 doi:10.20944/preprints201812.0296.v1

Article

Staircase Recognition and Localization using Convolution Neural Network (CNN) for Cleaning Robot Application

Muhammad Ilyas [1,2], Anirudh Krishna Lakshmanan [1,3], Anh Vu Le [1,4] and Mohan Rajesh Elara [1,*]

[1] Engineering Product Development Pillar, Singapore University of Technology and Design (SUTD), Singapore 487372.
[2] Department of Electrical Engineering, UET Lahore, NWL Campus, Pakistan - 54890.
[3] Department of Computer Science, Birla Institute of Technology and Science (BITS) Pilani, Pilani Campus, Vidyavihar, Rajasthan, India - 333031.
[4] Optoelectronics Research Group, Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, Vietnam - 70000.
* Correspondence: rajeshelara@sutd.edu.sg

Abstract: Multi-floor environments are usually ignored while designing an autonomous robot for indoor cleaning applications. However, for efficient operation in such environments, the ability of a robotic platform to traverse staircases is crucial. Staircase recognition and localization is highly important for planning the traversal on staircase by an autonomous robot. This paper describes a deep learning approach using Convolutional Neural Networks (CNNs) based Robot Operation System (ROS) to staircase recognition and localization. We use an object detection network to detect staircases in 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 the angle of approach to the target point. Experiments are performed with data obtained from images captured on different types of staircases at different viewpoints/angles. Results show that the approach is very accurate in identifying the presence of a staircase in the working environment and is also able to locate the target point with reasonable accuracy.

Keywords: Staircase recognition, Convolutional Neural Networks (CNN), Re-configurable robot, Contour detection.

1. Introduction

Robots are key for automation of various tasks and have revolutionized many fields in the 21^{st} century. They are extremely precise and efficient in doing tasks which otherwise would be difficult or impossible. One such type of robots that are becoming increasingly important are cleaning robots, programmed to work in the indoor environments [1–3]. These robots have vast potential to enhance the 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 would be sold worldwide [4].

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

 doi:10.20944/preprints201812.0296.v1

environments have multiple stories connected with staircase. These staircase may be of different types *e.g.* straight, spiral, L-shaped staircase *etc*. This severely limits the ability of the conventional cleaning robot to be effective in such scenarios. To enable robots to traverse staircases, accurate detection and location of staircases are highly critical. This would enable robots to plan and navigate through such environments, thereby making them much more effective for real-world indoor environments.

The problem of recognizing staircase seems straightforward for humans. However, for robots to be able to detect, recognize and localize the staircases, this task is much more complex. For this, the robot should be able to analyze the incoming images and detect and recognize stair-like structure among many others objects. Further, the knowledge of the location of the staircase is required. Also, knowledge of the angle of approach is required as well for aligning the robot with the staircase in order to start climbing. This makes staircase detection and localization a highly demanding task for autonomous robots.

In this article, we have designed our solution to work with the sTetro platform [7]. sTetro is a 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 bench-marking its performance with a fixed morphology robot. The results indicated that sTetro robot could achieve superior coverage performance through its shape-shifting ability. However, the validation was done by passing manual commands to navigate the robot towards 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 manipulations modules with the sTetro on ROS 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. So, after detecting staircases, the next step is to localize the robot with respect to stairs. For this, we use our own 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 is organized as follows: We first discuss related works and other approaches used for staircase detection. This is followed by details of our proposed approach. Finally, we describe the experimental setup used to test our approach and the results obtained in this work.

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 being detecting parallel lines in the image [8]. Though this algorithm works for many scenarios, they have many limitations. Hough transform [9], the preferred approach for detection of parallel lines, fails to detect staircases which are curved or spiral. Also, 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 their 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 [10] *etc*. These are able to handle approaching staircases from different angles. Even LiDARs have been used for detection of staircases [11–13]. 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 [14–17] and stereo sensors [18].

Object detection and classification by deep learning CNNs have been researched intensively over decades and has set revolutionary era in many applications especially in robotics [19,20]. Deep learning using large neural networks can find the unique features of various objects autonomously, thereby reducing the need of pre-defined kernel based solutions. Since the features of the objects can

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

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 outperformed the conventional methods significantly [21]. 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 [22] and NVIDIA CUDA Deep Neural Network CuDNN [23], enable parallel computing using multiple threads to process large calculations in separate graphics cards with their own Graphics Processing Unit (GPU). Consequently, the training time of these networks has reduced sharply. However, deploying these deep learning approaches in the compactly embedded controllers is still a big challenge for commercial applications. However, with the dawn of IoT devices, using a server-client model to reduce this real-time computation load is more practical.

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 prove to be very difficult for standard algorithms. Also, 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 specifying them in the algorithm. Recently, the advances in object detection, including object localization and objects 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) [24]. The issue with deep learning approaches was the computation power required to run these models in real-time. However, with the recent update to MobileNets [25,26] and the release of SSD (Single Shot multi-box detectors) [27], these models are now capable of running in real time on low cost hardware. We are also computing 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

