applied sciences

MDPI

Article

Staircase Detection, Characterization and Approach Pipeline for Search and Rescue Robots

José Armando Sánchez-Rojas [1], José Aníbal Arias-Aguilar [1], Hiroshi Takemura [2] and Alberto Elías Petrilli-Barceló [2,*]

[1] Graduate Studies Division, Technological University of the Mixteca, Km. 2.5 Carretera a Acatlima, Huajuapan de León 69000, Oaxaca, Mexico; asanchez7714@gmail.com (J.A.S.-R.); anibal@mixteco.utm.mx (J.A.A.-A.)
[2] Department of Mechanical Engineering, Faculty of Science and Technology, Tokyo University of Science, 2641 Yamazaki, Noda, Chiba 278-8510, Japan; takemura@rs.tus.ac.jp
* Correspondence: petrilli@rs.tus.ac.jp

Abstract: Currently, most rescue robots are mainly teleoperated and integrate some level of autonomy to reduce the operator's workload, allowing them to focus on the primary mission tasks. One of the main causes of mission failure are human errors and increasing the robot's autonomy can increase the probability of success. For this reason, in this work, a stair detection and characterization pipeline is presented. The pipeline is tested on a differential drive robot using the ROS middleware, YOLOv4-tiny and a region growing based clustering algorithm. The pipeline's staircase detector was implemented using the Neural Compute Engines (NCEs) of the OpenCV AI Kit with Depth (OAK-D) RGB-D camera, which allowed the implementation using the robot's computer without a GPU and, thus, could be implemented in similar robots to increase autonomy. Furthermore, by using this pipeline we were able to implement a Fuzzy controller that allows the robot to align itself, autonomously, with the staircase. Our work can be used in different robots running the ROS middleware and can increase autonomy, allowing the operator to focus on the primary mission tasks. Furthermore, due to the design of the pipeline, it can be used with different types of RGB-D cameras, including those that generate noisy point clouds from low disparity depth images.

Keywords: object detection; point cloud segmentation; ROS; fuzzy control system

Citation: Sánchez-Rojas, J.A.; Arias-Aguilar, J.A.; Takemura, H.; Petrilli-Barceló, A.E. Staircase Detection, Characterization and Approach Pipeline for Search and Rescue Robots. *Appl. Sci.* **2021**, *11*, 10736. https://doi.org/10.3390/app112210736

Academic Editor: Luca Bruzzone

Received: 30 September 2021
Accepted: 11 November 2021
Published: 14 November 2021

1. Introduction

Search and rescue robots allow first responders and emergency professionals to perceive and act at a distance from a disaster site [1]. These robots can be used after natural disasters such as earthquakes, hurricanes, floods and after accidents or terrorism on man-made structures and facilities. Some examples of the tasks that a robot can execute include search of victims, reconnaissance and mapping, rubble removal and structural inspection. To successfully complete these tasks, ground search and rescue robots must be able to navigate in complex environments with surfaces covered in rubble and debris and cross obstacles such as ramps and staircases. These obstacles require a careful inspection by the teleoperator to determine whether or not it can be crossed and the types of maneuvers the robot must execute to safely cross it. Unfortunately, it is often difficult for humans to gain enough information of the robot's surroundings to allow them to teleoperate the robot safely and efficiently and a simple miscalculation of the obstacle's properties can cause the robot to become stuck and result in a failed mission. For this reason, pure teleoperation is often undesirable [2].

Obstacle detection and characterization plays an important role in the robot's autonomy by allowing the implementation of obstacle crossing algorithms and traversability analysis. One of the most common and difficult obstacles in disaster areas include staircases. The difficulty of crossing staircases arises from the fact that a robot has a low contact area

with the outer edges of the staircase and thus slippage can occur, causing abrupt changes in direction, which are difficult to control. For this reason, this work focuses on implementing a staircase detection, characterization and approach pipeline that would allow the teleoperator to gain useful information about the size and orientation of the staircase and also allow the robot to align itself with the staircase. This information obtained can also help determine whether or not the staircase is traversable or could be used to implement an autonomous staircase crossing algorithm. The implementation of the pipeline in this work consists of three main parts: the detection of the staircase in the RGB image and the characterization of the staircase using the point cloud corresponding to the region of interest and the design of the fuzzy controller for the approach and alignment.

The related works in this area have focused mainly on the detection of the staircase using data from RGB images, depth images or point clouds. For RGB images, the work presented in [3–7] first apply a filtering stage to reduce noise and lighting effects and later apply edge detection algorithms, such as a Canny operator, to obtain the edges present in the image. After edge extraction, non-candidate stair edges are eliminated and a linking algorithm is used to determine the lines corresponding to the staircase's parallel edges. Using these lines, certain staircase characteristics can be extracted, such as orientation with respect to the camera.

Additionally, some state of the art staircase localization systems have been implemented using RGB images, deep learning and point clouds. Deep learning methods to detect staircases use neuronal networks such as YOLOv3-tiny [8,9], YOLOv4 [8] and SSD [10,11] or implement a stair step detection using YOLOv3-tiny [12] and staircase segmentation using AlbuNet [13]. The use of CNNs to detect an object in an image allows further processing in the detected area or region of interest (ROI). In [9], they fit lines to the stair edges by using an edge detector and Hough Transform. With these lines, a PID controller is implemented that allows the robot to align with the stairs. On the other hand, in [11] they present a complete pipeline where SSD is used to detect the staircase and later, using a 3D point cloud, obtain the planes within the detected region. By doing so they are able to estimate the orientation of the stairs in a simulated environment.

Other algorithms that use depth cameras or LIDARs detect the horizontal and vertical planes of the staircase by either clustering points in a point cloud [14] or by finding planes using algorithms such as RANSAC [15–17]. In [14] they first start by computing surface normals using a two stage approach. In the first stage, the normal of a point is calculated using tangential vectors computed on a sparse multi-scale neighbourhood. In the second stage, these normals are analyzed in a filtering kernel and normals in a neighborhood are combined into a final result. The direction of the normal vectors obtained are modified based on the orientation data provided by an IMU. They then extract normal surfaces using an edge detector that has as inputs both the the depth and normal map and use patch filtering to eliminate non-candidate regions. The remaining regions are used to decide whether a region contains a staircase or not. In comparison, in [15] an improvised plane fitting algorithm, based on RANSAC, is implemented to calculate the staircase plane and its slope. Once this is done, the staircase bottom mid-point is calculated. Finally in [18], they present a complete pipeline to detect, localize and estimate the characteristics of the staircase using point cloud data obtained from a LIDAR. For detection, they use a plane-based approach and later estimate the staircase parameters with an error of only 2.5 mm.

While most of the works have focused mainly on the staircase detection and characterization, some of them offer solutions that are not robust enough to use in noisy sensors and only a few offer information on the implementation of the algorithm in physical robots. This is a big concern due to the fact that some rescue robots have constraints on energy consumption, computational resources, or communication to the teleoperator's computer. For this reason, this work focuses on providing a robust pipeline using the OAK-D camera that would allow for a quick implementation on robots running ROS and also would allow the robot to keep its autonomy by running all algorithms on board.

