NimbleNav: A Multi-Agent Co Operative System

Mirza Belal Ahmed [1], Shazzad Sakim Sourav [1], Ifty Mohammad Rezwan [1], S M Adnan Faroque[1] and Dr. Shahnewaz Siddiqui[2]

***Abstract*— This electronic document is a live template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

I. INTRODUCTION

For path-planning in a GPS denied environment, the options that are usually available to us are the local sensors of the robotic elements in our navigation stack. Normally, to perform a path planning task, we need an overall map that gives a heuristic view of the terrain. This map is plotted by sending an unmanned or manned ground vehicle. This ground vehicle is armed with a local sensor such as a LIDAR or a SONAR. In our case it would a LIDAR that surveys the scene by exploring all areas of a certain unexplored region. This can be quite time-consuming. Also, surprises from the unexplored terrain like the appearance of a sudden cliff can cause us to lose precious robotic elements.

To mitigate such possibilities, we propose a solution. We have a semi-autonomous unmanned aerial vehicle in our navigation stack which provides us with an aerial view from above in the form of an overhead image of the region. This image can be plotted into a map. This eases the mapping concept of the navigation stack also known as SLAM. This also helps us to directly send a robot with the predefined map so that it can take advantage of both its local object avoidance mechanism and predefined SLAM based map.

II. RELATED WORKS

In the last several years with emergence of powerful mobile computers, there have been numerous works on collaborative unmanned ground and aerial vehicle focusing robot navigation. In many cases, aerial vehicle aids the navigation system by providing overhead view to generate a map for the operational environment. On the contrary, simultaneous localization and mapping (SLAM) has enabled ground vehicles to heuristically generate possible paths through environment analysis. In both ways, robot vision has played an important role to generate operational map where different path planning algorithms could be implemented to achieve a collective goal.

A common traditional method of obstacle avoidance is to use Light Detection and Ranging (Lidar) in ground vehicle to measure distance and implement SLAM. A ground vehicle has to roam around the entire operational environment to generate such a map. On the contrary, aerial vehicles equipped with Lidar facilitates 3D map from overhead. Stereo camera is also a choice for 3D mapping. While both Lidar and stereo camera are expensive solution for robot navigation, Complementary metal oxide semiconductor (CMOS) cameras are being used to create 2D images which are then transformed into 2D occupancy grid map. In many cases, GPS is used to adjust an agent's current location with the provided map.

In case of navigation and mapping in adversarial condition, neither SLAM based mapping nor GPS dependent navigation might be a practical option. When we think of mapping and path planning, two types of environments generally come to mind - controlled small environments and large complex environments. If we consider Lidars, Lidars are too complex and expensive for a simple environment due to their collection of heavy datasets. Lidars also require multiple iterations in an individual place to collect the data properly which can be quite time consuming. In case of large complex ennvironments, Lidars are not effective in high altitudes, so overhead mapping is out of the question. Also, lidars are not able to see beyond solid objects. Now, GPS can not be used in a simple controlled environment as radio waves are known to not work in enclosed structures and spaces. In case of large complex environments, GPS normally loses the location unless initialized with a pre-defined map. GPS is also known to be quite expensive and battery hungry. At high altitudes, electromagnetic interference might occur GPS to drastically fail.

III. NIMBLENAV

We explore two possibilities to convert our aerial overhead image to a map. One of them involves a controlled image segmentation mechanism involving a handcrafted feature exploration and thresholding to isolate our region of interest along with possible obstacles.

The second approach involves a convolutional neural network based solution. This helps us to incorporate more complicated environments in our navigation stack. This is to compensate for the probability of our controlled solution not faring well in outdoor environments. We discuss both approaches in detail in the following section.

[1] Authors are students of Department of Electrical & Computer Engineering, North South University, Dhaka, Bangladesh mirza.ahmed@northsouth.edu, shazzad.sourav@northsouth.edu, mohammad.rezwan@northsouth.edu, s.faroque@northsouth.edu

[2]Bernard D. Researcheris with the Department of Electrical Engineering, Wright State University, Dayton, OH 45435, USA b.d.researcher@ieee.org

A. Controlled Image Segmentation

For this approach, at first, we provide an image of the total region. We want to isolate the region of interest and obstacle from the image. As one can clearly see from the image (inputImageGoesHere) there is a lot of unnecessary information that we have to remove to get our region of interest. There are certain adversarial elements in the image too. For instance, there is a clear presence of a hard shadow which removes the possibility of an edge detector being able to isolate our region of interest.

