Application of Wireframe Detection Technology in Construction Monitoring

Ding Zhou[1, 2a*], Guohua Wei[1b], Xiaojun Yuan[3c]

[1]Southern University of Science and Technology, Shenzhen, 518055 China
[2]Griffith University, Brisbane, 4104 Australia
[3]University of Electronic Science and Technology of China, Chengdu, 610054 China

[a]zhoud3@sustech.edu.cn, [b]weigh@mail.sustech.edu.cn,
[c]xjyuan@uestc.edu.cn
*Corresponding author

Abstract. The increasing demand for automation technology in the construction industry has driven the research on computer vision applications for construction monitoring. Spurred by current trends, this paper introduces an automated comparison system for construction monitoring based on computer vision, focusing on integrating and applying wireframe detection. Specifically, by utilizing the F-Clip++ algorithm, the proposed system automatically extracts line segment features from architectural images, thereby significantly reducing human intervention and enhancing the real-time nature and efficiency of the monitoring process. Traditional construction monitoring methods rely on manual inspection and experiential judgment, which are time-consuming, labor-intensive, and subjective. Nevertheless, F-Clip++ affords efficient line segment detection and rapidly and accurately identifies the boundaries of architectural elements in complex construction environments. This technological advancement provides precise data support for subsequent comparative analysis, allowing the rapid identification and analysis of discrepancies between building information models (BIM) and actual installations. In practical applications, the system first preprocesses architectural images to enhance the accuracy of line segment extraction. Then, F-Clip++ effectively extracts complex lines and edges in the images extracted, forming a clear wireframe structure. Experiments demonstrate that F-Clip++ improves the real-time nature and efficiency of monitoring and lays the foundation for the intelligent development of the construction industry.

Keywords: Wireframe Detection, Construction Monitoring, Computer Vision, Deep Learning, F-Clip++ Algorithm

1 Introduction

In accelerating the intelligent transformation of traditional industries, advanced technologies are showing broad application prospects in Mechanical, Electrical, and Plumbing (MEP) engineering. The maturity of computer vision and deep learning technologies has enabled the intelligent processing of engineering data, especially onsite images

W. Nie et al. (eds.), *Proceedings of the 2024 6th International Conference on Hydraulic, Civil and Construction Engineering (HCCE 2024)*, Atlantis Highlights in Engineering 37,
https://doi.org/10.2991/978-94-6463-726-7_35

and videos, playing a crucial role in enhancing efficiency, accuracy, and automation. Wireframe detection, as an effective technical means, provides significant support for monitoring buildings and their components, as it identifies the buildings' geometric structure and layout, offering a deep understanding of architectural design. This understanding aids subsequent structural analysis, improves construction precision, and quickly detects deviations between the construction process and design drawings (such as building information models [BIM]), enhancing construction quality and reducing maintenance costs in the later stages.

Recording issues have become standard practice, with construction sites capturing photos and videos daily, forcing almost every construction worker to be equipped with a camera-enabled smartphone [1]. Effectively managing these media files using computer vision technology to gain deep insights and analysis brings immense value. However, the installation quality of concealed works, such as the layout and installation of critical facilities like pipelines, cables, and ventilation systems, directly affects the safety and functionality of the buildings. The quality of these installations impacts the overall performance of the building and is also a concern for later maintenance and safety.

Traditional methods for monitoring concealed work installation primarily rely on construction drawings and manual inspections. However, these are inefficient and cannot ensure consistency between actual installations and design plans. Furthermore, environmental changes and human errors may lead to deviations of the concealed works from the original BIM model during construction. If these deviations are not promptly detected, they affect the later maintenance and potentially cause safety incidents. Currently, the main challenges faced in monitoring the installation of concealed works include the lack of real-time monitoring methods, low data accuracy, and cumbersome comparative analysis. These issues make quality control of concealed works more challenging, urgently requiring the introduction of advanced monitoring technologies and automation tools to enhance monitoring efficiency and accuracy.

Spurred by this necessity, this paper develops F-Clip++, an efficient and rapid line segment detection fully convolutional model that employs an innovative continuous angle error function, accelerating convergence speed and enhancing detection accuracy. The F-Clip++ model utilizes deep learning technology to accurately extract line segment features of building components from onsite photos, capturing geometric contours and identifying the directions and connection points of building components.

As the foundation of the automated comparison system, wireframe detection technology offers significant advantages, including automated feature extraction, high-precision comparative analysis, and simplified processing procedures. Through F-Clip++, line segment features are automatically extracted, reducing human intervention and enhancing the timeliness and efficiency of monitoring. Accurate line segment detection provides data support for subsequent comparative analysis, allowing for rapid identification of discrepancies between the BIM model and actual installations. In summary, applying the F-Clip++ model in construction monitoring enhances the automation level of monitoring and provides robust technical support for achieving efficient quality control of pipeline installations.

2 Related Work

At CVPR 2018, Wireframe detection was introduced [2], where a wireframe comprises visible line segments and their intersections in an image that reflects the scene's geometric structure. Although this task is similar to general image line detection, it focuses on structural line segments that reflect geometric structures in a scene, such as wall intersections and object outlines, rather than lines produced by changes in object textures or lighting [3].

