Real-Time Computation of Distance to Dynamic Obstacles With Multiple Depth Sensors

Flacco Fabrizio and Alessandro De Luca

***Abstract*—We present an efficient method to evaluate distances between dynamic obstacles and a number of points of interests (e.g., placed on the links of a robot) when using multiple depth cameras. A depth-space oriented discretization of the Cartesian space is introduced that represents at best the workspace monitored by a depth camera, including occluded points. A depth grid map can be initialized off line from the arrangement of the multiple depth cameras, and its peculiar search characteristics allows fusing on line the information given by the multiple sensors in a very simple and fast way. The real-time performance of the proposed approach is shown by means of collision avoidance experiments where two Kinect sensors monitor a human-robot coexistence task.**

***Index Terms*—RGB-D Perception, Distance Computation, Sensor Fusion, Collision Avoidance, Motion Control of Manipulators, Physical Human-Robot Interaction.**

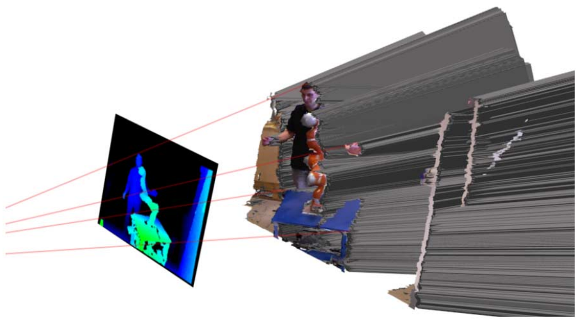


Fig. 1. Illustration of the gray area generated by a single depth sensor (with a human, a robot, and a table in the environment).

I. Introduction

THE most common approach for an artificial system to perceive the real world is by vision [1], one goal being to let a robot see the environment in the same way (e.g., stereo or in motion) as we do. Apart from perception capabilities, humans have also a huge information background that allows recognizing objects and estimating qualitatively spatial information, such as regions of free space or relative distances. Computer vision methods are being combined with machine learning techniques, using large information databases (say, from Google) to recognize objects and reconstructing the environment for different purposes and goals. Unfortunately, with the available computing power of standard robotic systems, most of these approaches are not suitable for hard real-time applications that require fast and reliable detection of dynamic objects, such as in human-robot collision avoidance.

Recently, there has been a spread in the use of depth (RGB-D) camera sensors, like the Microsoft Kinect [2], as a mean to provide 3D information about the environment in a compact form and at low cost. In these devices, each pixel in the 2D sensor image is associated to the shortest distance between the camera and an object point along the projection ray through that pixel, namely a depth information. However, points along the same ray that are behind the objects (i.e., with a greater depth) will remain unobserved, and may belong to the free space or not. At a given time, the collection of all these Cartesian points related to a view by the RGB-D sensor is called *gray area* (Fig. 1).

To reduce the gray area, multiple views of the same scene can be acquired [3]. By combining the depth information from different points of view it is possible to decrease the amount of unobserved space. In some applications one can use a single camera that moves around the scene (as for object reconstruction [4]), while multiple (depth) cameras are simultaneously needed in other cases, in order to monitor a dynamic environment. In general, the latter situation includes all applications with moving objects (e.g., robots, humans).

When dealing with multiple dynamic obstacles, a basic requirement in applications using RGB-D sensors is the on-line estimation of distances between the obstacles and some *control points* (or points of interest), where a control point may either belong to a real object (e.g., attached to a robot link) or be a virtual one. Interested applications include: *augmented reality*, where simulated objects have to interact with a real environment [5]; *virtual fixtures* in tele-manipulation, where objects and shapes should generate force feedback to the operator via a haptic device [6]; *collision avoidance* of a robot moving in a dynamic environment cluttered with obstacles [7]; *object recognition*, when a mobile robot has to be distinguished from other moving objects [8]; simultaneous *localization and mapping* (SLAM), where a map of the environment is built and used to localize the pose of a moving camera [9]; and *human-robot collaboration*, when a robot and a human have to share a common workspace, possibly getting in physical contact and exchanging forces [10].

A most desirable characteristic of methods that evaluate time-varying distances is real-time performance. In fact, too slow updates of estimated distances could easily impair the

Manuscript received August 29, 2015; revised December 21, 2015; accepted January 30, 2016. Date of publication February 29, 2016; date of current version April 11, 2016. This paper was recommended for publication by Associate Editor M. Vincze and Editor J. Kosecka upon evaluation of the reviewers' comments. This work was supported by the European Commission, within the H2020 projects FoF-637080 SYMPLEXITY and ICT-645097 COMANOID.

The authors are with the Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, 00185 Roma, Italy (e-mail: fflacco@diag.uniroma1.it; deluca@diag.uniroma1.it).

Digital Object Identifier 10.1109/LRA.2016.2535859

correct behavior of an application, e.g., obstacles will not be avoided or virtual fixtures badly simulated. In this respect, a common misconception is to consider as the upper bound for the rate of distance evaluation simply the camera frame rate (usually, 30 Hz). However, in the case of dynamic agents, control points may continue to move (in a commanded way) during the time interval between two camera frame acquisitions, while scene information is not updated. Thus, actual distances may still change faster than the video rate and their evaluation should be updated as fast as possible.

The actual bottleneck in the distance evaluation process based on multiple depth cameras is not in the distance computation itself, which usually requires few simple equations, but is the on-line merging of data coming from all sensors in a common representation, which allows then computing distances easily. One of the favorite approaches for estimating distances from depth sensor data uses the cloud of points that are obtained by projecting the depth image in the Cartesian space [11], and often relies on the availability of open libraries such as the Point Cloud Library (PCL) [12], possibly speeded up using parallel processing on GPU [13]. While this approach suits the natural reasoning about distances in Cartesian space, it does not fully exploit the information associated to pixels in RGB-D sensors. For instance, occluded points in space, which should be considered part of an obstacle, are not taken into account directly, requiring an extra computational load.