2. Materials and Methods

To implement the staircase detection and characterization pipeline we begin by gathering images from different sources, which allows the deep learning model to be trained. Later, we used this dataset to train the YOLOv4-tiny model necessary to obtain the bounding box for the characterization step. To implement the staircase characterization we make use of the region of interest (ROI), defined by the bounding box size and coordinates, and extract the main parameters needed by the fuzzy controller. Figure 1 shows the steps involved in the pipeline.

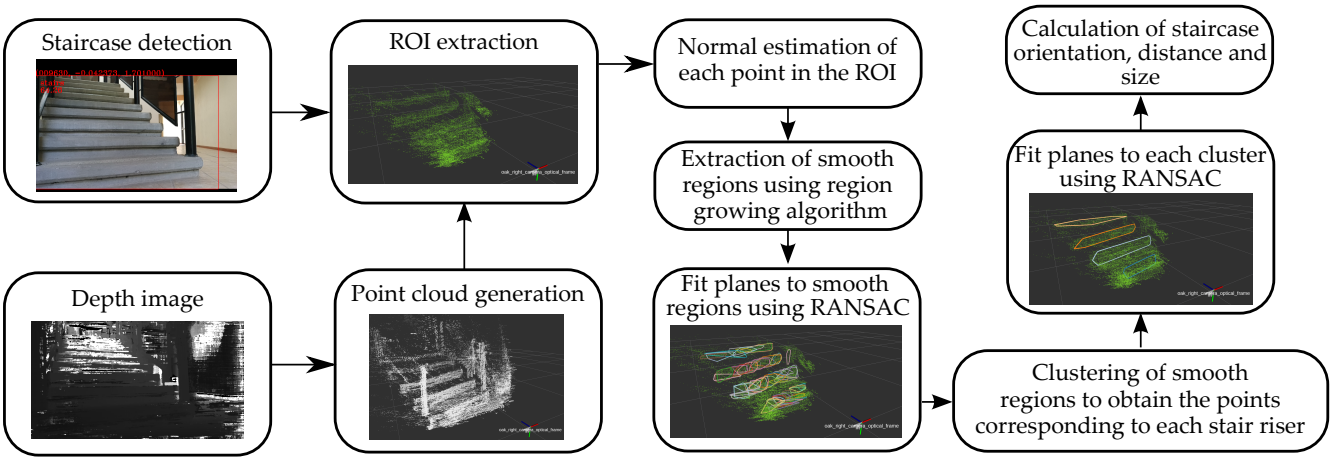


Figure 1. Staircase detection and characterization pipeline used for the OAK-D camera.

2.1. Dataset

One of the drawbacks of training deep neural networks is the amount of data needed to obtain aceptable results. Nevertheless, in this work, we were able to obtain sufficient images by using two available datasets and an image scraping script. The two datasets used were Open Images V6 [19] and MCIndoor20000 [20], and an image scraping script that automatically downloaded images from Google Images [21]. The complete dataset used in this work contains a total of 5826 labeled images with staircases and 4288 non labeled images. The non labeled images were used during training to improve the staircase detector. A total of 4668 previously labeled images were downloaded from the Open Images V6 dataset. The images in this dataset were obtained through image scraping from Flickr (Image and video hosting service, flickr.com) and were manually labeled by professionals [19]. On the other hand, 573 staircase images were obtained from the MCIndoor20000 dataset, which were taken inside the Marshfield Clinic in Wisconsin, US in the summer of 2017. Finally, the images downloaded using Google Images were obtained using a script made for image scraping. None of the images from the last two datasets were labeled, thus the labeling was done using LabelImg [22]. The complete labeled set was split into 70% for training, 20% for validation and 10% for testing. The negative image set (images not labeled and without staircases) were obtained from Open Images V6 and it contained images of objects that could be confused as staircases such as doors, shelves, windows, buildings, bookcases, etc.

2.2. YOLOv4

YOLOv4 is a state-of-the-art object detector that proves to be faster (in terms of frames per second) and more accurate (MS COCO $AP_{50...90}$ and AP_{50}) than the currently available alternatives [23]. This object detector is made up of three parts, a backbone, a neck and a head. YOLOv4 uses a CSPDarknet53 [24] backbone, SSP [25] and PAN [26] as a neck and YOLOv3 [27] for the head.

In this work, the compressed version of YOLOv4, YOLOv4-tiny, was used. This version consists of only 45 layers in total, with just 3 being YOLO (detection) layers. The network uses a 416×416 RGB image as input and outputs the predicted class and a bounding box

width, height and position within the image. This bounding box is needed to determine the ROI in the organized point cloud data that will be used to obtain the planes corresponding to the staircase's tread and riser.

2.3. Staircase Plane Extraction and Characterization

Using the bounding box obtained during the detection step, a segmentation and characterization algorithm is applied to the ROI of the organized point cloud to extract the planes corresponding to riser and calculate the distance and the orientation with respect to the camera's reference frame (shown in Figure 2). Taking into account the fact that a bounding box may contain background information and other objects (such as handrails), we implement a ROI extraction algorithm to eliminate points that do not belong to the staircase's thread and riser. A simple but efficient implementation is to shrink the width of the bounding box by a scaling factor BB_f that varies linearly with the distance to the staircase. The coefficients of the linear function that determines the scaling factor were obtained experimentally and must be changed for each different camera. To obtain the linear function that determines the value of the scaling factor, we first place the robot at a distance d_1 from the staircase and vary the size of detected bounding box by a factor BB_{f1} until its size is equal to the width of the first stair riser. Afterwards, we position the robot at a distance $d_2 < d_1$ and perform the same steps to obtain a factor BB_{f2}. Finally, we obtain the function of the line that passes through the points (d_1, BB_{f1}) and (d_2, BB_{f2}). By doing this we can determine the scaling factor BB_f for different distances from the staircase. This procedure is simple and more than two points can be used by fitting a line through them using linear regression.

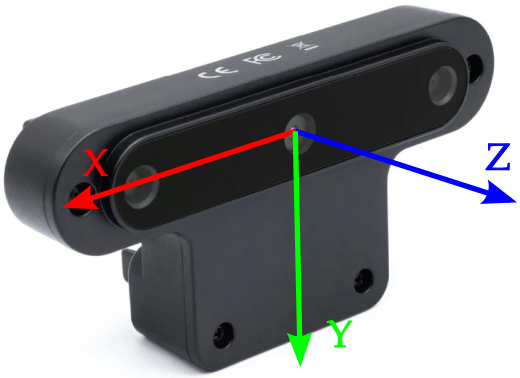


Figure 2. Reference frame used to calculate the staircase parameters.

To segment the staircase's risers, we begin by applying a well known region growing segmentation algorithm [28] to the point cloud corresponding to the detected staircase. This algorithm consists of two main steps, normal estimation and region growing. The normal estimation algorithm fits a plane, using least squares, through the k nearest neighbors of each point in the point cloud and determines the plane normal parameters. By using least squares, the algorithm is able to determine areas of high curvature using the residuals of plane fitting. The region growing step uses the normals and the curvature, along with local connectivity and surface smoothness constraints, to cluster points belonging to the same region. Once the input point cloud has been segmented into smooth regions, we iterate through each region to find its centroid and a best fitting plane using the RANSAC algorithm [29]. By doing this last step, the coefficients corresponding to the Hessian Normal Form defined by $ax + by + cz + d = 0$ and the location of the centroid with respect to the cameras's reference frame are obtained.

