*Recognition of Handwriting Signature*

Prashant Pratik   
prash.pratik@gmail.com

*Abstract*—The K-Means Algorithm is a Clustering Algorithm which has been extended as a Function Recognition Algorithm in LearningApi which is a Machine Learning foundation of useful libraries written in .NET Core developed by Frankfurt University of Applied Sciences. Along with this algorithm, a Windows Form Application named MouseGestureRecognition has also been created which uses this algorithm. This application capture mouse pointer movements and collect coordinates. This set of coordinates is a learning set of mouse points as an array of two-dimensional vectors. After learning, this application also predicts whether the current mouse gesture is same as previous learned one or not. Also, this application has been extended by adding time as a third component in the vector making it three-dimensional. This paper is an analysis of the application with and without time approach.

Keywords—K-Means Algorithm, Clustering Algorithm, Recognition Algorithm, MouseGestureRecognition, LearningApi, Machine Learning, .NET Core.

# Introduction

The K-Means Clustering Algorithm is an unsupervised Machine Learning Algorithm. This algorithm basically categorizes a set of data into groups. This algorithm can also be extended as a Function Recognition Algorithm for recognition of handwriting signatures. To achieve this, a library named AnomDetect.KMeans in which K-Means Algorithm is already implemented and also extended as Function Recognition Algorithm and a Windows Form Application named MouseGestureRecognition which uses this algorithm has been created in LearningApi which is a Machine Learning foundation on top of .NET Core developed by Frankfurt University of Applied Sciences. This application capture mouse pointer movements and collect two-dimensional space coordinates of those mouse points. This set of coordinates is a learning set of mouse points as an array of two-dimensional vectors for the algorithm. After learning, this application predicts the right or wrong gesture with the help of algorithm.

Also, this application has been extended by adding time as a third component in the vector making it three-dimensional and hence, there are two approaches for gesture recognition, one is without time and another one with time. This paper deals with these two approaches. Basically, this is an analysis of the application and algorithm used by this application using these two approaches. The algorithm and the application with these two approaches are discussed in more details in Methods section. The comparison results between these two approaches are discussed in Results section. Finally, the summary, the future perspectives and the better approach is described in Discussion and Conclusion section.

# Methods

In this section, the methods which are required to create the application and the algorithm used by this application are discussed in detail. This discussion includes algorithm used, libraries used, application development, architecture and implementation of project.

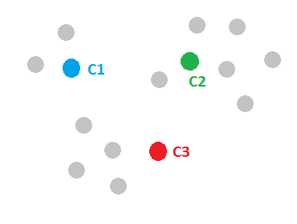
## K-Means Algorithm

The K-Means Algorithm is useful for forming a small number of clusters from a large number of observations. The objective of this algorithm is to divide N observations with P dimensions into K clusters so that the within-cluster sum of squares is minimized. The K-Means clustering algorithm can be applied to relatively large sets of data. The user specifies the number of clusters to be found. The algorithm then separates the data into spherical clusters by finding a set of cluster centers, assigning each observation to a cluster, determining new cluster centers, and repeating this process [1]. The step by step explanation to this algorithm is following [2].



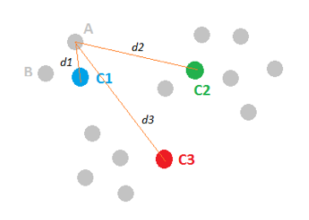
1. Given set of coordinates or observations.

We will take a 2D example to understand the steps of K-Means algorithm as illustrated in Fig. 1. Here, we want to assign these given coordinates or observations into three clusters. K-Means follows the following four steps.

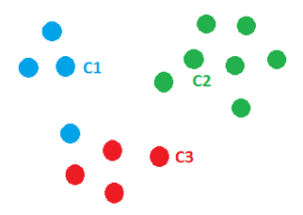


1. Cluster centers are initialized.

Step One: Initialize cluster centers randomly as illustrated in Fig. 2.

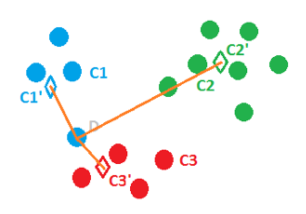


1. Assigning observations to the closest cluster center.



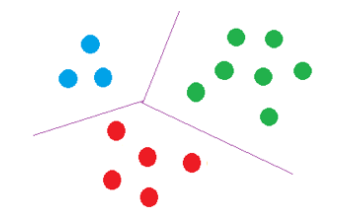
1. Observations are assigned to the closest cluster center.

Step Two: Assign observations to the closest cluster center as illustrated in Fig. 3 and Fig. 4.



1. Cluster centers are revised as mean of assigned observations.

Step Three: Revise cluster centers as mean of assigned observations as illustrated in Fig. 5.



1. Observations that are finally assigned to the clusters.

Step Four: Repeat step 2 and step 3 until convergence. And hence, we get the final solution as illustrated in Fig. 6 [2].

