Designing an Intelligent Pathfinder Drone with Deep Learning and Fuzzy Logic

Fariborz Mollaie
Master's degree in software engineering, Faculty of Engineering and Technology, Lorestan University, Khorramabad, Iran

Abdolmajid Mousavi·Affiliation
Assistant Professor in software engineering, Faculty of Engineering and Technology, Lorestan University, Khorramabad, Iran

Mohammad Bagher Dowlatshahi[1]
Associate Professor in software engineering, Faculty of Engineering and Technology, Lorestan University, Khorramabad, Iran

Abstract
Drones are flying robots with many structures that can fly unmanned and can composite with some degree of intelligence to help them make appropriate decisions when facing new situations. Also, with the progress of artificial intelligence (AI) in recent years, its application can be seen in most other branches. This paper presents the design of such an intelligent drone that is capable of doing proper maneuvering when it is flying over roads. specifically, we used a deep learning network to train it by a big dataset of images from nature and urban roads gathered by 3 cameras and we got help from fuzzy logic to gain the best angle and speed that keep the drone in the path. The goal is to improve the accuracy of learning by extending the dataset and the network compared to previous works and decreasing the complexity by fuzzy logic. We used this method on the hexacopter on the road outside the city at low altitude which had good results.

Keywords: UAV, Deep Learning, Fuzzy Logic, Neural Networks, Auto Flight, path finder, autonomous

1-Corresponding Author

Introduction

Recently, automatic locomotion of robots has been one of the main subjects for research and development by many automotive and military sectors. The idea is to place robots on roads by using the existing path patterns and estimating distance from surrounding obstacles via appropriate sensors. The challenge is to detect and extract patterns to train robots on paths such as unpaved forest roads where environmental patterns are not completely visible. Moreover, locomotion in natural environments is prone to much harsher complications than in man-made roads, especially for flying robots. on the other hand, with the success of approaches such as deep learners in the automatic extraction of attributes, this paper proposes a method based on deep learning to train a drone. Moreover, fuzzy logic is employed for robot locomotion to estimate the drone speed and angle.

According to previous research, deep learning paves the way for facility and accuracy to extract features. Moreover, researchers reap the benefits of deep learning to extend robot control. The unmanned aerial vehicle requires a precise estimation to achieve a successful flight. Do et al. (2018) address the problem of autonomous quadrotor navigation in indoor spaces. They use the area's visual map as a graph of linked images to determine visual paths for the quadrotor to follow. In autonomous drone racing, a Drone is required to fly through the gates quickly without any collision. Jung et al. (2018) introduced a convolutional neural network to estimate a gate's center to fly through the gates without collision. Achieving precise estimation is needed for the unmanned aerial vehicle to perform a successful flight with a high degree of stability. Al-Sharman et al. (2019) developed a deep learning-based framework to enhance the state estimator's performance. In the discussion of automating drones' movement, Giutsi et al. (2016) proposed an approach based on the neural network used to control a quadrotor to trail forest areas. Also, Ki Kim and Chen (2015) used a deep learning model to learn a control strategy that mimics an expert pilot's choice of action in corridor spaces.

PROPOSED SOLUTION

Our approach comprises three phases. In the first (dataset development), a great deal of dataset, including images taken from the environment, was gathered to insert the deep learning model. Secondly (Build and train deep neural network model), it is supposed to build a deep neural network model structure trained by the dataset. In the third (drone movement), an image from the environment is taken by a camera installed on the Drone to transfer to the neural network model. Additionally, the neural network model delivers an output to the fuzzy system to acquire an estimate of the speed and turn angle of the robot. Before describing this approach's details, let us take a look at the hardware and software employed in the research. The research uses a handmade hexacopter shown in Figure 1, including EMAX 3510 brushless motors and 50A speed controllers with a 6000 Mah battery. The Pixhawk flight controller, equipped with a GPS and ultrasonic sensor, is responsible for controlling the flight. The xu4 single-board computer process the information to be sent to the flight controller. The ov2710 camera takes photos from the environment and transfers them to the computer for processing.

Figure 1 The Drone used in this paper

The computer used in the training phase is equipped with an Nvidia GeForce 750ti graphics card and 12 G.B. of RAM, which is used to perform GPU learning operations. The Pixhawk flight controller mounted in this Drone utilizes the PX4 ROM and connects to the odroid-xu4 single-board computer through the Mavlink protocol. The Ubuntu 16.04 OS, the FlytOS package, and ROS Kinetic are installed on the odroid-xu4 to control the flight controller. The Keras Library with TensorFlow Backend has been used for Learning operations. All software and libraries use on odroid-xu4 are also installed on a more powerful computer.

