Identifying the Vanishing Points of Surgical Tools via Conic Convolution Using a Neural Network

Bhavani Venkatesan
Pioneer Academics
bhavaniv1101@gmail.com

Abstract

Robot-assisted minimally invasive surgery (RMIS) is becoming increasingly popular due to its numerous advantages, such as higher precision and shorter patient recovery time. However, one of the main drawbacks is the lack of force feedback from the surgical tool. Force sensors are not a viable solution because they tend to be very expensive and can only be used once. A new idea that could enable the widespread adoption of RMIS is using computer vision to get force feedback. Finding the vanishing point of the surgical tool is a crucial step to get accurate results. The vanishing point is the point where the two parallel edges of the tool seem to converge. In this paper, a novel idea was implemented to find the vanishing point of surgical tools using a neural network. The datasets were created by the author as there no datasets were found online. In real life, surgical tool images are often obstructed due to tissues and blood; this is called occlusion. The model was trained on a dataset of occluded images and a data set of unoccluded images. The results showed that the model was able to learn equally well for both datasets and also performed well when cross-tested on the two datasets. This shows that this approach has a significant advantage over traditional methods of identifying the vanishing point, as the results are not affected by occlusion. This approach has a lot of potential to enable the application of computer vision to improve RMIS, leading to better outcomes for millions of patients.

1 Introduction

1.1 Motivation

Robot-assisted minimally invasive surgeries (RMIS) are becoming more and more popular with advancing technologies and have significant advantages, such as increased stability and precision, which leads to shorter recovery time and less medication for patients. However, one major drawback is that there is currently no technology that can effectively give force feedback from the surgical tool to the surgeon [1] [2]. This can lead to unintentional tissue damage, as too much force might be applied. While force sensors can be used, they need to be sanitized at high temperatures after each procedure, so they can be used only once. Therefore, they are very expensive, making them an unviable solution.

A new approach that has been suggested is to get force feedback using computer vision. In this process, tool segmentation is used to locate the tool in the image. The deformation of tissues around the tool tip is then used to predict the force being applied. [3]. It has been shown that tool segmentation is more effective when the image is transformed to polar coordinates with the tool's vanishing point (shown in Figure 4) as the center [4]. Therefore, identifying the vanishing point is a critical part of the process. Section 1.2 explains the different methods that have been used to identify the vanishing point. Using a neural network is expected to be the most effective method, as it can be used to detect the vanishing point in segmented tools and in images with multiple tools, and they may be less likely to be affected by occlusion. They can make sophisticated decisions because of their consecutive, interconnected layers; this is something which cannot be done using any of the other methods listed below.

1.2 Related Work

Based on the author's research, the following three methods to detect vanishing points seem to be the most relevant. However, each method has a few drawbacks and limitations which are discussed below.

Minimum Area Enclosing Triangle One of the methods that is currently used to find the vanishing point of surgical tools is the minimum area enclosing triangle method [4]. However, this method can only be used for images with a single unjointed tool, as a single triangle is drawn around the entire tool. If there are multiple tools or tools with joints, the triangle will be drawn around all the tools and joints, leading to an incorrect calculation of the vanishing point. Therefore, this method has limited use and is not always accurate.

Edge Detection Another method that has been used to find the vanishing point in images with natural scenes (e.g. roads and buildings) is getting the edge image of the picture, finding all the relevant lines through filtering with respect to angle and length, and mathematically calculating the vanishing point based on the longest lines [5]. This method works for natural scenes because all the lines generally converge in the same direction, and the few lines which are pointing in other directions can be filtered out. However, for surgical tools, there are very few lines and jointed tools will have an equal number of lines pointing in different directions, so it will be difficult to autonomously find the lines only from the last segment of the tool. Therefore, this method is not effective for surgical tools.

Neural Networks for Natural Scenes Currently, there are a few neural networks which have been trained to find the vanishing point in images with natural scenes (roads, buildings, etc.). The results from both these studies [6] [7] show that the neural networks were more effective than traditional methods of identifying the vanishing point in natural scenes. To the knowledge of the author, there are no neural networks that have been trained to identify the vanishing point of surgical tools, but this is a promising approach as a pre-existing model can be trained on surgical images.

1.3 Contributions

Training a neural network to identify the vanishing point of a surgical tool is a novel approach that can greatly increase the efficiency of the process. While some mathematical approaches to identify the vanishing point exist, they are more laborious and cannot be easily generalized to images with multiple, jointed tools. Although there are neural networks trained to find the vanishing point in natural scenes, these are unlikely to be effective in predicting the vanishing point in surgical images as they are two very different scenarios. This has been confirmed using the NeurVPS Conic Convolution Neural Network [7] which was trained on the Tmm17 dataset of natural scene images. When this pre-trained model was tested on surgical tool images, it performed very poorly (see Section 3.1). The performance improved after being trained on surgical tool images (see Section 3.2), showing that this is a promising method.

2 Methods

2.1 Dataset

There are many datasets available for identifying the surgical tool in an image, but there do not seem to be any datasets for identifying the vanishing point. Therefore, a Python program was written to create a dataset of 10,000 images with randomly generated surgical tool images and ground truth masks with the corresponding vanishing point. The dataset was created using the ImageDraw module from Python's 'pillow' library. The content from endoscopic cameras is limited to a circular area since the image sensor is usually larger than the image circle of the endoscope [8], so the tools in the dataset are confined within a circle.

