mathematics

MDPI

Article

Staircase Recognition and Localization Using Convolutional Neural Network (CNN) for Cleaning Robot Application

Muhammad Ilyas [1,*,†], Anirudh Krishna Lakshmanan [2,†], Anh Vu Le [3,†] and Mohan Rajesh Elara [2,*]

[1] School of IT and Engineering, Kazakh-British Technical University (KBTU), Almaty 050000, Kazakhstan
[2] Engineering Product Development Pillar, Singapore University of Technology and Design (SUTD), Singapore 487372, Singapore; anikl9705@gmail.com
[3] Communication and Signal Processing Research Group, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam; leanhvu@tdtu.edu.vn
* Correspondence: m.ilyas@kbtu.kz (M.I.); rajeshelara@sutd.edu.sg (M.R.E.)
† These authors contributed equally to this work.

Abstract: Floor-cleaning robots are primarily designed to clean on a single floor, while multi-floor environments are usually not considered target applications. However, it is more efficient to have an autonomous floor-cleaning robot that can climb stairs and reach the next floors in a multi-floor building. To operate in such environments, the ability of a mobile robot to autonomously traverse staircases is very important. For this operation, staircase detection and localization are essential components for planning the traversal route on staircases. This article describes a deep learning approach using a convolutional neural network (CNN)-based robot operation system (ROS) framework for staircase detection, localization, and maneuvering of the robot to the detected stair. We present a real-time object detection framework to detect staircases in incoming images. We also localize these staircases using a contour detection algorithm to detect the target point: a point close to the center of the first step, and an angle of approach to the target point with respect to the current location of the robot. Experiments are performed with data from images captured on different types of staircases at different viewpoints/angles. The experimental results show that the presented approach can achieve an accuracy of 95% and a recall of 86.81%. A total runtime of 155 ms is taken to identify the presence of a staircase and the detection of the first step in the working environment, as well as being able to locate the target point with an accuracy of ±2 cm, ±1 degree.

Keywords: staircase recognition; convolutional neural networks (CNNs); re-configurable robot; contour detection

MSC: 68T40; 93C85

Citation: Ilyas, M.; Lakshmanan, A.K.; Le, A.V.; Elara, M.R. Staircase Recognition and Localization Using Convolutional Neural Network (CNN) for Cleaning Robot Application. *Mathematics* **2023**, *11*, 3964. https://doi.org/10.3390/math11183964

Academic Editors: Shuai Li, Dechao Chen, Mohammed Aquil Mirza, Vasilios N. Katsikis, Dunhui Xiao and Predrag S. Stanimirovic

Received: 14 July 2023
Revised: 21 August 2023
Accepted: 14 September 2023
Published: 18 September 2023

1. Introduction

Robots are key for the automation of various tasks and have revolutionized many fields in the 21st century. They are extremely precise and efficient in doing tasks that otherwise would be difficult or impossible. One such type of robot that is becoming increasingly important is a cleaning robot, which is programmed to work in indoor environments [1–3]. These robots have vast potential to enhance productivity in cleaning tasks in domestic and commercial settings and have witnessed a steep rise over the last two decades. It is estimated that between 2015 and 2018, about 25.2 million robotic cleaning units were sold worldwide [4].

Indoor robots (for cleaning and/or logistics) are designed for fully autonomous traversal in indoor environments [1]. Typical indoor traversal requires the robot to be able to avoid obstacles by integrating the sensing models and communication modules [5,6]. However, if the environment is multi-floored, the robot is required to have the ability to detect and localize the staircases in order to perform cleaning tasks on the staircase and reach

the next floor. Conventional indoor cleaning robots are usually designed for single-floor operations, i.e., they cannot climb the stairs and reach the next floor. However, many real-world indoor environments (residential and commercial buildings) have multiple floors connected by staircases. The stairs may be of different types, e.g., straight, spiral, and L-shaped staircases, and of different materials as well. This severely limits the ability of a conventional cleaning robot to be effective in such scenarios.

The significance of stair-climbing robots has been realized in modern buildings in domestic as well as commercial settings, such as malls, offices, schools, factories, etc. Conventional floor-cleaning robots without stair-climbing capabilities have limited usage in such cases. Stair-climbing robots can detect and localize the staircase and plan its path to climb the stairs and reach the next floor. A single robot can perform cleaning tasks on multiple floors as well as on staircases connecting these floors, thus making them more versatile, efficient, and cost-effective. In summary, the significance of stair-cleaning robots lies in their ability to provide convenience, efficiency, safety, and consistency in cleaning staircases. They offer practical solutions for both residential and commercial environments, making the maintenance of stairs easier and more effective.

To enable robots to traverse staircases, the accurate detection and location of staircases is highly critical. This would enable robots to plan and navigate through such environments, thereby making them much more effective for real-world indoor environments.

Recognizing a staircase seems straightforward for humans. However, for robots to be able to detect, recognize, and localize the staircases, this task is much more complex and challenging. For this, robots should be able to analyze incoming images and detect and recognize stair-like structures, among many other objects, in the working environment. Furthermore, accurate knowledge of the location of the staircase and the angle of approach is very important for reaching and aligning the robot with the staircase in order to start climbing it. This makes staircase detection and localization a highly challenging task for fully autonomous robots.

In this article, we present a solution to work with the sTetro platform [7] for object detection tasks in real time. sTetro is a small, reconfigurable cleaning robot that can change its shape to climb staircases autonomously. In previous work, we validated the sTetro robot with respect to area coverage by benchmarking its performance with a fixed morphology robot. The results indicated that the sTetro robot could achieve superior coverage performance through its shape-shifting ability. However, the validation was conducted by passing manual commands to navigate the robot toward the first-step position of the staircase, and there were no autonomous strategies applied. In this paper, we extend our previous works by integrating the sensing modules and manipulation modules with sTetro robot using an ROS middleware environment that enables the robot to navigate autonomously to the staircase. For this, we use a deep learning approach to detect and recognize staircases in the incoming image stream. However, the robot also requires knowledge of how to move based on the staircase position/location. To this end, after detecting staircases, the next step is to localize the stairs with respect to the robot. For this, we use our own developed contour detection algorithm to find the target point (center of the first step of the staircase) and angle of approach. This enables the robot to approach the center of the staircase, align itself to the first stair, and then start climbing.

This article proposes a real-time staircase detection and localization for a low-cost stair-climbing robot (sTetro) with limited computing resources onboard. Toward achieving this aim, the following contributions are made:

1. A client-server computational framework for real-time object detection and motion control with ROS middleware for embedded-system-based small robots.
2. A contour-based algorithm for calculating the staircase's middle point and angle of approach from the current position of the robot.
3. Tests in real environments with motion control of the robot [8].

The rest of this article is organized as follows: Section 2 describes the related work. In Section 3, the technical details of the major components of the proposed system are

presented. The experimental setup is given in Section 4, and the experimental results are presented and discussed in Section 5. Finally, Section 6 concludes this article with some future work proposed in this direction.

2. Related Work

Many different approaches have been proposed in the literature to address the problem of staircase detection. These approaches can be grouped based on the sensors and algorithms used. Standard sensors, like RGB and RGB-D, have been used for staircase detection, with the prominent approach of detecting parallel lines in the image [9]. Though this type of algorithm works for many simple scenarios, they have many limitations. Hough transform [10], the preferred approach for the detection of parallel lines in traditional computer vision methods, fails to detect staircases that are curved or spiral. Furthermore, these algorithms assume that the robot is parallel to the staircase and facing it. However, this is not the case for most real-world scenarios. The robot must be able to detect and then plan its approach to the staircase. These approaches are also not designed for staircase-climbing robots. These robots require alignment with the staircase, which requires data pertaining to the first step of the staircase. Other approaches using RGB or RGB-D cameras include using Gabor filters, 3D column maps [11], etc. These are able to handle approaching staircases from different angles. Even LiDARs have been used for the detection of staircases [12–14]. However, these approaches also assume that the robot is parallel to the staircase and facing it. Some other sensors that have been used are monocular [15–18] and stereo sensors [19]. In the aforementioned approaches, there is a research gap for real-time staircase detection and localization from arbitrary viewpoints for low-cost mobile robot applications, for example, cleaning and logistic robots.