Fig. 1: Input Image Dilation Gaussian

So, to extract our region of interest, we take a two-step-based approach. The first step removes as much unnecessary information as possible along with the adversaries. The second step thresholds the image to a clear and obstacle-based binary map that helps our ground vehicle navigate the terrain using it.

a) First Step: As our goal is to remove as much noise and unnecessary information from the image as possible, we introduce a customized region of interest detector of our own.

At first, we apply a morphological transformation to our image followed by a smoothing operation. The morphological operation used here was a Dilation with a kernel of (5×5) over the whole image. The dilation operation can be explained with the following equation,

$$\mathbf{F}_D(x,y) = \mathbf{Convolve}\Big(\mathbf{K}_D(x,y), \mathbf{F}_I(x,y)\Big) \quad (1)$$

where:

$\mathbf{K}_D(x,y) =$ Dilation Kernel
$\mathbf{F}_I(x,y) =$ Input Image
$\mathbf{F}_D(x,y) =$ Dilation Output

This operation helps us extend the bigger regions in our image (such as our area in the maze) while adversarial and small regions like the shadows lose their region.

This is followed by a Gaussian blurring operation over our image with a kernel size of (55×55). Gaussian blurring is applied as a smoothing function.They are known to remove high frequency content such as (edges, frequent noises) etc. With the use of a large gaussian filter we can remove unnecessary background noise while at the same time lose enough focus on unnecessary small edges. Image (DilationGaussImageReference) shows the output from the Gaussian filter which was followed by the previous dilation operation. As seen from the image, the information representing background noise and unnecessary edges is decreased. This Gaussian operation requires a $sigma$ value which is calculated by the following operation.

$$\sigma = 0.3*\Big((K-1)*0.5-1\Big) \quad (2)$$

where:

$\sigma =$ Standard deviation of Gaussian Kernel
$K =$ Kernel Size

The following equation represents the Gaussian operation in two dimensions,

$$\mathbf{G}(x,y) = \frac{1}{2\pi\sigma^2}^{-\frac{x^2+y^2}{2\sigma^2}} \quad (3)$$

where:

x = Distance from the horizontal origin
y = Distance from the vertical origin
$\mathbf{G}(x,y) =$ Gaussian Value at a point

Using both of these pre-requisites, we perform the following operation.

$$\mathbf{F}_{DG}(x,y) = \mathbf{Convolve}\Big(\mathbf{K}_G(x,y), \mathbf{F}_D(x,y)\Big) \quad (4)$$

where:

$\mathbf{K}_G(x,y) =$ Gaussian Kernel
$\mathbf{F}_D(x,y) =$ Dilation Input
$\mathbf{F}_{DG}(x,y) =$ Gaussian Output

Next, we calculate the absolute difference between the output from the Gaussian function and the original image. This operation can be defined as,

$$\mathbf{A}_I(x,y) = \mathbf{Substraction}\Big(\mathbf{F}_I(x,y), \mathbf{F}_{DF}(x,y)\Big) \quad (5)$$

where:

$\mathbf{F}_I(x,y) =$ Input image
$\mathbf{F}_{DF}(x,y) =$ Gaussian output
$\mathbf{A}_I(x,y) =$ Absolute Difference

The result after the absolute difference operation can be in the image (AbsoluteDifferenceImage).

As depicted in the image above, most of the unnecessary information has been removed from the picture. But, the outline of the shadow still remains. Consequently, we run this image through another Gaussian blurring operation of kernel size (55×55). This operation can be defined as,

$$\mathbf{I}_P(x,y) = \mathbf{Convolve}\Big(\mathbf{K}_G(x,y), \mathbf{A}_I(x,y)\Big) \tag{6}$$

This is done across every color plane. After the above operations, the planes are merged together. This can be defined as,

$$\mathbf{I}_F(x,y) = \mathbf{Concatenation}\Big(\mathbf{I}_R, \mathbf{I}_G, \mathbf{I}_B\Big) \tag{7}$$

where:

$\mathbf{I}_R$ = Red Plane
$\mathbf{I}_G$ = Green Plane
$\mathbf{I}_B$ = Blue Plane
$\mathbf{I}_F(x,y)$ = First Step Output

The output of this merged image (MergedImagHistogram) shows that we have removed almost all unnecessary noise that can cause obstructions for our second step.

