Hindawi
Journal of Sensors
Volume 2017, Article ID 4853915, 16 pages
https://doi.org/10.1155/2017/4853915

Research Article

Projection Surfaces Detection and Image Correction for Mobile Robots in HRI

Enrique Fernández-Rodicio, Víctor González-Pacheco, José Carlos Castillo, Álvaro Castro-González, María Malfaz, and Miguel A. Salichs

Departamento de Sistemas y Automática, Universidad Carlos III de Madrid, 28911 Madrid, Spain

Correspondence should be addressed to José Carlos Castillo; jocastil@ing.uc3m.es

Received 1 March 2017; Revised 22 May 2017; Accepted 4 June 2017; Published 12 July 2017

Academic Editor: Eugenio Martinelli

Projectors have become a widespread tool to share information in Human-Robot Interaction with large groups of people in a comfortable way. Finding a suitable vertical surface becomes a problem when the projector changes positions when a mobile robot is looking for suitable surfaces to project. Two problems must be addressed to achieve a correct undistorted image: (i) finding the biggest suitable surface free from obstacles and (ii) adapting the output image to correct the distortion due to the angle between the robot and a nonorthogonal surface. We propose a RANSAC-based method that detects a vertical plane inside a point cloud. Then, inside this plane, we apply a rectangle-fitting algorithm over the region in which the projector can work. Finally, the algorithm checks the surface looking for imperfections and occlusions and transforms the original image using a homography matrix to display it over the area detected. The proposed solution can detect projection areas in real-time using a single Kinect camera, which makes it suitable for applications where a robot interacts with other people in unknown environments. Our Projection Surfaces Detector and the Image Correction module allow a mobile robot to find the right surface and display images without deformation, improving its ability to interact with people.

1. Introduction

An interesting possibility of enhancing Human-Robot Interaction (HRI) by offering information to the user is to embark a video projector in a mobile robot. These devices possess the advantage of providing a big surface to display multimedia content, making viewing the information more comfortable for the user. Therefore, mobile robotic platforms can take advantage of these capabilities to generate augmented reality environments that move along with the robot and users. However, projectors were traditionally developed to be static and use a known, fixed surface to project the image. Thus, if we want to deploy this technology in a mobile robot, two issues arise derived from projecting on unknown surfaces: (i) the main challenge is the need to place the robot in front of a planar surface big enough to fit the projected content. This is an important issue, especially in real environments where those surfaces might not be always suitable for projecting; and (ii) another significant problem is that following the usual way of projecting, the robot needs to be placed exactly on an axis perpendicular to the selected surface in order to avoid the Keystone effect [1]: the deformation of the image is caused by trying to project it onto a surface at an angle.

This paper addresses these two challenges to tackle the task of projecting on unknown surfaces correcting deformations. We will use a Kinect RGB-D camera and RANSAC to find proper projection surfaces in real-time. In order to deal with the second problem, our method will correct the image to counteract the Keystone effect finding the homography matrix that relates the image to be projected with the projection area. Figure 1 offers a first example of what problem our approach is intended to solve.

This manuscript is structured as follows: a review of techniques for planar surface detection, Projection Mapping, and Image Correction is offered in Section 2. Next, the main phases of the proposal are detailed in Section 3. After the approach is presented, it is evaluated, presenting and discussing the results in Section 4. In Section 5, the integration

(a) (b)

Figure 1: Projecting an image without correction (a) and with correction (b).

in a real hardware platform together with a real application in which the mobile robot acts as a guide is shown. Finally, the conclusions drawn in this work are summarized in Section 6.

2. Related Work

Projection Mapping (PM) is a technology that aims to use irregular-sharped objects as surfaces for video or image projection. Broadly speaking, many approaches follow the two same steps to solve this problem: detecting planar surfaces and correcting the image so it is displayed undistorted in the selected surface.

The bibliography offers interesting approaches in PM. For example, Li et al. [2] developed an algorithm for displaying images over a hand-held sphere that can be freely moved. The sphere is tracked using infrared (IR) markers, an RGB camera, and the IR camera embedded in a Nintendo Wiimote. Chen et al. [3] proposed a PM system able to project RGB light patterns enhanced for 3D scenes, using a high-frame-rated vision system. Sueishi et al. [4] presented a tracking method for dynamic PM (over moving objects) that does not need markers on the object. Low-intensity episcopic light is projected over the object, and the light reflected from the retroreflective background is used by high-speed cameras for tracking. Okumura et al. [5] proposed a PM method for moving objects using a mirror-based device in order to get a high-speed optical axis controller and a tracking algorithm based on HSV color detection.

In our proposal, we perform static PM using a single RGB-D camera mounted on a mobile robot. It can work in any environment without any previous knowledge, and, since we are working with 3D information, our approach is robust against changes in the illumination, although our method cannot use moving objects as a projection surface.

2.1. Detection of Planar Surfaces. Planar surfaces detection in three-dimensional space is an active field of research due to the spread of low-cost depth sensors. One of the main challenges of the field is to achieve real-time detection, since many of the current algorithms for planar segmentation require heavy computation.

Random Sample Consensus (RANSAC) [6] is a well-known iterative method that, given a set of points, finds the geometric figure that contains the highest number of points. Considering 3D information, RANSAC can be used to find the biggest plane that fits better to the input data. The main advantage of RANSAC, and the main reason to use it, is that it gives a very robust estimation of the model parameters at the cost of a high computational cost. Awwad et al. [7] proposed a method for extracting planar surfaces from a point cloud based on a variation of RANSAC called Seq-NV-RANSAC that checks the normal vector between the point clouds and the plane that RANSAC is testing. Matulic et al. [8] developed an ubiquitous projection method to create an immersive interactive environment using RANSAC to extract planar surfaces. Mufti et al. [9] used time information to estimate planar surfaces and overcome the low resolution and the measurement errors of infrared time-of-flight cameras for autonomous vehicle navigation coupled with RANSAC to find the planes.

Another well-known technique for feature extraction is the Hough Transform that tries to find instances of objects with a specific shape by a voting procedure [10]. The classical transform was conceived to identify lines in the image but this method was extended to identify arbitrary shapes such as circles or ellipses. The version of the Hough Transform for plane detection is very similar, but using a spherical Hough space, and instead of checking planes for a single point, it uses clusters of points. Following this approach, Okada et al. [11] developed a plane segment finder in real-time using the Hough Transform to extract all the candidates to fit a plane and then fitted those plane segment candidates using distance information to detect partial planes. Qian et al. [12] used a combination of RANSAC and Hough Transform to develop a method for robot navigation in unknown corridors. First, RANSAC is applied to a downsampled version of the point cloud to find the different planes that define the walls and the floor, and then Hough Transform is used to find the walls boundaries. Hulik et al. [13] developed an optimization of the 3D Hough Transform for plane extraction in point cloud data. They tested this approach using data extracted from a mobile robot and simulations.

Although RANSAC and Hough Transform are the two most widespread methods for surfaces segmentation, there are other approaches in the literature. For example, Haines and Calway [14] use a machine learning approach to perform plane detection. This work uses Markov random fields to segment the image into planer regions with their corresponding orientations the image is segmented into planar regions. Jun et al. [15] proposed a global correlation method to find

the ground plane using *v-disparity* images for vehicle on-road and off-road navigation. Hemmat et al. [16] developed a method for real-time plane segmentation detecting 3D edges and their intersections.

In our approach, we decided to use the version of RANSAC included within the Point Cloud Library (RANSAC in PCL: http://pointclouds.org/documentation/tutorials/random_sample_consensus.php) (PCL) because it provides a robust estimation of the model we are trying to find in different conditions. However, due to the computational requirements of RANSAC and the size of the point cloud captured by the Kinect, we performed a downsampling of the data before performing the actual detection.

2.2. Image Correction. Keystone correction, or Keystoning, is a group of techniques that allow correcting the deformation of an image projected at an angle without aligning the projector and the screen. There are two different types of correction depending on the orientation, vertical Keystoning, and horizontal Keystoning.

Sukthankar and Mullin [17] developed a camera-assisted portable projection system that automatically corrects the image distortion using an uncalibrated camera to calculate the geometry of the projected image. Kim et al. [18] presented an algorithm for Keystone correction using a single camera that computes the homography from a pattern, determining a scaled rigid body transform. Gacem et al. [19] developed system aimed at locating objects inside warehouse-like environments using a robotic arm equipped with a pico projector and 8 IR cameras. The distortion is corrected by estimating the corners of the projection surface and then calculating the homography between biggest rectangle inside the projection area and the original image.

We use a similar approach to [19] to correct the image, although our system uses a single camera. Because of this, we can only find those rectangles that fall inside the region of the projection surface captured by the camera.

3. Materials and Methods

This section presents the details of the proposed approach as well as the robotic platform in which the system is integrated and tested. We propose a method that allows a mobile robot, equipped with a 3D camera and a projector, to find suitable projection surfaces on unknown environments. Figure 2 shows the main steps of our proposal, which is roughly distributed into two phases:

(i) Projection Surfaces Detector (PSD), which takes 3D point clouds from the camera and finds the biggest rectangle inscribed inside the biggest planar surface in the cloud. This rectangle will be the projection area. The operation modules of this phase are detailed in Section 3.2.

