Journal of Intelligent & Robotic Systems
Published online 24 Oct 2014, DOI: 10.1007/s10846-014-0146-2

A Depth Space Approach for Evaluating Distance to Objects

with Application to Human-Robot Collision Avoidance

Fabrizio Flacco · Torsten Kroeger · Alessandro De Luca · Oussama Khatib

Special Issue on "Cognitive Robotics Systems: Concepts and Applications"
Received: 30 May 2014 / Accepted: 2 October 2014

Abstract We present a novel approach to estimate the distance between a generic point in the Cartesian space and objects detected with a depth sensor. This information is crucial in many robotic applications, e.g., for collision avoidance, contact point identification, and augmented reality. The key idea is to perform all distance evaluations directly in the depth space. This allows distance estimation by considering also the frustum generated by the pixel on the depth image, which takes into account both the pixel size and the occluded points. Different techniques to aggregate distance data coming from multiple object points are proposed. We compare the Depth space approach with the commonly used Cartesian space or Configuration space approaches, showing that the presented method provides better results and faster execution times. An application to human-robot collision avoidance using a KUKA LWR IV robot and a Microsoft Kinect sensor illustrates the effectiveness of the approach.

Keywords Depth space · Depth sensor · Kinect · Distance · Collision avoidance

1 Introduction

Evaluating distances between a generic point in space and multiple objects in the environment is an essential step for many applications, in robotics and beyond. The use of vision systems is the most common approach for this purpose, because of the capability of monitoring large workspaces and due to the rich nature of the information returned. While using a single camera allows to obtain only qualitative information about distances to moving objects (see, e.g., [6]), resorting to stereo vision makes it possible to collect full 3-D spatial information [12].

F. Flacco and A. De Luca
Dipartimento di Ingegneria Informatica, Automatica e Gestionale
Sapienza Università di Roma, Via Ariosto 25, 00185 Rome, Italy
E-mail: {fflacco,deluca}@diag.uniroma1.it

T. Kroeger and O. Khatib
Artificial Intelligence Laboratory
Stanford University, Stanford, CA 94305, USA
E-mail: {tkr, khatib}@stanford.edu

In the last few years, the release of powerful and cheap RGB-D sensors, like the Microsoft Kinect [23], that provide for each pixel in the image plane also the depth of the closest object along that pixel's projection, gave rise to novel uses and research solutions in a large variety of applications, including: *augmented reality*, where simulated objects have to interact with a real environment [16,17]; *virtual fixtures* in telemanipulation, with objects and shapes generating force feedback to the operator via a haptic device [20]; *collision avoidance* of a robot moving in a dynamic environment cluttered with obstacles [5,21,15]; *object recognition*, when the robot has to be distinguished from other moving objects [18]; simultaneous *localization and mapping* (SLAM), where a map of the environment is built and used to localize the camera position [13]; and, last but not least, *human-robot collaboration*, when robot and human have to coexist, physically get in contact, and exchange forces [3,2].

In all these works, as in most applications based on the use of depth sensors, the on-line estimation of distances between multiple obstacles and *control points*, which may either belong to a real object (e.g., attached to the robot links) or be virtual ones, is a basic requirement which needs to be performed in real time.

The most common approach for estimating distances uses the cloud of points obtained by projecting the depth image in the Cartesian space [14,8], often relying on the availability of open sources codes such as the Point Cloud Library (PCL) [19]. While this approach is suited to human natural reasoning about distances in Cartesian space, it does not exploit the information associated to a pixel in a complete way. This is because neither the pixel size nor the occluded points lying behind the detected obstacle along the projection ray(s) associated to the pixel are taken into account. In particular, this approach does not consider the 3D region related to each pixel called *frustum*, i.e., the portion of a pyramid left after its upper part has been cut off by a (skewed) plane.

In this paper, we show that the evaluation of point-to-object distances performed directly in the depth space allows a large performance improvement in terms of computational times. Moreover, a correct consideration of pixel frustum can be achieved in this way. The manuscript is based on our preliminary results presented in [5], where the use of the depth space to estimate the distance between robot points and obstacles was proposed for the first time. With respect to the original conference paper [5]: *(i)* we provide a more detailed comparison between Depth space and Cartesian space characteristics; *(ii)* the effect of finite pixel size is taken into account in distance computations; *(iii)* an experimental validation is added to illustrate the effectiveness and performance of the proposed approach; and, *(iv)* new collision avoidance experiments with a KUKA LWR IV robot are reported.

The paper is organized as follows. In Sect. 2, the representation of a point in the Cartesian, Configuration, and Depth spaces is recalled and their relations are detailed. The distance evaluation is presented in Sect. 3, where different techniques are proposed for aggregating distances to multiple points into a single information. Section 4 reports the results of an experimental comparison, where a virtual point is moved in the environment and different methods are used to compute the distances between the virtual point and real objects. Finally, the proposed approach is applied to human-robot collision avoidance experiments with a KUKA LWR IV and the results are reported in Sect. 5 and in the accompanying video.

2 Spaces for object representation

We consider an environment monitored by a depth sensor. The information given by the sensor has to be represented in a suitable, possibly discretized, space. The classical spaces used in robotic applications are the Cartesian space and the Configuration space. The former is the representation that humans are used to handle, while the latter is robot (and control) oriented. The Depth space is the native representation of depth sensor data, but it is not typically used as final representation space of object data, which are instead projected in one of the two previous spaces. The main characteristics of Cartesian, Configuration, and Depth spaces are recalled next.

2.1 Cartesian space

The 3-dimensional Cartesian space is characterized by a reference frame that identifies the origin (zero position) of the space and is used to specify the position of points and their distances, and the dimensions of objects.

A generic Cartesian point ${}^{C}\boldsymbol{P} = \begin{pmatrix} {}^{C}x & {}^{C}y & {}^{C}z \end{pmatrix}^T$ is described by three (dimensionally homogeneous) coordinates, which represent the distances of the point to the three orthogonal planes defined by the X, Y, and Z axes of the reference frame (Fig. 1). Given two points ${}^{C}\boldsymbol{P}_1$ and ${}^{C}\boldsymbol{P}_2$, their Cartesian distance is defined by using the (Euclidean) norm as

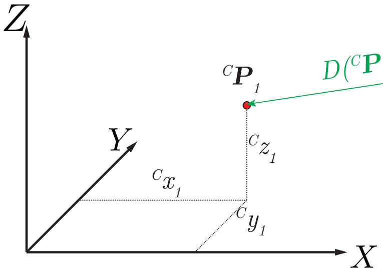

Fig. 1 Two points in Cartesian space, and their distance

$$\begin{aligned} D({}^{C}\boldsymbol{P}_1, {}^{C}\boldsymbol{P}_2) &= \|{}^{C}\boldsymbol{P}_1 - {}^{C}\boldsymbol{P}_2\| \\ &= \sqrt{({}^{C}x_1 - {}^{C}x_2)^2 + ({}^{C}y_1 - {}^{C}y_2)^2 + ({}^{C}z_1 - {}^{C}z_2)^2}. \end{aligned} \tag{1}$$

2.2 Configuration space

The information given by the depth sensor is often used to command and control the robot motion. In this situation, it is quite common to represent objects (usually defined as obstacles in this scenario) in the robot Configuration space, or C-Space. The C-Space is an n-dimensional manifold, where n is the minimum number of generalized coordinates (organized in an array $\boldsymbol{q}$) needed to describe the robot *posture*. These coordinates may have non-homogeneous units. For example, the generalized coordinates $\boldsymbol{q}$ of a mobile robot include the Cartesian position (x, y) on the plane and its orientation angle θ; similarly, the joint variables $\boldsymbol{q}$ of a manipulator may contain linear and angular quantities.