A. Dataset Development

The research dataset consisted of the photos collected from the source dataset, which capture a film taken by three cameras installed on the human agent's head passing through forest areas. As shown in Figure 2, these cameras were positioned on the agent's head at specific angles and a human's height.

The photos taken by the left camera were put in the TL folder, and the photos taken by the right camera were stored in the TR folder. Also, the photos taken by the middle camera were saved in the TS folder. The collected photos were modified and refined due to specific problems such as inaccurate angles and obscurity. Finally, a dataset of 40,981 photos was created. The data set was expanded horizontally by adding rotated images to the right class, and the images in the right class were returned and placed in the left class. Similarly, middle-class photographs were rotated horizontally and placed in the same class. An example of this task is shown in Figures 3 and 4.

In total, the dataset consisted of 81,962 photos. The number of photos in each class is listed below:

- TR:28881

- TL:28881

- TS:24200

30% of photos were used for testing, whereas 70% of them were used for training.

Figure 2 How to install the cameras on the man's head

Figure 3. extending the dataset by copying the reversed left class pictures to the dataset

Figure 4 extending the dataset by copying the reversed center class pictures to the dataset

B. Build and train the deep neural network model

1. Building the model: The model built at this phase is a 16-layer deep neural network model, which is depicted very well in Figures 5 and 6. The first layer is a 16x16x3 input layer and is a layer for injecting photos. The next layer is a convolutional layer, which uses 4x4 filters. The third layer is the maxpooling layer, which reduces the network's complexity by reducing the photos' dimensions. In this layer, we have used 2 * 2 filters. Moreover, using a dropout layer help to prevent overfitting of the network. The end layers include the flatten, dense, and output layer, in which the output layer has three neurons that represent the three classes, left, right, and middle.

The elu function was used as the activation function in this model.

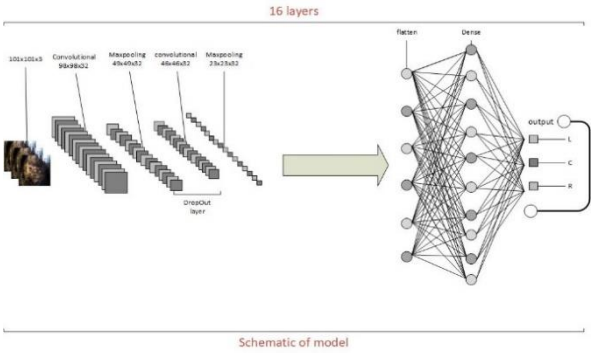

Figure 5 The presented model in this article

```
model=Sequential()
model.add(conv with elu activation function)
model.add(MaxPooling2D())
model.add(Dropout(rate))
model.add(conv with elu activation function)
model.add(MaxPooling2D())
model.add(Dropout(rate))
model.add(conv with elu activation function)
model.add(MaxPooling2D())
model.add(Dropout(rate))
model.add(conv with elu activation function)
model.add(MaxPooling2D())
model.add(Dropout(rate))
model.add(Flatten())
model.add(Dense(200))
model.add(Dense(3,activation='softmax'))
```

Figure 6 Model's layers.

2. Train the model: We want to teach the neural network model using the prepared dataset. As mention, the number of photos in the database was 81962, which generally have a large volume and dimensions, and cause a lot of overhead in the neural network model, so the dimensions of the photos are changed to 101*101*3, which results in the number of neurons in the first layer decreases dramatically. In this phase, we increase the dataset size by the data augmentation function (Figure 7 and 8) to 322560 photos, of which 2580488 photos are for training, and 64512 photos are for testing.

Figure 7 extending the dataset by data augmentation

```
datagen =ImageDataGenerator(
rotation_range=10,
rescale=0.1,
zoom_range=0.2,
width_shift_range=0.1,
height_shift_range=0.1,
fill_mode='constant',
horizontal_flip=False)
```

Figure 8 image data generator structure

Due to hardware limitations we face in the learning phase, it is impossible to load the entire dataset into the memory because the maximum system memory is 12 G.B. At the same time, the total size of files amounts nearly to seven G.B Since some of these files are summoned multiple times for the insertion of variables and modification of formats, they would require a minimum of 30 G.B. of RAM. Therefore, a better solution is using a custom generator function (the code is given in Figure 8) to put data into RAM in independent separate.

