}_c. \tag{9}$$

Note that we want the contact point expressed in the world frame, not in the model frame. Thus, the model transformation matrix $\boldsymbol{T}_{world}$ does not appear in (9). The contact point localization processing is illustrated in Fig. 3.

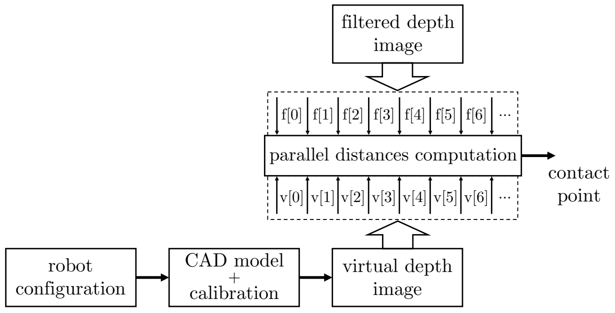
filtered depth image
f[0] f[1] f[2] f[3] f[4] f[5] f[6] ···
parallel distances computation
contact point
v[0] v[1] v[2] v[3] v[4] v[5] v[6] ···
robot configuration
CAD model + calibration
virtual depth image

Fig. 3. Contact point localization processing scheme.

D. Contact Jacobian computation

The presented algorithm is able also to identify which robot link is in contact with the environment. Consider a generic contact point on the surface of the i-link, whose absolute position is given by $\boldsymbol{x}_{c_i}$. Its Cartesian velocity is given by

$$\begin{aligned} \dot{\boldsymbol{x}}_{c_i} &= \boldsymbol{v}_i + \boldsymbol{\omega}_i \times \boldsymbol{x}_{i,c_i} = \boldsymbol{v}_i - \boldsymbol{x}_{i,c_i} \times \boldsymbol{\omega}_i \\ &= \begin{pmatrix} \boldsymbol{I} & -\boldsymbol{S}(\boldsymbol{x}_{i,c_i}) \end{pmatrix} \begin{pmatrix} \boldsymbol{v}_i \\ \boldsymbol{\omega}_i \end{pmatrix} = \begin{pmatrix} \boldsymbol{I} & -\boldsymbol{S}(\boldsymbol{x}_{i,c_i}) \end{pmatrix} \boldsymbol{J}_i \dot{\boldsymbol{q}}, \end{aligned} \tag{10}$$

where $\boldsymbol{v}_i \in \mathbb{R}^3$ and $\boldsymbol{\omega}_i \in \mathbb{R}^3$ are, respectively, the linear and the angular velocity of the frame i, matrix $\boldsymbol{J}_i$ is the $6 \times n$ geometric Jacobian associated to link i (having the last $n - i$ columns identically zero), $\boldsymbol{S}(.)$ represents the skew-symmetric matrix for vector $(\times)$ product, and $\boldsymbol{x}_{i,c_i}$ is the position of the contact point with respect to the origin of frame i. This is computed as $\boldsymbol{x}_{i,c_i} = \boldsymbol{x}_{c_i} - \boldsymbol{x}_i$, being $\boldsymbol{x}_i$ the position of the origin of frame i. From (10), the $3 \times n$ contact Jacobian matrix is finally computed as

$$\boldsymbol{J}_{c_i} = (\boldsymbol{I} \quad -\boldsymbol{S}(x_{i,c_i}))\, \boldsymbol{J}_i,$$

remembering that $\dot{\boldsymbol{x}}_{c_i} = \boldsymbol{J}_{c_i} \dot{\boldsymbol{q}}$. From now on, we will drop for compactness the index i of the link under consideration.

Remark 1. In this paper, we handle only one contact point. However, the algorithm discussed here can be easily extended to the case of multiple contact points. In fact, when (4) is verified, instead of considering only the robot point at minimum distance, we can classify as contact points all those points which have a distance $d(\boldsymbol{O}, \boldsymbol{P}) < d_{th}$, for a given threshold $d_{th} > 0$. Thanks to the parallel computing offered by the CUDA framework, the time needed to localize one or more the one contact point is exactly the same.