Object detection and classification using convolutional neural networks (CNN) have been researched intensively in recent decades and have set a revolutionary era in many diverse applications with different kinds of input data, e.g., one such application of CNN using acoustic emission data to predict the roughness of surfaces is given in [20,21]. The use of object detection with CNN models has many exciting applications in robotics [22–24]. Deep learning using large neural networks can find the unique features of various objects automatically, thereby reducing the need for pre-defined kernel-based solutions. Since the features of the objects can be identified, the most significant achievement of these deep-learning-based methods is the ability to identify uncanny features in object classification. As a result, the effectiveness and accuracy of deep-learning-based methods outperform conventional computer vision methods significantly [25]. However, the biggest challenge is training these large networks, as they require a large amount of computation to converge by estimating the various parameters defined in the network. Recently, technologies in parallel computing, such as Compute Unified Device Architecture (CUDA) [26] and NVIDIA CUDA deep neural network (CuDNN) [27], have enabled parallel computing using multiple threads to process large calculations in separate graphic cards with their own graphic processing unit (GPU). Consequently, the training time of these networks has reduced sharply. However, real-time deployment of these deep learning approaches in low-cost and compact embedded controllers is still a big challenge for commercial applications. However, with the dawn of IoT devices, using a server-client framework has reduced the computation load on embedded systems and made it practical to use deep learning models in many real-time applications.

There are many advantages to using deep learning for staircase detection. The model can be trained to detect different types of staircases, including spiral and curved, which proves to be very difficult for standard algorithms. Furthermore, they can be trained to recognize staircases from different angles. This eliminates the need for the robot to be in front of the staircase. These models are highly accurate as well. This is because these models learn the features from data rather than specifying them in the algorithm. Recently, the advances in object detection, including object localization and object classification, are driven by the success of state-of-the-art-convolution network (ConvNet)-based object

detectors called the region-based convolutional network method (RCNN) [28]. The issue with deep learning approaches is the computation power required to run these models in real-time. However, with the recent update to MobileNets [29,30] and the release of single-shot multi-box detectors (SSDs) [31], these models are now capable of running in real-time on low-cost hardware. We also compute the target point and the angle of approach, which allows the robot to align itself with the center of the staircase, thereby having enough space to be able to climb the staircase.

3. Methodology

In this section, we describe the details of our proposed system of real-time staircase detection and localization for stair-cleaning robotic applications.

3.1. ROS-Based Client–Server Computational Framework

It is well known that neural networks have large computation costs. Due to this, the implementation of CNN models on low-cost low-powered hardware in real time is almost impossible. Hence, we propose a client-server model for real-time applications. The proposed system is built in the ROS environment [32]. ROS provides the infrastructure and mechanism for ROS modules. The ROS master installed on the remote server monitors the entire ROS ecosystem. An ROS-based block diagram of the proposed system using a client-server model is shown in Figure 1.

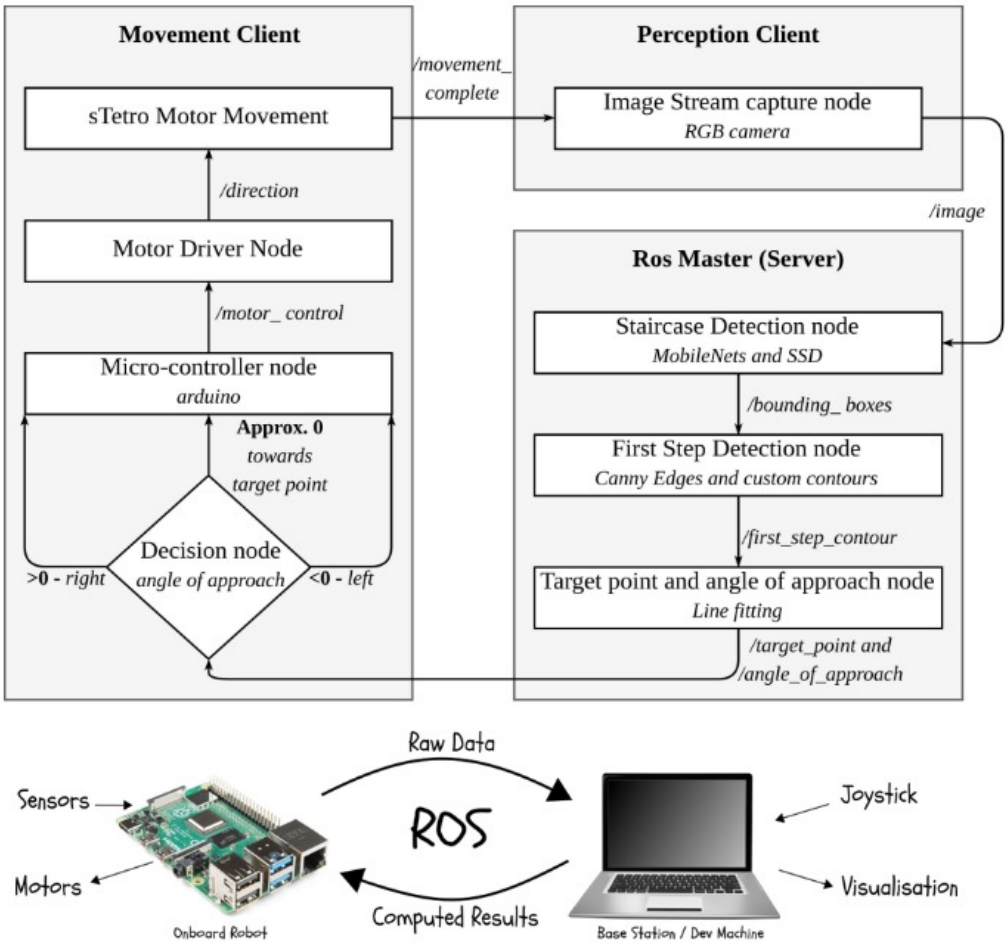


Figure 1. Client–server framework for real-time staircase detection with ROS middleware: (**Left**) sTetro robot with a camera, (**right**) ROS-based remote server.

The robot itself has a small single-board computer (SBC) onboard, and it is connected to a larger, powerful computer (e.g., mid-range laptop or desktop: remote server) off-board that acts as both a "base station" as well as a development machine. The SBC processes the raw sensor data into ROS-compatible messages and transmits them over the network to the base station, which is running ROS. The base station uses these incoming data to compute the appropriate motion commands and send them back over the network to the robot. The onboard SBC will then convert these into signals to send to the motor driver to move the robot. This "base station" PC is also used as a development machine to run CNN models, process input image data, visualize the data, and run algorithms in real-time, as described below.

The ROS topics transmitted to the ROS network through WiFi or data networks enable communication between ROS nodes. Specifically, first, the perception client uses an image sensor to extract an image from the image stream through the ***Image Stream Capture*** node and send it over to the server running ROS. The server uses two steps to detect and localize staircases. The first step involves the detection of staircases. This includes extracting features from an RGB image using MobileNet architectures. We then classify and predict a bounding box using the SSD architecture. This is conducted in the ***Staircase Detection*** node.

The second step has two components: The detection of the first step of the staircase. This is conducted in the ***First-Step Detection*** node. The stair's first-step information is then passed to the next node in ROS, the ***Target Point and Angle of Approach*** node, which is used to detect the target point, $p_{x,y}$, a point close to the center of the first step. The angle of approach, θ, is also computed in this step. We then send this information to the portion of the client responsible for robot movement. Based on the angle of approach, we decide on a movement strategy, e.g., move forward, left, or right. This movement command is executed by the onboard microcontroller with the help of actuators (DC motors) on robot wheels. For the next cycle, the perception client feeds the ROS server with new input image data. This process is repeated until the target point is reached.

3.2. Feature Extraction Using MobileNets

Specialized CNN-based architectures like MobileNets [29,30], AlexNet [33], Inception [34], ResNet [35], etc., are highly accurate at classifying images. The recently proposed Inception-ResNet [36] has the highest accuracy. However, we use the MobileNet architecture due to its low computation cost, which allows for real-time image classification on mobile hardware [37]. The MobileNet is an architecture model of the CNN that explicitly focuses on image classification for real-time applications. Rather than using the standard convolution layers of CNN, it uses depthwise separable convolution layers, which makes it much more efficient than contemporary CNN architectures. This reduces the computational cost and requires very low computational power to run or apply transfer learning (note that by using transfer learning, we do not need to build a CNN model from scratch). Specifically, we use MobileNetv2 [30], which is an improvement over the standard MobileNet [29]. The basic blocks of both architectures are shown in Figure 2. The extracted features of MobileNet are used by SSD architecture to classify and localize the staircases. The structure of SSD is discussed in the next section.

