Comparison of MLP and SVM for Autonomous Vehicle Navigation

[1]Prof. Tejal Deshpande, [2]Joel Kinny, [3]Tanmay Malekar
[1]Assistant Professor
[1]Department of Electronics and Telecommunications Engineering
[1]Xavier Institute of Engineering, Mahim, Mumbai - 400016

Abstract: Lane detection, overall trajectory estimation and reliable navigation in an unknown environment are the key challenges of autonomous driving. This paper compares two machine learning based approaches; SVM and MLP, for lane trajectory detection. Key focus is on estimating the scope and applicability of an offline trajectory learning model for real time steering control. The actual indoor navigation is carried out to compare the performance of SVM and MLP algorithms and verify the reliability of navigation for unknown trajectories. Single Raspberry Pi camera has been used for trajectory sensing and path planning. Several real indoor tests are implemented to assess the performance. Results indicate that SVM provides way better accuracy estimation of trajectory curving, therefore the steering angle as compared to an MLP model. Major contribution of this experimentation is the reliability and the study of images captured from low cost Pi-camera and confirming the scope of using it for in-house robotic navigation.

IndexTerms **- Autonomous navigation, machine learning, image classification, neural networks, camera, robotics.**

I. INTRODUCTION

Significant improvements in the last decade have greatly advanced self-driving car technology. This fairly new technology will have profound global impacts that could bring about improvements in the overall efficiency, convenience, and safety of our roadways and transportation systems. Including countries like UK and USA, several Asian countries like India, China, Japan, Korea and Singapore, are making significant contributions to the field. Although these countries are at different stages of building a fully autonomous vehicle system, more research and effort is needed for the advancement of these technologies so that they can be reliable enough on a large scale for real life practical applications. The tech giant Google has developed a fake city, Castle, a 100 acre area in the California desert which consists of only roads, driveways and intersections. The main goal is to recreate situations, the robot cars are likely to encounter in real-life environments. Google engineers use different elements like signs, cones, mannequins, and even other cars, to devise scenarios that challenge the driver-less cars to respond as human drivers would. Main contribution of this work is to create a scaled down version of an autonomous self-driving environment to evaluate the driving reliability through paths with randomly varying curvatures, while trajectory sensing is done by a low cost camera. Motivation is to replicate proto type robotics navigation or toy car in lab scenario. If the toy car detects any obstacles it should change its path or stop in case an object is in front of the car and be able to decide whether to stop or move depending on some other admiring support. Major contribution of this experimentation is the consistency study of trajectory estimation accuracy for images capture from a low-cost Pi-camera, so that an off-line trained model can then be used for real time control the steering of motors. Our expected outcome is to have such features incorporated in our system which can as well help to overcome obstacles.

The analysis and literature survey for inculcating the concepts used in the experimentation and the major methods used by other authors and their work have been presented in Section II. Section III and Section IV is an overview of the methodologies implemented in the paper where the various steps involved in the experimentation and setup have been noted down. Section V talks about the two techniques, Support Vector Machine and Multi-layered Perceptron being used in this project for autonomous driving. The comparison of all parameters and their performance is recorded along with the observations recorded during the course of this project. The various methods and tuning of parameters and their results are mentioned in these sections.

II. LITERATURE REVIEW

Autonomous driving systems and self-driving cars have been under development for the past few decades. There are various methods and implementations of such autonomous vehicles which vary from one another. Every method of implementation has certain advantages and disadvantages. These methods will vary as per the user requirement, environment, hardware limitations, processing accuracy and so forth. In autonomous driving systems, the main components include sensor technology, processing unit, driving controller and some feedback mechanism. The major emphasis in this set up is kept on the sensing methods which includes a single low-resolution camera system.

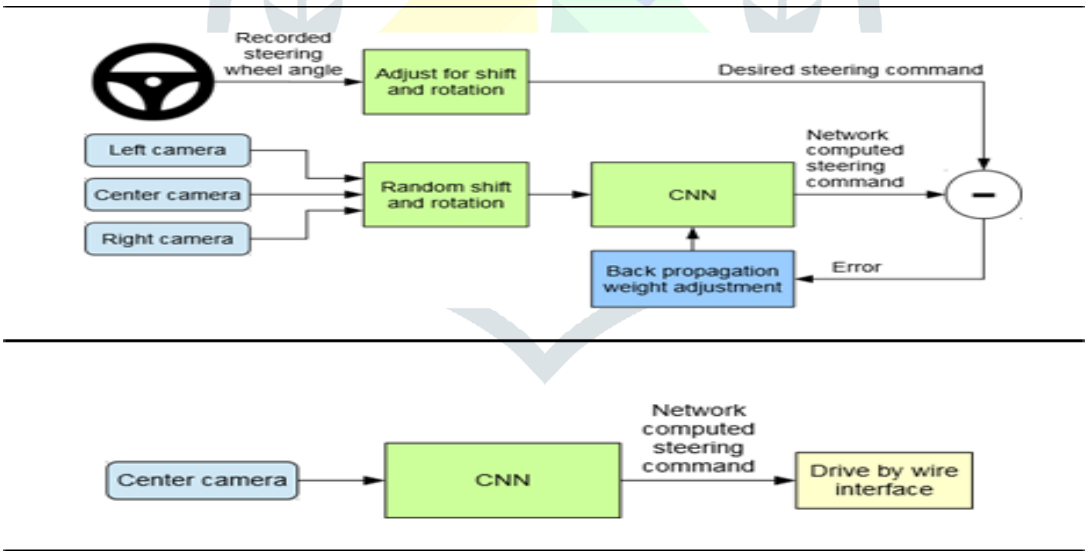
2.1. Lane Detection and Recognition

In the paper presented by Truong et al., we see that the system comprises of a single monocular camera for providing input images in lane detection and recognition [1]. This camera is the sole input of the autonomous driving system and hence requires further processing of input data. The input video from this camera is used only to detect the road lanes and hence serves as the basis for an autonomous driving system [1]. This paper talks about a lane and road detection algorithm which provides a robust and