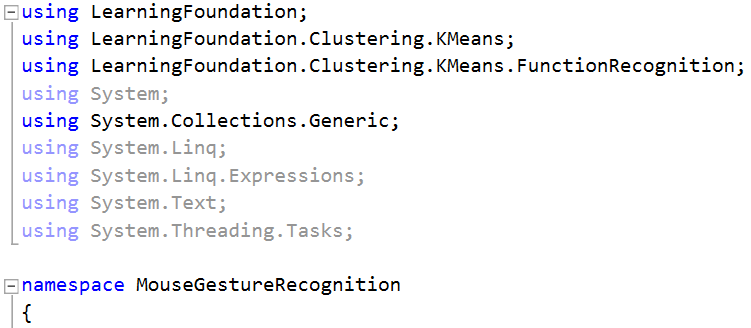
This K-Means Algorithm is the base for MouseGestureRecognition application which directly uses its extended Function Recognition Algorithm.

## LearningApi

LearningApi is a Machine Learning Foundation of useful libraries implemented in .NET Core developed by Frankfurt University of Applied Sciences [3]. The K-Means Algorithm is already implemented and also extended as Function Recognition Algorithm in this foundation as a library named AnomDetect.KMeans [4]. This library is directly used by MouseGestureRecognition application for recognition of gestures. Hence, we can say that LearningApi foundation provides the backbone for this application.

## AnomDetect.KMeans

AnomDetect.KMeans is a library inside LearningApi written in .NET Core [4]. In this library, the K-Means Algorithm is implemented and also extended as Function Recognition Algorithm. The MouseGestureRecognition application uses this library directly as a nuget package to access Function Recognition Algorithm for recognition of gestures as illustrated in Fig. 7.



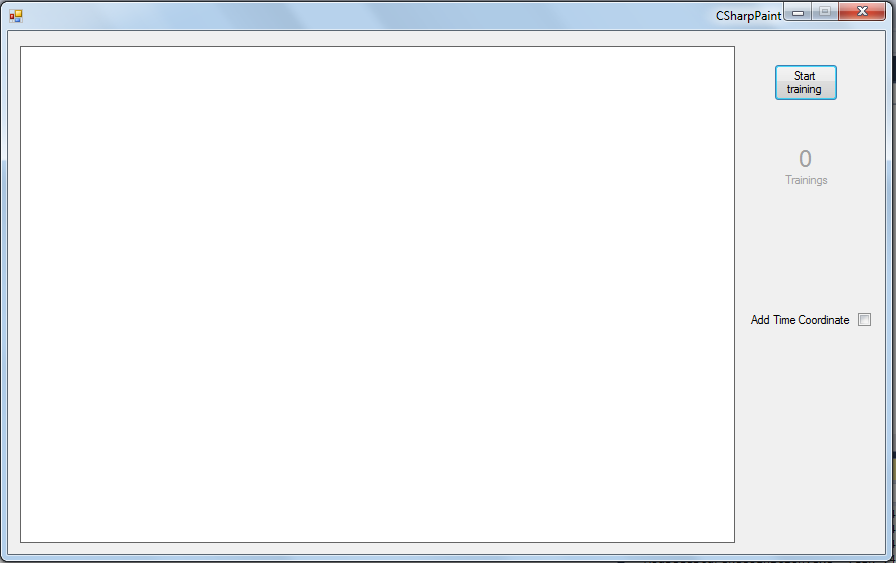
1. Code Snippet showing use of AnomDetect.KMeans library from LearningApi in MouseGestureRecognition application.

## MouseGestureRecognition

MouseGestureRecognition is a Windows Form Application created in .NET Framework and written in C# programming language. This application uses Function Recognition Algorithm for recognition of gestures. The K-Means Algorithm is already implemented and extended as Function Recognition Algorithm in AnomDetect.KMeans library from LearningApi. Hence, this application uses this library directly as a nuget package.

The main functions of this application is to capture mouse pointer movements, to collect two-dimensional space coordinates of those mouse points, to train the algorithm with this set of coordinates which is a learning set of mouse points as an array of two-dimensional vectors and to predict the right or wrong gesture after training or learning with the help of algorithm. Also, this application has been extended by adding time as a third component in the vector making it three-dimensional and hence, there are two approaches for gesture recognition, one is without time and another one with time.

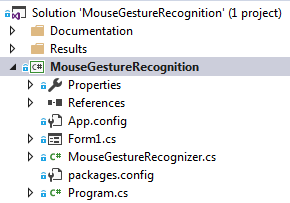
This application contains a picture box, buttons, labels and a check box for its working as illustrated in Fig. 8.



1. MouseGestureRecognition Application.

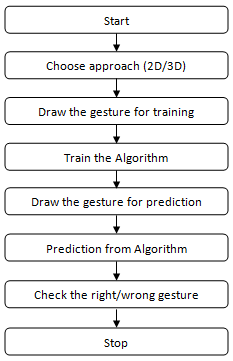
## Architecture

The MouseGestureRecognition application is created as a Windows Form Application in .NET Framework and written in C# programming language. The name of the Solution is MouseGestureRecognition and it contains only one project named MouseGestureRecognition of type Windows Forms App. The project also uses AnomDetect.KMeans library from LearningApi as a nuget package to access Function Recognition Algorithm for recognition of gestures. The project structure is shown in Fig. 9.