Three versions of the dataset were created: unjointed tools without occlusion, unjointed tools with occlusion, and jointed tools without occlusion. The dataset with occlusion has black blobs covering parts of the tool to simulate the occlusion caused by tissues and blood in real life. Both datasets were split into 96% for training, 4% for testing.

2.1.1 Creating the Tool Images

The point from which the tool appears is randomly selected along the bottom half of the image circle, as this is generally the case in real life. The second point is chosen randomly to the right of the first point within a range of $\pi/9$ radians to $\pi/5$ radians. This forms the bottom edge of the first quadrilateral. To create the rest of the tool, two points are randomly chosen to create a top edge. Two methods for generating the top edge of the tool are discussed below.

This process is repeated with the top edge of the first quadrilateral being the bottom edge of the second quadrilateral and so on to create two or three quadrilaterals to form a jointed tool. Approximately half the tools have one joint and the remaining tools have two joints. Finally, a small semi-circle is drawn at the tip of the last quadrilateral to make the tool shape look realistic. To avoid any biases, all parameters are chosen randomly within given ranges. Note that the coordinates are chosen in an anti-clockwise order with the first coordinate being at the bottom left (see Figure 4).

The code that was used to create the datasets can be accessed here: `https://github.com/bhavaniv1101/vpd_data`

Method 1: Generating random points within a triangle After generating the bottom edge, a random point is chosen along the perpendicular bisector of the bottom edge. This forms a triangle with the first two vertices. This triangle is divided into two identical triangles by the perpendicular bisector. In order to get the two upper vertices of the quadrilateral, a random point is chosen from each of the two triangles. However, many of the images generated by this method were not satisfactory. The two upper vertices were often too close together, making the quadrilateral look like a triangle. In some other cases, the two points formed a line with one of the base vertices, again making it look like a triangle instead of a quadrilateral. This could be because the points are not being generated uniformly within the triangle. This method was not used as it did not give the expected results.

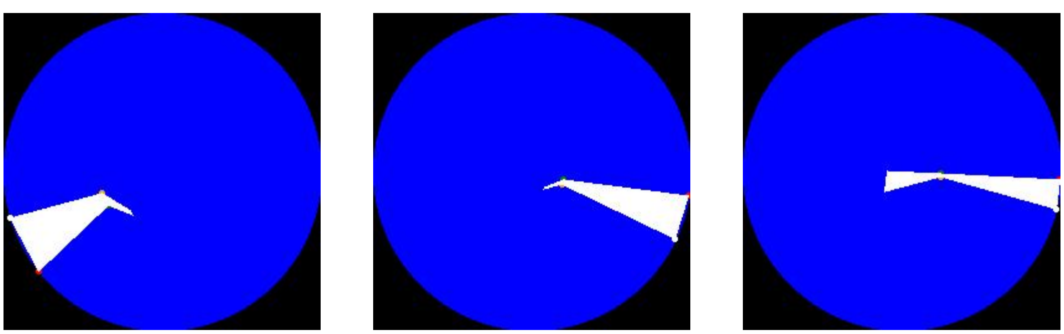


Figure 1: (left) The second quadrilateral is flattened into a line; (middle) The second quadrilateral is extremely small, making the tool look triangular; (right) The upper edge of the second triangle is larger than the lower edge, creating an unrealistic shape.

Method 2: Rotating a line about the perpendicular bisector A random point is chosen along the perpendicular bisector (within the range of 45.0 to 85.0 pixel units) of the quadrilateral's bottom edge. A random length between 40% and 60% of the base length is chosen for the length of the upper edge. A line of this length is drawn parallel to the base, with the point on the perpendicular bisector as its center, forming a trapezium. This line is then rotated by a random angle between $-\pi/5$ and $\pi/5$. This forms the upper edge of the quadrilateral. This method gave better tool images because the length of the upper edge and size of the quadrilateral can be controlled more easily, and the randomization is more reliable, preventing the cases which occurred with the previous method. This method can be easily generalised for any number of quadrilaterals using the top edge of the first quadrilateral as the base for the second one. This is the method that was finally used to generate a dataset of 10,000 images.

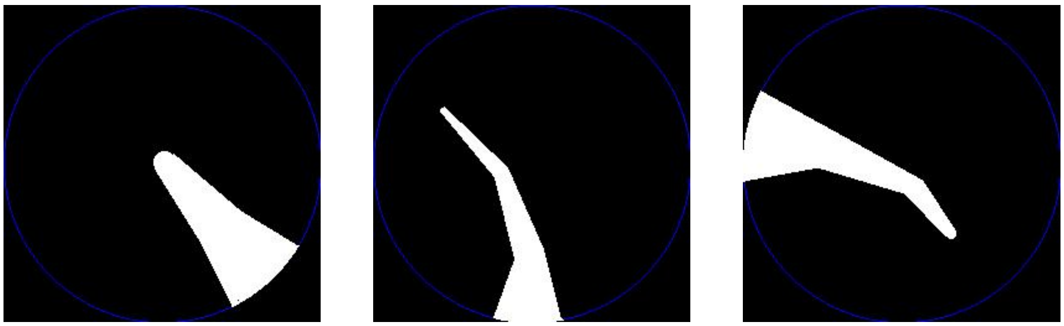


Figure 2: (left) Tool with a single joint; (middle and right) Tools with two joints where each quadrilateral is smaller than the previous one.