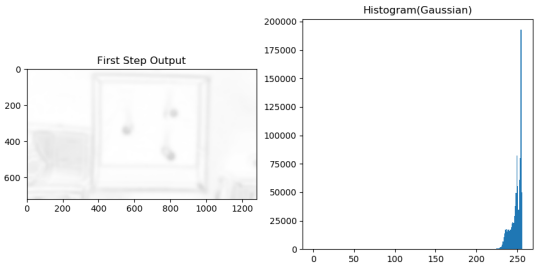


Fig. 2: Fig:First Step Output

b) Second Step: So, now we can move on to the second step of our algorithm which involves thresholding our input images to a binary map. After achieving our binary map, we have to isolate our region of interest.

At first, we perform the operation of thresholding our image to a binary image.Through careful investigation of the result of our first operation and the corressponding histogram, it is visible that our image is not a perfect bimodal image. However, it has two peak values. Moreover, we still have artefacts to remove. So, our image is a perfect candidate for Otsu based thresholding.

As Otsu is a special variant of the adaptive thresholding algorithm, it tries to find a threshold value. In this case we consider it as σ, which lies between the two peak values so that the variance can be a minimized. If we observe the histogram closely, we can see that there are many values smaller than the two peak values. If one views the first step output image closely, they will realize that the artefacts such as the shadow obstruction is very light which corresponds to the fact that it is among the low values. So, in a sense, the methodology of Otsu helps us to get rid of it completely.

The equation through which Otsu finds the threshold value is as follows,

$$\sigma_w^2 = \Big(q_1(t)\sigma_1^2 + q_2(t)\sigma_2^2\Big) \tag{8}$$

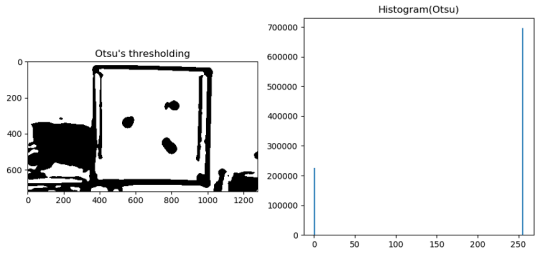


Fig. 3: Otsu Threshold

$$q_1(t) = \sum_{\mathrm{i}=1}^{\mathrm{t}}\Big(P(i)\Big) \;\&\; q_2(t) = \sum_{\mathrm{i}=1}^{\mathrm{t+1}}\Big(P(i)\Big) \tag{9}$$

$$\mu_1(t) = \sum_{\mathrm{i}=1}^{\mathrm{t}}\left(\frac{iP(i)}{q_1(t)}\right) \;\&\; \mu_2(t) = \sum_{\mathrm{i}=1}^{\mathrm{t+1}}\left(\frac{iP(i)}{q_2(t)}\right) \tag{10}$$

$$\begin{aligned}\sigma_1^2 &= \sum_{\mathrm{i}=1}^{\mathrm{t}}\left([i-\mu_1(t)]^2\frac{P(i)}{q_1(t)}\right) \;\&\\ \sigma_2^2 &= \sum_{\mathrm{i}=1}^{\mathrm{t+1}}\left([i-\mu_2(t)]^2\frac{P(i)}{q_2(t)}\right)\end{aligned} \tag{11}$$

$$\mathbf{F}_O(x,y) = \mathbf{Otsu's Thresh}\Big(\sigma_w^2, \mathbf{I}_F(x,y)\Big) \tag{12}$$

where:

q_1 = cumulative sum before threshold
q_2 = cumulative sum after threshold
$P(i)$ = Probability of i
σ_1 = standard deviation before threshold
σ_2 = standard deviation after threshold
μ_1 = mean before threshold
μ_2 = mean after threshold
$\mathbf{I}_F(x,y)$ = First Step Output
$\mathbf{F}_O(x,y)$ = Otsu Output

Upon, finding the threshold value σ which in our case was 244, we threshold the image to produce the following image and histogram

We can observe from the image (OTSUREFIMAGE) that our area of interest has been nicely segmented and we have managed to remove most of the artefacts. Furthermore, the histogram also shows us that our image has been perfectly segmented.

