Contributors : Donipolo Ghimire, Minwon Seo

Project objective

-We propose to classify simple Human Hand Gestures (Go, Right , Left) using Ultra-Wide Band(UWB) sensor data.

- This project integrates machine learning techniques for potential robotic applications. Since the homework problems covered 2 layer neural networks and two classification problems, we made our problem slightly more complicated by introducing three class classification, and more hidden layers. Since both of us are researching robotic applications, we have implemented the techniques to move a simple Turtlebot.

- ROS is a very powerful tool in the field of robotics. We did not have prior experience handling ROS before this project and it has been a good learning experience.

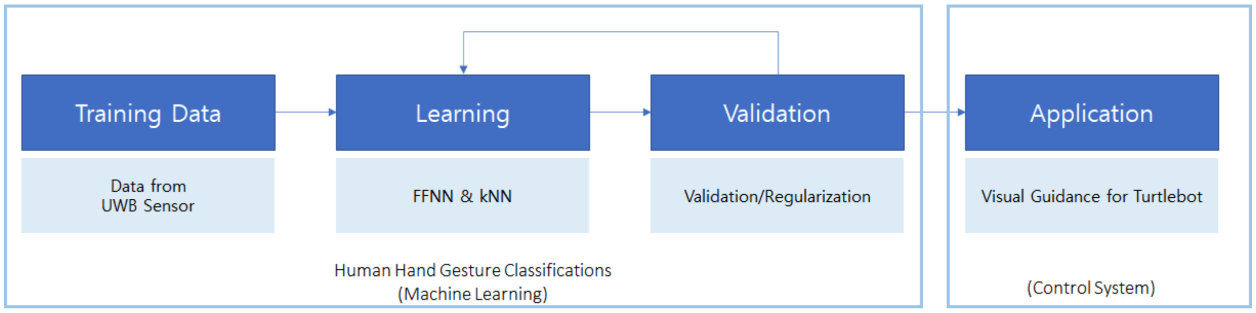
Training Data
Data from UWB Sensor
Learning
FFNN & kNN
Validation
Validation/Regularization
Application
Visual Guidance for Turtlebot
Human Hand Gesture Classifications (Machine Learning)
(Control System)

Fig 1: The framework of project for Learning Control Systems

Source of Data

- UWB Sensor data is used to classify the different hand gestures.

- The UWB relative distance measurement is based on time-of-arrival (TOA) algorithms, which measures the propagation time of an impulse that travels from transmitter to the receiver. However, there are noises in the data as bellows

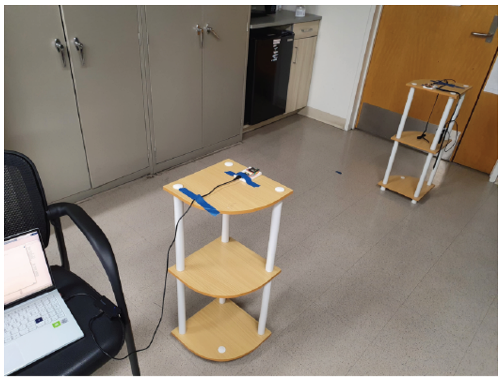
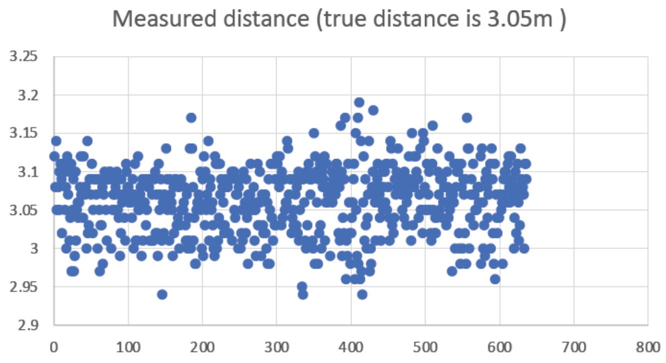

Fig 2: UWB range measurement and distribution(noisy data)

- **The data will be collected** using the UWB sensors(10 Hz) for 30 sec in the KCS laboratory using hand gestures of two different individuals. The UWB measurement for each gesture will be collected 4 times for each individual.

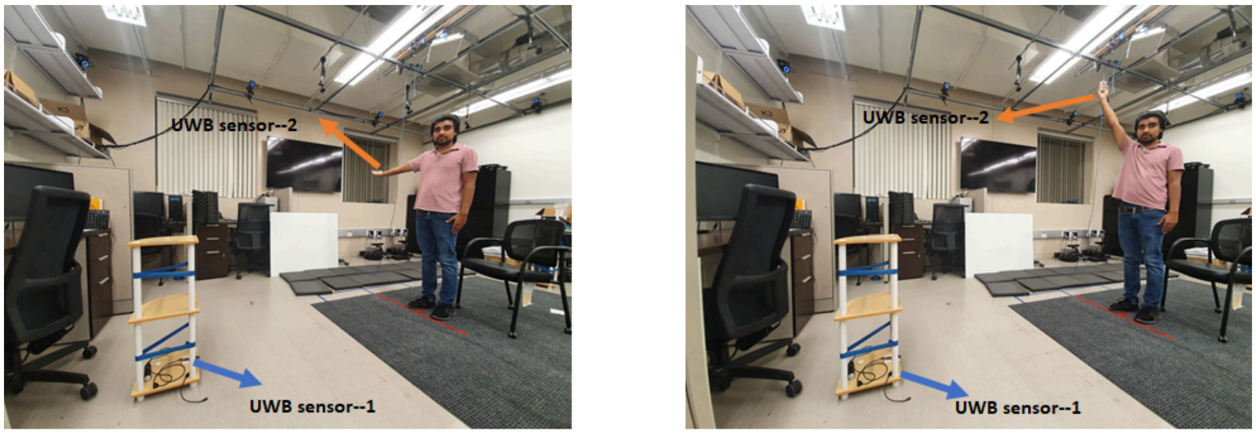

Fig 3: Demonstration about how we collected the data from the sensors

	Forward	Right	Up	Down
Individual 1	4 trials (1200 Data Set)	4 trials (1200 Data Set)	4 trials (1200 Data Set)	4 trials (1200 Data Set)
Individual 2	4 trials (1200 Data Set)	4 trials (1200 Data Set)	4 trials (1200 Data Set)	4 trials (1200 Data Set)