2.1.2 Creating Occluded Images

In real life, the tool image is often occluded due to tissues covering the tool. The occluded dataset was generated by creating blobs on the image to cover parts of the tool. The code for creating the blobs was generated based on code found on Stack Overflow [9]. The 'seedval' and 'threshold' parameters were randomized within a range to vary the amount of occlusion created by the blobs. Without this, the same blobs were being created for each image, so randomizing it made it more realistic.

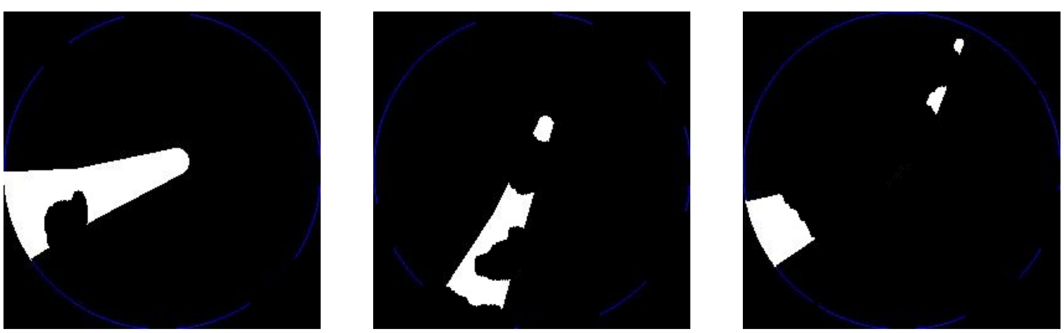


Figure 3: (left) Tool with some occlusion; (middle) Tool with more occlusion; (right) Tool with lot of occlusion.

2.1.3 Creating the ground truth masks

The approach that was originally chosen for the ground truth masks was to create images with a black background and a small white circle at the tool's vanishing point. However, after trying to run the neural network on this dataset, it was found that the neural network expected the coordinates of the vanishing point with respect to the image center, not an image with the vanishing point. Therefore, a '.txt' file with the coordinates was given for each image. A third value was also required in the '.txt' file: the focal length. However, this value can only be found based on the camera parameters, and this information is not available, so it was set to a common value of 1.0. The figure below shows how the coordinates of the vanishing point are calculated. The vanishing point is the point of intersection of the lines forming the two sides of the tool's final quadrilateral.

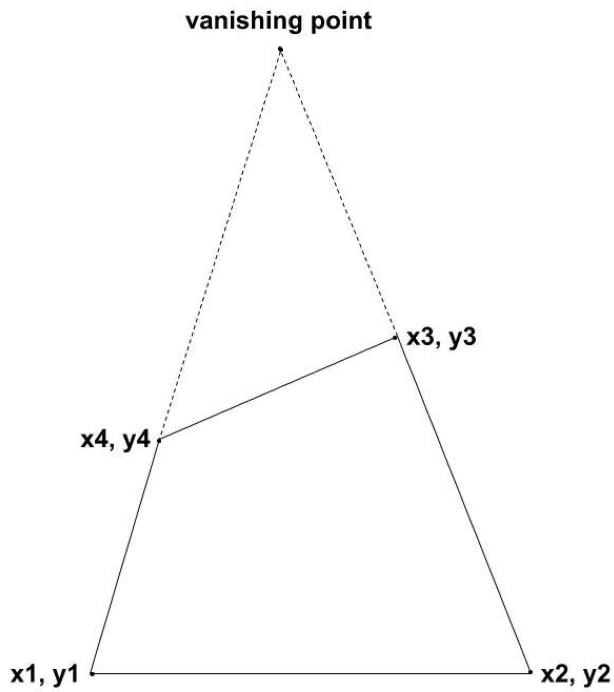


Figure 4: The points (x1, y1) and (x4, y4) form the line on the left and the point (x2, y2) and (x3, y3) form the line on the right.

To find the point of intersection of the two lines, we must first find the equations of the lines by calculating their respective slopes and y intercepts.

$$
\begin{aligned}
m_l &= \text{slope of the line connecting } (x_1, y_1) \text{ and } (x_4, y_4) \\
m_r &= \text{slope of the line connecting } (x_2, y_2) \text{ and } (x_3, y_3) \\
c_l &= \text{y intercept of the line connecting } (x_1, y_1) \text{ and } (x_4, y_4) \\
c_r &= \text{y intercept of the line connecting } (x_2, y_2) \text{ and } (x_3, y_3)
\end{aligned}
$$

$$
\begin{aligned}
m_l &= \frac{y_4 - y_1}{x_4 - x_1} & \qquad m_r &= \frac{y_3 - y_2}{x_3 - x_2} \\
y_1 &= m_l \cdot x_1 + c_l & y_2 &= m_r \cdot x_2 + c_r \\
c_l &= y_1 - m_l \cdot x_1 & c_r &= y_2 - m_r \cdot x_2
\end{aligned}
$$

At the point where the two lines intersect, we know that $y_1 = y_2$. We can use this to solve for the coordinates of the point of intersection, which is the vanishing point (x_v, y_v):

$$
\begin{aligned}
m_l \cdot x_v + c_l &= m_r \cdot x_v + c_r \\
\therefore x_v &= \frac{c_l - c_r}{m_r - m_l} \\
y_v &= m_l \cdot x_v + c_l
\end{aligned}
$$

To find the coordinates of the vanishing point with respect to the center, the coordinates of the image center were subtracted from the coordinates of the vanishing point (x_v, y_v).