With the regions obtained and the planes fitted to each region we apply a clustering algorithm to obtain the points corresponding to the staircase's riser. The clustering takes into account three constraints:

- The angle between planes fitted to each region.

- The vertical distance between the centroid of each region.
- The lateral distance between the centroid of each region.

The algorithm uses the region centroids (represented as a point cloud) $\{P_c\}$, the coefficients of each plane normal $\{N\}$, the clusters corresponding to each smooth region $\{C_p\}$ and the total directions, from the initial seed, it will search for D_{total} as inputs and outputs new clusters corresponding to each staircase riser $\{C_r\}$. To do this, we start by using the camera's reference frame origin as a seed and find the k nearest neighbors using a neighbor finding function Ω that only considers the points available in $\{P_c\}$. For each neighbor, we calculate the angle between the planes corresponding to each centroid and the vertical and horizontal distance between them. If the angle is below a threshold θ_{th} and the distances are below thresholds d_{vt} and d_{ht} respectively, then we have obtained two regions belonging to the same staircase riser. After joining these two regions, we remove the neighbor that met the criteria from $\{P_c\}$ and use it as the next seed point and execute the mentioned steps. The above process continues until no more smooth regions (corresponding to the seed point neighbors) meet the criteria. When this state is reached during the joining process, we return to the initial seed point and now begin to search and join neighboring smooth surfaces. By returning to the initial seed point, we are able to search in D_{total} different directions from the initial seed point, and thus joining all the valid regions for each riser. The complete process continues until all of the points in $\{P_c\}$ have been used as seed points. The steps mentioned above are shown in Algorithm 1.

Once the points corresponding to each staircase riser have been joined we use a RANSAC algorithm to find a best fitting plane and obtain the corresponding coefficients. To determine the staircase orientation with respect to the camera frame, we project each plane normal into the $y = 0$ plane in the camera's reference frame (XZ plane). The orthogonal projection of a vector $\vec{v}$ into a subspace H of $\mathbb{R}^n$ with an orthonormal base $\{\vec{u}_1, \vec{u}_2, \ldots, \vec{u}_k\}$ is given by Equation (1) [30].

$$proy_H v = (\vec{v} \cdot \vec{u}_1)\vec{u}_1 + (\vec{v} \cdot \vec{u}_2)\vec{u}_2 + \cdots + (\vec{v} \cdot \vec{u}_k)\vec{u}_k \tag{1}$$

For the plane given by the equation $y = 0$, the orthonormal base is given by: $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. Therefore, the projection of any plane normal $\vec{v}$ into plane $\pi = \left\{ \begin{pmatrix} x \\ y \\ z \end{pmatrix} : y = 0 \right\}$ is given by:

$$proy_\pi \vec{v} = \left[\vec{v} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right] \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \left[\vec{v} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right] \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{2}$$

Finally, the angle θ between normal $\vec{v}$ and the unit vector $\vec{x} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ (representing the right direction in the camera's reference frame) is given by:

$$\theta = atan2((\vec{x} \times proy_\pi \vec{v}) \cdot \vec{y}, \vec{x} \cdot proy_\pi \vec{v}) \tag{3}$$

where $atan2$ is the four-quadrant inverse tangent and $\vec{y} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ is the vector perpendicular to $\vec{x}$ and $proy_\pi \vec{v}$. Through this process we are able to determine the orientation of the staircase even when the risers do not have a completely flat surface or when part of the riser is missing. The angle θ gives the angle, with respect to the camera frame. Furthermore, the segmentation of the staircase into planes allowed the measurement of thread and riser

size by measuring the vertical and horizontal distance between the centroid of each cluster of points (represented by planes). With this information, stair angle can also be determined.

Algorithm 1 Algorithm to cluster smooth surfaces

Inputs: Point cloud of centroids = $\{P_c\}$, point normals = $\{N\}$,
clusters of smooth regions = $\{C_p\}$, neighbor finding function $\Omega(.)$,
number of directions from initial seed to search for D_{total}
Outputs: Clusters of stair risers $\{C_r\}$

```
r ← 0
while size({P_c}) > 1 do
    Nearest neighbors of reference seed {K} ← Ω((0,0,0))
    current seed p_s ← K{0}
    initial p_0 ← p_s
    P_c --remove--> p_0                          ▷ remove seed from initial point cloud
    Indexes of nearest neighbors of current seed {K} ← Ω(p_s)
    i ← 0
    d ← 1
    while i < size({K}) do
        θ ← angle between N{p_s} and N{K{i}}
        d_z ← distance between p_s and K{i} in z direction
        d_y ← distance between p_s and K{i} in y direction
        if | d_z | < d_ht and | d_y | < d_vt and 0 ≤ θ ≤ θ_th then
            C_current <--insert-- points of C_p{p_s}
            C_r{r} <--insert-- points of C_current
            current seed p_s ← K{i}
            P_c --remove--> p_s                  ▷ remove seed from initial point cloud
            Indexes of nearest neighbors of current seed {K} ← Ω(p_s)
            i ← 0
        else
            i ← i + 1
            if i = size({K}) and d < D_total then
                {K} ← Ω(p_0)
                d ← d + 1
                i ← 0
            end if
        end if
    end while
    if size(C_r{r}) > 0 then
        r ← r + 1
    end if
end while
```

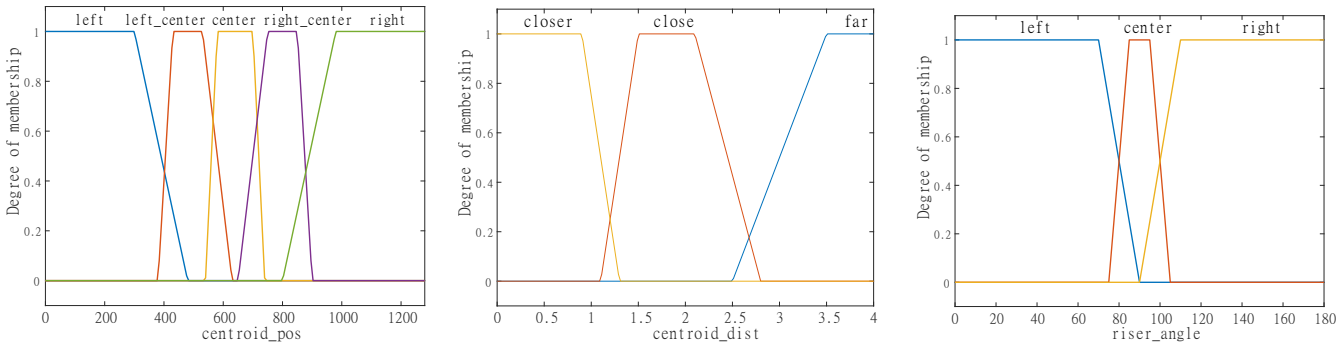
2.4. Fuzzy Controller for Alignment

To implement the staircase approach and alignment algorithm, a fuzzy controller was used. To do this, 45 rules (shown in Table A1 in Appendix A) were implemented using 3 inputs: the position within the image of the detected staircase, the distance to the staircase and the orientation of the riser with respect to the robot. The logic behind this controller is to use the staircase's position within the image frame and the distance to the staircase to approach the obstacle slowly until it can determine the staircase's orientation. The fuzzy controller executes the initial approach in such way that the staircase bounding box is always centered in the image. The robot moves slowly towards the staircase until it can segment at least two risers and determine the orientation of the staircase with respect to the robot. Once this is achieved, the fuzzy controller enters into a second phase where the robot will try to move parallel to the risers and at the same time keep moving towards them, but without losing the obstacle from the image frame. Once the robot has gotten closer to the

staircase, it inters the third phase of alignment where it will now use the orientation of the staircase to align itself with the staircase, in such way that the robot will end up centered in front of it. The fuzzy variables and fuzzy sets are the following:

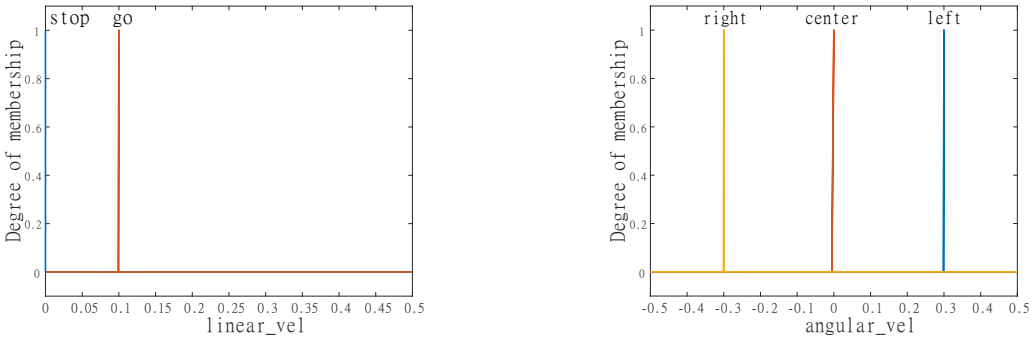
- Fuzzy variables and fuzzy sets:
  - Input variables:
    - Center of the bounding box with the detected staircase *(bb_center_xcentroid_pos)*.
      - Fuzzy sets: *left*, *left_center*, *center*, *right_center* and *right*.
    - Distance to the centroid of the staircase *(centroid_dist)*.
      - Fuzzy sets: *closer*, *close*, *far*.
    - Staircase orientation *(riser_angle)*.
      - Fuzzy sets: *left*, *center* and *right*.
  - Output variables:
    - Linear velocity (linear_vel).
      - Fuzzy sets: *fast*, *slow* and *stopstop* and *go*.
    - Angular velocity (angular_vel).
      - Fuzzy sets: *left*, *center* and *right*.

The membership functions of the variables are shown in Figures 3 and 4. Figure 3a shows the membership functions used for the *centroid_pos* input. The universe of discourse is the width of the RGB image and each fuzzy set represents a zone within this image. On the other hand, in Figure 3b shows the fuzzy sets corresponding to the three distances considered in this controller. For this variable, the universe of discourse was kept between 0 and 4 m and three fuzzy sets were used. When the staircase is *far* from the robot, it is impossible to obtain the staircase orientation due to camera limitations, but once the robot gets *close* or *closer*, the staircase orientation can be obtained. Finally, Figure 3c shows the fuzzy sets that represent the orientations considered. Considering the way the orientation is measured, the universe of discourse for the *riser_angle* variable is between 0 and 180 degrees, where values below 90 degrees represent a staircase rotated to the left, values above 90 degrees represent a staircase rotated to the right and values equal to 90 degrees represent a staircase completely parallel to the camera's frontal plane (XY plane in Figure 2). Finally, for the outputs, we used single value constants or singletons. For the linear velocity, the universe of discourse is kept between 0 and 0.5, but we only use values at 0 (zero velocity) and 0.1 (see Figure 4). For the angular velocity, the universe of discourse is keep between -0.5 and 0.5, where negative numbers represent a rotation to the right and a positive value a rotation to the left. For both output variables, large velocities are not required and they can be unfavorable due to the rate at which the staircase parameters are published.



(**a**) (**b**) (**c**)

Figure 3. Membership functions for the input variables: (**a**) center of bounding box within the image, (**b**) distance to the centroid of the staircase, (**c**) riser orientation with respect to the camera frame.



(**a**) (**b**)

Figure 4. Membership functions for the output variables: (**a**) linear velocity, (**b**) angular velocity.

Due to the ongoing development of the Fuzzy Controller ROS package, only triangular and trapezoidal shapes were available for the input fuzzy sets and singletons for the outputs sets. Furthermore, the package only supports Mamdani Inference Systems. The package limitations did not inhibit our primary goal of proving that the proposed pipeline works for staircase approach and alignment.

2.5. Description of the Cameras Used

In this work, we tested the proposed pipeline with two different low cost RGD-D cameras, the Intel® RealSense™ D435i and the OpenCV AI Kit with Depth (OAK-D) from Luxonis Holding Corporation. The RealSense™ D435i contains a stereo camera system, an RGB camera and an Intel™ RealSense D4 Vision Processor Unit (VPU). The RGB camera can obtain video with a resolution up to 1920 × 1080 and at 30 Frames per Second (FPS). On the other hand, the stereo system uses an infrared pattern projector and the VPU to acquire depth images with a resolution up to 1280 × 720 and up to 90 FPS. In comparison, the OAK-D camera uses a Myriad X VPU, which contains two Neural Compute Engines (NCEs) which are capable of executing 4 Tera Operations per Second (TOPs). By being able to perform neuronal inference on the camera. This camera is also capable of obtaining RGB images at a resolution of 4K and 30 FPS and depth images at a resolution of 1280 × 720 and 30 FPS.

The proposed pipeline was designed to increase autonomy in rescue robots robots, and for this reason, communication to the teleoperator computer should only be for monitoring purposes. The RealSense™ D435i pipeline requires additional hardware to perform the detections, and the OAK-D does not. For this reason, the OAK-D camera is the best option for robots are limited not only in computing resources, but also in space inside their chassis and energy consumption (for example for prebuilt rescue robots). Unfortunately, one of the disadvantages of this camera are the low disparity levels it produces. Due to this, the clustering Algorithm 1 was implemented and it allowed for better comparisons between the two cameras used.

To implement the pipeline using the D435i camera, a teleoperator computer with an Nvidia® RTX 2070 with Max-Q Design was used. This allowed a detection of over 100 FPS, but increased the implementation cost of the pipeline and it required a continuous communication to the teleoperator computer. In comparison, the OAK-D camera can be obtained for about the same price as a D435i camera and it includes NCEs that allows neural inference on the camera. This benefit outweighs the drawbacks of using this camera (for example the low disparity levels).

2.6. ROS Pipeline

To evaluate our proposal we implemented two different ROS pipelines using the two different cameras previously described. The main difference between the two were in the image acquiring and detection steps and also in the necessary steps for plane extraction. fixed minor labeling mistakes in the Figures 5 and 6 show the two pipelines and the

different nodes used for the image acquisition and staircase detection. For the D435i camera we used a pipeline that needed access to a GPU to implement the model, and to obtain the detections. Furthermore, communication with the teleoperator's computer was needed at all times. In comparison, the OAK-D camera pipeline was processed on the robot and the communication with the teleoperator's computer was mainly for monitoring and launching nodes. Another important different is that the RealSense has more disparity levels than the OAK-D and, for this reason, only the region growing based segmentation was needed. In comparison, the OAK-D camera required an extra clustering step (outlined in Algorithm 1), to perform the segmentation.