Total : 32 trials
(9600 Data set)

Machine learning algorithm(s) ,Used

1. Feed Forward Neural Network(FFNN): Parametric method

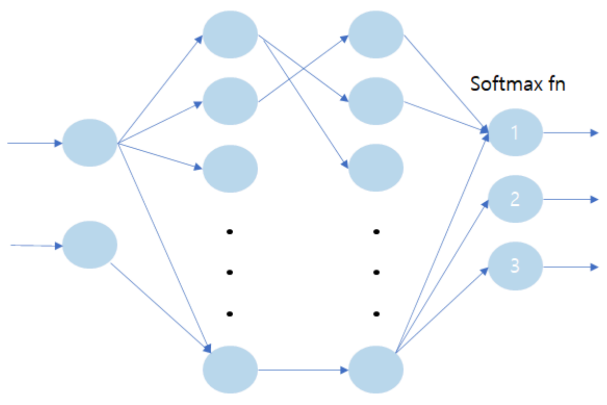

Fig 4 : The framework of human guidance command feed forward neural network

Decision metric for using FFNN:

Table 1: Trial and selected values for feed forward neural network

Parameters	Trial Values	Selected Values
Number of input dimension	2 ~ 4 (Range, Power, FPPL, RSS)	2 (Range, Power)
Number of output classification	3, 4	3
Number of hidden layers	1, 2, 3 layers	2 layers
Number of node per layers	3 ~ 15	6
Loss	MSE	MSE
Learning rate, α	0.01 / 0.001 / 0.0001	0.001
Optimizer	Ada, Adagrad, RMSProp	RMSProp
Regularization	ϱ = [0, 0.1, 0.01, 0.001, 0.0001]	ϱ = 0.01
Validation Data	20%	20%
Activation function (hidden layers)	Bipolar Sigmoid ($\boldsymbol{\lambda}$ = 1 ~ 4)	Bipolar Sigmoid ($\boldsymbol{\lambda}$ = 3.5)
Activation function (output layers)	Softmax	Softmax

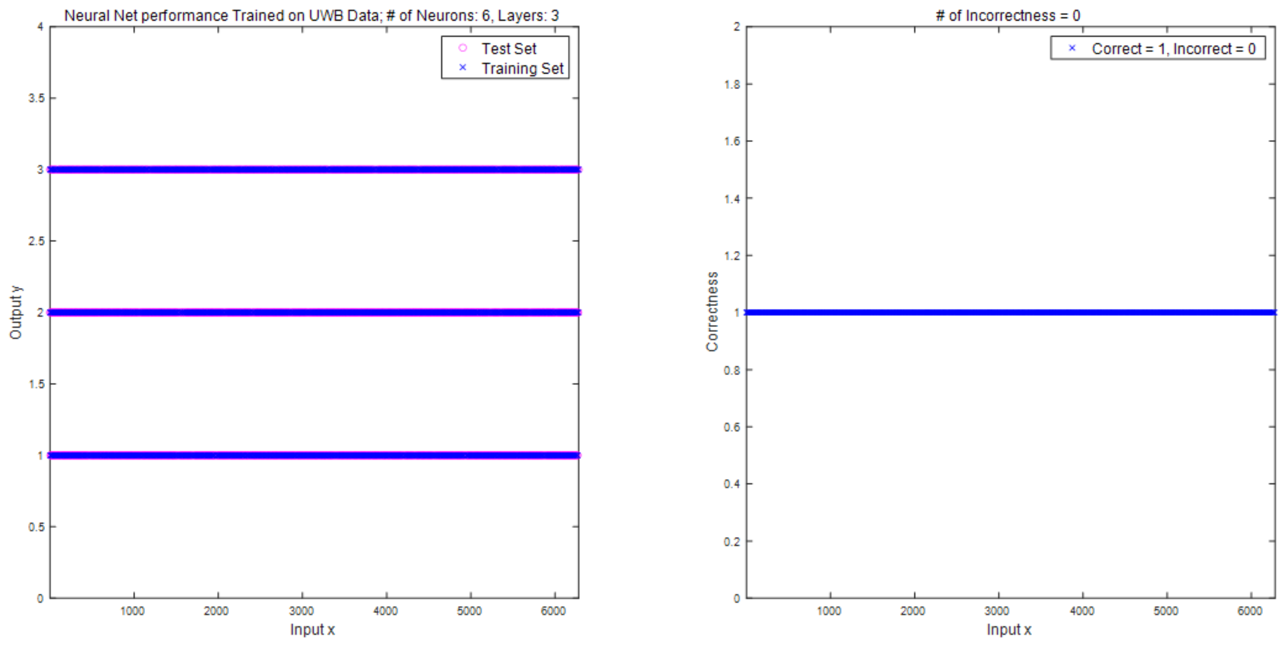

Fig 5: (Left) Neural Network Performance (3 Classifications) trained on UWB Data (6 Nodes, **2 Hidden layers**, Optimizer: RMSProp)
(Right) The correctness of the test data set (Correct = 1, In-correct = 0). There is no in-correctness data