2.2 Training the Neural Network

The neural network used in this paper is an "end-to-end trainable deep network" which uses geometry-inspired convolutional operators to detect the vanishing points. This uses a novel approach involving conic convolution to extract features like structural lines. This model was chosen because it is expected to be more accurate as it has been created specifically for identifying vanishing points, although in a different context of natural scenes.

2.2.1 Using Conic Convolution

This model [7] is based on a convolutional neural network. The conic convolution operators use geometric priors (symmetry and scale separation) of vanishing points so the model does not rely on line detectors. This explicitly enforces the extraction of features such as structural lines while using the same number of parameters as the regular 2D convolution. For the network to learn line features related to the vanishing point, convolutions are applied in the space where related lines can be determined locally. The conic space for each pixel is a rotated local coordinate system where the x-axis points from the pixel to the vanishing point. In this space, related lines can be identified locally by checking whether its orientation is horizontal. Conic convolution applies the regular convolution in this conic space. This helps the model effectively classify whether a candidate point is a valid vanishing point.

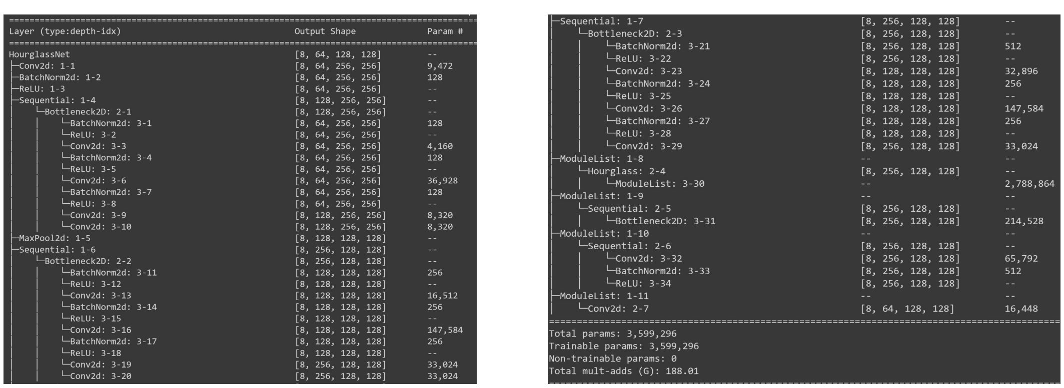
2.2.2 Changes Made to the Code

The code used for the neural network requires a CUDA enabled GPU, which was not available on the local device. Therefore, the code was transferred from the PyCharm IDE on a Mac OS laptop to Google Colaboratory, where a T4 GPU was available. The original code uses two GPUs, but only one GPU was used for this paper due to hardware limitations.

The source code for the neural network was written to accept one of three specific datasets: Wireframe dataset, ScanNet dataset, or Tmm17 dataset. The 'datasets.py' file contained three classes which were written specifically to process each of these datasets. Since the dataset of surgical tools has a different format and structure, a new class was written to process the surgical tool data and change it from JPEG images to the tensor format required for the neural network.

2.2.3 Neural Network Structure

The configurations used for the neural network are similar to those for the Tmm17 model. This is the model that was trained to find the vanishing point in natural scenes in hte NeurVPS paper [7].