Remark 2. In some cases, obstacles get in surface contact with the robot and not only at a single point. In such situations, even if we could localize multiple contact points, these would often be so close to each other (e.g., a human hand interacting with the robot) that it is more convenient to assume still a point-wise contact. In [13], we verified that the error introduced by this assumption is negligible, at least for our purposes.

Remark 3. Discarding the occluded points of the robot leads to benefits in terms of computational time. On the other hand, we are not able to localize contacts (and thus obstacles) behind the robot. This is the main limitation of using just one depth sensor. Occluded areas can be reduced introducing more than one depth sensor in the framework, as in [8].

IV. Motion Control

In this section, we discuss how the estimated contact point between (human) obstacle and robot, and thus the estimated contact force, can be used in a simple way to modify the robot behavior.

The main task for a robot is typically to control the motion of its end effector. Without loss of generality, we consider a 3D-positional task, i.e., with $m = 3$, and assume that the robot is commanded at the joint velocity level. Kinematic control with exponential error recovery is obtained by choosing the joint velocity commands as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\dot{\boldsymbol{x}}_d + \boldsymbol{K}_p(\boldsymbol{x}_d - \boldsymbol{x})) + \boldsymbol{P}\dot{\boldsymbol{q}}_0, \tag{11}$$

where $\boldsymbol{x}_d$ and $\dot{\boldsymbol{x}}_d$ are, respectively, the desired end-effector position and velocity, $\boldsymbol{K}_p > 0$ is a diagonal gain matrix, and $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{J}^{\#}\boldsymbol{J}$ is the orthogonal projection matrix in the null space of $\boldsymbol{J}$. Due to redundancy, the $(n-3)$-dimensional auxiliary velocity vector $\dot{\boldsymbol{q}}_0$ can be arbitrarily specified without affecting the main task.

A. *Combined contact reaction and task preservation strategy*

Following the paradigms of robot redundancy exploitation in [16], we will define a law for the choice of $\dot{\boldsymbol{q}}_0$ so that the robot is able to react safely to external forces while continuing to execute as much as possible the original task.

As long as there is no contact (or no contact is detected), the null-space velocity vector is chosen as $\dot{\boldsymbol{q}}_0 = \boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q})$, with $\boldsymbol{K}_0 > 0$, in order to limit self-motions of the redundant arm and keep the robot as much as possible close to a desired configuration $\boldsymbol{q}_d$. This leads to the commanded joint velocity

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}\dot{\boldsymbol{x}} + \boldsymbol{P}\boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q}).$$

Consider next the occurrence of a contact between an obstacle and the robot. The momentum-based observer (2) will output a residual vector $\boldsymbol{r}$ and, as soon as (4) is verified, the contact force estimation (5) will generate an estimate of the contact force $\widehat{\boldsymbol{F}}_c = \left(\boldsymbol{J}_c^T\right)^{\#}\boldsymbol{r}$. This contact force estimate will be included in the null-space term $\dot{\boldsymbol{q}}_0$. However, to filter out contact forces erroneously produced by uncertainties in the dynamic model (and thus, in the residual computation) a threshold $F_{relax} > 0$ has been introduced on the norm of estimated contact force $\|\widehat{\boldsymbol{F}}_c\|$. The null-space velocity will then be defined as

$$\dot{\boldsymbol{q}}_0 = \begin{cases} \boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q}), & \text{if } \|\widehat{\boldsymbol{F}}_c\| \le F_{relax}, \\ \boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{J}_c^{\#}\boldsymbol{K}_f\widehat{\boldsymbol{F}}_c, & \text{if } \|\widehat{\boldsymbol{F}}_c\| > F_{relax}, \end{cases} \tag{12}$$

with $\boldsymbol{K}_f > 0$. The contact point will be forced away from the collision area along the direction of the estimated force $\widehat{\boldsymbol{F}}_c$, whereas the robot end-effector continues to accomplish the original trajectory.