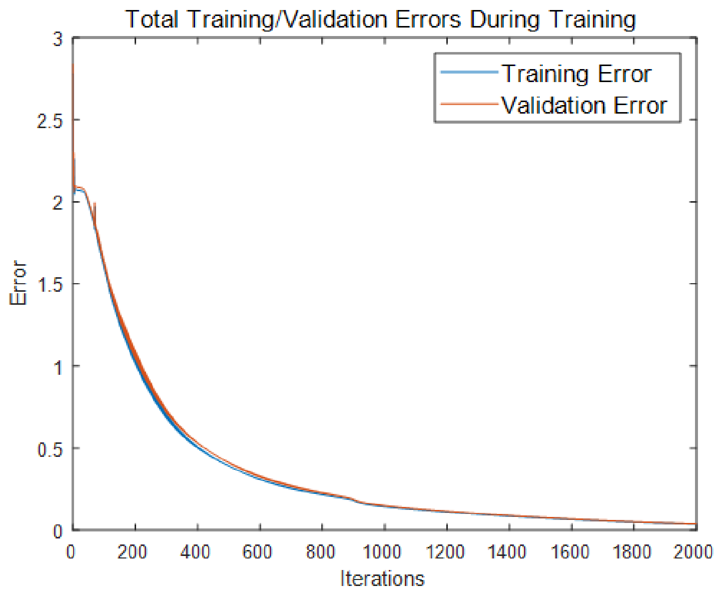

Fig 6: Total Training & Validation Errors during Training based on UWB data

** Note: The fnn2_2016 and bp2ma2018 provided by Professor Sideris for Homework 2 were modified and more hidden layer (2 and 3) were added [3].

Remark 1:

The data we collected from the UWB sensor has 4 dimensions such as relative range, power signal, and FPPL, RSS. However, FPPL and RSS (Received Signal Strength) are independent from the UWB position. Therefore, we only use relative range and power signals as input data. If we had a different setting, let's say we wanted to classify line of sight(LoS) and non line of sight data, then FPPL and RSS would be considered good parameters for classification of localization problems.

Remark 2:

We tried to change hidden layers from 1 to 3. In the case of 1 hidden layer, even after we tested the algorithm by increasing the number of nodes per layer, we did not observe any improvement in our results for the classification problem as shown in Fig 7 .We also realized that the convergence rate for 1 hidden layer is very slow. For 3 hidden layers, the datas, were not classified properly as demonstrated in Fig 8 . We came to the conclusion that we did not have enough data set to train the 3 hidden layers, and as you can see in Fig 8, the validation error starts to increase after a few iterations. Increment in validation error signifies that the learning algorithm should be stopped.

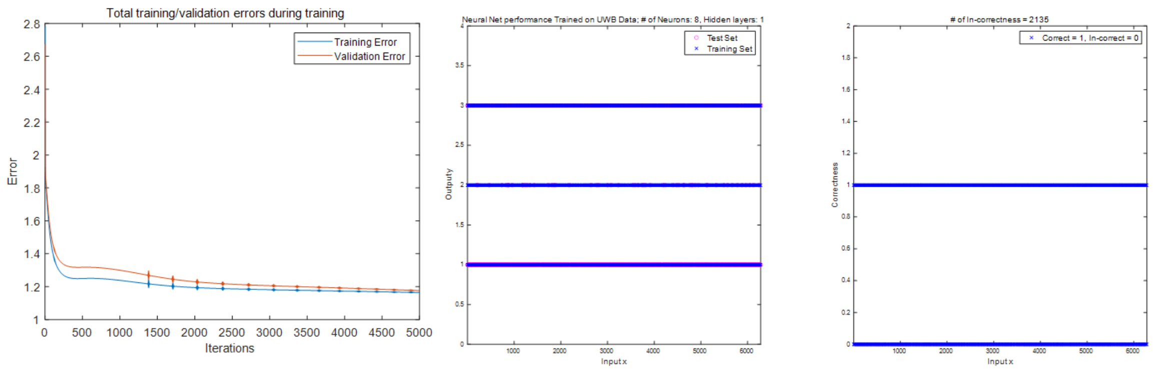

Fig 7: Neural Network Performance trained on UWB Data (6 Nodes, **1 Hidden layers**)

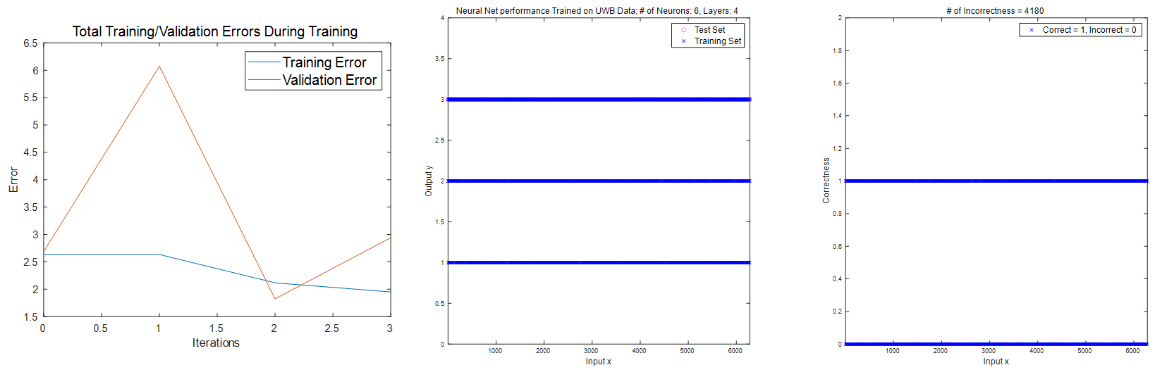

Fig 8: Neural Network Performance trained on UWB Data (6 Nodes, **3 Hidden layers**)

Remark 3:

We also tried to change optimizers such as Ada, Adagrad, and RMSProp. In the case of Ada, the training error was not convergent as shown in Fig 9. Even if the training and validation error of Adagrad has decreased, the correctness is not good as shown in Fig 10(right-most subplot). However, the RMSProp optimizer that we used in Fig 5, classifies the problem with better accuracy, i.e zero misclassified data. Although RMSProp is similar to the Adagrad in concept, it creates an exponentially weighted version of the accumulated squares of the gradient components and thus, is considered a suitable optimizer compared to Adagrad [3].