Layer (type:depth-idx)	Output Shape	Param #
HourglassNet	[8, 64, 128, 128]	--
├─Conv2d: 1-1	[8, 64, 256, 256]	9,472
├─BatchNorm2d: 1-2	[8, 64, 256, 256]	128
├─ReLU: 1-3	[8, 64, 256, 256]	--
├─Sequential: 1-4	[8, 128, 256, 256]	--
└─Bottleneck2D: 2-1	[8, 128, 256, 256]	--
└─BatchNorm2d: 3-1	[8, 64, 256, 256]	128
└─ReLU: 3-2	[8, 64, 256, 256]	--
└─Conv2d: 3-3	[8, 64, 256, 256]	4,160
└─BatchNorm2d: 3-4	[8, 64, 256, 256]	128
└─ReLU: 3-5	[8, 64, 256, 256]	--
└─Conv2d: 3-6	[8, 64, 256, 256]	36,928
└─BatchNorm2d: 3-7	[8, 64, 256, 256]	128
└─ReLU: 3-8	[8, 64, 256, 256]	--
└─Conv2d: 3-9	[8, 128, 256, 256]	8,320
└─Conv2d: 3-10	[8, 128, 256, 256]	8,320
├─MaxPool2d: 1-5	[8, 128, 128, 128]	--
├─Sequential: 1-6	[8, 256, 128, 128]	--
└─Bottleneck2D: 2-2	[8, 256, 128, 128]	--
└─BatchNorm2d: 3-11	[8, 128, 128, 128]	256
└─ReLU: 3-12	[8, 128, 128, 128]	--
└─Conv2d: 3-13	[8, 128, 128, 128]	16,512
└─BatchNorm2d: 3-14	[8, 128, 128, 128]	256
└─ReLU: 3-15	[8, 128, 128, 128]	--
└─Conv2d: 3-16	[8, 128, 128, 128]	147,584
└─BatchNorm2d: 3-17	[8, 128, 128, 128]	256
└─ReLU: 3-18	[8, 128, 128, 128]	--
└─Conv2d: 3-19	[8, 256, 128, 128]	33,024
└─Conv2d: 3-20	[8, 256, 128, 128]	33,024
├─Sequential: 1-7	[8, 256, 128, 128]	--
└─Bottleneck2D: 2-3	[8, 256, 128, 128]	--
└─BatchNorm2d: 3-21	[8, 256, 128, 128]	512
└─ReLU: 3-22	[8, 256, 128, 128]	--
└─Conv2d: 3-23	[8, 128, 128, 128]	32,896
└─BatchNorm2d: 3-24	[8, 128, 128, 128]	256
└─ReLU: 3-25	[8, 128, 128, 128]	--
└─Conv2d: 3-26	[8, 128, 128, 128]	147,584
└─BatchNorm2d: 3-27	[8, 128, 128, 128]	256
└─ReLU: 3-28	[8, 128, 128, 128]	--
└─Conv2d: 3-29	[8, 256, 128, 128]	33,024
├─ModuleList: 1-8	--	--
└─Hourglass: 2-4	[8, 256, 128, 128]	--
└─ModuleList: 3-30	--	2,788,864
├─ModuleList: 1-9	--	--
└─Sequential: 2-5	[8, 256, 128, 128]	--
└─Bottleneck2D: 3-31	[8, 256, 128, 128]	214,528
├─ModuleList: 1-10	--	--
└─Sequential: 2-6	[8, 256, 128, 128]	--
└─Conv2d: 3-32	[8, 256, 128, 128]	65,792
└─BatchNorm2d: 3-33	[8, 256, 128, 128]	512
└─ReLU: 3-34	[8, 256, 128, 128]	--
├─ModuleList: 1-11	--	--
└─Conv2d: 2-7	[8, 64, 128, 128]	16,448

Total params: 3,599,296
Trainable params: 3,599,296
Non-trainable params: 0
Total mult-adds (G): 188.01

Figure 5: Summary of neural network configuration

2.3 Loss Function

The loss function used to determine the performance of the neural network is the binary cross entropy loss function. It tracks incorrect labeling and penalizes the model if deviations in probability occur [10]. If the log loss value is low, it means that the model's accuracy is high.

$$\text{Binary cross entropy} = -1 \cdot \log(\text{likelihood}).$$

During evaluation, the angle error is calculated. This is the difference between the angle to the ground truth vanishing point and the angle to the predicted vanishing point, with respect to a common reference point. The angle error graph shows the cumulative distribution function of the angle error. The percentile rank on the y axis shows the percentage of images that have an angle error less than or equal to the corresponding angle error on the x axis. This means that the angle error graph of a well-performing model would have a steep slope at first and flatten out quickly, showing that a high proportion of predictions have a small error.

3 Experiments and Results

The neural network was trained and tested with three different datasets. For each experiment, the neural network was trained for 2 epochs on a dataset of 10,000 images, which was split into 96% for training and 4% for testing.

3.1 Testing the Pre-trained Model on Surgical Tool Images

Before training the model on the surgical tool images, the pre-trained Tmm17 neural network was evaluated on a dataset of unjointed surgical tools. The pre-trained model was originally trained on natural scene images.

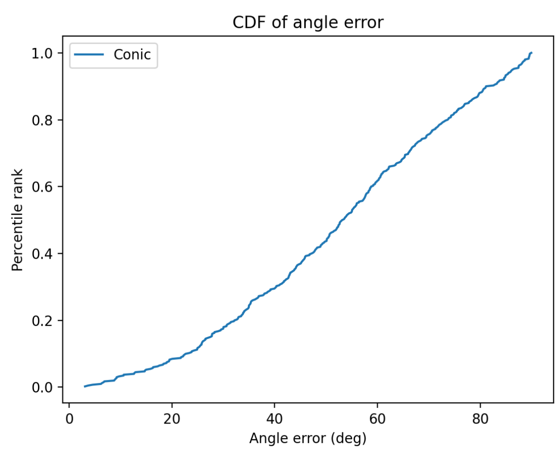


Figure 6: Angle Error of pre-trained model before training on the surgical tool dataset.

As expected, the accuracy is low as it has not been trained on surgical images. However, there are some accurate predictions, showing that this model has potential to perform well after being trained on surgical tool images.

3.2 Training on Unjointed Tool Images

The neural network was then trained and tested on the dataset of unjointed surgical tool images. The accuracy is significantly better than that of the pre-trained model, as shown in Figure 7. The accuracy could still be improved as the angle error goes up to around 21° before flattening out. However, this is significantly better compared to the pre-trained model which had angle errors of up to 80°, so the model has been learning.