In [14], the point cloud information coming from multiple depth cameras is clustered in objects, which are represented as convex hulls; then, the convex hull representation is used to compute distances between the robot and the obstacles. As an alternative, an octree representation [15] is used in [16]. The main drawback of these methods is the time wasted in order to represent (dynamic) obstacles in the Cartesian space, whereas all information needed for computing distances is already available in the so-called depth space of the sensors. Moreover, the geometry about rays of projection associated to each pixel is lost in this way, an information that can be used instead to speed up the sensor integration process. A different idea is explored in [17], where information merged from multiple Kinect sensors is used to improve the tracking of a human and then to compute distances from the robot to a bubble model representation of the human. Good real-time performance is achieved, but the method does not evaluate distances to obstacles other than the human. For instance, collision with an object carried by the human will not be considered.

We have presented in [7], and more recently improved in [18], a different approach that evaluates point-to-object distances working in the depth space of the sensor. This results in a large improvement of the overall computation time. Moreover, the method allows a correct consideration of the *pixel frustum* in the Cartesian space, i.e., of the portion of a pyramid left after its top part has been cut off by a (skewed) plane. Despite of its merits, a straightforward extension to multiple cameras is not possible because every camera has its own depth space.

In this letter, while inheriting from [7] the idea of using directly the depth space, we propose a new method that allows an efficient data fusion from multiple depth sensors with real-time performance. The main ingredients are: *i)* the introduction of a discretization of the Cartesian space that we call *Depth grid*; *ii)* a method to compute the shortest distance between an occupied cell of the depth grid and a control point; *iii)* an off-line procedure that creates the depth grid map relations between the different cameras; *iv)* a fast on-line method to check whether a cell in the grid is free or not, taking into account the data from all depth cameras. The basic assumptions under which the proposed method is applied are:

1) at least one camera monitors the whole space of interest;
2) the relative pose between all cameras does not change over time.

The letter is organized as follows. In Sec. II, the math that rules a depth space is recalled. The depth grid is introduced in Sec. III, while in Sec. IV we show how to compute the distance between a cell of this grid and a Cartesian point. Sections V and VI present, respectively, the off-line phase needed to build a map based on the depth grid and the on-line phase where the depth grid map is used for computing distances. In Sec. VII, we consider a human-robot collision avoidance problem as a possible application of the presented method. Experimental results using a KUKA LWR-IV robot are reported in Sec. VIII (and in the accompanying video).

II. Depth Space

A RGB-D sensor provides a 2D color image and a depth image. The depth image represents the perceived environment in the depth space, a non-homogeneous 2.5-dimensional space where two elements are the coordinates of the projection of an observed Cartesian point on the sensor plane and the third element is the depth of this point, namely its distance to the sensor plane along a ray. The depth sensor is modeled as a pin-hole camera with two sets of parameters: the intrinsic parameters (the focal length f, the size s_x and s_y of a pixel, the coordinates c_x and c_y of the focal axis in the image plane) model the projection of a Cartesian point on the image plane, while the extrinsic parameters (a rotation matrix $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$) specify the pose of the sensor frame C_s with respect to a reference (world) frame C_r. The representation in the sensor frame of a Cartesian point ${}^{C_r}\boldsymbol{P} = ({}^{C_r}x\ {}^{C_r}y\ {}^{C_r}z)^T$ expressed in the reference frame is

$$ {}^{C_s}\boldsymbol{P} = ({}^{C_s}x\ {}^{C_s}y\ {}^{C_s}z)^T = \boldsymbol{R}\,{}^{C_r}\boldsymbol{P} + \boldsymbol{t}, \tag{1} $$

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

$$ p_x = \frac{{}^{C_s}x f s_x}{{}^{C_s}z} + c_x, \quad p_y = \frac{{}^{C_s}y f s_y}{{}^{C_s}z} + c_y, \quad d_p = {}^{C_s}z. \tag{2} $$

In the reverse direction, a point in the depth space is mapped in the Cartesian (sensor) space as

$$ {}^{C_s}x = \frac{(p_x - c_x)d_p}{f s_x}, \quad {}^{C_s}y = \frac{(p_y - c_y)d_p}{f s_y}, \quad {}^{C_s}z = d_p. \tag{3} $$

Starting from the depth image of the environment, when a point sensed in the depth space is mapped back to the Cartesian space it represents only the closest point of an object to the image

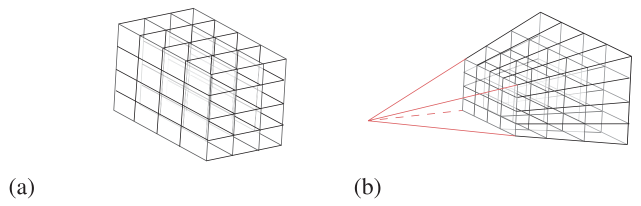


Fig. 2. Workspace discretization with Cartesian grid (a) or Depth grid (b).

plane along the projection ray. However, another simple information is coded in the depth space, namely that all Cartesian points generated by (3) with depth greater than d_p compose the uncertain gray area (see Fig. 1). Without extra information, this gray area should be considered (in a conservative way) as part of the perceived object.

Indeed, the image plane of the depth sensor is discretized in pixels, and each pixel contains a single depth information. For a given point ${}^D\boldsymbol{P} = (p_x\ p_y\ d_p)^T$ in the depth space, only its pixel discretization ${}^D\bar{\boldsymbol{P}} = (\bar{p}_x\ \bar{p}_y\ d_p)^T$ will be considered, with the truncation $\bar{p}_i = \lfloor p_i \rfloor$, $i = x, y$.