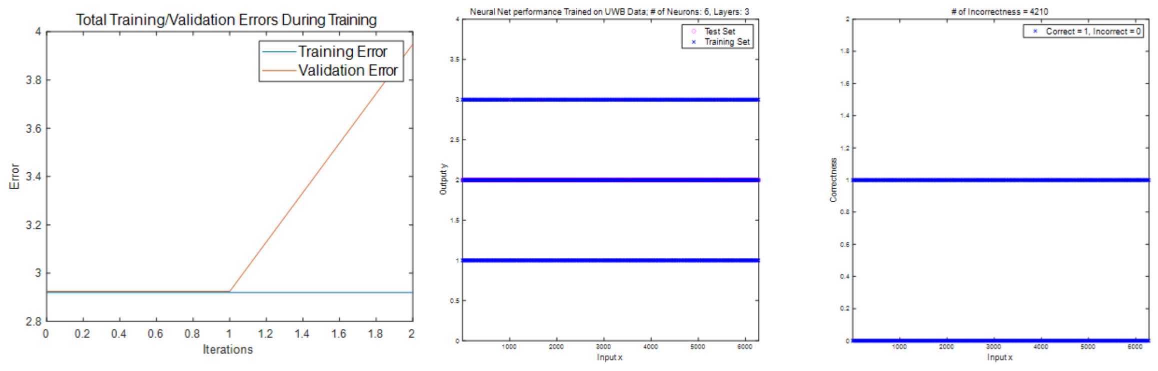

Fig 9: Neural Network Performance trained on UWB Data (**Optimizer: Ada**)

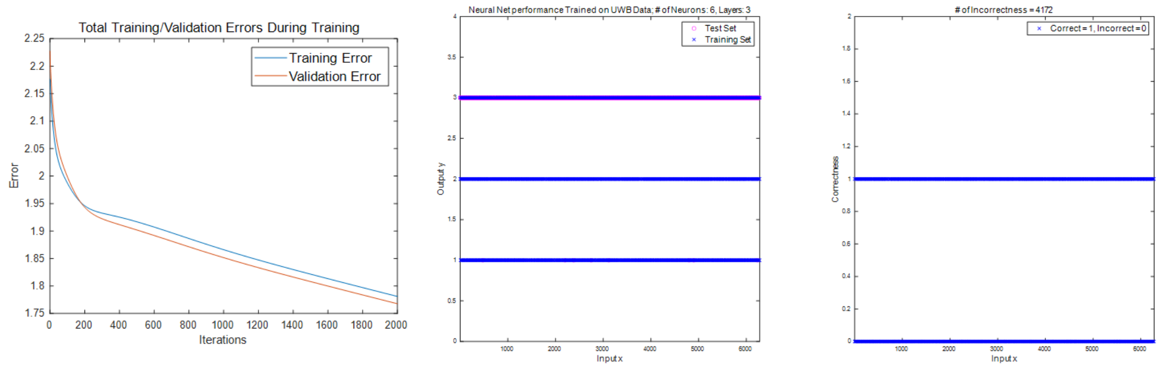

Fig 10: Neural Network Performance trained on UWB Data (**Optimizer: Adagrad**)

Remark 4:

In our classification problem, since $\boldsymbol{\lambda}$, the parameter of hidden activation function(bipolar), has an effect on gradient in SD algorithm, this in turn impacts the convergence of our training data. In addition, since our sensor data has a lot of noise, we need to achieve convergence through trial and error to find proper hyper-parameters.

Test Remark:

We modified the backpropagation algorithm code for 2 and 3 hidden layers. In addition, before handing the UWB data set, we wanted to check the softmax function for multi-classification and the effectiveness of parameter tuning. We selected fisheriris data that is multi-classification data in the MATLAB library. Finally, we can check the FFNN result as shown in fig 11. In this problem, we choose 12 nodes and 2 hidden layers. The result of correctness is just one miss-classification

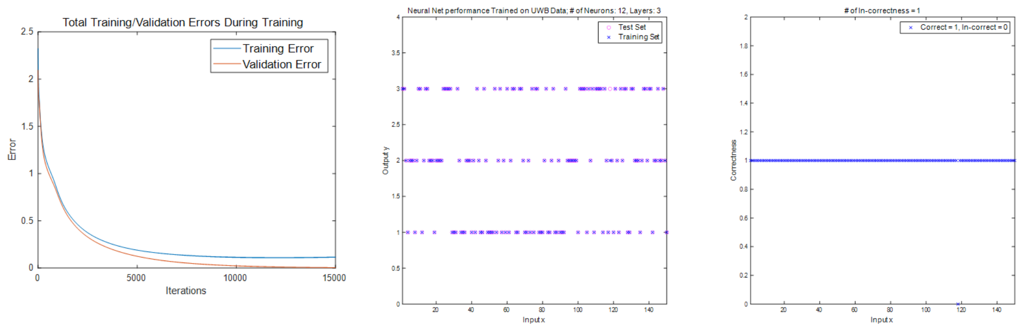

Fig 11 : Neural Network Performance (3 Classifications) trained on Fisheriris Data (12 Nodes, **2 Hidden layers**) and the correctness of the test data set (Correct = 1, In-correct = 0). There is only one in-correctness data