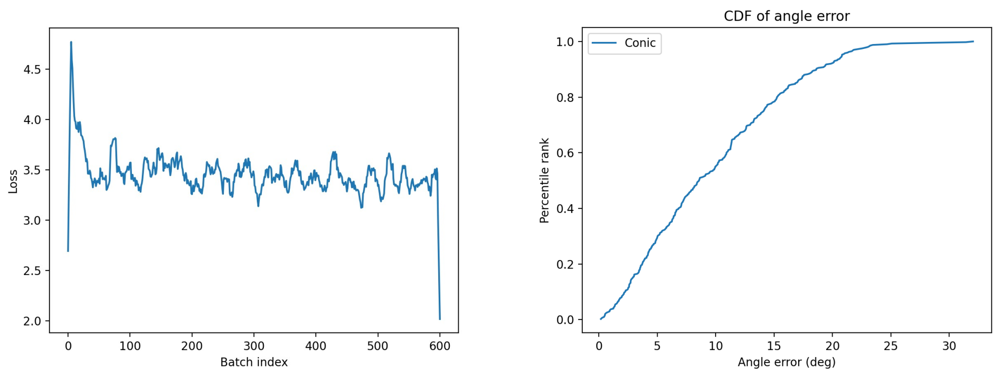


Figure 7: (left) Loss vs. Epoch and (right) Angle Error for unjointed surgical tools

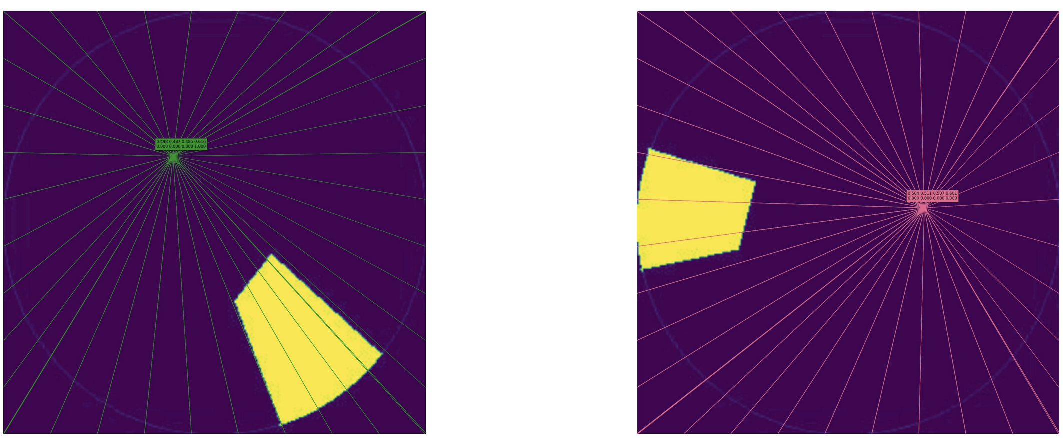


Figure 8: Unjointed tools with the predicted vanishing points

This model was then cross-tested on occluded unjointed tool images. The results were slightly better for the unoccluded dataset, but overall, the results were quite similar. This shows that occlusion does not have much effect on the model's performance.

3.3 Training on Occluded Unjointed Tool Images

The neural network was then trained on occluded unjointed tool images. As expected, the model performed slightly better on unoccluded images than on occluded images. The angle error graph for unoccluded images flatted out around 22°, compared to 21° for the unoccluded dataset. Similarly to the previous section, the accuracy of the model after being trained on occluded images is very close to the accuracy of the model trained on unoccluded images. This once again suggests that the model is not affected much by occlusion.

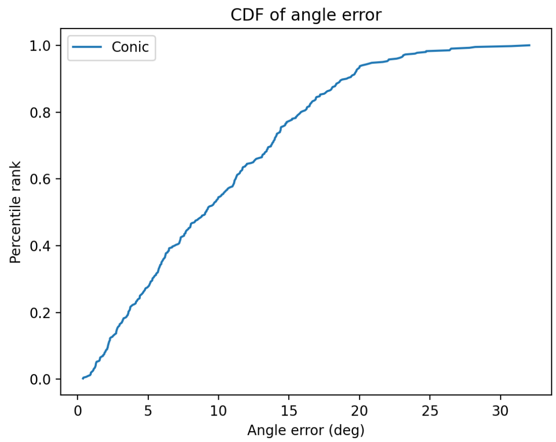


Figure 9: Angle Error of the model on the occluded unjointed tool dataset.

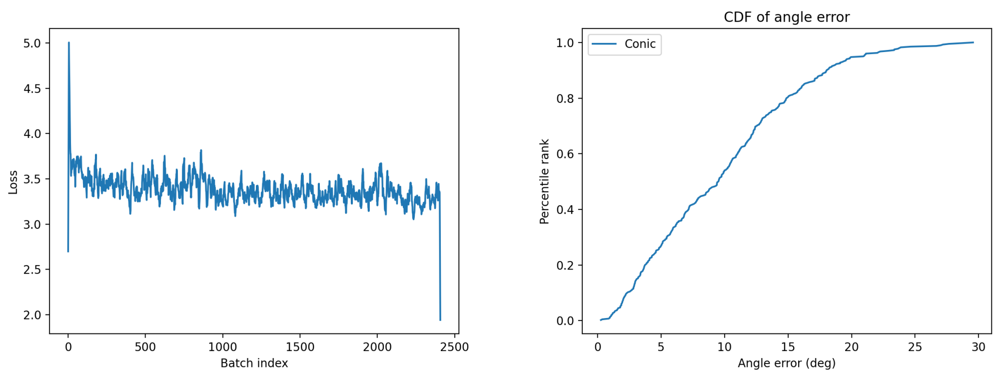


Figure 10: (left) Loss vs. Epoch and (right) Angle Error for occluded unjointed surgical tools