(ii) Keystoning, which takes the information from PSD and warps the image that is going to be projected so it fits inside the rectangle found. This process is detailed in Section 3.3.

Prior to that, there is also a necessary calibration stage to match the projector position and orientation with respect to the camera coordinate system. This is valid as long as the relative positions of the camera and the projector remain unchanged.

It is worth defining the concepts of *projector workspace* and *corrected workspace*, depicted in Figure 3, to ease the understanding of the following sections. The former is the maximum surface the projector can use from a determined position while the latter corresponds to the aim of this work, that is, a rectangular area to display the corrected image. The *corrected workspace* must be contained in the intersection area of the *projector workspace* and a suitable surface within the camera field of view. A calibration stage aims to find the vectors that characterize the *projector workspace* (green arrows in Figure 3). Each vector is defined by two points, where the origin point is the center of the projector lens.

3.1. Calibration: Characterizing the Projector Workspace. Our calibration method projects a predefined image to allow finding the center of a projected image in the camera frame. Figure 4 depicts the calibration process. This process is necessary to establish a first correspondence between the camera field of view and the projector workspace, that is, to match the surfaces detected in the camera point cloud to the projection areas. This process just needs to be executed once unless the relative positions between projector and camera change.

3.1.1. Finding the Projector Central Axis. The first step is to *project the calibration image*, a full-black image with a white circle in the middle since this high contrast combination eases the detection. In the calibration process, the image is projected on a hand-picked planar surface. This is a planar surface with enough space to display the calibration image. Since the calibration process is meant to be executed just once, we manually picked this surface. Therefore, the initial surface was selected by placing the robot facing a wall. The Kinect *acquires a single RGB-D image* (Figure 5(a)) and the process for finding the center of the calibration image starts following the steps depicted in Figure 4(b). First, a threshold is applied to *binarize the acquired image* (Figure 5(b)), which is filtered afterwards using an *opening* morphological operator that discards the small bright regions that still may appear in the binarized image (Figure 5(c)). The threshold value has been experimentally set to 50 since in our setup the contrast between the circle and the background was high. Nevertheless, this value could be adjusted to work under other conditions.

After isolating the circle within the calibration image, the next step is to apply a *Canny edge detector* [20] to find the contour of the circle as shown in Figure 5(d). Once the contour is extracted, the algorithm computes the spatial moments of the circle using the OpenCV implementation of Green's Theorem [21], which represents a weighted average of the image pixels intensities allowing *extracting the centroid* in the image. In our case, the centroid of the image and the center of the circle correspond to the same point (see Figure 5(e)). The 3D position of the centroid is found using the registered depth information extracted from the Kinect.

To reduce the impact of noise, this whole process is repeated 5 times placing the robot in different positions.

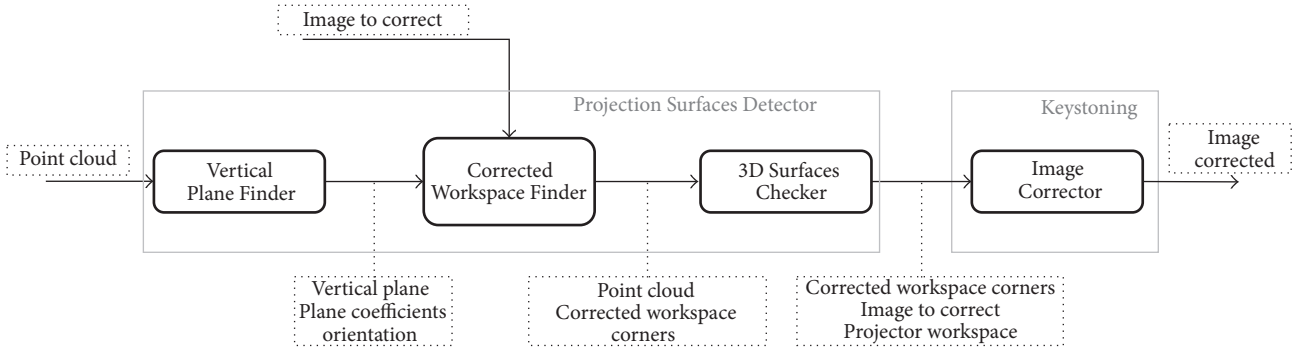


FIGURE 2: Description of the proposal with the main steps. The three first black boxes correspond to the modules of the PSD and the last box performs the Image Correction (Keystoning) phase. Dotted boxes correspond to the message exchange between module as well as the system inputs and output.

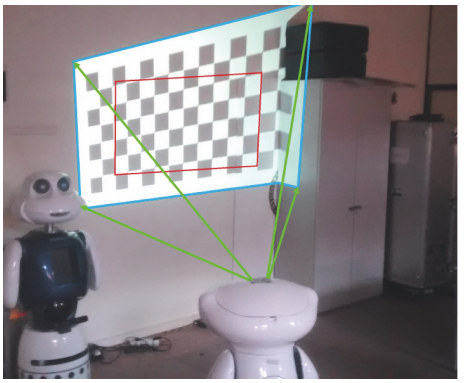


FIGURE 3: *Projector workspace* (blue lines) and *corrected workspace* (red lines). Green arrows correspond to the calibration vectors defining the *projector workspace*.

Once all the centroid points are found, we apply least squares optimization to find the straight line that fits the set of points found. This gives the central axis of the projector, which is used to find the projector workspace.

3.1.2. Finding the Corners of the Projector Workspace. Once the central axis of the projector is calculated, it is possible to obtain the projector workspace following the process shown in Figure 4(c). For this purpose, we need to apply a series of geometric transformations to find the vectors that intersect with the corners of the projector workspace. Knowing the central axis, and the horizontal and vertical angles of the projector field of view, in our case 39.3° and 24°, respectively (these values have been calculated from the values in the manufacturer's specifications), we can find the vectors that go from the center of the lens to the corners of the projector workspace (the projector field of view forms a tetrahedron, as shown in Figure 3).

Those vectors are calculated by applying two rotations around two rotation axes that we need to find. Our approach follows a procedure that derives from the traditional transformation matrices for rotation and translation (the translation is required so the origin of the central axis of the projector falls over the origin of the camera coordinate system). Equation (1) shows a translation matrix in a 3D space, where $t = [t_x, t_y, t_z]$ is the translation applied and (2) shows a rotation matrix around a given axis in a 3D space, where θ is the rotation applied and $u = [u_x, u_y, u_z]$ is the axis used to apply the rotation.

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1)$$

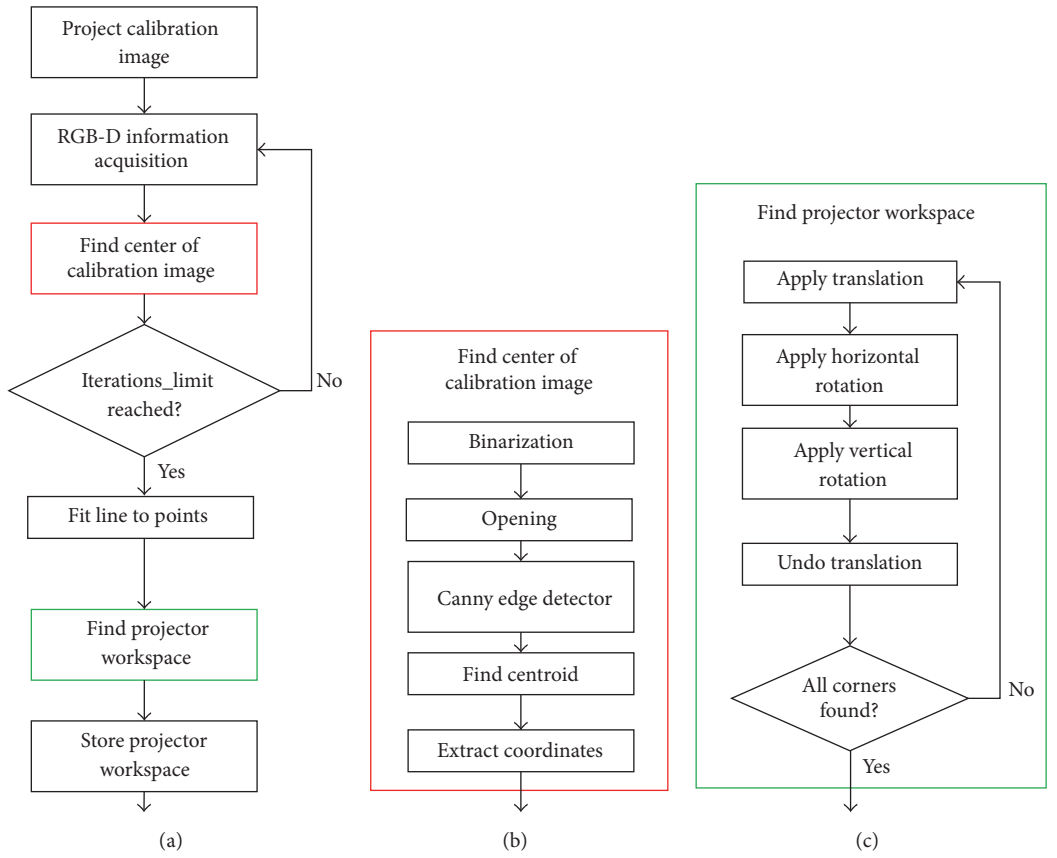


FIGURE 4: (a) Flow diagram for the calibration algorithm to find the projector workspace. Red and green blocks in the main algorithm match the subdiagrams on the center and right side of the image. (b) Process to find the centroid of the calibration image. (c) Process to find the projector workspace.

