Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

Wydział Informatyki

Praca dyplomowa

Planar segment-based global localization for autonomous agents

Globalna lokalizacja agenta w oparciu o segmenty płaszczyzn

Autor:	**Jan Rodzoń**
Kierunek:	**Informatyka**
Opiekun pracy:	**prof dr hab. inż. Bogdan Kwolek**

Kraków, 2024

Składam szczególne podziękowania mojemu Promotorowi, prof. dr hab. inż. Bogdanowi Kwolek za życzliwość, wszechstronne wsparcie, cierpliwość, cenne uwagi merytoryczne oraz poświęcony czas.

Streszczenie

Niniejsza praca bada obecne rozwiązania i architektury sieci neuronowych pod kątem lokalizacji agenta w oparciu o rozpoznawanie segmentów płaszczyzn na zdjęciach RGB. W ramach pracy wybrano oraz sprawdzono zastosowanie kilku najnowszych technologii opartych o sieci neuronowe w radzeniu sobie z rozpoznawaniem segmentów płaszczyzn w różnych warunkach życia codziennego.

Abstract

This thesis examines current neural network solutions and architectures in terms of agent localization based on recognition of plane segments from RGB images. The work selects and tests the application of several state-of-the-art technologies based on neural networks in dealing with the recognition of plane segments in various everyday conditions.

Contents

List of Figures

List of Tables

List of Listings

1. Preface

We live in the era of automation. Everywhere robots are taking over the positions, which require not only repetitive actions but also more complicated tasks. In this environment, there are more and more machines that are operating within some contained spaces like commercial buildings or warehouses. There is also a growing market for indoor drone usage. Because of that, there is a need for a means of autonomous localization of these machines, which we will call generically agents in this work. Recent advances in computer vision and artificial intelligence come in handy and provide some models for localization based on planar segments.

1.1. Motivation

Unfortunately, when one tries to use or apply the latest state-of-the-art models and algorithms there are a lot of obstacles to it—starting from missing environment requirements, through outdated libraries, which have dropped backward compatibility, and finishing on overfitted models, which are good only within specific circumstances and are not applicable in ordinary indoor applications. This thesis aims to contribute to the field of agent localization and plane segmentation by reviewing the current development of plane segmentation and measuring its potential on data that is completely different from the training or testing one. Additionally, some of the latest models are difficult to get started. In this work, I also try to ease the workload required to get these projects going for future contributors or researchers.

1.2. Content of this work

The content of the chapters in this thesis is organized as follows:

1. **Chapter 2: Plane Segmentation Development** 2

   This chapter goes through the recent development in plane segmentation. It compares different approaches and techniques to tackle this problem bringing up relevant literature.

2. **Chapter 3: Problem Description** 3

   This chapter explains the problem of agent localization and the difficulties in applying it in everyday use. It also covers the obstacles of running the latest state-of-the-art projects to deepen the research in this matter.

3. **Chapter 4: Implementation** 4

   This chapter shows the actions taken to get the related projects up and running and to evaluate their usefulness in indoor applications, including data preparation and processing.

4. **Chapter 5: Results Evaluation** 5

   This chapter presents and compares the results of the selected models. It also shows how to get the related projects going after the contribution.

5. **Chapter 6: Conclusions** 6

   This chapter concludes the results and shows potential directions for future work.

2. Plane Segmentation Development

2.1. RGB-D Approach

As the whole artificial intelligence field has been developing rapidly over the past few years, plane segmentation is no exception. In the previous century, it was almost unimaginable to parse the film in real time and interpret the frames. Furthermore, it seemed that the depth component was necessary to find plane segments in the pictures. Since the RBG-D sensors were only available in the form of expensive machines like time-of-flight cameras or scanning 3-D laser range finders, [1] there were no possibilities for a broad usage of this technology, and thus, it saw little interest. Nevertheless, there were multiple approaches to recognizing planar sections in the pictures.

2.1.1. Rabbani Classification

Rabbani [2] categorized contemporaneous plane segmentation algorithms into three main categories:

1. **Edge-based segmentation**

   This approach comprises two stages:

   - Edge detection

     It involves scanning the depth map for points in which there are some changes in the local surface properties (like gradients or normals) above a designated threshold.

   - Point grouping

     This part groups together points inside borders detected in the previous step.

   One of the first successful applications of this method was done by Bhanu et al. [3] Another more recent work with promising results was published by Sappa et al. [4]

2. **Surface-based segmentation**

   Algorithms of this type try to merge points with similar local surface properties. Generally, there are two approaches:

   - Bottom-up

     It begins by selecting some starting points and expanding from them using some predefined similarities. It is worth noting that the selection of these initial points is crucial as the results are dependent on them - i.e., you don't find segments without starting points within them.

   - Top-down

     It starts by associating one big plane with all of the points in the picture. Then, it cuts it into smaller pieces until it achieves a predefined fitting threshold.

   One of the most prominent uses of this approach is published by Xiang et al. [5] They utilize the bottom-up approach, as it is far more often used than the top-down technique.

3. **Scanline-based segmentation**

   This approach treats rows or columns of pixels as scanlines. It takes advantage of the fact that each scanline projection over a 3D surface creates a 3D line. Ultimately, the technique consists of two stages:

   - Line segments extraction

     This stage involves processing scanlines to find line segments.

   - Line segments grouping

     In this step, the extracted line segments are grouped with the neighboring ones based on their similar surface properties to form plane segments.

   One of the first publications regarding this technique is made by Jiang et al. [6] giving a groundbreaking performance at the time. Natonek [7] utilized a similar approach for robots just two years later having promising results.

As we can see, these various techniques try to tackle the plane segmentation problem from different quarters, but most implementations require a large amount of computation. This fact renders them inconvenient for real-time agent localization, because they are not able to process multiple frames per second.

