Vortex Detection

Sairama Sashank Kadiyala
IMT2018064

Ellanti Bharath Sai
IMT2018022

***Abstract*—Vortices are structures formed due to the presence of rotation and helicity in a three-dimensional hydrodynamic helically rotating turbulence. One of the phenomena believed to be responsible for the creation of large-scale magnetic structures in astrophysical systems is the inverse cascade of magnetic helicity in three-dimensional (3D-MHD) turbulence. The Region between two Vortices rotating in opposite directions is termed a Reconnection region. The existence of these regions signifies that there is a high possibility of the adjacent vortices converging into one vortex. Our project's objective is to detect the existence of these regions using a database as an input with hierarchical data.**

I. Introduction

A vortex is a region where the flow circulates around a straight or curved axis line. In their parent protoplanetary discs (A protoplanetary disk is a rotating circumstellar disk of dense gas surrounding a young newly formed star), high-mass planets leave behind distinctive features like a gap, spiral waves, vortices, and eccentricities. The Vortices are comprised of Magnetic flux and these rotate either clockwise or anti-clockwise. In order to grow, two adjacent vortices tend to converge into a single larger vortex. This happens when two adjacent vortices are rotating in opposite directions. The Region between these two Vortices(rotating in opposite directions) is called the Reconnection region.
Our aim is to find out these reconnection regions(if present), Formed by two oppositely rotating vortices in a three-dimensional hydrodynamic helically rotating turbulence, in a database, as input.

A. Approach

- In our Dataset, we had files in hdf5 format and we used a tool called VisIt to visualize its contents and generate a vector image in 3D and 2D formats.
- From the 2D format Image we detect the vortices using the HOUGH circle detection method and then detect the rotation of these vortices using a Convolutional Neural Network(CNN) Model.

II. Dataset

The Dataset contained files in "hdf5" file format. Each file represents one snapshot of the rotating magnetic flux. There are 40 different snapshots at different timeframes i.e. 40 different files in each folder.
Each folder represents a single combination of helicity and rotation speed.
For Example, as shown in Fig. 1, "maxrotminhel" is a flux with maximum rotation and minimum helicity.

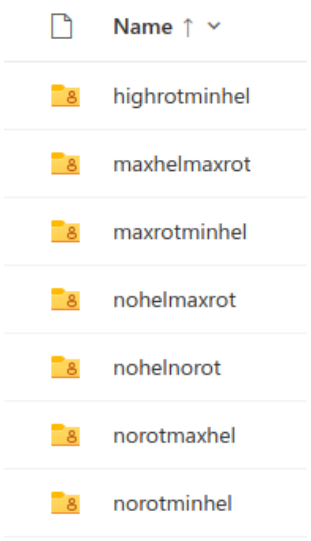


Fig. 1. Dataset

Each file(one timeframe) consists of 3 datasets - Vx, Vy, and Vz. Each of these datasets has magnitude values for 128x128x128 points in that particular direction(Vx in the x-direction, Vy in the y-direction, Vz in the z-direction) as shown in the figure.

```
HDF5 "velocity.10000.h5" {
GROUP "/" {
   GROUP "PS" {
      DATASET "vx" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 128, 128, 128 ) / ( 128, 128, 128 ) }
         DATA {
         (0,0,0): 0.875483, 0.884132, 0.881972, 0.86405, 0.832737, 0.798045,
         (0,0,6): 0.770155, 0.75291, 0.744397, 0.740588, 0.736937, 0.729145,
         (0,0,12): 0.715636, 0.699469, 0.686362, 0.680737, 0.683603,
         (0,0,17): 0.692668, 0.704034, 0.715262, 0.727248, 0.742888,
         (0,0,22): 0.764648, 0.793294, 0.827501, 0.86454, 0.902015, 0.938701,
         (0,0,28): 0.973744, 1.00597, 1.03395, 1.05599, 1.07076, 1.07897,
         (0,0,34): 1.08392, 1.08944, 1.09757, 1.10863, 1.1222, 1.13742,
```

We used a tool called "VisIt" to visit the contents of the dataset.

III. Installation and Visualization

A. Installation

The data in the hdf5 files need a special tool to visualize, so we used VisIt to visualize them because Python had a module for VisIt, and we could visualize the files and alter the required parameters using Python code.