An obstacle point in the Cartesian space is represented in the C-Space as a C-Obstacle, which is the set of all robot configurations for which the robot is in contact (collides) with the point. It is possible to define the distance between two configurations as

$$D_Q(\boldsymbol{q}_1, \boldsymbol{q}_2) = \|\boldsymbol{q}_1 - \boldsymbol{q}_2\|. \tag{2}$$

2.3 Depth space

The Depth space is a non-homogeneous $2\frac{1}{2}$-dimensional space, where two elements represent the coordinates of the projection of a Cartesian point on a plane, and the third element represents the distance between the point and the plane. The depth space of an environment is the native representation given by a depth sensor, which is usually modeled as a classic pin-hole camera. The pin-hole camera model is composed by two sets of parameters: the intrinsic parameters in matrix $\mathcal{K}$, which model the projection of a Cartesian point on the image plane, and the extrinsic parameters in matrix $\mathcal{E}$, which represent the coordinate transformation between the reference and the sensor frame, i.e.,

$$\mathcal{K} = \begin{pmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathcal{E} = \left(\boldsymbol{R} \mid \boldsymbol{t} \right). \tag{3}$$

In (3), f is the focal length of the camera, s_x and s_y are the dimensions of a pixel (in meters), c_x and c_y are the pixel coordinates of the center (on the focal axis) of the image plane, and $\boldsymbol{R}$ and $\boldsymbol{t}$ are the rotation matrix and translation vector between the sensor frame and a reference frame.

Each pixel of a depth image contains the depth of the observed point, namely the distance between the Cartesian point and the camera image plane. Note that only the depth of the closest point along a given ray is stored. All occluded points that are beyond compose a region of uncertainty called the *gray area*. A typical gray area is illustrated in Fig. 2.

Consider a generic Cartesian point ${}^{C_r}\boldsymbol{P} = \left({}^{C_r}x \; {}^{C_r}y \; {}^{C_r}z \right)^T$, as expressed in the reference (world) frame. Its expression in the sensor frame is

$${}^{C_s}\boldsymbol{P} = \left({}^{C_s}x \; {}^{C_s}y \; {}^{C_s}z \right)^T = \boldsymbol{R}\,{}^{C_r}\boldsymbol{P} + \boldsymbol{t}, \tag{4}$$

and its projection ${}^{D}\boldsymbol{P} = \left(p_x \; p_y \; d_p \right)^T$ in the depth space is given by

$$p_x = \frac{{}^{C_s}x f s_x}{{}^{C_s}z} + c_x, \qquad p_y = \frac{{}^{C_s}y f s_y}{{}^{C_s}z} + c_y, \qquad d_p = {}^{C_s}z, \tag{5}$$

where p_x and p_y are the pixel coordinates in the image plane and d_p is the depth of the point. In the reverse direction, a point in the depth space is projected in

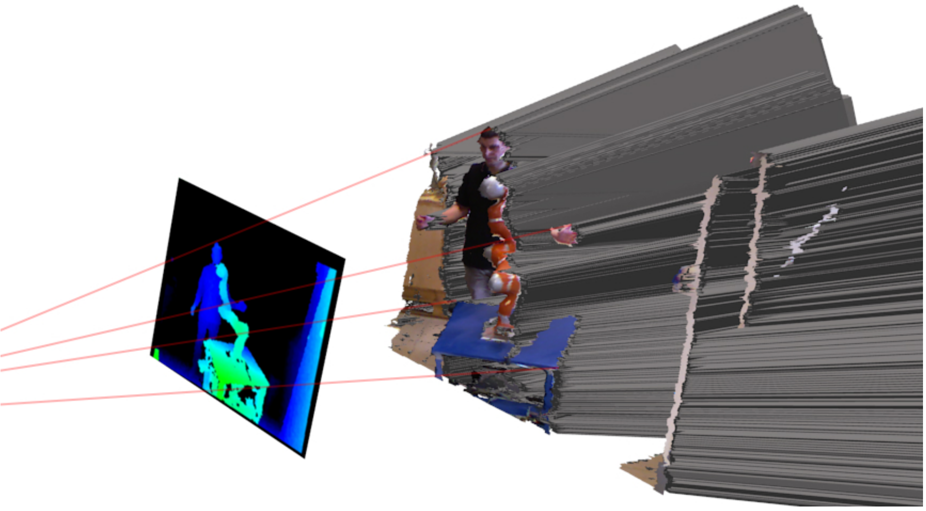

Fig. 2 Illustration of the gray area generated by a depth sensor (with a human, a robot, and a table in the environment)

the Cartesian sensor space as

$$^{C_s}x = \frac{(p_x - c_x)\, d_p}{f s_x}, \qquad ^{C_s}y = \frac{(p_y - c_y)\, d_p}{f s_y}, \qquad ^{C_s}z = d_p. \tag{6}$$

Note that when a point in the camera depth image is mapped in the Cartesian space, it represents only the object point nearest to the image plane projected in that pixel. On the other hand, also another information is simply coded in the depth space, namely that all Cartesian points generated by (6) with depth greater than d_p compose the gray area. Without any further information, this gray area should be considered as part of the perceived object.

3 Distance Evaluation

Consider a point of interest $\boldsymbol{P}$ in the Cartesian space[1] that will be called Control Point (CP). We would like to estimate the (minimum) distance between the Control Point and a generic object point $\mathbf{O}$ detected by the depth sensor. The steps needed depend on the space used to represent the points.

3.1 Cartesian space

The most common procedure to obtain the distance between the control point $\boldsymbol{P}$ and the recognized obstacle point in the depth image $^{D}\mathbf{O}$ is to project the latter in the Cartesian space by (6), and then use the simple point-to-point distance evaluation (1).

[1] In the rest of the paper, we omit the superscript for points expressed in the Cartesian reference frame.

This solution does not consider entirely the information embedded in the depth data, since occluded points are not included in this way as part of the object. Furthermore, we have to take into account that the sensor provides a discretized version of the depth space. In particular, an object point $\mathbf{O}$ is projected on the pixel ${}^{D}\bar{\mathbf{O}}$ in the image plane, with coordinates $\begin{pmatrix}\bar{o}_x & \bar{o}_y & d_o\end{pmatrix}^T = \begin{pmatrix}\mathrm{trunc}(o_x) & \mathrm{trunc}(o_y) & d_o\end{pmatrix}^T$. The depth information given by the sensor refers to the whole pixel, and thus also the pixel size has to be considered in the distance evaluation.

The correct procedure should consider the frustum generated by the depth space pixel, as illustrated in Fig. 3. Therefore, after the projection in the Cartesian space of depth data, the frustum representing the pixel object has to be also computed, then the minimum distance between the obtained frustum and the control point has to be evaluated.

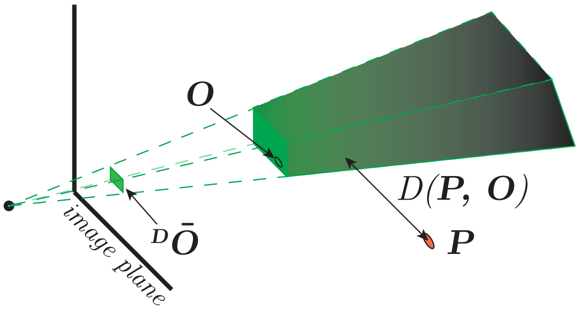

Fig. 3 Example of the frustum generated by a pixel in the image plane ${}^{D}\bar{\mathbf{O}}$ given by a point object $\mathbf{O}$ detected with the depth sensor, and its minimum distance to the control point $\boldsymbol{P}$

3.2 Configuration space

When the control point belongs to a robot and moves thus rigidly with it, the knowledge of the distance between the control point and the object in the configuration space is very useful for controlling the robot reaction or its interaction with the detected object. Despite this advantage, the representation of the C-Obstacle is not immediate, and in fact even a single point is represented as a region in the configuration space. A method for obtaining a discretized representation of the C-Obstacle associated to a real obstacle as detected by a depth sensor (or by stereo vision) was presented in [22]. The approach is indeed too costly in terms of computational time, and especially unsuitable whenever the dimension of the configuration space becomes large (e.g., for robots that are kinematically redundant w.r.t. the task).