III. Depth Grid

Data coming from multiple depth sensors should be merged in a common space, in order to infer if a 3D region is free or not. A natural choice is the Cartesian space, which is common to all cameras, and the most popular approach performs a workspace discretization using a Cartesian grid (also referred to as voxel grid) of cells with fixed size (Fig. 2a). This discretization is the one used, e.g., in [16], [19], where an octree provides a hierarchical way to represent the grid with different resolutions. Then, the information coming from the different depth cameras is fused to distinguish free from occupied cells. Despite its simple structure, the main drawback of this Cartesian grid is that it is not 'pixel oriented', namely the pixel discretization of the depth image is not used. For instance, many pixels may be represented in the same cell, resulting in a loss of resolution. Moreover, the information contained in the depth along the ray of projection of a pixel is not exploited. This is particularly important, because knowledge of the Cartesian region that belongs to the same pixel can speed up the process.

Based on these considerations we introduce the Depth grid, see Fig. 2b. We have assumed that at least one of the depth camera covers the whole space we are interested to monitor (space of interest). We refer to this sensor as the *principal* (or master) camera. All other cameras will monitor in general only a region of the space of interest (and some extra space that will be discarded). Clearly, each camera will contribute only for that region.

The reference frame of the depth grid is set on the frame of the principal camera (we use a prime to denote data in the principal depth space, e.g., ${}^{D'}\boldsymbol{P}$). The shape of the depth grid is ruled by the pin-hole model (2)-(3) of the principal camera. Just as for the depth space, for a cell in the depth grid the first two coordinates $(\ \bar{g}_x\ \bar{g}_y\ ) \in \mathbb{N} \times \mathbb{N}$ represent a pixel in the image plane. Thus, the quantization is intrinsically given by the pixel dimension. The third coordinate $\bar{g}_d \in \mathbb{N}$ is related to a depth d by means of a suitable discretization function $\bar{g}_d = r(d)$. The function $r(d)$ may take into account that the depth sensor has higher resolution for lower depths, varying then the depth quantization linearly with the depth value. For the sake of simplicity, we use here a fixed quantization Δd. Considering that the grid starts from a minimum depth d_{min} (according to the workspace to be monitored), we have

$$\bar{g}_d = r(d) = \left\lfloor \frac{d - d_{min}}{\Delta d} \right\rfloor. \tag{4}$$

The peculiar characteristics of this grid are that: *i)* all cells belonging to the same ray of projection (associated to a pixel of the principal camera) have the same first two coordinates; *ii)* given an object point observed by the principal depth camera ${}^{D'}\bar{\mathbf{O}} = (\bar{o}_x\ \bar{o}_y\ d_o)^T$, the coordinates of the associated cell in the depth grid are straightforwardly $(\bar{g}_x\ \bar{g}_y\ \bar{g}_d) = (\bar{o}_x\ \bar{o}_y\ r(d_o))$.

IV. Evaluation of Point-to-Cell Distance

The second needed ingredient is a method to compute the shortest distance between a generic cell of the depth grid and a control point. For this, we use the approach in [7], [18].

Consider the cell $(\bar{o}_x\ \bar{o}_y\ \bar{o}_d)$, with $\bar{o}_d = r(d_o)$, associated to the object point $\mathbf{O}$ and the point of interest $\boldsymbol{P}$, which is represented in the depth space as ${}^D\boldsymbol{P} = (p_x\ p_y\ d_p)^T$, using eqs. (1) and (2). In order to evaluate the Cartesian distance between the obstacle point $\mathbf{O}$ and the point of interest $\boldsymbol{P}$, we take into account the pixel dimension by considering the edge of the cell $(\bar{o}_x\ \bar{o}_y)$ nearest to $(p_x\ p_y)$:

$$\hat{o}_x = \begin{cases} \bar{o}_x & p_x < \bar{o}_x \\ \bar{o}_x + 1 & p_x > \bar{o}_x + 1 \\ p_x & \text{otherwise,} \end{cases}$$
$$\hat{o}_y = \begin{cases} \bar{o}_y & p_y < \bar{o}_y \\ \bar{o}_y + 1 & p_y > \bar{o}_y + 1 \\ p_y & \text{otherwise.} \end{cases} \tag{5}$$

The depth covered by the cell spans from $d_1 = d_{min} + \bar{o}_d\,\Delta d$ to $d_2 = d_{min} + (\bar{o}_d + 1)\Delta d$. The surface nearest to the point $\boldsymbol{P}$ has depth

$$\hat{o}_d \simeq \begin{cases} d_1 & d_p < d_1 \\ d_2 & d_p > d_2 \\ d_p & \text{otherwise.} \end{cases} \tag{6}$$

Thus, we assume $\hat{o}_d = d_p$ when the depth of the point of interest is between d_1 and d_2. This is not the nearest surface, but the difference is negligible.

Finally, the distance $D(\boldsymbol{P}, \mathbf{O})$ is evaluated as

$$\begin{aligned} v_x &= \frac{(\hat{o}_x - c_x)\,\hat{o}_d - (p_x - c_x)\,d_p}{f s_x} \\ v_y &= \frac{(\hat{o}_y - c_y)\,\hat{o}_d - (p_y - c_y)\,d_p}{f s_y} \\ v_z &= \hat{o}_d - d_p \end{aligned}$$
$$D(\boldsymbol{P}, \mathbf{O}) \simeq D_D({}^D\boldsymbol{P}, {}^D\hat{\mathbf{O}}) = \sqrt{v_x^2 + v_y^2 + v_z^2}. \tag{7}$$

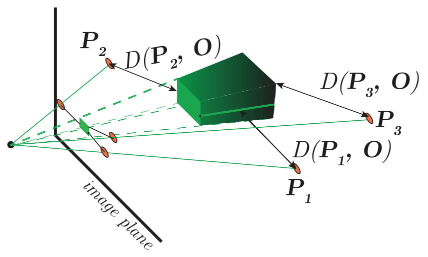
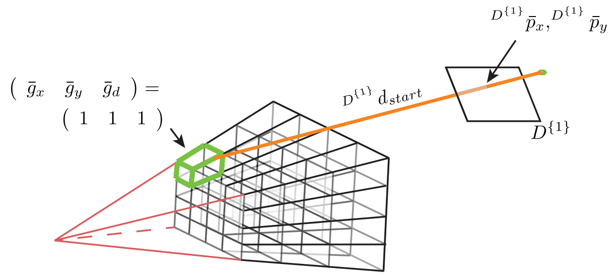


Fig. 3. Depth space distance evaluation from a cell of the depth grid to three points of interest. Three possible cases are shown of points of interest whose depth is within the depth spanned by the cell ($\boldsymbol{P}_1$), before the cell ($\boldsymbol{P}_2$), or beyond the cell ($\boldsymbol{P}_3$). The pixel dimension is taken into account, in order to consider the real frustum associated to the pixel.