1. Project Structure of MouseGestureRecognition Application.

The architecture of the project describes the process of the application as illustrated in Fig. 10. First, when the application starts, the user chooses the approach. There are two approaches, one approach is 2D (two dimension) in which the time is not taken into account and another approach is 3D (three dimension) in which the time is taken into account. By default, the approach is set to 2D. To select the 3D approach, one can check Add Time Coordinate checkbox. Then, the user draws the gesture to train the algorithm. After training or learning, the user draws the gesture to check whether the current gesture is same as previous learned one or not (whether the gesture is right or wrong) with the help of prediction from algorithm.

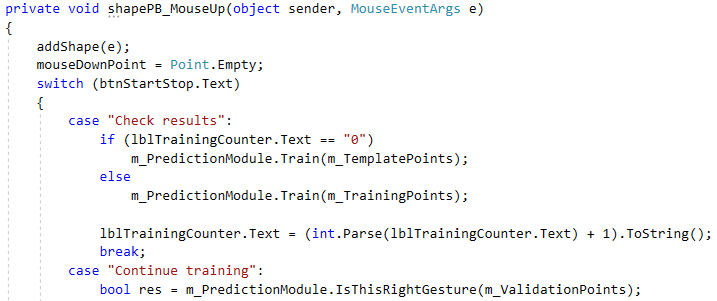


1. Architecture of the Project.

## Implementation

### Windows Form Class:

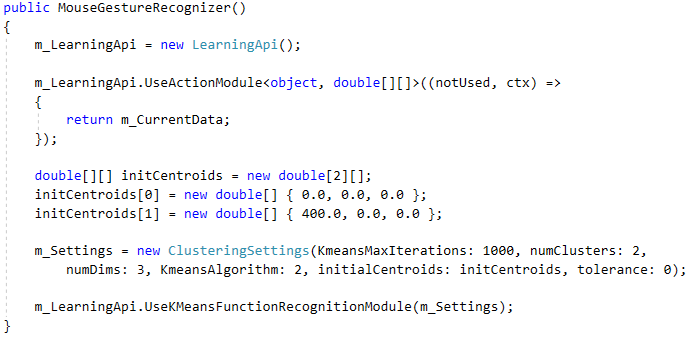
The MouseGestureRecognition application contains a picture box, buttons, labels and a check box for its working as illustrated in Fig. 8. These all are implemented and defined in Form Class. Along with this, all their event handlers (methods) are also defined in this class. These event handlers or methods are written according to the process of the application. The training of the algorithm and prediction from the algorithm are also called from one of these methods as illustrated in Fig. 11.



1. Code Snippet from Form Class.

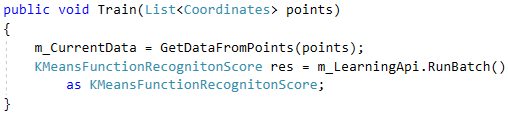
### MouseGestureRecognizer:

The MouseGestureRecognition application uses Function Recognition Algorithm from AnomDetect.KMeans library. This procedure is implemented in MouseGestureRecognizer Class. For using Function Recognition Algorithm, first step is to give Clustering Settings as a parameter to it as illustrated in Fig. 12. These settings include number of iterations, number of clusters, number of dimensions, initial centroids etc. As we have only right and wrong solution, so, we will have only 2 clusters. Also, we are working up to three dimensions, so, we will give number of dimensions as 3. For initial centroids, we can choose it randomly or we can follow some steps to increase its efficiency. These steps are following. Step 1: From n objects calculate a point whose attribute values are average of n-objects attribute values. So, first initial centroid is average on n-objects. Step 2: Select next initial centroids from n-objects in such a way that the Euclidean distance of that object is maximum from other selected initial centroids. Step 3: Repeat step 2 until we get k initial centroids [5]. Here, we have chosen random values for initial centroids.



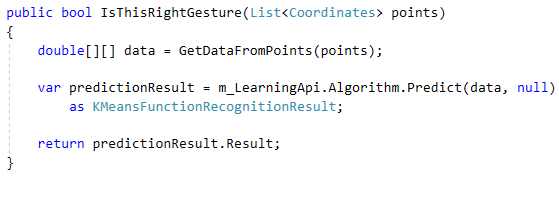
1. Code Snippet from MouseGestureRecognizer showing Clustering Settings.

The training of Algorithm is called in MouseGestureRecognizer through Train method as illustrated in Fig. 13.



1. Code Snippet from MouseGestureRecognizer showing Train method.

The prediction from Algorithm is called in MouseGestureRecognizer through IsThisRightGesture method as illustrated in Fig. 14.