So, after this operation we are required to separate our area of interest which we will be using later as our binary map. For this purpose, we use a simple contour finding algorithm. Contours are fundamentally curves that connect the boundary points of objects forming a certain shape.In our case, this would be our region of interest. The image (FinalImageRef) shows the contour algorithm isolating the region of interest. After, we find the designated region of interest we cut it out from the Otsu binary image and achieve the following

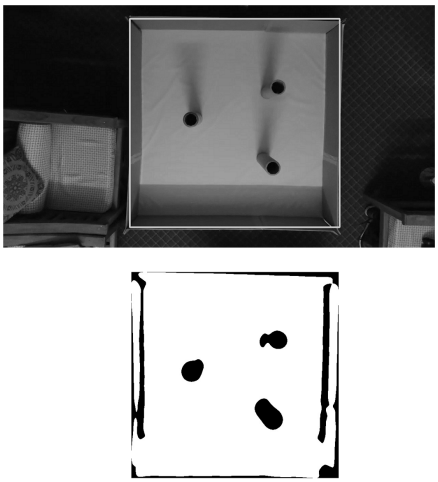


Fig. 4: Contour Output ROI

final image (FinalImageRef) The methodology can be simply stated as,

$$\mathbf{F}_{ROI}(x,y) = \mathbf{ContourCut}\Big(\mathbf{CD}, \mathbf{F}_O(x,y)\Big) \tag{13}$$

where:

$\mathbf{CD}$ = Contour Detection
$\mathbf{F}_O(x,y)$ = Otsu Output
$\mathbf{F}_{ROI}(x,y)$ = Binary Region Of Interest Image

We provide a pseudo-code for our controlled hand-crafted segmentation in Algorithm (RefAlgo)

B. Convolutional Neural Network Based Image Segmentation

In the last section, we convey that image-based segmentation is quite an arduous task. If an environment presented itself that is more complex than the circumstances presented earlier, we would be in a dilemma. One such example is given in Image(OtsuRoadOutput). In this case our first algorithm, gives us the output on the left of the image.Looking at the image, we can conclude that although it does segment regions of interest well at places.But it fails in other places as it either misclassifies the regions of interest as non-regions of interest and vice-versa.

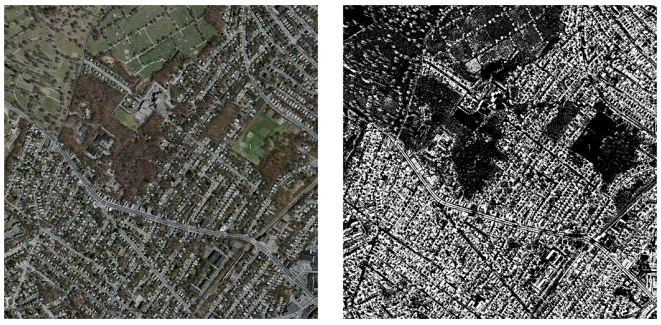


Fig. 5: Hand Crafted Road Output

So, to handle more complex situations we propose an alternative approach. This alternative is based on convolutional neural networks. Convolutional neural networks have been on the rise since the advent of (Alexnet) (Paper) in 2012. Convolutional neural networks are very efficient feature extractors that can be trained to learn using only a decent amount of data. As this is a segmentation problem, we try to adopt a fast but efficient and accurate popular segmentation network. The network we take our inspiration from is the famed Unet (UnetPaper) which was originally created for medical image segmentation. For our problem here which is the detection of roads and no-go zones we adopted our problem into a specific form of segmentation called semantic segmentation. In essence, semantic segmentation is the classification of each individual pixels into a certain class or another. After this, we designate a specific color to the class. In our case, as there are two classes - roads and background - we assign white for roads and black for no-go zones. Thus, we are presented with a binary map.

In the following section, we will be explaining our architecture, training procedure, relevant functions along with the dataset we used. Furthermore, a visualization of the left image (OtsuRoadOutput) is shown as well after it passes through the network. This proves the effectivenes of convolutional neural network in binary map generation and complex image segmentation.

1) Architecture: The architecture is based on Unet. It can be defined into three main parts.

- Downsampling Block.
- Bottleneck Block.
- Upsampling Block.

a) Downsampling Block: The downsampling block is in charge of scaling the image to a compressed form. In our architecture this is done through a few convolution layers. Our input images are 3-channel images of shape $(512 \times 512 \times 3)$. The convolution layers have a filter shape of (3×3) and an arbitrary number of filters that starts from 64 to 1024, gradually increasing with a multiple of 2 every two layers. All kernels throughout the network are initialized through $?HeInitialization?$. This ensures the absence of vanishing or exploding gradients to some extent. The output spatial window of every layer can be calculated by using the following convolution arithmetic.

$$\mathbf{F}_{(out)} = \lfloor \frac{F_{in} + (2 * P) - K}{S} \rfloor + 1 \tag{14}$$

where:

F = features
P = padding
K = kernel size
S = stride

The output channel can be calculated by the number of the filters Fil used.