2.1.2. RGB-D Affordability

Plane segmentation started drawing more attention at the beginning of the 2010s with the availability of inexpensive RGB-D sensors (e.g. Microsoft Kinect). [1] With the lower price came higher availability and more promising research results. Multiple algorithms arose during the popularization of this topic. One of the early works with successful SLAM with Kinect 3D sensor was presented by Taguchi et al. [8] They have incorporated the RANSAC [9] algorithm (a widely used technique in the PCL library [10]) with planes and points as primitives. Then, multiple other promising methods arose, like:

- Published by Salas-Moreno et al. [11], which uses PCA to find plane candidates
- Developed by Michael Kaes [12], which uses infinite planes and least-squares for mapping, combined with minimal representation.
- Brought by Hsiao et al. [13], which uses keyframe-based approach and is viable on CPU-only devices.

All of them utilize some variation of plane extraction from the depth property with a veraiety of optimizations.

2.1.3. RGB-D To Point Cloud Mapping

For every RGB-D algorithm, it is essential to correctly calculate the real point coordinates based on the pixel location in the image. It defines the mapping between the camera and the world. It includes the focal length and the principal point (often the image center). To perform the computation, we need to know the camera matrix (i.e., camera intrinsic parameters):

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

where:

- f_x is the focal length on x-axis
- f_y is the focal length on y-axis
- c_x is the principal point x-axis coordinate
- c_y is the principal point y-axis coordinate

Then, for each pixel with 2D coordinates $p(u, v)$ we can calculate the world point $P(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ as follows:

$$\begin{cases} \mathbf{X} = \frac{(u-c_x)\cdot\mathbf{Z}}{f_x} \\ \mathbf{Y} = \frac{(u-c_x)\cdot\mathbf{Z}}{f_x} \\ \mathbf{Z} = d_p \end{cases} \tag{2.2}$$

where:

- d is the depth component in RGB-D image for pixel $p(u, v)$
- u is the pixel p x-axis coordinate
- v is the pixel p y-axis coordinate

The pixel projection mapping can be visible in the Figure 2.1

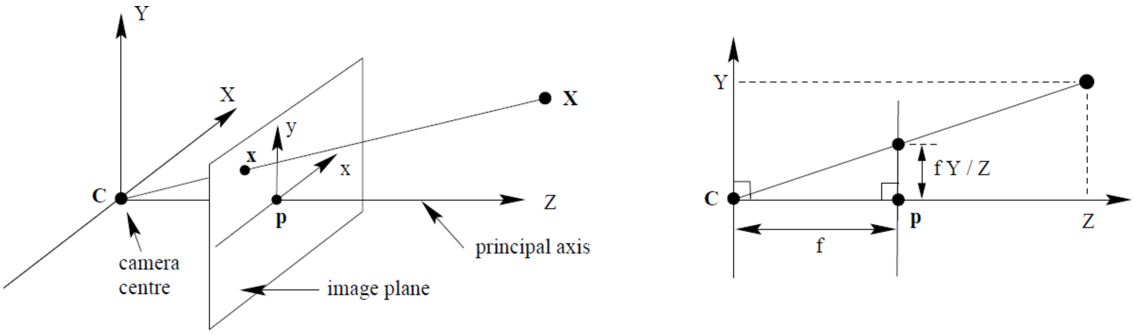

Figure 2.1.: Pinhole Camera Model pixel projection. [14]

The resulting set of 3D points is called a Point Cloud. [15] In addition, the Point Cloud is organized, meaning that the adjacent points are spatially placed next to each other. Also, the memory layout corresponds to the same ordering, which speeds up the steps of many algorithms like searching for nearest neighbors.

2.1.4. RGB-D State of the art

One of the best algorithms regarding agent localization was proposed by Zhang [1] et al. Their technique is based on scanlines. The high-level overview of the algorithm can be seen in the Figure 2.2.

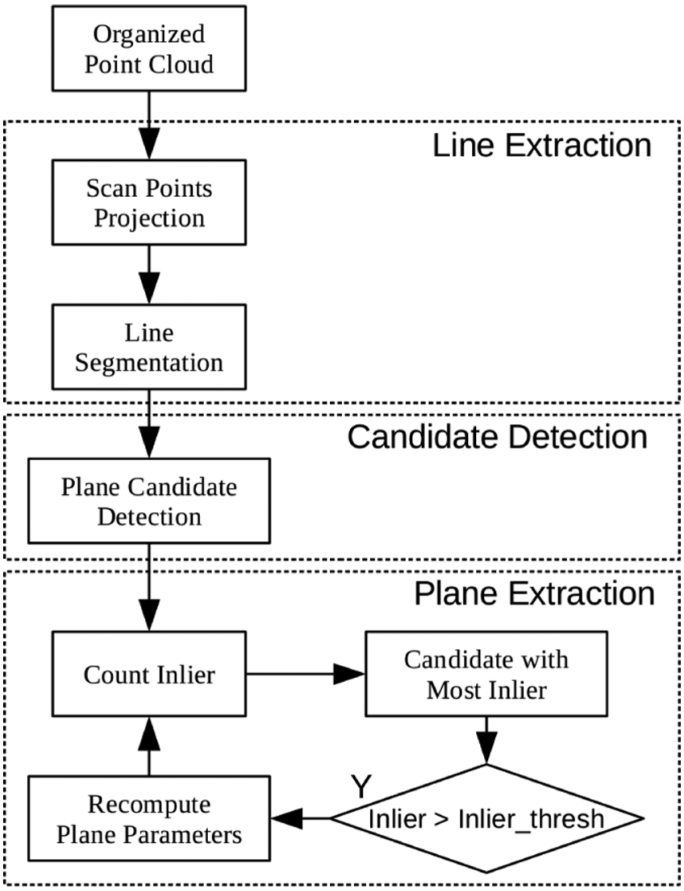

Figure 2.2.: Flowchart of the plane segmentation approach. [1]

There are three main stages of the algorithm:

1. **Line Extraction**

   The method, instead of looking for line segments on all possible scanlines, selects a few scanlines periodically, e.g., every thirty rows. It is worth mentioning that it is possible to select either rows or columns, and we can adjust the interval size to achieve a compromise between accuracy and performance.