3.2.1. Mobilenet Architecture

For computer vision tasks, CNN models are extensively used nowadays. One specialized CNN-based architecture, MobileNet, uses so-called depthwise separable convolutions instead of conventional convolutional layers. Note that in 2D convolutions, the 2D convolution operation is performed over all input channels, whereas in depthwise convolution, each channel is kept separate. This reduces the huge computation burden on the CNN model for real-time classification tasks.

The MobileNet architecture consists of a single 3×3 convolution layer followed by the batch norm and a modified rectified linear unit (ReLU). The convolution layer is split

into a 3×3 depthwise convolution and a 1×1 pointwise convolution. The input data are filtered by the depthwise convolution and the pointwise convolution layer combines the features created by the depthwise convolution to create new features.

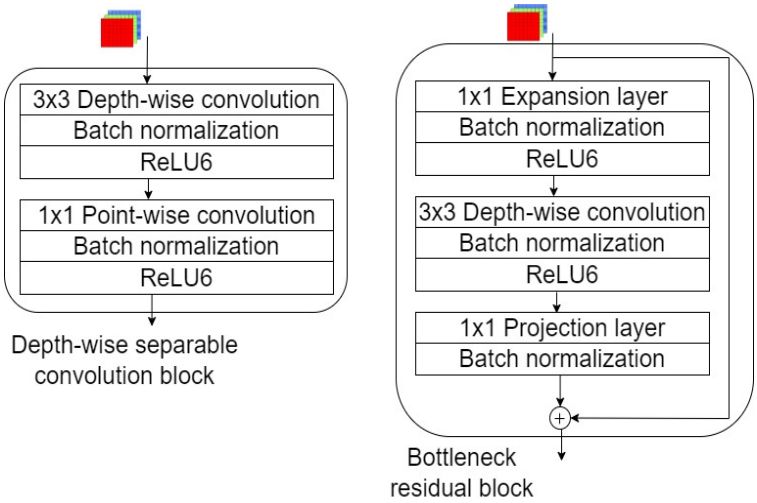


Figure 2. MobileNet v1 (**left**) and MobileNet v2 architecture (**right**).

One of the main innovations in MobileNet is depthwise separable convolution, which is formed by depthwise and pointwise convolutions together. The traditional convolution kernel is split into two kernels by the separable convolution, e.g., a 3×3 kernel is split into two kernels of size 3×1 and a 1×3. This separation reduces the number of operations needed to perform the convolution and is, therefore, much more efficient. Therefore, we obtain approximately the same output as with conventional convolution layers but at a much faster rate with the same accuracy. A regular 3×3 convolution layer is the input layer in a full architecture of MobileNet V1, followed by 13 times the "depthwise separable" convolution block. In between these depthwise separable blocks in the MobileNet architecture, there are no pooling layers. Furthermore, to enhance the efficiency of MobileNet, an improved version of the activation function, named ReLU6, is used, with an upper limit of 6 to the conventional ReLU function. Equation (1) is used to compute the improved activation function, $f(x)$ [38].

$$f(x) = min(6, max(0, x)) \tag{1}$$

where x is the input. One building block structure of MobileNet V1 and V2 are illustrated in Figure 2.

3.2.2. MobilenetV2 Architecture

The next version of MobileNets is named MobileNetV2, and its main building block is shown on the right-hand side of Figure 2. There are three convolution layers in the bottleneck residual block. ***Expansion layer***: The first 1×1 expansion layer expands the input data and increases the number of channels. The data are expanded based on the expansion factor, which is a hyperparameter of an architecture. The default expansion factor is 6. ***Depthwise convolution layer***: The second layer is the depthwise convolution layer, which is the same as in MobileNetV1. ***Projection layer***: In MobileNetV2, contrary to MobileNetV1, the 1×1 pointwise convolution makes the number of channels smaller. This is known as the projection layer. This layer is also called the bottleneck layer since it reduces the amount of data flowing through it [38]. It does the opposite of the projection layer. Some of the parameters of the models used in this work are given in Table 1.

Table 1. Parameters of models used in this work.

CNN Arch.	MACs	Parameters	FPS	mAP/Boxes
MobileNet V1	569	4.24	118	-
MobileNet V2	300	3.47	145	-
SSD [2]	-	-	60	74.3/8732 [1]

[1] MACs: multiply-accumulate operations, and the number of parameters are in millions. [2] Some other parameters used in SSD architecture are grid size, 4×4; zoom, 1.0; aspect ratio, (1.0, 1.0).

3.3. Localization Using Single-Shot Multibox Detector (SSD)

Though CNN-based architectures with fully connected layers can be used to classify images and detect staircases, autonomous traversal requires the knowledge of the location of the staircase as well. For this, we need a bounding box over the staircase to calculate its position, which is required for calculating the target point and the angle of approach. For this purpose, object localization techniques such as single-shot multibox detector (SSD) [31], faster R-CNN [39], YOLO [40], etc., can be used. These use the features extracted from CNN-based architectures to classify and localize objects. Generally, faster R-CNN is the preferred method for object localization due to the best accuracy. However, SSDs have been shown to perform better in most scenarios for large objects, which is the case for staircases [37]. SSD is also extremely fast since it requires only one forward pass for the computation of all bounding boxes. Due to these reasons, SSD is highly appropriate for the current scenario.

The SSD algorithm divides the image using a grid, and each grid cell is responsible for detecting objects in that region of the image, instead of using a sliding window, as in RCNN. Object detection means predicting the class and location of an object within that region. If there is no object present in that grid, we consider it as the background class. For instance, we use a 4×4 grid cell in this work, and each grid cell is able to output the position and shape of the object (staircase, not staircase) it contains.

3.3.1. SSD Architecture

SSD passes the input through multiple convolution layers. These layers progressively decrease in size. Each of these layers generates a fixed set of predictions. This enables predictions of various sizes. In addition to this, SSD uses a set of bounding boxes (a.k.a anchor box) of different dimensions. These anchor boxes are pre-defined, and each one is responsible for size and shape within a grid cell. The SSD architecture allows pre-defined aspect ratios of the anchor boxes to account for the shapes of objects, with some longer and some wider, to varying degrees.

Further, these anchor boxes can be scaled up or down with respect to each grid cell in order to find smaller or larger objects within a grid cell. The zoom parameter is used to specify how much the anchor boxes need to be scaled up or down with respect to each grid cell.

These bounding boxes are applied to the predictions from different layers, which allows for predictions of boxes of different sizes and scales. Although the bounding boxes may not be pixel-perfect, the loss in accuracy is found to be very minimal. However, due to this, the performance is drastically increased.

3.3.2. Loss in SSD Training

Since SSD involves predicting a bounding box along with the class for an object, typical loss computation cannot be followed. Loss in SSD training is a weighted sum of loss due to two aspects: confidence and localization. Equation (2) is used for computing total loss, L.

$$L = \frac{1}{N}(L_{conf} + \alpha L_{loc}) \tag{2}$$

where N is the number of matching boxes and α is the weight term that balances the confidence loss, L_{conf}, and localization loss, L_{loc}. Confidence loss is the loss that occurs

due to classification. This is computed as the softmax of confidence over multiple classes. Localization loss is the loss in predicting the bounding box of the object. This is computed as a smooth, L_1, loss between the predicted box and the marked box.

3.3.3. Loss Optimization

We use the root mean square propagation (RMS prop) [41] algorithm to optimize the total loss during training. Equations (3)–(5) represent the RMS prop algorithm at any time t.

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{3}$$

$$\Delta w^{rms} = \frac{-\eta}{\sqrt{v_t + \epsilon}} \times g_t \tag{4}$$

$$w_{t+1}^{rms} = w_t^{rms} + \Delta w^{rms} \tag{5}$$

where g_t is the gradient along weight w^{rms}. w_t^{rms} is the weight at any time t. v_t is the exponential average of gradients. η is the initial learning rate. β is a hyperparameter to be tuned. ϵ is constant with a value close to zero to avoid dividing by zero errors.

3.4. First-Step Detection

Through SSD and MobileNets, we are able to detect the staircase and predict a bounding box over it. Here, we require knowledge of the first step to determine the target point and angle of approach. We do this by analyzing the staircase inside the bounding box. These values enable a sTetro robot to move toward the detected staircase autonomously. For the detection of steps, the prominent method is Hough transform [42], which detects straight lines. However, the steps of staircases may be curved as well. Traditional contour detection is difficult to handle in this situation. Furthermore, we do not need to detect all the contours in the image, as we require only the first step in the calculations. The proposed method for the first step of staircase detection can be divided into two parts. First, we detect edges using Canny edge detection, which is followed by contour detection.

