Automatic plane reconstruction using realistic initialization for TILT

Oscar Ortega
University of California, Berkeley
Berkeley, CA, USA
oscar.g.ortega.5@berkeley.edu

Yiteng Zhang
University of California, Berkeley
Berkeley, CA, USA
yiteng@berkeley.edu

Abstract

TILT (Transform Invariant Low-Rank Textures) is a technique for efficiently and effectively extracting a class of low-rank structures from a 3D image which encompass conventional local features such as edges and corners. Experiments have also shown that under a wide variety of differing initialization, this algorithm has a decent range of convergence and is proven to be very robust in most scenarios. However, there exist scenarios where "adversarial" initialization lead the TILT algorithm to fail to achieve proper convergence to a correct solution. We design a pipeline to avoid the scenarios where we may encounter an "adversarial" initialization with edge and line detection. In this paper we propose a pipeline for the heuristic formation of TILT bounding boxes using edge and line detection techniques.

1 Intoduction

The recovery of a low-rank texture and of a domain transformation in a scene as defined in [1], is an optimization procedure whose solution depends on the initialization. More formally, given a deformed image $I = (I^0 + E) \circ \tau^{-1}$, where E is an error matrix, I^0. Our goal is to recover the image I^0 and the domain transformation $\tau \in \mathbb{G}$. This lends us to the following optimization problem.

$$\min_{I^0,E,\tau} \text{rank}(I^0) + \gamma\|E\|_0 \text{ s.t.} I \circ \tau = I^0 + E \quad (1)$$

Although it has been shown that TILT is quite robust, and can converge to an optimal solution under a vast array of initialization, as exemplified in 1, there are some cases where this does not hold.

As noted in 1, in both these cases, the large deformation of the input window as well as including too much background in the image led to the TILT algorithm converging to only an approximately correct situation. For the purposes of our discussion we will refer to the input windows as "adversarial initialization windows". It is noted in [1], that with rough manual adjustments to these "adversarial initialization windows", in both cases adjusting the width of the window, the TILT algorithm managed to produce a correct solution as in 2.

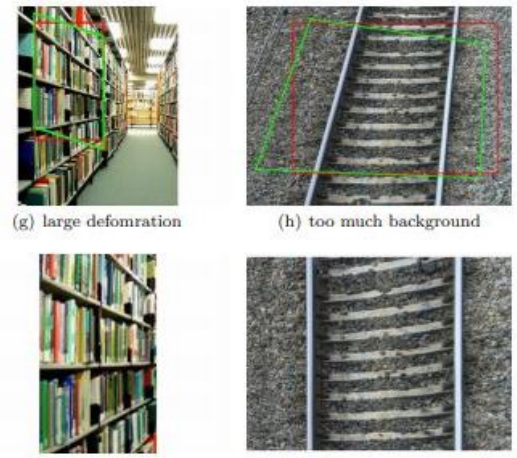


Figure 1: convergence of TILT to approximate solution

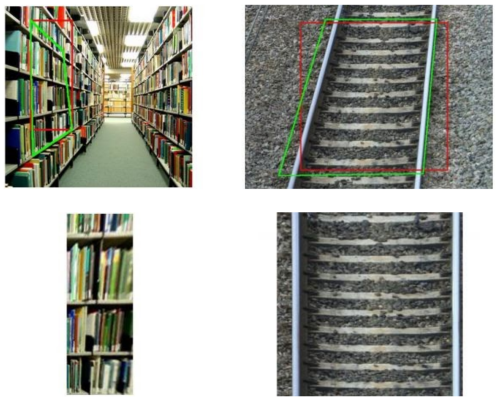


Figure 2: convergence of TILT to correct solution

2 Motivation

2.1 Challenging Initialization

As illustrated in figures 1 and 2, there are scenarios described where the input bounding box is either too large to correct for the perspective distortion of certain images which results in convergence to an incorrect solution.

2.2 Manual Initialization

It is also noted that the initial bounding box selected in the TILT algorithm is manual. Our goal is to automate the selection of the bounding box by taking into account the symmetries present in the scenes through the use of line detection techniques.

2.3 Applications

Although the applications of TILT are wide-spread, we seek to explore the applications of this heuristic initialization of TILT bounding boxes in the field of Augmented Reality.