   - Scan Points Projection

     Each selected line is treated as a 2D laser scan for the purpose of segmentation. Then, for each pixel on these lines, the projection is calculated according to the equation 2.2 from the RGB-D To Point Cloud Mapping section. 2.1.3

- Line Segmentation

  To extract line segments, the following algorithm is used:

  Algorithm 1: Line Regression [1]

  ```
  Initialize sliding window size $N_f$.
  Fit a line to every $N_f$ consecutive points.
  Compute a line fidelity array. Each element of the array contains the sum of Mahalanobis distances between every three adjacent windows.
  Construct line segments by scanning the fidelity array for consecutive elements having values less than a threshold.
  Merge overlapped line segments and recompute line parameters for each segment.
  ```

  This algorithm was initially proposed in a study by Arras and Siegwart. [16] It yielded good results without much computational complexity.

2. **Plane Candidate Detection**

   For each line segment, a plane candidate is calculated in the following way:

   - Selection of points for local normals

     The more points are selected, the more accurate the result, but with the cost of computational complexity. The middle point for each line segment is a good compromise, which is confirmed empirically in the article.

   - Calculation of local normals

     For each selected point, the smoothing region is chosen as a square with a fixed size (for example 20 pixels). Then, for each region, the principal component analysis (PCA [17]) algorithm is executed, which gives us the local normal.

   - Selection of plane candidates

     Lastly, the valid points regarding PCA are chosen as an inlier (part of the plane) from the smoothing region. Then, the local curvature is calculated, and the inliers with the normals with curvature less than a maximum threshold are selected as plane candidates.

   Example plane candidates result can be seen in the Figure 2.3

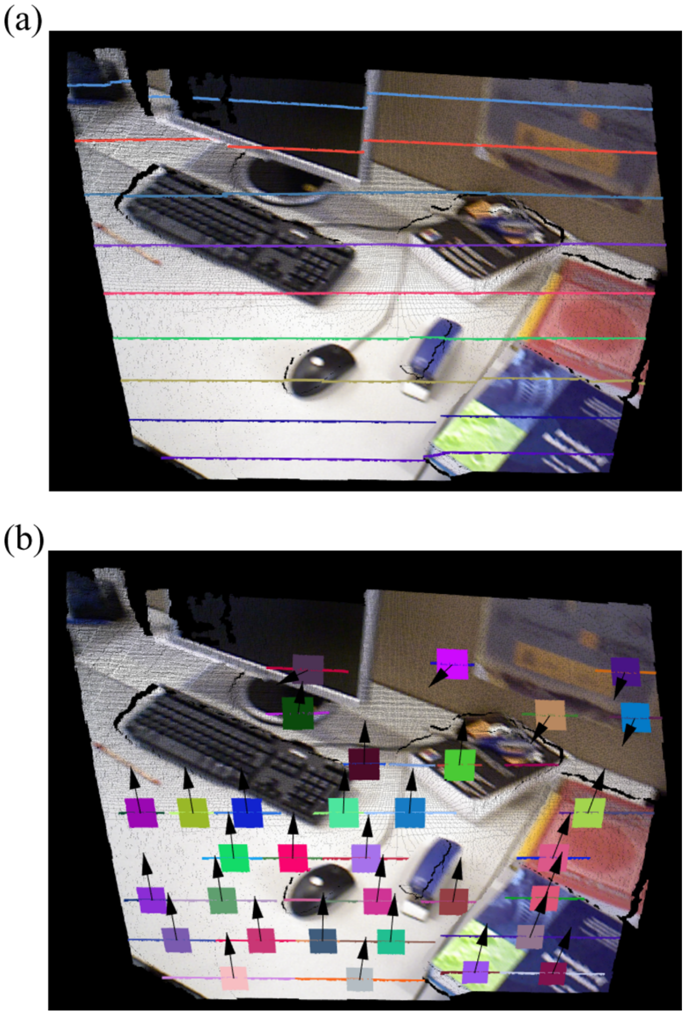

Figure 2.3.: Examle showing plane candidates calculation. [1] (a) Colored lines are scanlines (b) Each plane candidate is represented by part of the line segment (colored line), normal smoothing region (colored square) and local normal emerging from the middle point (black arrow)

3. **Plane Extraction**

For plane extraction, the recursive algorithm is used, which is presented in the Algorithm 2 It takes every plane candidate and discards each with its inlier N_s smaller than the minimal threshold $N_{sthresh}$. Then, it calculates the quality of the inlier by counting how many neighbors within range d_n are closer to the plane than a distance threshold d_τ. The candidate is also invalidated if the number of valid points is smaller than the minimal threshold $N_{\xi_{thresh}}$.

Next, the biggest approved inlier $N_{\xi_{max}}$ is considered to be part of the plane segment from the final result. Ultimately, principal component analysis [17] is calculated again for the selected candidate $N_{\xi_{max}}$, and the actual inlier of the chosen plane segment is deducted directly from the point cloud. After adding the plane segment to the result set, the plane candidate $N_{\xi_{max}}$ is removed from the possibilities for the next iterations. This sequence of actions is repeated recursively until there are no valid plane candidates.