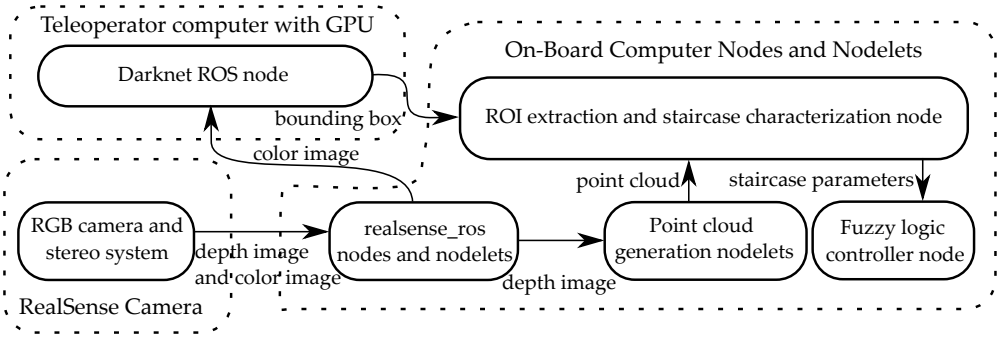


Figure 5. ROS pipeline using the RealSense™ D435i camera.

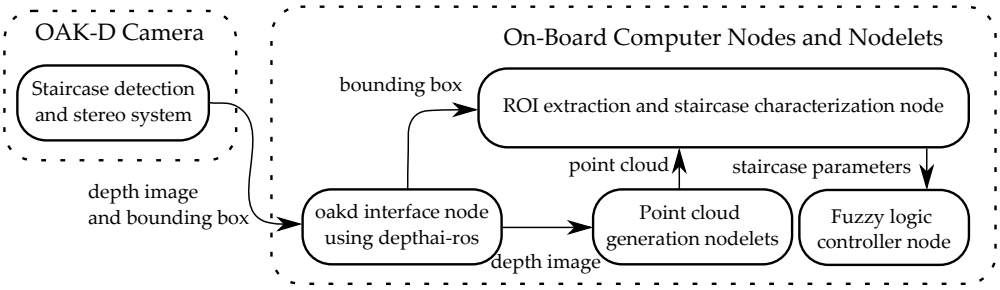


Figure 6. ROS pipeline using the OAK-D camera.

The D435i pipeline uses the *realsense_ros* nodes and nodelets to adquire the RGB and depth images at a resolution of 848 × 480 and they were published at 30 FPS. The RGB image was sent to the detection node (darknet_ros) which published the bounding boxes at approximately 100 FPS. For the OAK-D camera we used the depthai-ros wrapper [31], which allowed the use of the depthAI library. Using this wrapper, we implemented the node required to obtain the RGB and depth images and the spatial detection from the camera. The node configured the RGB camera with a 1920 × 1080 resolution and at 30 FPS, and it was compressed to 416 × 416 for the neural network. Additionally, the stereo cameras were configured with a resolution of 1280 × 720 and at 30 FPS, and the depth image was synchronized with the detections obtained on the RGB image.

The generation of the point cloud from the depth images was done using the *depth_img_proc* [32] nodelets and, for both cameras, a reasonable amount of noise is generated. To decrease the effect of noise, we downsampled the point clouds generated using Voxel filters with a leaf size of 0.11 × 0.11 × 0.11. Furthermore, we only used the points that were in the range of 0 to 5 m and eliminated the ground points. Also, for both pipelines, the bounding box and the point cloud were synchronized before starting the ROI extraction and characterization algorithm. The point cloud handling and processing was done using the Point Cloud Library [33].

The staircase and characterization node published the staircase parameters and the node with the fuzzy logic controller used these parameters to determine the linear and angular velocity commands necessary for the staircase approach and alignment. Additionally, to adjust the membership function values, we first used a simulation of our pipeline

in Gazebo [34] and later performed minimal adjustments on the physical differential drive robot.

3. Results

Multiple YOLOv4 configurations were trained and evaluated on the test images and using these results the best model for the pipeline was chosen. The complete pipeline was first tested in simulation using Gazebo and later was implemented on a physical robot running the Robot Operating System (ROS) and tested with two cameras previously described.

Training of the Detector

The configurations used and the results on the test set are shown in Tables 1 and 2. Seven different YOLOv4-tiny configurations were trained and for comparison purposes, YOLOv4 (test 2) and YOLOv3-tiny (test 3) were also trained. In the tables, each test shows the configurations and evaluation metrics obtained for the best detector in that specific run. Google Colaboratory Hosted Jupyter notebook service (colab.research.google.com) was used for training and evaluation of the models.

Table 1. Configurations of YOLOv4-tiny used and mAP@0.5IoU obtained.

No.	Total Iterations	Subdivisions	Total Layers	Detection Layers	Random Flag	Negative Images	mAP@0.5
1	2000	24	38	2	1	NO	59.25%
2	2000	24	127	3	0	NO	76.74%
3	4000	24	31	3	1	NO	45.24%
4	4000	24	38	2	0	NO	66.20%
5	2000	24	38	2	0	NO	63.08%
6	4000	24	38	2	0	YES	65.53%
7	10,000	8	38	2	0	YES	70.93%
8	10,000	4	45	3	0	YES	68.17%
9	10,000	8	45	3	0	YES	65.32%

Table 2. Test results of the different YOLOv4-tiny configurations used.

No.	Approximate Training Time [min]	Approximate Inference Time [ms]	Precision	Recall	F1 Score	Avg. IoU	mAP@0.5
1	40	5.3	0.76	0.49	0.59	55.09%	59.25%
2	186	44.1	0.80	0.69	0.74	63.62%	76.74%
3	104	5.6	0.85	0.17	0.29	61.86%	45.24%
4	100	5.2	0.73	0.62	0.67	52.76 %	66.20%
5	60	5.2	0.74	0.56	0.64	53.62%	63.08%
6	75	5.47	0.73	0.62	0.67	54.00 %	65.53%
7	175	5.4	0.81	0.62	0.70	61.36%	70.93%
8	160	6.1	0.88	0.52	0.65	67.57 %	68.17%
9	154	6.2	0.77	0.57	0.65	58.22%	65.32%

Test number 1 was used to obtain an initial detector and, in order to improve it, in each test there was a change in parameters or data used. According to [35], detection may be improved by:

- Selecting the appropriate number of iterations and subdivisions of the batch.

- Setting random flag = 1 in config file will increase precision by training YOLO for different resolutions.
- To detect both large and small objects use modified YOLO versions, which include more detection stages (more YOLO layers).
- Use non-labeled images during training.

From the results obtained after training, it can be seen that the YOLOv4-tiny versions trained in this work were not able to match the performance (in terms of mAP) of the full size YOLOv4 version, but they were clearly increased through the use of negative images, and also by choosing the appropriate parameters for training. Furthermore, even though it is recommended to train YOLO at different resolutions (setting random flag = 1), our results showed that disabling this flag resulted in a better mAP on the test set.