Deep learning-based wireframe detection can be divided into two-stage and one-stage algorithms. Two-stage algorithms typically generate a candidate set and a verification step, requiring a verification module to filter out false positive candidate line segments to enhance model performance. For instance, Deep wireframe parsing (DWP) [2] utilizes convolutional neural networks to predict wireframe endpoints, line segment branching angles, and online probabilities by proposing candidate line segment sets through endpoint branching and obtaining the final predicted line segments based on the direction of the endpoint branches. Line-convolutional neural network (LCNN) [4] is a typical two-stage method inspired by Faster region-based CNN (FasterRCNN) [5]. Unified line segment detection (ULSD) [6] adopts a similar structure, using predictions of Bezier curves to detect straight-line segments in distorted images. Holistic attraction wireframe parsing version 1 (HAWP v1) [7], HAWP version 2 (HAWP v2) [8], and segmentation refinement wireframe-net (SRW-Net) [9] also implement similar operations, dividing the model into candidate generation and verification phases to achieve wireframe detection. Besides, the point pair graph network (PPGNET) [10] extracts line segment endpoints and assesses the connectivity between these points. Traditional algorithms such as line segment detector (LSD) [11], deep line segment detector (DeepLSD) [12], and LSDNet [13] are considered two-stage methods, first constructing line segment support regions and then estimating line segment parameters and filtering.

One-stage methods directly output line segment parameters without requiring a candidate proposal and verification steps, typically affording faster processing speeds. Our preliminary work, F-Clip [14], introduced a one-stage detection model that directly predicts line segment midpoints, lengths, and angles, balancing performance and detection speed well. Efficient line segment detection (ELSD) [15] also adopts a similar approach for line segment detection. Unlike ELSD, the true position line segment detector (TP-LSD) [16] achieves one-stage wireframe detection by predicting the line segment midpoint and two endpoints. Moreover, the line endpoint transformer (LETR) [17] references the object detection algorithm, i.e., detection transformer (DETR) [18], by using a transformer structure. LETR perceives global structural information about line segments through two phases, from coarse-grain to fine-grain interactions. Additionally, line segment and feature interactions are realized through a cross-attention mechanism, while a self-attention mechanism is employed to achieve information interchange among line segments. Owing to the high computational demand of transformers and the complex structure of LETR's self-attention cross-attention, LETR has slower detection speeds.

Depending on the line segment representation method used in wireframe detection algorithms, they can be categorized into endpoint representation, field representation,

midpoint representation, and query-based representation. Endpoint representation methods such as LCNN, DWP, and PPGNET focus on endpoint detection and extract line segments by assessing the connectivity of the endpoints. The field representation method introduced by attraction field map (AFM) [19] defines a field at a pixel's location as a two-dimensional vector formed by the pixel and its projection point on the nearest line segment. In AFM, the line segments in the image are predicted by interpreting the field using a neural network. HAWP based on AFM introduces a 4D field containing endpoint information. Wireframe detection algorithms based on line segment midpoints emphasize the detection of line segment midpoints, with TP-LSD [16] and midpoint line segment detector (MLSD) [20] using midpoints and endpoint offsets to represent line segments. Unlike these methods, ELSD [15] and our preliminary work, F-Clip, use midpoints, line segment angles, and lengths to represent line segments. Finally, LETR, which is based on query representation, uses the transformer structure to convert latent layer queries into predicted line segment outputs.

3 Methodology

3.1 Line Representation Methods

Two endpoints determine line segment l and can be represented non-directionally. This paper utilizes the CLA (center, length, angle) representation strategy, where the line segment is described by its midpoint, radius (half the size of the segment), and angle. The midpoint is at the middle of the endpoints, and the radius is half the length of the line segment, as presented in Eq. 1 and 2. The endpoint with the smaller x-coordinate is the left endpoint p_l, and the endpoint with the larger x-coordinate is the right endpoint p_r. Suppose the x-coordinates of both endpoints are the same, meaning the line segment is vertical. In that case, the point with the larger y-coordinate is the right endpoint, and the other is the left endpoint to eliminate ambiguity. The upper endpoint p_u of the line segment is the endpoint with the larger y-value, and the lower endpoint p_d is the endpoint with the smaller y-value. If the y-values are the same, i.e., the line segment is horizontal, the endpoint with the larger x-value is the upper endpoint, and the other is the lower endpoint. The angle α of the line segment is the angle between the vector from the lower endpoint to the upper endpoint and the positive direction of the x-axis, ranging within $[0, \pi)$. Using these parameters, the upper and lower endpoints of the line segment can be calculated as follows and illustrated in Fig. 1:

$$p_c = 0.5 \times (p_u + p_d) \tag{1}$$

$$l = 0.5 \times \| p_u - p_d \|_2 \tag{2}$$

$$p_u = p_c + (l\cos\alpha, l\sin\alpha) \tag{3}$$

$$p_d = p_c - (l\cos\alpha, l\sin\alpha) \tag{4}$$

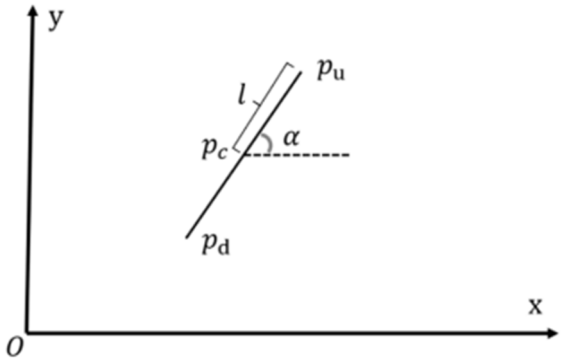

Fig. 1. Definitions of line radius, midpoint, and angle

Line segments can be mathematically represented in various ways. For instance, TP-LSD [16] describes line segments using the midpoint and two endpoints. However, this approach is redundant and cannot ensure that the endpoints are symmetrical about the midpoint. Therefore, this paper uses the line center, radius, and angle to represent line segments, which is a concise and non-redundant strategy. The angle of the line segment is easily inferred from local regions, which facilitates learning by deep learning models and reduces the demand on the receptive field. Furthermore, the statistical properties of angles in man-made environments assist in recognizing line segments.