Algorithm 2: Recursive plane extraction [1]

```
Input: plane candidates $\hat{\xi}_i, i = 1, 2, \cdots, n$.
Output: plane segments $\xi_j, j = 1, 2, \cdots, m$.
while number of valid plane candidates $\neq 0$ do
    initialize maximum inlier number $N_{\hat{\xi}_{max}} \leftarrow 0$, corresponding plane
      parameters $\hat{\xi}_{max} \leftarrow invalid$.
    for $i = 1$ to $n$ do
        if $\hat{\xi}_i$ is invalid then
            count number of valid points $N_s$ in normal smoothing region
            if $N_s > N_{sthresh}$ then
                count number of valid points $N_{\hat{\xi}_i}$ by point-plane distance
                  threshold $d_\tau$ and neighbor distance threshold $d_n$
                if $N_{\hat{\xi}_i} > N_{\hat{\xi}_{max}}$ and $N_{\hat{\xi}_i} > N_{\hat{\xi}_{thresh}}$ then
                    $N_{\hat{\xi}_{max}} = N_{\hat{\xi}_i}$
                end
            else
                $\hat{\xi}_i \leftarrow invalid$
            end
        end
    end
    if $N_{\xi_{max}} > N_{\xi_{thresh}}$ then
        recompute plane parameter $\hat{\xi}_{max}$ using all the inlier
        save result plane $\xi_j \leftarrow \hat{\xi}_{max}$
        delete inlier in point cloud data
    else
        break
    end
end
```

2.2. RGB Approach

With the recent boom in artificial intelligence, especially in neural networks, the scientific community has realized, that it is possible to get rid of all depth scanners and extract information about plane segments from simple RGB images, similar to our brain. The first successful attempt using a neural network to detect plane segments from a plain RGB was PlaneNet brought by Liu et al. [18] It took advantage of the recent advancement in neural network technology - Dilated Residual Network (DRM) [19], which was published just a year prior. This success drew much attention to the topic, and since then, there have been multiple enhancements and progressions. Based on the PlaneNet, SlamCraft was introduced by Rambach et al. [20] - a SLAM framework, which proved, that monocular SLAM systems are feasible with adequate neural network untilization. Yu et al. [21] tackled the problem from a different perspective. They have proposed a two-stage algorithm using an encoder-decoder approach and achieved promising results. A big step forward was the introduction of PlaneRCNN by Liu et al. [22] They used mask R-CNN [23] - a recent innovation in a neural network to computer vision. The state-of-the-art advancements are brought by Yaxu Xie et al. in the form of PlaneSegNet [24]. This framework achieves great results and was proved SLAM-worthy by Shy et al. in the form of Structure PLP-SLAM [25], which uses PlaneSegNet for plane segmentation.

2.2.1. Begin: Ask if needed neural network theory

All of the methods addressed in Section 2.2 use some variation of neural networks. Why is that the case? Well, many years ago, Kurt Hornik, Maxwell Stinchcombe, and Halbert White proved mathematically that multilayered feedforward neural networks are universal approximators. [26] So why haven't they been widely spread and entangled in multiple tasks? Training a neural network is highly resource-intensive. To train a neural network, huge amounts of data and computation are required, and thus, the hardware capabilities prevented the scientific community from satisfactory results. With the era of GPUs, the situation has changed and we can see the presence of neural networks almost everywhere. In the section 2.2.2 the basic principles of neural networks are presented.

2.2.2. Neural Networks

2.2.3. End: Ask if needed neural network theory

2.2.4. RGB state-of-the-art - PlaneSegNet

The current state-of-the-art RGB plane segmentation is represented by PlaneSegNet proposed by Yaxu Xie et al. [24] The architecture is based on the YOLACT++ instance segmentation framework published by Bolya et al. [27] with added optimizations in some of the modules. The general overview of the framework can be seen in Figure 2.4. The architecture is quite complex and consists of:

- Encoder

  Produces a group of multi-scale feature maps P_3, P_4, P_5, P_6, P_7.

  - Backbone Network It is a deep residual network based on the ResNet by He et al. [28]
  - Feature Pyramid Feature Pyramid Network structure based on the work of Lin et al. [29]

- Residual Feature Augmentation

  Apart from the standard connection, layer C_5 of the backbone is connected to the prediction layer P_5 through the Residual Feature Augmentation brought by Guo et al. [30] It improves the context of the spatial placement of the features.

- Prediction Head For each layer of the Feature Pyramid Network, it produces c class confidences, four bounding box regressors, and k mask coefficients.

- Fast Feature NMS

  The most significant innovation brought by PlaneSegNet. It aims to enhance overlapping instances detection reliability. It is discussed more in detail in the Subsection 2.2.4.

- ProtoNet

  It is a convolutional neural network. It takes the feature maps from the P_3 layer and predicts k channels for prototype masks.

- Mask Assembly

It assembles prototype masks based on the following linear combination equation:

$$M = \sigma(P\mathbf{C}_{\mathbf{T}}) \tag{2.3}$$

where:

- σ is a non-linear sigmoid function
- P is a tensor of dimensions $h \times w \times k$ containing prototype masks
- $\mathbf{C}$ is a matrix of dimensions $n \times k$ containing mask coefficients for the output of Fast Feature NMS 2.2.4

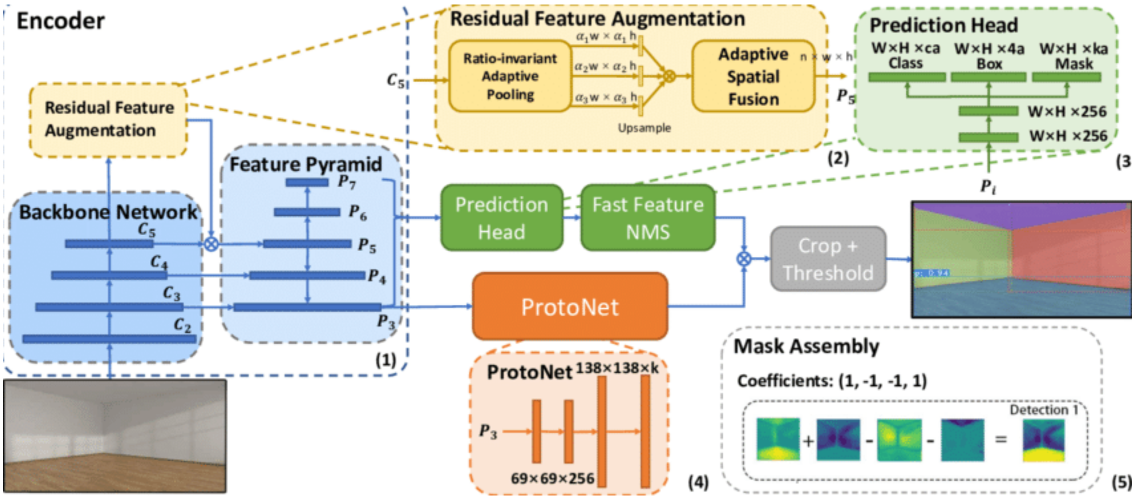

Figure 2.4.: PlaneSegNet Architecture. (1) Encoder (2) Residual Feature Augumentation (3) Prediction Head (4) ProtoNet (5) Mask Assembly [24]