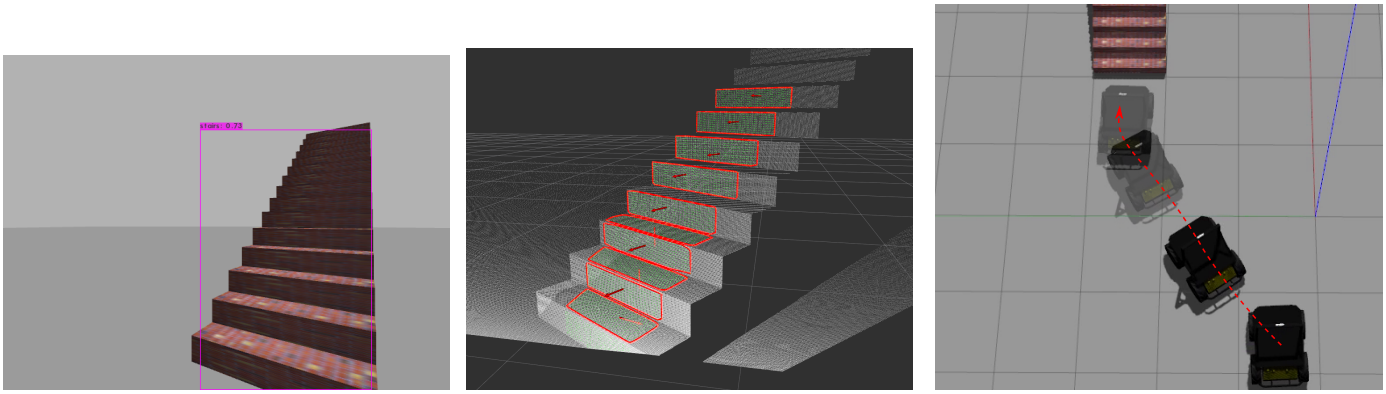
Considering the results in Table 2, the detectors 7, 8 and 9 (obtained in tests 7, 8 and 9, respectively) were chosen to be implemented in the different pipelines. The main differences between theses detectors is the fact that detectors 8 and 9 contain 3 detection layers and detector 7 only contains 2. Furthermore, for detector 8, a recalculation of the anchor boxes was done using the labeled dataset. On the other hand, detector 9 used the original bounding boxes specified by [35]. The three detectors were tested in the two pipelines, and it was found that the 3 YOLO layers in detectors 8 and 9 helped to detect staircases that were far from the robot, meaning that the staircase in the RGB image was small and other objects were also present. Additionally, the recalculated bounding boxes improved mAP@0.5, but decreased recall. Considering the results obtained, detector 9 was chosen to be used in our proposed pipeline.

The images in Figure 7 show the predictions obtained using the detector in test 9 using images taken from the test set. From the images, it can be seen that the detector can make correct predictions for ascending and descending staircases with different textures and viewing angles and even staircases whose step edges are not completely parallel. Figure 7 also shows that multiple staircases in an image can be detected and staircases that are partially occluded by fences or handrails.

Figure 7. Results obtained using detector 9 in images taken from test set.

Before testing the complete pipeline on a physical robot, we implemented it in simulation, mainly to adjust the clustering parameters and to obtain initial parameters for the membership functions that represent each fuzzy set. The use of simulation in this work

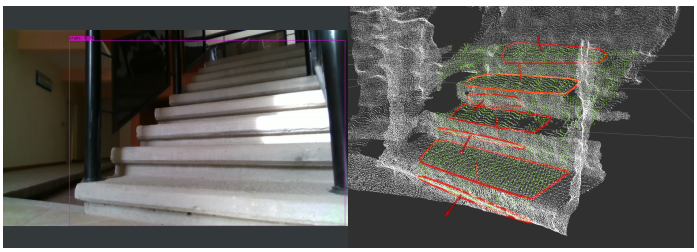
also allowed us to test different fuzzy rules and helped us obtain the fuzzy controller implemented on the physical robot. Figure 8 shows the staircase detection and segmentation of smooth regions on a simulated point cloud and the path the robot takes to arrive at the first step of the staircase.



(**a**) (**b**) (**c**)

Figure 8. Pipeline test in simulation: (**a**) detection, (**b**) smooth region segmentation, (**c**) approach and alignment using Fuzzy controller.

Using the deep learning model obtained in test 9, we proceeded to test both of the pipelines described. For both tests we mounted the camera on top of the differential drive robot and placed it in front of a staircase. The results obtained with the D435i pipeline are shown in Figure 9. From Figure 9b it is possible to observe that the pipeline is able to detect the staircase in the RGB image, obtained from the D435i camera, and using the bounding box we extract the points, from the initial point cloud (points in white), corresponding to the ROI (points in green). Due to the quality of the point cloud, mainly dependent on the disparity levels and the use of an infrared pattern projector, the segmentation was done using only region growing and Figure 9b shows the planes extracted (red polygons).



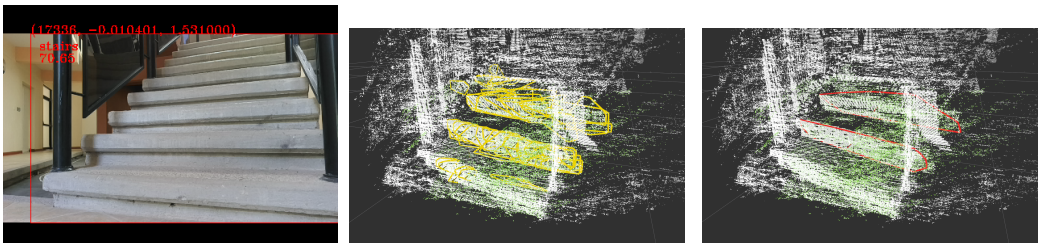
(**a**) (**b**)

Figure 9. Results obtained using the D435i camera pipeline: (**a**) staircase detection, (**b**) smooth region segmentation.

Figure 10 shows the results obtained using the OAK-D camera pipeline. Figure 10a shows the detection obtained using the NCEs of the OAK-D camera and Figure 10b shows the initial point cloud (white points), the points corresponding to the ROI (green points) and the smooth regions obtained after the region growing algorithm (yellow polygons). Compared to the D435i camera, the point cloud is rather noisy and has less levels of disparity, making it necessary to implement the smooth region clustering algorithm. Figure 10c shows the clustered smooth regions (red polygons) and, therefore, the segmented staircase risers.

Considering all the components in the staircase detection and characterization pipeline, the staircase parameters were published at a rate of approximately 2 Hz for both the D435i and OAK-D camera. Also, taking into account the low computing resources available in the test robot (it only contains a second generation Intel® Core™ i5 CPU @ 2.60 GHz), it was decided to use the OAK-D pipeline to test the fuzzy logic controller. The fuzzy logic controller used the staircase parameters to perform fuzzy inference and publish linear and

angular velocity commands, needed by the differential drive robot to approach and align with the staircase. To test the controller, the robot was placed in different positions, with respect to the staircase and at different distances from the staircase. Then, the complete pipeline was executed using the teleoperator computer and the robot moved, autonomously, towards the staircase until the orientation could be obtained (at approximately 1.5 m) and then it aligned itself with the staircase. The robot stopped moving when the staircase centroid is at 90 cm and the camera's front plane is parallel to the planes of the staircase risers. The fuzzy controller was tested using two different staircases.