1. Code Snippet from MouseGestureRecognizer showing IsThisRightGesture method.

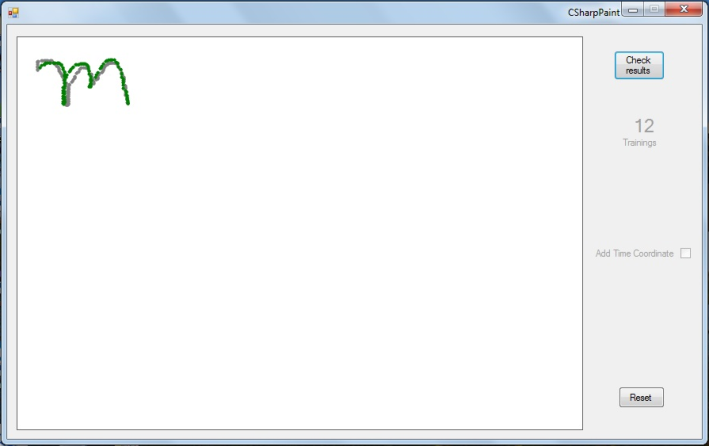
# Results

According to the MouseGestureRecognition application, the following results are obtained for both with and without time approach.

## Without Time Approach (2D vectors)

### Sign 1:

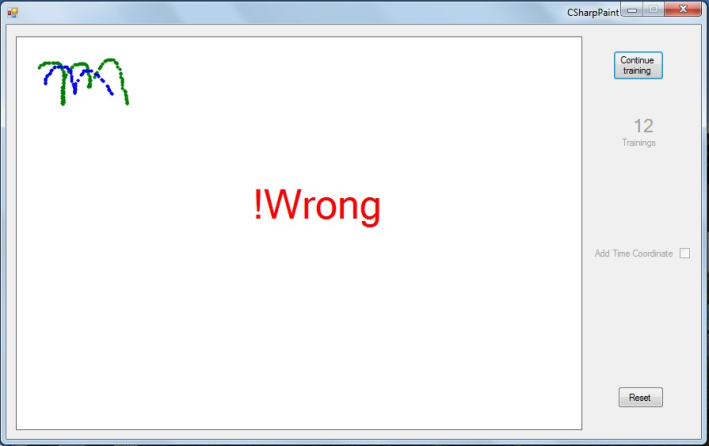
The trained image of Sign 1 is shown in Fig. 15 and its predictions are shown in Fig. 16 and Fig. 17. We can see that Prediction 1 is right and Prediction 2 is wrong because Prediction 1 has approx. same gesture as the trained gesture but Prediction 2 has a different gesture. This shows that our algorithm is working fine for this gesture.



1. Trained Image of Sign 1.



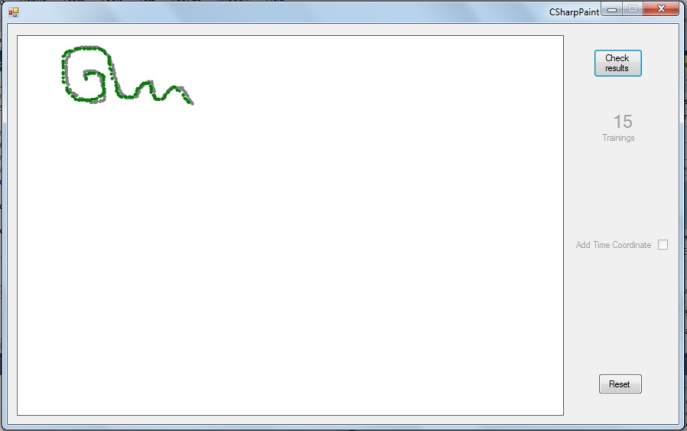
1. Prediction 1 of Sign 1.



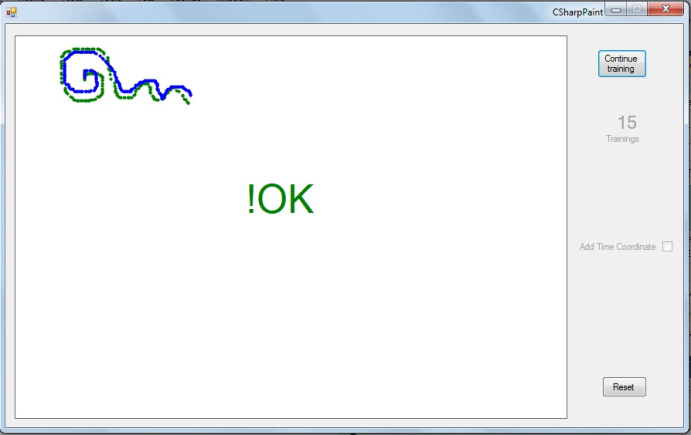
1. Prediction 2 of Sign 1.