3.2 Model Structure

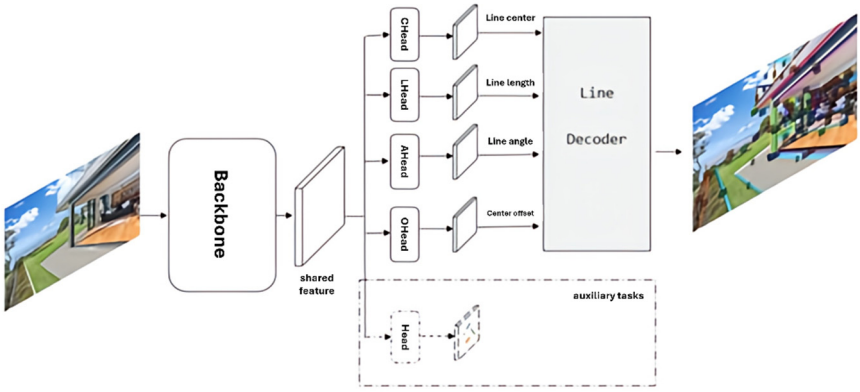

Fig. 2. Model structure

Fig. 2 illustrates the structure of F-Clip++, which primarily comprises a convolutional neural network (CNN) backbone, multiple heads, and a decoder. The backbone is responsible for extracting features from red-green-blue (RGB) images, the heads provide

the prediction results according to task requirements, and the decoder converts the network output into the final format without containing learnable parameters. F-Clip++ is a single-stage model where a fully convolutional network directly outputs results, eliminating complex post-processing steps. Unlike the two-stage method of LCNN [4], F-Clip++ avoids candidate line proposals and complex verification networks, simplifying the model structure and inference process while reducing the aliasing effects in line rasterization, thereby enhancing model performance. The subsequent sections detail the design philosophy of the model.

Backbone. When monitoring architectural components, accurately inferring the angles and midpoints of line segments is crucial for line detection. Although all points on a line segment share the same angle, and thus, it is trivial to derive them from local regions, determining the midpoint and length of a line segment requires a larger receptive field, requiring the observation of the entire segment area.

Therefore, to expand the model's receptive field without sacrificing feature resolution, this paper employs HourglassNet and high-resolution network (HRNet) as the backbone networks, which rely on multi-scale feature fusion instead of special convolutional operations. Specifically, we decompose the wireframe detection task into midpoint detection, angle regression, and length regression tasks. The model predicts whether each pixel is a line segment midpoint and estimates the line segment radius and angle. It should be noted that the designed heads are independent, i.e., CHead predicts midpoints, OHead predicts offsets, AHead predicts angles, and LHead predicts half-lengths. These heads have output dimensions of 2×H×W, 2×H×W, 1×H×W, and 1×H×W, respectively, and the CHead directly outputs midpoint probabilities. Notably, these heads follow the design philosophy of YOLOX, featuring lightweight structures and independence from each other.

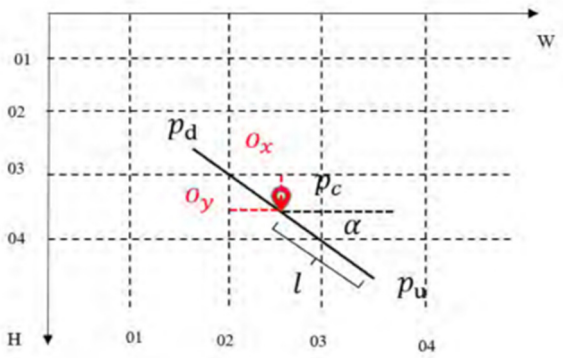

Fig. 3. Model output illustration

Line Decoder. As illustrated in Fig. 3, the Line Decoder receives outputs from CHead, OHead, AHead, and LHead and calculates the endpoints of the line segments using specific formulas. Based on the midpoint probabilities predicted by CHead, the model

selects the top-K line segments with the highest output probabilities, provided that these exceed threshold ct. It is worth noting that to reduce duplicate predictions, this paper employs soft non-maximum suppression (softNMS) to attenuate the probabilities of non-maximal values within a k×k window. Additionally, structured non-maximum suppression (StructNMS) further reduces duplicate line segments by sorting and removing the nearby line segments. Although StructNMS enhances performance, it also impacts inference speed.