1. Installing VisIt on Linux is done using the visit-install script. Make sure that the visit-install script is executable by entering the following command:
- chmod +x visit-install

2. To install an x86_64 version of visit 3.2.0, first download the tar file go to the official website (https://visit-dav.github.io/visit-website/), and download the latest version.

3. After downloading go to the directory where the tar file is present and run the following command:
- visit-install 3.2.0 linux-x86_64 /usr/local/visit
This command will install the 3.2.0 version of VisIt into the /usr/local/visit directory.

B. Visualization

Manual Visualization of one 3D timeframe(one .h5 file) using the tool directly without any python code or modules, is shown in the Figure below.

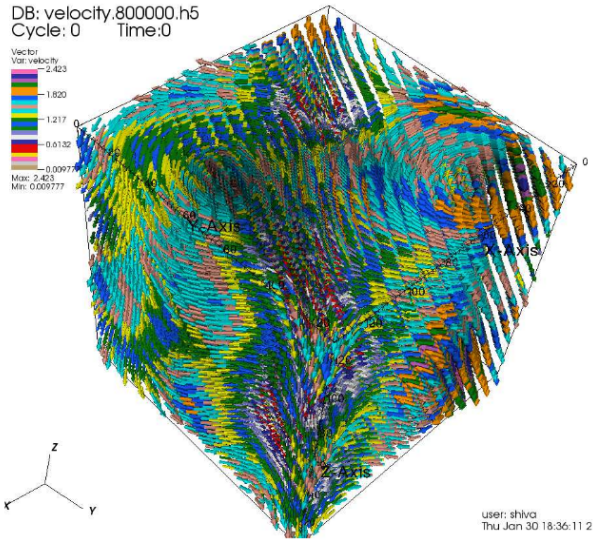


Fig. 2. 3D image

Our approach was to use a 2D projection of this figure to first detect vortices. VisIT supports python scripting where we can import different VisIt functions into our python code.

In our data file, we have information about vector magnitude in the x, y, and z directions at all points in the space. We need to find the magnitude and direction of each vector to generate an image. VisIt provides a function called DefineVectorExpression() which helps us to create vectors with the variables that are present in the database.
The Vector plot uses glyphs to represent vector variables and shows the magnitude and direction of each vector in a vector field.

A vector plot frequently has too many vectors when displaying a sizable database. With too many vectors, the

```
DefineVectorExpression("v","{<PS/vx>,<PS/vy>,<PS/vz>}")
DeleteAllPlots()
AddPlot("Vector","v")
```

Fig. 3.

Vector plot is impossible to understand. VisIt offers tools to reduce the amount of vectors to a level that makes a visualisation appealing. This decrease can be achieved by setting nVectors variable.

```
va=VectorAttributes()
va.colorTableName="amino_shapely"
va.nVectors=40000
va.lineStem=0
SetPlotOptions(va)
```

Fig. 4.

Once we adjust all the parameters as required. The projection we get is shown in the figure below.

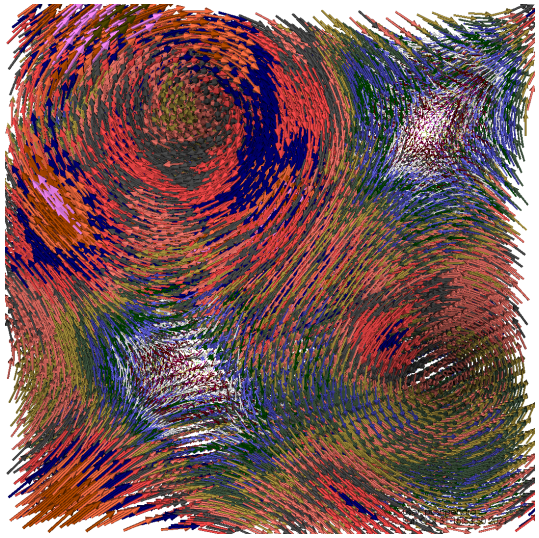


Fig. 5. Image with 4000 vectors

To visualize the direction of vectors clearly, we can decrease the number of vectors in the plot.