Remarks about encountered Errors while classifying with FFNN

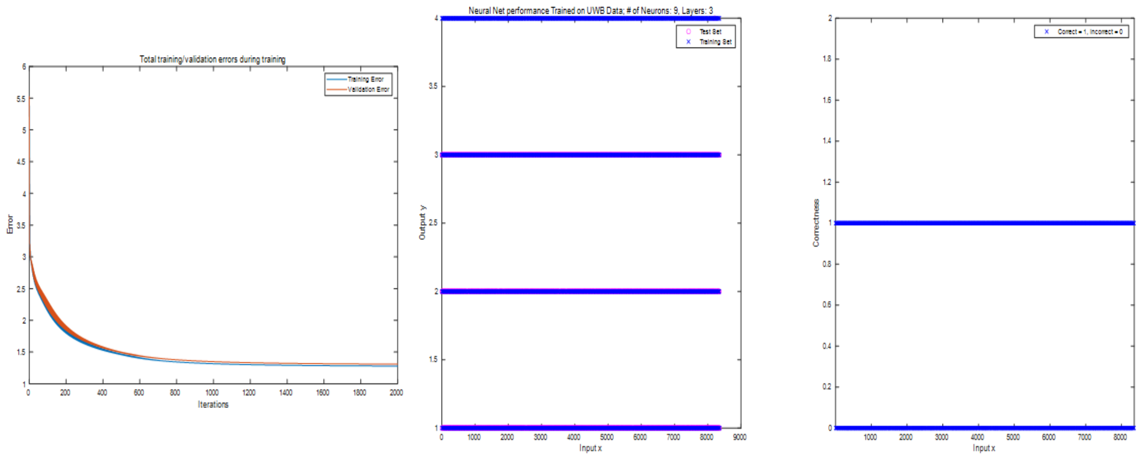

Fig 12: Neural Network Performance (4 Classifications) trained on UWB sensors

Our initial problem promised a 4 classification FFNN algorithm. However, we were not able to classify the initial problem as shown in figure 12. We have examined that our input is two dimensional and since our output is four dimensional (for softmax, we had to increase the dimensionality), the FFNN is not able to classify the problem. In Fig 12 (right-most subplot), we can see that a lot of data have been misclassified.

2. K Nearest-Neighbor (kNN): Non-parametric method

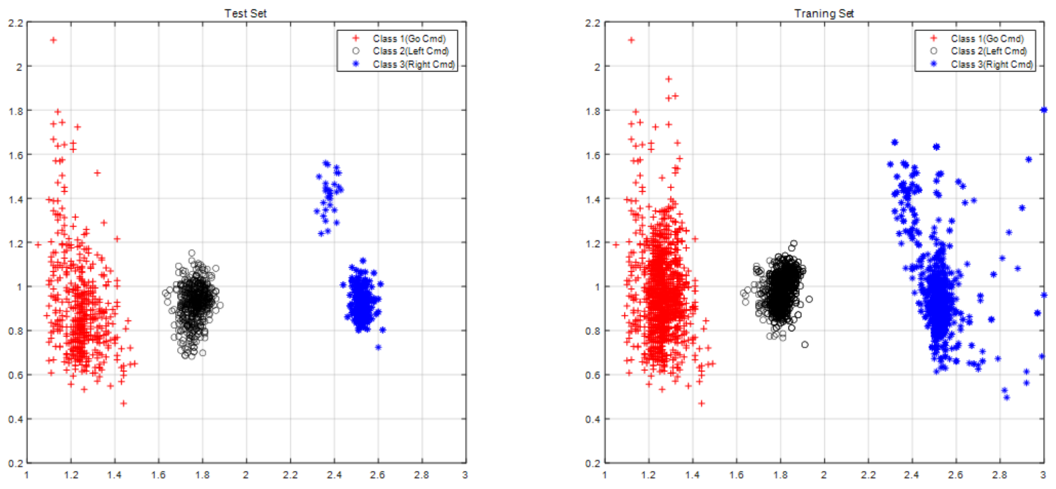

Fig 13: Testing Set and Training Set of the data set

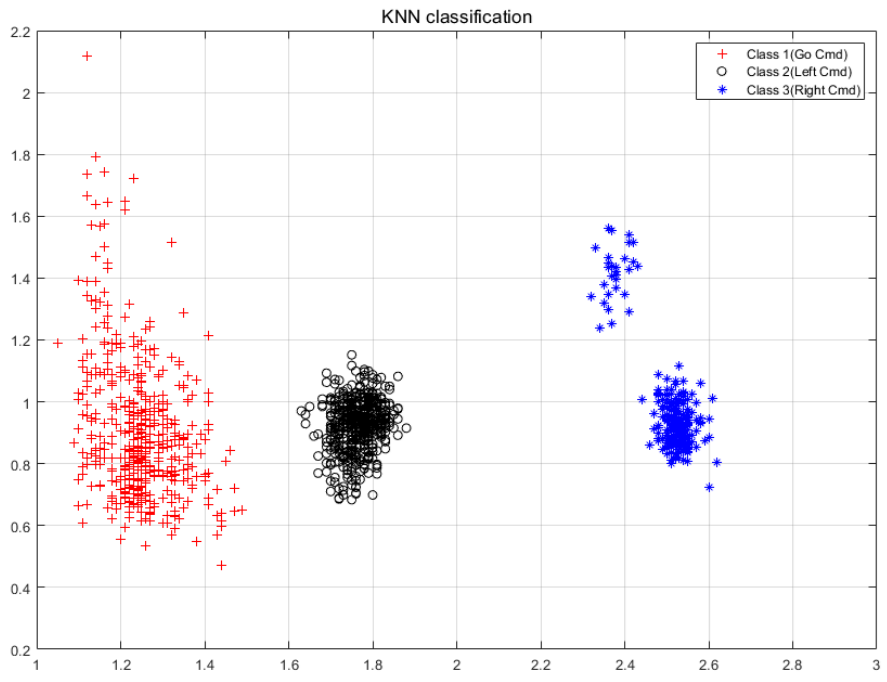

Fig 14: KNN classification for new data

Decision metric for using KNN:

Table 2 : Trial and selected values for K-NN

Parameters	Trial Values	Selected Values
Number of input dimension	2 ~ 4 (Range, Power, FPPL, RSS)	2 (Range, Power)
Number of output classification	3, 4	3
K (Number of Nearest Neighbors)	5,7	5

The K-NN algorithm is non parametric and has no specific parameters to tune like FFNN shown in Table 1. In K-NN we simply find the nearest samples and assign a class with the most samples among the K samples. However, because of this simplicity, the computation complexity significantly increases. The complexity scales as `O(N*d+NlogK)[3]`.

Since, we already worked on a parametric learning algorithm, like FFNN. And since we used 2 hidden layers, it can be called a Deep Neural Network (DNN) Algorithm. We wanted to use a non- parametric method, and compare and analyze these two methods. We used different values for K, and evaluated that **5** and **7** were the numbers that provided the best estimation for classification. This speculation was purely based on trial and error.