Loss. This study decomposes the wireframe detection task into point classification, center offset regression, line segment length, and angle regression, with predictions made using dedicated heads. The center point classification task is trained using the cross-entropy loss presented in Eq. 5, where N represents the total number of cells output by the head, $C_{i,j}$ indicates whether a line segment center falls into the cell at the i-th row, and j-th column (1 if yes, 0 if no), and $C_{i,j}$ is the probability value predicted by the model for containing a line segment center. Given the small number of line segments in images, which leads to an uneven distribution of positive and negative samples, Focal Loss (Eq. 6) is used, where the parameters α and γ adjust the impact of positive and negative samples and easy and hard samples on the loss.

The horizontal and vertical offsets of the line segment midpoint relative to the responsible cell, o_x, and o_y, and the line segment length and angle α, are normalized between 0 and 1 to allow direct prediction using the neural network. The regression issues are measured using the L1 error to assess the difference between predicted and actual values. Moreover, the midpoint offset error L_O, line segment length error L_L, and line angle error L_α are calculated using Eq. 6, 7, and 8, respectively. A detailed discussion and the related solutions addressing the angle discontinuity issue in angle prediction will be presented later. When images do not have line segments leading to an undefined 0/0 form, the value is set to 0. The final model training error L is the weighted sum of these errors, with weight coefficients determined using a parameterized search, as follows:

$$\mathcal{L}_C \doteq -\frac{1}{N}\sum_{i,j}\left(C_{i,j}\log\left(\hat{C}_{i,j}\right)+\left(1-C_{i,j}\right)\log\left(1-\hat{C}_{i,j}\right)\right) \tag{5}$$

$$\mathcal{L}_C \doteq -\frac{1}{N}\sum_{i,j}\left(C_{i,j}\log\left(\hat{C}_{i,j}\right)+\left(1-C_{i,j}\right)\log\left(1-\hat{C}_{i,j}\right)\right) \tag{6}$$

$$\mathcal{L}_o \doteq \frac{1}{\sum C_{i,j}}\sum_{i,j} C_{i,j}\left|\hat{\boldsymbol{O}}_{i,j}-\boldsymbol{O}_{i,j}\right| \tag{7}$$

$$\mathcal{L}_L \doteq \frac{1}{\sum C_{i,j}}\sum_{i,j} C_{i,j}\left|\hat{L}_{i,j}-L_{i,j}\right| \tag{8}$$

$$\mathcal{L}_\alpha \doteq \frac{1}{\sum C_{i,j}}\sum_{i,j} C_{i,j}\left|\hat{A}_{i,j}-A_{i,j}\right| \tag{9}$$

$$\mathcal{L} = \lambda_C \mathcal{L}_C + \lambda_o \mathcal{L}_O + \lambda_l \mathcal{L}_l + \lambda_\alpha \mathcal{L}_\alpha \tag{10}$$

3.3 Reparameterization

To enhance the model's expressiveness without increasing the computational burden during inference, this paper adopts the reparameterization technique proposed by RepVGG [21], which is applied to the primary and bottleneck blocks. During training, RepVGG introduces parallel 1×1 convolutions and a residual structure alongside the 3×3 convolution found in ResNet [22]. This strategy increases the model's parallel branches and aids learning. In the inference phase, RepVGG uses a new 3×3 convolutional kernel to equivalently replace the 3×3 convolutions, 1×1 convolutions, and the residual structures used during training, thereby reducing computational load.

Regarding the method for replacing convolutions, if there are three parallel 3×3 convolutions, their equivalent convolution is represented by a single 3×3 convolution, whose kernel parameters are the sum of the parameters from the original three kernels. The three parallel branches in RepVGG are converted into an equivalent convolutional kernel using the following process. First, the residual branch is transformed into a 1×1 convolution, with all weights set to 1. Then, the equivalent 1×1 convolutional kernel of the residual and the 1×1 convolutions used during training are converted to 3×3 convolutions. The method for converting a 1×1 convolution to a 3×3 convolution involves assigning the weights of the 1×1 convolution to the central position of the 3×3 kernel, setting the other positions to zero, and finally adding the weights of the three 3×3 kernels to obtain the equivalent convolutional kernel weights.

3.4 Continuous Angle Error Function

Section 3.1 defined the line segment angle α as the angle formed between the vector from the lower endpoint to the upper endpoint and the positive x-axis direction within $(0, \pi]$.

Fig. 4 highlights that if l_1 represents the true angle of the line segment and the annotated angle is 0, then for the model to predict line l_3, it needs to rotate the predicted line segment clockwise to 0, which requires a rotation greater than $\pi/2$. However, the model achieves a better prediction result by rotating the line segment counterclockwise using a smaller angle. Similarly, if line segment l_1 is annotated as π, a similar situation occurs for the model's prediction result l_0. The angular error between l_0 and l_1 should be less than that between l_4 and l_1, but the angle error defined by previous models (Eq. 9) does not adequately reflect this. This is because the model does not recognize that an inclination of 0 and π are the same. Angle prediction may experience a significant discontinuity at 0 and π. As demonstrated above, setting all horizontal line angles in the dataset to 0 does not resolve this issue effectively. Suppose the true inclination of the line segment is infinitely close to π but not equal to π. In this case, the predicted line segment l_3 needs a significant counterclockwise rotation, whereas when the true inclination is π, the predicted line segment needs a significant clockwise rotation to 0. Between inclinations approaching π and π, a critical angle must exist that causes the model's

prediction angle to change from counterclockwise to clockwise rotation, resulting in a jump discontinuity in model error [23].