3.3 Depth space

Consider the depth space representation of the object point ${}^{D}\mathbf{O} = \begin{pmatrix}o_x & o_y & d_o\end{pmatrix}^T$ captured by the sensor. In order to evaluate a useful Cartesian distance between the obstacle point $\mathbf{O}$ and a point of interest $\boldsymbol{P}$, which is also represented in the depth space as ${}^{D}\boldsymbol{P} = \begin{pmatrix} p_x \; p_y \; d_p \end{pmatrix}^T$ via eqs. (4) and (5), two possible cases can arise (see Fig. 4). If the obstacle point has a larger depth than the point of interest ($d_o > d_p$), then the distance is computed as

$$\begin{aligned} v_x &= \frac{(o_x - c_x)\, d_o - (p_x - c_x)\, d_p}{f s_x} \\ v_y &= \frac{(o_y - c_y)\, d_o - (p_y - c_y)\, d_p}{f s_y} \\ v_z &= d_o - d_p \end{aligned} \tag{7}$$

$$D(\boldsymbol{P}, \mathbf{O}) \simeq D_D({}^{D}\boldsymbol{P}, {}^{D}\mathbf{O}) = \sqrt{v_x^2 + v_y^2 + v_z^2}\ .$$

Otherwise, the distance w.r.t. the occluded points needs to be considered. For this, we assume the depth of the obstacle to be $d_o = d_p$ and the distance is then computed again from eq. (7). While the resulting value is not the actual Cartesian distance, the difference is expected to be negligible. Note that this distance evaluation is based on very simple relations, using only depth space data associated to the camera. Moreover, it takes into account properly and efficiently also the gray area contrary to what is done on occluded points with other methods.

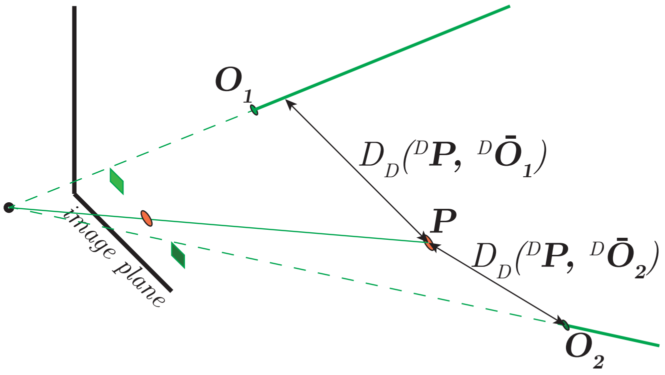
$\boldsymbol{O}_1$
$D_D({}^{D}\mathbf{P},\ {}^{D}\bar{\mathbf{O}}_1)$
$\boldsymbol{P}$
$D_D({}^{D}\mathbf{P},\ {}^{D}\bar{\mathbf{O}}_2)$
image plane
$\boldsymbol{O}_2$

Fig. 4 Depth space distance evaluation to a point of interest $\boldsymbol{P}$, showing the two possible cases of obstacle points whose depth is smaller ($\mathbf{O}_1$) or larger ($\mathbf{O}_2$) than the depth of the point of interest

At this stage, we can consider also the sensor discretization of the depth space. As already mentioned, the object point $\mathbf{O}$ is projected on the pixel ${}^{D}\bar{\mathbf{O}}$ in the image plane with coordinates $\begin{pmatrix} \bar{o}_x \; \bar{o}_y \; d_o \end{pmatrix}^T = \begin{pmatrix} \mathrm{trunc}(o_x) \; \mathrm{trunc}(o_y) \; d_o \end{pmatrix}^T$. With reference to Fig. 5, the Cartesian object generated by the (finite) object pixel ${}^{D}\bar{\mathbf{O}}$ is a frustum with base at d_o. The minimum distance between the square frustum and the control point is on the frustum surface. To obtain this distance, we work directly in the depth space by finding the sub-pixel point ${}^{D}\hat{\mathbf{O}} = \begin{pmatrix} \hat{o}_x \; \hat{o}_y \; d_o \end{pmatrix}$ nearest to $\begin{pmatrix} p_x \; p_y \end{pmatrix}$ that belong to the frustum, i.e.,

$$\hat{o}_x = \begin{cases} \bar{o}_x & p_x < \bar{o}_x \\ \bar{o}_x + 1 & p_x > \bar{o}_x + 1 \\ p_x & \text{otherwise,} \end{cases} \qquad \hat{o}_y = \begin{cases} \bar{o}_y & p_y < \bar{o}_y \\ \bar{o}_y + 1 & p_y > \bar{o}_y + 1 \\ p_y & \text{otherwise.} \end{cases} \tag{8}$$

As illustrated in Fig. 5, the distance can be finally evaluated as $D_D({}^D\boldsymbol{P}, {}^D\hat{\mathbf{O}})$.

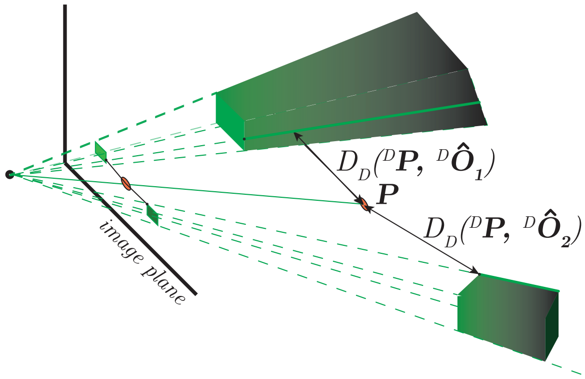
$D_D({}^D\boldsymbol{P},\ {}^D\hat{\mathbf{O}}_1)$
$\boldsymbol{P}$
$D_D({}^D\boldsymbol{P},\ {}^D\hat{\mathbf{O}}_2)$
image plane

Fig. 5 Depth space distance evaluation to a point of interest $\boldsymbol{P}$ when also the pixel size is taken into account, showing the two possible cases of obstacle depth smaller or larger than the depth of the point of interest

In some applications, as in collision checking, retrieving the distance information is sufficient, while in some other cases, e.g., for collision avoidance, we need also the unit (normalized) vector between the control point and the nearest point on the frustum. This vector is simply given by

$$\boldsymbol{V}({}^D\boldsymbol{P}, {}^D\hat{\mathbf{O}}) = \frac{\begin{pmatrix} v_x & v_y & v_z \end{pmatrix}^T}{D_D({}^D\boldsymbol{P}, {}^D\hat{\mathbf{O}})}. \tag{9}$$

The complete distance evaluation method is summarized in pseudocode form as Algorithm 1.

3.4 Aggregation of multiple obstacle points

We would like now to evaluate distances between the control point $\boldsymbol{P}$ and all obstacles sufficiently close to it. Consider a Cartesian *region of surveillance* $\boldsymbol{S}$, made by a cube of side 2ρ centered at $\boldsymbol{P}$, where the presence of obstacles must be detected. The associated region of surveillance in the image plane has dimensions

$$x_s = \rho \frac{f s_x}{d_p - \rho}, \qquad y_s = \rho \frac{f s_y}{d_p - \rho}. \tag{10}$$

Thus, the distance evaluation should be applied to all pixels in the depth image plane within the region of surveillance

$$\boldsymbol{S}_D = \left[p_x - \frac{x_s}{2}, p_x + \frac{x_s}{2}\right] \times \left[p_y - \frac{y_s}{2}, p_y + \frac{y_s}{2}\right] \times \left[d_p - \rho, d_p + \rho\right]. \tag{11}$$

Since the evaluation for each obstacle pixel is completely independent, distances may be computed also in parallel, thus speeding up the method.