2.4 Exploration

The brunt of research was into fundamental computer vision techniques to construct a pipeline for line-segmentation and for input into running the TILT algorithm, particularly our research into how one could use Hough Transforms and Canny Edge Detection for the pipeline we designed for our project. These techniques have been available since the 70s and 80s respectively and we decided it would be interesting to use these older, yet fundamental techniques, for our pipeline.

3 Related Work

3.1 Hough Transformation

The classical Hough Transform [2] is a feature extraction technique commonly used in the fields of image analysis and computer vision. The purpose of this technique is to find imperfect instances of objects within a certain class of shapes via a voting procedure. Although the Hough transform can be extended to finding the positions of arbitrary shapes, for our purposes our proposed pipeline is concerned with the identification of lines within an image. This is a achieved by assuming each input measurement indicates its contribution to a globally consistent solution. This is analytically achieved by using a polar representation of a line. The set of ponts (r, θ) in this context is referred to as the *Hough space* of an image.

$$x\cos(\theta) + y\sin(\theta) = r \tag{2}$$

Where r is the length of a normal from the origin to this line and θ is the orientation of r with respect to the x axis. The algorithm then uses a two-dimensional array to detect the existence of lines as parameterized in 2. Given an image, we intend to use the point cloud representation of the edges given by an edge detection technique, such as Canny Edge Detection, to extract lines using the Hough Transform.

3.2 Edge Detection

The first steps of our pipeline include edge detection. So, we decided to look into the following techniques for the detection of edges.

3.2.1 Roberts Cross Operator

The Roberts cross operator seeks to approximate the gradient of an image through discrete differentiation achieved by computing the sum of squares of differences between diagonally adjacent pixels [3]. To perform a Roberts edge detection, one convolves the original image with the following kernels:

$$x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3}$$

$$y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{4}$$

If we let $G_x(x, y)$ be a point in an image convolved by x and $G_y(x, y)$ be a point in an image convolved by y. We can then evaluate the gradient of the image at a point (x, y) as follows:

$$\nabla I(x, y) = \sqrt{(G_x^2 + G_y^2)(x, y)} \tag{5}$$

3.2.2 Sobel Transform

The Sobel operator seeks to approximate the gradient of in a similar manner to that of the Robert [4][3]. To perform a Sobel edge detection, one convolves the original image with the following kernels:

$$x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{6}$$

$$y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{7}$$

Similarly to the Robert Edge Detection described above, after performing the convolution in the manner described, we then compute the image gradients using equation (6) and (7) .

3.2.3 Canny Algorithm

The edge detection procedure proposed by John Canny [5][3] consists of 5 different steps :

1. Apply a Gaussian filter to smooth the image.
2. Find intensity gradients through the use of edge detection operators such as the Sobel operator.
3. Thin potential edges to width of 1 pixel.
4. Track edges by hysteresis and filtration of weak edges that are not connected to strong edges.

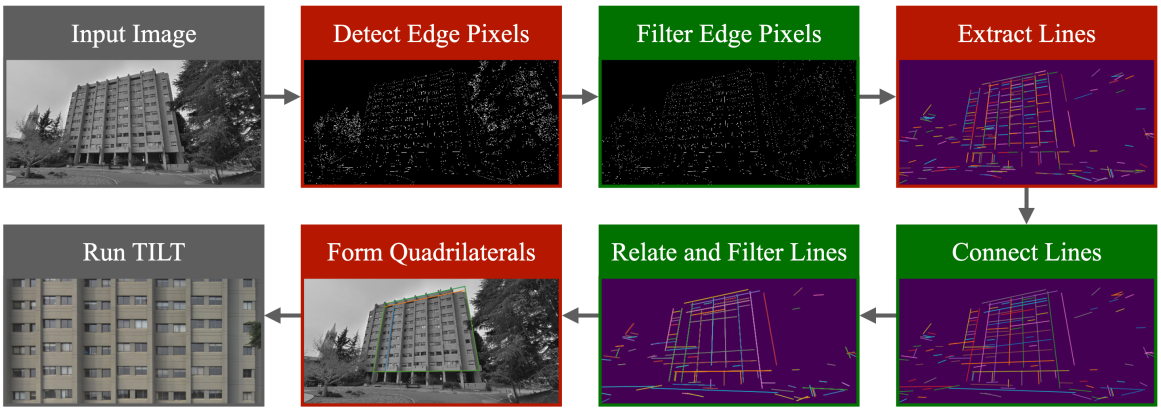


Figure 3: our proposed pipeline for TILT initialization

3.3 Inscribing Rectangles

We explored the two following algorithms for inscribing rectangles as part of our pipeline.