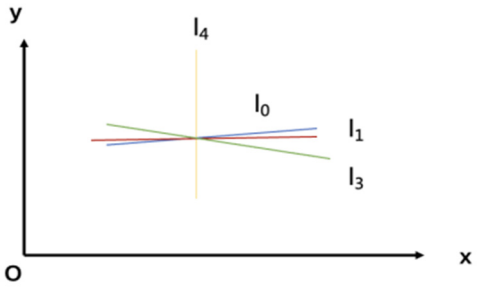

Fig. 4. Discontinuity in angle ambiguity and angle prediction

Spurred by the discontinuity in line segment angle prediction presented above, this paper proposes a new method to address this concern. Specifically, we divide the angle prediction process into two parts: one predicts the angle sign, and the other predicts the angle value. For example, the model predicts the cosine value of angle α, and it lies within (0, π], the range of the cosine value is (-1, 1]. Eq. (9) directly optimizes the neural network using the L1 loss to compare the model's predictions with the true values. The developed model, F-Clip++, divides the angle prediction process into two parts: the angle value prediction and the angle sign prediction. The former part predicts |cosα|, which reveals that at angles 0 and π, |cosα|=1, eliminating any discontinuity. The angle sign prediction part performs a binary classification to predict the sign of cos(α), which is positive if the upper endpoint is to the right of the lower endpoint and negative if it is to the left. Indeed, when the line segment is vertical, cos(α)=0, and its sign is undefined. When the line segment is horizontal, both positive and negative cos(α) values represent the same line segment. Thus, to eliminate this ambiguity, we consider the sign positive or negative when the absolute value of cos(α) is 0 or 1. This is achieved by adding weights to the classification error function (Eq. 11), which are zero at angles 0, π/2, and π, gradually decreasing as the angle approaches these values. This strategy effectively handles the issue of not distinguishing between the positive and negative signs of cos(α) for horizontal and vertical line segments.

$$w(x) = x(1 - x) \tag{11}$$

The L1 loss is commonly used to measure the difference between the predicted and actual values for the newly proposed angle error function. In contrast, this study uses a cross-entropy similar to Eq. 5 for the sign prediction part, combined with the weights proposed in Eq. 11. The final angle error function is presented in Eq. 12. Sign (·) is the sign function, outputting 1 if its argument is non-negative and 0 otherwise, λ_s is the weight for the sign error, λ_v is the weight for the value error, p_s is the probability that the model predicts the sign as positive, and p_v is the model's prediction of the angle value.

A similar idea can be employed for scenarios where the model directly predicts the angle in degrees or radians. For instance, when directly predicting the angle in degrees within a 0-180 degrees range, we add an offset of -90 degrees to transform the range to [-90 degrees, +90 degrees], then divide by 90 to normalize it to [-1, 1], similar to predicting cos(α). The process is reversed to convert the model prediction back to angle values in degrees: multiply by 90 degrees, then add 90 degrees. This method effectively handles the discontinuity in angle prediction errors at 0 and 180 degrees.

$$L_\alpha \doteq \lambda_s w(|\cos\alpha|)\, CE(+\lambda_v |p_v - |\cos\alpha||) \tag{12}$$

4 Experimental Results and Evaluation

4.1 Dataset

Deep learning models require a large amount of training data for learning. Nevertheless, relatively few datasets are available for wireframe detection, with the primary ones being ShanghaiTech [2] and YorkUrban. The former is presented in Table 1 and comprises 5,462 images, including 5,000 in the training set and 462 in the test set. These images encompass man-made indoor and outdoor environments, featuring scenes such as living rooms, bedrooms, and courtyards. Each of the 5,462 images has been manually annotated to highlight structural lines that imply scene structure, disregarding texture and shadow-induced texture lines. The YorkUrban dataset annotates line segments in images that meet the Manhattan criteria. Still, with only 102 images, it is relatively small and often used to test the generalizability of models trained on other datasets.

Table 1. Dataset statistical information

ID	Dataset	Training set	Test set	Total images
0	ShanghaiTech	5000	462	5462
1	YorkUrban	/	102	102

4.2 Evaluation Metrics

The performance of the wireframe detection models is evaluated using two metrics: Average precision on heatmap (AP^H) and the newly proposed structure average precision (sAP). AP^H and sAP are based on the area under the precision-recall (PR) curve, which measures model performance. Precision measures the proportion of correct predictions, while recall measures the proportion of true results that are correctly predicted, as defined respectively in Eq. 13 and 14. TP (true positives) represents the number of correct predictions, FP (false positives) represents the number of incorrect predictions, and FN (false negatives) represents the number of missed detections.