3.4.1. Edge Detection

To be able to detect the first step in the image, we first need to detect edges present in the image. For this, we use the Canny edge detection algorithm [43] to enhance the accuracy of detected edges; the Canny edge algorithm is divided into four steps: the Gaussian filter step to remove noise, gradient calculation to find the edge pixel candidates, non-max suppression steps to remove the edges with the weak gradient responses, and the hysteresis thresholding step to thin the detected edge lines. This returns an edge representation of the image, i.e., only the pixels that constitute edges are positive. Other pixels are 0.

3.4.2. Contour Detection Algorithm

We propose a contour detection algorithm for determining the first step as the following Algorithm 1. Specifically, all points within a certain distance and certain bounds while giving preference to the points along the direction of the most recent detected points are determined.

Algorithm 1 contains the pseudocode of the algorithm used for first-step detection. Before computation, we generate *canny*, which is the Canny edge representation of the image. We use two functions for this algorithm. The function ***getContour***() returns the contour representing the first step of the staircase. This function uses *canny* to generate a contour representing the first step. It checks the image from bottom to top (represented by i) and left to right (represented by j) directions. If an edge point is detected, it calls ***getContourWithIJ***(), with a contour consisting of only the point, i, j. Finally, it returns the contour only if its horizontal length is greater than a certain threshold, $thres_{filter}$.

We set this to 60% of the width of the staircase during experiments. This is conducted using ***getBounds***(), which obtains the difference between the horizontal bounds of the contour.

Algorithm 1 Contour detection algorithm pseudocode.

```
function GETCONTOUR(canny, thres_filter)
    i = canny.length − 1; j = 0
    contour = []
    while i >= 0 do
        while j < canny.width do
            contour = GETCONTOURWITHIJ(canny, i, j, contour.append([i, j]))
            if getBounds(contour) > thres_filter then return contour
            j = j + 1
        i = i − 1
function GETCONTOURWITHIJ(canny, i, j, contour)
    arr = empty; t = 0
    line = fitLine(contour)
    while t < thres_width do
        y_coord = line.slope × (t) + i
        arr.push(y_coord, j + t, 0)
        for item in arr do
            bottom = item[0] + item[2]
            bottomLimit = item[1] + thres × item[1]
            if canny[bottom][item[1]] > 0 and bottom <= bottomLimit then return
GETCONTOURWITHIJ(canny, bottom, item[1], contour.append([bottom, item[1]]))
            top = item[0] − item[2]
            topLimit = item[1] − thres × item[1]
            if canny[top][item[1]] > 0 and top >= topLimit then return
GETCONTOURWITHIJ(canny, top, item[1], contour.append([top, item[1]]))
            if top < topLimit and bottom > bottomLimit then
                arr.remove(item)
            item[2] = item[2] + 1
        t = t + 1
    while arr.length > 0 do
        for item in arr do
            bottom = item[0] + item[2]
            bottomLimit = item[1] + thres × item[1]
            if canny[bottom][item[1]] > 0 and bottom <= bottomLimit then return
GETCONTOURWITHIJ(canny, bottom, item[1], contour.append([bottom, item[1]]))
            top = item[0] − item[2]
            topLimit = item[1] − thres × item[1]
            if canny[top][item[1]] > 0 and top >= topLimit then return
GETCONTOURWITHIJ(canny, top, item[1], contour.append([top, item[1]]))
            if top < topLimit and bottom > bottomLimit then
                arr.remove(item)
            item[2] = item[2] + 1
    return contour
```

The function ***getContourWithIJ***() finds the remaining contour from the reference point, *i*, *j*, in *canny*. This is merged with the contour detected prior to this point, which is represented by *contour*. This consists of four steps to identify the next point in the contour. First, the equation of the line that fits the previous five points in the contour is generated. The function ***fitLine***() does this using a line-fitting linear regression approach [44]. We also set a bound on the slope of the line. *line* represents this line equation. *line.slope* represents

the slope of this line. Next, we check all points within a range of X coordinates, represented by variable t. This variable is bound by a threshold, $thres_{width}$. Using this, we compute y_{coord}, which is the next possible location of the edge. We push this onto an array, *arr*. Then, for each element in *arr*, we search both above and below the elements. If any of them is part of an edge, then we call ***getContourWithIJ***() with the coordinates of this point. If the top and bottom of this element exist at the boundary, computed using an angle threshold, *thres*, the element is removed from the array.

The working of an algorithm for three different scenarios based on the slope of *line* is shown in Figure 3. In this figure, each box represents one pixel in the image. The value in each box represents the iteration in which they would be visited. The box with value -1 represents the current pixel on which ***getContourWithIJ***() is called. The line represents the slope for the given scenario, with the value of the slope given below each figure. The algorithm clearly gives preference to points along the slope while also examining points in the area bounded by *thres*. The numbers in boxes represent which iteration of the algorithm would evaluate those pixels. For two pixels having the same iteration number, the one that is closest to the slope will be evaluated first.

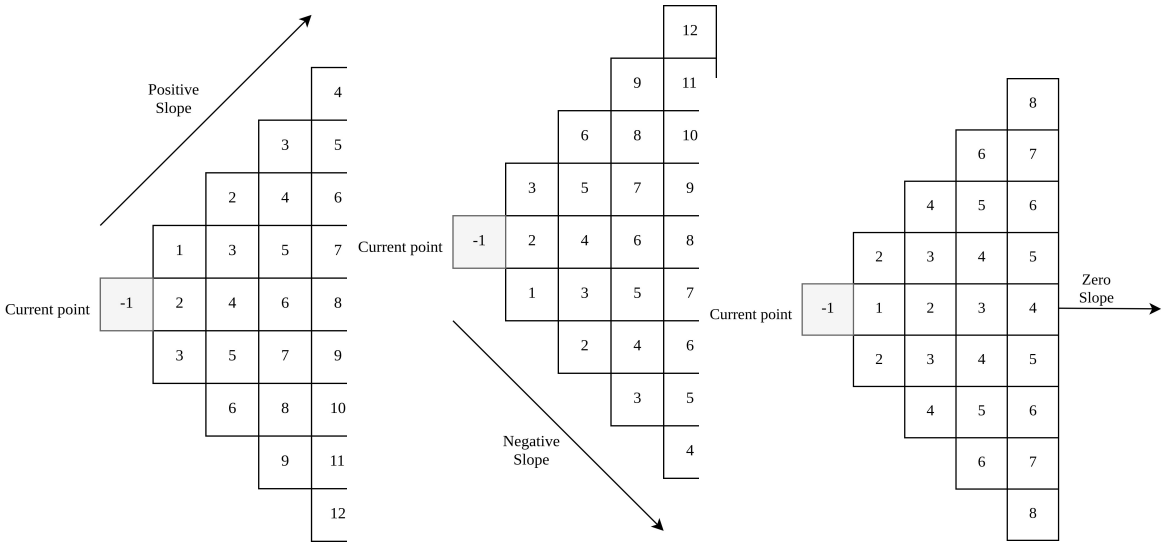


Figure 3. Order of evaluation using the contour algorithm. Positive slope ($+45°$) in (**a**). Negative slope ($-45°$) in (**b**). Zero slope ($0°$) in (**c**).

3.5. Determining the Target Point

Since we have the contour representing the first step, determining the target point, $p_{x,y}$ (approximate central position of the first step of the staircase), is rather simple. We can find n points that are closest to the horizontal center. The distance here is the linear distance between the X-coordinates since there would not be multiple points with the same X-coordinate. We can then fit a line through these points and predict the Y-coordinate of the horizontal center. The horizontal point, x, along the X-coordinate and vertical point, y, along the Y-coordinate give the central target point, $p_{x,y}$. The slope of this line is the angle of approach (θ). This is necessary so that the robot can rotate itself to align with the staircase. In general, negative angles mean the robot is located to the right of the staircase and should rotate clockwise while moving left so that the robot will become aligned with the staircase. Similarly, positive angles mean the robot is located to the left of the staircase and should rotate counter-clockwise while moving right. Angles close to zero mean the robot is already aligned with the staircase. We use a small threshold around zero to accommodate for slight variations, which may occur during detection.

4. Experimental Setup

In this section, we discuss the dataset specification and specification of the models used. The dataset was generated by capturing images of different staircases using an RGB camera. Details of this are discussed below. The weights of the network were initialized to the weights of a network trained on the COCO dataset [45], which is a large dataset containing 80 of the most common classes for labeling. This was conducted so that there would be a faster convergence while training. This also helps in achieving better overall accuracy.