Remarks about encountered Errors while classifying with KNN:

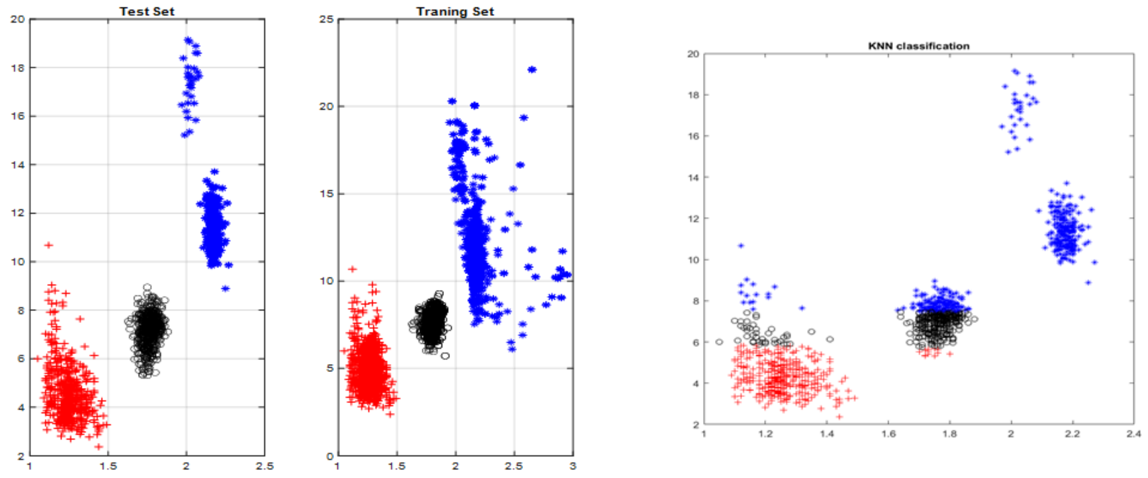

Fig 15: K-NN classification before normalizing the y axis to get a better fit.

Error Remark 1:

In our initial trials, we were not able to classify the data we collected from the UWB sensors properly . We use euclidean distance or norm, to classify the data, however the scale of the x2(y-axis) is significantly larger than the x1(x-axis) as shown in above Fig 15. Thus, we *normalized* the y-coordinates(x2, Power signal) and were able to classify the data correctly as shown in Fig 14.

Error Remark 2:

Similarly, since it we are solving a three classification problem, we experienced a lot of misclassification, because number of neighbor for some classification would be the same, even though we used an odd number for our parameter K. For Example, k = 15, if a, b, c are three numbers representing the number of neighbors for each class, we experience a = b , b=c, a= c, etc in some cases. Thus we ran the KNN algorithm again to classify the misclassified datas, and were able significantly improve the performance of our algorithm

Test Remark

We used the famous machine learning Iris dataset to test our algorithm, and check the efficacy of our algorithm for different kinds of data sets. Similarly this dataset was very useful to debug errors while we were formulating the algorithm as the dataset has lower dimensionality.

We observed that the KNN algorithm works very well and is able to classify the three species. We imported the data from the MATLAB libraries and classified the three different species in the dataset.

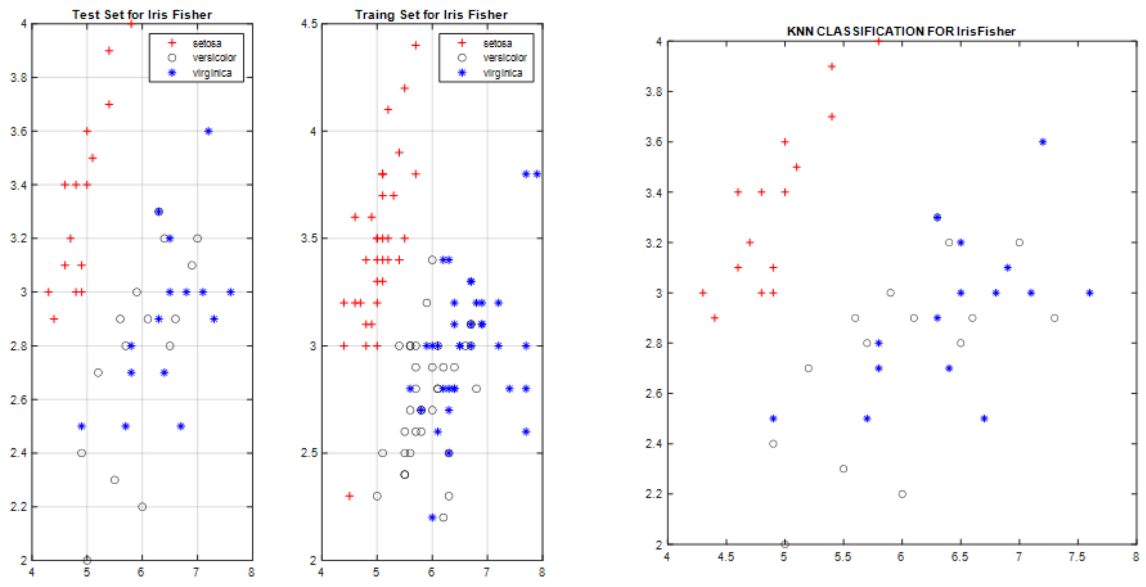

Fig 16: Example of KNN classification for Iris Fisher dataset

Simulation Results

- We compared the efficiency of Feed Forward Neural Network and kNN. In our problem settings, FFNN(Parametric) performed better compared to the kNN(Non-parametric) as the FFNN classified all the classes correctly.
- We collected new data where we changed our hand gestures to record certain instructions as shown in Fig 17. Go (Forward), turn left and move, turn right and move, Go (forward) and turn left. We will feed this data to the trained FFNN and kNN and classify the new data.