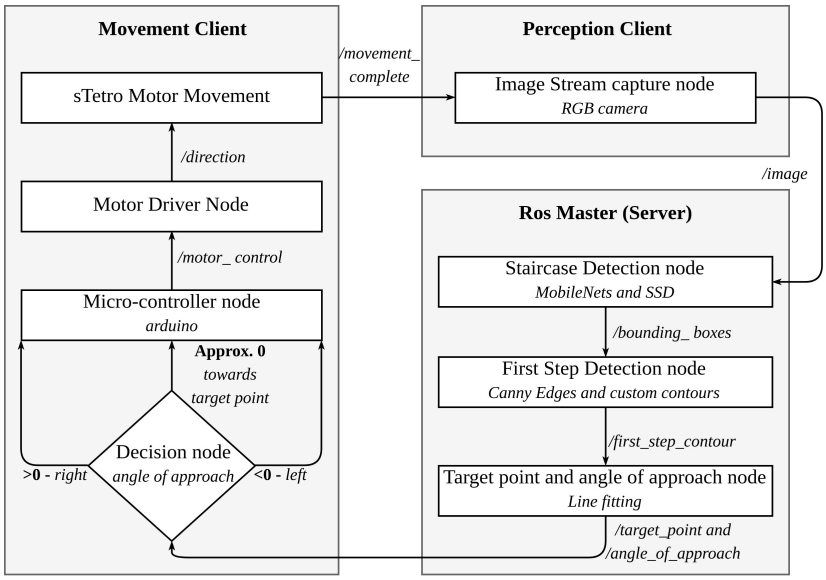

Figure 1. Model Structure

Neural networks have large computation costs. Due to this, computing on 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 on the ROS platform [28]. ROS provides the infrastructure and mechanism for ROS modules. The ROS master installed on remote server monitors the entire

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

ROS system. The ROS-based block diagram of the proposed system using a client- server model is shown in Figure 1. The ROS topics transmitted in ROS network through WiFi or data networks enable the communication between ROS nodes. Specifically, first, the perception client uses an image sensor to extract an image from the image stream and sends it over to the server. The server uses two steps to detect and localize staircases. The first step involves detection of staircases. This includes extracting features from an RGB image using the MobileNet architectures. We then classify and predict a bounding box using the SSD architecture. These rely on the CNN architecture, which is explained below. The second step has two components. First, we detect the first step of the staircase. The first step information is then 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 client responsible for movement. Based on our angle of approach, we decide a movement strategy which is executed by the onboard Micro-controller with the help of a motor. While this happens, the perception client updates the server on the new data. This is repeated until the target point is reached.

3.1. Convolutional Neural Networks (CNN)

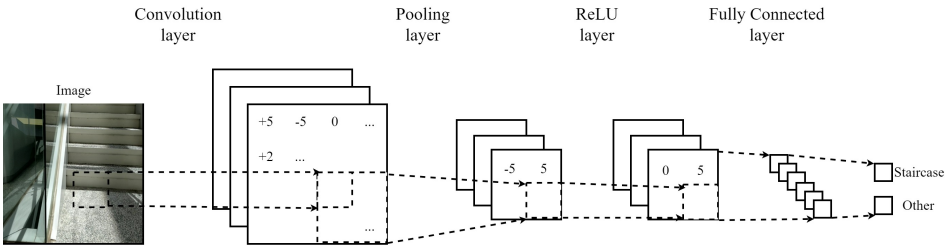

Figure 2. CNN Structure

The CNNs are frequently used in image classification due to their high accuracy. Their structure is the basis for many object detection and image classification networks, including MobileNets and SSD, which we use in our approach. The CNNs consist of four types of layers - Convolution layer, Pooling layer, ReLU layer and a Fully Connected layer. The combination of Convolution layer(s) and Pooling layer(s) extract features from the image. This is fed to the Fully Connected layer(s) that classify the image. Figure 2 shows the structure of a simple CNN.

3.1.1. Convolution layer

This layer involves applying a filter over the input layer. Typically, multiple filters are applied on the input layer. Filters are applied using a sliding window, which slides over the image and applies the filter on the region covered by the window. The input layer is zero padded so that the window can cover the entire input. The filter used is called the weight function. Equation 1 is used to compute the activation function $f(x)$.

$$f(x) = w \times x + b \tag{1}$$

where w represents the filter (weights), b represents the bias and x represents the input.

3.1.2. ReLU layer

This layer applies a ReLU function to the input layer. This layer is typically used after the Convolution layer. This layer is used to remove the negative values which may have been formed from the filter. Equation 2 computes the ReLU activation function $f(x)$.

$$f(x) = max(0, x) \tag{2}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

where x represents the input layer.

3.1.3. Pooling layer

This layer partitions the input layer into non-overlapping regions on which a pooling function is applied. Each region outputs a single value. This reduces the dimension of the input while retaining the important features.

3.1.4. Fully connected layer

This layer is a typical artificial neural network (ANN) layer. It is fully connected to the input layer. Multiple fully connected layers are used together to classify the image. These are used with the final pooling layer as input. The final pooling layer represents the extracted features.

3.2. Feature Extraction using MobileNets

Specialized CNN based architectures like MobileNets [25,26], AlexNet [29], Inception [30], ResNet [31] *etc*. are highly accurate at classifying images. The recently proposed Inception-ResNet [32] 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 [33]. Specifically, we use MobileNetv2 [26], which is an improvement over the standard MobileNet [25]. The basic blocks of both architectures are shown in Figure 3 and Figure 4. The extracted features of MobileNet is used by SSD architecture to classify and localize staircases. The structure of SSD is discussed in the next section.

3.2.1. MobileNet architecture

The core idea behind MobileNet is depth-wise separable convolutions. These replace the typical convolution layers as computing convolutions are highly expensive. Depth-wise separate convolutions first apply a depth-wise convolution on the input. This filters the input data. This is followed by 1×1 convolutions which combine these filters into features. These depth-wise separable layers approximately mimic the function of typical convolution layers but with much faster speed. Another optimization done was to use ReLU6 instead of ReLU, which applies an upper limit of 6 on the activation function. Equation 3 is used to compute the activation function $f(x)$.

$$f(x) = min(6, max(0, x)) \tag{3}$$

where x is the input. Structure of MobileNet v1 is shown in Figure 3.