The training of staircase models was performed offline on a server. From the training dataset, different types of staircase models, namely, the left-curved, right-curved, and straight staircase models, were trained separately on a 4× Tesla V100 GPU server running on a Linux platform. During the training phase, the model was trained at a learning rate of 0.004 and used a root mean square propagation optimizer for loss optimization on input images of size 640 × 480 with a batch size of 20. Data augmentation was used to improve model generalization. The training time needed to train a model was approximately 5 h. The testing images are from a separate dataset, and only the detections with a confidence level higher than a threshold of 0.80 were considered. The scope of these tests was to evaluate the efficacy of the developed schemes to detect and localize different types of staircases commonly used in residential as well as commercial multi-floor buildings.

Dataset Preparation

To gather the images in the working environment for the training phase, an RGB camera, with a resolution of 640 × 480 pixels, was fitted on top of a sTetro robot [7], as shown in Figure 4. It was taken into consideration that the images were taken from different staircases at different angles, e.g., approximately from 0 to 180 degrees in front of stairs. It was also considered that images were taken from different distances (approximately 2–10 m) and different positions to generalize the training dataset. Some images were captured with a hand-held camera at the height of the sTetro robot. Most of the images were captured in the Singapore Univerity of Technology and Design (SUTD), e.g., in Figures 3–7, while some images were taken from residential and shopping malls in the city center to ensure diversity in the dataset. The dataset consists of eleven different types, shapes, and materials (metal, wood, and plastic) of staircases, with a total of 206 images, which include both straight and curved staircases. The whole dataset was split in a ratio of 90:10 into training and testing datasets, respectively. Some of the sample images present in the dataset are shown in Figure 5.

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

Figure 4. sTetro robot capturing images from different angles. (**a**) Left of staircase, (**b**) in front of staircase, (**c**) right of staircase.

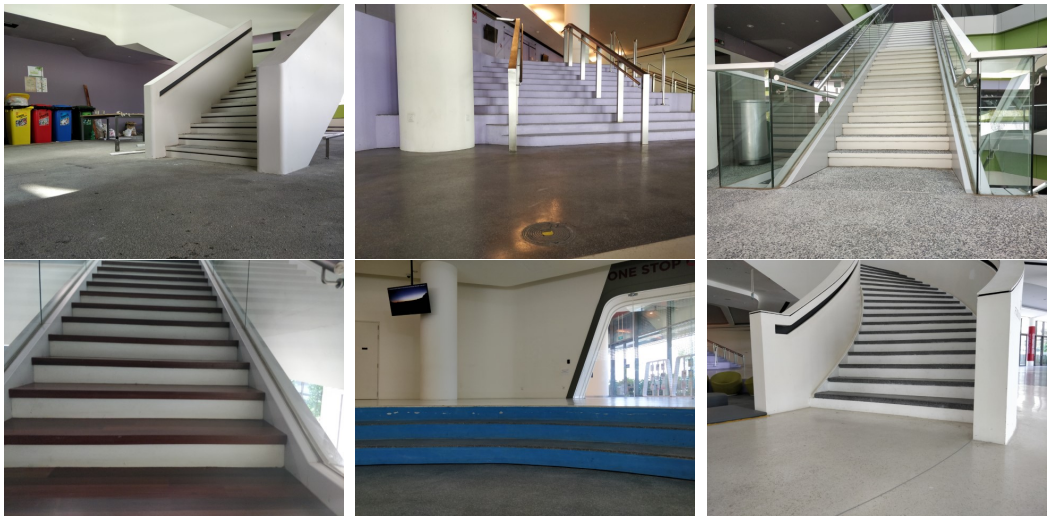


Figure 5. Some of the images present in the dataset.

5. Results Discussion

In this section, we discuss the results obtained on the staircase dataset used. We divide this section into two categories: Staircase detection and first-step detection.

5.1. Staircases Detection

This section discusses results pertaining to the detection of bounding boxes over staircases. Some images of staircases were taken along with images having similar features to staircases. This includes structures with parallel lines, ladders, floors with textures, etc. The model detected almost all staircases correctly. This includes different types of staircases and images taken from different angles and distances. This is shown in Figure 6. The box represents the detected staircase. The confidence of its prediction is shown as a percentage above/below the box. We filtered out boxes with confidence less than 80% for staircase localization. Figure 6a–c show the model is able to detect staircases viewed from different angles. Figure 6d is another staircase with different materials. Figure 6e is a staircase with curved steps, which the model is able to recognize. Figure 6f,g are curved staircases with different orientations. The model is able to detect these as well. In Figure 6h, it can be seen that the staircase is not detected. This can be attributed to bad lighting conditions and the larger distance between the robot and the staircase. The model did not detect ladders as staircases, which is a common issue with traditional methods. Furthermore, the model is able to detect curved staircases, which proves to be a challenge for traditional linear-line-detection-based approaches.

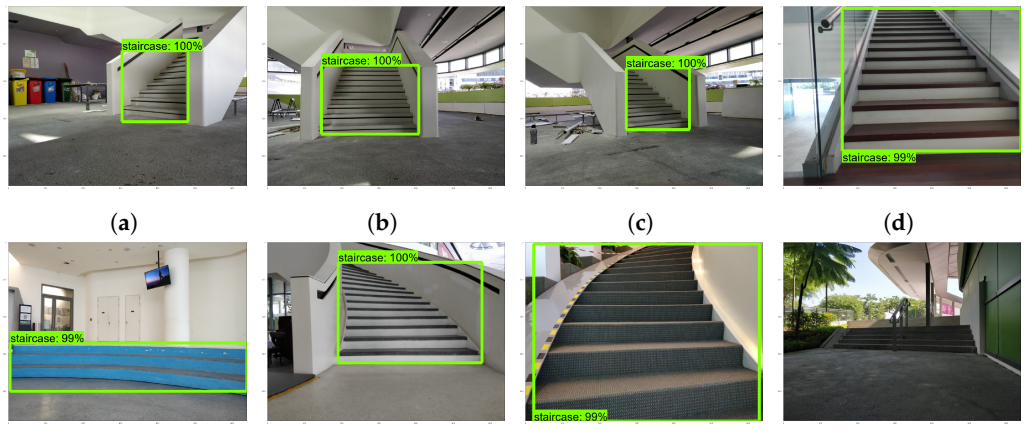


Figure 6. Staircase detection. Staircase detected in (**a**–**g**). Not detected in (**h**).

The proposed model, however, struggled in certain cases to differentiate between stairs and not stairs, where the images are very similar to staircases. Some examples of these scenarios are given in Figure 7. The model is able to differentiate between staircases and tiled floors, which have parallel lines, as given in Figure 7a. This is a very common scenario where traditional classifiers struggle since the floor has parallel lines, which is a common feature used to detect staircases. However, in Figure 7b, it can be observed that the hand railing is detected as a staircase. This can be attributed to the fact that the railings look like a staircase rotated by a 90-degree angle. However, it is easy to fix this false positive detection of hand railings as staircases by taking note of the angle of the lines detected. In Figure 7c, it can be seen that a wall with features similar to a staircase causes the model to detect it falsely as a staircase. This scenario is challenging because the features on the wall look very much similar to the stair's parallel lines. The model also does not detect ladders as staircases. This is shown in Figure 7d. In Figure 7e, the combination of table and chair is also not recognized as staircases, although they have similar features. However, when combined with lines on the floor, the model detects them as a staircase, as shown in Figure 7f. However, if we put a constraint on the height of steps, i.e., a constraint on the distance between two contour lines, this false detection may be avoided. The overall results are given in Table 2.

Hence, it can be seen from the above that recognizing the stairs is much more challenging as compared to common objects in a working environment due to the fact that many features and texture designs on daily life objects have parallel lines that pose difficulty in the detection of stairs in real working environments.