Every convolution layer is followed by an activation function, in our case $RELU$, which introduces non-linearity to the network. After this we add batch normalization followed

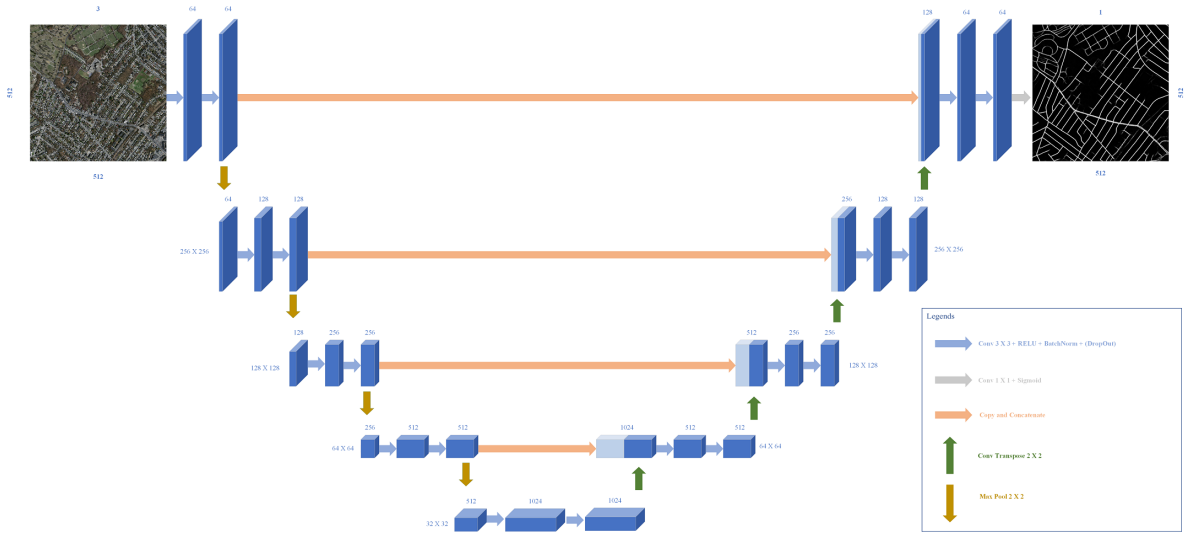


Fig. 6: Fig:UNet Architecture

by dropout to reduce overfitting and gain accuracy at the same time. We also perform $(2X2)$ maxpooling after every two layers to ensure downscaling. This block helps us downscale the image to a compressed version while retaining as much information as possible.

b) Bottleneck Layer: The bottleneck layer is in between the downsampling and upsampling block. It can be considered the layer which holds the compressed version of the information required to convert our input image to a binary map. This layer consists of two convolutional layers. The first convolution is similar to the one performed in the earlier block.This is followed by a simple transpose convolution layer which can be considered the opposite of convolution layers. They upscale the compressed information.

c) Upscaling Block: The upscaling blocking block is exactly the opposite of the downsampling block. It upscales our information from the bottleneck layer to a different form than the input, in our case the binary map of shape $(512X512X1)$. This is achieved by following the same convolutional kernels, activation, batch normalization and dropout layers. Except in place of maxpooling, we perform $(2X2)$ transposed convolution to upscale the image and the kernels are in reverse order from 1024 to 64. We also add skip connections from every downsampling block to every corresponding upsampling block. This ensures the throughput of information in the network without which the network?s performance worsens through the rise of the vanishing gradient problem. The skip connections are managed through the use of $(1X1)$ convolutions followed by a sigmoid which acts as a binary activation. The sigmoid operation is as follows,

$$\phi(input) = \frac{1}{1 - e^{-input}} \tag{15}$$

The operations explained above can be visualized through the image of the architecture (Architecture Image Reference). Thus, we get our binary map from the complex input image.

2) Training Procedure: For our training procedure, we require a loss function. We chose a customized loss function, which is a mixture of binary cross entropy and dice loss(Reference). A brief description of our customized loss function is as follows,

$$\mathbf{BCE} = \sum_{\text{i}} \Big(y_i log(p_i) + (1 - y_i) log(1 - log(p_i)) \Big) \tag{16}$$

$$\mathbf{DiceLoss} = 2 \Bigg(\frac{\sum_{\text{i}} y_i p_i}{\sum_{\text{i}} (y_i + p_i)} \Bigg) \tag{17}$$

$$\mathbf{CustomLoss} = \Big(\mathbf{BCE} - \alpha \mathbf{DiceLoss} \Big) \tag{18}$$

where:

BCE = Binary Cross Entropy
$DiceLoss$ = Dice Co-Efficient Loss
α = 0.0005
y_i = if observation is correct or not
p_i = predicted probability of observation

Other than this to ensure the performance of our model, we use multiple metrics to evaluate our model. We use the Mean Iou and Dice Coefficient besides our custom loss to evaluate the model. An explanation for both Mean Iou and Dice Coeffcient metric is as follows,

$$\mathbf{IOU\ Loss} = -log \Bigg(\frac{Intersection(GT, Pred)}{Union(GT, Pred)} \Bigg) \tag{19}$$

$$\mathbf{Dice\ Loss} = -log \Bigg(\frac{2 * Intersection(GT, Pred)}{Union(GT, Pred)} \Bigg) \tag{20}$$

where:

$Intersection$ = Overlap of Bounding Boxes
$Union$ = Total area of Bounding Boxes
GT = Ground Truth Bounding Box
$Pred$ = Predicted Bounding Box

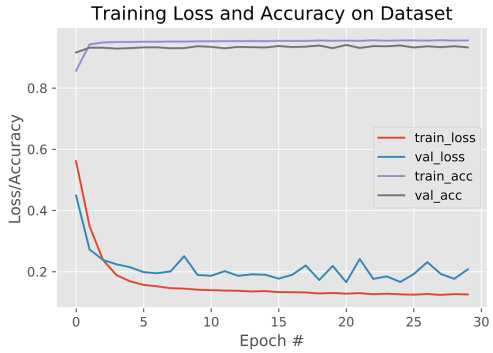


Fig. 7: Plot Training(trainloss)

The training dataset we used was the Massachusetts Roads Dataset.This dataset consisted of 1171 aerial images. We used 1108 images as training images and 14 for validation images. The rest were test images and had no labels. The training and validation dataset covers around 2.25 square kilometers whereas the test set covers around 110 square kilometers. This provides a great to see how robust our model is. The input images are of shape $(1500 \times 1500 \times 3)$. On top of that, the training dataset covers a mixture of urban, suburban and rural areas which makes it quite complex.

We use Adam as our optimizer function with an initial learning rate of 0.0001. In addition to this, we use a few data augmentations namely a rotation of 0.2, width shift, height shift, shear and a zoom of 0.05. We also added a horizontal flip. We train for around 30 epochs with learning rate plateau. The results of our experiment are as follows, (Two Training Images)

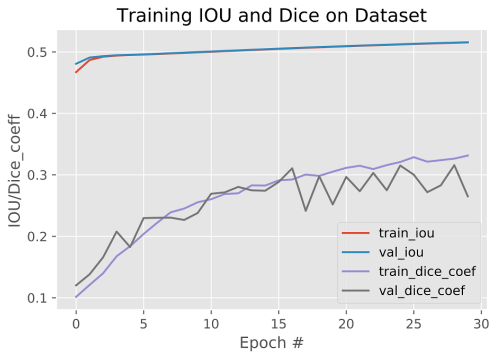


Fig. 8: Fig:Plot Training(dice, iou)

The image above shows a mean iou of 63.25% percent and dice coefficient of 34%. We also have a validation accuracy of 96% which is quite acceptable. As mean iou is above 50% and dice coefficient is above 30%, we can conclude we have a robust model. We also got a 2.8% increase on the test set of this dataset from the original authors. The authors of the dataset achieved 90.03%, we on the other hand managed to get 92.73%. The following image (UnetRoadOutput), further supports our result.

Fig. 9: UNet Road Output

IV. Experimental Setup

A. Hardware

TABLE I: Hardware Specification

	Controlled	Unet
CPU	AMD A12 9720p	AMD RYZEN 7 3800X
Process Node	28nm	7nm
Clock Speed	2.6GHz-3.7GHz	3.9GHz-4.5GHz
Cores	4	8
Threads	4	16
Ram	8GB DRR3	16GB DDR4
GPU	N/A	GTX 1070ti

1) Robot Vision Unit: Ground Station...

a) Controlled: Test and Analaysis: We conducted a standard benchmark of our total algorithm to see how long it takes to operate. The median of ten runs was 0.069 seconds. Our input frame is an image of $(1280 \times 768 \times 3)$. So,our algorithm works at about 14 frames per second on a CPU. The device used was an AMD A12 9720p CPU which is 2.6GHz-3.7GHz (four core, four thread) 28nm processor. If one were to use a GPU,they would get significant speedups for our algorithm is extendable to GPUs.

b) Unet: Test and Analysis: The above test image is an image not seen by the network before. Our Network can be extended for other types of terrain or mode of travel as well. The training time takes around 40 minutes per epoch. The segmentation of $(512X512X3)$ image to a $(512X512X1)$ image takes around 2 seconds on a GTX1070ti, AMD-RYZEN-3800X, 16GB RAM hardware based PC for a single image.