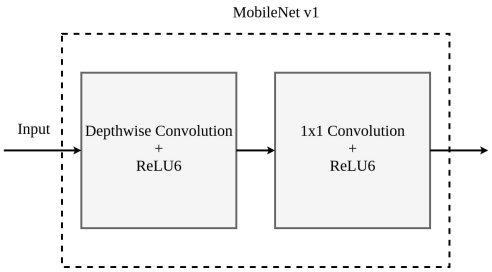

Figure 3. MobileNet v1

3.2.2. MobileNetv2 architecture

MobileNet v2 also uses special convolution structures to replace typical convolution layers. First, there is a expansion layer, which expands the dimension of the input layer tensor. Then the data is passed through a depth-wise convolution layer. The output of this layer is passed through a

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

projection layer, which adds a linear bottleneck to the tensor. This reduces the dimension of the tensor. Structure of MobileNet v2 is shown in Figure 4.

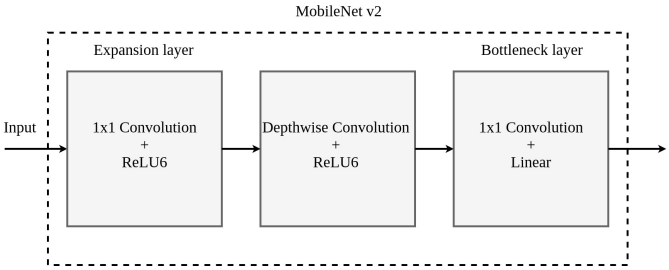

Figure 4. MobileNet v2

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 location of staircase as well. For this, we need a bounding box over the staircase to calculate its position, which is needed for calculating the target point and the angle of approach. For this purpose, object localization techniques such as Single Shot Multi-Box Detector (SSD) [27], faster R-CNN [34], YOLO [35] *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 best accuracy. However, SSDs have been shown to perform better in most scenarios for *large objects*, which is the case for staircases [33]. SSD is also extremely fast since it requires only one forward pass for computation of all bounding boxes. Due to these reasons, SSD is highly appropriate for the current scenario.

3.3.1. SSD architecture

SSD passes the input through multiple convolution layers. These layers progressively decrease in size. Each of these layers generate a fixed set of predictions. This enables predictions of various sizes. In addition to this, SSD uses a set of default bounding boxes of different dimensions. These 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 4 is used for computing total loss L.

$$L = \frac{1}{N}(L_{conf} + \alpha L_{loc}) \tag{4}$$

where N is the number of matching boxes, α is the weight term which 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 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 predicted box and marked box.

3.3.3. Loss optimization

We use the Root Mean Square propagation (RMS prop) [36] algorithm to optimize the total loss during training. Equations 5, 6, 7 represent the RMS prop algorithm at any time t.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

$$v_t = \beta v_{t-1} + (1-\beta) g_t^2 \tag{5}$$

$$\Delta w^{rms} = \frac{-\eta}{\sqrt{v_t + \epsilon}} \times g_t \tag{6}$$

$$w_{t+1}^{rms} = w_t^{rms} + \Delta w^{rms} \tag{7}$$

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 inital learning rate. β is a hyperparameter to be tuned. ϵ is constant with 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. Now, 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 our sTetro robot to move towards the detected staircase autonomously. For detection of steps, the prominent method is Hough transform [37], which detects straight lines. However, steps of staircases may be curved as well. Traditional contour detection is difficulty to handle this situation. Also, we do not need to detect all the contours in the image, as we require only the first step in our calculations. The proposed method for 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 [38].to enhance the accuracy of detected edges, the Canny Edge algorithm is divided into 4 steps - Gaussian filters step to remove the noise, gradient calculation to find the edge pixel candidates, non-max suppression step to remove the edges with the weak gradient responses, hysteresis thresholding step to thin the detected edge lines. This returns a 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

This paper proposes a contour detection algorithm for determining the first step as shown in Algorithm 1. The algorithm detects points within a certain distance and certain bounds while giving preference to the points along the gradient of the most recent detected points. Before computation, we generate *canny*, which is the canny edge representation of the image. We use two functions for this algorithm. 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), left to right (represented by j) direction. 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 done using ***getBounds***(), which gets the difference between horizontal bounds of the contour.

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*. It consists of 4 steps to identify the next point in the contour. First, the equation of the line that fits the previous 5 points in the contour is generated. The function ***fitLine***() does this by line-fitting linear regression approach [39]. 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

 doi:10.20944/preprints201812.0296.v1

Algorithm 1 Contour detection algorithm psuedocode

```
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
            end if
            j = j + 1
        end while
        i = i − 1
    end while
end function
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]]))
            end if
            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]]))
            end if
            if top < topLimit and bottom > bottomLimit then
                arr.remove(item)
            end if
            item[2] = item[2] + 1
        end for
        t = t + 1
    end while
    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]]))
            end if
            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]]))
            end if
            if top < topLimit and bottom > bottomLimit then
                arr.remove(item)
            end if
            item[2] = item[2] + 1
        end for
    end whilereturn contour
end function
```