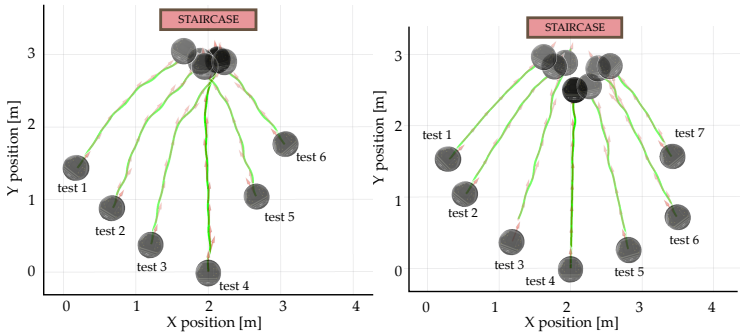
(**a**) (**b**) (**c**)

Figure 10. Results obtained using the OAK-D camera pipeline: (**a**) staircase detection, (**b**) smooth region segmentation, (**c**) clustering of smooth regions.

Figure 11 shows examples of the tests done using staircase 1 and staircase 2. In each figure, the path the robot takes to reach the staircase are shown with a red dashed line. Additionally, Figure 12 shows the trajectories obtained in 13 different tests using the staircases shown in Figure 11. From Figure 12, we determined that the pipeline struggles to align the robot when the staircase risers have no texture (as is the case for staircase 2). This is due to the fact that the camera depends on texture to obtain the disparity map, and consequently the point cloud. Without an acceptable point cloud, the staircase orientation cannot be determined. In comparison, the texture of the risers of staircase 1 allow the robot to determine its orientation and therefore better align itself with the obstacle. The tests also show that the fuzzy controller struggles to position the robot correctly when it is placed a a distance greater than 1.5 m to the left or to the right of the staircase's center line. When this happens, the robot cannot center itself completely with the staircase. Considering this, a better way to implement the staircase approach and alignment would be to use a path planning algorithm that uses the staircase parameters. By doing so, it would not be necessary to keep the staircase inside the camera image at all times. This would allow a significant improvement to the staircase alignment phase of the algorithm.

(**a**) (**b**) (**c**)

Figure 11. Staircase approach and alignment paths obtained using Fuzzy Logic: (**a**) positioned to the left of staircase 1, (**b**) robot positioned to the left of staircase 2, (**c**) robot positioned to the right of staircase 2.



(**a**) (**b**)

Figure 12. Staircase approach and alignment paths obtained using the Fuzzy Logic Controller: (**a**) tests done with staircase 1, (**b**) tests done with staircase 2.

4. Conclusions

The results prove that the proposed staircase detection and characterization works with different cameras and can be used to implement an approach and alignment algorithm. By using downsampling, region growing and clustering algorithms we were able to reduce the effects of noise in the plane segmentation process and also increase the execution speed. Due to this, the algorithm can be used in point clouds obtained from cameras with low disparity levels or with a large amount of noise and it can be used to implement a fuzzy logic controller. The staircase approach algorithm can be improved by implementing a path planning algorithm that would allow the robot to reach the staircase without the need to have the staircase inside the image frame at all times.

Our proposal can significantly improve the teleoperatibiliy and autonomy of a search and rescue robot or other robots requiring staircase detection and characterization. For this reason, for future work we intend to detect and characterize more objects, such as ramps or rubble, that the robot could encounter and traverse during its mission.

The empty at the end of this page is eliminated once the changes are accepted and the marked changes, and the corresponding footnotes, are removed.

Author Contributions: Conceptualization, J.A.S.-R. and A.E.P.-B.; methodology, J.A.S.-R. and A.E.P.-B.; software, J.A.S.-R.; validation, J.A.S.-R., J.A.A.-A., H.T. and A.E.P.-B.; formal analysis, J.A.S.-R., J.A.A.-A., H.T. and A.E.P.-B.; investigation, J.A.S.-R. and A.E.P.-B.; resources, J.A.S.-R., J.A.A.-A., H.T. and A.E.P.-B.; data curation, J.A.S.-R.; writing—original draft preparation, J.A.S.-R.; writing—review and editing, J.A.A.-A., H.T. and A.E.P.-B.; visualization, J.A.S.-R., J.A.A.-A., H.T. and A.E.P.-B.; supervision, J.A.A.-A., H.T. and A.E.P.-B.; project administration, J.A.A.-A. and A.E.P.-B.; funding acquisition, J.A.A.-A. and A.E.P.-B. All authors have read and agreed to the published version of the manuscript.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data that support the findings of this study are available from the corresponding author, upon reasonable request.

Acknowledgments: The authors would like to thank the Technological University of the Mixteca for providing its facilities for this research project. We would also like to thank CONACYT and the Tokyo University of Science for their financial support.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The 45 fuzzy rules used for the approach and alignment algorithm are shown in Table A1.

Table A1. Fuzzy rules used for the staircase alignment.

Rule	*IF*		*THEN*		
	Centroid_pos AND	**Centroid_dist AND**	**Riser_angle**	**Linear_vel AND**	**Angular_vel**
1	left	far	left	go	left
2	left	far	center	go	left
3	left	far	right	go	left
4	left	close	left	go	left
5	left	close	center	go	left
6	left	close	right	go	left
7	left	closer	left	stop	left
8	left	closer	center	stop	left
9	left	closer	right	stop	left
10	left_center	far	left	go	left
11	left_center	far	center	go	left
12	left_center	far	right	go	left
13	left_center	close	left	go	right
14	left_center	close	center	go	left
15	left_center	close	right	go	left
16	left_center	closer	left	stop	left
17	left_center	closer	center	stop	left
18	left_center	closer	right	stop	left
19	center	far	left	go	center
20	center	far	center	go	center
21	center	far	right	go	center
22	center	close	left	go	right
23	center	close	center	go	center
24	center	close	right	go	left
25	center	closer	left	stop	left
26	center	closer	center	stop	center
27	center	closer	right	stop	right
28	right_center	far	left	go	right
29	right_center	far	center	go	right
30	right_center	far	right	go	right
31	right_center	close	left	go	right
32	right_center	close	center	go	right
33	right_center	close	right	go	left
34	right_center	closer	left	stop	right
35	right_center	closer	center	stop	right
36	right_center	closer	right	stop	right
37	right	far	left	go	right
38	right	far	center	go	right
39	right	far	right	go	right
40	right	close	left	go	right
41	right	close	center	go	right
42	right	close	right	go	right
43	right	closer	left	stop	right
44	right	closer	center	stop	right
45	right	closer	right	stop	right