${}^{D^{\{1\}}}\bar{p}_x, {}^{D^{\{1\}}}\bar{p}_y$
$(\bar{g}_x \ \bar{g}_y \ \bar{g}_d) =$
$(1 \ 1 \ 1)$
${}^{D^{\{1\}}}d_{start}$
$D^{\{1\}}$

Fig. 4. Illustration of the off-line mapping between cell $(1\ 1\ 1)$ in the depth grid and the additional depth camera $D^{\{1\}}$.

An illustrative example of distances between a cell of the depth grid and three points of interest is shown in Fig. 3.

V. Off-Line Phase: Building the Depth Grid Map

The depth grid is built considering only the principal camera, and maintains its useful properties only for this camera. Having a depth grid for each camera is not a solution for merging information from multiple cameras, since each grid would create a different representation of the Cartesian space. The idea is to use then the depth grid as a *map* between the depth space of the principal camera and the depth spaces of the other cameras. In this map, each cell contains a reference to depth pixels of other cameras that monitor the same cell (i.e., part of the cell is projected in those pixels). In this phase, no specific assumption is made on the presence of static or dynamic obstacles in the environment, and no depth information from the sensors is used. The only needed data are the intrinsic and extrinsic parameters of the cameras.

Assume there are l depth cameras, in addition to the principal one. Each cell of the depth grid map has an associated list of pixels, each pixel being represented by $\left({}^{D^{\{k\}}}\bar{p}_x \ {}^{D^{\{k\}}}\bar{p}_y \ {}^{D^{\{k\}}}d_{start}\right)$, where $\left({}^{D^{\{k\}}}\bar{p}_x, \ {}^{D^{\{k\}}}\bar{p}_y\right)$ are the pixel coordinates in the kth image plane where the cell is projected, with $k \in \{1, \ldots, l\}$, and ${}^{D^{\{k\}}}d_{start}$ is the minimum depth of the cell in the kth camera frame. The latter information is needed to infer whether camera k is seeing an object before the cell, thus if the kth camera is assuming that the cell is part of an object. An illustration of the mapping for a single depth grid cell and an additional depth camera $D^{\{1\}}$ is given in Fig. 4.

When the relative pose of the cameras remains constant (e.g., all cameras are stationary or are mounted on board of the same vehicle), as we have assumed and as it is common in many robotic applications, the depth grid map does not change. Therefore, it can be computed completely off line. One can consider this as a third phase of the calibration process of a monitoring system, determining: *i)* intrinsic parameters; *ii)* extrinsic parameters; and *iii)* depth grid map, i.e., the relation between the depth spaces of the cameras.

Note that, in our framework, the issue about decay of resolution and the associated larger uncertainty at higher depths are not addressed for secondary cameras.

VI. On-Line Phase: Distance Evaluation using the Depth Grid Map

During the execution of a task, we have a depth grid map initialized off line and a point of interest $\boldsymbol{P}$ (or many of them), projected in the depth point of the principal camera ${}^{D'}\boldsymbol{P} = (p_x \ p_y \ d_p)^T$. We would like to evaluate on line the distance of $\boldsymbol{P}$ to the obstacles in the environment, inferring information from all depth cameras. At each time step[1], we use the last available depth image of each camera.

Depending on the application, we may be interested in the whole monitored space or wish to consider only a region close to the point of interest. In the former case, all pixels of the principal depth image have to be evaluated. In the latter, we consider a Cartesian *region of surveillance* $\boldsymbol{S}$, made by a cube of side 2ρ centered in $\boldsymbol{P}$, where the possible presence of obstacles should be detected. The associated region of surveillance in the image plane has its size specified by

$$x_s = \rho \, \frac{f s_x}{d_p - \rho}, \qquad y_s = \rho \, \frac{f s_y}{d_p - \rho}. \tag{8}$$

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

$$\begin{aligned}\boldsymbol{S}_{D'} = &\left[p_x - \frac{x_s}{2}, p_x + \frac{x_s}{2}\right] \times \left[p_y - \frac{y_s}{2}, p_y + \frac{y_s}{2}\right] \\ &\times \ [d_p - \rho, d_p + \rho]\,.\end{aligned} \tag{9}$$

Only those pixels in the principal camera that belong to the region of surveillance (9) will be evaluated. The choice of a specific size for the region of surveillance is indeed a trade off between computational burden and the need of monitoring a larger region, e.g., in order to start a robot reaction earlier.

Consider a pixel $(\bar{s}_x \ \bar{s}_y)$ of interest in the image plane of the principal camera, and let d_o be the measured depth information at this pixel. Different decisions are taken, depending on the relative values of d_0 and d_p, the depth of the control point in the principal camera. For illustration, in Fig. 5 we use a simplified planar representation limited to row $\bar{s}_y$ of the image plane. All cells of the depth grid associated to the pixel $(\bar{s}_x \ \bar{s}_y)$ should be taken into account in the distance evaluation. However, thanks to the peculiar features of the depth grid, this will not be strictly necessary.

[1]Images are updated at a common frame rate for all cameras. As already mentioned, distance to obstacles can and should be computed at a faster rate, in order to consider also the possible motion of the point of interest.