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

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

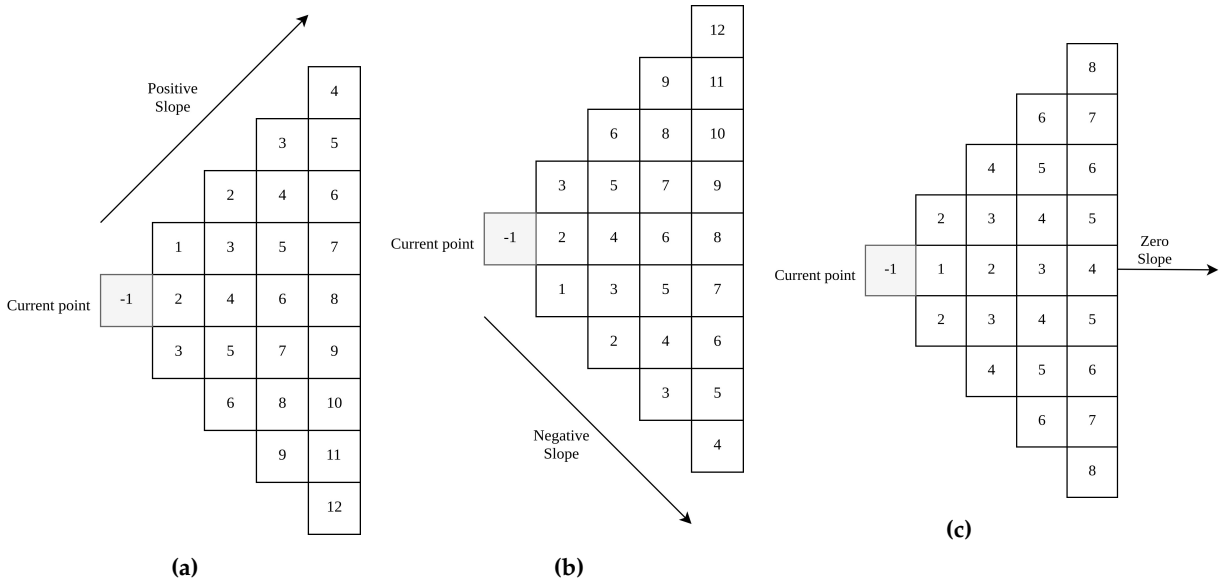

Figure 5. Order of evaluation using the contour algorithm. Positive slope ($+45°$) in (a) . Negative slope ($-45°$) in (b). Zero slope ($0°$) in (c). The angle of 45 is chosen for easy visualization.

this element exists the bound, computed using an angle threshold *thres*, the element is removed from the array.

The working of algorithm for three different scenarios based on slope of *line*, has been shown in Figure 5. 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 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 algorithm would evaluate those pixels. For two pixels having same iteration number, the one which is closest to the slope will be evaluated first.

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 first step of staircase) is rather simple. We can find n points which are closest to the horizontal center. The distance here is linear distance between the X-coordinates, since there would not be multiple points with 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 X-coordinate and vertical point y along Y-coordinate gives 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 to 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 means the robot is already aligned to 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 about the data-set specification and specification of the models used. The data-set was generated by capturing images of different staircases using a RGB camera. Details

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

(a) (b) (c)

Figure 6. sTetro robot capturing images from different angles. (a) left of staircase, (b) in front of staircase, (c) right of staircase.

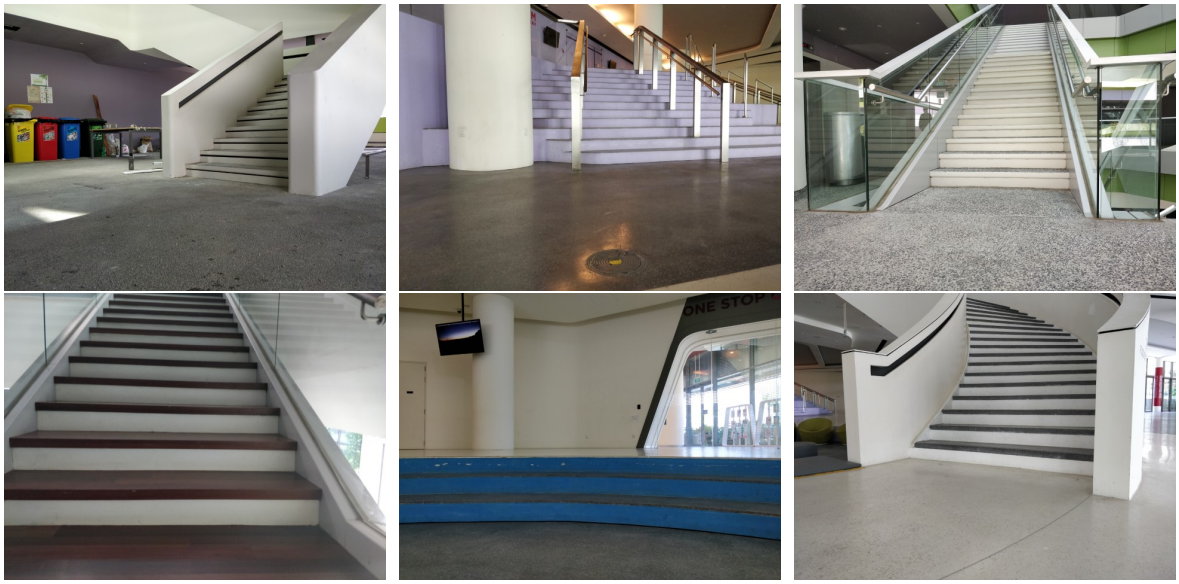

Figure 7. Some of the images present in the dataset.

of this is discussed below. The weights of the network were initialized to the weights of a network trained on the COCO data-set [40], which is a large data-set containing 80 most common classes for labelling. This was done so that there would be a faster convergence while training. This also helps in achieving a better overall accuracy. We used a batch size of 20 while training. We used a RMS prop optimizer for loss optimization with initial learning rate 0.004 and a decay factor of 0.9.

4.1. Obtaining Data-set

The sTetro robot [7] is used to capture images of the working environment. A RGB camera is fitted on top of sTetro robot, which is used to take the images, as shown in Figure 6. Initially the images are taken from different staircases at different angles ranging from approximately 0 to 180 in front of stairs. Images from different distances and different positions were also taken. All images have a resolution 640×480 pixels. The data-set consists of 1025 images. The data-set contains images of 11 different staircases, which include both straight and curved staircases. Staircases with curved steps were also taken. The data-set also has staircases with different materials. A $90 : 10$ split was taken for training and testing respectively during the training phase. Figure 7 contains some of the images present in the data-set.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

The final results are obtained from a set of 102 images not present in the data-set. Approximately half of these images contain staircases in them. The other half includes images of corridors, open areas and other common environments for cleaning robots where objects with features similar to staircases may be present.