Algorithm 1 Distance evaluation in the depth space

```
function [D, $\boldsymbol{V}$]=getDistanceDepthSpace($^D\bar{\mathbf{O}}$, $^D\boldsymbol{P}$)
if $p_x < \bar{o}_x$ then
    $\hat{o}_x = \bar{o}_x$
else
    if $p_x > \bar{o}_x + 1$ then
        $\hat{o}_x = \bar{o}_x + 1$
    else
        $\hat{o}_x = p_x$
    end if
end if

if $p_y < \bar{o}_y$ then
    $\hat{o}_y = \bar{o}_y$
else
    if $p_y > \bar{o}_y + 1$ then
        $\hat{o}_y = \bar{o}_y + 1$
    else
        $\hat{o}_y = p_y$
    end if
end if

if $d_o < d_p$ then
    $d_o = d_p$
end if

$v_x = \left(\left(\hat{o}_x - c_x\right) d_o - \left(p_x - c_x\right) d_p\right) / f s_x$
$v_y = \left(\left(\hat{o}_y - c_y\right) d_o - \left(p_y - c_y\right) d_p\right) / f s_y$
$v_z = d_o - d_p$
$\boldsymbol{V}' = \begin{pmatrix} v_x & v_y & v_z \end{pmatrix}^T$
$D = \mathrm{norm}(\boldsymbol{V}')$
$\boldsymbol{V} = \boldsymbol{V}'/D$
```

Most of the times, distances to multiple obstacle points are computed in order to generate a reactive motion of a (robot) control point in face of dynamic obstacles. To this end, the contribution of all points in the region of surveillance can be aggregated in different ways into a single information, according to the desired intended robot behavior. We present next a few common aggregation methods, and illustrate how to apply them within our depth space approach.

3.4.1 Minimum distance vector

When only the minimum distance is required, the number of distance evaluations can be reduced by considering pixels that are closer to (p_x, p_y) first. As soon as a new local minimum

$$D_{\min}(\boldsymbol{P}) = \min_{^D\hat{\mathbf{O}} \in \boldsymbol{S}'_D} D_D(^D\boldsymbol{P}, ^D\hat{\mathbf{O}}) < \rho \tag{12}$$

is found among the pixels in the already explored area $\boldsymbol{S}'_D \subset \boldsymbol{S}_D$, the region of surveillance can be shrunk by setting $\rho = D_{\min}$ and using again eq. (10). This contraction of the surveilled area, together with the fact that distance computation is applied only to pixels whose depth is in $\boldsymbol{S}'_D$, reduces the computational burden of the algorithm. The associated unit vector $\boldsymbol{V}_{\min}(\boldsymbol{P}) = \boldsymbol{V}(^D\boldsymbol{P}, ^D\hat{\mathbf{O}}_{\min})$ is the one obtained with the obstacle point that generates the minimum distance ${}^{D}\hat{\mathbf{O}}_{\min} = \arg\min_{{}^{D}\hat{\mathbf{O}}\in \boldsymbol{S}'_D} D_D({}^{D}\boldsymbol{P}, {}^{D}\hat{\mathbf{O}})$.

3.4.2 Mean distance vector

In some cases, we would like to have a single distance information about all objects surrounding the control point. A possible aggregation method is to compute the mean distance as

$$D_{\text{mean}}(\boldsymbol{P}) = \frac{\sum_{{}^{D}\hat{\mathbf{O}}\in \boldsymbol{S}'_D} D_D({}^{D}\boldsymbol{P}, {}^{D}\hat{\mathbf{O}})}{N}, \tag{13}$$

where N is the number of object depth points detected by the sensor inside the surveillance area $\boldsymbol{S}_D$. Similarly, the associated unit vector is

$$\boldsymbol{V}_{\text{mean}}(\boldsymbol{P}) = \frac{\sum_{{}^{D}\hat{\mathbf{O}}\in \boldsymbol{S}'_D} \boldsymbol{V}({}^{D}\boldsymbol{P}, {}^{D}\hat{\mathbf{O}})}{N}. \tag{14}$$

3.4.3 Hybrid distance vector

In applications where a control point is commanded to react to the presence of objects, e.g., in collision avoidance, both the minimum and the mean distance approaches are not particularly effective. In fact, the minimum distance method could drive the control point toward a second object, and if this second object becomes then the nearest one, it could push the control point back toward the first object, resulting in an undesirable oscillating effect. On the other hand, the mean distance approach is affected by the topology of the obstacles, namely it depends on the ratio of the number of near and far obstacles. Such behavior is also not desirable, since the presence of a close object should provide always the same control reaction, no matter if other obstacles are near or far to it. In such cases, we propose to use a hybrid method with

$$D_{\text{hybrid}}(\boldsymbol{P}) = D_{\min}(\boldsymbol{P}) \qquad \text{and} \qquad \boldsymbol{V}_{\text{hybrid}}(\boldsymbol{P}) = \boldsymbol{V}_{\text{mean}}(\boldsymbol{P}). \tag{15}$$

This allows to react according to the nearest object for the intensity, while taking into consideration all the objects in the surveillance area for the reaction direction.

3.5 Avoiding self distances

When the control point belongs to a real object which is also detected by the depth sensor, it is important to remove it from the depth image. Without removing the control point, the minimum distance to the object would always be equal to zero. Different techniques can be used to remove the object that contains the control point. The simplest method is to remove a portion of the image compatible with the actual shape of the object, or removing all adjacent points. If a 3D-model of the object is known, an efficient method for object removal from the depth image using the GPU is presented in [1].

4 Validation and comparison

To validate the proposed Depth space approach, and to compare it with a common Cartesian space approach, we present a simple experiment where a virtual control point moves in a real environment. The relevant environment is mainly composed by two walls, one horizontal and one vertical, positioned on the robot supporting table as shown in Fig. 6. The presence of the robot manipulator is here irrelevant, but for convenience we used the robot base frame as reference frame.

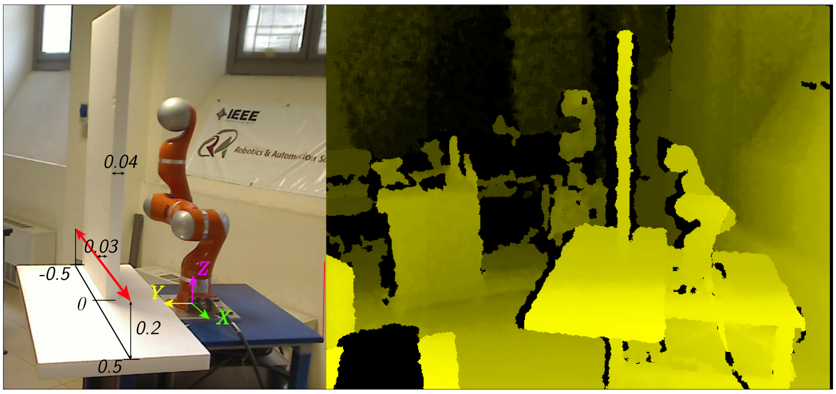

Fig. 6 Environment used for validation and comparison of methods. The virtual control point moves on the red segment shown in the left picture, while the depth image given by the Kinect is shown on the left (lighter colors refer to obstacle points with smaller depth)

The virtual control point $\boldsymbol{P}$ follows a line defined by $y = 0.4$ and $z = 0.2$ [m], while the X coordinate moves in the range $x \in [-0.5, 0.5]$ [m]. The control point sees the horizontal wall as a planar $X-Y$ surface at a height $Z = 0.04$ [m], while the vertical wall is seen as a planar $Y-Z$ surface, with $X = 0$ and $Y \in [0.33, 0.37]$ [m], and a planar $X-Z$ surface, with $Y = 0.37$ and $X \in [-0.5, 0]$ [m]. From the depth sensor view of the environment shown in Fig. 6, it follows clearly that only the $Y-Z$ surface of the vertical wall is captured, while the $X-Z$ surface is completely occluded. By construction, the nearest obstacle to the control point is the vertical wall when $x \in [-0.5, 0.1572]$ [m], and the horizontal wall when $x \in [0.1572, 0.5]$ [m]. The region of surveillance used in the following tests is defined by $\rho = 0.3$ [m].