### Sign 2:

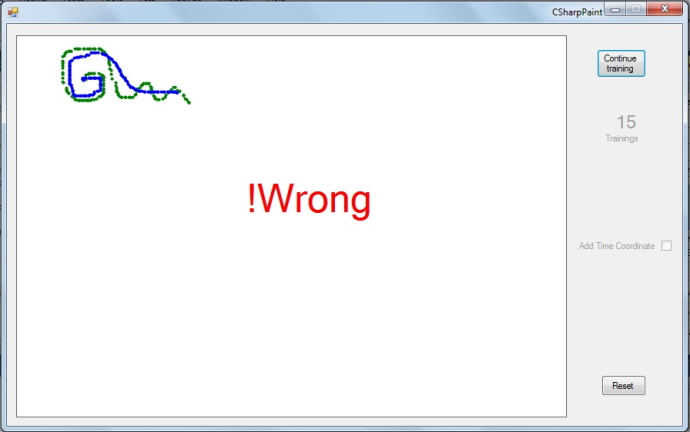
The trained image of Sign 2 is shown in Fig. 18 and its predictions are shown in Fig. 19 and Fig. 20. We can see that Prediction 1 is right and Prediction 2 is wrong because Prediction 1 has approx. same gesture as the trained gesture but Prediction 2 has a different gesture. This shows that our algorithm is working fine for this gesture also.



1. Trained Image of Sign 2.



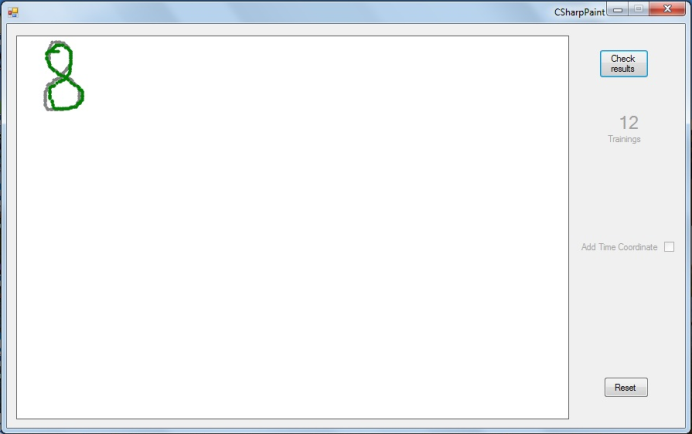
1. Prediction 1 of Sign 2.



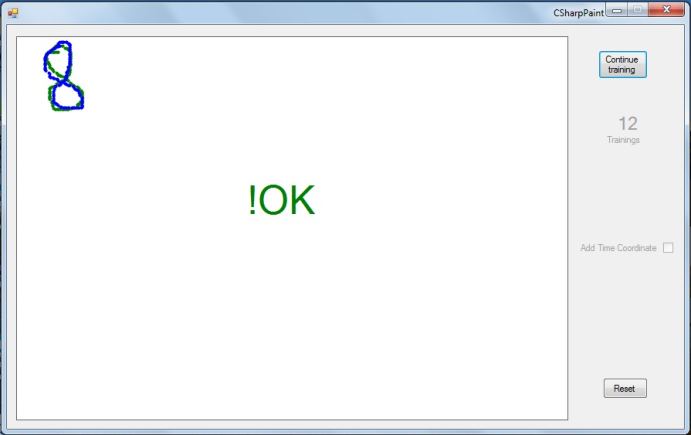
1. Prediction 2 of Sign 2.

### Sign 3:

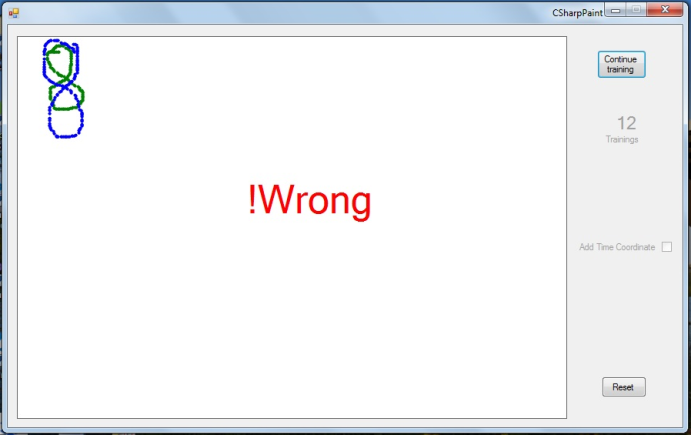
The trained image of Sign 3 is shown in Fig. 21 and its predictions are shown in Fig. 22, Fig. 23 and Fig. 24. We can see that Prediction 1 is right, Prediction 2 is wrong and Prediction 3 is also right. Here, Prediction 1 has approx. same gesture as the trained gesture and Prediction 2 has a different gesture. So, these two predictions are giving correct results but though Prediction 3 has a different gesture, the output is right. This is not a correct result. This shows that our algorithm is not working fine for this gesture.



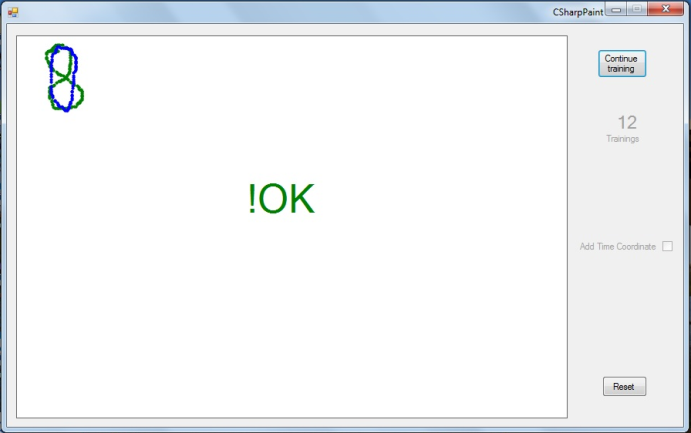
1. Trained Image of Sign 3.