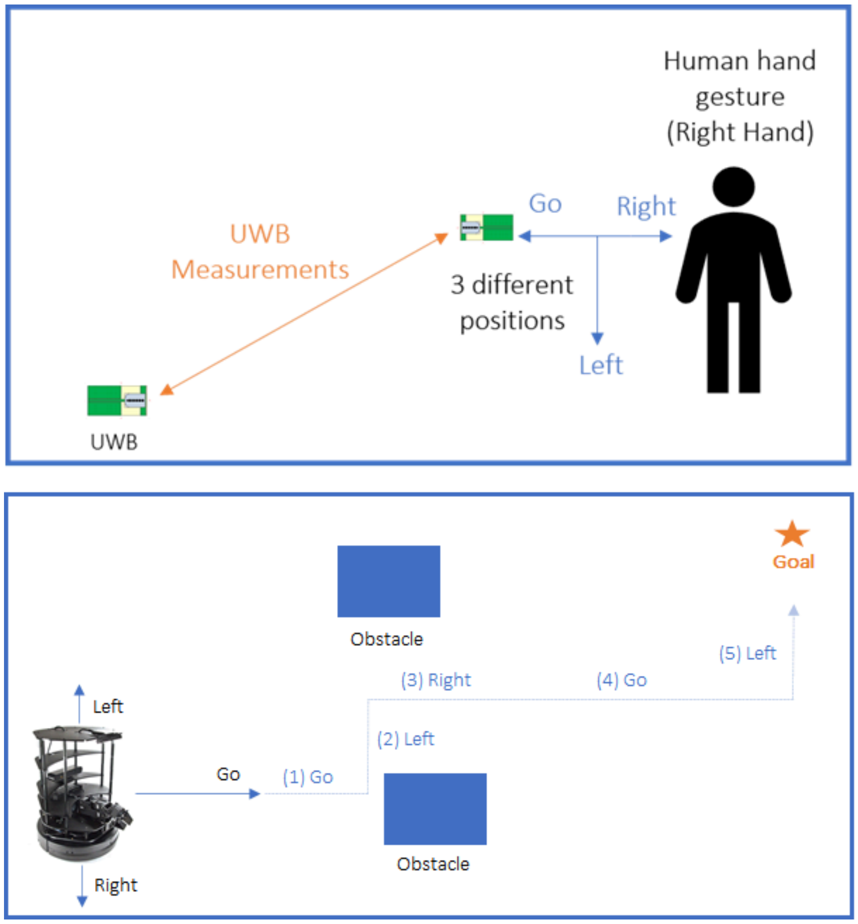

Fig 17: Human Guidance Command & Turtlebot Scenario

FFNN algorithm was able to classify the gestures accurately as shown in Fig 18, but KNN has some few misclassified data as shown in Fig 19. As the data set gets larger the computational complexity of kNN will significantly increase compared to FFNN. We performed some literature review on the comparison of kNN and FFNN for classification of hand gestures and other settings. Through papers like [1] and [2], we found that people have been using Artificial Neural Network (a type of FFNN) and kNN for classification problems. In these two papers ANN performs slightly better, however, we cannot generalize that one performs better than the other.

Our new data set that represents the commands for the turtlebot, consists of 1372 data points.

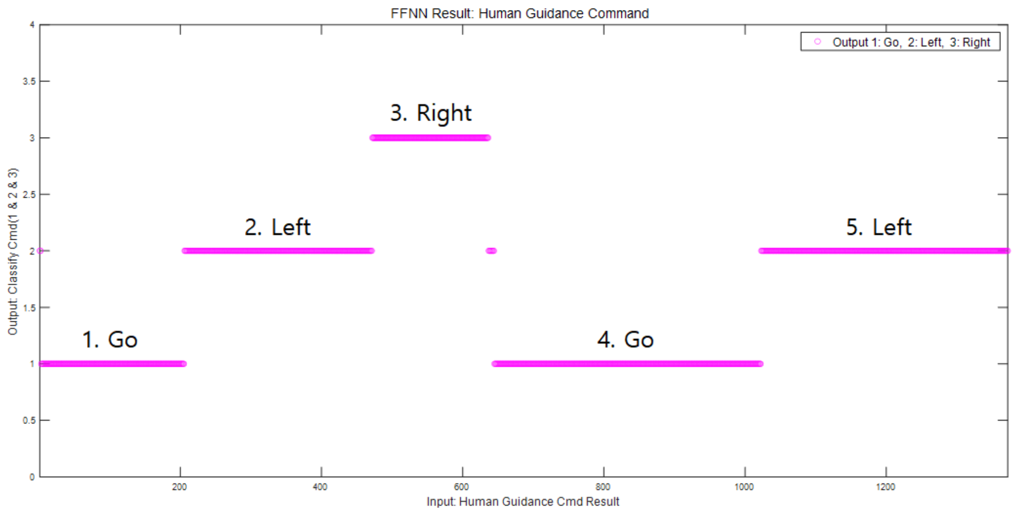

Fig 18: **FFNN Result** for Human Guidance Command (data set = 1372).

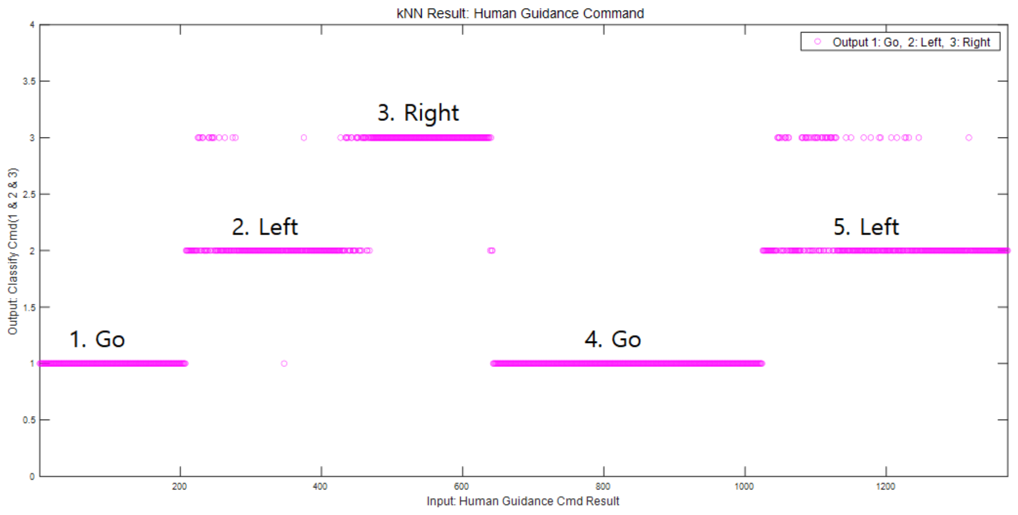

Fig 19: **kNN Result** for Human Guidance Command (data set = 1372).