The first approach is a Cartesian space method that evaluates the minimum distance between the control point and the point cloud associated to the pixels in the surveillance area. Each pixel in the surveillance area $\boldsymbol{S}_D$ is projected in the Cartesian reference frame using (4) and (6), and the distance to the control point is evaluated then by eq. (1). Figure 7 reports the minimum distance estimated during the experiment as a function of the X-coordinate of the control point. Having considered only the point cloud, and not the entire frustum, the vertical wall is not correctly taken into account, and points on the horizontal plane are recognized as the nearest ones even around $x \in [-0.5, -0.1572]$ [m], where the vertical plane is in fact nearer. Furthermore, the sensor noise induces also a discontinuous behavior in the distance unit vector which may preclude its use in practical applications.

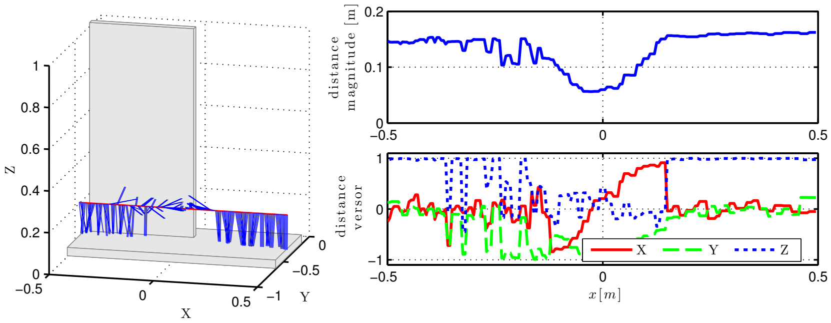

Fig. 7 Minimum distance estimated with the Cartesian space approach. Distance vectors in the Cartesian space [left]; evaluated magnitude of the distance vector [right, top] and components of the distance unit vector [right, bottom]

Figures 8, 9 and 10 refer to the proposed Depth space approach, using different methods for aggregating multiple object points. The results obtained with the minimum distance method are reported in Fig. 8. It can be verified that the minimum distance is correctly estimated, since both walls are now taken into account. The minimum estimated distance to the vertical wall falls below 0.03 [m] (which is its real lower bound) since, due to occlusion, the gray area appears as nearer. The obtained distance unit vector is much less sensitive to sensor noise than with the previous point cloud approach, but it still experiences a discontinuity.

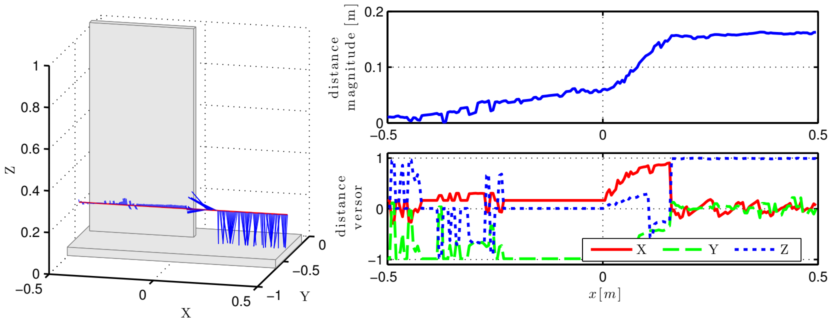

Fig. 8 Minimum distance estimated with the Depth space approach. Distance vectors in the Cartesian space [left]; evaluated magnitude of the distance vector [right, top] and components of the distance unit vector [right, bottom]

When the mean aggregation method is used, undesired discontinuities of the distance unit vector are eliminated, see Fig. 9. Moreover, the mean distance vector considers all obstacles in the surveillance area, which may be useful in some applications. The drawback is that the magnitude of this distance vector averages between near and far obstacles, and thus the main information we were looking for, namely minimum distance, will not be provided. The hybrid method is a trade off between having information about how close is the control point is to other objects and how these objects are distributed around the control point. The result

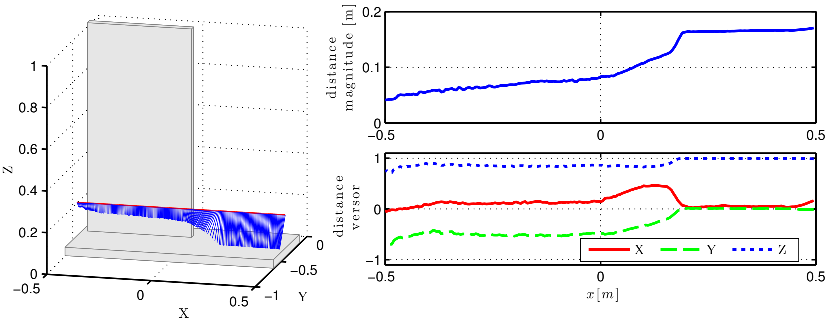

Fig. 9 Mean distance vector estimated with the Depth space approach. Distance vectors in the Cartesian space [left]; evaluated magnitude of the distance vector [right, top] and components of the distance unit vector [right, bottom]

obtained with the hybrid distance vector is shown in Fig. 10. A collision avoidance algorithm based on this method is presented in Sect. 5.

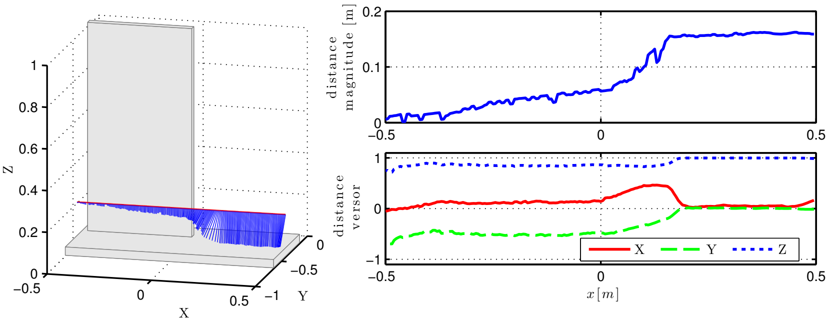

Fig. 10 Hybrid distance vector estimated with the Depth space approach. Distance vectors in the Cartesian space [left]; evaluated magnitude of the distance vector [right, top] and components of the distance unit vector [right, bottom]

Beside the extra capability of considering easily occluded points and pixel size, another main feature of the proposed Depth space approach is its computational efficiency, and thus its suitability for tracking fast motion. We remark that, in general, it may not be sufficient to compute distances at the same frequency rate of the camera/sensor frames. In fact, the control point could move at a fast speed, so that distances have to be recomputed on the fly even before the depth image is updated using the next image frame.

Figure 11 shows a comparison of the actual computational times involved in the presented validation experiments. Because of the large differences in computational times between the Cartesian and the Depth approach, and also among aggregation methods used in the latter, a logarithmic scale has been used. Experiments were conducted on a Intel Core i7-2600 CPU 3.4GHz, with 8GB of RAM. Despite of the fact that only the point cloud (and no frustum) has been considered in the Cartesian space approach that we implemented, this approach has 70.6 [ms] (14.17 [Hz]) as worst (longest) execution cycle time during the entire motion. With

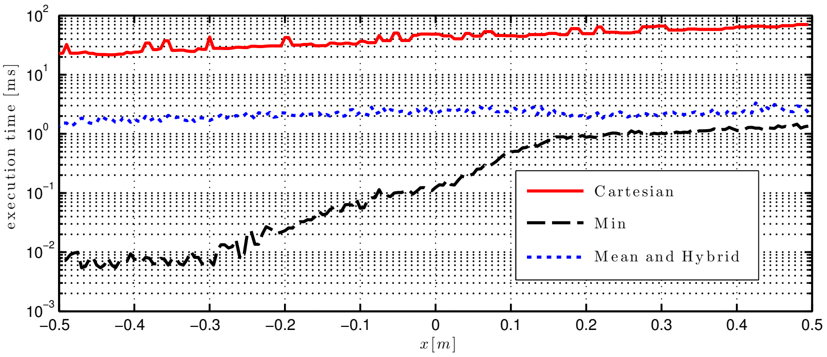

Fig. 11 Execution times for estimating the final distance vector with the Cartesian space approach, and with the Depth space approach when using the three reported methods

the proposed Depth space approach. the minimum distance method has a worst execution time of 1.46 [ms] (684.76 [Hz]), almost two order of magnitudes faster than before. Thanks also to the contraction of the surveillance area, as detailed in Sect. 3.4.1, when an obstacle is very close to the control point only few pixels need to be checked, which is then reflected in a small execution time: in this experiment, the minimum execution time was 5.3 [μs]. The mean and the hybrid methods for aggregation of data have the same computational cost, with their worst execution time equal to 3.335 [ms] (299.86 [Hz]).