accurate algorithm for detection as well as road recognition. First, the road image is enhanced using some conventional image processing techniques. In the enhanced image, possible lane marking pixels are detected using Canny operator. Next, apply parallel thinning algorithm into edge map to obtain skeleton image. Detected lane marking pixels after taking skeleton process, then, are utilized to select control points for NUBS interpolation to construct left and right road lanes. In that case, the right and left lanes to be detected are well separated, which means that each lane can be considered as separated from the other (unlike most of previous works which have lane models of a uniform width) [1]. They have used vector-lane-concept to select and correct these control points, overcome noise problem to get more robust result of road lane boundaries. Finally, a simple mathematical model to estimate left lane and right curvature for autonomous vehicle system. This method functioned well even in the presence of noise, however did not see any kind of neuro or fuzzy logic to control the operation of the car. The problem with such a method is that it is requires high computations for processing and recognition of data. This could be fatal in real time scenarios where decisions have to be made in milliseconds.

2.2. Neural Networks and Machine Learning in autonomous systems

Neural computing can achieve massive parallelism and very high computation speeds. The main approach of our project is the use of artificial neural networks for simplifying complicated tasks such as recognition and classification which is not possible with traditional methods. The next approach adopted by Mariusz Bojarski et al., makes use of the convolution neural computing for their autonomous vehicle system. The system developed, DAVE-2 is an autonomous vehicular system which has a much better performance as compared to the previous adopted method [2]. The system automatically learns internal representations of the necessary processing steps such as detecting useful road features with only the human steering angle as the training signal. They never explicitly trained it to detect, for example, the outline of roads. Their end-to-end system optimizes all processing steps simultaneously. This system consists of three cameras for collection of data on the road. This video is captured simultaneously with the steering angle applied by the human driver. This steering command and video data is captured in the Controller Area Network (CAN) [2]. The steering command is represented as 1/r where r is the turning radius in meters. We use 1/r instead of r to prevent a singularity when driving straight (the turning radius for driving straight is infinity. 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values). This model is trained with human steering command and during testing phase, only a single center camera is used for vision and the network computed steering command is given to the driver controller. A small amount of training data from less than a hundred hours of driving was sufficient to train the car to operate in diverse conditions, on highways, local and residential roads in sunny, cloudy, and rainy conditions. The CNN is able to learn meaningful road features from a very sparse training signal (steering alone). The main hardware requirement of this method of computing or image processing task is the use of Graphics Processing Unit (GPU) [2]. Fig.2 shows us the overview of the implementation in the paper by Mariusz Bojarski et al.



2.3. SVM for robotic navigation

Konstantinos Charalampous et al. [3] have used maximum margin notion to constructs a global 3D metric map of the environment and then unrolling planer path finding on the corresponding 2D map. However Global path planning (GPP) operates in high level and are only suitable for finding shortest path to a goal by avoiding dead ends. In [25] authors have used SVM (Support Vector Machine) for wheel chair navigation by using EEG signals. EEG being prone to noise and a multidimensional signal is likely to fail under real environment. In the paper by Yuanqing Lin and et al, for SVM training they had approximately 1000 classes, each class containing more than 1000 images on an average. [22] In spite of having such a large number of input images for each class they managed to obtain an overall accuracy of 71.8 % for their large-scale image classification system. In a

paper by Kuo-Ho Su et al., an autonomous wheeled agent is implemented using SVM for path planning. An autonomous agent is placed in a room full of obstacles and a camera is placed overviewing the entire area. The camera then uses the image to plot a path for the autonomous agent to travel from one point to another. A Gaussian SVM is used to smoothen the path and provide reliable navigation. [22].

III. RASPBERRY PI CAMERA AND VIDEO STREAMING

The design for any good system is the scalability factor so that it can be downsized to a local region or upgraded for a major operation. The Autonomous driving system proposed in our project is a scaled down model of a remote control car. To suit the requirements at such a small scale and operating in a confined local environment, high end systems would be redundant and quite expensive. The proposed system would include a scaled down version of a car, with a Raspberry Pi Model 3 as the on board system controller. The Raspberry Pi will acquire data from the sensors on the car with the help of Pi Camera which will be the visual sensor of this system. The Pi Camera is a very lightweight and versatile camera module and would go on board the model car. In the paper proposed by Rupali Ikhankar et al., a robot makes use of the Raspberry Pi and Pi Camera for localization and navigation within an environment. The images captured from the Pi Camera are overlaid together to form a continuous video stream and helps in faster processing of the video. The usual video format of recording of the Pi Camera is the H.264 format encoding which is usually has a bit rate of 17 Mbps. This data is streamed over a local wireless network to another device such as a laptop or another client. Similarly, Valeriu Manuel et al., designed a system where the Raspberry Pi platform as an add-on to a classic TV set in order to transform it into an internet enabled device [5]. The Raspberry Pi is used to control the local storage, display and speaker, and is able to communicate with a server in order to control the play and update the local content with the data received from the server [5]. The chosen communication solution was TCP because of the file size, the need to transfer files correctly and low processor computational power (that would be necessary in the case of an alternate UDP implementation). This solution is easy to implement, upgrade and allows for personalized advertisements.

IV.DESIGN AND EXPERIMENTAL SETUP

4.1. Experimental Setup

A Raspberry Pi board attached with a Pi camera module and a sensor is used to collect input data. Two client pro- grams run on Raspberry Pi for streaming colour video data to the computer via local Wi-Fi connection. A multi thread TCP server program runs on the computer to receive streamed image frames and data from the Raspberry Pi. Image frames are captured and acts as visual sensor for the car. This data is then sent to the processing unit which handle tasks like object detection and neural network training. Once this data has been processed, it then predicts the command that has to be given to the car for navigation (forward, left, right). The car unit is controlled via the Raspberry Pi which performs steering operations of the car like moving forward, stopping, turning left and tuning right. The project implementation will consist of three stages. Fig. 2 shows us the block diagram of the setup used for the training phase and later for the autonomous driving where the Raspberry Pi will make decisions based on input data.