References

1. Murphy, R.R. *Disaster Robotics*; MIT Press: Cambridge, MA, USA, 2014.
2. Tadokoro, S. *Disaster Robotics: Results from the ImPACT Tough Robotics Challenge*; Springer: Cham, Switzerland, 2019; Volume 128.
3. Cong, Y.; Li, X.; Liu, J.; Tang, Y. A stairway detection algorithm based on vision for ugv stair climbing. In Proceedings of the 2008 IEEE International Conference on Networking, Sensing and Control, Sanya, China, 6–8 April 2008; pp. 1806–1811.
4. Hernández, D.C.; Jo, K.H. Stairway tracking based on automatic target selection using directional filters. In Proceedings of the 2011 17th Korea-Japan Joint Workshop on Frontiers of Computer Vision (FCV), Ulsan, Korea, 9–11 Febuary 2011; pp. 1–6.
5. Huang, X.; Tang, Z. Staircase Detection Algorithm Based on Projection-Histogram. In Proceedings of the 2018 2nd IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), Xi'an, China, 25–27 May 2018; pp. 1130–1133.
6. Lee, H.W.; Wang, C.; Lu, B.Y. Study on the Computer Vision of the Biped Robot for Stable Walking on the Stairs. In Proceedings of the 2019 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Yilan, Taiwan, 20–22 May 2019; pp. 1–2.
7. Zhong, C.; Zhuang, Y.; Wang, W. Stairway detection using Gabor filter and FFPG. In Proceedings of the 2011 International Conference of Soft Computing and Pattern Recognition (SoCPaR), Dalian, China, 14–16 October 2011; pp. 578–582.
8. Kajabad, E.N.; Begen, P.; Nizomutdinov, B.; Ivanov, S. YOLOv4 for Urban Object Detection: Case of Electronic Inventory in St. Petersburg. In Proceedings of the 2021 28th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 27–29 January 2021; pp. 316–321.
9. Patil, U.; Gujarathi, A.; Kulkarni, A.; Jain, A.; Malke, L.; Tekade, R.; Paigwar, K.; Chaturvedi, P. Deep learning based stair detection and statistical image filtering for autonomous stair climbing. In Proceedings of the 2019 Third IEEE International Conference on Robotic Computing (IRC), Naples, Italy, 25–27 February 2019; pp. 159–166.
10. Ilyas, M.; Lakshmanan, A.K.; Le, A.V.; Mohan, R.E. Staircase recognition and localization using convolution neural network (cnn) for cleaning robot application. *Preprints* **2018**. [CrossRef]
11. Miyakawa, K.; Kanda, T.; Ohya, J.; Ogata, H.; Hashimoto, K.; Takanishi, A. Automatic estimation of the position and orientation of stairs to be reached and climbed by a disaster response robot by analyzing 2D image and 3D point cloud. *Int. J. Mech. Eng. Robot. Res.* **2020**, *9*, 1312–1321. [CrossRef]
12. Choi, Y.J.; Rahim, T.; Ramatryana, I.N.A.; Shin, S.Y. Improved CNN-Based Path Planning for Stairs Climbing in Autonomous UAV with LiDAR Sensor. In Proceedings of the 2021 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Korea, 31 January–3 Febuary 2021; pp. 1–7.
13. Panchi, N.; Agrawal, K.; Patil, U.; Gujarathi, A.; Jain, A.; Namdeo, H.; Chiddarwar, S.S. Deep Learning-Based Stair Segmentation and Behavioral Cloning for Autonomous Stair Climbing. *Int. J. Semant. Comput.* **2019**, *13*, 497–512. [CrossRef]
14. Ciobanu, A.; Morar, A.; Moldoveanu, F.; Petrescu, L.; Ferche, O.; Moldoveanu, A. Real-time indoor staircase detection on mobile devices. In Proceedings of the 2017 21st International Conference on Control Systems and Computer Science (CSCS), Bucharest, Romania, 29–31 May 2017; pp. 287–293.
15. Sharma, B.; Syed, I.A. Where to begin climbing? Computing start-of-stair position for robotic platforms. In Proceedings of the 2019 11th International Conference on Computational Intelligence and Communication Networks (CICN), Honolulu, HI, USA, 3–4 January 2019; pp. 110–116.
16. Souto, L.A.; Castro, A.; Gonçalves, L.M.G.; Nascimento, T.P. Stairs and doors recognition as natural landmarks based on clouds of 3D edge-points from RGB-D sensors for mobile robot localization. *Sensors* **2017**, *17*, 1824. [CrossRef] [PubMed]
17. Woo, S.; Shin, J.; Lee, Y.H.; Lee, Y.H.; Lee, H.; Kang, H.; Choi, H.R.; Moon, H. Stair-mapping with point-cloud data and stair-modeling for quadruped robot. In Proceedings of the 2019 16th International Conference on Ubiquitous Robots (UR), Jeju, Korea, 24–27 June 2019; pp. 81–86.
18. Westfechtel, T.; Ohno, K.; Mertsching, B.; Hamada, R.; Nickchen, D.; Kojima, S.; Tadokoro, S. Robust stairway-detection and localization method for mobile robots using a graph-based model and competing initializations. *Int. J. Robot. Res.* **2018**, *37*, 1463–1483. [CrossRef]
19. Kuznetsova, A.; Rom, H.; Alldrin, N.; Uijlings, J.; Krasin, I.; Pont-Tuset, J.; Kamali, S.; Popov, S.; Malloci, M.; Kolesnikov, A.; et al. The open images dataset v4. *Int. J. Comput. Vis.* **2020**, 128, 1956–1981. [CrossRef]
20. Bashiri, F.S.; LaRose, E.; Peissig, P.; Tafti, A.P. MCIndoor20000: A fully-labeled image dataset to advance indoor objects detection. *Data Brief* **2018**, *17*, 71–75. [CrossRef] [PubMed]
21. Google. Google Images. Available online: https://www.google.com/imghp?hl=en (accessed on 10 December 2020).
22. Tzutalin. LabelImg. Git code. Available online: https://github.com/tzutalin/labelImg (accessed on 7 January 2021).
23. Bochkovskiy, A.; Wang, C.Y.; Liao, H.Y.M. Yolov4: Optimal speed and accuracy of object detection. *arXiv* **2020**, arXiv:2004.10934.
24. Wang, C.Y.; Liao, H.Y.M.; Wu, Y.H.; Chen, P.Y.; Hsieh, J.W.; Yeh, I.H. CSPNet: A new backbone that can enhance learning capability of CNN. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Seattle, WA, USA, 14–19 June 2020; pp. 390–391.
25. He, K.; Zhang, X.; Ren, S.; Sun, J. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *37*, 1904–1916. [CrossRef]
26. Liu, S.; Qi, L.; Qin, H.; Shi, J.; Jia, J. Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–23 June 2018; pp. 8759–8768.
27. Redmon, J.; Farhadi, A. Yolov3: An incremental improvement. *arXiv* **2018**, arXiv:1804.02767.

28. Rabbani, T.; Van Den Heuvel, F.; Vosselmann, G. Segmentation of point clouds using smoothness constraint. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2006**, *36*, 248–253.
29. Fischler, M.A.; Bolles, R.C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* **1981**, *24*, 381–395. [CrossRef]
30. Grossman, S.I. *Álgebra Lineal*, 7th ed.; McGraw Hill/Interamericana Editores, S.A. de C.V.: Mexico City, Mexico, 2012.
31. Luxonis. Depthai-Ros. Git Code. Available online: https://github.com/luxonis/depthai-ros (accessed on 1 August 2021).
32. ROS. Depth_img_proc. Package Summary. Available online: http://wiki.ros.org/depth_image_proc (accessed on 17 July 2021).
33. Rusu, R.B.; Cousins, S. 3d is here: Point cloud library (pcl). In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011; pp. 1–4.
34. Gazebo. Robot Simulation Made Easy. Available online: http://gazebosim.org/ (accessed on 1 September 2021).
35. AlexeyAB. Darknet. Git Code. Available online: https://github.com/AlexeyAB/darknet (accessed on 15 August 2021).