2) Agents:

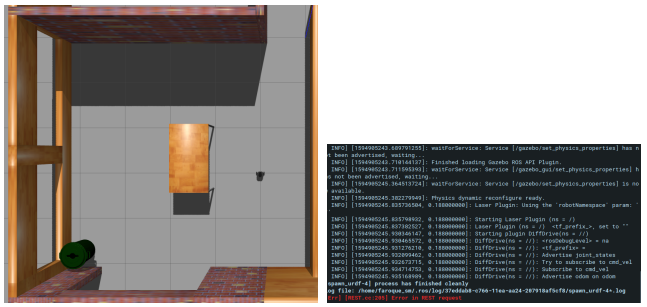
B. Robot Navigation System

a) Agent Assignment: When dealing with multiple agents, the standard method for using them with ROS is to assign them different names($namespaces$). This allows us to separately handle each of their navigation. ROS has a package called $navigation$ which is commonly known as the navigation stack. This takes odometry and sensory streams

TABLE II: Agent Specification

	UGV	UAV
Name	TurtleBot 3 Burger	Custom Made
Computational Unit	Raspberry Pi 3 B	Raspberry Pi 3 B
Control Unit	OpenCR	PixHawk 2.1
Battery	3 cell	3 cell
Threads	4	16
Sensors	LiDAR	Camera
GPU	N/A	GTX 1070ti

as inputs and outputs to mobile base velocity commands. However to use it on any robot requires some tweaking. The robot which will be leveraging the navigation stack will need to have ROS installed and running and will need to have a tf transform tree in place. Moreover, the sensor data needs to be published using the correct ROS Message types. Each node in a single instance of ROS Master must have a unique name. As such, when using the same type of robot multiple times in the same ROS Master, the individual links as well as joint names should be prepended by robot name(*namespace*). This way we can avoid redundancy of maintaining multiple separate URDFs.

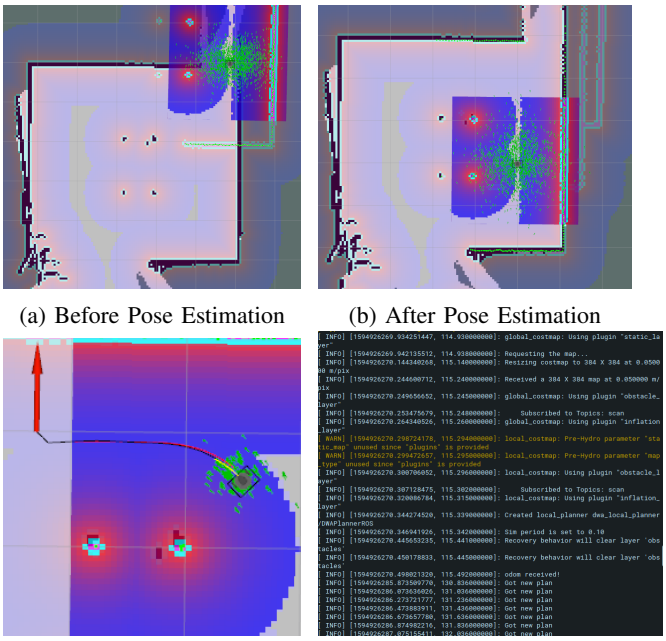


(a) Turtle World (b) Subscription to Topics

Fig. 10: Turtle World to Topic Subscription

b) Map deployment: Upon spawning the agents in the simulation environment, the robots' initial pose is estimated in *rviz*. This is absolutely necessary because it is the only way we can ensure that the robots are localized in the simulation environment. This localization helps to align the robot with the provided global map. This global map is shared by all the robots in the same environment. This is facilitated through the use of *map_server* node. In order to receive feedback from the robots during navigation and receive response after a long time, the ROS navigation stack is based on ROS Actions. There is also the *move_base* ROS node without which the navigation would be quite impossible. This node enables configuring, running and interacting with the navigation stack. Communication with the *move_base* node is carried out using *SimpleActionClient* interface. By combining global and local motion planners, this node endeavours to accomplish a desired pose and carries out navigation tasks while avoiding obstacles. The figure shows this process.