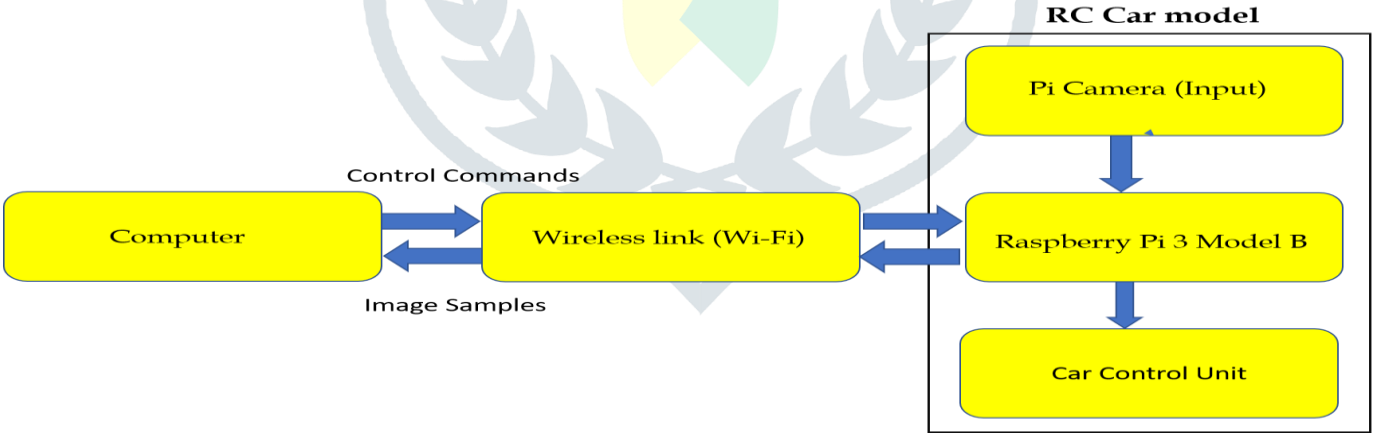


Fig. 2. Overview of the autonomous vehicle operation

4.2 Acquiring training set

The Raspberry Pi camera module mounted on top of the car captures the images required for the training and the autonomous navigation of the car. The camera module is attached to a piece of cardboard which was not sturdy and would change the angle of inclination of the camera. A slight shift in the angle of inclination would cause the model trained on the previous set of images to perform very poorly due to the mismatch between the current viewport and the earlier viewport. We fixed this issue by mounting the camera on a sturdy stand and ensured that the camera would be centered thus, providing a very broad view. The training phase of the project involves supervised learning where the target directions will be provided by human input. The car is placed on our test track which will be used for all training purposes and later for autonomous navigation. The program which is run on the Raspberry Pi is used to control the motors, acting as a remote control with the keyboard keys as the human input for

acquiring the target direction. Along with this, images are captured by Raspberry Pi camera and stored along with the direction vector. All the images captured during this training process need to be segregated manually in their respective folders based on the images, these folders include data for forward, idle, left and right.

4.3 Training and testing phase

In this paper, we will be comparing two different methods for image classification and autonomous driving. Support Vector Machine model and Multi Layered Perceptron model have been used for this purpose. The different advantages and disadvantages of both models have been analyzed and results stated accordingly in the later sections. Both these models take input from the Raspberry Pi camera. The camera position provides a stabilized set of images, but a large area of the new image set would contain information irrelevant for the navigation of the car. Since portions of the road necessary for the next navigation action would be almost entirely in the lower half of the image, we used this as inputs to the model for training and prediction. The collected images along with their respective class, are used to start training the model. We split the image horizontally and convert the lower half into an array where the color layers flatten into a single gray scale layer. The color information does not provide additional information to identify features and hence the images are converted into gray scale. We resize the array according to the dimensions specified in the configuration file and flatten it. The flattened array for each image serves as input to the neural network, and its corresponding class label value is the expected output. This optimization is done by the scipy and sklearn libraries of Python. This optimization program generates a pickle file which contains the array of minimized error values and network parameters which will be later used in the testing and validation phase. As optimization and learning of neural networks is computationally done with the help of Graphical Processing Unit (GPU), the training is not done on the Raspberry Pi but instead done on a laptop with a dedicated GPU and CUDA cores. Fig. 3 is the basic architecture for a neural approach to the task of self-driving.

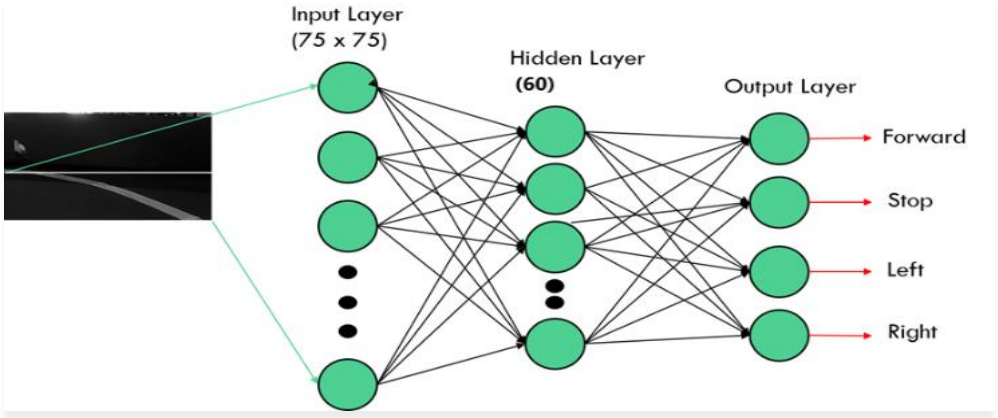


Fig. 3. Neural Network architecture implemented for self-driving

4.4 Autonomous navigation using trained parameters

The model generated is used for autonomous navigation. The lower half of the images captured by the Pi Camera are again used as input to the model where the lower half of the images are considered as input nodes and the trained model file is used along with the input images for autonomous navigation [3]. The accuracy of the prediction and driving will depend on the training data given to it and varies with the conditions during the prediction or validation phase.