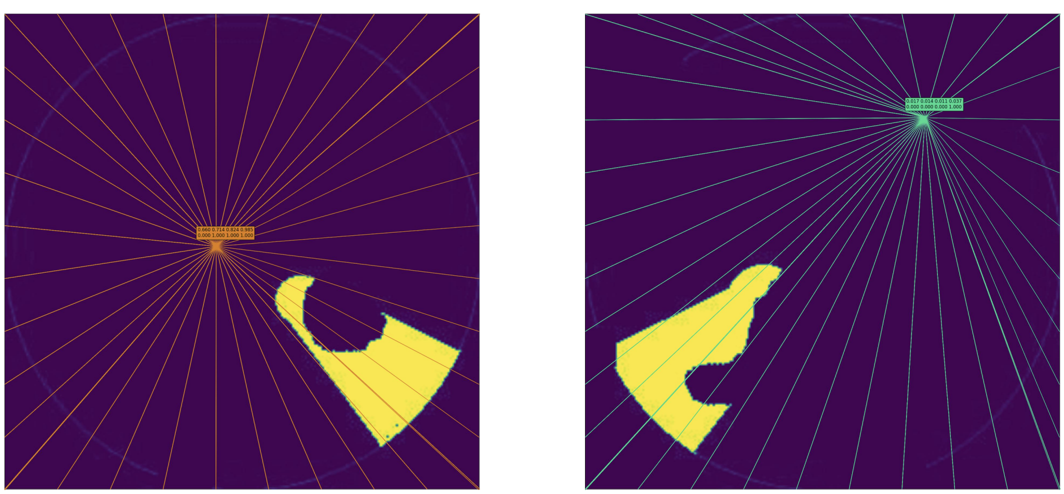


Figure 11: Occluded unjointed tools with the predicted vanishing points

3.4 Training on Jointed Tool Images

Finally, the model was trained on a dataset of jointed tool images, where approximately half the tools have one joint and the remaining tools have two joints. The graphs below show that the performance was quite poor on jointed tools as compared to unjointed tools because the loss is higher at around 4.5 for jointed tools while it was around 3.5 for unjointed tools. Some approaches that were tried to improve the accuracy include inverting the image colors, changing the learning rate and hyperparameters, and removing the image circle outline. None of these approaches helped much with the model's performance. However, there are a few instances where the prediction is close to the ground truth, suggesting that the model could be improved with more training.

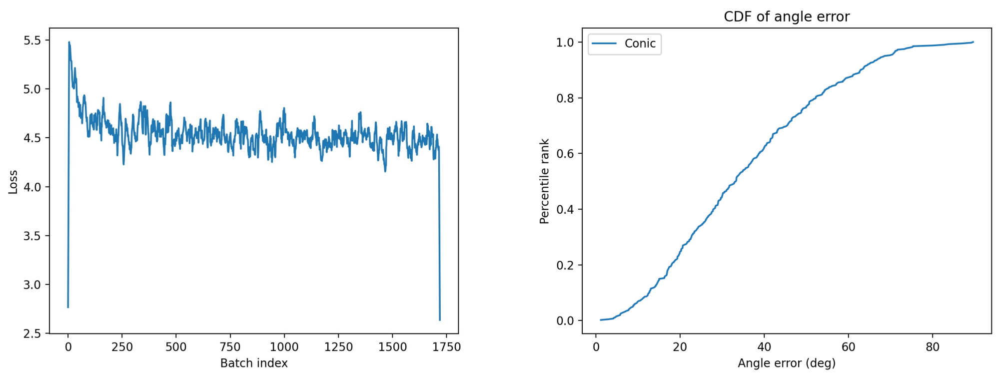


Figure 12: (left) Loss vs. Epoch and (right) Angle Error for jointed surgical tools