$$R = \begin{bmatrix} \cos(\theta) + u_x^2(1-\cos(\theta)) & u_x u_y(1-\cos(\theta) - u_z\sin(\theta)) & u_x u_z(1-\cos(\theta) + u_y\sin(\theta)) & 0 \\ u_y u_x(1-\cos(\theta) + u_z\sin(\theta)) & \cos(\theta) + u_y^2(1-\cos(\theta)) & u_y u_z(1-\cos(\theta) - u_z\sin(\theta)) & 0 \\ u_z u_x(1-\cos(\theta) - u_y\sin(\theta)) & u_z u_y(1-\cos(\theta) + u_x\sin(\theta)) & \cos(\theta) + u_z^2(1-\cos(\theta)) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

In order to simplify the implementation of the spatial transformation needed to find the corners of the projector workspace, we decided to use the Euler-Rodrigues formula [22]. It describes the rotation of a vector in three dimensions using the four parameters shown in (3). In this formula, $[a, b, c, d]$ are the Euler parameters, θ is the rotation angle, and $[k_x, k_y, k_z]$ are the components of the rotation axis.

$$\begin{aligned} a &= \cos(\theta), \\ b &= k_x \sin(\theta), \\ c &= k_y \sin(\theta), \\ d &= k_z \sin(\theta). \end{aligned} \quad (3)$$

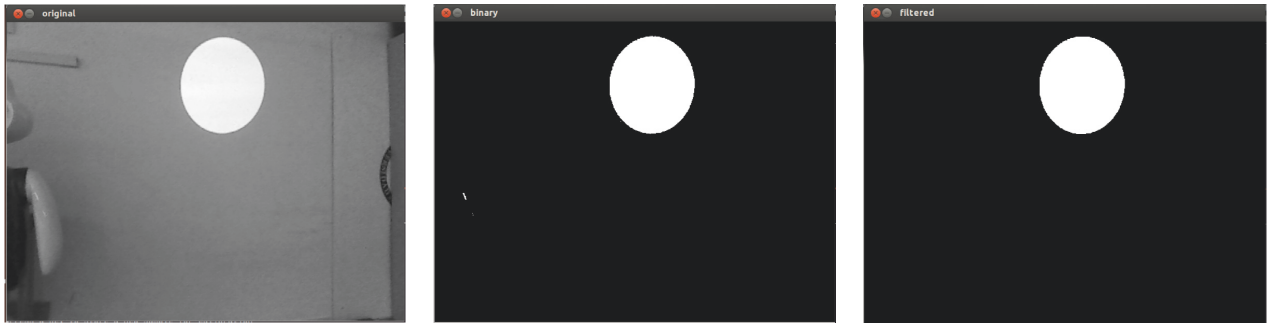
Using these four parameters, the transformation matrix can be replaced by the vectorial formulation shown in (4), where P is the vector being rotated, P_r is the rotated vector, a is the first Euler parameter, and $\vec{\omega} = [b, c, d]$.

$$\overrightarrow{P_r} = \vec{P} + 2a\left(\vec{\omega} \times \vec{P}\right) + 2\left(\vec{\omega} \times \left(\vec{\omega} \times \vec{P}\right)\right). \quad (4)$$

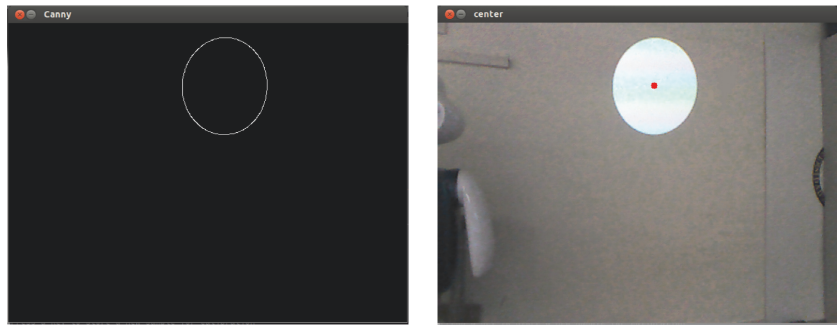
The process to find one of the vectors that intersect with a corner of the projector workspace is depicted in Figure 6, where $\vec{v}$ is the projector central axis, and can be summarized in four steps (the complete procedure along with the mathematical formulation can be found in the Appendix). The steps of this process match the numbers in the figure and also correspond to the main steps shown in Figure 4(c).

(1) *Apply a translation* to $\vec{v}$ so the origin of this segment falls over the camera origin of coordinates, having $\vec{v}'$.

(2) The second step is to *apply a horizontal rotation* α to $\vec{v}'$ using (4) to get $\vec{v}''$. This rotation is equal to half the



(a) Calibration image as acquired by the Kinect (b) Binarization (c) Opening



(d) Canny edges (e) Centroid of the circle

FIGURE 5: Visual representation of the process to find the center of the calibration image (see Figure 4(b)).

projector horizontal field of view. The rotation axis is the one perpendicular to $\vec{v}'$ in the Y-Z plane.

(3) A *vertical rotation* is applied following a vector perpendicular to $\vec{v}''$ in the X-Z plane. In this case, a rotation β is applied to $\vec{v}''$ equal to half the projector vertical field of view using again (4) to get $\vec{v}'''$.

(4) Finally, the *initial translation is inverted*, giving as a result $\vec{v}''''$, the vector that has its origin in the lens of the projector and intersects the corner of the projector workspace.

3.2. Projection Surfaces Detection. After finding the correspondence between the projector and the camera, the process of detecting suitable surfaces for projection can be implemented. The Projection Surfaces Detection component receives a point cloud from the Kinect camera and finds the biggest rectangular area (corrected workspace) within the vertical plane contained on it. The Projection Surfaces Detector is divided into three different modules, Vertical Plane Finder, Corrected Workspace Finder, and 3D Surface Checker, as shown in Figure 2.

3.2.1. Vertical Plane Finder. Using point clouds as input (see Figure 7(a)), this module finds the biggest plane inside parallel to the vertical axis, that is, the wall or surface we are going to use for projection. Despite the inherent noise of a point cloud, to ensure that the surface is at 90° with respect to the ground plane, a maximum deviation of 0.1 radians is established.