AP^H rasterizes the model's predicted wireframes to form a heatmap P, which is compared with the rasterized true label line segments forming G. If P and G have pixel values of 1 at the same location, it is considered a correct prediction, i.e., TP. If a pixel in P is 1 while the corresponding position in G is 0, it is regarded as an FP. Conversely,

if G is 1 and P is 0, it is considered a FN. The precision (p^H) and recall (r^H) are defined by Eq. 15 and 16. AP^H allows for some error, with precision and recall calculated using bipartite graph matching.

sAP addresses the shortcomings of AP^H in handling duplicate line segments by avoiding using the rasterized results and directly defining the distance between predicted and actual line segments. A predicted line segment, denoted as $\hat{l}$, is described by its endpoints $\hat{p}_1$ and $\hat{p}_2$, and the distance to the annotated line segment l, dist ($\hat{l}$, l), is defined using Eq. 17. Considering two endpoint matching scenarios, the minimum distance is used as the distance between line segments. Under a given threshold θ, the correctness of a prediction is determined based on the distance between the predicted and actual results. Calculating sAP includes ordering the predicted lines using descending probability values and calculating the distance of each predicted line segment to all annotated line segments. If the minimum distance is less than θ and the annotated line segment is not yet matched, the prediction is considered correct, and the match status is updated. Precision is the proportion of correctly predicted line segments out of all predicted line segments, and recall is the proportion of matched annotated line segments.

$$precision \doteq \frac{TP}{TP+FP} \tag{13}$$

$$recall \doteq \frac{TP}{TP+FN} \tag{14}$$

$$p^H \doteq \frac{|P\&G|}{|P|} \tag{15}$$

$$r^H \doteq \frac{|P\&G|}{|G|} \tag{16}$$

$$\mathrm{dist}(\hat{l}, l) = \min\{||\widehat{p_1} - p_1||_2^2 + ||\widehat{p_2} - p_{(2)}||_2^2, ||\widehat{p_2} - p_1||_2^2 + ||\widehat{p_1} - p_2||_2^2\} \tag{17}$$

The proposed model is implemented using Python 3.7, based on the Detectron2 framework. CHead, OHead, AHead, and LHead in the model utilize the head structure of LCNN, which involves a 3×3 convolution, followed by ReLU as the activation function. After activation, features are passed through a 1×1 convolution to transform them to the required output dimensions. OHead and LHead employ the Sigmoid activation function after the conv1x1 convolution to transform the output range to (0,1). Besides, the Softmax function converts the outputs into probability values for the classification task corresponding to heads. The default backbone used is Hourglass, and only one Hourglass Module is utilized, with the depth of the Hourglass Modules set to 4. The HRNet used is constructed using lighter basic blocks.

The model input resolution is 512 × 512, with an output feature map resolution of 128 × 128. The model learning rate is set at 1e-3, with a weight decay of 1e-4. During training, the batch size is as large as possible to improve GPU utilization efficiency, i.e., the default batch size is 24. The model is trained using the AdamW optimizer for 480 epochs, with the learning rate decay utilizing a factor of 10 at 70% and 80% of the total number of iterations. Besides, 5% of the total iterations are used for model warm-up based on the WarmupParamScheduler with a linear growth strategy. The focal loss

α coefficient for midpoint classification is 0.5, the γ coefficient is 5.0, and the error weight λc is 3.0. The error weight for midpoint offset λ_o is set at 0.25, angle sign weight λs at 4.0, angle value weight λv at 0.5, and line length error weight λ_l at 3.0. During model inference, the Line Decoder outputs the top-1000line segments with the highest probability, i.e., K = 1000. Furthermore, softNMS uses a 3 × 3 window for non-maximum suppression to minimize duplicate line segments, with the probability decay coefficient δ set at 0.8. The StructNMS line segment threshold is 2.0.

The model is trained on an Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz and an NVIDIA GeForce RTX 3090 GPU. When measuring the processing speed of different models, a TITIAN V GPU and an Intel(R) Xeon(R) Silver 4116 CPU @ 2.10GHz are used.

5 Experimental Results and Evaluation

5.1 Experimental Results

Table 2 compares the performance of the proposed F-Clip++ algorithm with current algorithms. F-Clip++ uses Hourglass as the backbone network and has improved the 1×1 convolutions in the bottleneck through reparameterization and the channel attention mechanism (SE module). F-Clip++ (Lite) reduces the model's parameter count and computational load by lowering the channel cardinality from 64 to 32, reducing the parameters from 12.6M to 3.1M and the computational load from 79.1 Gflops to 19.9 Gflops. F-Clip++ (HR) uses HRNet as the backbone network and does not apply reparameterization and SE modules due to the already substantial parameter count and computational load of HRNet. F-Clip (HG1) is similar to F-Clip++, but F-Clip++ introduces a continuous angle error function. F-Clip++ (HR) adds the continuous angle error function compared to F-Clip (HR), maintaining HRNet as the backbone network.

All algorithms were trained on 5000 images from the ShanghaiTech dataset and tested on the remaining images. Moreover, their generalization performance was evaluated on the YorkUrban dataset without using this dataset for training or fine-tuning. Notably, LSD, as a non-learning-based algorithm, does not require training. Wireframe detection performance is assessed using the sAP and APH metrics, with sAP utilizing structNMS for deduplication and APH using LCNN's post-processing for deduplication. The results highlight that F-Clip++ has a significant performance improvement over F-Clip. F-Clip++ (HR), with the use of structNMS, performs about 0.9% better on ShanghaiTech than F-Clip (HR), and even without using structNMS, it matches the performance of F-Clip (HR) with structNMS due to the continuous angle error function. F-Clip++ surpasses the original F-Clip (HR) by incorporating reparameterization and attention mechanisms, achieving a sAP of 66.7. Although F-Clip++ has a slightly higher parameter count and computational load than HAWPv2, it is faster to execute than HAWPv2. Without structNMS, the sAP performance of F-Clip++ slightly drops to 65.8, on par with HAWPv2, but its processing speed increases to 63.7, higher than HAWPv2's 34.2. LCNN, despite having a smaller parameter count, has a larger computational load, likely due to its two-stage method. F-Clip++ also exhibits the best sAP performance on YorkUrban, demonstrating excellent generalization capabilities.