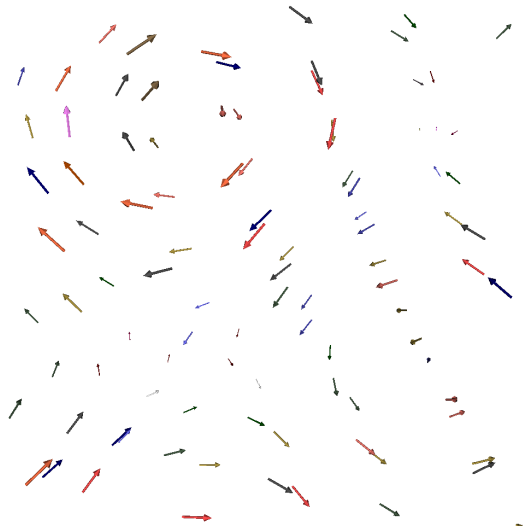


Fig. 6. Image with 100 vectors

IV. Detecting the Reconnection Region

A. HOUGH Circle Detection

For Detecting the reconnection region, we first detected the presence of circles(the vortices) present in the 2D projection image. For this, we used the HOUGH circle detection method.

HOUGH circle detection is a basic feature extraction technique for detecting circles. The detection of circles using this method mainly involves 2 stages.

- The first stage involves edge detection and finding the possible circle centers.
- The second stage finds the best radius for each candidate center.

First, we load the Image, grayscale, and Blur it to Reduce Noise.

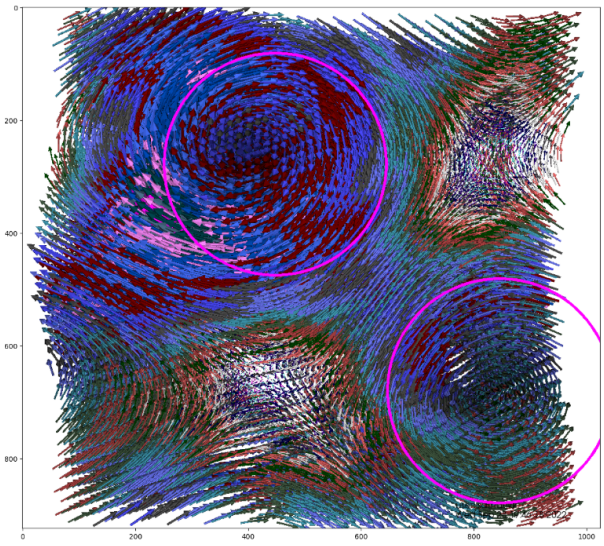
Then we use the circle detection algorithm. The Parameters

```
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
blur = cv.GaussianBlur(gray,(9,9),10)

ret,thresh = cv.threshold(blur,127,255,cv.THRESH_TRUNC)

rows = thresh.shape[0]
circles = cv.HoughCircles(thresh, cv.HOUGH_GRADIENT, 1, rows /4,
                          param1=50, param2=50, minRadius=150, maxRadius=200)
```

for the Hough function are:

- circles: A vector that stores sets of 3 values: Center coordinates (x,y) and radius r.
- thresh: Input image (grayscaled and blurred)
- HOUGH_GRADIENT: Define the detection method. Currently, this is the only one available in OpenCV.
- dp = 1: The inverse ratio of resolution.
- min_dist = gray.rows/4: Minimum distance between detected centers.
- param_1: Upper threshold for the internal edge detector.
- param_2: Threshold for center detection.
- min_radius: Minimum radius to be detected. If unknown, we put zero as the default value.
- max_radius: Maximum radius to be detected. If unknown, we put zero as the default value.

After adjusting the parameters as required,i.e., param1=50, param2=50, minRadius=150, maxRadius=200, we detect the exact position of the vortices(the center and radius).



After detecting the individual location of the circles, we cropped out the portion in between to detect the presence of a reconnection region.

B. CNN Model - Detecting the direction Vector

We are using a Convolution neural Network(CNN) to find the direction of vectors.

To train the model we first crop each vector from a sparse image and classify them into 4 classes left, right, up, and down. We can use findcontours method in OpenCV to crop each vector from an image.

Fig. 7.

1) Generating Training Data: We first try to open train data images using OpenCV and store the image features (image array) in variable X and store their corresponding class labels in variable Y. The class labels are 0 for ”left”, 1 for ”right”, 2 for ”up”, 3 for ”down”. We use pickle to save these training features on our disk for faster operations.