In conclusion, the presented validation experiment shows that the Depth space approach not only provides more information but distance information can also be computed faster than with common Cartesian space approaches.

5 Human-robot collision avoidance

To show the effectiveness of the Depth space approach, we present as a case study some laboratory experiments where a fast and correct distance evaluation is crucial. This occurs in collision avoidance, where robot-obstacle (or robot-human) distances need to be computed in real time so as to generate evasive maneuvers. More specifically, we will use the evaluated distances in two different ways, as a repulsive action at the velocity level for the robot end-effector and as a virtual obstacle for a number of other control points placed along the robot body.

5.1 Repulsive action

Once the robot-obstacle distances have been evaluated, they are used to modify on-line the current trajectory of the manipulator so as to avoid collision. Many different approaches for obstacle avoidance have been proposed, see, e.g., [9, 7, 21]. We present here a simple but effective method based on the generation of repulsive vectors in Cartesian space, which can then be used as basic input for any preferred collision avoidance algorithm.

Associated to the hybrid distance vector from detected obstacles to the end-effector position $\boldsymbol{P}_{EE}$, as obtained by Algorithm 1 and the aggregation method (15), a repulsive vector is defined as

$$^{C_s}\boldsymbol{\Psi}\left(\boldsymbol{P}_{EE}\right) = v\left(\boldsymbol{P}_{EE}\right)\boldsymbol{V}_{\text{hybrid}}(\boldsymbol{P}_{EE}). \tag{16}$$

All obstacle points are taken into account for the *direction* of the unit vector $\boldsymbol{V}_{\text{hybrid}}(\boldsymbol{P}_{EE})$ of the repulsive action. For its *magnitude*, we set

$$v\left(\boldsymbol{P}_{EE}\right) = \frac{V_{\max}}{1 + e^{\left(D_{\text{hybrid}}(\boldsymbol{P}_{EE})(2/\rho)-1\right)\alpha}}, \tag{17}$$

where $V_{\max}$ is the maximum admissible magnitude and $\alpha > 0$ is a shape factor. The magnitude v of the repulsive vector will approach $V_{\max}$ when $D_{\text{hybrid}}(\boldsymbol{P}_{EE}) = 0$, and will approach zero when the distance reaches ρ (beyond ρ, $^{C_s}\boldsymbol{\Psi}$ is not defined). A typical profile of the magnitude as a function of the hybrid distance is shown in Fig. 12.

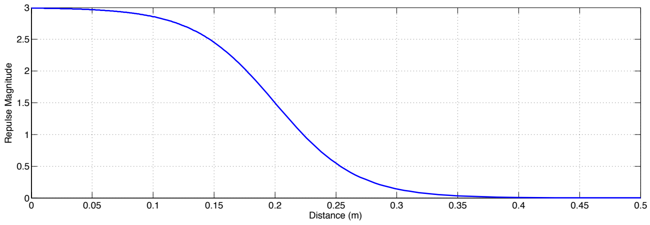

Fig. 12 Repulsive magnitude in eq. (17), with $V_{\max} = 3$ [m/s], $\rho = 0.4$ [m], and $\alpha = 6$

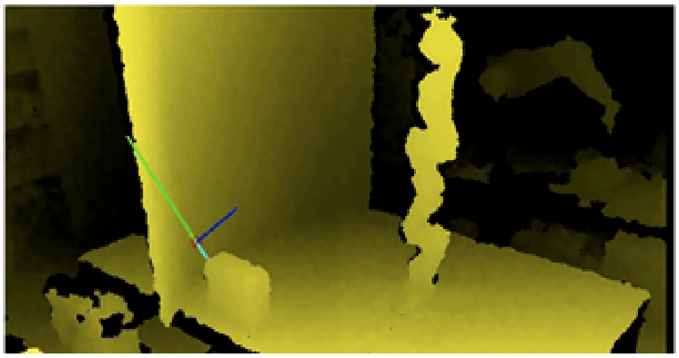

Fig. 13 Example of repulsive vector computation. The point of interest $\boldsymbol{P}$ is represented by a red circle, and the minimum distance is represented in cyan. The repulsive vector obtained by using the minimum distance only is shown in green, while the one obtained by using all points in the range of surveillance is in blue. It can be seen that the green repulsive vector points to another obstacle (dangerous), while the blue vector points to a free area (safer)

In this way, all obstacle points contribute to the direction of the resulting repulsive vector, while the magnitude depends only on the minimum distance to all obstacle points. If the magnitude were computed using all points, it would be influenced by the number of obstacle points. Similarly, if the magnitude were given by the mean value of the distances, it would be affected by the ratio of near to far obstacles. Such behaviors are not desirable, especially for a close obstacle with high risk of collision. The main benefits of using all points for computing the repulsive (unit) direction are that *i)* the repulsive vector is less sensible to noise of the depth sensor, producing a smoother variation of the pointing direction, and *ii)* the presence of multiple obstacles is handled in a better way, as shown in Fig. 13.

All above repulsive vectors are expressed in the camera frame, but can be transformed in the reference frame as ${}^{C_r}\boldsymbol{\Psi}(\boldsymbol{P}) = \boldsymbol{R}^{T\,C_s}\boldsymbol{\Psi}(\boldsymbol{P})$. The motion task for the robot is specified by a desired end-effector velocity $\dot{\boldsymbol{x}}_d$ in the Cartesian space. For obstacle avoidance of the end-effector control point $\boldsymbol{P}_{EE}$, we simply take the repulsive vector as a repulsive velocity. Thus, the original desired end-effector velocity $\dot{\boldsymbol{x}}_d$ will be modified into a commanded one $\dot{\boldsymbol{x}}_c$ as

$$\dot{\boldsymbol{x}}_c = \dot{\boldsymbol{x}}_d + {}^{C_r}\boldsymbol{\Psi}(\boldsymbol{P}_{EE}). \tag{18}$$

Without loss of generality, we consider the manipulator to be commanded at the joint velocity level. The joint velocity obtained by (pseudo)inversion as

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{\#}(\boldsymbol{q})\,\dot{\boldsymbol{x}}_c \tag{19}$$

is then used as target velocity command for the control algorithm.

This is indeed a simple, particular form of the classical artificial potential field method [9], which has been chosen here mainly to prove the effectiveness of the computed repulsive vectors. It is well known that the main drawback of this method is the presence of local minima. However, note that from a safety point of view (especially in human-robot interaction) it is acceptable that the robot stops whenever it is not able to pass by the obstacles. In any event, starting from this basic algorithm, more complex versions can be developed —see, e.g., [7].

5.2 Cartesian constraints

For the other control points placed along the robot structure, we use a slightly different approach. Obstacles do not produce repulsive velocities on these control points, but are treated rather as Cartesian constraints with artificial forces that are translated into joint velocity constraints as detailed in [4]. Our approach, based on the modification of joint velocity constraints while exploiting kinematic redundancy, will preserve the desired end-effector task as far as possible. Had we considered instead repulsive velocities as for the end-effector, we would need to manage multiple robot tasks using the magnitudes of the repulsive vectors as associated priorities. While this approach is indeed feasible, it presents some conflicting issues. If the end-effector task has always the highest priority, then collision avoidance for the robot links could not be guaranteed. On the other hand, if the end-effector task is not privileged, then its trajectory could be arbitrarily modified even when there is no risk of end-effector collisions.