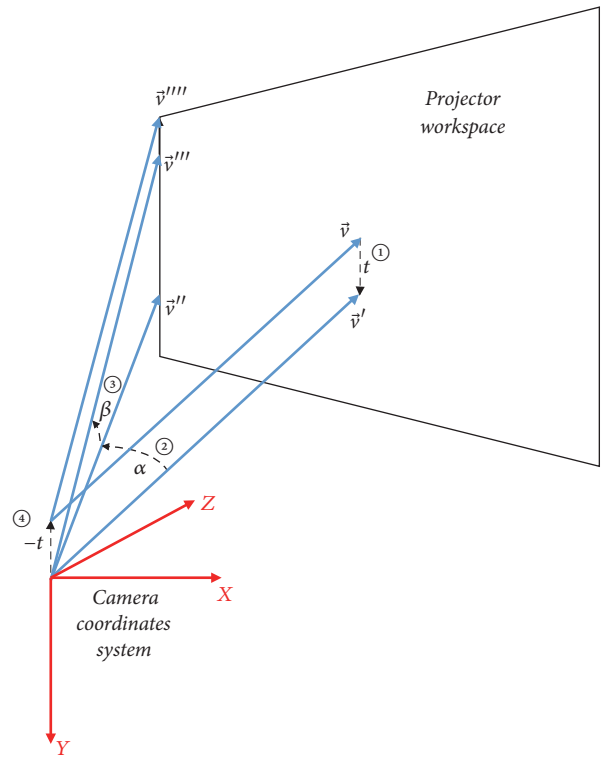
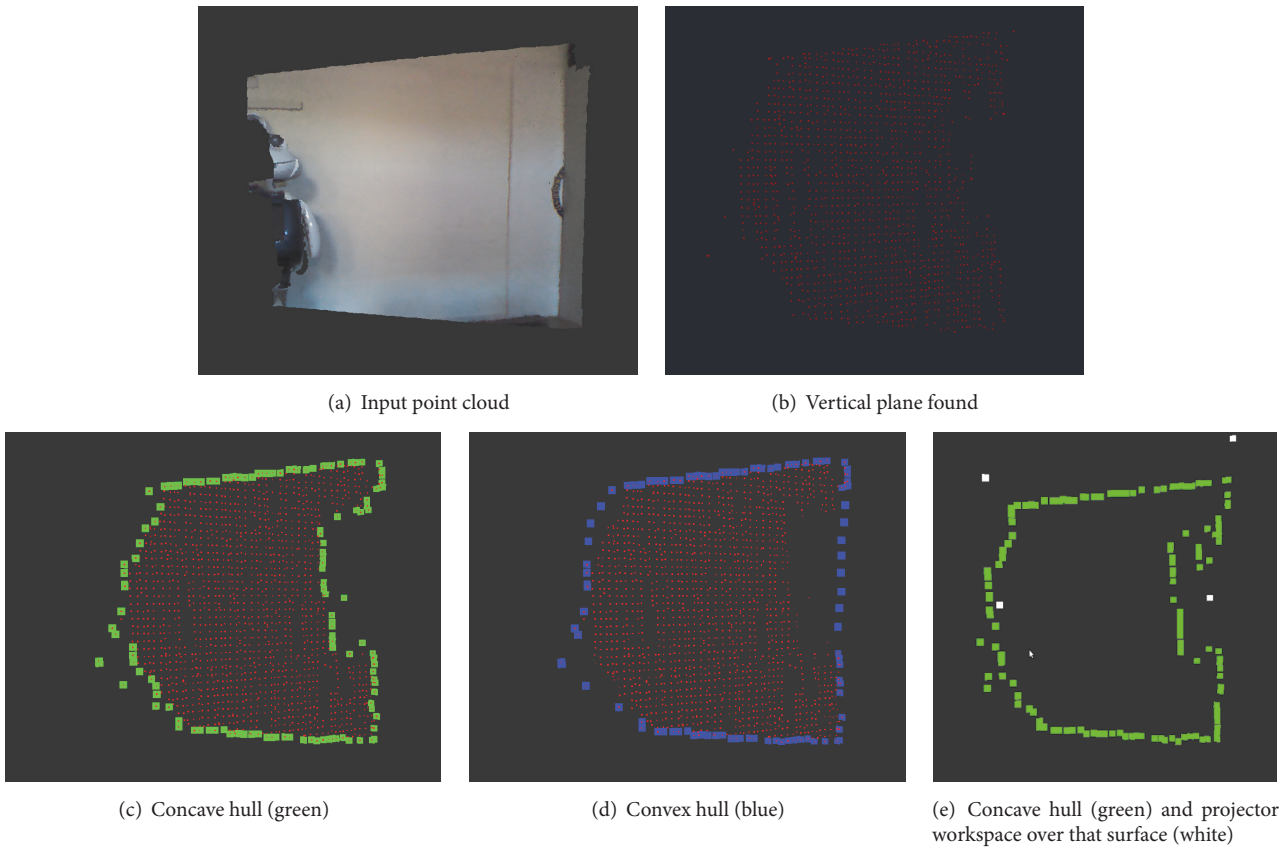


FIGURE 6: Main steps for the spatial transformation to find the vector that intersects with a corner of the projector workspace. The numbers in the figure correspond to the steps of the process as well as the description in the Appendix.



(a) Input point cloud

(b) Vertical plane found

(c) Concave hull (green)

(d) Convex hull (blue)

(e) Concave hull (green) and projector workspace over that surface (white)

Figure 7: Projection Surfaces Detection, from the initial point cloud to the projector workspace.

The Vertical Plane Filter consists of three steps. First of all, the point cloud is downsampled to reduce the computational load. A voxel grid filter is in charge of this task, discretizing the space in a 3D grid and representing all measures contained within 0.05 meters of side cubes as their centroid. Next, RANSAC is applied on the resulting point cloud to extract the biggest vertical plane.

Once the plane is detected, the last step is to calculate its orientation with respect to the vertical axis. If the plane model is defined by the equation $ax+by+cz+d = 0$, then the normal to that plane is $\vec{n} = [a, b, c]$. The orientation of the plane related to the robot is computed as the angle between the normal to the plane and the z-axis of the camera. The Vertical Plane Finder returns the found plane, the coefficients a, b, c, d of that plane model, and the orientation (see Figure 7(b)).

3.2.2. Corrected Workspace Finder. This module receives the vertical plane found in the previous step, calculates the workspace of the projector, and finds the biggest projection area. The aspect ratio of this area has to match the one of the images to be projected, so this input information is also considered.

The first step is to extract the boundaries of the vertical plane. For doing this, two approaches are considered. On the one hand, we compute the convex hull, the smallest convex set that contains all the points in the cloud (see Figure 7(d)). Convex hull is faster to find and less sensitive to noise in the point cloud, because only the outside contour is found. On the other hand, the concave hull, defined as the accurate envelope of a set of points in the plane (see Figure 7(c)), was also tested. This second approach is slower and can even detect internal contours produced by noise or occlusions. Both methods were implemented using Qhull library (Qhull library website: http://www.qhull.org/).

The next step is to find the projector workspace on the vertical plane. For this purpose, the algorithm uses those vectors found during the calibration stage and calculates the intersection between those lines and the vertical plane. There is a plane defined by a normal vector $\vec{n}$ and a point A, and a line defined by its parametric equation, $P(t)$, shown in

$$P(t) = P_0(t) + t\left(P_1(t) - P_0(t)\right), \tag{5}$$

where P_0 and P_1 are two points of the line and t defines the distance from a given point of the line to P_0. This distance is related to the length of the segments P_0, P_1. For example, if a point C is in the middle point of the segment, t would have a value of 0.5 because the distance between C and P_0 is half the length of the segment. So, the value for t at the intersection point between them can be computed as shown in

$$t_{\text{int}} = \frac{\vec{n}\left(A - P_0\right)}{\vec{n}\left(P_1 - P_0\right)}. \tag{6}$$

Once we find the value of t, we can find the coordinates of the corner of the projector workspace just by introducing

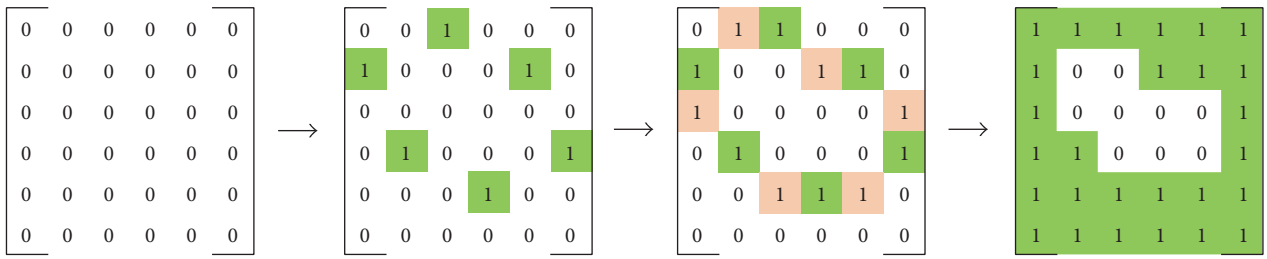


Figure 8: Example of mapping the vertical plane in a 2D matrix: the second matrix contains the contour points of the vertical plane, the third image shows the results of applying Bresenham's algorithm, and the last one highlights the space outside the surface.

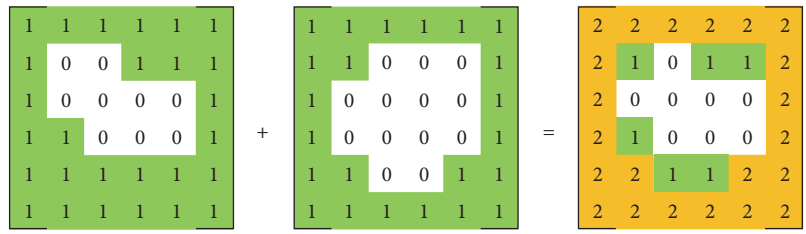


Figure 9: Obtaining the final projectable space by adding the matrices result of discretizing the projector workspace and the contour of the vertical plane.

this value in (5). The result of this operation can be seen in Figure 7(e).

After obtaining the projector workspace and the contour of the vertical plane detected by the Vertical Plane Finder, we can finally aim to find the corrected workspace. Figure 8 shows a simplified example of the main steps of this process. The first step is to transform the point cloud so both planes fall in the XY plane of the camera coordinate system. Therefore, all the points have the same Z coordinate and we can map our 3D plane into a 2D matrix. Next the algorithm creates two empty 2D matrices, $M1$ and $M2$ of the same size, with the maximum values of the rows and columns of the plane found by the Vertical Plane Finder and the projector workspace.

For each point p in the contour of the plane, we use its coordinates as indices and set the corresponding cell of the matrix $M1$ to 1. This means that if p has coordinates $[12, 20]$, then the cell $M1(12, 20)$ is set to 1. We do the same with the other matrix, $M2$, and the corners of the projector workspace. An example of this step is shown in the second matrix of Figure 8.

In the next step, the cells corresponding to the segments between the contour points are set to 1. In order to do this, we apply Bresenham's line algorithm [23] that allows finding the set of cells of a matrix that better fits to a line that connects two random cells of that same grid. The third matrix in Figure 8 depicts an example of the result obtained with Bresenham's algorithm. The cells in green are the initial and final points and the cells in yellow are the ones that connect those points.

Once the borders are found, the next step is setting to 1 all the cells that fall outside the area defined by them. We check both matrices row by row, knowing that the first cell of each row can not be inside the contour. This is repeated for all rows in the matrix, obtaining the result shown in the matrix on the right in Figure 8.