2) Training Model: In our CNN model we have 3 Convolutional Layers and 2 Hidden layers. CNN takes tensors of shape(image height, image width, color channels)

as input ignoring the batch size.

In order to build a tensor of outputs, the Conv2D layer creates a convolution kernel that is convolved with the layer input. ReLU is the activation function we are utilising. MaxPooling2d downsamples the input along its spatial dimensions (height and width) by taking the highest value for each input channel over an input window with a pool size-determined size. Each dimension of the window is moved one step at a time. Each Conv2D and MaxPooling2D layer outputs a 3D form tensor (height, width, channels). As you move further into the network, the proportions of width and height tend to get smaller.

The final output tensor from the convolutional base is sent into one or more Dense layers to perform classification, completing the model. Vectors, which are 1D, are the input for dense layers, and the current output is a 3D tensor. The 3D output will first be flattened (or unrolled) into 1D, and then one or more Dense layers will be added on top. You employ a final Dense layer with 4 outputs because our dataset has 4 output classes.

This model is similar to that of our seniors because the final requirement was same but we have tested it out with different parameters for a better accurate model for our data. We have run our model with batch size 32, no of epochs 40 and validation data is 5% of training data. We have tried with different combinations of above parameters. We got good results for above values.

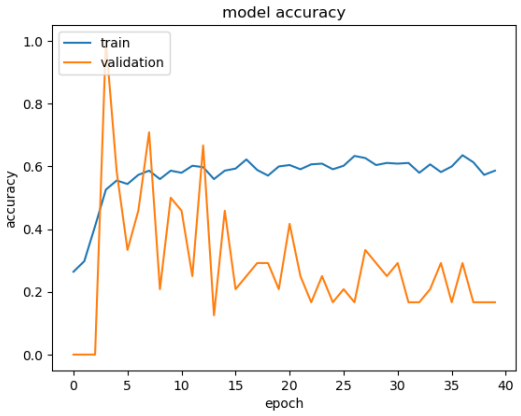


Fig. 8.

3) Testing model: We test our model on the region between the two vortices. If the two vortices are in opposite directions then the direction of vectors in the center region would allign on same axis. So we try to find the direction of each vector in the segmented region using our CNN model. Then we check if we have any vectors with opposite directions i.e we can't have both left and right direction vectors or up and down direction vectors. If there are no such vectors then we can say that region as reconnection region. We tried this approach on all the images we have and it worked on all images except for two images where direction of vectors was affected by CNN model. All the images we have has a reconnection region so we couldn't get the images that have no reconnection region to check our model performance.

V. Literature

To optimize the above approach we tried to use other techniques to detect circle effectively. We tried to use Clustering for the same. We read some existing literature on detecting cancer elements in a Mammogram image.

- They tried image segmentation techniques like clustering to segment the region of interest.
- They use pixel color as a feature in the clustering model.

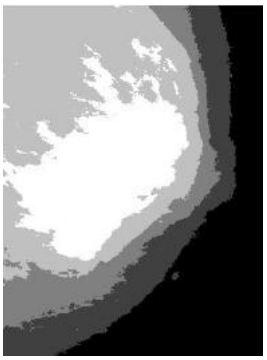


Fig. 9.

- In the above image if we try to cluster the image using Pixel colors then the similar color region would form a single cluster.
- This way they could segment the cancer region from the image.

In our case we can't use pixel color as a feature as doesn't have any significance in the image. we have colors to the vectors only to distinguish their magnitude. We need to get other features that have significance in detecting circles in our image. We have values regarding Vx, Vy, and Vz of a vector in the raw data file. Using these values we could get the magnitude and direction of a vector. We tried to use these magnitudes and directions as features to cluster regions in our image.

VI. Clustering

To optimize the above approach, we had the idea of using Clustering methods. This could have the possibility of leading to a better and more efficient approach if we could find the perfect possible parameters on which to base the clusters.
To find more efficient parameters we went on to view and understand the raw data.