5. Results Discussion

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

5.1. Staircases Detection

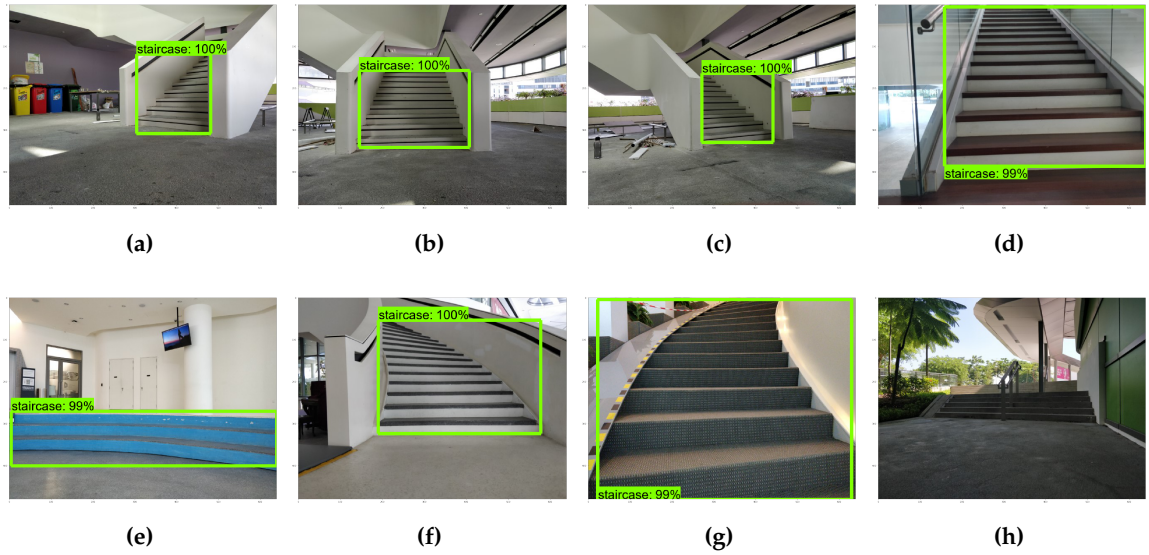

Figure 8. Staircase detection. Staircase detected in (a), (b), (c), (d), (e), (f) and (g). Not detected in (h).

This section discusses results pertaining to 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 8. The box represents detected staircase. The confidence of its prediction is shown as a percentage above/ below the box. We filter out boxes with confidence less than 80% for staircase localization. Figures 8a, 8b and 8c show the model is able to detect staircases viewed from different angles. Figure 8d is another staircase with different material. Figure 8e is a staircase with curved steps, which the model is able to recognize. Figures 8f and 8g are curved staircases with different orientations. The model is able to detect these as well. In Figure 8h, the staircase is not detected. This can be attributed to bad lighting conditions and larger distance between the robot and the staircase. The model *did not* detect ladders as staircases, which is a common issue with traditional methods. Also, the model is able to detect curved staircases, which proves to be a challenge for traditional linear line detection based-approaches.

However, the model did fail on certain cases where the images were very similar to staircases. Some examples of these scenarios are given below in Figure 9. The model is able to differentiate between staircases and tiled floors, which have parallel lines. One such example is Figure 9a. 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. Figure 9b is detected as a staircase. This can be attributed to the fact that the railings look like a staircase rotated by 90. However, it is easy to differentiate between staircases and railings by considering the angle of lines detected. In Figure 9c, the wall has

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

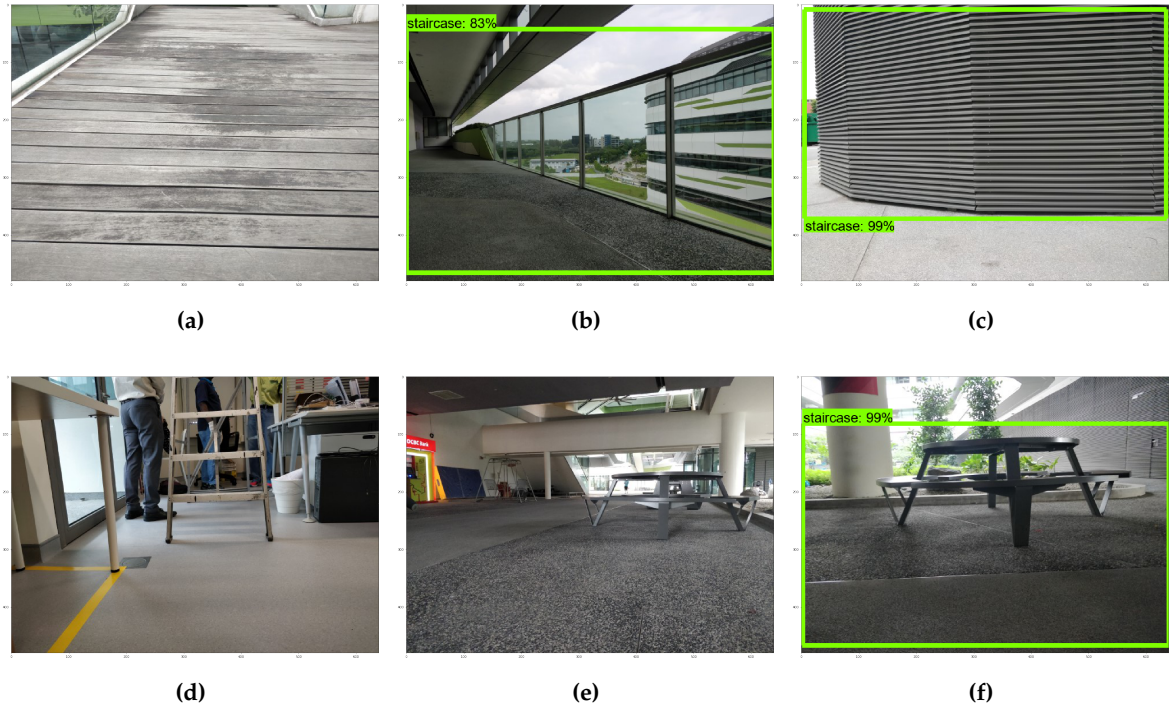

Figure 9. Negative samples for staircase detection. Staircases are not detected in (a), (d) and (e). False detection in (b), (c) and (f).