Nevertheless, if the contact persists despite the robot self-motion and the estimated contact force will increase too much due to the specific task (e.g., the contact force is oriented against a desired motion direction), the robot should abandon the trajectory execution. To this end, it is useful to introduce a further threshold $F_{abort} > F_{relax}$. This upper threshold avoids damages to a (human) obstacle in the workspace and/or to the robot itself. When the contact force exceeds F_{abort} the task is completely abandoned, and the generalized admittance control scheme (6–7) will be realized at the contact point $\boldsymbol{x}_c$. The control strategy is summarized in the following expressions:

$$\dot{\boldsymbol{q}} = \begin{cases} \boldsymbol{J}^{\#}\dot{\boldsymbol{x}} + \boldsymbol{P}\boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q}), & \text{if } \|\widehat{\boldsymbol{F}}_c\| \le F_{relax} \\ \boldsymbol{J}^{\#}\dot{\boldsymbol{x}} + \boldsymbol{P}\big(\boldsymbol{K}_0(\boldsymbol{q}_d - \boldsymbol{q}) + \boldsymbol{J}_c^{\#}\boldsymbol{K}_f\widehat{\boldsymbol{F}}_c\big), & \\ \qquad\qquad \text{if } F_{relax} < \|\widehat{\boldsymbol{F}}_c\| \le F_{abort} & \\ \boldsymbol{J}_c^{\#}\dot{\boldsymbol{x}}_c + \boldsymbol{P}_c\boldsymbol{K}_n(\boldsymbol{q}_{c,d} - \boldsymbol{q}), & \text{if } \|\widehat{\boldsymbol{F}}_c\| > F_{abort}. \end{cases} \tag{13}$$

When the robot abandons the task and switches to admittance control, there is in general a discontinuity at the joint velocity level. This would lead to a jerky behavior of the robot that is not recommended, especially when the robot arm is physically interacting with a human. To avoid this, we use a very simple transition strategy. Let $\dot{\boldsymbol{q}}_{abort} = \dot{\boldsymbol{q}}(t_a)$ be the velocity of the robot when the norm of the contact force $\|\widehat{\boldsymbol{F}}_c\| = F_{abort}$, namely at $t = t_a$, and $\dot{\boldsymbol{q}}_{adm}$ be the velocity provided by the admittance control at $t > t_a$. The transition velocity $\dot{\boldsymbol{q}}_{tr}$ will be

$$\dot{\boldsymbol{q}}_{tr}(hT_c) = \alpha(h)\dot{\boldsymbol{q}}_{adm}(hT_c) + (1 - \alpha(h))\dot{\boldsymbol{q}}_{abort},$$

with $0 < \alpha \le 1$ defined as

$$\alpha = \frac{h}{N_s}, \qquad h = 1, \dots, N_s, \tag{14}$$

where N_s is the number of cycle times after which the transition will be completed, and T_c is the robot sampling interval. The larger the number N_s of the cycles is, the smoother and longer will be the transition.

B. *Discrete-time implementation*

The user-defined control law (13) has been implemented in discrete time, with control samples applied uniformly over time at $t = t_k = kT_c$. It has been shown in [17] how to determine a discrete-time joint velocity command for a redundant robot that executes a desired task while optimizing an acceleration/torque norm. This optimization can be achieved by introducing a forgetting factor $|\lambda| < 1$ in a discrete-time velocity control. Thus, the velocity used to command the robot joints becomes

$$\dot{\boldsymbol{q}}_{c,k} = \dot{\boldsymbol{q}}_k + \lambda\boldsymbol{P}_k\dot{\boldsymbol{q}}_{c,k-1}, \tag{15}$$

where $\dot{\boldsymbol{q}}_k$ is the joint velocity (13) and $\boldsymbol{P}_k = \boldsymbol{I} - \boldsymbol{J}_k^{\#}\boldsymbol{J}_k$, both evaluated at $t = kT_c$.

V. Experiments

A. Experimental setup

The scenario is composed by a robot that executes trajectories in the Cartesian space, while unknown obstacles may enter in contact with the manipulator arm. Experiments have been performed on a 7R KUKA LWR robot controlled though FRI at a cycle time of $T_c = 5$ ms. The workspace is monitored by a Microsoft®Kinect V2 depth sensor, positioned at a distance of 2.2 m behind the robot. The Kinect provides 512×424 depth images at 30 Hz rate. The hardware platform is a 64-bit Intel®Core i7-4790 CPU @4 Ghz, equipped with 16 GB DDR3 RAM. The implementation of our runs on a high-performance graphic board with a NVIDIA GTX970 GPU, organized in 1884 CUDA cores and capable of 26624 concurrent threads.