At this point, only the projectable cells in each matrix will have a value of 0, being free cells, while all the cells belonging to nonprojectable zones have a value of 1, being nonfree cells. The next step is to add the matrix that contains the discretized projector workspace, $M1$, and the matrix that contains the discretized contour of the vertical plane, $M2$. Then, the result of the addition is a matrix where cells with value 0 represent the space over which we can project, $M3$ (see Figure 9).

The last step in the Projection Surfaces Detection module is finding the biggest corrected workspace inside our free space, that is, the set of free cells of our matrix. We selected a rectangle-fitting algorithm proposed by Vandevoorde [24] that retrieves the biggest rectangle within $M3$. For instance, in the example of Figure 9, this algorithm returns the rectangle starting in position (2, 2) and finishing in (4, 3).

3.2.3. 3D Surface Checker. The candidate surface for projection area has to be checked for undetected occlusions or irregular areas. The method takes all the points of the original point cloud that fall inside the corrected workspace and checks if all those points belong to the same vertical plane. Finally, if the vertical plane does not contain enough points, the corrected workspace is rejected.

The operation of this method follows the scheme in Figure 10. First, we rotate the point cloud so the region where the screen is located is perpendicular to the camera z-axis. Then, the cloud is filtered in order to erase any point outside the projection area limits. Finally, for each point, we calculate the quadratic error between its z coordinate and the z mean value of all the points inside the projection area. If the error is

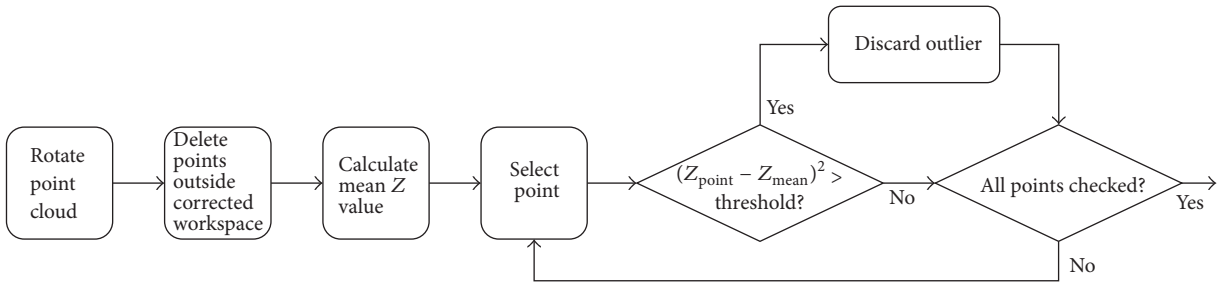


Figure 10: Flow diagram for the Surface Checker module. The method discards those points in the original point cloud that are beyond the projection.

bigger than a threshold, that point is considered an outlier. A previous study demonstrated that the random error of depth measurements increases with the distance and reaches 4 cm at the maximum range of the Kinect device, 5 meters [25]. Additionally, during our testing stage, we observed that a conservative value of 5 cm offered good performance, so that is the reason why we established a threshold of 0.05^2. If the number of outliers is bigger than a given value, then the surface is rejected.

Choosing a suitable value for the maximum number of outliers depends mainly on the environment the system is going to be used in. For instance, in uncluttered environment this limit could be raised to reduce the impact of the camera noise. To establish this value, in Section 4.3 we developed an experiment to measure the number of outliers detected by our system with different obstacle configurations. We will discuss the best value for this limit in that section.

If the algorithm does not detect any occlusions or irregular zones inside the projection area, the Projection Surfaces Detector returns the projection area found.

3.3. Correcting the Keystoning. Once the Projection Surfaces Detector has found a suitable flat surface, we need to transform the image so it falls within the corrected workspace, counteracting the Keystone effect. Therefore, this module takes the information of the corrected workspace, finds the homography matrix between the corrected and uncorrected image, and uses this matrix to transform the image to be projected.

The correction process can be divided into three main steps. First, we have to find the closer pixels of the uncorrected image that would be projected over the corners of the corrected workspace if the image is displayed without any transformation. For each corner of the corrected workspace, we compute the vertical plane that is perpendicular to the projector z-axis and contains that corner. The plane is defined with a normal vector to the surface and a point that belongs to the plane. The point of the plane corresponds to the corner of the projected workspace and the normal vector is the negative projector central axis, $-z$. Then, we calculate the projector workspace over this plane with the same method used in Section 3.2.2. Since the plane is perpendicular to the projector axis, the workspace will not suffer the Keystone effect and will be a perfect rectangle. Finally, coordinates of the pixel are found as shown in (7) where pixel(i) and pixel(j) are the coordinates of the pixel we are looking for, x_{corner} and y_{corner} are the coordinates of the corner of the corrected workspace in the camera coordinate system, x_{max}, y_{max}, x_{min}, and y_{min} are the coordinates that define the projector workspace over the new plane $((x_{min}, y_{max}), (x_{max}, y_{max}), (x_{max}, y_{min}), (x_{min}, y_{min}))$, and $h\_resolution$ and $\vec{v}\_resolution$ correspond to the resolution of the image we want to correct. This process is repeated for all the corners of the corrected workspace.

$$\begin{aligned} \text{pixel}(i) &= \frac{x_{corner} - x_{min}}{x_{max} - x_{min}} * h\_resolution, \\ \text{pixel}(j) &= \frac{y_{corner} - y_{min}}{y_{max} - y_{min}} * v\_resolution. \end{aligned} \tag{7}$$

Next, using these pixels and the corners of the undistorted image $((0,0), (0, h\_resolution), (v\_resolution, h\_resolution), (v\_resolution, 0))$, we used a function provided by the *OpenCV* to calculate the homography matrix and then used it to adapt the image we want to project and store it in a new image. The result of correcting an image is shown in Figure 1.

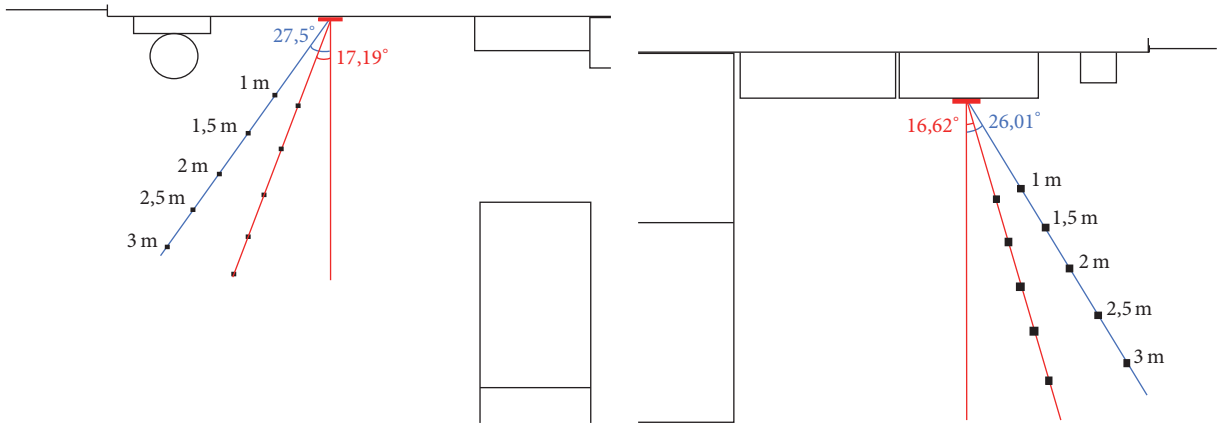
4. Results and Discussion

The performance of the proposal has been evaluated in different stages corresponding to the main components of the system. The first test aims at comparing the two methods for detecting the hull of the vertical plane, *convex hull* and *concave hull* presented in Section 3.2.1. The next experiment helps assessing the performance of the overall Projector Surfaces Detector including the hull method with better performance. The performance of the 3D Surfaces Checker (described in Section 3.2.3) is also tested and finally the performance of the Keystoning module is also evaluated.

4.1. Comparison between Convex and Concave Hull for Surface Detection. In this experiment, the robot was manually placed at different distances and angles with respect to a hand-picked surface as shown in Figure 11(a). The distances ranged from 1 to 3 meters with the robot facing the wall at angles of 27.5° and 17,19° with respect to the one perpendicular to the surface (wall) plane (see Figure 12(a)). For each robot pose, 10 detection rounds were performed, making a total of 100 samples for each method for detecting the hull.

(a) (b)

Figure 11: Surfaces used for the experiments. (a) corresponds to a simple scenario (uncluttered); (b) corresponds to a more complex scenario (cluttered).



(a) (b)

Figure 12: Diagram of the setups for the experiments. (a) Layout of the uncluttered scenario; (b) Layout of the cluttered scenario.

Regarding computational load, we found that there were no significant differences between angles. On the other hand, time grows almost lineally with distance in both concave and convex hulls with the same time, 1.29 s for both options at a distance of 1 meter and 1.46 s at the maximum distance, 3 meters.