features similar to a staircase, which causes the model to detect it as a staircase. The model also does not detect ladders as staircases. This is shown in Figure 9d. In Figure 9e, 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 9f. However, if we put a constraint on the height of steps *i.e* a constraint on distance between two contour lines, this false detection may be avoided. The overall results are given in Table 1.

Table 1. Results of staircase detection.

	Labelled Staircase	**Not Detected**	**Total**	**Precision (%)**
Staircase	42	2	44	95.45
Not Staircase	14	44	58	75.86
Any Image	56	46	102	-

5.2. First step detection results

This section discusses results pertaining to detection of first step of staircases. We use 60% width of staircase as the $thres_{filter}$ to filter out small contours. Also, we filter out all vertical edges by using a *thres* parameter of $\approx 60^{\circ}$. This value is obtained from grid searching through possible values for *thres* and selecting the one with best accuracy. The algorithm is able to detect both the target point $p_{x,y}$ and the angle of approach θ with good accuracy. The results are showcased in Figure 11. 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 Figures 11a, 11b, 11c, 11e, 11f. Figure 11d is the image of a textured staircase, where the approach is not very consistent due to the

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

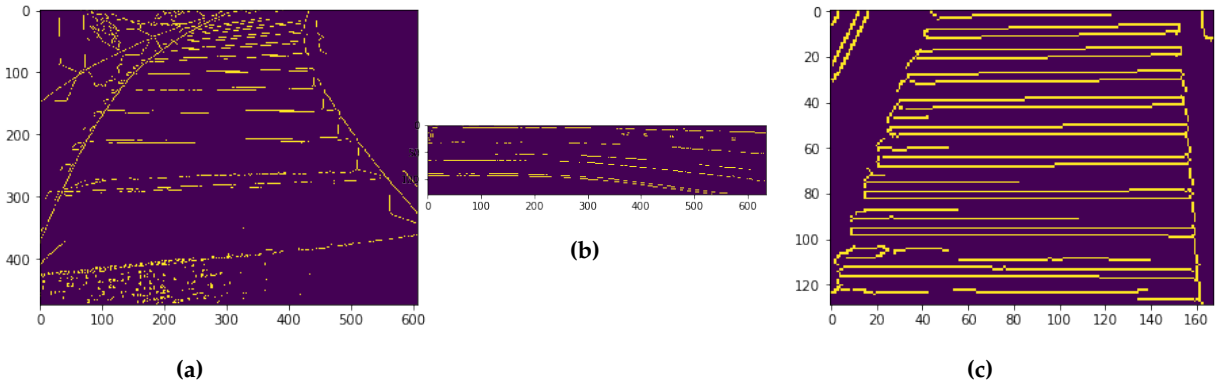

Figure 10. Canny edge detection. (a) Noisy edge detection. (b) Curved step edge detection. (c) Straight staircase edge detection.