Three different processes coexist, running at different frequencies:

1) The vision process writes the GPU memory buffer with a new image as soon as a new depth image is provided by the Kinect (30 Hz).
2) The robot process produces the velocity command needed to control the robot (200 Hz).
3) The contact point estimation process, running on GPU, removes the manipulator from the depth image starting from its CAD model, and provides a new contact point ($\simeq 170$ Hz). Note that, even if a new image is provided only at 30 Hz, the robot is moving during this interval and the contact point may change.

To obtain more robust results, we have applied a scaling along the normal directions to the CAD model surfaces of the manipulator, enlarging them by about 3 cm. The main reason is to filter out the Kinect noise on depth images during the robot filtering process. Moreover, the vision sensor cannot capture very fast movements of the robot so that it could happen that the CAD model is not perfectly overlapping with the real robot. In such situations, the robot may recognize itself as an obstacle. Last but not least, for this robot in particular, we considered a *reduced* CAD model in which the first and second link of the manipulator are missing. In fact, even if we had localized the contact point on one of these two links, we would not be able to move away any of them. By doing so, we increase the performance of the algorithm. Obviously, for the robot filtering scheme we consider the entire CAD model. Having evaluated the average time needed to localize a contact point ($\simeq 6$ ms), we can classify this approach as real time for our purposes.

B. Results

During the experiments, moving obstacles (or human parts) touch the robot and push at different contact points. Our purpose is to illustrate how the original task is preserved, relaxed, or aborted by the proposed method. The control law is given by (13), with the contact force thresholds set to $F_{relax} = 5$ N and $F_{abort} = 25$ N. The other parameters were chosen as $\boldsymbol{K}_0 = 5 \cdot \boldsymbol{I}_7$ and $\boldsymbol{K}_f = 0.05 \cdot \boldsymbol{I}_3$. The admittance control parameters are $\boldsymbol{K}_a = 0.5 \cdot \boldsymbol{I}_3$, $\boldsymbol{K}_e = 3.5 \cdot \boldsymbol{I}_3$, and $\boldsymbol{K}_n = 0.6 \cdot \boldsymbol{I}_7$. The surveillance region size has been set to $\rho = 10$ cm. To complete the velocity transition in (14) the number of samples needed has been set to $N_s = 20$. Finally, the forgetting factor in (15) has been chosen as $\lambda = 0.8$.

Making reference to the plots of the signals in Fig. 5, in the first part of the experiment, a moving (human) obstacle enters the workspace, while the robot is executing an hexagonal trajectory on a vertical plane placed at $x = -0.55$ m, with a velocity of 20 cm/s. During normal operation, the robot is in its *idle state* (idle with respect to the more demanding contact handling and collaboration). When a contact is recognized on link 3 (at $t = 2.6$ s), the contact force increases and as soon as $\|\widehat{\boldsymbol{F}}_c\| \in (F_{relax}, F_{abort})$ the robot will enter in the *relax state*. Thus, the obstacle will move the contact point along the pushing force direction, without perturbing the task trajectory. Figure 4 shows few snapshots of this situation.

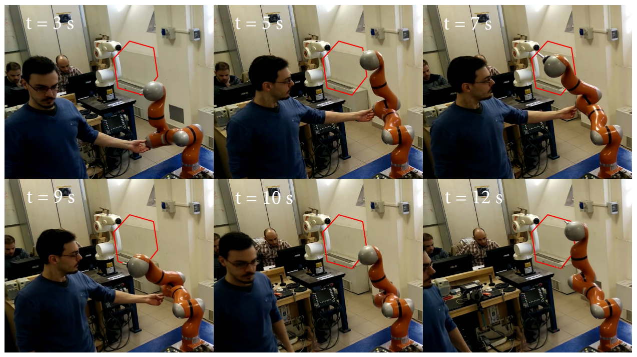

Fig. 4. Image flows when the robot reacts to a collision without perturbing the task. As soon as the contact breaks, the robot recovers its initial posture.