1. Prediction 1 of Sign 3.



1. Prediction 2 of Sign 3.



1. Prediction 3 of Sign 3.

## With Time Approach (3D vectors)

### Sign 1:

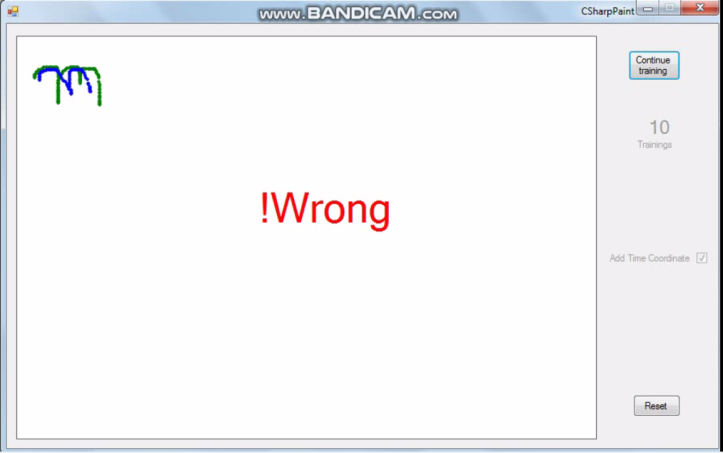
The trained image of Sign 1 is shown in Fig. 25 and its predictions are shown in Fig. 26, Fig. 27 and Fig. 28. We can see that Prediction 1 is right, Prediction 2 is wrong and Prediction 3 is also wrong. Here, Prediction 1 has approx. same gesture and same time taken to draw the gesture as the trained gesture and Prediction 2 has a different gesture. So, these two predictions are giving correct results but though Prediction 3 has approx. same gesture, the output is wrong. This is also a correct result because the time taken to draw the gesture in Prediction 3 is more than the time taken to draw the gesture in trained gesture. This shows that our algorithm is working fine for this gesture. The animation can be seen in Fig. 29 by double click on it in MS Word.



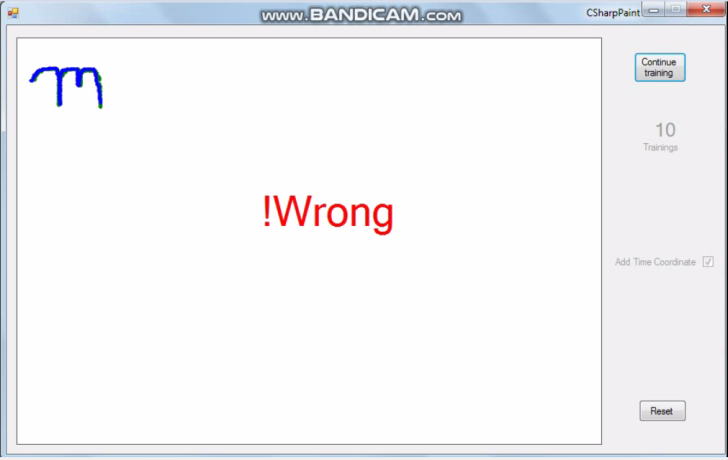
1. Trained Image of Sign 1.



1. Prediction 1 of Sign 1.



1. Prediction 2 of Sign 1.



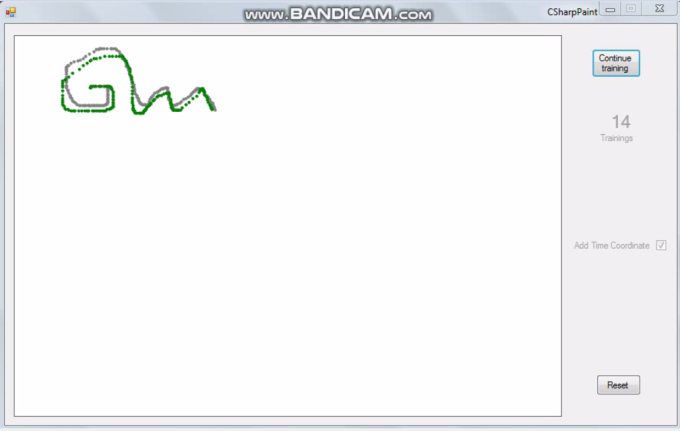
1. Prediction 3 of Sign 1.