3.3.1 Finding largest rectangles in convex polygons

The following geometric optimization problem proposed in [6] seeks to a find a maximum-area rectangle that solves this problem in $\mathcal{O}(n^3)$ time. Alternative $(1-\epsilon)$-approximation algorithms are also proposed which take $\mathcal{O}(\epsilon^{-\frac{3}{2}} + \epsilon^{-\frac{1}{2}} \log n)$ time are proposed.

3.3.2 Algorithm for finding the largest inscribed rectangle in polygon

The following geometric optimization proposed by [7] presents the problem of finding the largest inscribed rectangle in a polygon in a manner that is simpler compared to comparable work in the field and shows better results on some images. The four key steps of the algorithm are as follows:

1. Dividing polygon into subareas
2. Identifying rectangular subareas
3. Forming RA graph
4. Identifying cycles and paths and identifying the largest rectangle.

4 Technical Outline

The TILT algorithm takes in an image and a quadrilateral bounding box, so in order to automatically initialize TILT, we need to find the bounding box automatically. Here is our proposed pipeline for doing that.

There are three key steps in our pipeline: detect edge pixels from input image, extract lines from detected edge pixels, form bounding boxes from the extracted lines. However, these steps along could not produce promising bounding boxes (there are some examples in the section 6.1), so we also added three helper steps to filter noise and augment the results along the way. With these helps, we are then able to reach good initialization for the TILT algorithm.

4.1 Edge Detection

The first step is to detect edge pixels from the input image using either Canny, Sobel, or Roberts Edge Detectors. We found that the flexibility offered by the ability to tune the Canny edge detection led to better results when using the Canny edge detection in our pipeline. Both the Roberts and Sobel edge detection methods suffered from very high density of the detected edges from our input images and became a bottleneck in the run-time of our proposed pipeline. For the purposes of our experimentation, we found that that Canny Edge Detection with the following hyperparameter settings performed best:

- $\sigma = 1$
- high threshold $= 0.95$
- low threshold $= 0.9$

Any point above the high threshold will be labeled as edges. Any point above the low threshold that is 8-connected to a labeled point will be labeled as edges.

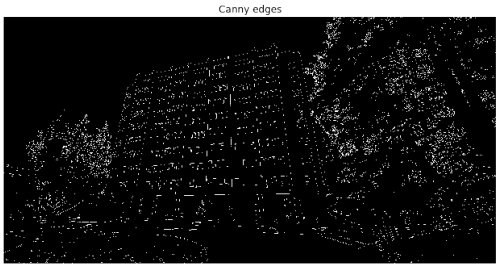


Figure 4: detected edge pixels

4.2 Edge Filtration

After the we receive the point clouds from the edge detection, the next step is to filter the corresponding edges. As we can see from the results, the noisy area have much more local density than the other area, so in order keep the general structure and remove noises, we just need to remove edge pixels that are surrounded in a dense area.

Given an image of detected edge pixels, for each edge pixel in the image, filter the edge pixel if:

- more than 14 edge pixels in the 7x7 matrix around the edge pixel
- more than 24 edge pixels in the 11x11 matrix around the edge pixel
- more than 34 edge pixels in the 15x15 matrix around the edge pixel

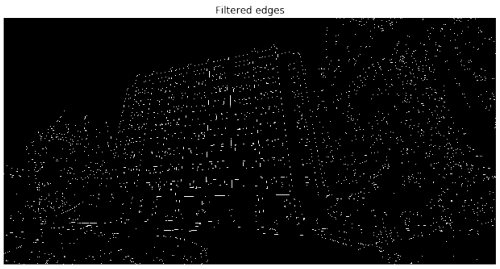


Figure 5: filtered edge pixels

After filtering, the edge pixels that form lines remain there, while the noises are much reduced as noted by figure 5.