4.5. SVM for navigation

Support Vector Machine (SVM) is a class of machine learning method developed by Vapnik et al. in the early 1990s. SVM transforms a nonlinear separable problem into a linear separable problem with different kernel functions and then attempts to find the separating hyperplane that maximizes the margin between the samples from two classes. SVM finds the optimized hyperplane by simultaneously minimizing the risk and maximizing the margin between the hyperplane and support vectors. Mathematical representation of SVM is as follows:

$$\begin{cases} min\, f\,(W)\frac{1}{2}\|W\|^2 + C\sum_i^N \xi_i \\ st = y_i\left[W.K(x_i, x_j) + b\right] + \xi_i \geq 1 \quad i = 1,2,\dots N \end{cases} \tag{4.1}$$

where W is the weight vector and b is the bias, both of which are optimized by the training process. The regulating parameter C is a penalty factor, which can balance the model complexity and tolerable risk. In addition, ξ_i 's are positive parameters called slack variables, which represent the distance between the misclassified sample and the optimal hyperplane. Function $K(x_i, x_j)$ is the kernel function. The commonly used optimization method of SVM classifier is cross-validation accompanied with grid-search. In our implementation, a 10-fold cross-validation is adopted to enhance the performance of the proposed SVM classifier.

In this experimentation radial basis function is used as the kernel along with one-vs-all classification. SVM performs well in most of the scenarios because of its high generalization performance without the need to add a prior knowledge, even when the dimension of the input space is very high. During testing phase, given a pattern vector to an SVM model, the result will be a measure of the distance of the pattern vector from the hyper plane constructed as a decision boundary between this class and rest of the classes. A positive value represents that the pattern belongs to the target class and vice versa [19]. In our case, this can be used to classify our predefined four classes required for navigation (Forward, Left, Right, Idle). Another advantage of SVM is its ability to handle large scale data efficiently

4.6 Autonomous wheeled agent and car control unit

The main focus of providing a self-driving car requires some sort of a model which will demonstrate the functioning of an actual vehicle. It will be able to drive on linear and curvilinear paths, by having some sort of speed control to handle steering and smooth driving. This is achieved by using two motors each of 150 rpm which work along with the IC L293D [10]. Fig 4 shows the image of our self-driving bot. The basic construction of the bot is done using a chassis and two wheels. The Raspberry Pi which is powered by the power bank and the Pi camera both are placed on top of the chassis. The two wheels are both connected to a 150 rpm DC motor. which are powered by a rechargeable battery fixed at the back of the bot. IC L293D is used for direction and speed control of the bot [10].

V. EXPERIMENTAL RESULTS

5.1. Creation of dataset for comparison and analysis

In Table 1, the dataset which has been created by taking a small subset of images captured by the Raspberry Pi wheeled agent. We have taken images from three different tracks. Since there are four directions of movement for the wheeled agent, there are four output classes, namely Forward, Left, Right and Idle. For the purpose of analysis, we have made a dataset of about 1230 images, with 100 images per direction. By doing so, the network will not be biased towards any direction and ensures proper training of the network. The three tracks also ensure generalisation of the network so that it can perform well in presence of a new environment. The database contains 100 images per direction for every track. Hence, there are 300 images for each direction of the entire database. The three datasets used for comparison of algorithm accuracy contain unbiased data and have almost equal number of images for all directions. The datasets are split into training and testing sets in a 70-30 split. The training set consists of 861 image samples whereas the test set consisted of 369 image samples. The test images are exclusively selected for testing which is unseen data and the performance of the model is evaluated based on this data. The test analysis and training are done using Python's scikit-learn and scipy packages run on Spyder on a system with NVIDIA 940MX GPU 4GBDDR5.