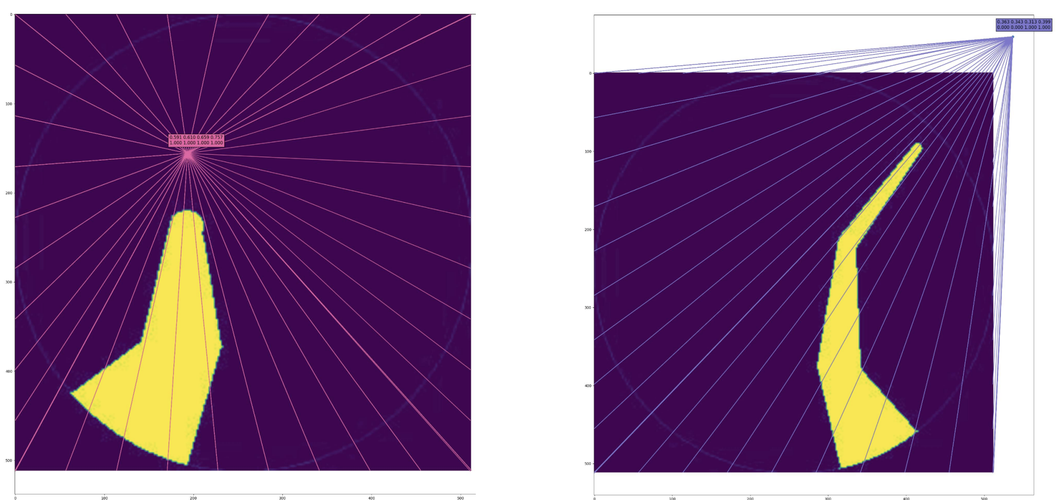


Figure 13: Jointed tools with the predicted vanishing points

4 Conclusion

In this paper, a neural network was trained to identify the vanishing point of surgical tools in three different types of datasets: unjointed tools without occlusion, unjointed tools with occlusion, and jointed tools without occlusion. A pre-trained neural network was tested on surgical tool images and the performance was quite low, but after training the model on the unjointed dataset, the results improved significantly. This shows that the model has potential to improve further with more training. The model did not perform as well with jointed tool images but this could potentially be improved by training it on more images and for more epochs.

One of the key takeaways is that when the model was cross-tested on occluded images, the angle error for the occluded images was quite close to that of the unoccluded images. Also, the result after training the model on occluded images was similar to the result of the model trained on unoccluded images. This suggests that the model's performance is not affected much by occlusion. Therefore, neural networks have a significant advantage compared to traditional, mathematical approaches as neural networks can adapt more easily to variations in the images, such as occlusion. This will be very useful in practical applications as real-life images are often messy and not very clear.

These results are summarised in the table below. The 'Mean Training Loss' represents the binary entropy loss while training. The 'Angle Error Flattening Point' is the point where the angle error graph flattens out, showing that few predictions had a higher error than this.

Model + Dataset	**Mean Training Loss**	**Angle Error Flattening Point (deg)**
Pre-trained Model Tested on Unjointed Tools	N/A	80°
Training on Unjointed Tools	3.4	21°
Cross-testing on Occluded Tools	N/A	24°
Training on Occluded Unjointed Tools	3.5	22°
Training on Jointed Tools	4.5	75°

4.1 Future Work

Datasets with multiple tools A dataset could be created with multiple surgical tools in each image, with the amount of overlap between tools varying randomly. The code would need to be modified to give multiple vanishing point coordinates as the output, based on the number of tools in the image. A layer of complexity could be added by having tools which overlap each other, so the model would have to learn to differentiate the tools.

Datasets with continuous motion A dataset could be created where the images simulate the motion of the tool in real-life surgeries. In each consecutive image, the tool would be displaced by a few pixels in a given direction. When the series of images is viewed side-by-side, it would appear that the tool follows a random path within the image circle. This can be repeated for different tools, creating around 20 images for each tool. To go one step further, the tool could also be rotated when the direction changes. The neural network could be trained on this dataset and the results could be compared with those from the other datasets.

Comparing with traditional methods The accuracy of the neural network in detecting the vanishing point could be compared with the accuracy and efficiency of the traditional methods mentioned in Section 1.2. This could be done by comparing the percentage of predictions which are accurate for each approach and by comparing the time taken by the program to predict the coordinates of the vanishing point. These methods would need to be implemented specifically for surgical tools as they currently exist mainly for natural scenes. This should be done for all the different types of datasets that were generated. This will help confirm that using a neural network is the best approach for identifying the vanishing point of surgical tools.

References

[1] A. M. Okamura, "Haptic feedback in robot-assisted minimally invasive surgery," *Current opinion in urology*, vol. 19, no. 1, p. 102, 2009.

[2] C. R. Wagner, R. D. Howe, and N. Stylopoulos, "The role of force feedback in surgery: analysis of blunt dissection," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, International Symposium on*, pp. 73–73, IEEE Computer Society, 2002.

[3] D.-K. Ko, K.-W. Lee, D. H. Lee, and S.-C. Lim, "Vision-based interaction force estimation for robot grip motion without tactile/force sensor," *Expert Systems with Applications*, vol. 211, p. 118441, 2023.

[4] K. Huang, D. Chitrakar, W. Jiang, I. Yung, and Y.-H. Su, "Surgical tool segmentation with pose-informed morphological polar transform of endoscopic images," *Journal of Medical Robotics Research*, vol. 7, no. 02n03, p. 2241003, 2022.

[5] R. Kedia, "Vanishing point: Image processing - opencv, python amp; c++ by: Rahul kedia."

[6] A. Borji, "Vanishing point detection with convolutional neural networks," *arXiv preprint arXiv:1609.00967*, 2016.

[7] Y. Zhou, H. Qi, J. Huang, and Y. Ma, "Neurvps: Neural vanishing point scanning via conic convolution," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[8] K. Huang, D. Chitrakar, W. Jiang, and Y.-H. Su, "Enhanced u-net tool segmentation using hybrid coordinate representations of endoscopic images," in *2021 International Symposium on Medical Robotics (ISMR)*, pp. 1–7, IEEE, 2021.

[9] F. Weinhaus and Rifly, "Is it possible to create a random shape on an image in python?," 2022.

[10] A. Roberts, "Binary cross entropy: Where to use log loss in model monitoring," 2023.