We applied this classified data, to instruct the turtlebot to move in a certain direction. Figure 20, shows our experimental setup. The robot does not perform obstacle avoidance using sensors. These obstacles were placed according to the predescribed motion of the robot to make the visualization of the motion easier . After classifying the data, we used the data as the command for turtlebot. The turtlebot moved according to the pattern of hand gestures.

Forward -> Left turn and move -> right turn and move -> Forward -> Left turn
We have also attached a video demonstration of the turtlebot, in CANVAS. A GIF is also attached in this document.

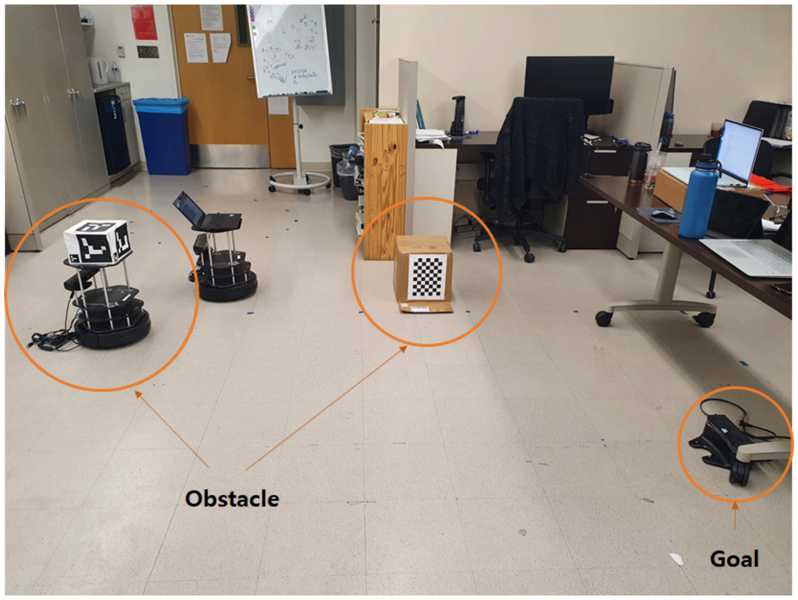

Fig 20: Human Guidance Command & Turtlebot Demonstration (Off-line process)

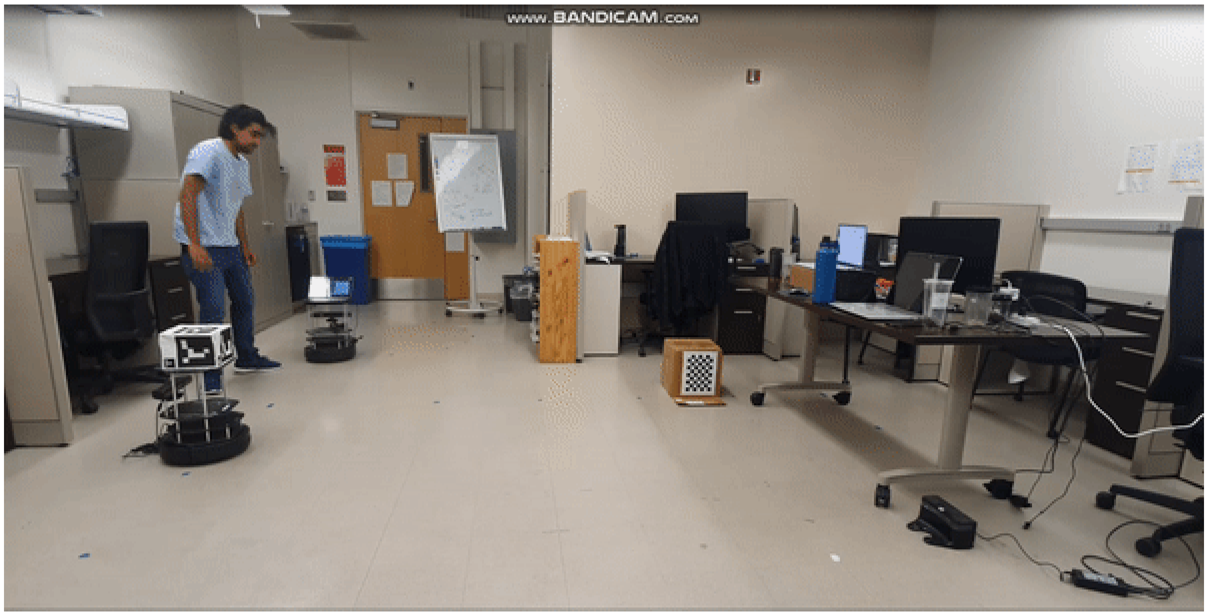

Figure 21: This is a generated GIF of the experiment.

☐ Note: A video is attached in CANVAS if the GIF doesn't load

AGENT : Turtlebot 2 Kuboki (KCS lab)
The platform used for automating the turtlebot was using Robots Operatings System

REFERENCES

[1] P. Trigueiros, F. Ribeiro and L. P. Reis, "A comparison of machine learning algorithms applied to hand gesture recognition," *7th Iberian Conference on Information Systems and Technologies (CISTI 2012)*, 2012, pp. 1-6.

[2] Nonlinear forecasting of stream flows using a chaotic approach and artificial neural networks, Earth Sci. Res. J. vol.17 no.2, 2013

[3] Lecture Notes and Codes from Professor A. Sideris, Learning Control Systems , SPRING 2021