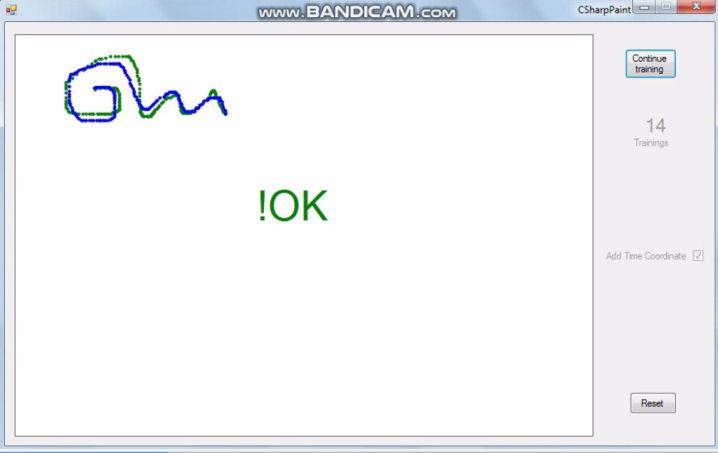
1. Animation of Sign 1 (Played by double click on it in MS Word).

### Sign 2:

The trained image of Sign 2 is shown in Fig. 30 and its predictions are shown in Fig. 31, Fig. 32 and Fig. 33. We can see that Prediction 1 is right, Prediction 2 is wrong and Prediction 3 is also wrong. Here, Prediction 1 has approx. same gesture and same time taken to draw the gesture as the trained gesture and Prediction 2 has a different gesture. So, these two predictions are giving correct results but though Prediction 3 has approx. same gesture, the output is wrong. This is also a correct result because the time taken to draw the gesture in Prediction 3 is more than the time taken to draw the gesture in trained gesture. This shows that our algorithm is working fine for this gesture also. The animation can be seen in Fig. 34 by double click on it in MS Word.



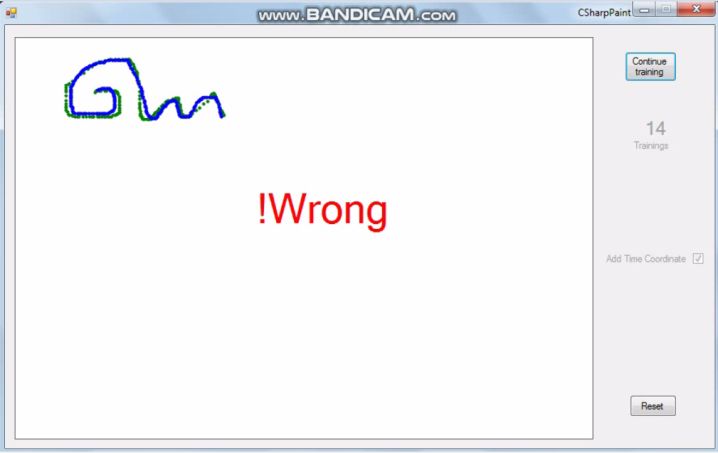
1. Trained Image of Sign 2.



1. Prediction 1 of Sign 2.



1. Prediction 2 of Sign 2.



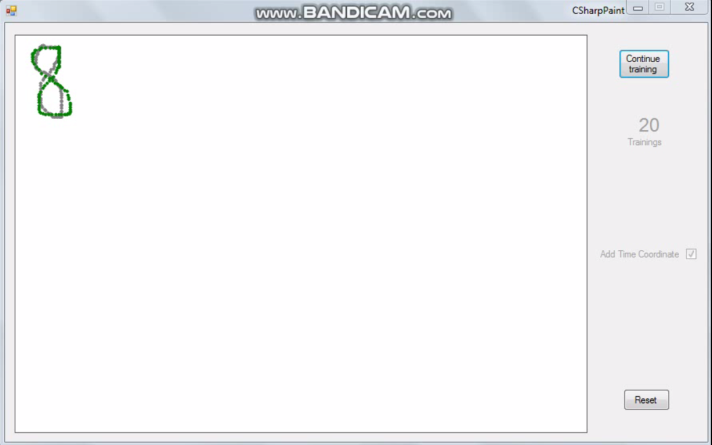
1. Prediction 3 of Sign 2.



1. Animation of Sign 2 (Played by double click on it in MS Word).

### Sign 3:

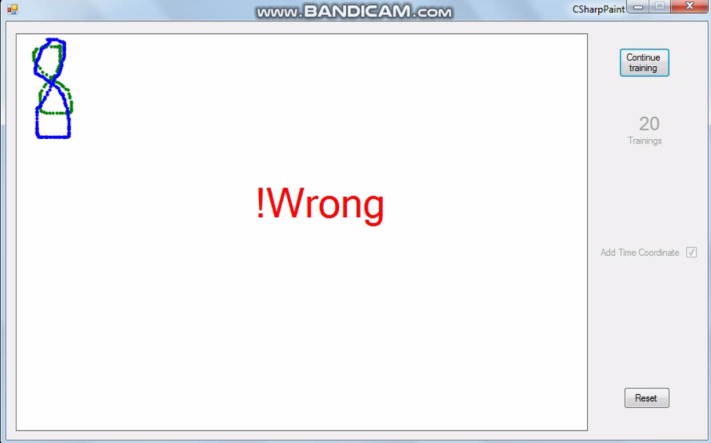
The trained image of Sign 3 is shown in Fig. 35 and its predictions are shown in Fig. 36, Fig. 37, Fig. 38 and Fig. 39. We can see that Prediction 1 is right, Prediction 2 is wrong, Prediction 3 is wrong and Prediction 4 is also wrong. Here, Prediction 1 has approx. same gesture and same time taken to draw the gesture as the trained gesture and Prediction 2 has a different gesture. So, these two predictions are giving correct results but though Prediction 3 has approx. same gesture, the output is wrong. This is also a correct result because the time taken to draw the gesture in Prediction 3 is more than the time taken to draw the gesture in trained gesture. Also, Prediction 4 has a different gesture and the output is wrong, so, the result is correct. But for Prediction 4 in case of 2D approach (without time approach), this gesture was giving incorrect result. This shows that our algorithm is working fine for this gesture in case of 3D approach (with time approach) but not for 2D approach. The animation can be seen in Fig. 40 by double click on it in MS Word.