4.3 Line Extraction

After filtering the noises, we then run progressive probabilistic Hough transform on the image of edge pixels to begin our process of line extraction. We found great success with the following hyperparameter settings:

- maximum gap $= 20$
- minimum length $= 80$

The parameters define the maximum allowable gap between potential edge points in the construction of a line and the minimum length of a line in pixels accordingly. Note that we should adjust these parameters accordingly for different image resolutions. As one can see, if the maximum-gap parameter is too large, then there will be too much noise line segments generated, but if we lower the gap in order to suppress invalid line segments, then some line segments that are intended to be connected will also be disconnected.

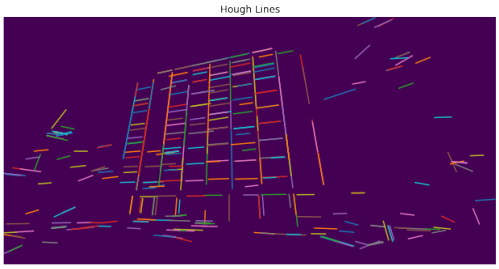


Figure 6: extracted lines

4.4 Line Connection

We defined a mechanism to connect the short line segments to form the longest possible line segments. For any two line segments in the input, we connect them if they can be added up to form a longer line segment. Specifically, the lines need to satisfy four requirements.

1. First, if lines should have a similar orientation, so the difference in their gradient or slope has to be minimal. Here we set the maximum differences in angles to be 15 degrees.
2. Also, they need to be close to each other. So the shortest connected line between two lines need to be below a threshold. We set the threshold to be 80% of length of the input lines.
3. Finally, there should be minimal shift between two lines. For example, if both lines are horizontal, then we should only connect them if their y-intercept are close to each other. We checked this by first calculating the center of the four vertices from two lines, and check whether the distance between the center and two lines are both below a threshold. We set the threshold to be 10 pixels.

After we connected all line segments using the algorithm above, we will successfully eliminate all shorter line segments and form the longest line segments possible.

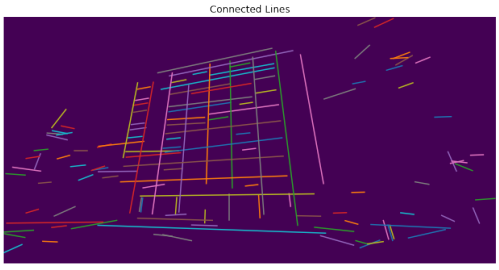


Figure 7: connected lines

4.5 Line Relation and Filtration

Before we actually construct the quadrilateral bounding boxes, we first register the relations between line

segments and filter out those line segment that are isolated.

We relate the two end points of two line segments if they can potentially form the two adjacent sides of a quadrilateral bounding box. Specifically, they need to satisfy two requirements:

1. There is enough difference in their orientation. This means that their gradient or slope has to be different, to avoid relate two parallel or almost-parallel lines. Here I set the differences in angles to be between 30 and 150 degrees so it also comply with distorted planes.

2. Also, they need to be close to each other. So the distance between the two end points need to be below a threshold. TODO: what was this threshold? what would happen if this was not the threshold?

After we relate all line segments, we then remove all other line segments with no relation to any other line segments.

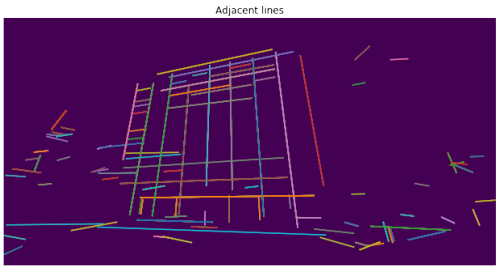


Figure 8: filtered lines

4.6 Quadrilateral Formation

If there are enough relations among a set of lines segments, then we use these line segments to form a quadrilateral bounding box. There are two cases:

1. If there are four line segments where each end point of each line segment is related to another end point of another adjacent line segment without overlap, then we can easily form a quadrilateral by using their intersections as the four vertices of the quadrilateral.