The loss function

It takes into consideration each of the mask, confidence, and bounding box (localization) losses. It is represented with the following equation:

$$\mathscr{L}(x, C, B, M) = \frac{1}{N}(\mathscr{L}_{conf}(x, C) + \alpha\mathscr{L}_{loc}(x, B, B_{gt}) + \beta\mathscr{L}_{mask}(x, M, M_{gt})) \tag{2.4}$$

where:

- C is a confidence representation
- B is a bounding box (localization) representation
- M is a mask representation
- N is the number of positive matchings
- $\mathscr{L}_{conf}$ and $\mathscr{L}_{loc}$ are loss function taken from the single-shot detector presented by Liu et al. [31]
- $\mathscr{L}_{mask}$ is the pixel-wise binary cross entropy between the ground truth and predicted masks.

Fast Feature Non-maximum Suppression

It was observed, that strongly overlapping bounding boxes often don't belong to the same object. Because of that, usage of the classic NMS method brings unnecessary calculation overhead. To resolve this issue, a new method is introduced, Fast Feature Non-maximum Suppression. The technique is based on the research of Salcheider and Niels, who came up with Feature Non-maximum Suppression. [32]

This method is presented in the form of Algorithm 3 The N_1 and N_2 are thresholds for the common area of overlapping bounding boxes (IoU). If IoU is smaller than N_1, the boxes are considered to be from different objects. However, if IoU is bigger than N_2, the boxes are considered to be from the same object. Finally, if IoU is between N_1 and N_2, the similarity S_i of the objects is calculated, and if it is below the given likeness threshold T, the boxes are considered to be from different objects. It is worth noting that this approach adds minimal overhead compared to the Fast NMS.

Algorithm 3: Fast Feature NMS [24]

```
Input: $P \leftarrow Sort(Proposals)$ with Scores, $D \leftarrow \emptyset$
$X^{triu} \leftarrow GetPairwiseIoU(P)$
$K \leftarrow max(X^{triu})$ column-wise
if $K_i \leqslant N_1$ then
    $PUSH(p_i, D)$
else
    if $K_i \leqslant N_2$ then
        $C^{triu} \leftarrow GetCosineSim(p, D)$
        $S \leftarrow max(C^{triu})$ column-wise
        if $S_i \leqslant T$ then
            $PUSH(p_i, D)$
        end
    end
end
return $D$
```

3. Problem Description

3.1. Plane Segmentation Real-life Usability

Many technologies claim that their results are unprecedented, with almost endless application possibilities. Indeed, this is often true, but some algorithms or frameworks are tested on specific datasets. This is especially the case for neural-network-based solutions, where the test and train data are extracted from the same dataset. Therefore, when one tries to apply such technology practically and subjects it to completely different data (from the test and train data), it sometimes turns out that the results are not as good as claimed.

I have selected two state-of-the-art methods that use RGB-only data:

1. planercnn [22]
2. PlaneRecNet [33]

Both of these technologies, besides performing plane segmentation, provide depth estimation. This feature can give insight into the particular algorithm's 'understanding' of an image.

4. Implementation

4.1. Working environment

This section describes the working environment from both, software and hardware perspectives.

4.1.1. Technology stack

The following technology stack was used:

- **Ubuntu** - Linux operating system widely used in the computer vision community. Multiple versions were used: `16.04`, `20.04`, `22.04`, `24.04`. [34]

- **Conda** - Package manager for isolated environments for each of the used projects. [35]

- **Python** - Main programming language for writing code and conducting experiments. Multiple versions were used: `3.7.16`, `3.9.10`, `3.9.18`. [36]

- **CUDA** - A development environment for GPU-accelerated applications. Multiple versions were used: `11.6.1`, `11.7.1` [37]

- **PyTorch** - A machine learning framework with CUDA integration. Multiple versions were used: `0.4.1`, `1.10.1`, `1.13.1` [38]

4.1.2. Workstation parameters

The experiments were prepared and conducted on a personal computer with the following specifications:

- CPU: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz [39]

- GPU: NVIDIA GeForce RTX 4060 [40]

- Hard Drive: Samsung SSD 990 PRO 2TB [41]

4.2. Preparing test dataset

To evaluate the algorithms' real-life usefulness, I prepared a small test dataset of photos. The dataset contains a variety of indoor scenes with plenty of tricky floor layouts, including glass walls, windows, and mirrors. Additionally, there are several photos of the outskirts of the building to evaluate the algorithms' coping with the outdoor environment within the building's proximity. Each scene was shot in two different lighting conditions - day and night. Thus, the dataset is capable of testing the effect of illumination on segmentation quality.

The photos were taken with the rear camera of the `Samsung Galaxy A54 5G` phone with the model name `SM-A546B/DS`. The building that served as a source of the indoor and outdoor scenes is the main building of the AGH faculty of Computer Science - D17.