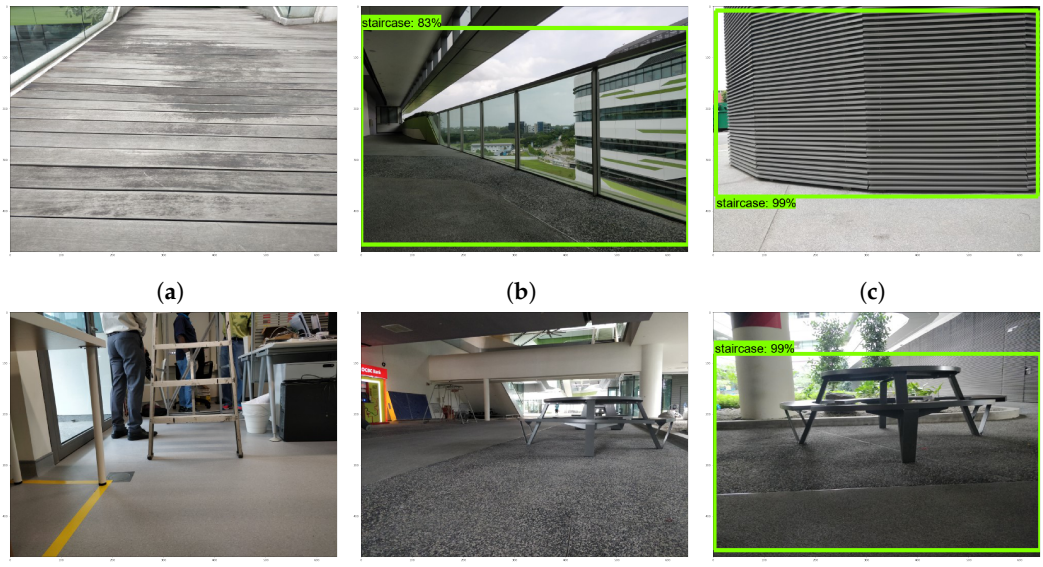


Figure 7. Negative samples for staircase detection. Staircases are not detected in (**a**,**d**,**e**). False detection in (**b**,**c**,**f**).

Table 2. Results of staircase detection. Success (%) indicates the percentage of entries labeled correctly for each class.

	Labeled Staircase	Not Labeled	Total	Success (%)
Staircase	21	1	22	95.45
Not Staircase	7	22	29	75.86
Any Image	28	23	51	-

5.2. First-Step Detection Results

This section discusses results pertaining to the detection of the first step of staircases. We use 60% width of the staircase as the $thres_{filter}$ to filter out small contours. Furthermore, we filter out all vertical edges by using a *thres* parameter of ≈60°. The algorithm is able to detect both the target point, $p_{x,y}$, and the angle of approach, θ, with good accuracy. The accuracy in localizing the middle point of the first step of the staircase was achieved as ±2 cm, and the accuracy in finding the orientation of the middle point was ±1 degree, with respect to the true center point.

The results are shown in Figure 8. Here, the first step of the staircase detected using contour detection is denoted by the line. The cross represents the computed target point, $p_{x,y}$. The detected angle of approach, θ, is also shown below each figure. The algorithm is able to detect these with good accuracy, as shown in Figure 8a–c,e,f. Figure 8d is the image of a textured staircase, where the approach is not very consistent due to the noise generated during Canny edge detection.

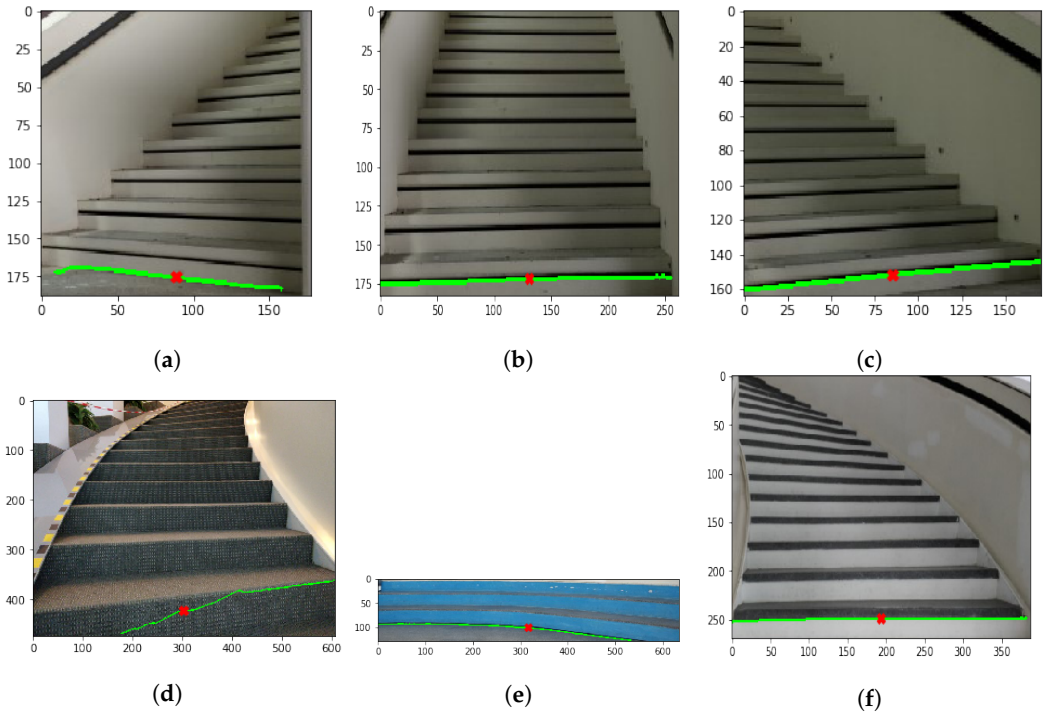
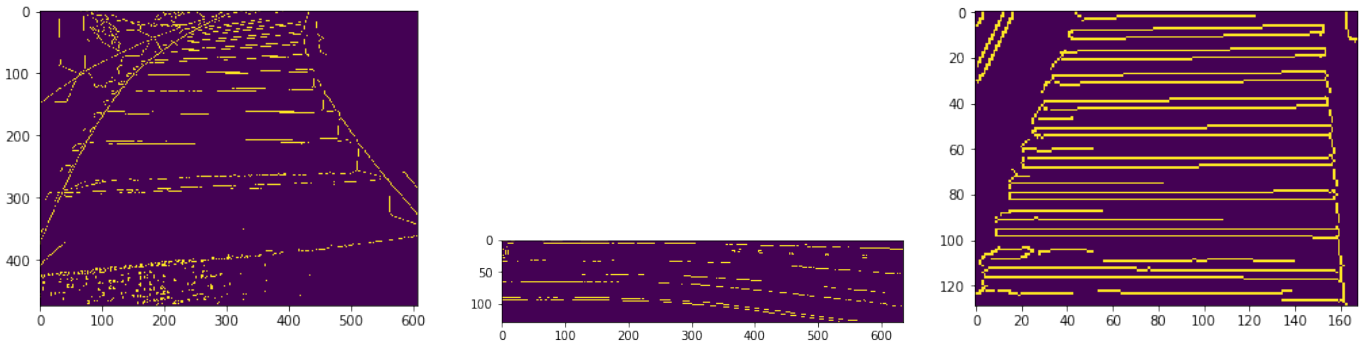


Figure 8. First step detection. (**a**) Detected angle: −6.724 degrees, move left. (**b**) Detected angle: 0.678 degrees, move toward the target point. (**c**) Detected angle: 5.386 degrees, move right. (**d**) Detected angle: 18.336816 degrees, move right. (**e**) Detected angle: −4.586 degrees, move left. (**f**) Detected angle: 0 degrees, move toward the target point.

The edges detected for this image are shown in Figure 9a. Optimal edge detection results are shown in Figure 9b,c.

5.3. Real-Time Performance

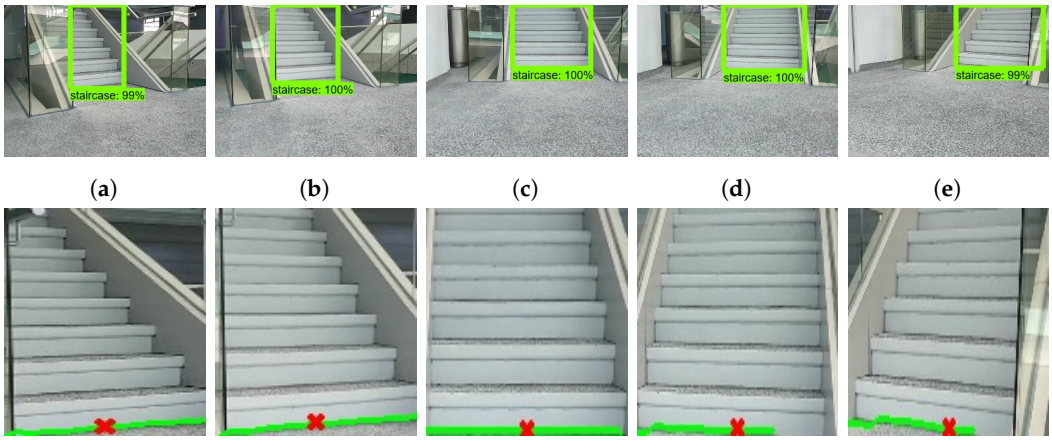
This section details the real-time tests performed using our proposed model. An image stream coming from the sTetro camera is used to determine the movement direction of the robot in real time. The first step of the detected staircase is extracted from the processed images in the server, which, ultimately, returns a direction logic (i.e., angle of approach) to the robot. The details of the target point (position and angle) are sent to the robot to move toward the target point. In this test, the robot moves from the left to the right of the staircase.