Adam and SGD optimizers tested this model. Finally, Adam was selected as the primary optimizer. There were 190 epochs and 32 batches. The number of existing photos was equal to the dataset size ratio to each part's batch size. All deep learning processes are performed on the more powerful system mentioned earlier. Eventually, the entire compiled model saves as an h5 file (Figure 9) and Then transfer to the single-board computer on the Drone.

```
#Custom Genator Function
class My_Custom_Generator(keras.utils.Sequence) :
def __init__(self, file_names, labels, batch_size) :
self.filenames = file_names
self.labels = labels
self.batch_size = batch_size
def __len__(self) :
return (np.ceil(len(self.filenames) /
float(self.batch_size))).astype(np.int)
def __getitem__(self, idx) :
batch_x = self.filenames
[idx * self.batch_size : (idx+1) * self.batch_size]
batch_y = self.labels
[idx * self.batch_size : (idx+1) * self.batch_size]
return np.array([
resize(imread('dataset_location'+
str(file_name)), (101, 101, 3))
for file_name in batch_x])/255.0, np.array(batch_y(
#import Data
filenames_counter = 0
labels_counter = -1
for subdir, dirs, files in
os.walk(train_dir):
for file in files:
filenames.append(file)
labels[filenames_counter, 0] = labels_counter
filenames_counter = filenames_counter + 1
labels_counter = labels_counter+1
#categorizing the photos
y_labels_one_hot = to_categorical(labels)
#shuffling the Data
filenames_shuffled,lablel_shuffled =
shuffle(filenames,y_labels_one_hot)
```

Figure 9 Custom Data Generator Function, import data and categorizing the data.

```
model .save ( " filelocation\filename.h5 " )
```

Figure 10 image data generator structure

C. Drone movement

After the learning phase, the learned model file transfer to the Drone's computer. Each time a picture was taken by the camera embedded in the Drone, infusion to the model to recognize the picture class. This model returns a decimal number that specifies the degree of dependence of the image in the class. In the direct routes, the Drone's speed increases, and according to the degree of dependence of the image on the class, the Drone moves at a shallow angle to the left or right or directly. In a turn of roads, the Drone's speed decreases, and it moves to the left or right at a greater angle. The fuzzy schematic shown in Figure 11 consists of 4 variables include the previous robot's speed (p velocity), model output(learning result), drone speed(velocity), and rotation(rotation).

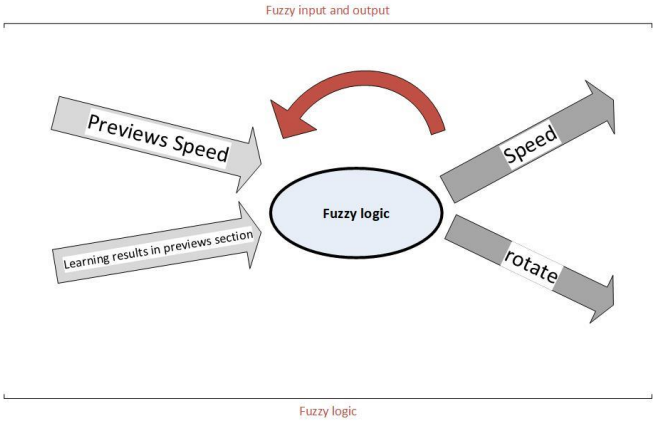

Figure 11 Fuzzy logic input and outputs

The quantity of each of the variables and their limits is determined (Figures 12 and 13), and then the fuzzy rules are defined in Figure 14.

```
#fuzzy step
learn=ctrl.Antecedent(np.arange(-1,1.01,0.01),'learn')
p_velocity=ctrl.Antecedent(np.arange(0,1.41,0.01),'p_velocity')
velocity=ctrl.Consequent(np.arange(0,1.41,0.01),'velocity')
rotate=ctrl.Consequent(np.arange(-5,5.1,0.1),'rotate')

#membership function
learn['left'] = fuzz.trimf(learn.universe, [-1, -1, 0])
learn['center'] = fuzz.trimf(learn.universe, [-0.5, 0, 0.5])
learn['right'] = fuzz.trimf(learn.universe, [0, 1, 1])

p_velocity['low'] = fuzz.trimf(p_velocity.universe, [0, 0, 0.6])
p_velocity['medium'] = fuzz.trimf(p_velocity.universe, [0.4, 0.7, 1])
p_velocity['high'] = fuzz.trimf(p_velocity.universe, [0.8, 1.4, 1.4])

velocity['low'] = fuzz.trimf(velocity.universe, [0, 0, 0.6])
velocity['medium'] = fuzz.trimf(velocity.universe, [0.4, 0.7, 1])
velocity['high'] = fuzz.trimf(velocity.universe, [0.8, 1.4, 1.4])

rotate['t_left'] = fuzz.trimf(rotate.universe, [-5, -5, 0])
rotate['straight'] = fuzz.trimf(rotate.universe , [-1, 0, 1])
rotate['t_right'] = fuzz.trimf(rotate.universe, [0, 5, 5])
```