The accuracy of the detection was also tested. The criteria used for classifying a result as a success or not is the following. The 10 values for the area of the surface detected were extracted, and then, the maximum value was found. If the difference between this area and the one checked is bigger than 0.05 m^2, that detection is classified as a failed one. It is important to remark here that failed cases are not just cases where the robot is unable to find a projection area over a surface where we know that has to be one, but also cases where the robot finds a projection area, but not the biggest one. Some of the failed detection rounds could still be acceptable projection areas, but because they are not the best areas possible, we decided to reject those cases. We also measured the mean error as the difference between the biggest area overall and the other areas detected.

In general terms, this experiment offered close results for both methods, which are summarized in Table 1. The Projection Surfaces Detector with the *concave hull* method performed better at an angle of 27.5°, obtaining an 84% mean success rate, against the 76% obtained when using the *convex hull*. On the other hand, for the measures taken at an angle of 17.19°, convex hull method performed better, obtaining mean value of 76% against the 66% scored by the *concave hull*. Convex hull method proved to be more regular in the size of the projection areas found, while concave hull is more affected by the orientation of the surface. Nevertheless, the surfaces given by the concave hull are more accurate; therefore this approach will be the one used in the next experiments.

4.2. Analysis of the Performance of the Projection Surfaces Detector. Once the comparison between the two methods for extracting the contour of the point cloud is established, this experiment is meant to give a better insight about the performance of the Projection Surfaces Detector. Apart from the surface used in the previous experiment, a more complex one has been added (see Figures 11(b) and 12(b)) consisting of two different vertical planes corresponding to a big closet and the wall behind it. With these experiments, we wanted to test how the complexity of the surface, the relative orientation between the robot and the surface, and the distance between the robot and the wall affect the computation time and the accuracy of the detection to find a projection area in a point cloud. Again, 100 samples were acquired for each scenario, 10 per position and angle.

Results show how again the CPU time grows with the distance in both scenarios although there are differences, being

Table 1: Detections performed successfully computing the concave and convex hull.

	Distance (m)	1	1,5	2	2,5	3	Average
Concave hull							
27,5°	Success rate (%)	100	100	80	90	50	84
	Mean area error (m^2)	0	0	0,05	0,07	0,073	0,07
17,19°	Success rate (%)	90	100	50	40	50	66
	Mean area error (m^2)	0,05	0	0,07	0,07	0,7	0,15
Convex hull							
27,5°	Success rate (%)	70	100	100	50	60	76
	Mean area error (m^2)	0,17	0	0	0,08	0,12	0,07
17,19°	Success rate (%)	100	90	60	80	50	76
	Mean area error (m^2)	0	0,06	0,06	0,08	0,12	0,06

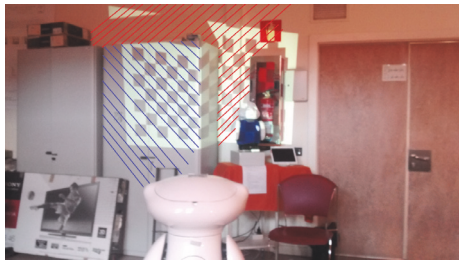


Figure 13: Surfaces detected in the cluttered scenario.

the cluttered scenario the one being more computationally expensive as expected (1,6 s at 3 meters versus 1,55 for the uncluttered scenario) due to the higher number of candidates to projection surface. Apart from the computation time, we measured the accuracy of the Projection Surfaces Detection procedure in the scenarios (see Table 2). The uncluttered scenario results correspond to the ones obtained in Section 4.1 for the concave hull. The accuracy for the tests in the cluttered scenarios is lower, with a high variability in the detection of the candidate surfaces for the corrected workspace.

As shown in Figure 13, the Projection Surfaces Detector starts switching detection rounds between two surfaces when the distance goes beyond 2 meters, causing a drop in performance. When the robot is far enough, the wall behind the closet, the red surface in the figure, becomes the biggest vertical plane, so the closet is considered just as an obstacle. But because of the occlusion generated by the closet and the error introduced by the camera, there is going to be times when the biggest surface detected with RANSAC will be the wall and other times when it would be the closet (the blue surface). This depends on which plane contains more points of the Kinect point cloud. These cases where always marked as failures, because if the biggest surface is the wall, then it should be detected always, regardless of the input data received by the system.

4.3. Analysis of the Performance of the Surface Checker. The performance of the Surface Checker, presented in Section 3.2.3, is also tested. This module takes the original point cloud generated by the camera and removes all the points that are not contained inside the projection area detected. Then, it finds those points inside the corrected workspace that are too far from the rest and marks them as outliers. If too many outliers are detected, the surface is rejected. To test this module, we used the same scenario as in Section 4.1, but with three variants depicted in Figure 14 surface with no obstacles (NO), surface with a small obstacle (SO), and surface with a big obstacle (BO). We used small boxes as small obstacles and a big cardboard box as the big obstacle. Then, for different distances to the wall, we measured the number of outliers detected in each situation.

The experimental setup of this test was the same as in Section 4.1, but repeating the process for the three variants, taking a total of 300 samples, 100 for each case. The maximum distance between a point and the mean used to categorize a point as outlier was 0.05 m, so the threshold used was 0.05^2. Under these conditions, results show that obstacles have a great impact on the density of outliers (see Figure 15). On the other hand, a clear relationship between the number of outliers and the angle for NO and SO situations was not found. Conversely, this aspect seemed to matter in the BO case, although it might be caused by the specific obstacle placed in the scenario.

Another conclusion we extract from this experiment is that distance affects in a different way the outlier detection, depending on the presence of obstacles. The Kinect camera introduces some noise in the measures taken for the points in the point cloud, and this error grows with the distance. This can be the reason why the number of outliers grows with the distance when there is not any obstacle. In the other situations, this noise is still present, but it does not affect the number of false positives detected because those were already outliers, as they belong to the surface of the obstacle. Since the number of outliers is similar for all the distances but the size of the corrected workspace grows, the density of outliers is smaller at bigger distances.

Finally, the analysis of the results shows that the highest median value for the number of outliers when there were not obstacles was 4285, while the smallest median value when there were obstacles, regardless of its size, is 7383 outliers. This allowed us to establish that 5000 outliers could be an acceptable value for the outlier limit. This is closer to the

Table 2: Successful detections for the uncluttered and cluttered scenarios.

	Distance (m)	1	1,5	2	2,5	3	Average
Uncluttered scenario							
27,5°	Success rate (%)	100	100	80	90	50	84
	Mean area error (m^2)	0	0	0,07	0,07	0,07	0,04
17,19°	Success rate (%)	90	100	50	40	50	66
	Mean area error (m^2)	0,05	0	0,07	0,07	0,7	0,17
Cluttered scenario							
−26,01°	Success rate (%)	100	100	60	40	10	62
	Mean area error (m^2)	0	0	0,25	0,27	0,38	0,18
−16,62°	Success rate (%)	100	70	20	10	10	42
	Mean area error (m^2)	0	0,06	0,35	0,36	0,35	0,23

(a) (b) (c)

Figure 14: Scenario to test the 3D Surface Checker. (a) No obstacles (NO) scenario; (b) small obstacles (SO) scenario; and (c) big obstacles (BO) scenario.

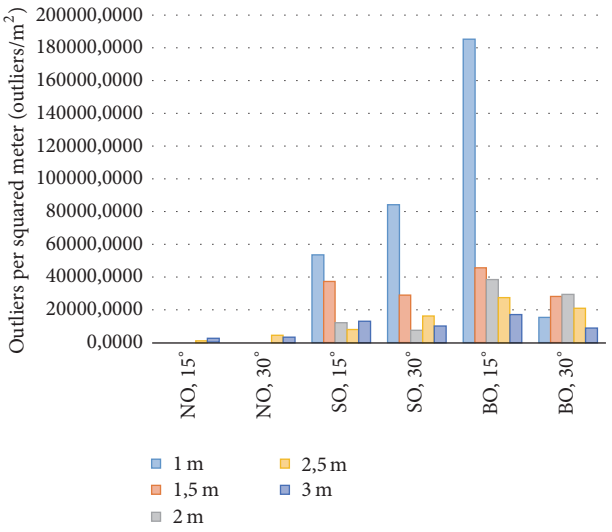


Figure 15: Number of outliers detected for the no obstacles (NO), small obstacles (SO), and big obstacles (BO) scenarios. Those points beyond a threshold of 5 cm with respect to the mean z value of the corrected workspace are considered as outliers.

maximum value obtained for the NO case; this could lead our system to reject a valid surface, but this is preferable over accepting a surface which is nonsuitable for projecting over it.

4.4. Analysis of the Performance of the Image Correction. This last experiment assesses the performance of the Image Correction module, thus completing the testing of the performance of the overall approach. In this case, the scenario is the same as in Section 4.1 but in this case we add one more angle to the original ones, testing now different distances at 27.5°, 17.19°, and −12.06°. In this set of tests, we chose 1-, 2-, and 3-meter distances combined with the three angles, taking a total of 540 measurements, 90 for each segment of the corrected image (top, bottom, left, and right sides, and both diagonals). For this experiment, we defined three errors to test the performance of our Image Correction described in

$$\begin{aligned}
\text{horizontal_error} &= \left(\frac{|top_side - bottom_side|}{\min(top_side, bottom_side)} \right) * 100, \\
\text{vertical_error} &= \left(\frac{|left_side - right_side|}{\min(left_side, right_side)} \right) * 100, \\
\text{diagonal_error} &= \left(\frac{|diagonal_1 - diagonal_2|}{\max(diagonal_2, diagonal_1)} \right) * 100,
\end{aligned} \tag{8}$$

where *top_side*, *bottom_side*, *left_side*, *right_side*, *diagonal_1*, and *diagonal_2* are the six segments characterizing our corrected image. In a perfect rectangle, both diagonals should have the same size, as should be the top and bottom sides and the left and right sides, so the difference between them is an interesting metric to evaluate the correction.