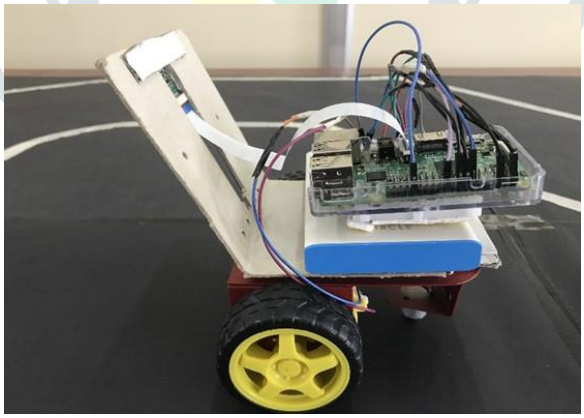


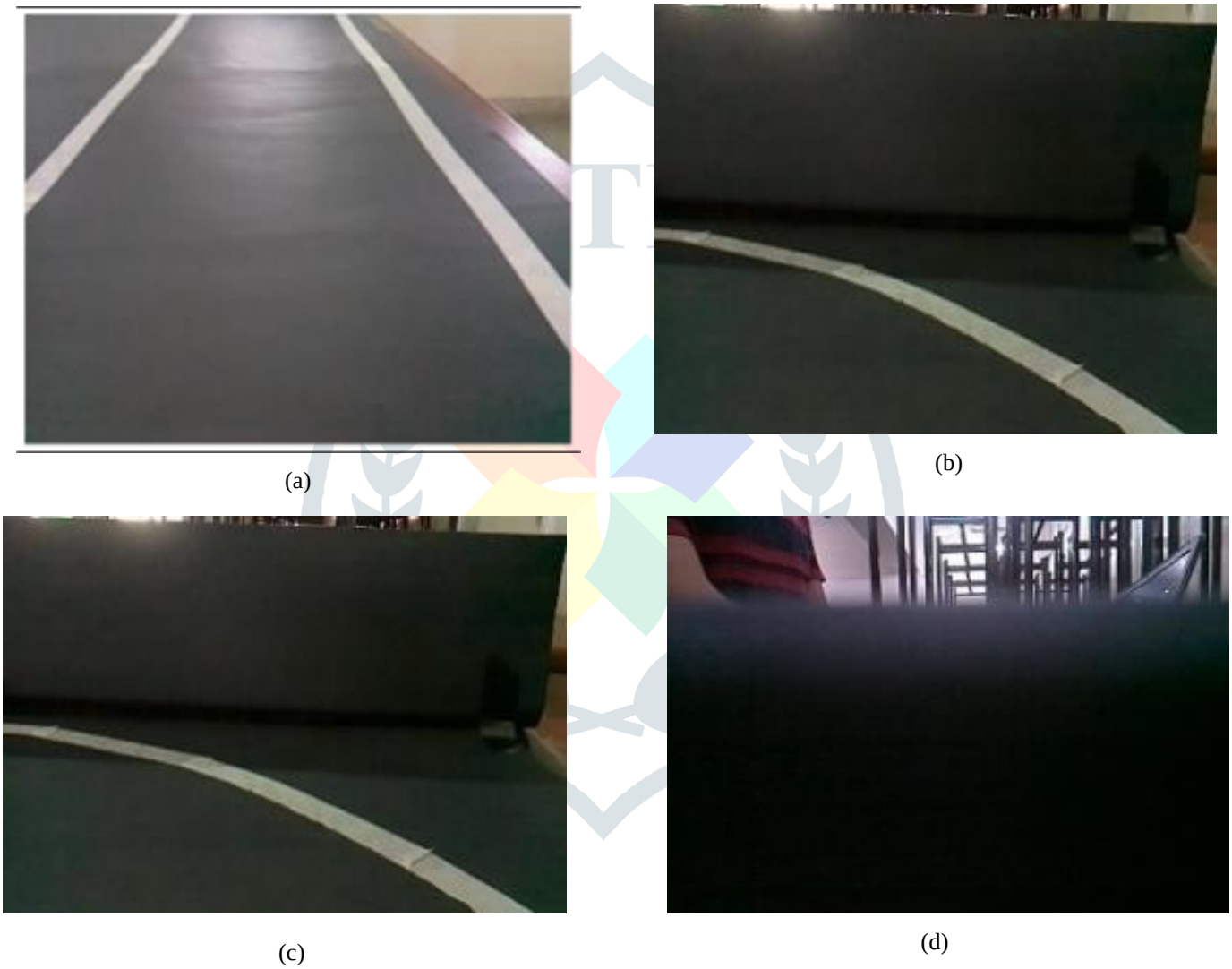
Fig. 4. Autonomous wheeled agent

Table 1
Datasets used for comparison and target direction

Track 1	110 Forward, 80 Idle, 106 Left, 118 Right
Track 2	103 Forward, 125 Idle, 100 Left, 80 Right

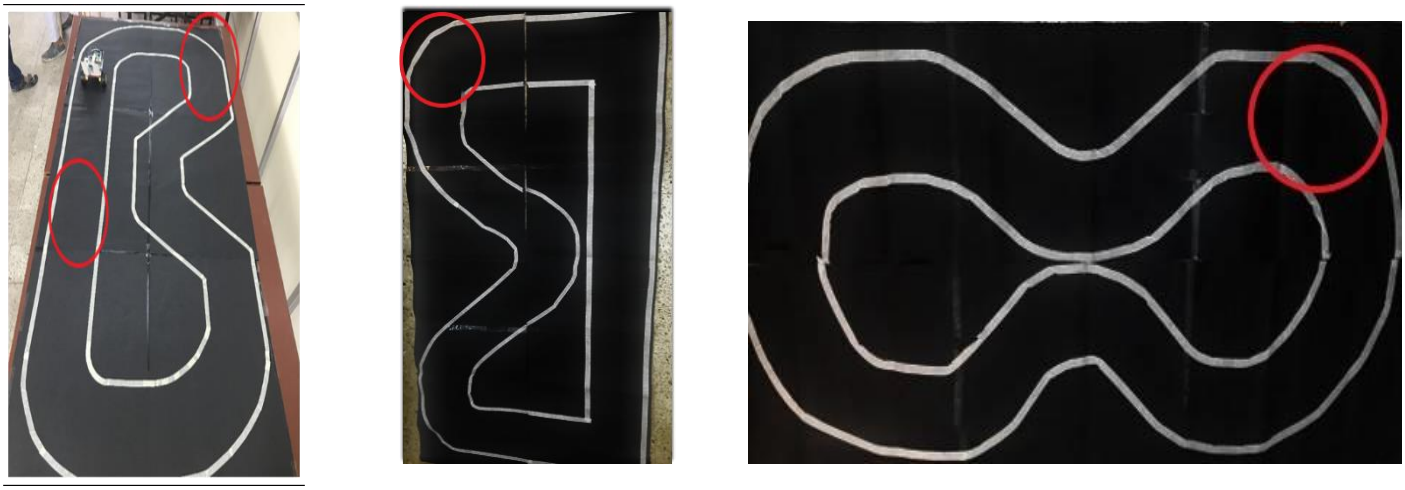
Track 3	90 Forward, 99 Idle, 99 Left, 104 Right

The sample images which form the dataset for image classification and also serve as the input to the classification models are shown in Fig. 5. Panel (a) is the image for identification of the forward class. Subsequently, Panel (b), Panel (c) and Panel (d) are the images for identification of left, right and idle classes respectively. The three tracks that were created for training and test purposes are shown in Fig. 6. The track in Fig. 6. (a), has a straight path with smooth left turns and moderate left and right turns in the middle. Similarly, the track in Fig. 6. (b), has a straight path followed with sharp left turns as contrast to the smooth left turns in Fig. 6. (a). The track shown in Fig. 6. (c) consists only of smooth left and right turns with no straight path or edges. The three tracks are made in order to reduce the model being too biased to a certain direction with each track having its own peculiarities. Also, multiple tracks ensure generalization of the model and it will able to classify new data with good accuracy. The inset circle in Fig. 6. (a) correspond to the forward and left directions shown in Fig. 5. (a) and Fig. 5. (b) respectively. The inset circle in Fig. 6. (b) corresponds to the idle position shown in Fig. 5. (d) The inset circle in Fig. 6. (c) corresponds to the right direction shown in Fig. 5. (c).