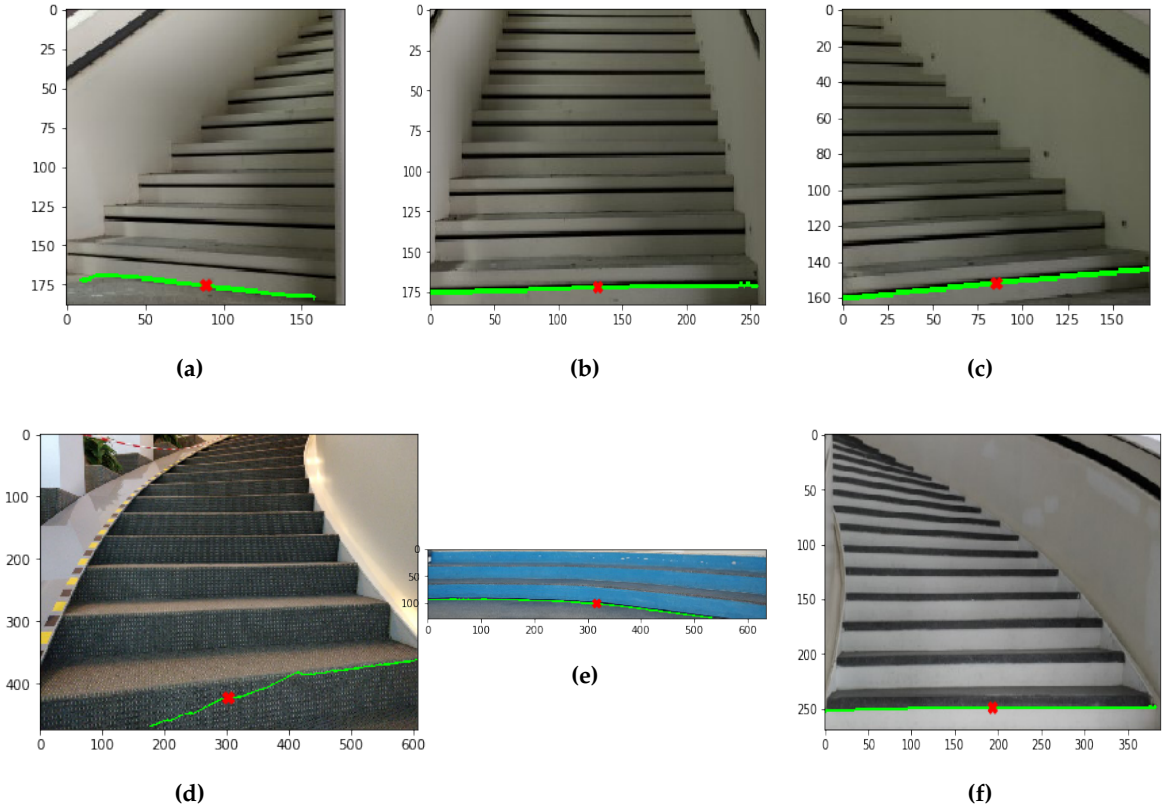

Figure 11. First step detection. (a) Detected angle: -6.724, move left. (b) Detected angle: 0.678, move towards target point. (c) Detected angle: 5.386, move right. (d) Detected angle: 18.336816, move right. (e) Detected angle: -4.586, move left. (f) Detected angle: 0, move towards target point.

noise generated during Canny Edge detection. The edges detected for this image is shown in Figure 10a. Optimal edge detection results are shown in Figures 10b and 10c.

5.3. Real-time Performance

This section details the real-time tests performed using our proposed model. We used an image stream captured from the sTetro robot to determine movement direction in real-time. The robot was

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 25 December 2018 doi:10.20944/preprints201812.0296.v1

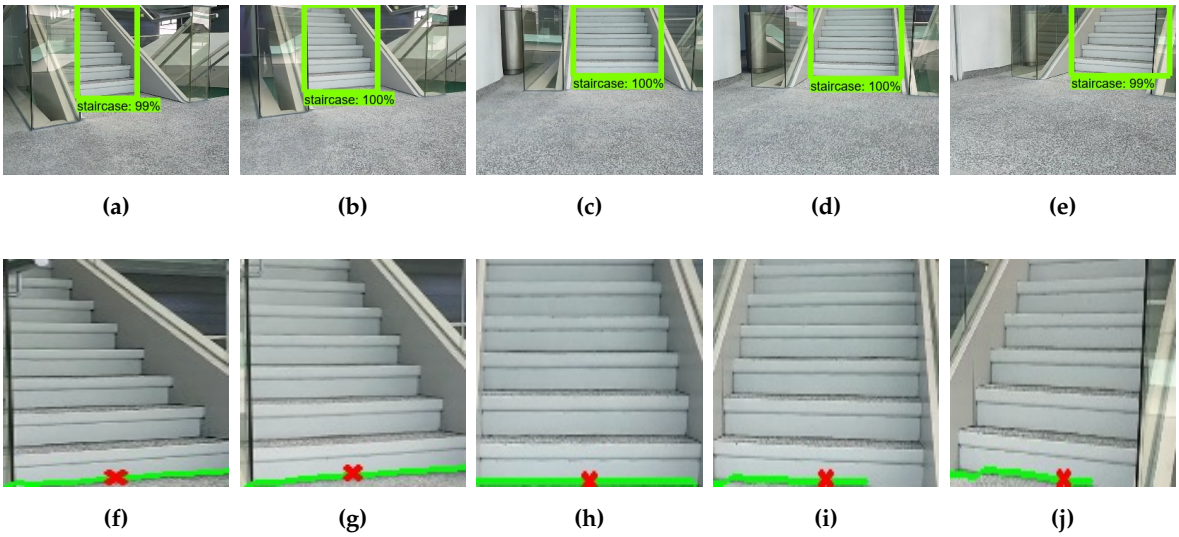

Figure 12. Real time staircase detection. (a), (b), (c), (d) and (e) are staircase detection results. (f) is the fist step detected in (a) (instruction: move right). (g) is the fist step detected in (b) (instruction: move right). (h) is the fist step detected in (c) (instruction: move straight). (i) is the fist step detected in (d) (instruction: move left). (j) is the fist step detected in (e) (instruction: move left).

Table 2. Performance of proposed approach. *Times do not include network delays*.

Scenario	Maximum Time(ms)	Average Time(ms)
Staircase Detection	2652	71
First Step Detection	930	83
Total	2684	155

able to detect staircases by sending frames after every interval to a server where the server processed and returned a direction logic (*i.e* angle of approach) to the robot. Also, the details of the target point was sent so that the robot can move towards this location. In this test, the robot is moving from the left to the right of staircase. Detection of staircase during this test is shown in Figures 12a, 12b, 12c, 12d and 12e. The first step detection for these Figures are shown in Figures 12f, 12g, 12h, 12i and 12j respectively. 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 2. The overall approach takes about 150*ms* on an 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 model into memory.

6. Conclusion

In this paper, we described an approach to staircase recognition and localization. We first used a deep learning model to recognize a staircase in the environment. This was done using an object detection network consisting of MobileNet and SSD architectures. We then used canny edge detector followed by our own contour detection algorithm for first step detection of staircase. Through this, we identified the target point, a point close to the center of the first step and the angle of approach, which is used to determine the direction to staircase. This scheme allows to align the robot to the staircase, so that it can start traversing the staircase. We trained and tested our proposed model using our own data-set consisting of images of 11 different staircases captured with sTetro robot from different viewpoints. We also tested our model against images that have features similar to staircases.

The model is able to detect staircase and determine the target point and the angle of approach to the first step of the staircase with good accuracy. We also tested this model for real-time scenarios and found that it can be used in slow moving platforms like sTetro. In future, this work can be extended to recognize the type of staircases also, like straight, curved, spiral *etc*. This would allow the robot to switch between operation modes according to the type of staircase encountered.