Let $\boldsymbol{C}$ be one of the control points belonging to a generic robot link, and $\boldsymbol{J}_C$ the Jacobian of the direct kinematics for the position of $\boldsymbol{C}$. Let $D_{\min}(\boldsymbol{C})$ be the minimum distance between the control point and all obstacle points $\mathbf{O} \in \boldsymbol{S}(\boldsymbol{C})$ in its associated surveillance region. The risk of collision is defined by the function

$$f\left(D_{\min}(\boldsymbol{C})\right) = \frac{1}{1 + e^{(D_{\min}(\boldsymbol{C})(2/\rho)-1)\alpha}}\ , \tag{20}$$

where ρ and α have been introduced in eqs. (10) and (17), respectively. Scaling by (20) the unit vector $\boldsymbol{V}_{\min}$ along the minimum distance direction, we treat the resulting vector as a Cartesian *constraint force* and convert it in the joint space as

$$\boldsymbol{s} = \boldsymbol{J}_C^T \left[\boldsymbol{V}_{\min}(\boldsymbol{C})\, f\left(D_{\min}(\boldsymbol{C})\right)\right]. \tag{21}$$

The component s_i of $\boldsymbol{s}$ represents the 'degree of influence' of the Cartesian constraint on the ith joint, for $i = 1, \ldots, n$. From these, we reshape the admissible velocity limits of all joints that are influenced by the Cartesian constraint using again the risk of collision function as

$$\begin{aligned} \text{if } s_i \geq 0, \quad & \dot{q}_{\max,i} = V_{\max,i}\big(1 - f\left(D_{\min}(\boldsymbol{C})\right)\big) \\ \text{else,} \quad & \dot{q}_{\min,i} = -V_{\max,i}\big(1 - f\left(D_{\min}(\boldsymbol{C})\right)\big), \end{aligned} \tag{22}$$

where $V_{\max,i}$ is the original bound on the ith joint velocity, i.e., $|\dot{q}_i| \leq V_{\max,i}$, for $i = 1, \ldots, n$. In practice, joint motions that are in contrast with the Cartesian constraint are scaled down. When the constraint is too close, all joint motions that are not compatible with the constraint will be denied. Multiple Cartesian constraints are taken into account by considering, for each joint i, the minimum scaling factor obtained for all the constraints. With this approach, collision avoidance for the robot body has always the highest priority, while the end-effector task will continue to be correctly executed until it is compatible with the Cartesian constraints. Otherwise, the robot stops and a recovery method should be applied.

5.3 Experiments

5.3.1 Setup

The experimental setup consists of a KUKA LWR IV manipulator with $n = 7$ revolute joints, executing tasks that are defined only in terms of the position of its end-effector (i.e., of dimension $m = 3$) while unknown dynamic obstacles, including a human, enter its workspace. For the primary Cartesian motion task, this robot has degree of redundancy $n - m = 4$. The robot operates at a control cycle of 2 ms. The workspace is monitored by a Microsoft Kinect$^{\text{TM}}$ depth sensor, positioned at a horizontal distance of 1.5 [m] and at a height of 1.2 [m] w.r.t. the robot base frame. The Kinect captures 640×480 depth images at a frequency of 30 Hz. The implementation of our collision avoidance approach runs on an eight-core CPU. Four processors execute the repulsive velocity computation, and the other four enable visualization and robot motion control.

Note that three different run-time processes are present, working at three different frequencies.

1. The vision process captures the depth image and removes the manipulator from each new image at the sensor frequency (30 Hz).
2. The on-line trajectory generation algorithm of [11, 10] produces a joint velocity command at the same cycle time of the robot controller (500 Hz).
3. The obstacle avoidance process computes a repulsive vector at a frequency lying between those of the vision and control processes. In fact, even if a new depth image is available only at 30 Hz, the manipulator is moving during this interval and the repulsive vector changes accordingly.

5.3.2 Results

We present three different scenarios that highlight the features of the presented approach. The basic manipulator task is to continuously move the end-effector through six Cartesian points that forms an hexagon in the $Y-Z$ plane defined by $X=-0.6$ [m]. The parameters used are $\rho=0.4$ [m], $V_{\max}=1.5$ [m/s], and $\alpha=5$.

The first scenario (Fig. 14) is one of human-robot coexistence, in which the human tries to touch the robot end-effector with his hand. With the proposed approach, robot-to-hand distances are evaluated at a high rate, allowing the robot to perform an immediate evasive maneuver. The accompanying video shows also more results of this kind. In a second scenario (Fig. 15), collision between a planar moving obstacle and the robot end-effector has to be avoided. In this case, the importance of having considered also occluded points is emphasized. For instance, at $t=5$ [s] the plane is almost completely occluded; nonetheless, the correct repulsive velocity is obtained. In the third scenario (Fig. 16), an obstacle is inserted on the motion trajectory of the robot elbow. Between $t=0$ and $t=6$ [s], the manipulator executes the desired Cartesian hexagon going through the same robot postures. When the obstacle is inserted at $t=10$ [s], it is considered as a Cartesian constraint and converted into virtual joint velocity limits by our algorithm. The robot exploits its task redundancy to accommodate the new limits, and reconfigures its posture so as to continue successfully the execution of the desired end-effector trajectory while avoiding the obstacle. The complete experiments are included in the video.

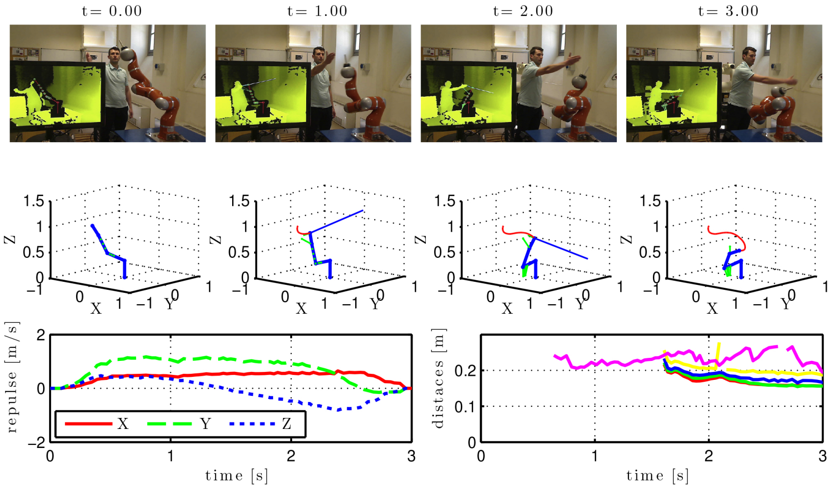

Fig. 14 Scenario 1. A human operator tries to touch the robot end-effector. First and second rows shows four instant of the experiment, at $t=0,1,2,3$ [s], with snapshots in the first row and evolution of variables in the second row: end-effector trajectory [red]; distances between a control point and the nearest obstacle [green]; end-effector repulsive velocity [blue]. The last row shows the components of the end-effector repulsive velocity [left] and the minimum distances for the other control points [right]

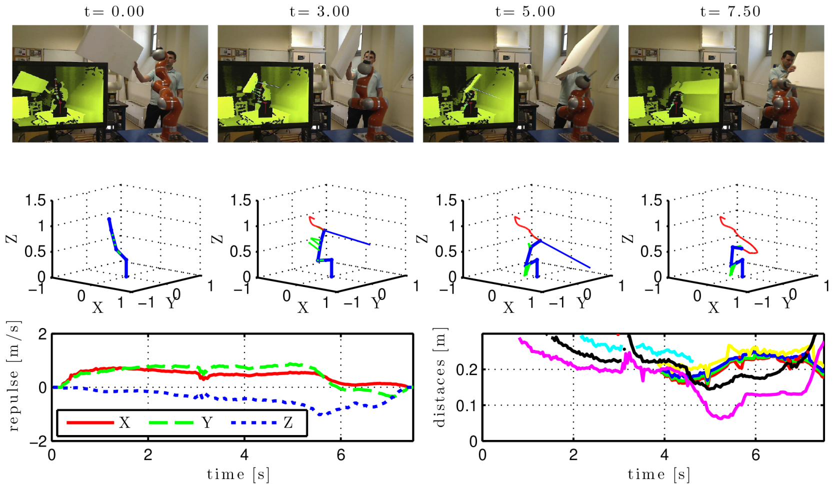

Fig. 15 Scenario 2. A collision between the robot end-effector and a planar object is avoided. First and second rows shows four instant of the experiment, at $t = 0, 3, 5, 7.5$ [s], with snapshots in the first row and evolution of variables in the second row: end-effector trajectory [red]; distances between a control point and the nearest obstacle [green]; end-effector repulsive velocity [blue]. The last row shows the components of the end-effector repulsive velocity [left] and the minimum distances for the other control points [right]