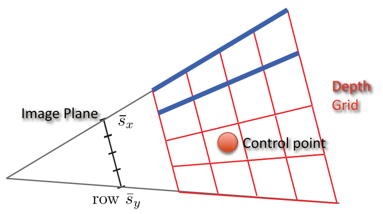


Fig. 5. Illustration of the depth grid associated to row $\bar{s}_y$ of the image plane. Cells associated to the pixel $(\bar{s}_x\ \bar{s}_y)$ are highlighted in bold blue.

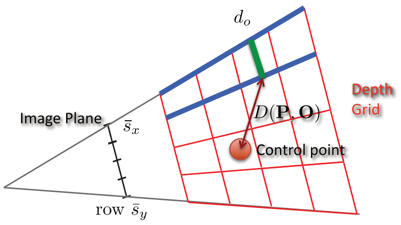


Fig. 6. Distance evaluation when $d_o \geq d_p$.

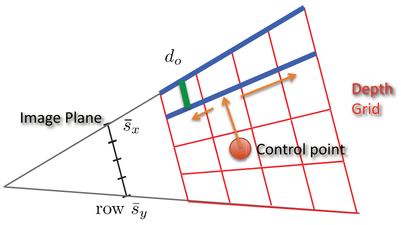


Fig. 7. Distance evaluation when $d_o < d_p$.

The following situations should be considered:

i) If the observed obstacle and the related occluded points are outside the region of surveillance, $d_o > d_p + \rho$, we conclude, without using other cameras, that no obstacle should be considered for these grid cells. The distance evaluation continues to the next pixel.

ii) If the observed obstacle is in the region of surveillance and $d_o \geq d_p$ (Fig. 6), we determine, again without using other cameras, the smallest distance to the obstacle by eqs. (5)–(7), with $(\bar{o}_x\ \bar{o}_y) = (\bar{s}_x\ \bar{s}_y)$ and $\hat{o}_d = d_o$.

iii) Finally, if $d_o < d_p$, information from the other cameras is needed (Fig. 7). First, the nearest cell in the depth grid $(\bar{s}_x\ \bar{s}_y\ r(d_p))$ is considered, and the presence of an obstacle has to be confirmed using the other cameras. If this presence is not confirmed by other cameras, we proceed checking other cells in both directions along the third component (ray direction) in the grid, setting $r(d_p) + j$, $j = 1, \ldots, \left\lfloor \frac{\rho}{\Delta d} \right\rfloor$ and $j = -1, \ldots, \left\lfloor \frac{d_o - d_p}{\Delta d} \right\rfloor$, until the possible confirmation of the presence of an obstacle. If a cell is confirmed to be part of an obstacle, eqs. (5)–(7) are used to compute the distance.

For the core step of confirming the presence of an obstacle in a cell, a very simple procedure is used. Thanks to the information stored in the depth grid map, we know in advance which pixels of the other cameras monitor the current cell. Camera k confirms the presence of an obstacle if the current depth measure ${}^{D\{k\}}d_o$ in the pixel $\left({}^{D\{k\}}\bar{p}_x\ {}^{D\{k\}}\bar{p}_y\right)$ associated to the cell verifies ${}^{D\{k\}}d_o < {}^{D\{k\}}d_{start}$. The existence of an obstacle is considered only if all involved cameras do confirm its presence.

While by using information from multiple sensors one can decrease the amount of gray area, it is still not possible to remove in general the possibility of having occluded points. Thus, a correct positioning of the cameras is crucial [3]. However, we remark that our approach uses only a two-dimensional scan, with (part of) the third dimension being considered only when needed, contrary to the classical case of Cartesian grids, where a three-dimensional scan is always needed[2].

As described in [18], depending on the specific application, one could be interested in finding only the minimum distance, in which case an even smarter scanning process is possible, or in collecting distances generated by all obstacles. The second approach is used in the following application example.

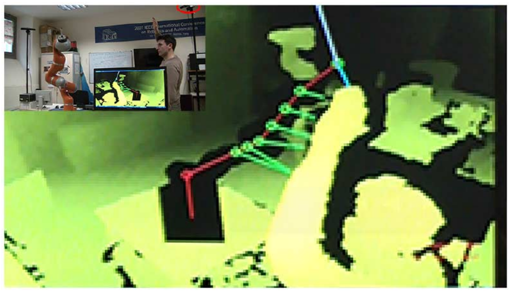


Fig. 8. Small distances to a shadow obstacle are obtained when the human moves between the camera (placed inside the circle in the top left picture) and the robot, even when still far away from it. Evaluated distances from control points on the robot links are represented by robot-to-human lines in the monitor frame. Here and in the following Fig. 9, the larger image is a low-quality magnification of the screen in the smaller image, which is presented to highlight the camera view.

VII. Application Example: Robot Collision Avoidance

Collision avoidance of a robot moving in a dynamic environment is one of the applications in which multiple depth sensors are convenient. The use of our integration method preserves real-time performance, while producing the usual benefits of multiple cameras. In fact, in this application the presence of gray (unobserved) areas typically result in undesired behaviors. Without additional information, occluded points must be considered as part of an obstacle for safety reasons, especially when the robot is sharing its workspace with humans. The drawback of this conservative approach is that the robot avoids collisions also with 'shadows' of obstacles.

Figure 8 shows an example of an extreme situation, with a human moving far from a robot, but between the robot and a single depth camera, in such a way that the generated gray area falls close to the robot. With this partial information, the robot would abandon its task so as to avoid a shadow obstacle which

[2]In the case of methods using octrees, the hierarchical structure of the octree allows to simplify the search for occupied cells, but data insertion in the structure requires (in principle) a complete three-dimensional scan.