4.2.1. Calculating Camera Matrix

Many plane segmentation algorithms require camera intrinsic parameters to correctly perform plane segmentation and depth prediction. In this thesis, PlaneRCNN required the following camera matrix:

$$C = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

where:

- f_x is the focal length in pixels on x-axis
- f_y is the focal length in pixels on y-axis
- c_x is the principal point x-axis coordinate in pixels
- c_y is the principal point y-axis coordinate in pixels

Based on the pinhole camera model from Figure 2.1 from Section 2.1.3, these are the mathematical recipes for all the necessary parameters:

$$\begin{cases} f_x = \frac{\frac{1}{2}R_h}{\tan(\frac{1}{2}FoV_h)} \\ f_y = \frac{\frac{1}{2}R_v}{\tan(\frac{1}{2}FoV_v)} \\ c_x = \frac{1}{2}R_h \\ c_y = \frac{1}{2}R_v \end{cases} \tag{4.2}$$

where:

- R_h is the camera horizontal resolution in pixels
- R_v is the camera vertical resolution in pixels
- FoV_h is the camera horizontal field of view in degrees
- FoV_v is the camera vertical field of view in degrees

The image resolution is `4080x3060`, so $R_h = 4080$ and $R_v = 3060$. From the publicly available documentation about the rear camera in the phone, I got the horizontal and vertical field of view (FoV). [42] Thus, we get the respective values for $FoV_h = 72.7°$ and $FoV_v = 57.8°$. Applying these values to the Equation 4.2 gives the following result:

$$\begin{cases} f_x = 2772.05 \approx 2772 \\ f_y = 2771.59 \approx 2772 \\ c_x = 2040 \\ c_y = 1530 \end{cases} \tag{4.3}$$

4.3. Setting-up the environment

The first step in using every repository is setting the working environment up. Nowadays, almost every project using some form of artificial intelligence is Python-oriented. This is especially true for algorithms that incorporate some forms of neural networks. This fact is not coincidental - the Python environment is rich in multiple sophisticated libraries. These libraries provide means for many highly complicated tasks, such as matrix operations, which are essential for any neural network manipulations. Additionally, training a neural network is a process, which requires huge amounts of computation. Because of that, graphic cards (Graphic Processing Unit - GPU) are entangled to do most of the demanding calculations. GPUs are normally optimized for graphic computations, o there are separate dedicated drivers that transform a classic GPU into a General Purpose Graphic Processing Unit (GPGPU), which is tuned for mathematical calculations.

Ultimately, we end up with a tremendously complicated environment with dedicated graphic drivers, python libraries, system libraries, etc. Each of these components adds an additional layer of complexity. Taking into consideration the environment specification, getting a state-of-the-art neural network project going may be a highly challenging and time-consuming task.

4.3.1. PlaneRCNN

I have started my work with an attempt to set up the PlaneRCNN repository. [43] At first glance, everything looks good - there is an installation instruction together with a ‘requirements.txt‘ file. All the installation went smoothly, but when I tried to run an example evaluation:

```
$ python evaluate.py --methods=f --suffix=warping_refine --
    dataset=inference --customDataFolder=example_images
```

The following exception occurred:

```
    Traceback (most recent call last):
    File "evaluate.py", line 20, in <module>
      from models.model import *
    File "/home/agent/Masters/Repos/planercnn/models/model.py", line 23, in <module>
      from nms.nms_wrapper import nms
    File "/home/agent/Masters/Repos/planercnn/nms/nms_wrapper.py", line 11, in <module>
      from nms.pth_nms import pth_nms
    File "/home/agent/Masters/Repos/planercnn/nms/pth_nms.py", line 2, in <module>
      from ._ext import nms
    File "/home/agent/Masters/Repos/planercnn/nms/_ext/nms/__init__.py", line 3, in <module>
      from ._nms import lib as _lib, ffi as _ffi
  ImportError: /home/agent/Masters/Repos/planercnn/nms/_ext/nms/_nms.so: undefined symbol:
      __cudaRegisterFatBinaryEnd
```

After some thorough investigation, I discovered that the PyTorch [38] in version `0.4.1` supports only old CUDA (Compute Unified Device Architecture [37]) versions, which are incompatible with modern GPUs.

Based on this knowledge, I tried to bump the PyTorch version to at least `1.10`, which offers support with some of the latest CUDA releases. Then I encountered another obstacle - there are two custom CUDA modules written in C [44]:

- `nms` - implementation of Non-Maximum Suppression algorithm
- `roi_align` - implementation of RoIAlign algorithm proposed by He et al. [23]

Unfortunately, PyTorch from version `1.0.0` removed support for `C` modules and left only the `C++` [45] option. Since these custom modules needed to be rewritten anyway, after some additional research, I was able to get rid of them completely, thanks to the publicly available replacement by WeihongPan [46]. Thanks to this, the process of setting up the PlaneRCNN environment was simplified.

We can get rid of both custom modules:

- `nms` PyTorch library, in higher versions, provides the Non-Maximum Suppression algorithm, so a version update is enough in this case.
- `roi_align` Implementation of the RoIAlign algorithm is now available as a separate package [47], so it is installed instead of using the `roi_align` custom module.

WeihongPan also published his own method of setting up the PlaneRCNN environment, but it also failed on the package incompatibilities.

Package requirements installation:

```
$ pip install -r requirements.txt
```

Failed with the following error:

```
Collecting cffi==1.15.0 (from -r requirements.txt (line 10))
  Using cached cffi-1.15.0-cp39-cp39-manylinux_2_12_x86_64.manylinux2010_x86_64.whl.metadata (1.2 kB)
Collecting numpy==1.19.2 (from -r requirements.txt (line 11))
  Using cached numpy-1.19.2.zip (7.3 MB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... error
  error: subprocess-exited-with-error
  RuntimeError: Running cythonize failed!
```

which suggests package incompatibilities.