A. Slicing

First, we started off by Slicing the 3D vortex into 2D sheets. We achieved this by keeping the x-coordinate of the 3D Vortex points constant and plotting their varying y and z coordinates at that specific 'x' value.
To visualize this using VisIT we put the 'Vx' magnitude to zero to eliminate all the vector directions that lead them above and below the plane of interest. We get each sheet as seen in the figure below

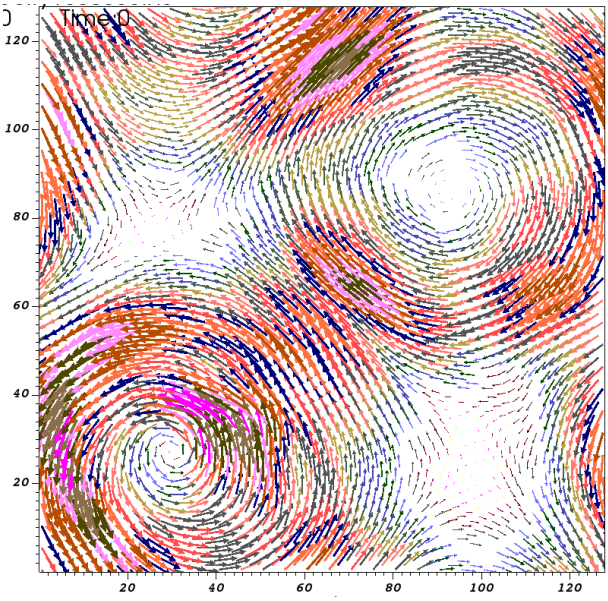


Fig. 10.

We tried to check if there are any behavior changes or slight dissimilarities between the slices. After using the HOUGH circle detection algorithm on each slice individually, we reached the conclusion that there were no significant differences between the slices.

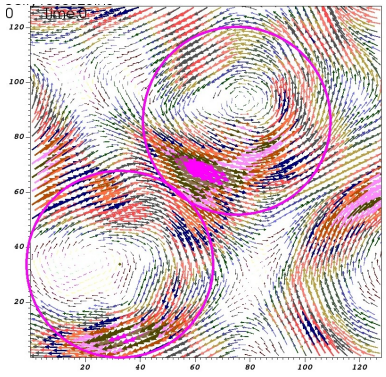


Fig. 11. Slice1

From the figures (Slice1, Slice2, Slice3), we could conclude that there was only a slight shift in the location of the centers

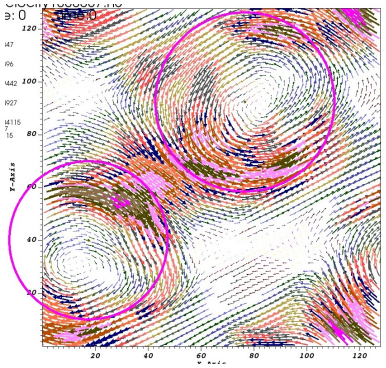


Fig. 12. Slice2

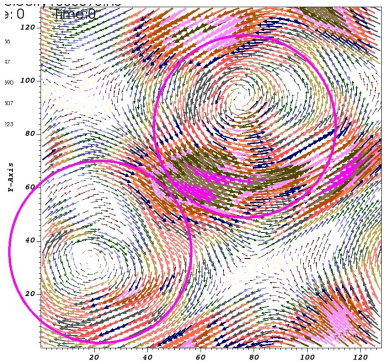


Fig. 13. Slice3

and a slight change in the radius of the vortex.

B. Magnitude

From, the raw data available, the parameters which can be used for clustering are Magnitude and Direction.
We moved forward with the idea that the vector magnitude in and around the vortex would be similar and a large cluster formed while using magnitude as a parameter would signify that there is a vortex in that region.
We tried clustering the points manually based on their magnitude by introducing Threshold values in between the maximum and minimum values of the magnitude of the vectors.
The output was very inconclusive. In some cases, we were able to distinguish a cluster as a circle but it had some extra area as part of it which was not a part of the vortex.
Figures 14 and 15 are two sliced planes of the same 3D vortex(the one used for slicing and displayed in Figure 10).

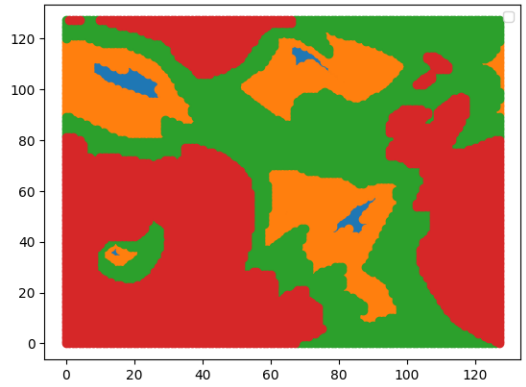


Fig. 14.