2. Otherwise, if there are three line segments where one is related to each of the other two on each end point, then we construct a quadrilateral by setting the two intersections as two vertices of the quadrilateral, and the two unconnected end points as the other two vertices of the quadrilateral. Note that we don't need perfect quadrilaterals here because these quadrilateral just help us to generate better rectangular bounding boxes as the input of the TILT algorithm.

Figure 9: formed quadrilaterals

4.7 Run TILT

We select the largest area quadrilateral from the previous steps, and run the TILT algorithm on an image I and this quadrilateral (shown in red). The TILT algorithms then produces the best-fit low-rank quadrilateral (shown in green) and the recovered plane based on homography.

Figure 10: TILT result

Figure 11: recovered plane

5 Results

We performed the procedure detailed on the Facades dataset provided by [1]. The experiment can be broken into a few key steps as follows:

1. For each of the following initialization methods, we select 20 random samples from the building facades dataset provided in [1].

2. For each sample, we run the selected pipeline on the sample, generating the bounding boxes for which to run the TILT algorithm.

3. Given the bounding boxes for the image, we then run TILT with the provided bounding boxes as input.

We then did qualitative comparisons on the corresponding output quadrilaterals for each of the proposed pipelines.

We present more examples of our results in Appendix A.

5.1 Comparison to Baseline Initialization (Canny + Hough)

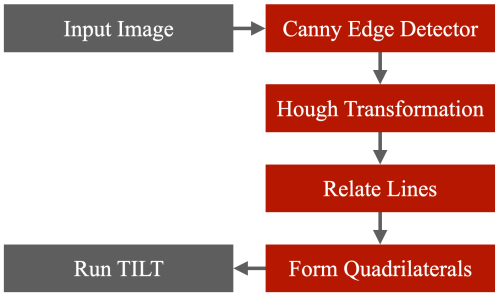
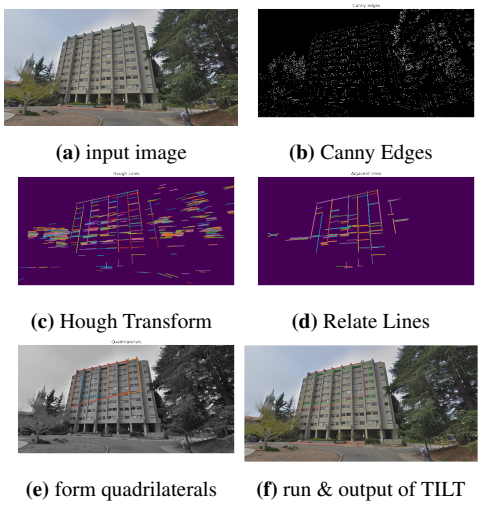


Figure 12: baseline pipeline for TILT initialization

The structure of the baseline model is simple. We used a Canny edge detector and a Hough transform to replace the step 1-4 in our pipeline. Note that this change causes significant amount of noises in the output, so we fine-tuned the hyperparameters specifically for this baseline model to increase result quality.



(a) input image **(b)** Canny Edges

(c) Hough Transform **(d)** Relate Lines

(e) form quadrilaterals **(f)** run & output of TILT

Figure 13: Step-by-step visualization of baseline TILT initialization (Hough + Canny) without edge filtering on Evans Hall, UC Berkeley

When comparing the results of the baseline pipeline to the other techniques proposed it should be noted that this pipeline suffered from a longer run-time, most likely due to lack of filtered edges that in this version of the pipeline. Similarly, due to the lack of filtered edges, there are a lot more lines-generated from the Hough Transform.

At the same time, since the baseline models lacks a mechanism of connecting shorter lines to form longer lines, this leads to in an inability for the present algorithm to produce an initialization which encompasses the full low-rank structure on the facade of Evans Hall.

5.2 Comparison to Rectangular Initialization

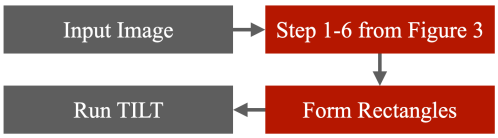


Figure 14: our pipeline for rectangular TILT initialization

We propose three different methods to extract rectangular bounding boxes from quadrilateral bounding boxes.