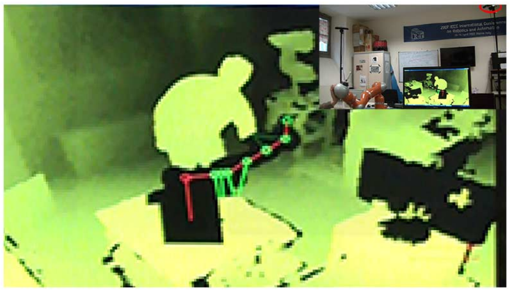


Fig. 9. In this screenshot of an experiment, the human hand is right behind the robot and thus is not detected. As a result, a collision may occur.

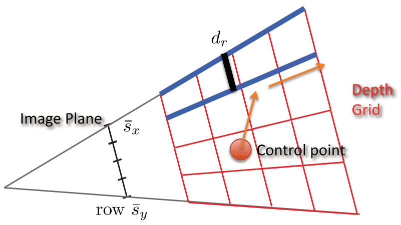


Fig. 10. Distance evaluation when a robot part is present at depth d_r.

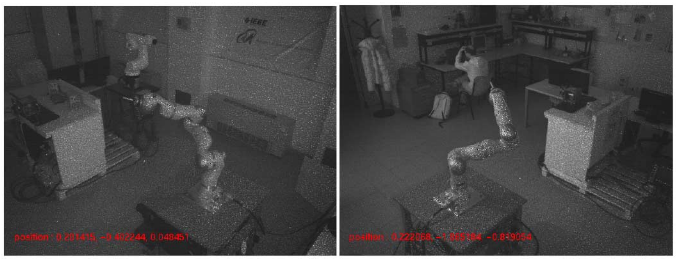


Fig. 11. Points of view of the two depth cameras used in the experiments (infrared images).

is in fact not there. A similar problem is caused by the presence of the robot in the scene. Obviously, the robot has to be removed from the depth image captured by the camera, otherwise it would try to avoid itself just as any other obstacle. As a result, actual obstacles that are behind the robot (from a single camera view) are not visible and will not be avoided (see Fig. 9). Because of the critical real-time aspects of human-robot interaction, the above two major problems were the main motivations for extending our efficient depth-space based approach for collision avoidance to the case of multiple depth sensors.

Since part of the image is being removed (to avoid a fictitious robot self-collision), this should be taken into account during the distance evaluation. With reference to the pixel example in Sec. VI, a fourth case has to be considered:

iv) If the robot has been removed at depth d_r, the only certain information from the principal camera is that there are no obstacles up to a depth d_r. The possible presence of an obstacle behind the robot has to be verified using the other cameras, checking only the grid cells with third coordinates between $r(d_r) + 1$ and $r(d_r) + \lfloor \frac{\rho}{\Delta d} \rfloor$ (Fig. 10).

While in some applications, e.g., in collision checking, retrieving the information on distance to obstacles would be sufficient, for collision avoidance we need to compute also the normalized direction from the control point to the nearest point on the frustum. This unit vector is simply given by

$$\boldsymbol{V}(^D\boldsymbol{P}, ^D\hat{\mathbf{O}}) = \frac{(\, v_x \;\; v_y \;\; v_z \,)^T}{D_D(^D\boldsymbol{P}, ^D\hat{\mathbf{O}})}. \qquad (10)$$

When implementing a reactive control scheme for robot collision avoidance, neither the minimum distance obstacle point alone nor the mean of the distances to all detected obstacles are convenient choices. In fact, using a reaction method based on minimum distance could drive a robot control point toward a second object, which would become then the nearest one pushing thus the robot back toward the first object, with an

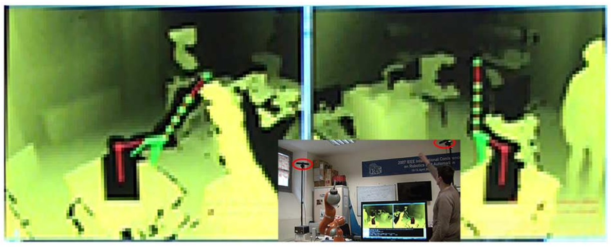


Fig. 12. Thanks to the fusion of information coming from the two depth sensors, the problem of a false detected obstacle when using a single camera shown in Fig. 8 is solved. Here and in the following Fig. 13, the larger image is a low-quality magnification of the screen in the smaller image, which is presented to highlight the camera view.

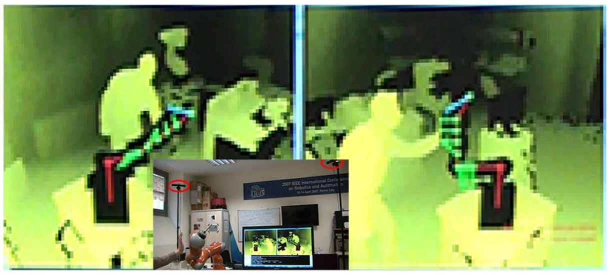


Fig. 13. The second camera allows detecting obstacles that the principal camera is not able to see, avoiding the dangerous situation of Fig. 9.

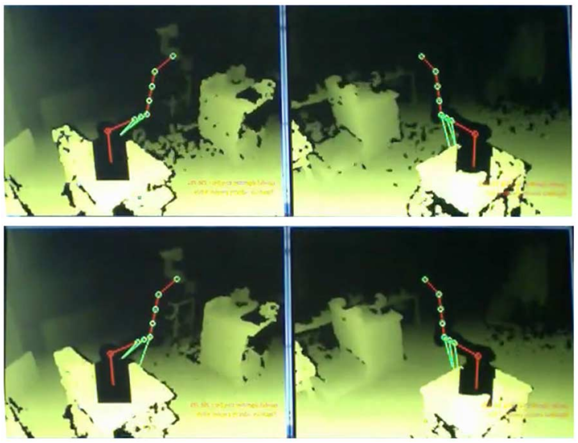


Fig. 14. Screenshots of depth maps of two Kinect cameras monitoring the same space. Due to interference of infrared textures, the depth images display noise effects(in the form of black spots) [top], which are reduced when one of the two sensors is vibrating [bottom].