(a) (b) (c) (d)

Fig. 5. Sample images captured from the Raspberry Pi camera during image acquisition



(a) (b) (c)

Fig. 6 Simulation tracks

5.2 Comparison of SVM and MLP

5.2.1 Evaluation Metrics*:* The performance of the algorithms is measured by the various parameters such as precision, recall and f1-score. Precision is the ratio of the correctly predicted positive observations to the total predicted positive observations.

$$Precision = \frac{TP_c}{TP_c+FP_c} \quad (5.1)$$

where c is the class label, TP is considered true positive and FP is considered the false positive sample. Recall is the ratio of correctly predicted positive observations to all the observations in the actual class.

$$Recall = \frac{TP_c}{TP_c+FN_c} \quad (5.2)$$

where c is the class label and FN, the false negative sample. f1-score is another metric which is the weighted average of precision and recall. Usually, it is much more useful than normal accuracy measures where the ratio of correctly predicted observations to the total observations are taken into account. Micro-average will aggregate the contributions of all classes to compute the average metric. A confusion matrix is also used to determine the correctly predicted samples of a particular class for a particular model.

$$Micro - Average = \frac{\sum TP_c}{\sum TP_c+\sum FN_c} \quad (5.3)$$

5.2.2 Performance of Multi-layered Perceptron on test dataset

The hidden layer size can be changed but we give it a value of around 60 for our model. The learning rate (α) is initialized to be 0.1 for training purposes. The values of learning rate and the hidden layer size should be chosen carefully to prevent the network from overfitting/underfitting the data. The network will underfit for large values of lambda and small values of the hidden layer size. Similarly, lower values of lambda and large values of the hidden layer size cause the network to overfit the data and will start incorporating noise into the model. We have experimented with various values of lambda and found that values between 0 and 0.1 are adequate in most cases. The rule of the thumb for the number of nodes in the hidden layer is that they are usually between the size of the input layer and the size of the output layer. The multi-layered perceptron is the most straightforward way for implementing neural networks. As stated earlier, the two parameters of learning rate and hidden layer size are tuned in order to optimize the training and distinguish the various classes for prediction. The experimental test set was subjected to the two different algorithms in MLP, i.e. BACKPROP and RPROP. The training was carried out for 200 iterations and the testing accuracy is shown in the Table II below.

Table 2
Testing accuracy of backprop and prop for multi-layered perception

		α = 0.01	**α = 0.05**	**α = 0.1**
Hidden layer =	BACKPROP	68.56%	65.31%	63.96%

30	RPROP	70.73%	72.36%	70.73%
Hidden layer = 50	BACKPROP	71.54%	70.82%	64.50%
	RPROP	74.80%	74.53%	69.92%
Hidden layer = 60	BACKPROP	73.98%	71.54%	68.56%
	RPROP	75.34%	73.98%	72.09%

The algorithms give almost equal accuracy when trained on an unbiased dataset with a slight improvement seen while using RPROP. The accuracy results are comparably better with hidden layer size set at 60 neurons and with learning rate set to 0.01. As the learning rate increases, the test accuracy takes a degradation. Also, the time taken for training using RPROP is significantly less as compared to BACKPROP. The MLP performs better with the Canny edge detection done on the images before giving as input to the network. Without Canny edge detection, the test accuracy drops down to 54% and is very poor. The performance metrics of the MLP were recorded for various set of parameters but the best possible accuracy was recorded with learning rate at 0.01 and hidden layer size set to 60. MLP did not perform well when as the images are quite overlapping and the model overfitted for noise. As we have seen, MLP does not provide good enough accuracy. The errors seen in the MLP prediction suffers due to ambiguous classification of image samples. In Table 3, we see the confusion matrix as observed from the MLP model which give us the number of correctly predicted samples from each class. In this Table 4, the direction written vertically in the first column are the actual classes while the direction written horizontally is the predicted classes.

Table 3

Performance of MLP (learning rate = 0.01 hidden layer size = 60)

	Precision	**Recall**	**F1-score**	**Micro Average**
Forward	0.69	0.71	0.70	0.73
Left	0.72	0.77	0.74	
Right	0.69	0.62	0.65	
Idle	0.76	0.75	0.76	

Table 4

Confusion matrix of multilayered perceptron

	Forward	**Left**	**Right**	**Idle**
Forward	75	6	7	17
Left	5	58	11	1
Right	11	16	54	6
Idle	18	1	6	77

5.2.3 Performance of Support Vector Machine on test dataset

The SVM is a more versatile option for image classification and regression tasks. However, we are only interested in the classification part for now. Machine learning algorithms such as SVM have parameters that can be fine-tuned to control the learning process of the classification. Parameter C is to set the amount of regularization which controls the trade-off between the errors on training data and the model complexity. A small value for C will generate a simple model with more training errors, while a large value will lead to a complicated model with fewer errors. Kernel is to introduce different non linearities into the SVM model by applying kernel functions on the input data. Gamma defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. coef0 is an independent parameter used in sigmoid and polynomial kernel function. [22]. SVM with properly tuned parameters even outperforms convolutional neural networks in terms of computation cost and performance [22]. The parameters for tuning the SVM model are useful in improving the performance. The c parameter is set as 1.0 whereas we have chosen the Radial Basis Function (RBF) kernel. Gamma value is usually taken as the inverse of number of samples, where for this analysis is set to ‘auto’. SVM parameters tuning are also observed along with their score for each tuned parameter as seen in Table V. The graph given below in Fig. 6 is the tuning of the various parameters of SVM classification. The parameter c is varied from a range 1 to 50, whereas the kernels chosen are Radial Basis function and Polynomial. The gamma values are also chosen using scikit-learn and given values, ‘auto’ and ‘scale’. The coefficient ‘coef0’ is significant only for polynomial and hence varied between 0 and 1. The analysis is shown below in Fig. 7. As seen from the graph, the RBF kernel with Gamma set to

Auto and parameter c set to 10 gives us the best score. The use of RBF kernel greatly improves the performance of the classification and better class-wise prediction can be seen in Table 5, where the micro-average is observed at 93% accuracy.