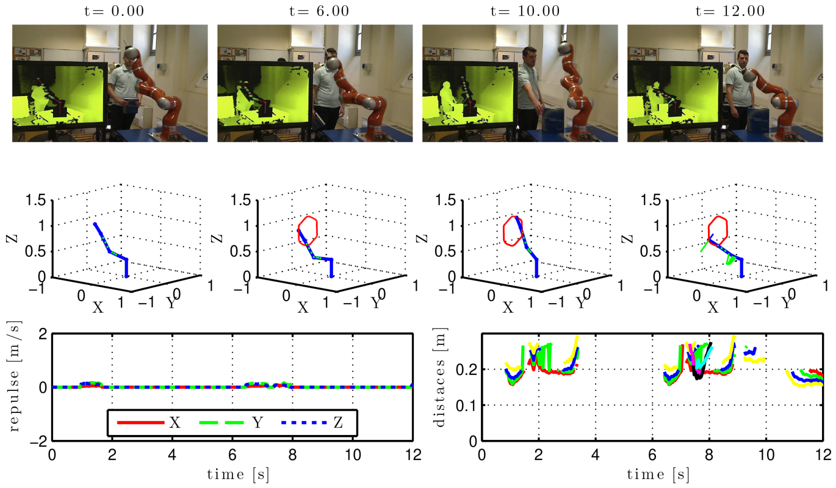

Fig. 16 Scenario 3. An obstacle is positioned on trajectory of the robot elbow. First and second rows shows four instant of the experiment, at $t = 0, 6, 10, 12$ [s], with snapshots in the first row and evolution of variables in the second row: end-effector trajectory [red]; distances between a control point and the nearest obstacle [green]; end-effector repulsive velocity [blue]. The last row shows the components of the end-effector repulsive velocity [left] and the minimum distances for the other control points [right]

6 Conclusions

We have presented a new general approach to evaluate the distance between a point of interest in the Cartesian space and the objects detected by a depth sensor. Performing all necessary operations in the Depth space allows to obtain distance information with a reduced computational burden, while taking into account the whole frustum generated by the depth information stored in the pixels. We have shown the superiority of the proposed approach both in terms of correctness and performance by comparing it with a state-of-the-art method based on clouds of points in the Cartesian space. The real-time capabilities and the practical effectiveness of the presented approach have been demonstrated using a high dynamically human-robot collision avoidance task.

An open issue is whether and how would it be possible to integrate the information coming from multiple depth sensors. In fact, each depth sensor has its own depth space, and the associated data cannot be directly merged without losing some essential information, e.g., on occluded points.

Acknowledgements Work supported by the European Community, within the FP7 ICT-287513 SAPHARI project.

References

1. Realtime URDF filter. URL http://github.com/blodow/realtime_urdf_filter
2. Cherubini, A., Passama, R., Meline, A., Crosnier, A., Fraisse, P.: Multimodal control for human-robot cooperation. In: Proc. 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2202–2207 (2013)
3. De Luca, A., Flacco, F.: Integrated control for pHRI: Collision avoidance, detection, reaction and collaboration. In: Proc. 4th IEEE Int. Conf. on Biomedical Robotics and Biomechatronics, pp. 288–295 (2012)
4. Flacco, F., De Luca, A., Khatib, O.: Motion control of redundant robots under joint constraints: Saturation in the null space. In: Proc. 2012 IEEE Int. Conf. on Robotics and Automation, pp. 285–292 (2012)
5. Flacco, F., Kroger, T., De Luca, A., Khatib, O.: A depth space approach to human-robot collision avoidance. In: Proc. 2012 IEEE Int. Conf. on Robotics and Automation, pp. 338–345 (2012)
6. Gecks, T., D., H.: Human-robot cooperation: Safe pick-and-place operations. In: Proc. 2005 IEEE Int. Works. on Robot and Human Interactive Communication, pp. 549–554 (2005)
7. Haddadin, S., Belder, S., Albu-Schaeffer, A.: Dynamic motion planning for robots in partially unknown environments. In: Proc. IFAC World Congr., pp. 6842–6850 (2011)
8. Jia, P., Ioan, S., Sachin, C., Dinesh, M.: Real-time collision detection and distance computation on point cloud sensor data. In: Proc. 2013 IEEE Int. Conf. on Robotics and Automation, pp. 3593–3599 (2013)
9. Khatib, O.: Real-time obstacle avoidance for manipulators and mobile robots. Int. J. of Robotics Research **5**(1), 90–98 (1986)
10. Kröger, T.: Opening the door to new sensor-based robot applications — The Reflexxes Motion Libraries. In: Proc. 2011 IEEE Int. Conf. on Robotics and Automation (ICRA Communications). Shanghai, China (2011)
11. Kröger, T., Wahl, F.M.: On-line trajectory generation: Basic concepts for instantaneous reactions to unforeseen events. IEEE Trans. on Robotics **26**(1), 94–111 (2010)
12. Ma, Y., Soatto, S., Kosecka, J., Sastry, S.S.: An Invitation to 3-D Vision: From Images to Geometric Models. Springer Verlag (2003)
13. Meilland, M., Comport, A.: On unifying key-frame and voxel-based dense visual SLAM at large scales. In: Proc. 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 3677–3683 (2013)

14. Mémoli, F., Sapiro, G.: Distance functions and geodesics on point clouds (2003). DOI 10.1.1.154.7728
15. Peasley, B., Birchfield, S.: Real-time obstacle detection and avoidance in the presence of specular surfaces using an active 3D sensor. In: Proc. 2013 IEEE Works. on Robot Vision, pp. 197–202 (2013)
16. Piumsomboon, T., Clark, A., Billinghurst, M.: Physically-based interaction for tabletop augmented reality using a depth-sensing camera for environment mapping. In: Proc. 26th Int. Conf. on Image and Vision Computing New Zealand, pp. 161–166 (2011)
17. Placitelli, A., Gallo, L.: Low-cost augmented reality systems via 3D point cloud sensors. In: Proc. 7th Int. Conf. on Signal-Image Technology and Internet-Based Systems, pp. 188–192 (2011)
18. Rakprayoon, P., Ruchanurucks, M., Coundoul, A.: Kinect-based obstacle detection for manipulator. In: Proc. 2011 IEEE/SICE Int. Symp. on System Integration, pp. 68–73 (2011)
19. Rusu, R.B., Cousins, S.: 3D is here: Point Cloud Library (PCL). In: Proc. 2011 IEEE Int. Conf. on Robotics and Automation (ICRA Communications). Shanghai, China (2011)
20. Ryden, F., Chizeck, H.: A method for constraint-based six degree-of-freedom haptic interaction with streaming point clouds. In: Proc. 2013 IEEE Int. Conf. on Robotics and Automation, pp. 2353–2359 (2013)
21. Saveriano, M., Lee, D.: Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles. In: Proc. 2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 5380–5387 (2013)
22. Schiavi, R., Flacco, F., Bicchi, A.: Integration of active and passive compliance control for safe human-robot coexistence. In: Proc. 2009 IEEE Int. Conf. on Robotics and Automation, pp. 259–264 (2009)
23. Zhang, Z.: Microsoft Kinect sensor and its effect. IEEE MultiMedia **19**(2), 4–10 (2012)