Table 2. Wireframe detection performance comparison

Method	ShanghaiTech			YorkUrban			FPS	#Params	Flops
	sAP^5	sAP^{10}	AP^H	sAP^5	sAP^{10}	AP^H		(M)	(G)
Two-stage methods									
LSD (320)	6.7	8.8	52.0	7.5	9.2	51.0	26.3	/	/
AFM	18.5	24.4	69.2	7.3	9.4	48.2	/	42.3	58.7
DWP	3.7	5.1	67.8	1.2	2.1	51.0	/	32.8	57.4
L-CNN	58.9	62.9	83.4	28.5	31.2	64.5	9.7	14.3	72.3
HAWP	62.5	65.6	84.2	28.6	31.2	62.4	10.1	10.3	62.2
HAWP v2	65.7	69.7	88.0	28.8	31.2	64.6	11.3	11.5	62.2
One-stage methods									
TP-LSD (Res34-Lite)	56.4	59.7	/	24.8	26.8	/	72.8	23.2	48.3
TP-LSD (Res34)	57.5	60.0	/	25.0	27.4	/	41.1	23.3	72.5
ELSD	64.3	68.9	87.3	28.7	32.9	62.6	-	-	-
LETR	59.2	65.6	86.3	25.6	28.4	62.7	4.5	55.6	190.7
F-Clip (HG1)	58.6	63.6	83.0	26.3	28.9	60.3	25.5	4.2	33.1
F-Clip (HR)	64.3	68.3	85.7	28.5	30.8	65.0	5.0	28.5	41.1
F-Clip++	66.7(65.8)	70.8(69.8)	88.0	29.8(29.6)	32.3(32.0)	59.9	20.9(63.7)	12.6	79.1
F-Clip++ (Lite)	62.4(61.7)	66.9(66.2)	86.0	27.0(26.8)	29.6(29.3)	60.1	21.6(70.0)	3.1	19.9
F-Clip++ (HR)	65.2(64.3)	69.8(68.9)	87.9	26.8(26.7)	29.4(29.2)	59.6	13.0(21.8)	28.5	41.1

5.2 Visual Comparison

Fig. 5 displays the prediction results of different algorithms on the ShanghaiTech test dataset, where all line segments with a prediction probability below 0.05 were removed. The first two rows of each subplot show the model's performance detecting wireframes in outdoor scene images, while the last three rows show the performance in indoor scenes. Fig. 5(h) displays the annotation information for this dataset, which primarily includes the structural lines in the images. The results suggest that the traditional algorithm LSD detects all line segments in the images, including structural and texture lines, such as the striped texture of the sofa shown in the figure. LSD also tends to break the lines in the images into multiple segments due to its line support region discovery method, which easily divides a line area into several blocks. Additionally, the algorithm lacks inferential capability. For instance, it cannot deduce the structural lines between the horizontal and vertical surfaces of the coffee table and sofa in the last image in Fig. 5(a), a task that other learning-based algorithms can more easily accomplish. LETR exhibits numerous duplicate predictions of the same line segment and has more significant prediction errors between the predicted line segments. This is not conducive to subsequent deduplication processing using post-processing. In Fig. 5(g), F-Clip++ demonstrates a noticeable improvement in line segment angle prediction compared to other algorithms. Still, it has a few duplicate predictions, such as in the third image of Fig. 5(g) in indoor scenes where multiple parallel redundant predictions are on the edges of a chair. A solution for this is increasing the size of the midpoint NMS convolution kernel. Comparing Fig. 5(g) F-Clip++ with Fig. 5(e) HAWP v2, both models perform well but exhibit different errors. Specifically, F-Clip++ duplicates line segment

predictions, while HAWP v2 mispredicts some line segments. Fig. 5 highlights that the wireframe effectively reflects the geometric information of the scenes, clearly delineating the contours of objects.

(a) LSD (b) LCNN (c) TPLSD (d) LETR

(e) HAWP v2 (f) F-Clip (g) Ours (h) GT

Fig. 5. Wireframe detection algorithm visual comparison

6 Conclusion

Applying wireframe detection in construction monitoring represents a significant advancement in the automation and efficiency of the construction process. The proposed F-Clip++ algorithm effectively leverages deep learning technology to extract line segment features from complex architectural images, significantly reducing manual intervention. This enhancement accelerates the monitoring process and improves accuracy, facilitating the rapid identification of discrepancies between the BIM and actual installations.

The continuous angle error function introduced in F-Clip++ addresses a key challenge in angle prediction, ensuring robust model performance in various scenarios. Therefore, the model demonstrates superior capability in identifying geometric structures and layouts, which is crucial for understanding architectural design and improving construction precision. Furthermore, this research highlights the importance of adopting advanced monitoring technologies to address the inefficiencies and inaccuracies inherent in traditional methods, particularly in monitoring the quality of concealed engineering installations. By providing precise data support for subsequent comparative analysis, F-Clip++ lays a solid foundation for the intelligent development of the construction industry.

Overall, this study underscores the transformative potential of computer vision and deep learning in enhancing the process of architectural quality control, paving the way for safer and more efficient construction practices. Future work will focus on expanding the applicability of the F-Clip++ model in various architectural settings and further optimizing its algorithms to enhance performance and adaptability in real-time monitoring scenarios.