5.2.1 Perimeter Averaging

To perform the rectangle extraction via perimeter averaging we perform the following three steps:

1. We compute x_{left} and x_{right}, the average x values of the two vertices from the left and right sides of the corresponding quadrilateral.

2. We compute y_{top} and y_{bottom}, the average y values of the two vertices from the top and bottom sides of the corresponding quadrilateral.

3. We then use the four computed values to construct the four points of the rectangle $(x_{\text{left}}, y_{\text{top}})$, $(x_{\text{right}}, y_{\text{top}})$, $(x_{\text{right}}, y_{\text{bottom}})$, and $(x_{\text{left}}, y_{\text{bottom}})$.

5.2.2 Rectangle Extraction using Area and Corner based methods

For both the Area based and Corner based rectangle extraction methods, we perform the following procedure to extract the input windows needed to run the TILT algorithm.

1. We first extract the bounding rectangles of the corresponding quadrilateral via four steps:

   (a) We first compute the outer bounding rectangle, the SAR, of the input quadrilateral.

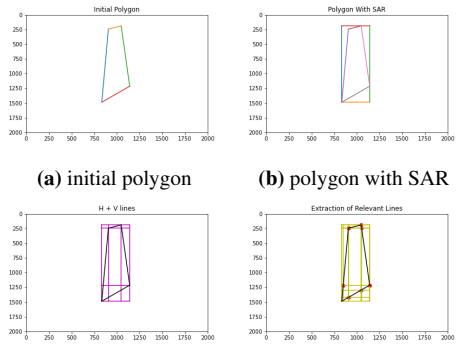
(b) For every edge of the corresponding quadrilateral, we then construct both horizontal and vertical line segments which intersect at the corresponding edge. Each line segment is bounded by the SAR of the input quadrilateral.

(c) For every horizontal and vertical line segment constructed in the step above, we determine where the line segment intersects the corresponding quadrilateral. If the corresponding line-segment intersects at a point x, we then we construct either a vertical line segment in the case our line segment is a horizontal line segment, and vice-versa. Again, our line segments will be bounded by the SAR of the corresponding quadrilateral.

(d) We then enumerate the rectangles determined by the intersections of the corresponding horizontal and vertical lines constructed in steps a-c.

2. We then perform a filtering of the irrelevant rectangles based on either an area or corner based thresholding.

3. From the set of relevant rectangles, we then extract the maximum area rectangle as input into the TILT algorithm.



(a) initial polygon **(b)** polygon with SAR

(c) horizontal and vertical **(d)** full line extraction

Figure 15: Step-by-step visualization of step 1 of the procedure for the rectangle extraction process when using either corner or area based thresholding.

5.2.3 Area-based Thresholding

We propose the following algorithm for performing an area based filtration of the set of rectangles generated as part of step one.

Algorithm 1: Area-based Thresholding

Result: $\mathcal{R}'$: a filtered set of rectangles
Input:
$\mathcal{R}$: a set of rectangles
γ: a thresholding hyperparameter $\in [0, 1]$
Q: input quadrilateral;
for $R \in \mathcal{R}$ **do**
  Let $r = \text{area}(R)$ *;*
  Let $r' = \text{area}(R \bigcap Q)$*;*
  Let $a = \frac{r'}{r}$*;*
  if $a \leq \gamma$ **then**
    remove R from $\mathcal{R}$*;*
  end
end
Let $\mathcal{R}' = \mathcal{R}$*;*

5.2.4 Corner-based Thresholding

We also propose the following algorithm for a corner based thresholding of the set of rectangles.

Algorithm 2: Corner-based Thresholding

Result: $\mathcal{R}'$: a filtered set of rectangles
Input:
$\mathcal{R}$: a set of rectangles
k: a thresholding parameter $\in \{1, 2, 3, 4\}$
Q: input quadrilateral;
for $R \in \mathcal{R}$ **do**
  Let $k' = 0$ *;*
  for $\text{edge}(e) \in R$ **do**
    if $e \in \text{interior}(Q)$ **then**
      $k' = k' + 1$ *;*
    end
  end
  if $k' \leq k$ **then**
    remove R *from* $\mathcal{R}$*;*
  end
end
Let $\mathcal{R}' = \mathcal{R}$*;*