In the following part of the experiment, a human pushes the robot on link 6 and link 3 (at $t = 13$ s and $t = 26$ s, respectively) generating contact forces that lead the robot to its *abort state*, being $\|\widehat{\boldsymbol{F}}_c\| > F_{abort}$. In both cases, an admittance error arises and it will be completely recovered as soon as the contact breaks. Note that, the contact point $\boldsymbol{x}_c$ moves along the contact force direction, as shown in the admittance error plot in Figure 5. When the admittance error is recovered, the robot goes back to the *idle state*, resuming its original motion task.

The algorithm is not strictly related to human obstacles. In the second part of the experiment a box enters in contact with the robot on link 5 (at $t = 46$ s). For few seconds the robot remains in the *relax state*, continuing task execution without noticeable perturbations. As soon as the estimated contact force exceeds F_{abort}, the robot enters again in its *abort state* realizing the admittance control scheme at the contact point. Snapshots of this situation are shown in Fig. 6.

The complete experiment can be seen in the accompanying video clip. The plots can be much better appreciated when looking in parallel to the video. In particular, at $t = 73$ s the box is positioned between link 3 and table, and the robot goes into a stall position. It will enter and remain in its *abort state* due to the continued presence of the box that prevents admittance error recovering. When the box is removed, the robot recovers the error and resumes its original task.

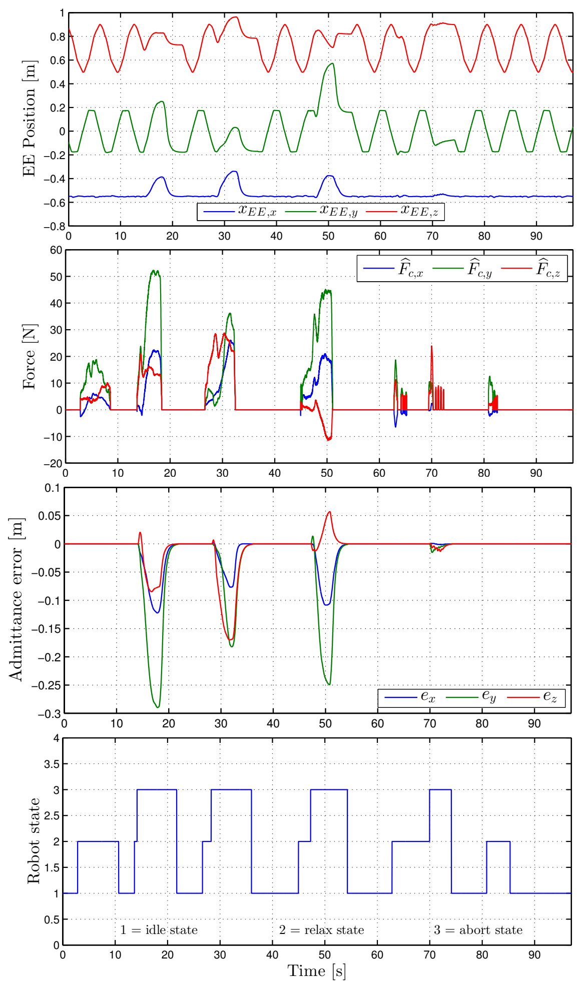

Fig. 5. Interaction control experiment. From the top: end-effector Cartesian position components, estimated contact force components, Cartesian components of the admittance error, and robot state monitor.

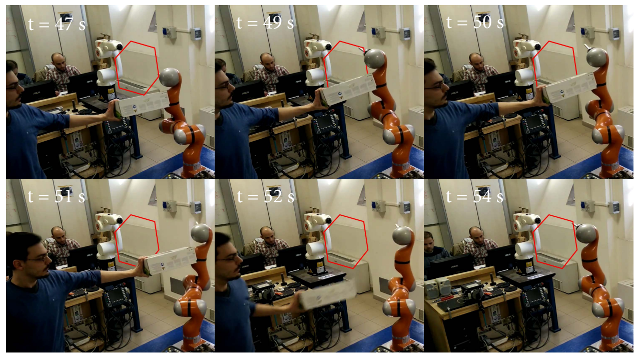

Fig. 6. Image flows when the robot reacts to a collision by abandoning the task. When contact is removed, the robot resumes its task.