Figure 12 Definition of fuzzy system variables

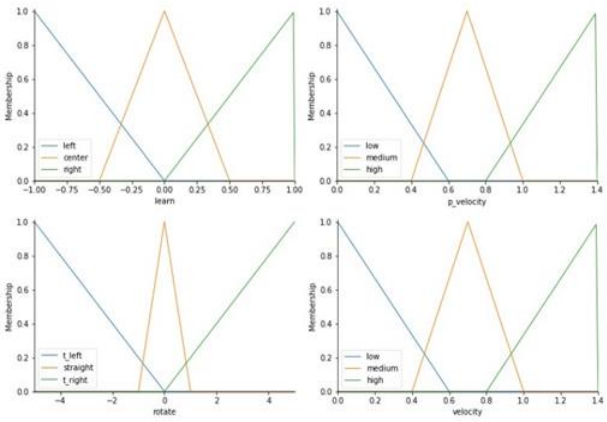

Figure 13 Fuzzy system membership

After determining the angle and the speed by the fuzzy system, the values obtained are given to the FlytOS API (shown in Figure 15) to generate the required commands for the flight controller.

```
rule1=ctrl.Rule((learn['left']&p_velocity['low']),velocity['low'])
rule2=ctrl.Rule((learn['left']&p_velocity['low']),rotate['t_right'])
rule3=ctrl.Rule((learn['left']&p_velocity['medium']),rotate['t_right'])
rule4=ctrl.Rule((learn['left']&p_velocity['medium']),velocity['low'])
rule5=ctrl.Rule((learn['left']&p_velocity['high']),velocity['medium'])
rule6=ctrl.Rule((learn['left']&p_velocity['high']),rotate['t_right'])
rule7=ctrl.Rule((learn['center']&p_velocity['low']),velocity['medium'])
rule8=ctrl.Rule((learn['center']&p_velocity['low']),rotate['straight'])
rule9=ctrl.Rule((learn['center']&p_velocity['medium']),velocity['high'])
rule10=ctrl.Rule((learn['center']&p_velocity['medium']),rotate['straight'])
rule11=ctrl.Rule((learn['center']&p_velocity['high']),velocity['high'])
rule12=ctrl.Rule((learn['center']&p_velocity['high']),rotate['straight'])
rule13=ctrl.Rule((learn['right']&p_velocity['high']),velocity['medium'])
rule14=ctrl.Rule((learn['right']&p_velocity['high']),rotate['t_left'])
rule15=ctrl.Rule((learn['right']&p_velocity['low']),velocity['low'])
rule16=ctrl.Rule((learn['right']&p_velocity['low']),rotate['t_left'])
rule17=ctrl.Rule((learn['right']&p_velocity['medium']),rotate['t_left'])
rule18=ctrl.Rule((learn['right']&p_velocity['medium']),velocity['low'])
```

Figure 14 fuzzy system rules

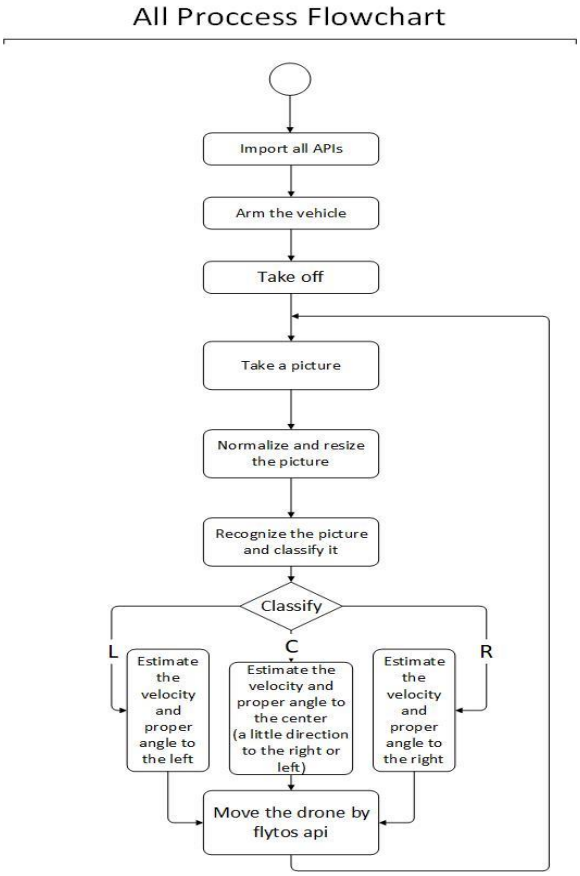

Figure 15 System schema

EXPERIMENTAL RESULTS

The Learning phase:

In the deep learning section, two optimizers (adam and sgd) and two activation functions (tanh and elu) are used to teach the model. We first taught the model by the SGD Optimizer and 10,798 data to measure the accuracy and error, which results in Table I demonstrate that overfitting has occurred. As shown in Table II, we used dropout layers after maxpooling layers to avoid overfitting. The model is also taught using the adam optimizer and the 81962 data sample, which also uses dropout layers, and the results are shown in Table III. Due to the increase in dataset size and hardware constraints, it is impossible to load all images at once, therefore place them in the memory as batches but causes to increases the learning time. Finally, the dataset volume has been increased to 322560 using augmentation, which increases the learning accuracy to 94.290 (Table IV). Test the trained model is performed by a set of images outside the data set and taken by another camera. To calculate the accuracy, we use $\frac{number\ of\ true\ recognition}{number\ of\ all\ images} * 100$, And according to the data in Table V, accuracy 93.18 is obtained.

Table I RESULTS FOR THE SGD METHOD

Epoch	*Acc*	*Loss*	*ValAcc*	*ValLoss*	*Data*
90	0.9988	0.0092	0.8294	0.6590	10798

Table II RESULTS FOR THE SGD METHOD WITH DROPOUT LAYERS

Epoch	*Acc*	*Loss*	*ValAcc*	*ValLoss*	*Data*	*DrR*
90	0.9336	0.1693	0.8432	0.4718	10798	0.1

Table III RESULTS FOR THE ADAM METHOD WITH DROPOUT LAYERS AND 81962 PICTURES.

Epoch	*Acc*	*Loss*	*ValAcc*	*ValLoss*	*DrR*	*Time (h)*
143	0.9343	0.1543	0.9393	00.1736	0.1	120
68	0.8745	0.3234	0.9114	0.2411	0.3	54
67	0.9043	0.2513	0.9127	0.2358	0.2	53

Table IV RESULTS FOR THE ADAM METHOD WITH DROPOUT LAYERS, DATA AUGMENTATION, AND ELU ACTIVATION FUNCTION.

Epoch	*Acc*	*Loss*	*ValAcc*	*ValLoss*	*DrR*	*Time (h)*
95	0.9429	0.1551	0.9628	0.1020	0.1	145

Drone Locomotion in Natural Environments:
The robot was tested in a low-traffic road in the suburbs under the mentioned conditions. Some information, such as Drone's route, speed, route length, etc., is shown in Figure 16. The Drone was filmed when it was flying and moving. Alessandro Giusti et al. [17] compared their results to those of the previous works. According to Table 7, the first row shows that the accuracy of the trained network in this paper was 94.290.

Table V Test Result

subject	*L*	*C*	*R*	*SUM*
number of images	16	9	19	44
True Recognition	16	7	18	41
false Recognition	0	2	1	3

Figure 16 The path is taken in this article

Table VI COMPARE OUR RESULTS TO PREVIOUS WORK

method	*this model*	*DNN*	*Sailenc*	*[37]*	*H1*	*H2*
accuracy	94.290	85.2	52.3	36.5	86.5	82

CONCLUSION

The method used in this article is more accurate than other methods. The accuracy was improved due to the use of dropout layers after maxpooling layers in addition to increasing the dataset size. The dataset was four times bigger than the dataset used in the source paper. This larger dataset helped to enhance accuracy significantly. According to Table 5, both the training and evaluation phases produced nearly the same accuracy, which shows that the results were acceptable. The fuzzy, TensorFlow, Keras, and other libraries installed on the Drone's computer provided artificial intelligence research foundations. More appropriate single-board computers have been developed for use in smart drones. Most of these computers have been developed by NVIDIA. Different boards such as Jetson TX1, Jetson TX2, and Jetson Nano differ in energy consumption and the number of cores. They provide acceptable computational performance for neural networks and machine learning

ACKNOWLEDGMENT

We thank the fly Base team (www.flytbase.com) for their guidance in installing flytos package.