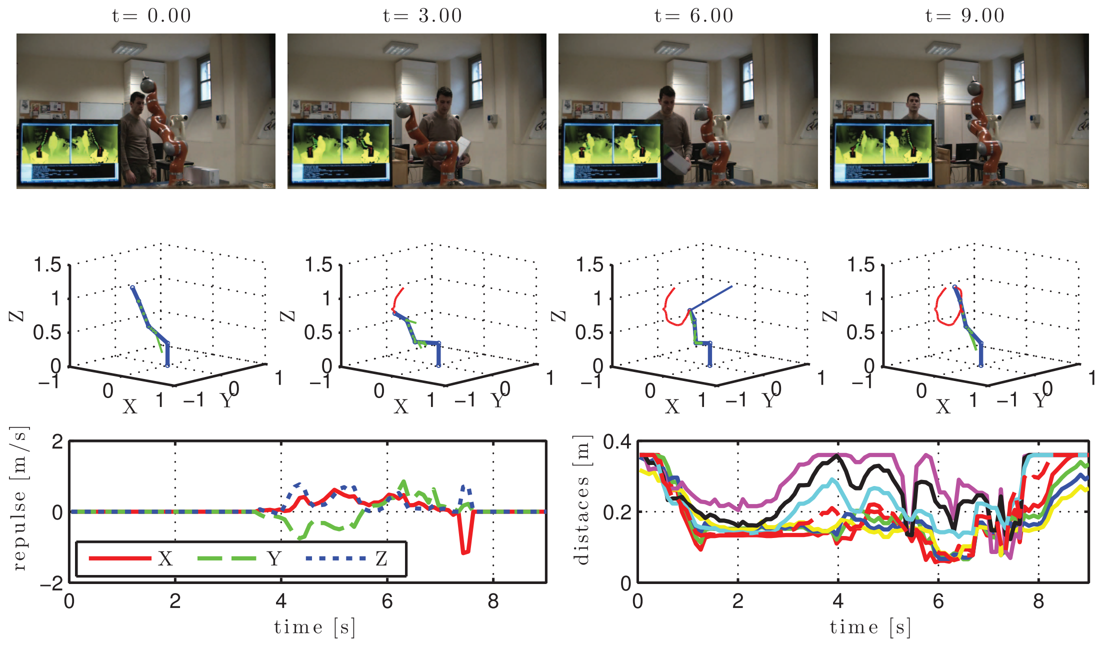


Fig. 15. Collision avoidance experiment. Screenshots of robot and human during motion [top]. End-effector trajectory (in red) and evaluated distances (in blue for the end-effector, in green for the other control points) [middle]. Evolution in time of the repulsive action applied to the end-effector [bottom-left] and of the evaluated distances for the other 8 points on the robot body [bottom-right].

undesirable oscillating effect due to switchings. On the other hand, the mean distance depends on the global topology of obstacles, being affected by the ratio of the numbers of near to far obstacles. This is also undesirable, since the presence of a close object should always provide the same robot reaction, no matter if other obstacles are near or far. Based on this qualitative analysis, we propose to use eqs. (7) and (10) in a hybrid method as in [7] with

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

$$\boldsymbol{V}_{\text{hybrid}}(\boldsymbol{P}) = \boldsymbol{V}_{\text{mean}}(\boldsymbol{P}) = \frac{\sum_{{}^D\hat{\mathbf{O}} \in \boldsymbol{S}_D} \boldsymbol{V}({}^D\boldsymbol{P}, {}^D\hat{\mathbf{O}})}{N}, \quad (11)$$

where N is now the number of object points detected by the multiple depth sensor system inside the surveillance area $\boldsymbol{S}_D$. Use of (11) allows the robot to react to the nearest object in terms of intensity, while considering all objects in the surveillance area for choosing the direction of reaction.

VIII. Experimental Results

To illustrate the practical effectiveness and real-time performance of the proposed method, we have run experiments on a 7R KUKA LWR-IV robot. The LWR is controlled using the Fast Research Interface (FRI), which allows commanding desired joint positions at high frequency rates —500 Hz in our experiments ($T = 2$ ms). All techniques have been developed in C++, on a Intel Core i7-2600 CPU 3.4GHz with 8Gb of RAM.

Two Kinect cameras are used to monitor the robot workspace from different points of view, as shown in Fig. 11, placed at a relative distance of about 4 m. The image plane of each sensor is composed by 640×480 pixels captured at 30 Hz. The robot workspace being monitored is $[-1.5, 1.5] \times [-1.5, 1.5] \times [-0.5, 1.5]$ [m], with origin at the robot base. The principal camera is the Kinect placed on the right, meeting the robot workspace at a distance $d_{min} = 0.2882$ m. With $\Delta d = 0.01$ m as quantization, the resulting depth grid has $640 \times 480 \times 343$ cells.

The performed experiments are fully reported in the accompanying video. We show first how the double sensor arrangement, in combination with the proposed depth space fusion method, solves both problems raised in Sec. VII. Figure 12 illustrates that when an obstacle (the human arm) is placed between the principal camera and the robot being still far from the robot, the second camera does not confirm the presence of an obstacle —all the distance lines to the shadow obstacle that were present in Fig. 8 have disappeared. Similarly, when the principal camera is not able to see an obstacle that is hidden by the robot, the processing of the depth data by the second camera with the proposed method allows to detect the obstacle (Fig. 13).

When multiple Kinect sensors are used to monitor the same space, the structured infrared textures used by each device to infer the depth of the objects overlap, making it harder to distinguish which infrared points belong to the correct texture. This interference produces noise on the depth images, as shown in the top part of Fig 14. To cope with this, we used the method by [20] with a motor producing a small vibration in one of the sensors. In this way, the vibrating Kinect continues to see correctly its own texture (emitter and receiver move together), while the other sensors are no longer affected by the infrared points of the vibrating device that are seen just as unfocused lines (bottom part of Fig. 14).