VI. Conclusions

Relying on our previous results on contact point localization, contact force estimation, and generalized control schemes for robots interacting with humans, we have presented a new contact localization algorithm based on GPU parallel processing. A novel enhancement is the capability to localize in real time a contact on the entire robot surface and with all possible obstacles that could be present in the scene, without distinguishing their nature (humans, objects, other robots, etc.) and/or geometry. Once the contact point has been localized, the corresponding estimated contact force has been used in a kinematic control scheme that exploits redundancy in order to accommodate the physical contact without perturbing the original task that the robot was executing. Based on a simple logic, a trade-off has been obtained between limiting contact forces and preserving accuracy in continuous task execution. The idea to abandon and later resume the task, realizing an admittance control scheme at the contact, proved to be useful for keeping a safe interaction with unknown dynamic environments. Experiments with a KUKA LWR robot confirmed the practical applicability.

References

[1] SAPHARI. Safe and Autonomous Physical Human-Aware Robot Interaction. [Online]. Available: www.saphari.eu

[2] A. De Luca and F. Flacco, "Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration," in *Proc. IEEE Int. Conf. on Biomedical Robotics and Biomechatronics*, 2012, pp. 288–295.

[3] A. De Luca, A. Albu-Schäffer, S. Haddadin, and G. Hirzinger, "Collision detection and safe reaction with the DLR-III lightweight robot arm," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 1623–1630.

[4] S. Haddadin, A. Albu-Schäffer, A. De Luca, and G. Hirzinger, "Collision detection and reaction: A contribution to safe physical human-robot interaction," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 3356–3363.

[5] M. Erden and T. Tomiyama, "Human-intent detection and physically interactive control of a robot without force sensors," *IEEE Trans. on Robotics*, vol. 26, no. 2, pp. 370–382, 2010.

[6] S. Kuhn and D. Henrich, "Fast vision-based minimum distance determination between known and unknown objects," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2007, pp. 2186–2191.

[7] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach for evaluating distance to objects – with application to human-robot collision avoidance," *J. of Intelligent & Robotic Systems*, vol. 80, Suppl. 1, pp. 7–22, 2015.

[8] F. Flacco and A. De Luca, "Real-time computation of distance to dynamic obstacles with multiple depth sensors," *IEEE Robotics and Automation Lett.*, vol. 2, no. 1, pp. 56–63, 2017.

[9] K. Kaldestad, S. Haddadin, R. Belder, G. Hovland, and D. Anisi, "Collision avoidance with potential fields based on parallel processing of 3D-point cloud data on the GPU," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2014, pp. 3250–3257.

[10] A. Cirillo, F. Ficuciello, C. Natale, and S. Pirozzi, "A conformable force/tactile skin for physical human-robot interaction," *IEEE Robotics and Automation Lett.*, vol. 1, no. 1, pp. 41–48, 2016.

[11] G. Buondonno and A. De Luca, "Combining real and virtual sensors for measuring interaction forces and moments acting on a robot," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016, pp. 794–800.

[12] L. Manuelli and R. Tedrake, "Localizing external contact using proprioceptive sensors: The contact particle filter," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016, pp. 5062–5069.

[13] E. Magrini, F. Flacco, and A. De Luca, "Estimation of contact forces using a virtual force sensor," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2014, pp. 2126–2133.

[14] ——, "Control of generalized contact motion and force in physical human-robot interaction," in *Proc. IEEE Int. Conf. on Robotics and Automation*, 2015, pp. 2298–2304.

[15] E. Magrini and A. De Luca, "Hybrid force/velocity control for physical human-robot collaboration tasks," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2016, pp. 857–863.

[16] A. De Luca and L. Ferrajoli, "Exploiting robot redundancy in collision detection and reaction," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2008, pp. 3299–3305.

[17] F. Flacco and A. De Luca, "Discrete-time redundancy resolution at the velocity level with acceleration/torque optimization properties," *Robotics and Autonomous Systems*, vol. 70, pp. 191–201, 2015.