Table 3: Errors found in the projection of the corrected image.

	Distance (m)	1	2	3
27,5°	Horizontal error (%)	1,8 (0,2)	1,14 (0,34)	1,36 (0,5)
	Vertical error (%)	1,77 (1,18)	1,15 (0,69)	0,84 (0,95)
	Diagonals error (%)	1,68 (0,71)	**3,2 (0,51)**	2,95 (0,5)
17,19°	Horizontal error (%)	0,89 (0,54)	0,75 (0,29)	1,47 (0,6)
	Vertical error (%)	0,71 (0,67)	**1,81 (0,61)**	1,58 (1,24)
	Diagonals error (%)	1,55 (0,56)	1,2 (0,32)	1,49 (0,67)
−12.06°	Horizontal error (%)	2,24 (0,93)	2,36 (0,8)	1,41 (0,94)
	Vertical error (%)	0,67 (0,78)	2,27 (1,33)	**3,46 (0,8)**
	Diagonals error (%)	1,3 (0,62)	1,11 (0,46)	0,79 (0,26)

We decided to calculate each error referring to the smaller dimension because that would give us the worst case scenario. The results obtained are summarized in Table 3.

The worst result observed is an error of 3.46% at a distance of 3 m. Just for notice, for that case, the smaller dimension was 1.127 m, so we have an error of 3.38 cm, which is a small difference in comparison. In the end, we can observe that there still exists a little distortion of the image, but not enough to make the user uncomfortable when trying to read the information inside the image. As for the effect of the angle, we appreciate that although there seems to be a direct relation between the angle and the error for the diagonals, this relation does not appear to exist for the other two errors, so we can not infer that the angle has an observable effect over the error.

4.5. Discussion. The success rate of the detectors was lower than expected, but as it was explained, this was caused because we chose conservative criteria, being many of the failed cases usable as projection areas. The Projection Surfaces Detector is more accurate when working with a simple surface, that is, when the vertical plane we are looking for is the main element in the point cloud, while it gives a worse performance when there are many elements in the scene. This usually is not too important, because the robot should choose the cleanest surface to project, but it could be a problem if it needs to project over a specific surface, like the door of a closet, for example. This experiment also showed that our system works best when looking for surfaces at a certain range of distances, although this is mainly caused by the sensor used to extract the point cloud.

The overall performance of our method satisfied our initial requirements, as the detection phase is robust and fast enough (around 1.5 seconds for the whole process) for the scenarios where we want to use this skill, although there is still room for improvement through future works.

5. Integration into a Real Robotic Platform

The hardware platform used in this work is a Mbot robot, which was designed as a part of the MOnarCH European project (MOnarCH project website http://monarch-fp7.eu/) for HRI with children [26, 27]. Its height is 1.15 m, so it looks more child friendly. Among its hardware capabilities, the robot is equipped with hardware for navigation (2 Hokuyo 2D laser range finders), perception (Microsoft Kinect, RFID reader), and iteration (touch screen and touch sensors, a pico projector). Two of these hardware elements are directly linked to the current proposal: the Microsoft Kinect camera provides RGB and depth information to detect the surfaces to project; and the Aaxa P300 pico projector allows displaying the calibration images and shows the final result. Regarding the computational capabilities, the robot integrates two CPUs, one of them running our system, that consist of an Intel® Core I7 CPU at 2,5 GHz with 8 GB of RAM. With this hardware, we also studied the computational load of the different modules of the system, getting a peak load of 12% (in single core execution) for the 3D Surface Checker module. This is the most demanding module as it works with the whole point cloud, checking whether or not the points belong to the projection surface. Table 4 offers the details for the different modules.

Table 4: CPU usage of the main system modules assuming single core execution.

Module	CPU load (%)
Vertical Plane Finder	5%
Corrected Workspace Finder	4,33%
Surface Checker	12%
Image Correction	4%

Although the methods proposed in this paper have been programmed to be generic, running in any ROS-based system, in our case both the Projection Surfaces Detector and the Keystoning module were integrated in the Mbot, following the scheme shown in Figure 16. The Projection Surfaces Detector waits until the Mbot Control Module sends a signal to start the process described in Section 3.2. Once the corrected workspace is found, the Keystoning module corrects the image to be projected following the steps described in Section 3.3. If no projection surface is found, the robot displays an error message in the touch screen as a visual feedback to the user and keeps moving around repeating the process of looking for a suitable surface.

After assessing the performance of our system quantitatively, we developed an application that make use of the previous capabilities of our robot for navigation. Therefore, we designed a scenario where the robot serves as a guide

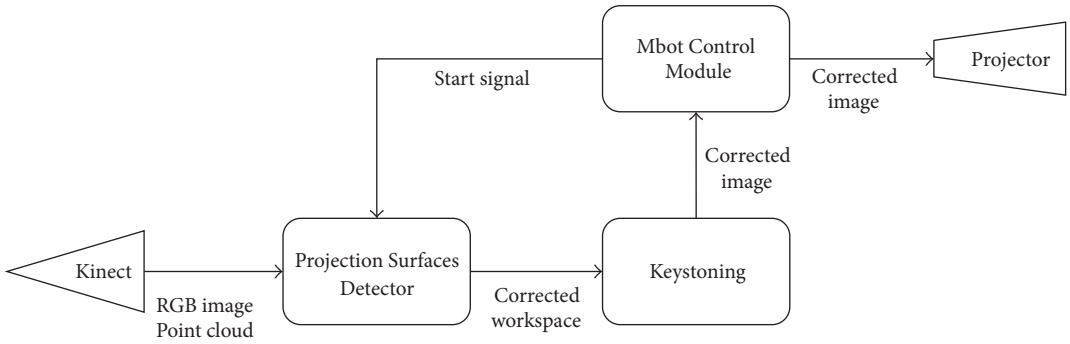


Figure 16: Integration of the proposal in Mbot. Our system sees the robot as an external component (Mbot Control Module) that sends the start signal and waits for the corrected image to project it.

for visitors at the Robotics Lab of the Carlos III University, in Madrid, Spain. The user can choose which office to visit among a set of options displayed on the embarked tablet and the robot guides him/her there. Once it arrives to the destination's office door, it checks for the biggest available projection surface and shows a picture of the person working in that office. The following video shows the application running in the selected scenario: https://youtu.be/huxdOsnf1fE.

6. Conclusions

We have presented a system that allows a robot equipped with a projector and a Kinect to find projection surfaces, adapting the output image to correct the distortion due to the angle between the robot and a nonorthogonal surface. We also proposed a calibration algorithm to find the projector's central axis.

The main contribution of this work is that the method allows a mobile robot to find a surface in the environment that can be used to display multimedia content. Since the robot will be able to use the projector in unknown environments as an interaction tool without human intervention, this allows us to develop new applications that relay on sharing multimedia content with the user. The operation of the system is divided into two phases, namely, Projection Surfaces Detection, that detects suitable planar surfaces using a RGB-D camera, and Image Correction, that adapts the content to be projected eliminating the Keystone effect. For the PSD, different techniques have been compared as in the case of the convex and concave hull in Section 3.2. There were no appreciable differences in terms of computation time and success rate but the concave hull was considered as it provided accurate contours.

Several experiments were carried out in order to assess the performance of the different phases in an incremental way, showing the results of the overall system in Section 4.4. These results have proven the feasibility of the proposal in terms of both computation time and the accuracy of the corrected image. Results confirmed that our system is able to find a suitable surface for projection in a time short enough for HRI applications (around 1.5 seconds) in different environments (although showing some weakness when using it in complex environments). Also, although there is still some distortion of the image being projected, with a maximum error found of 3.46%, this is not enough to hinder the understanding of the information inside the image.

Endowing a mobile robot with Projection Mapping capabilities is of importance to achieve richer HRI as it allows integrating the environment as part of the interaction. In Section 5, we proposed a real application which, using the previous navigation capabilities of our robot, becomes an interactive guide, offering information about the areas to be visited, specifically a picture of the people working in the destination office.

This contribution opens a series of interesting and challenging lines of work such as adapting the algorithms to work on a moving robot or projecting on nonplanar surfaces. Another improvement would be coupling the method described here with multimedia software to allow projecting more complex contents such as videos or videogame-like 3D rendered contents.

Appendix

Detailed Procedure for Finding the Corners of the Projector Workspace

This Appendix contains the details about the procedure for finding the corners of the projector workspace. The four steps enumerated here correspond to the extended version of the ones offered on Section 3.1.2.