Based on this knowledge, I decided to select my own set of compatible libraries and ship them as a Conda [35] environment definition. As a result of this operation, the `environment.yml` file was created:

```
name: planercnn_pytorch_1_13_cuda_11_7
channels:
  - pytorch
  - conda-forge
  - nvidia
  - defaults
dependencies:
  - cffi=1.15.1
  - matplotlib=3.8.4               # required by RoIAlign test.sh https://github.com/longcw/RoIAlign.pytorch
  - mkl=2024.0                     # due to a pytorch bug https://github.com/pytorch/pytorch/issues/123097
  - numpy=1.21.2
  - opencv=4.5.5
  - pytorch::pytorch=1.13.1
  - scikit-image=0.19.3
  - pytorch::torchvision=0.14.1    # required by RoIAlign test.sh https://github.com/longcw/RoIAlign.pytorch
  - tqdm=4.66.5
prefix: $HOME/.conda/envs/planercnn_pytorch_1_13_cuda_11_7
```

Additionally, I have prepared a convenience script for RoIAlign installation:

```
#!/bin/bash

WORKDIR="roi_align_installation"
GITHUB_URL="https://github.com/longcw/RoIAlign.pytorch"

mkdir $WORKDIR
git clone $GITHUB_URL $WORKDIR
cd $WORKDIR

python setup.py install
./test.sh

cd ..
rm -rf $WORKDIR
```

Due to this, I was able to add an instruction, which sets up the PlaneRCNN with just a few simple steps:

```
$ conda env create -f environment.yml
$ conda activate planercnn_pytorch_1_13_cuda_11_7
$ conda install h5py=3.7.0
$ ./install_roi_align.sh
```

4.3.2. PlaneRecNet

With PlaneRecNet [33], there was a similar situation as with the PlaneRCNN from Section 4.3.1. There is the whole conda environment exported to the file `environment.yml`. So, in theory, anyone should be able to get the environment running with a simple command:

```
$ conda env create -f environment.yml
```

Unfortunately, I encountered the following error:

```
Solving environment: failed

LibMambaUnsatisfiableError: Encountered problems while solving:
  - package libopencv-4.5.3-py39h70bf20d_1 requires ffmpeg >=4.3.2,<4.4.0a0, but none of the providers can be installed
```

```

Could not solve for environment specs
The following packages are incompatible
  - ffmpeg ==4.4.0 h6987444_5 is requested and can be installed;
  - libopencv ==4.5.3 py39h70bf20d_1 is not installable because it requires
    - ffmpeg >=4.3.2,<4.4.0a0 , which conflicts with any installable versions previously reported.
```

In other words, some of the libraries are incompatible. To resolve the problem, I tried manipulating versions of the conflicting libraries. But as a result, I always received even more errors.

I decided to build a compatible set of libraries from scratch as a conda environment definition. I selected the same PyTorch and CUDA versions as in Section 4.3.1, and then, fit the rest of the dependencies. As a result, the following `environment.yml` file was created:

```yaml
name: PlaneRecNet
channels:
  - pytorch
  - conda-forge
  - nvidia
  - defaults
dependencies:
  - cffi=1.15.1
  - cuda-toolkit=11.7
  - numpy=1.21.2
  - opencv=4.5.3
  - pytorch::pytorch=1.13.1
  - scipy=1.7.1
  - pytorch::torchaudio=0.13.1
  - pytorch::torchvision=0.14.1
prefix: $HOME/.conda/envs/PlaneRecNet
```

Finally, anyone can set up their working environment for PlaneRecNet with just one simple command:

```
$ conda env create -f environment.yml
```

5. Results Evaluation

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.1. Running state-of-the-art projects

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

5.2. Model usefulness evaluation

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

6. Conclusions and future works

In the past several years, there have been many possible uses for RGB-D-based technologies. However, most of them focus on detailed scanning like Honti et al. [48] with little use for simultaneous localization and mapping (SLAM). Resuming, the more progress in neural networks, the more the scientific community moves away from RGB-D in favor of plain RGB.

6.1. Achieved goals and observations

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

6.2. Areas for development

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Bibliography

[1] Lizhi Zhang, Diansheng Chen, and Weihui Liu. "Fast plane segmentation with line primitives for RGB-D sensor". In: *International Journal of Advanced Robotic Systems* 13 (Dec. 2016), p. 172988141666584. DOI: 10.1177/1729881416665846.

[2] Tehreem Rabbani, F.A. Heuvel, and George Vosselman. "Segmentation of point clouds using smoothness constraint". In: *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 36 (Jan. 2006).

[3] Bir Bhanu et al. "RANGE DATA PROCESSING: REPRESENTATION OF SURFACES BY EDGES." In: 1986. eprint: https://users.cs.utah.edu/~tch/publications/pub70.pdf. URL: https://api.semanticscholar.org/CorpusID:268107315.

[4] Angel Sappa and Michel Devy. "Fast range image segmentation by an edge detection strategy". In: vol. 0. Feb. 2001, pp. 292–299. ISBN: 0-7695-0984-3. DOI: 10.1109/IM.2001.924460.

[5] Rihua Xiang and Runsheng Wang. "Range image segmentation based on split-merge clustering". In: vol. 3. Sept. 2004, 614–617 Vol.3. DOI: 10.1109/ICPR.2004.1334604.

[6] X.Y. Jiang, U. Meier, and H. Bunke. "Fast range image segmentation using high-level segmentation primitives". In: *Proceedings Third IEEE Workshop on Applications of Computer Vision. WACV'96.* 1996, pp. 83–88. DOI: 10.1109/ACV.1996.572006.

[7] E. Natonek. "Fast range image segmentation for servicing robots". In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146).* Vol. 1. 1998, 406–411 vol.1. DOI: 10.1109/ROBOT.1998.676445.

[8] Yuichi Taguchi et al. "Point-plane SLAM for hand-held 3D sensors". In: May 2013, pp. 5182–5189. ISBN: 978-1-4673-5641-1. DOI: 10.1109/ICRA.2013.6631318.

[9] Martin A. Fischler and Robert C. Bolles. “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[10] Radu Rusu and Steve Cousins. “3D is here: Point cloud library (PCL)”. In: May 2011. DOI: 10.1109/ICRA.2011.5980567.

[11] Renato Salas-Moreno et al. “Dense planar SLAM”. In: Sept. 2014, pp. 157–164. DOI: 10.1109/ISMAR.2014.6948422.

[12] Michael Kaess. “Simultaneous localization and mapping with infinite planes”. In: *Proceedings - IEEE International Conference on Robotics and Automation* 2015 (June 2015), pp. 4605–4611. DOI: 10.1109/ICRA.2015.7139837.