As you can see, the one vortex is identifiable in Figure 14 on the bottom left in red, but the same is not true in Figure 15.

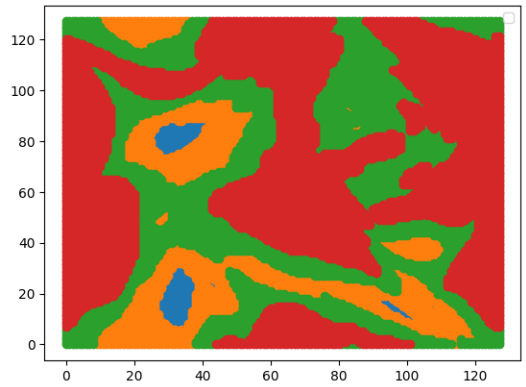


Fig. 15.

As you can see, we are unable to distinguish the vortex for other areas of the plot.
After testing out the behavior of magnitude on different slices of the same timeframe and also testing it out on different timeframe plots, we conclude that that magnitude is an independent feature and in no way follows a pattern when a vortex is formed even in different slices of one 3D timeframe. Therefore we went on to test the behavior of the second available parameter i.e. the direction of the vectors.

C. Direction

While using the magnitude as a parameter for clustering, we tried finding the vortices in the plot we have. When coming to direction, we tried focusing more on the initial problem statement we have i.e. finding the reconnection region in the plot. All the vectors present in the reconnection will be in the same direction because of the magnetic field generated by the opposite rotating vortices.
Initially, we manually clustered based on the angle it made with one of the axes. We got two more large clusters with other smaller clusters as seen in figure 16.

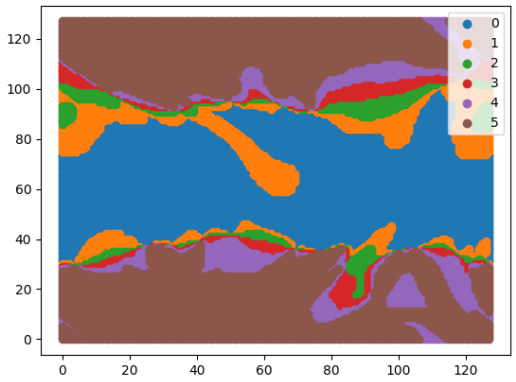


Fig. 16.

The reason for this large cluster is that we tried finding out the angle using the formula

$$cos^{-}1(dotproduct/magnitude)$$

This gives us an angle in the region of 0°-180°(all the trigonometric functions give us an output range of 180°). This put all the reflex-angled vectors in the same cluster as their mirror-imaged vectors.
We fixed that issue by taking the angle made by the vector with both axes and then adjusting the angle accordingly to 360°. Then we manually clustered based on the new angle of each vector.

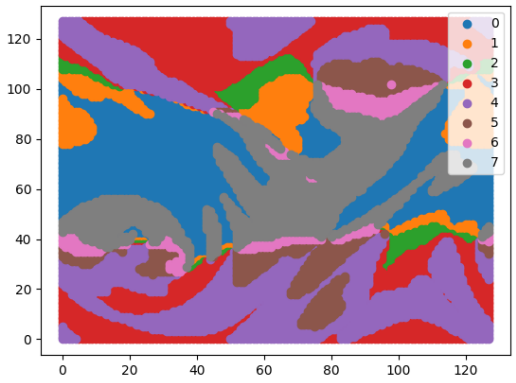


Fig. 17.

From the manually clustered image, we were not able to conclude anything effective as the clusters were not clear and there were many out-of-place points in each cluster. We were not able to figure out the exact location of the reconnection region. This was because small groups of vectors having the same direction are present in different locations of the same plot and all these groups are part of the same threshold and

denoted by the same color.

After working on the manual partition we have implemented k-means clustering on the dataset. Inferences from this approach are minimal. k-means is dividing the region between the vortices into different clusters which is not the desired result. We worked on optimizing the hyperparameters for better results but we couldn't get the reconnection region into a single cluster. We couldn't get conclusive evidence of large clusters being reconnection regions.