(1) First, we need to translate $\vec{v}$ so the origin of this segment falls over the camera origin of coordinates. This is needed because the rest of the operations are applied in the camera coordinate system and are simplified if both origins match. This translation is shown in

$$\vec{v}' = \vec{v} - t, \tag{A.1}$$

where t is equal to the point that belongs to $\vec{v}$ and has value 0 in its z component.

(2) The next step is finding the rotation axis, R_1, that directs the rotation we need to apply over $\vec{v}'$. In this case, R_1 is the vector perpendicular to $\vec{v}'$ in the Y-Z

plane. Because they are perpendicular, we know that the angle γ between $\vec{v}'$ and the Z coordinate axis is the same as the one between R_1 and the Y coordinate axis. This angle is found as depicted in

$$\gamma = \arctan\left(\frac{y_1^t}{z_1^t}\right). \tag{A.2}$$

Once the angle is known, R_1 can be described as

$$R_1 = \left(0, \cos\left(\text{angle}_\gamma\right), \sin\left(\text{angle}_\gamma\right)\right). \tag{A.3}$$

Then, we apply a rotation α to $\vec{v}'$ equal to half the projector horizontal field of view angle. Using (3), the Euler-Rodrigues parameters for this particular rotation are

$$\begin{aligned} a &= \cos(\alpha), \\ b &= R_1^x * \sin(\alpha), \\ c &= R_1^y * \sin(\alpha), \\ d &= R_1^z * \sin(\alpha), \end{aligned} \tag{A.4}$$

with R_1^x, R_1^y, R_1^z being the components of R_1. $\vec{v}''$ can be found using (4):

$$\vec{v}'' = \vec{v}' + 2a\left(\vec{\omega} \times \vec{v}'\right) + 2\left(\omega \times \left(\vec{\omega} \times \vec{v}'\right)\right). \tag{A.5}$$

(3) Now we have to find the second rotation axis, R_2, which has to be perpendicular to $\vec{v}''$ and belongs to the X-Z plane. R_2 is the result of applying (A.5) to the x-axis ($X = [1, 0, 0]$):

$$R_2 = X + 2a\left(\vec{\omega} \times X\right) + 2\left(\vec{\omega} \times \left(\vec{\omega} \times X\right)\right). \tag{A.6}$$

We can now find $\vec{v}'''$ by applying a rotation β to $\vec{v}''$ equal to half the projector vertical field of view. In this case, the Euler-Rodrigues parameters are

$$\begin{aligned} a &= \cos(\beta), \\ b &= R_2^x \sin(\beta), \\ c &= R_2^y * \sin(\beta), \\ d &= R_2^z * \sin(\beta), \end{aligned} \tag{A.7}$$

where R_2^x, R_2^y, and R_2^z are R_2 components. Now, $\vec{v}'''$ is

$$\vec{v}''' = \vec{v}'' + 2a\left(\vec{\omega} \times \vec{v}''\right) + 2\left(\vec{\omega} \times \left(\vec{\omega} \times \vec{v}''\right)\right). \tag{A.8}$$

(4) Finally, we apply a translation with value t, as we did in (A.4), but with positive sign, in order to get $\vec{v}''''$, which is the vector that has its origin in the lens of the projector and intersects the corner of the projector workspace:

$$\vec{v}'''' = \vec{v}''' + t. \tag{A.9}$$

We can find the other 3 vectors just by changing the sign of α and β. Then the algorithm stores this information in a YAML file, ending the calibration stage.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The research leading to these results has received funding from the projects: Development of Social Robots to Help Seniors with Cognitive Impairment (ROBSEN), funded by the Ministerio de Economia y Competitividad, and RoboCity2030-III-CM, funded by Comunidad de Madrid and cofunded by Structural Funds of the EU.

References

[1] A. J. Woods, T. Docherty, and R. Koch, "Image distortions in stereoscopic video systems," in *Proceedings of the IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology, International Society for Optics and Photonics*, pp. 36–48, 1993.

[2] Z. Li, K. Wong, M. Leung, and H. Ko, "A projector-based movable hand-held display system for interactive 3d model exhibition," *Multimedia Systems*, vol. 17, no. 5, pp. 435–447, 2011.

[3] J. Chen, T. Yamamoto, T. Aoyama, T. Takaki, and I. Ishii, "Simultaneous projection mapping using high-frame-rate depth vision," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '14)*, pp. 4506–4511, Hong Kong, China, June 2014.

[4] T. Sueishi, H. Oku, and M. Ishikawa, "Robust high-speed tracking against illumination changes for dynamic projection mapping," in *Proceedings of the IEEE Virtual Reality Conference (VR '15)*, pp. 97–104, Arles, France, March 2015.

[5] K. Okumura, H. Oku, and M. Ishikawa, "Acitve projection AR using high-speed optical axis control and appearance estimation algorithm," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '13)*, pp. 1–6, IEEE, San Jose, Calif, USA, July 2013.

[6] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the Association for Computing Machinery*, vol. 24, no. 6, pp. 381–395, 1981.

[7] T. M. Awwad, Q. Zhu, Z. Du, and Y. Zhang, "An improved segmentation approach for planar surfaces from unstructured 3D point clouds," *Photogrammetric Record*, vol. 25, no. 129, pp. 5–23, 2010.

[8] F. Matulic, W. Büschel, M. Y. Yang et al., "Smart ubiquitous projection: discovering surfaces for the projection of adaptive content," in *Proceedings of the the CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*, pp. 2592–2600, New York, NY, USA, May 2016.

[9] F. Mufti, R. Mahony, and J. Heinzmann, "Robust estimation of planar surfaces using spatio-temporal RANSAC for applications in autonomous vehicle navigation," *Robotics and Autonomous Systems*, vol. 60, no. 1, pp. 16–28, 2012.

[10] R. O. Duda and P. E. Hart, "Use of the Hough transformation to detect lines and curves in pictures," *Communications of the ACM*, vol. 15, no. 1, pp. 11–15, 1972.

[11] K. Okada, S. Kagami, M. Inaba, and H. Inoue, "Plane segment finder: algorithm, implementation and applications," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '01)*, pp. 2120–2125, Seoul, South Korea, May 2001.

[12] K. Qian, Z. Chen, X. Ma, and B. Zhou, "Mobile robot navigation in unknown corridors using line and dense features of point clouds," in *Proceedings of the 41st Annual Conference of the IEEE Industrial Electronics Society (IECON '15)*, pp. 1831–1836, Yokohama, Japan, November 2015.

[13] R. Hulik, M. Spanel, P. Smrz, and Z. Materna, "Continuous plane detection in point-cloud data based on 3D Hough transform," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 86–97, 2014.

[14] O. Haines and A. Calway, "Detecting planes and estimating their orientation from a single image," in *Proceedings of the 23rd British Machine Vision Conference (BMVC '12)*, Guildford, England, September 2012.

[15] Z. Jun, J. Katupitiya, and J. Ward, "Global correlation based ground plane estimation using *v*-disparity image," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA '07)*, pp. 529–534, Roma, Italy, April 2007.

[16] H. J. Hemmat, E. Bondarev, and P. H. N. De With, "Real-time planar segmentation of depth images: from three-dimensional edges to segmented planes," *Journal of Electronic Imaging*, vol. 24, no. 5, Article ID 051008, 2015.

[17] R. Sukthankar and M. D. Mullin, "Automatic keystone correction for camera-assisted presentation interfaces," in *Advances in Multimodal Interfaces—ICMI 2000*, vol. 1948 of *Lecture Notes in Computer Science*, pp. 607–614, Springer, Berlin, Germany, 2000.

[18] J. Kim, Y. Hwang, and B. Choi, "Automatic keystone correction using a single camera," in *Proceedings of the 12th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI '15)*, pp. 576-577, Goyang, South Korea, October 2015.

[19] H. Gacem, G. Bailly, J. Eagan, and E. Lecolinet, "Finding objects faster in dense environments using a projection augmented robotic arm," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9298, pp. 221–238, 2015.

[20] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679–698, 1986.

[21] G. Green, An essay on the application of mathematical analysis to the theories of electricity and magnetism, author, Nottingham, 1828.

[22] H. Cheng and K. C. Gupta, "An historical note on finite rotations," *American Society of Mechanical Engineers. Transactions of the ASME. Journal of Applied Mechanics*, vol. 56, no. 1, pp. 139–145, 1989.

[23] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965.

[24] D. Vandevoorde, "The maximal rectangle problem," 1998, http://www.drdobbs.com/database/the-maximal-rectangle-problem/184410529.

[25] K. Khoshelham and S. O. Elberink, "Accuracy and resolution of kinect depth data for indoor mapping applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, 2012.

[26] J. Messias, R. Ventura, P. Lima et al., "A robotic platform for edutainment activities in a pediatric hospital," in *Proceedings of the IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC '14)*, pp. 193–198, Espinho, Portugal, May 2014.

[27] J. Sequeira, P. Lima, A. Saffiotti, V. Gonzalez-Pacheco, and M. Salichs, "Monarch: multi-robot cognitive systems operating in hospitals," in *Proceedings of the ICRA 2013 Workshop on Many Robot Systems*, Multi-robot cognitive systems operating in hospitals in. ICRA 2013 workshop on many robot systems, 2013.