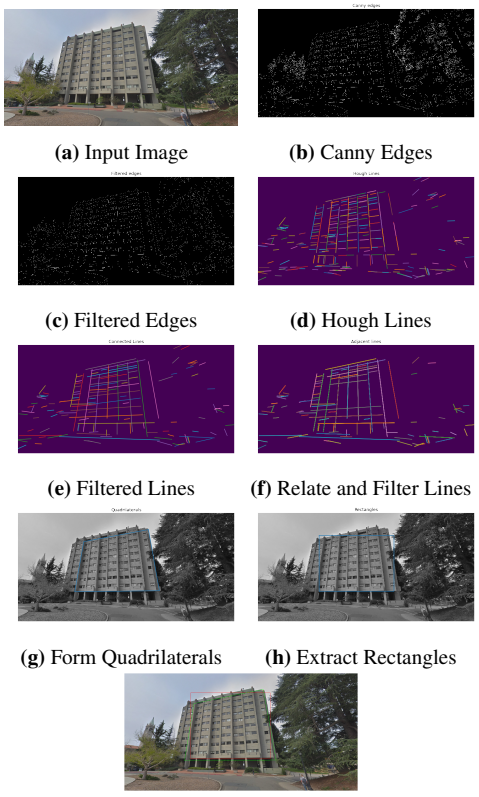
5.2.5 Comparison of Corner versus Area based thresholding

One can see that setting $k = 4$ and $\gamma = 1.0$ results in the same filtered set of rectangles. This is intuitive as it will always be the case that a fully inscribed rectangle will have its whole area in the quadrilateral as well as its four corners inside the rectangle.

5.2.6 Results with Rectangular Initialization

When comparing the results of Rectangular Initialization pipeline compared to the other techniques proposed, it should be noted that the various rectangle-

extraction techniques, particularly suffer from the problem of eliminating the structure encoded by the quadrilaterals produced in previous steps of the pipeline. This was noticed predominantly when using the Perimeter Averaging method of rectangle extraction. Although the run-time of this pipeline was still on the order of magnitude of seconds rather than minutes like the L-CNN based pipeline, it should be noted that this method has the added overhead of extracting the rectangle from the quadrilateral.



(a) Input Image **(b)** Canny Edges

(c) Filtered Edges **(d)** Hough Lines

(e) Filtered Lines **(f)** Relate and Filter Lines

(g) Form Quadrilaterals **(h)** Extract Rectangles

(i) input & output of TILT

Figure 16: Rectangular TILT initialization using Perimeter Averaging on Evans Hall, UC Berkeley

5.3 Comparison to Neural-based Wireframe Parsing Initialization

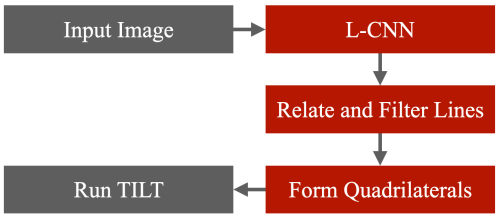
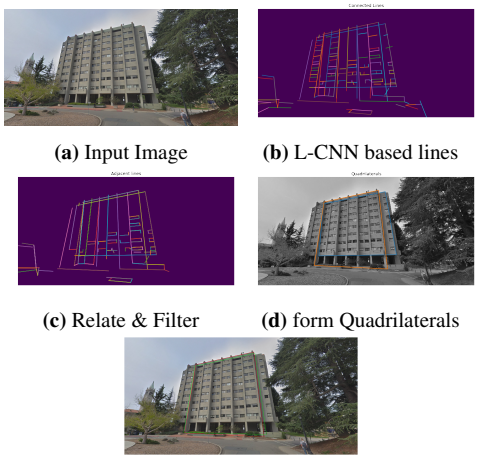


Figure 17: L-CNN based pipeline for TILT initialization

In this pipeline, we applied the L-CNN [8], which is a neural network that takes in an image and returns detected wireframes. We use this model to replace the step 1-4 in our pipeline. The results are shown below.