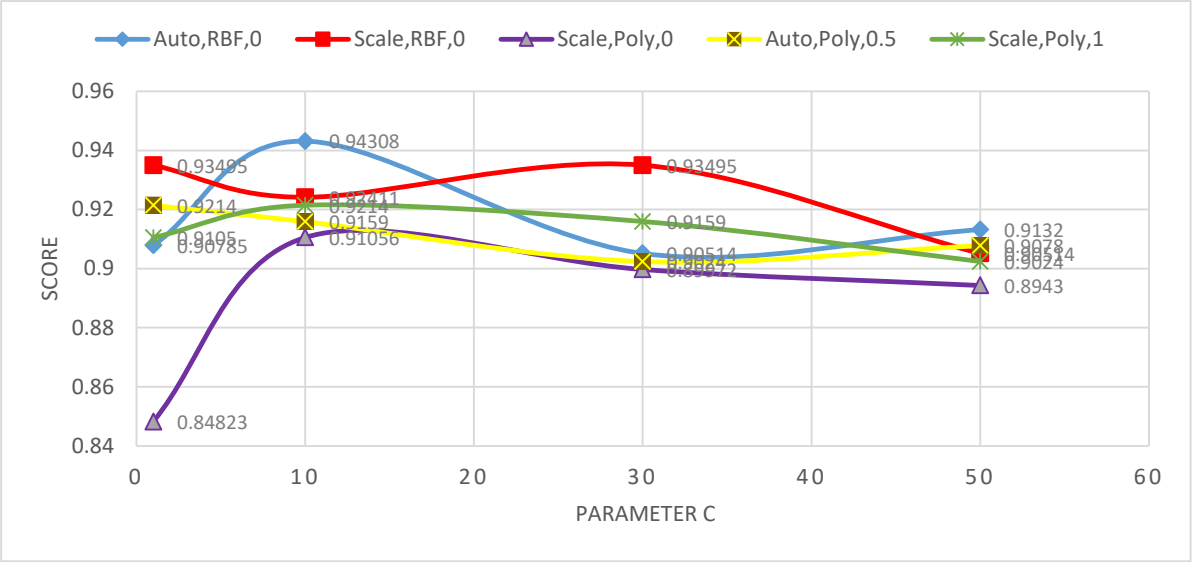


Fig. 7 Graph of SVM parameter tuning with performance score

For SVM, we observed that the model performs better when Canny edge detection is not done on the images before fitting the model and we can achieve accuracy far more superior than Multi-layered Perceptron models. SVM is very useful for multi-class classification and performs better than most neural network projects which require high-end computational power. In Table 6, the confusion matrix as observed with the tuning parameters give us the number of correctly predicted samples from the whole set of images. In this table, the direction written vertically in the first column are the actual classes while the direction written horizontally is the predicted classes. As we can see, the accuracy has greatly increased using SVM. There are some ambiguities which still exist in the track especially for Forward along with Idle classification. Idle class is used on images when the wheeled agent goes off the track or comes up against any obstacle and will stop the vehicle.

Table 5

Performance of Support Vector Machine (kernel = rbf , gamma = auto , c = 10)

	Precision	**Recall**	**F1-score**	**Micro Average**
Forward	0.92	0.88	0.90	0.93
Left	0.99	0.96	0.97	
Right	0.94	0.94	0.97	
Idle	0.88	0.90	0.89	

Table 6

Confusion matrix of Support Vector Machine

	Forward	**Left**	**Right**	**Idle**
Forward	82	0	5	6
Left	1	74	2	5
Right	4	0	88	5
Idle	17	0	8	72

The observations recorded for the two methods of MLP and SVM shows that SVM outperforms the MLP for this dataset and accordingly the task at hand. The improvement in accuracy in accuracy dwells from the fact that the RBF kernel is used to decide the hyperplane for classification. Also, the training time as observed for SVM is greatly minimized as compared to MLP. SVM is computationally faster and hence speeds up the process of fitting the data. Fig.8 shows the output of the python code which is run on the Raspberry Pi during training, each steering command given through the keyboard to the vehicle is noted by the program and is saved along with the JPEG file captured by the Pi camera as explained above. The increase and reduction in speed is also noted and saved. This data will later help the vehicle run autonomously through the track. The training images should be able to successfully identify the trajectory that has to be followed by predicting the direction in which the wheeled agent will move and reduce the number of misclassifications.

VI. CONCLUSION

The training of this wheeled agent in this simulated environment will demonstrate the ability of the wheeled agent's autonomy by navigating itself without any human control. The comparison of the two methods, MLP and SVM and their parameters have been observed and found that SVM performs better by having far better testing accuracy and subsequently will perform better on unseen and untested data, in this case self-driving in a new environment. Also, the wheeled agent should be able to adjust its speed accordingly during curved paths so as to avoid going off the track. With proper training and testing, we aim on achieving proper navigation and driving which will make the wheeled agent capable for self-driving.