7. Acknowledgement

This research was supported by Grant No. RGAST1702 from National Robotics Program Office, Singapore (NRPO) to the Engineering Product Development (EPD) at Singapore University of Technology and Design (SUTD).

References

1. Prassler, E.; Ritter, A.; Schaeffer, C.; Fiorini, P. A short history of cleaning robots. *Autonomous Robots* **2000**, *9*, 211–226.
2. Jones, J.L.; Mack, N.E.; Nugent, D.M.; Sandin, P.E. Autonomous floor-cleaning robot, 2017. US Patent App. 15/451,817.
3. Prabakaran, V.; Elara, M.R.; Pathmakumar, T.; Nansai, S. Floor cleaning robot with reconfigurable mechanism. *Automation in Construction* **2018**, *91*, 155–165.
4. Choset, H. Coverage for robotics–A survey of recent results. *Annals of mathematics and artificial intelligence* **2001**, *31*, 113–126.
5. Biswas, J.; Veloso, M. Wifi localization and navigation for autonomous indoor mobile robots. Robotics and Automation (ICRA), 2010 IEEE International Conference on. IEEE, 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 *Robotics Research*; Springer, 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. Journal Of Sensors (in press).
8. Munoz, R.; Rong, X.; Tian, Y. Depth-aware indoor staircase detection and recognition for the visually impaired. Multimedia & Expo Workshops (ICMEW), 2016 IEEE International Conference on. IEEE, 2016, pp. 1–6.
9. Duda, R.O.; Hart, P.E. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM* **1972**, *15*, 11–15.
10. Sinha, A.; Papadakis, P.; Elara, M.R. A staircase detection method for 3D point clouds. Control Automation Robotics & Vision (ICARCV), 2014 13th International Conference on. IEEE, 2014, pp. 652–656.
11. Oßwald, S.; Hornung, A.; Bennewitz, M. Improved proposals for highly accurate localization using range and vision data. Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on. IEEE, 2012, pp. 1809–1814.
12. Johnson, A.M.; Hale, M.T.; Haynes, G.; Koditschek, D.E. Autonomous legged hill and stairwell ascent. Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium on. IEEE, 2011, pp. 134–142.
13. 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. Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on. IEEE, 2009, pp. 1980–1985.
14. Hernández, D.C.; Jo, K.H. Stairway tracking based on automatic target selection using directional filters. Frontiers of Computer Vision (FCV), 2011 17th Korea-Japan Joint Workshop on. IEEE, 2011, pp. 1–6.
15. Cong, Y.; Li, X.; Liu, J.; Tang, Y. A stairway detection algorithm based on vision for ugv stair climbing. Networking, Sensing and Control, 2008. ICNSC 2008. IEEE International Conference on. IEEE, 2008, pp. 1806–1811.
16. Se, S.; Brady, M. Vision-based detection of staircases. Fourth Asian Conference on Computer Vision ACCV, 2000, Vol. 1, pp. 535–540.

17. Hesch, J.A.; Mariottini, G.L.; Roumeliotis, S.I. Descending-stair detection, approach, and traversal with an autonomous tracked vehicle. Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on. IEEE, 2010, pp. 5525–5531.
18. Lu, X.; Manduchi, R. Detection and localization of curbs and stairways using stereo vision. Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. IEEE, 2005, pp. 4648–4654.
19. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *nature* **2015**, *521*, 436.
20. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural networks* **2015**, *61*, 85–117.
21. Ji, S.; Xu, W.; Yang, M.; Yu, K. 3D convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence* **2013**, *35*, 221–231.
22. Weninger, F.; Bergmann, J.; Schuller, B. Introducing currennt: The munich open-source cuda recurrent neural network toolkit. *The Journal of Machine Learning Research* **2015**, *16*, 547–551.
23. Chetlur, S.; Woolley, C.; Vandermersch, P.; Cohen, J.; Tran, J.; Catanzaro, B.; Shelhamer, E. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* **2014**.
24. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Region-based convolutional networks for accurate object detection and segmentation. *IEEE transactions on pattern analysis and machine intelligence* **2016**, *38*, 142–158.
25. 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 preprint arXiv:1704.04861* **2017**.
26. Sandler, M.; Howard, A.; Zhu, M.; Zhmoginov, A.; Chen, L.C. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *arXiv preprint arXiv:1801.04381* **2018**.
27. Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S.; Fu, C.Y.; Berg, A.C. Ssd: Single shot multibox detector. European conference on computer vision. Springer, 2016, pp. 21–37.
28. Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: an open-source Robot Operating System. ICRA workshop on open source software. Kobe, Japan, 2009, Vol. 3, p. 5.
29. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012, pp. 1097–1105.
30. Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; Wojna, Z. Rethinking the inception architecture for computer vision. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
31. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.
32. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, inception-resnet and the impact of residual connections on learning. AAAI, 2017, Vol. 4, p. 12.
33. Huang, J.; Rathod, V.; Sun, C.; Zhu, M.; Korattikara, A.; Fathi, A.; Fischer, I.; Wojna, Z.; Song, Y.; Guadarrama, S.; others. Speed/accuracy trade-offs for modern convolutional object detectors. IEEE CVPR, 2017.
34. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 2015, pp. 91–99.
35. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779–788.
36. Tieleman, T.; Hinton, G. Lecture 6.5-RMSProp, COURSERA: Neural networks for machine learning. *University of Toronto, Technical Report* **2012**.
37. Ballard, D.H. Generalizing the Hough transform to detect arbitrary shapes. *Pattern recognition* **1981**, *13*, 111–122.
38. Canny, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence* **1986**, pp. 679–698.
39. Edwards, A.L. *An introduction to linear regression and correlation*; WH Freeman San Francisco, 1976.
40. MS COCO dataset. http://cocodataset.org/#home. Accessed: 2018-07-05.