Acknowledgements

This work was financially supported by the 2022 Stable Support Plan Program for Shenzhen-based 378 Universities (20220815150554000).

References

1. Yang, J., Park, M., Vela, P. A., Golparvar-Fard, M. (2015) Construction performance monitoring via still images, time-lapse photos, and video streams: Now, tomorrow, and the future. Advanced Engineering Informatics, 29(2), pp. 211-224. https://doi.org/10.1016/j.aei.2015.01.011.
2. Huang, K., Wang, Y., Zhou, Z., et al. (2018) Learning to parse wireframes in images of man-made environments. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, pp. 1234-1243. https://doi.org/10.1109/CVPR.2018.00134.
3. Kong, N., Park, K., Goka, H. (2021) Hole-robust wireframe detection. In: 2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), pp. 2684-2693. https://doi.org/10.1109/WACV51458.2022.00268

4. Zhou, Y., Qi, H., Ma, Y. (2019) End-to-end wireframe parsing. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 1234-1243. https://doi.org/10.1109/ICCV.2019.00123.
5. Ren, S., He, K., Girshick, R., et al. (2015) Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in Neural Information Processing Systems, 28, pp. 91-99. https://doi.org/10.5555/512775.512791.
6. Li, H., Yu, H., Wang, J., et al. (2021) ULSD: Unified line segment detection across pinhole, fisheye, and spherical cameras. ISPRS Journal of Photogrammetry and Remote Sensing, 178, pp. 187-202. https://doi.org/10.1016/j.isprsjprs.2021.05.013.
7. Xue, N., Wu, T., Bai, S., et al. (2020) Holistically-attracted wireframe parsing. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 2788-2797. https://doi.org/10.1109/CVPR42600.2020.00288.
8. Xue, N., Wu, T., Bai, S., et al. (2022) Holistically-attracted wireframe parsing: From supervised to self-supervised learning. arXiv:2210.12971. https://doi.org/10.48550/arXiv.2210.12971.
9. Gillsjö, D., Flood, G., Åström, K. (2022) Semantic room wireframe detection from a single view. In: 2022 26th International Conference on Pattern Recognition (ICPR), pp. 1886-1893. https://doi.org/10.1109/ICPR56361.2022.00192.
10. Zhang, Z., Li, Z., Bi, N., et al. (2019) PPGNet: Learning point-pair graph for line segment detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1234-1243. https://doi.org/10.1109/CVPR.2019.00123.
11. Von Gioi, R. G., Jakubowicz, J., Morel, J.-M., et al. (2008) LSD: A fast line segment detector with a false detection control. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(4), pp. 722-732. https://doi.org/10.1109/TPAMI.2008.139.
12. Pautrat, R., Barath, D., Larsson, V., et al. (2022) DeepLSD: Line segment detection and refinement with deep image gradients. arXiv preprint arXiv:2212.07766. https://doi.org/10.48550/arXiv.2212.07766.
13. Teplyakov, L., Erlygin, L., Shvets, E. (2022) LSDNet: Trainable modification of LSD algorithm for real-time line segment detection. IEEE Access, 10, pp. 45256-45265. https://doi.org/10.1109/ACCESS.2022.3172210.
14. Dai, X., Gong, H., Wu, S., et al. (2022) Fully convolutional line parsing. Neurocomputing, 506, pp. 1-11. https://doi.org/10.1016/j.neucom.2021.09.021.
15. Zhang, H., Luo, Y., Qin, F., et al. (2021) ELSD: Efficient line segment detector and descriptor. CoRR, abs/2104.14205. https://doi.org/10.48550/arXiv.2104.14205.
16. Huang, S., Qin, F., Xiong, P., et al. (2020) TP-LSD: Tri-points based line segment detector. CoRR, abs/2009.05505. https://doi.org/10.48550/arXiv.2009.05505.
17. Xu, Y., Xu, W., Cheung, D., et al. (2021) Line segment detection using transformers without edges. CoRR, abs/2101.01909. https://doi.org/10.48550/arXiv.2101.01909.
18. Carion, N., Massa, F., Synnaeve, G., et al. (2020) End-to-end object detection with transformers. In: Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I, pp. 213–229. https://doi.org/10.1007/978-3-030-58452-8_13.
19. Xue, N., Bai, S., Wang, F., et al. (2019) Learning attraction field representation for robust line segment detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1595-1603. https://doi.org/10.1109/CVPR.2019.00167.
20. Gu, G., Ko, B., Go, S., et al. (2022) Towards light-weight and real-time line segment detection. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1234-1241. https://doi.org/10.1609/aaai.v36i1.19730

21. Ding, X., Zhang, X., Ma, N., et al. (2022) RepVGG: Making VGG-style ConvNets great again. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1234-1243. https://doi.org/10.1109/CVPR52688.2022.00001.
22. He, K., Zhang, X., Ren, S., et al. (2015) Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385. https://doi.org/10.48550/arXiv.1512.03385.
23. Wang, J., Li, F.,Bi, H. (2022) Gaussian focal loss: Learning distribution polarized angle prediction for rotated object detection in aerial images. IEEE Transactions on Geoscience and Remote Sensing, 60, pp. 1-13. https://doi.org/10.1109/TGRS.2022.3141234.