In the collision avoidance experiment, the robot executes a continuous task by moving its end-effector at a nominal speed of 40 cm/s through the six vertex points of an hexagon in the vertical plane. A human enters in the robot workspace and carries objects around, getting very close to the robot and interfering with its Cartesian or joint trajectories. Collisions are avoided by reactive robot motions, while the end-effector task resumes as soon as it becomes feasible again.

Using the two depth sensors, the proposed method is used to evaluate on line the distances between 9 control points (including the end-effector) placed along the robot body and every static or dynamic obstacle in the workspace. Following our method in [7], the evaluated distance between the robot end-effector and the obstacles is used to generate a repulsive velocity to avoid collision, while distances between the other points of interest and the obstacles are used to generate and impose virtual Cartesian constraints that the robot cannot violate, exploiting also its redundant degrees of freedom. The algorithm is able to aggregate the distance information from multiple obstacle points, without any further assumption (e.g., on the number of obstacles or on human presence). The whole algorithm is performed at around 300 Hz (ten times the current sensor frame rate), a result that is currently beyond the capability of other state-of-the-art methods. Figure 15 shows a portion of this experiment. The high performance achieved with the proposed approach for the safe and long-term coexistence of human and robot can be appreciated better in the accompanying video.

IX. Conclusions and Future Work

A new method for distance evaluations in dynamic environments has been presented that fuses efficiently the information of multiple depth cameras. Excellent real-time performance is obtained by introducing in the monitored workspace a convenient depth grid, which can be initialized off line from the arrangement of the multiple cameras and then easily searched in the on-line phase of distance computation. The method has been successfully tested using two depth sensors in human-robot collision avoidance experiments, where both the robot and the human were moving fast and the overall algorithm was running every 3.3 ms. In view of the obtained linear growth of computations with the number of cameras, more depth sensors can be accommodated to cover complex environments crowded with dynamic obstacles and agents.

Current work aims at removing the two basic assumptions in Sec. I. When the workspace of interest is covered by multiple cameras, and not every camera has part of its field of view in common with a single master camera, an independent depth grid map can be computed off line for each camera as if it were the principal one. In the on-line phase, the current camera playing the principal role is chosen according to the position of the control points, and the related depth grid map is used for fusing sensor information. Considering relative motions between cameras is somewhat harder, and we will work on developing a description of the depth grid map as a function of the position of the moving cameras.

References

[1] K. Daniilidis and J.-O. Eklundh, "3-D vision and recognition," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. New York, NY, USA: Springer, 2008, pp. 543–562.

[2] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE MultiMedia*, vol. 19, no. 2, pp. 4–10, Apr. 2012.

[3] F. Flacco and A. De Luca, "Multiple depth/presence sensors: Integration and optimal placement for human/robot coexistence," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2010, pp. 3916–3923.

[4] S. Izadi *et al.*, "KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera," in *Proc. 24th ACM Symp. User Interface Softw. Technol.*, 2011, pp. 559–568.

[5] T. Piumsomboon, A. Clark, and M. Billinghurst, "Physically-based interaction for tabletop augmented reality using a depth-sensing camera for environment mapping," in *Proc. 26th Image Vis. Comput. New Zealand Conf.*, 2011, pp. 161–166.

[6] F. Ryden and H. Chizeck, "A method for constraint-based six degree-of-freedom haptic interaction with streaming point clouds," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2353–2359.

[7] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach to human-robot collision avoidance," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 338–345.

[8] P. Rakprayoon, M. Ruchanurucks, and A. Coundoul, "Kinect-based obstacle detection for manipulator," in *Proc. IEEE/SICE Int. Symp. Syst. Integr.*, 2011, pp. 68–73.

[9] M. Meilland and A. Comport, "On unifying key-frame and voxel-based dense visual SLAM at large scales," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 3677–3683.

[10] A. Cherubini, R. Passama, A. Meline, A. Crosnier, and P. Fraisse, "Multimodal control for human-robot cooperation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 2202–2207.

[11] P. Jia, S. Ioan, C. Sachin, and M. Dinesh, "Real-time collision detection and distance computation on point cloud sensor data," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 3593–3599.

[12] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA Commun.)*, Shanghai, China, 2011, pp. 1–4.

[13] 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. Robot. Autom.*, 2014, pp. 3250–3257.

[14] M. Fischer and D. Henrich, "Surveillance of robots using multiple colour or depth cameras with distributed processing," in *Proc. 3rd ACM/IEEE Int. Conf. Distrib. Smart Cameras*, 2009, pp. 1–8.

[15] A. Hornung, K. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[16] A. Fetzner, C. Frese, and C. Frey, "A 3D representation of obstacles in the robots reachable area considering occlusions," in *Proc. 41st Int. Symp. Robot.*, 2014, pp. 1–8.

[17] C. Morato, K. N. Kaipa, B. Zhao, and S. K. Gupta, "Toward safe human robot collaboration by using multiple kinects based real-time human tracking," *J. Comput. Inf. Sci. Eng.*, vol. 14, no. 1, pp. 011006-1–011006-9, 2014, doi: 10.1115/1.4025810.

[18] F. Flacco, T. Kröger, A. De Luca, and O. Khatib, "A depth space approach for evaluating distance to objects," *J. Intell. Robot. Syst.*, vol. 80, pp. 7–22, 2015.

[19] A. Hornung, M. Phillips, E. Jones, M. Bennewitz, M. Likhachev, and S. Chitta, "Navigation in three-dimensional cluttered environments for mobile manipulation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 423–429.

[20] A. Maimone and H. Fuchs, "Reducing interference between multiple structured light depth sensors using motion," in *Proc. IEEE Virtual Reality (VRW)*, 2012, pp. 51–54.