[13] Ming Hsiao et al. “Keyframe-based dense planar SLAM”. In: May 2017, pp. 5110–5117. DOI: 10.1109/ICRA.2017.7989597.

[14] *Pinhole Camera Model.* https://hedivision.github.io/Pinhole.html. Accessed: 2024-09-21.

[15] *Point cloud.* https://en.wikipedia.org/wiki/Point_cloud. Accessed: 2024-09-21.

[16] Kai Arras and Roland Siegwart. “Feature Extraction and Scene Interpretation for Map-Based Navigation and Map Building”. In: *Proceedings of SPIE, Mobile Robotics XII* 3210 (Aug. 1999). DOI: 10.1117/12.299565.

[17] Karl Pearson. “LIII. On lines and planes of closest fit to systems of points in space”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572. DOI: 10.1080/14786440109462720. eprint: https://doi.org/10.1080/14786440109462720. URL: https://doi.org/10.1080/14786440109462720.

[18] Chen Liu et al. “PlaneNet: Piece-wise Planar Reconstruction from a Single RGB Image”. In: (Apr. 2018). DOI: 10.48550/arXiv.1804.06278.

[19] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. “Dilated Residual Networks”. In: July 2017, pp. 636–644. DOI: 10.1109/CVPR.2017.75.

[20] Jason Rambach et al. “SlamCraft: Dense Planar RGB Monocular SLAM”. In: Mar. 2019. DOI: 10.23919/MVA.2019.8757982.

[21] Zehao Yu et al. “Single-Image Piece-Wise Planar 3D Reconstruction via Associative Embedding”. In: June 2019, pp. 1029–1037. DOI: 10.1109/CVPR.2019.00112.

[22] Chen Liu et al. “PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image”. In: (Dec. 2018). DOI: 10.48550/arXiv.1812.04072.

[23] Kaiming He et al. “Mask R-CNN”. In: Oct. 2017, pp. 2980–2988. DOI: 10.1109/ICCV.2017.322.

[24] Yaxu Xie et al. “PlaneSegNet: Fast and Robust Plane Estimation Using a Single-stage Instance Segmentation CNN”. In: Mar. 2021. DOI: 10.1109/ICRA48506.2021.9561693.

[25] Fangwen Shu et al. “Structure PLP-SLAM: Efficient Sparse Mapping and Localization using Point, Line and Plane for Monocular, RGB-D and Stereo Cameras”. In: May 2023, pp. 2105–2112. DOI: 10.1109/ICRA48891.2023.10160452.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[27] Daniel Bolya et al. “YOLACT++: Better Real-time Instance Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PP (Aug. 2020), pp. 1–1. DOI: 10.1109/TPAMI.2020.3014297.

[28] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: June 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[29] Tsung-Yi Lin et al. “Feature Pyramid Networks for Object Detection”. In: July 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[30] Chaoxu Guo et al. “AugFPN: Improving Multi-Scale Feature Learning for Object Detection”. In: June 2020, pp. 12592–12601. DOI: 10.1109/CVPR42600.2020.01261.

[31] Wei Liu et al. “SSD: Single Shot MultiBox Detector”. In: vol. 9905. Oct. 2016, pp. 21–37. ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2.

[32] Niels Salscheider. “FeatureNMS: Non-Maximum Suppression by Learning Feature Embeddings”. In: Jan. 2021, pp. 7848–7854. DOI: 10.1109/ICPR48806.2021.9412930.

[33] Yaxu Xie et al. *PlaneRecNet: Multi-Task Learning with Cross-Task Consistency for Piece-Wise Plane Detection and Reconstruction from a Single RGB Image*. 2021. arXiv: 2110.11219 [cs.CV].

[34] *Ubuntu*. https://ubuntu.com/. Accessed: 2024-09-25.

[35] *Conda Documentation*. https://docs.conda.io/en/latest/. Accessed: 2024-09-24.

[36] *Python*. https://www.python.org/. Accessed: 2024-09-25.

[37] *CUDA Toolkit*. https://developer.nvidia.com/cuda-toolkit. Accessed: 2024-09-24.

[38] *PyTorch*. https://pytorch.org/. Accessed: 2024-09-24.

[39] *Intel Core i5-10400F Processor*. https://www.intel.com/content/www/us/en/products/sku/199278/intel-core-i510400f-processor-12m-cache-up-to-4-30-ghz/specifications.html. Accessed: 2024-09-25.

[40] *GeForce RTX 4060 Family*. https://www.nvidia.com/en-us/geforce/graphics-cards/40-series/rtx-4060-4060ti/. Accessed: 2024-09-25.

[41] *Internal SSD 990 PRO*. https://semiconductor.samsung.com/consumer-storage/internal-ssd/990-pro/. Accessed: 2024-09-25.

[42] *CCamera FV-5 Samsung Galaxy A54 5G*. https://www.camerafv5.com/devices/manufacturers/samsung/scg21_scg21_0/. Accessed: 2024-09-24.

[43] *PlaneRCNN: 3D Plane Detection and Reconstruction from a Single Image*. https://github.com/NVlabs/planercnn. Accessed: 2024-09-24.

[44] *C language*. https://en.cppreference.com/w/c/language. Accessed: 2024-09-24.

[45] *C++ language*. https://en.cppreference.com/w/cpp/language. Accessed: 2024-09-24.

[46] *WeihongPan for of planercnn*. https://github.com/WeihongPan/planercnn-pytorch1.10.1_cuda11.3/tree/cuda11.3_pytorch1.10. Accessed: 2024-09-24.

[47] *RoIAlign for PyTorch*. https://github.com/longcw/RoIAlign.pytorch. Accessed: 2024-09-24.

[48] Richard Honti, Ján Erdélyi, and Alojz Kopacik. "Semi-Automated Segmentation of Geometric Shapes from Point Clouds". In: *Remote Sensing* 14 (18) (Sept. 2022). DOI: 10.3390/rs14184591.