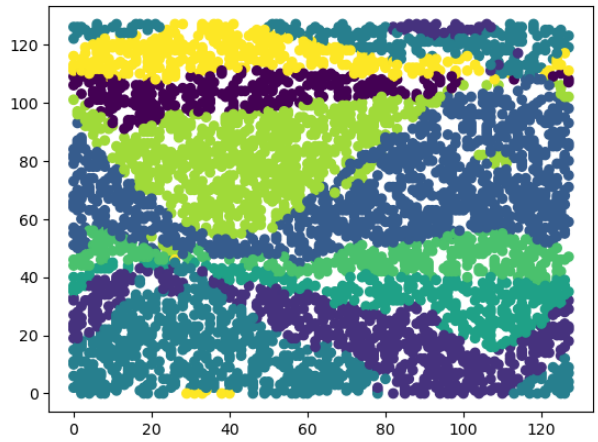


Fig. 18. Only Direction as feature

Instead of Having only direction as a clustering factor we then modified our algorithm to take data of both the magnitude and direction of the vector. There has not been any difference in the results by including magnitude.

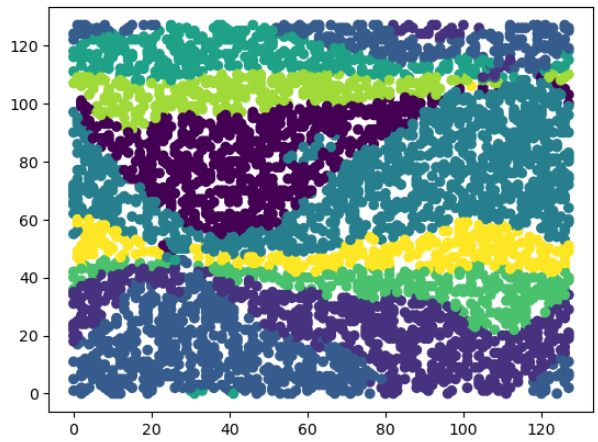


Fig. 19. Direction and Magnitude as features

VII. Main Dataset

Till now we have used only a small dataset where in each image we have either one or two vortices. Our aim is to work our algorithm on a bigger dataset where we have many reconnection regions in a single image. We tried to look at this large dataset so that we could get an idea, but the Python Visit module is not working on the server. We have tried many different ways to resolve this issue, but they didn't work. It was said that there was some problem in loading the Visit module on the server. So we continued our work on this small dataset only.

VIII. Conclusion

-We tried to explore all the features present in the data to train our clustering model. But due to the limited parameters available to us i.e. the magnitude and the direction of the vectors, we were not able to get a good clustering model that can detect the reconnection regions properly.
The initial approach of HOUGH circle detection + CNN model was more accurate when compared to clustering techniques.

IX. Acknowledgment

We would like to thank Professor Vinu.E.V for giving us the opportunity to work on the project and helping us whenever we were struck by giving us ideas to learn from. We are profoundly grateful for his expert guidance and continuous encouragement throughout to see that this project rights its target from its commencement to its completion. We thank Professor Shiva Kumar Malapaka for helping us out with the dataset.

X. Contributions

Bharath(IMT2018022):

- Visit Visualization using Python code
- CNN model
- Research for better methods
- Working to resolve main dataset issues.
- Clustering using K-means algorithm with magnitude and direction as parameters.

Sashank(IMT2018064):

- Visit Visualization using GUI(3D plots)
- HOUGH circle detection and model testing.
- Slicing and its observations.
- Finding contents and parameters in raw data files of the initial dataset.
- Manual clustering with magnitude and direction as parameters.

XI. References

- Documentation of Visit Visualization tool."https://visit-sphinx-github-user manual.readthedocs.io/en/develop/gui_manual/index.html "
- Hough Circle Transform. "https://docs.opencv.org/3.4/d4/d70/tutorial_hough_circle.html"
- "https://towardsdatascience.com/image-clustering-using-k-means-4a78478d2b83"
- Sample K-Means Clustering Method for Determining the Stage of Breast Cancer Malignancy Based on Cancer Size on Mammogram Image Basis.

- ”https://ieeexplore.ieee.org/abstract/document/5752535”
- ”https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/abs/10.1002/jemt.23908”
- ”https://docs.python.org/3/”
- ”https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939”