References

For journal papers, books and conferences papers use the following formats:

[1] Al-Sharman, M.K., Zweiri, Y., Jaradat, M.A.K., Al-Husari, R., Gan, D. and Seneviratne, L.D., 2019. Deep-learning-based neural network training for state estimation enhancement: application to attitude estimation. IEEE Transactions on Instrumentation and Measurement, 69(1), pp.24- 34. Fig. 16. The path is taken in this article

[2] ANH VO(2018). Deep Learning – Computer Vision and Convolutional Neural Networks, Available At: https://anhvnn.wordpress.com/2018/02/01/deep-learning-computervision-and-convolutional-neural-networks/ (Accessed 23 October 2018)

[3] Koubaa, A., Ammar, A., Alahdab, M., Kanhouch, A. and Azar, A.T., 2020. DeepBrain: Experimental Evaluation of Cloud-Based Computation Offloading and Edge Computing in the Internet-of-Drones for Deep Learning Applications. *Sensors*, *20*(18), p.5240.

[4] Do, T., Carrillo-Arce, L.C. and Roumeliotis, S.I., 2019. High-speed autonomous quadrotor navigation through visual and inertial paths. The International Journal of Robotics Research, 38(4), pp.486-504.

[5] Dronecode, Airframes Reference ,Available At:https://dev.px4.io/ v1.9.0/en/airframes/airframe reference.html(Accessed 10 September 2017).

[6] Flytbase team, Ros Documentation, Available At: http://docs.flytbase.com/ (Accessed 12 November 2017)

[7] Giusti, A., Guzzi, J., Cires¸an, D.C., He, F.L., Rodr´ıguez, J.P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G. and Scaramuzza, D.,(2015). A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters, 1(2), pp.661-667.

[8] Guru99, Fuzzy Logic Tutorial: What is, Application & Example99,Available At: https://www.guru99.com/what-is-fuzzy-logic.html (Accessed 25 July 2018).

[9] Hardkernel, ODROID-XU4Q, Available At: https://www .hardkernel.com/shop/. odroid-xu4q(Accessed 10 September 2017).

[10] Jason Brownlee, How to Configure Image Data Augmentation in Keras, Available At: https://machinelearningmastery.com/how-to-configureimage-data-augmentation-when-training-deep-learning-neural-networks/ (Accessed 24 August 2019)

[11] Jay Ricco(2017). What is max pooling in convolutional neural networks?, Available At: https://www.quora.com/What-is-max-pooling-inconvolutional-neural-networks (Accessed 23 October 2018)

[12] Jung, S., Hwang, S., Shin, H. and Shim, D.H., 2018. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. IEEE Robotics and Automation Letters, 3(3), pp.2539-2544.

[13] Kim, D.K. and Chen, T.(2015). Deep neural network for real-time autonomous indoor navigation. arXiv preprint arXiv:1511.04668.

[14] Open Source Robotics Foundation (2018). Ros Documentation, Available At: http://wiki.ros.org/ (Accessed 15 November 2019)

[15] Resendiz, V.M.A. and Rivas-Araiza, E.A.(2016). System Identification' of a Quadrotor in X Configuration from Experimental Data. Research in Computing Science, 118, pp.77-86.

[16] Sanchez-Lopez, J.L., Sampedro, C., Cazzato, D. and Voos, H., 2019, June. Deep learning based semantic situation awareness system for multirotor aerial robots using LIDAR. In 2019 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 899-908). IEEE.

[17] Santana, P., Correia, L., Mendonc¸a, R., Alves, N. and Barata, J. (2013). Tracking natural trails with swarm-based visual saliency. Journal of Field Robotics, 30(1), pp.64-86.

[18] the scikit-fuzzy team (2012). skfuzzy 0.2 docs, Available At: https://pythonhosted.org/scikit-fuzzy/ (Accessed 15 November 2019)

[19] Tony Yiu. Understanding Neural Networks, Available At: https://towardsdatascience.com/understanding-neural-networks19020b758230 (Accessed 12 December 2018)

[20] Towardsdatascience (2017) The mostly complete chart of Neural Networks,explained.Available at:https://towardsdatascience.com/themostlycomplete-chart-of-neural-networks-explained3fb6f2367464(accessed 10 June 2018)

[21] Wang, D., Li, W., Liu, X., Li, N. and Zhang, C., 2020. UAV environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. Computers and Electronics in Agriculture, 175, p.105523.