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

Figure 9. Canny edge detection. (**a**) Noisy edge detection. (**b**) Curved step-edge detection. (**c**) Straight staircase edge detection.

The detection of the staircase during this test is shown in Figure 10a–e. The first step detection for these figures is shown in Figure 10f–j. The directions predicted are accurate during our testing scenarios, with minor errors occurring on textured staircases. The target point is predicted accurately. However, for real-time applications, the running time of the algorithm is also crucial. This is given in Table 3. The overall approach takes about 150 ms on average, which is feasible for real-time applications. The high computation times usually occur during the first run, which can be attributed to the loading of the model into memory.



(**a**) (**b**) (**c**) (**d**) (**e**)

(**f**) (**g**) (**h**) (**i**) (**j**)

Figure 10. Real-time staircase detection. (**a**–**e**) Staircase detection results. (**f**) The first step detected in (**a**) (instruction: move right). (**g**) The first step detected in (**b**) (instruction: move right). (**h**) The first step detected in (**c**) (instruction: move straight). (**i**) The first step detected in (**d**) (instruction: move left). (**j**) The first step detected in (**e**) (instruction: move left).

Table 3. Performance of proposed approach (data adapted from [24]).

Scenario	Maximum Time (ms)	Average Time (ms)
Staircase detection	2652	71
First-step detection	930	83
Total	2684	155 [1]

[1] Times do not include network delays.

The performance measure definitions are given in [46], and the performance curves are plotted in Figure 11.

In summary, the recall–precision curve is a valuable tool for assessing the performance of a binary classification model across different threshold values. It helps in making informed decisions about the balance between correctly identifying positive instances (recall) and ensuring their accuracy (precision), whereas the F1 score, which is the harmonic mean of precision and recall, is often used to find a balance between these two metrics. The threshold point (0.5) on the curve where the F1 score (0.87) is maximized represents a balanced trade-off between precision (0.75) and recall (0.52). A sample test video of an experiment is referred to in [8].

In [47], Prasanna et al. make use of heterogeneous robots with high-end perception sensor payloads. These kinds of robots are very costly and can be used for inspection tasks only. In [48], Patil et al. use YOLOv3 and achieve an F1 Score of 0.81, as compared to our method with an F1 Score of 0.87. An RGBD camera is used in [49] to localize the steps of a staircase. It is observed that the localization error of the center point increases as the distance from the staircase increases. This is due to the sensitivity of depth error with distance. Errors are >5 cm, as compared to ours, which are <2 cm errors.

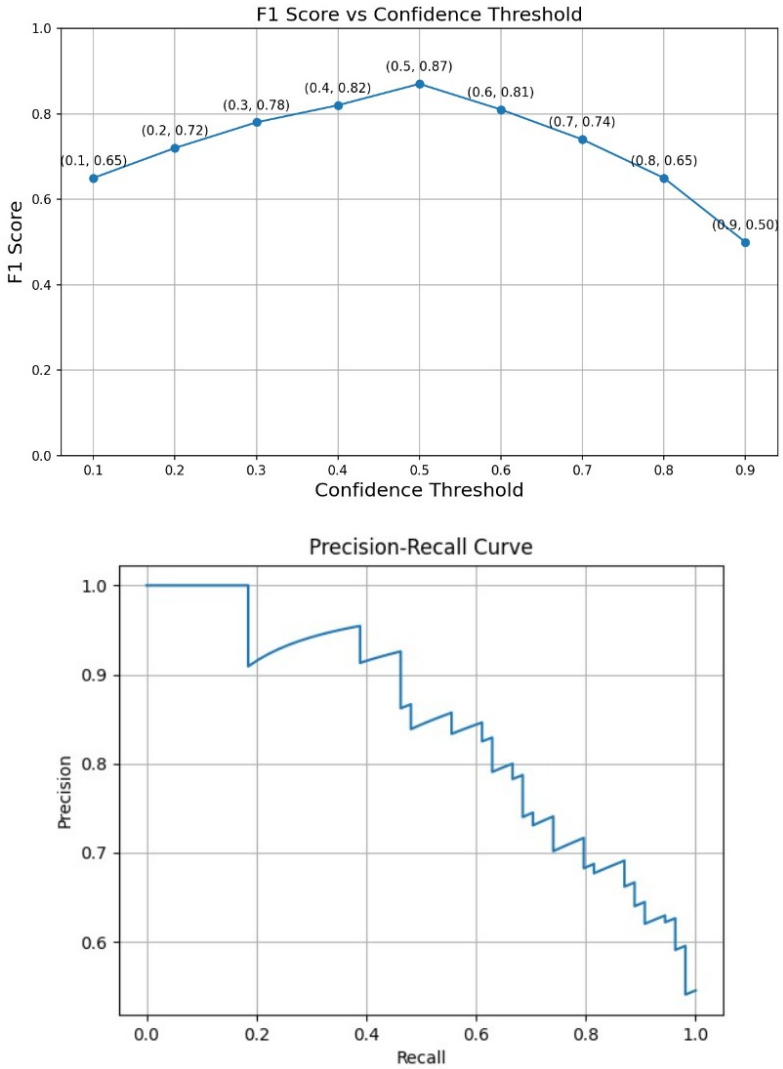


Figure 11. Model performance measure curves: (**top**) F1 score curve versus confidence threshold variation, (**bottom**) precision versus recall curve.

6. Conclusions and Future Work

In this article, we presented a framework for real-time detection and localization of staircases with low-cost embedded systems by incorporating ROS middleware as a computational environment into indoor cleaning robots for use in multi-floor buildings. Low-cost object deep learning detection networks, such as MobileNet and SSD architectures, were used for real-time recognition of staircases in the incoming video stream. In this robotic application, it was crucial to determine the middle point and angle of approach of the first step of the staircase. This was achieved by utilizing the well-known Canny edge detection followed by our proposed contour detection algorithm. Through this algorithm, the robot was able to identify the point close to the center of the first step (target point) and the angle of approach to the target point, which, in turn, are used to determine the direction of movement of the robot. This algorithm aligns the robot to the staircase so that it can reach the middle of the first step and start traversing the staircase. We leveraged the transfer learning technique and trained and tested the proposed model using a custom dataset consisting of 11 images of different staircases captured from a sTetro robot fitted with an RGB camera at the top. Further, the model was tested against objects that have features similar to staircases. The model was able to differentiate between the staircase and non-staircase classes with an accuracy of 95% and determine the target point and the angle of approach to the first step of the staircase with an accuracy of ±2 cm, ±1 degree. We tested this model in real-time scenarios and found that it can be used in slow-moving platforms, like sTetro, which have limited computational resources onboard. This work had limitations in distinguishing among the different shapes and types of staircases, such as straight, curved, spiral, and L-shaped staircases. Currently, it can recognize between stairs and non-stair classes. In the future, this work can be extended to recognize the types of staircases as well. This would allow the robot to switch between movement operational modes according to the type of staircase encountered. Further work can be conducted to implement the classification and localization algorithms onboard the mobile robot.

Author Contributions: Conceptualization, M.I.; Methodology, A.K.L. and A.V.L.; Software, A.K.L. and A.V.L.; Validation, A.V.L.; Formal analysis, M.I.; Investigation, M.I.; Resources, M.R.E.; Data curation, A.K.L.; Writing—original draft, A.K.L.; Writing—review & editing, M.I.; Supervision, M.R.E.; Project administration, M.R.E.; Funding acquisition, M.R.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research is supported by the National Robotics Programme under its National Robotics Programme (NRP) BAU, Ermine III: Deployable Reconfigurable Robots, Award No. M22NBK0054 and also supported by A*STAR under its "RIE2025 IAF-PP Advanced ROS2-native Platform Technologies for Cross sectorial Robotics Adoption (M21K1a0104)" programme.