(a) Input Image **(b)** L-CNN based lines

(c) Relate & Filter **(d)** form Quadrilaterals

(e) input & output of TILT

Figure 18: TILT initialization with the L-CNN based pipeline on Evans Hall, UC Berkeley

When comparing the results of L-CNN-based pipeline compared to the other techniques proposed, one strength compared to other techniques is that the continuity of the lines proposed by the L-CNN is superior to the other methods analyzed, including our own. However, as a result, many detected lines do not follow the real geometric edges in the image. Another weakness of this approach is that compared to our proposed framework, the time to run the pipeline is on the order of magnitude of minutes, rather than of the approximately 30 seconds our pipeline would take to run.

6 Future Work

Our experiments can easily be extended to include a more fine-grain tuning of the edge detection provided in our pipeline for the extraction of bounding boxes to run the TILT algorithm. Another possible path of experimentation would be to perform more in depth analysis of the extraction of TILT bounding boxes using different methods of line segmentation. The technique outlined in [8] also explores the use of quantitative metrics to compare our methods with other similar techniques.

References

[1] Z. Zhang, A. Gansesh, X. Liang, and Y. Ma, "Tilt: Transform invariant low-rank textures," *ArXiv*, 2010.

[2] R. O. Dude and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Stanford Research Institute*, 1971.

[3] G. Shrivakshan and C. Chandrasekar, "A comparison of various edge detection techniques used in image processing," *International Journal of Computer Science Issues*, vol. 9, pp. 269–276, 09 2012.

[4] N. Kanopoulos, N. Vasanthavada, and R. L. Baker, "Design of an image edge detection filter using the sobel operator," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 358–367, 1988.

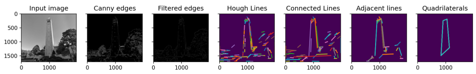
[5] J. Canny, "A computational approach to edge detection," *IEEE*, 1986.

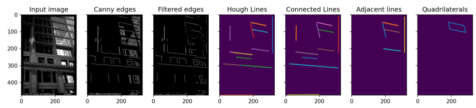
[6] S. Cabello, O. Cheong, C. Knauer, and L. Schlipf, "Finding largest rectangles in convex polygons," *ArXiv*, 2014.

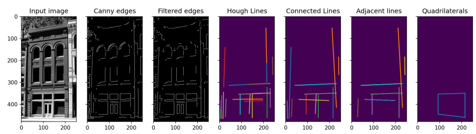
[7] Z. Marzeh, M. Tahmasbi, and N. Mirehi, "Algorithm for finding the largest inscribed rectangle in polygon," *Journal of Algorithms and Computation*, 2019.

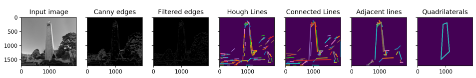
[8] Y. M. Yichao Zhou, Haozhi Qi, "End-to-end wireframe parsing," *ArXiv*, 2020.

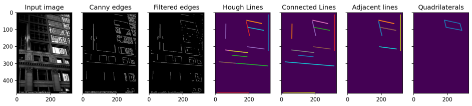
Appendix

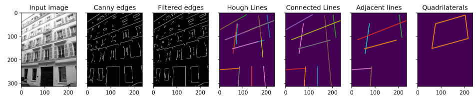


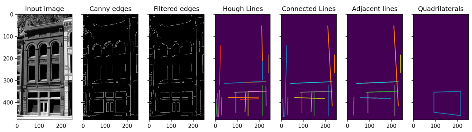


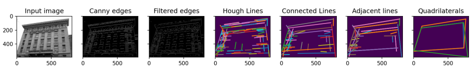


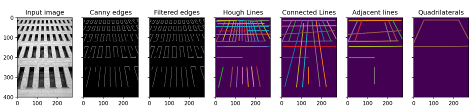



Input image
Canny edges
Filtered edges
Hough Lines
Connected Lines
Adjacent lines
Quadrilaterals


Input image
Canny edges
Filtered edges
Hough Lines
Connected Lines
Adjacent lines
Quadrilaterals


Input image
Canny edges
Filtered edges
Hough Lines
Connected Lines
Adjacent lines
Quadrilaterals


Input image
Canny edges
Filtered edges
Hough Lines
Connected Lines
Adjacent lines
Quadrilaterals


Input image
Canny edges
Filtered edges
Hough Lines
Connected Lines
Adjacent lines
Quadrilaterals