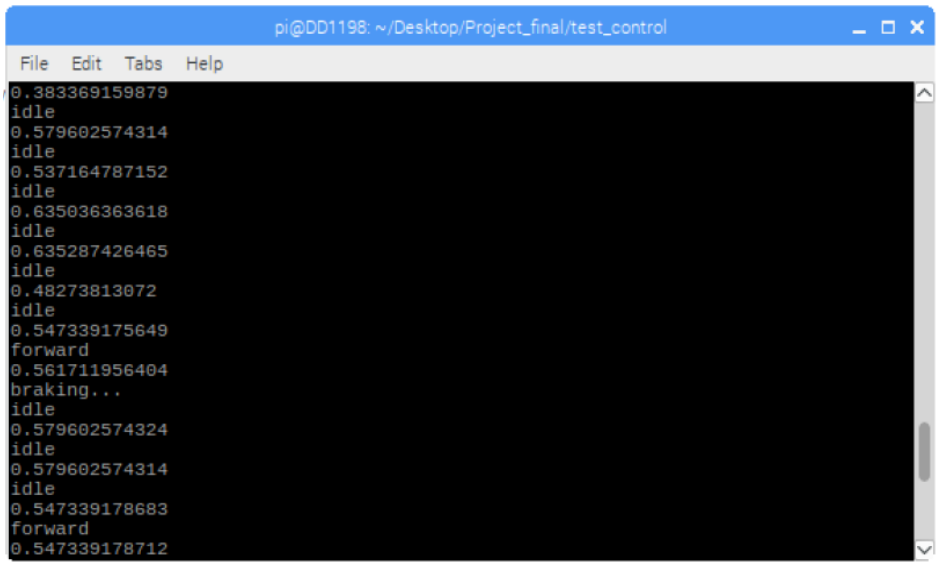


Fig. 8. Predicted command being generated during the autonomous mode.

REFERENCES

[1] Truong, Quoc-Bao, and Byung-Ryong Lee. "New lane detection algorithm for autonomous vehicles using computer vision." Control, Automation and Systems, 2008. ICCAS 2008. International Confer- ence on. IEEE, 2008.

[2] Bojarski, Mariusz, et al. "End to end learning for self-driving cars." arXiv preprint arXiv:1604.07316 (2016).

[3] Konstantinos Charalampous, Ioannis Kostavelis, Antonios Gasteratos, "Thorough robot navigation based on SVM local planning", Robotics and Autonomous Systems, Volume 70, August 2015, Pages 166-180.

[4] Ikhankar, Rupali, et al. "Pibot: The raspberry pi controlled multi- environment robot for surveillance and live streaming." Industrial Instrumentation and Control (ICIC), 2015 International Conference on. IEEE, 2015.

[5] Ionescu, Valeriu Manuel, Florin Smaranda, and Adrian-Viorel Dia- conu. "Control system for video advertising based on Raspberry Pi." Networking in Education and Research, 2013 RoEduNet International Conference 12th Edition. IEEE, 2013.

[6] Fu, Li-Chen, and Cheng-Yi Liu. "Computer vision based object detection and recognition for vehicle driving." Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on. Vol. 3. IEEE, 2001.

[7] Jiangwei, Chu, et al. "Study on method of detecting preceding vehicle based on monocular camera." Intelligent Vehicles Symposium, 2004 IEEE. IEEE, 2004.

[8] Memon, Qudsia, et al. "Self-driving and driver relaxing vehicle." Robotics and Artificial Intelligence (ICRAI), 2016 2nd International Conference on. IEEE, 2016.

[9] Loureiro, Pedro FQ, Rosaldo JF Rossetti, and Rodrigo AM Braga. "Video processing techniques for traffic information acquisition using uncontrolled video streams." Intelligent Transportation Systems, 2009. ITSC'09. 12th International IEEE Conference on. IEEE, 2009.

[10] Nugraha, BrilianTafjira, and Shun-Feng Su. "Towards self-driving car using convolutional neural network and road lane detector." Automa- tion, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT), 2017 2nd International Conference on. IEEE, 2017.

[11] Pannu, Gurjashan Singh, Mohammad Dawud Ansari, and Pritha Gupta. "Design and implementation of autonomous car using Raspberry Pi." International Journal of Computer Applications 113.9 (2015).

[12] Budiharto, Widodo. ”Intelligent surveillance robot with obstacle avoidance capabilities using neural network.” Computational intelligence and neuroscience 2015 (2015): 52.

[13] Le, Quoc V., et al. ”On optimization methods for deep learning.” Proceedings of the 28th International Conference on International Conference on Machine Learning. Omnipress, 2011.

[14] Pomerleau, Dean A. ”Alvinn: An autonomous land vehicle in a neural network.” Advances in neural information processing systems. 1989.

[15] https://picamera.readthedocs.io/e/n/release1.13/recipes1.html recording-to-a-network-stream - Picamera Docs

[16] https://opencvpythontutroals.readthedocs.io/en/latest/py tutorials/py tutorials.html - OpenCV Docs

[17] Hayward, David. ”Raspberry Pi operating systems: 5 reviewed and rated.

[18] Prasad, Navneel, Rajeshni Singh, and Sunil Pranit Lal. "Comparison of back propagation and resilient propagation algorithm for spam classification." Computational Intelligence, Modelling and Simulation (CIMSim), 2013 Fifth International Conference on. IEEE, 2013.

[19] Schraudolph, Nicol N., Jin Yu, and Simon Günter. "A stochastic quasi-Newton method for online convex optimization." *Artificial Intelligence and Statistics*. 2007.

[20] Vakkalanka, S., et al. "Combining multiple evidence for video classification." *Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, 2005.*. IEEE, 2005.

[21] Vapnik, Vladimir. "Statistical Learning Theory. John Wiley&Sons." *Inc., New York* (1998).

[22] Lin, Yuanqing, et al. "Large-scale image classification: fast feature extraction and svm training." (2011): 1689-1696.

[23] Fu, Wei, and Tim Menzies. "Easy over hard: A case study on deep learning." *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017.

[24] Su, Kuo-Ho, Feng-Li Lian, and Chan-Yun Yang. "Navigation design with SVM path planning and fuzzy-based path tracking for wheeled agent." *2012 International conference on Fuzzy Theory and Its Applications (iFUZZY2012)*. IEEE, 2012.

[25] Shenghong He, Rui Zhang, Qihong Wang, Yang Chen, Tingyan Yang, Zhenghui Feng, Yuandong Zhang, Ming Shao*, and Yuanqing Li, “A P300-based Threshold-free Brain Switch and Its Application in Wheelchair Control*”, *IEEE TRANSACTIONS ON NEURAL SYSTEMS AND REHABILITATION ENGINEERING* 1.