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

References

1. Prassler, E.; Ritter, A.; Schaeffer, C.; Fiorini, P. A short history of cleaning robots. *Auton. Robot.* **2000**, *9*, 211–226.
2. Jones, J.L.; Mack, N.E.; Nugent, D.M.; Sandin, P.E. Autonomous Floor-Cleaning Robot. US Patent 6883201 B, 26 April 2005.
3. Prabakaran, V.; Elara, M.R.; Pathmakumar, T.; Nansai, S. Floor cleaning robot with reconfigurable mechanism. *Autom. Constr.* **2018**, *91*, 155–165.
4. Choset, H. Coverage for robotics—A survey of recent results. *Ann. Math. Artif. Intell.* **2001**, *31*, 113–126.
5. Biswas, J.; Veloso, M. Wifi localization and navigation for autonomous indoor mobile robots. In Proceedings of the 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA, 3–7 May 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 4379–4384.
6. Huang, A.S.; Bachrach, A.; Henry, P.; Krainin, M.; Maturana, D.; Fox, D.; Roy, N. Visual odometry and mapping for autonomous flight using an RGB-D camera. In Proceedings of the Robotics Research: The 15th International Symposium ISRR, Flagstaff, Arizona, 9–12 December 2011; Springer: Berlin/Heidelberg, Germany, 2017; pp. 235–252.
7. Ilyas, M.; Yuyao, S.; Elara, M.R.; Devarassu, M.; Kalimuthu, M. Design of sTetro: A Modular, Reconfigurable and Autonomous Staircase Cleaning Robot. *J. Sens.* **2018**, *2018*, 8190802.
8. Elara, M.R. sTetro: A Modular Reconfigurable Staircase Cleaning Robot. Available online: https://www.youtube.com/watch?v=552_C1ZoYYI (accessed on 17 August 2021).

9. Munoz, R.; Rong, X.; Tian, Y. Depth-aware indoor staircase detection and recognition for the visually impaired. In Proceedings of the 2016 IEEE International Conference on Multimedia & Expo Workshops (ICMEW), Seattle, WA, USA, 11–15 July 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–6.
10. Duda, R.O.; Hart, P.E. Use of the Hough transformation to detect lines and curves in pictures. *Commun. ACM* **1972**, *15*, 11–15.
11. Sinha, A.; Papadakis, P.; Elara, M.R. A staircase detection method for 3D point clouds. In Proceedings of the 2014 13th International Conference on Control Automation Robotics & Vision (ICARCV), Singapore, 10–12 December 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 652–656.
12. Oßwald, S.; Hornung, A.; Bennewitz, M. Improved proposals for highly accurate localization using range and vision data. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura-Algarve, Portugal, 7–12 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1809–1814.
13. Johnson, A.M.; Hale, M.T.; Haynes, G.C.; Koditschek, D.E. Autonomous legged hill and stairwell ascent. In Proceedings of the 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics, Kyoto, Japan, 1–5 November 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 134–142.
14. Mihankhah, E.; Kalantari, A.; Aboosaeedan, E.; Taghirad, H.D.; Ali, S.; Moosavian, A. Autonomous staircase detection and stair climbing for a tracked mobile robot using fuzzy controller. In Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics, Bangkok, Thailand, 22–25 February 2009; IEEE: Piscataway, NJ, USA, 2009; pp. 1980–1985.
15. 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, Republic of Korea, 9–11 February 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 1–6.
16. 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; IEEE: Piscataway, NJ, USA, 2008; pp. 1806–1811.
17. Se, S.; Brady, M. Vision-based detection of staircases. In Proceedings of the Fourth Asian Conference on Computer Vision ACCV, Taipei, Taiwan, 8–11 January 2000; Volume 1, pp. 535–540.
18. Hesch, J.A.; Mariottini, G.L.; Roumeliotis, S.I. Descending-stair detection, approach, and traversal with an autonomous tracked vehicle. In Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 18–22 October 2010; IEEE: Piscataway, NJ, USA, 2010; pp. 5525–5531.
19. Lu, X.; Manduchi, R. Detection and localization of curbs and stairways using stereo vision. In Proceedings of the Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 18–22 April 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 4648–4654.
20. Ibarra-Zarate, D.; Alonso-Valerdi, L.M.; Chuya-Sumba, J.; Velarde-Valdez, S.; Siller, H.R. Prediction of Inconel 718 roughness with acoustic emission using convolutional neural network based regression. *Int. J. Adv. Manuf. Technol.* **2019**, *105*, 1609–1621.
21. Wang, D.; Han, C.; Wang, L.; Li, X.; Cai, E.; Zhang, P. Surface roughness prediction of large shaft grinding via attentional CNN-LSTM fusing multiple process signals. *Int. J. Adv. Manuf. Technol.* **2023**, *126*, 4925–4936.
22. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444.
23. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117.
24. Le, A.V.; Kyaw, P.T.; Mohan, R.E.; Swe, S.H.M.; Rajendran, A.; Boopathi, K.; Nhan, N.H.K. Autonomous floor and staircase cleaning framework by reconfigurable stetro robot with perception sensors. *J. Intell. Robot. Syst.* **2021**, *101*, 1–19.
25. Ji, S.; Xu, W.; Yang, M.; Yu, K. 3D convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.* **2012**, *35*, 221–231.
26. Weninger, F.; Bergmann, J.; Schuller, B. Introducing currennt: The munich open-source cuda recurrent neural network toolkit. *J. Mach. Learn. Res.* **2015**, *16*, 547–551.
27. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cudnn: Efficient primitives for deep learning. *arXiv* **2014**, arXiv:1410.0759.
28. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 142–158.
29. Howard, A.G.; Zhu, M.; Chen, B.; Kalenichenko, D.; Wang, W.; Weyand, T.; Andreetto, M.; Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv* **2017**, arXiv:1704.04861.
30. Howard, A.; Zhmoginov, A.; Chen, L.C.; Sandler, M.; Zhu, M. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *arXiv* **2018**, arXiv:1801.04381.
31. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. In Proceedings of the Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, 11–14 October 2016; Proceedings, Part I 14; Springer: Berlin/Heidelberg, Germany, 2016; pp. 21–37.
32. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. In Proceedings of the ICRA Workshop on Open Source Software, Kobe, Japan, 12–17 May 2009; Volume 3, p. 5.
33. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105.
34. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 2818–2826.

35. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
36. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
37. Huang, J.; Rathod, V.; Sun, C.; Zhu, M.; Korattikara, A.; Fathi, A.; Fischer, I.; Wojna, Z.; Song, Y.; Guadarrama, S.; et al. Speed/accuracy trade-offs for modern convolutional object detectors. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21 July–26 July 2017; pp. 7310–7311.
38. Hollemans, M. MobileNet Version 2. 2018. Available online: https://machinethink.net/blog/mobilenet-v2/ (accessed on 1 August 2023).
39. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Adv. Neural Inf. Process. Syst.* **2015**, *28*, 91–99.
40. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 779–788.
41. Tieleman, T.; Hinton, G. *Lecture 6.5-Rmsprop, Coursera: Neural Networks for Machine Learning*; Technical Report; University of Toronto: Toronto, ON, Canada, 2012; Volume 6.
42. Ballard, D.H. Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognit.* **1981**, *13*, 111–122.
43. Canny, J. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.* **1986**, *PAMI-8*, 679–698.
44. Edwards, A.L. *An Introduction to Linear Regression and Correlation*; The Correlation Coefficient; W. H. Freeman: New York, NY, USA, 1976; pp. 33–46.
45. MS COCO Dataset. Available online: http://cocodataset.org/home (accessed on 5 July 2018).
46. Liu, X.; Tian, Y.; Yuan, C.; Zhang, F.; Yang, G. Opium poppy detection using deep learning. *Remote. Sens.* **2018**, *10*, 1886.
47. Sriganesh, P.; Bagree, N.; Vundurthy, B.; Travers, M. Fast Staircase Detection and Estimation using 3D Point Clouds with Multi-detection Merging for Heterogeneous Robots. In Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA), London, UK, 29 May–2 June 2023; IEEE: Piscataway, NJ, USA, 2023; pp. 9253–9259.
48. 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; IEEE: Piscataway, NJ, USA, 2019; pp. 159–166.
49. Fourre, J.; Vauchey, V.; Dupuis, Y.; Savatier, X. Autonomous RGBD-based Industrial Staircase Localization from Tracked Robots. In Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 25–29 October 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 10691–10696.