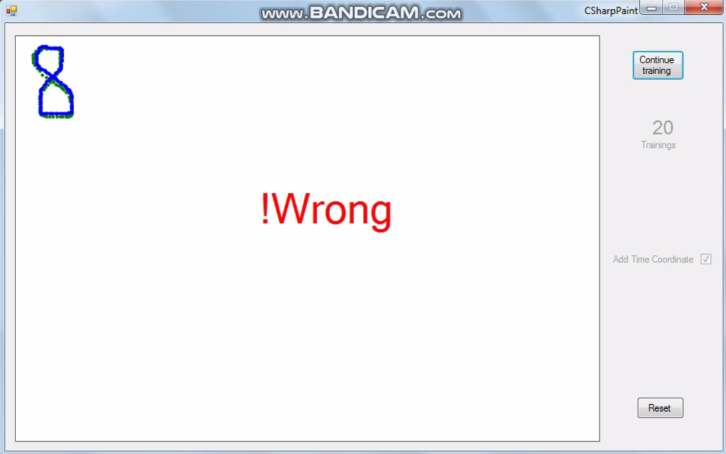
1. Trained Image of Sign 3.



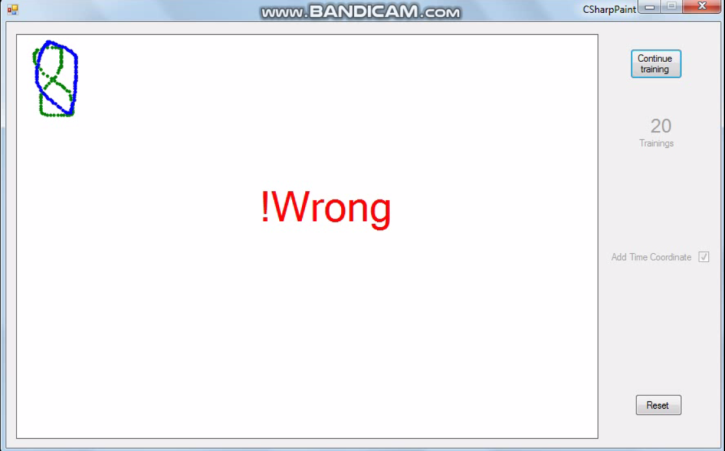
1. Prediction 1 of Sign 3.



1. Prediction 2 of Sign 3.



1. Prediction 3 of Sign 3.



1. Prediction 4 of Sign 3.



1. Animation of Sign 3 (Played by double click on it in MS Word).

# discussion and conclusion

After observing the results for different gestures with both 2D approach (without time approach) and 3D approach (with time approach), we can conclude that for some gestures 2D approach is working fine but for some others it is not. But by using 3D approach, same gesture which was giving incorrect results in 2D approach, is giving correct result. So, we can say that 3D approach is better than 2D approach. Hence, adding time component in vector can improve its recognition efficiency.

The time component in 3D approach can dominate over space coordinates and this can also give incorrect result. This is the disadvantage of using 3D approach.

For future perspectives, one can work on making a balance between time and space coordinates to make the recognition efficient. Apart from this, one can also improve the centre problem that current application is facing. The same gesture if drawn in different parts of picture box, gives incorrect results. So, these are the areas of improvement for this application.

# References

|  |  |
| --- | --- |
| [1] | NCSS Statistical Software, "K-Means Clustering," [Online]. Available: https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/K-Means\_Clustering.pdf. [Accessed 2019]. |
| [2] | healthcare.ai, "Step by Step to K-Means Clustering," [Online]. Available: https://healthcare.ai/step-step-k-means-clustering/. [Accessed 2019]. |
| [3] | Frankfurt, University of Applied Sciences, "LearningApi," 2019. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi. [Accessed 2019]. |
| [4] | Frankfurt, University of Applied Sciences, "AnomDetect.KMeans," 2019. [Online]. Available: https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi/tree/KMeans-branch/LearningApi/src/MLAlgorithms/AnomDetect.KMeans. [Accessed 2019]. |
| [5] | A. M. Baswade and P. S. Nalwade, "Selection of Initial Centroids for k-Means Algorithm," 2013. [Online]. Available: https://www.ijcsmc.com/docs/papers/July2013/V2I7201338.pdf. [Accessed 2019]. |