The processed image from the UNET and our handcrafted



(a) Before Pose Estimation (b) After Pose Estimation

(c) Following Nav Goal (d) Continuos Path Calculation

Fig. 11: TurtleBot Navigation

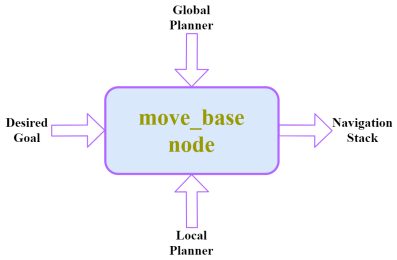


Fig. 12: Move Base Node

controlled environment image needs some adjustments in order for us to use it as a global map. From this image, a rectangular ROI(Region Of Interest) is selected containing the turtlebot. The longest side is chosen as the width(for example 100 pixels). This pixel is set to be 0.178m. From that we do a proper width calculation of the entire image. To do this, we divide the longest side of the image by the longest side of the ROI and multiply it by 0.178m. This gives us the dimension of the entire image in meters. Next, we multiply this dimension with 0.05 to get the proper pixels this image needs to be. Now we convert this image in calculated pixels. This allows us to retrieve an image which can be set inside the gray area of the map image. The resulting final image will always be 384x384 pixels.

V. CONCLUSIONS

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

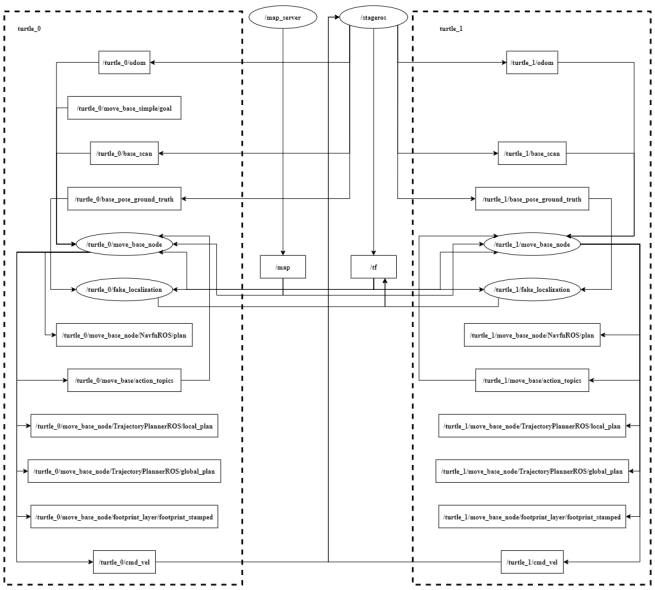


Fig. 13: Graph of the navigation stack

REFERENCES

[1] G. O. Young, Synthetic structure of industrial plastics (Book style with paper title and editor), in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 1564.

[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135.

[3] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.

[4] B. Smith, An approach to graphs of linear forms (Unpublished work style), unpublished.

[5] E. H. Miller, A note on reflector arrays (Periodical styleAccepted for publication), IEEE Trans. Antennas Propagat., to be publised.

[6] J. Wang, Fundamentals of erbium-doped fiber amplifiers arrays (Periodical styleSubmitted for publication), IEEE J. Quantum Electron., submitted for publication.

[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

[8] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, Electron spectroscopy studies on magneto-optical media and plastic substrate interfaces(Translation Journals style), IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].

[9] M. Young, The Techincal Writers Handbook. Mill Valley, CA: University Science, 1989.

[10] J. U. Duncombe, Infrared navigationPart I: An assessment of feasibility (Periodical style), IEEE Trans. Electron Devices, vol. ED-11, pp. 3439, Jan. 1959.

[11] S. Chen, B. Mulgrew, and P. M. Grant, A clustering technique for digital communications channel equalization using radial basis function networks, IEEE Trans. Neural Networks, vol. 4, pp. 570578, July 1993.

[12] R. W. Lucky, Automatic equalization for digital communication, Bell Syst. Tech. J., vol. 44, no. 4, pp. 547588, Apr. 1965.

[13] S. P. Bingulac, On the compatibility of adaptive controllers (Published Conference Proceedings style), in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 816.

[14] G. R. Faulhaber, Design of service systems with priority reservation, in Conf. Rec. 1995 IEEE Int. Conf. Communications, pp. 38.

[15] W. D. Doyle, Magnetization reversal in films with biaxial anisotropy, in 1987 Proc. INTERMAG Conf., pp. 2.2-12.2-6.

[16] G. W. Juette and L. E. Zeffanella, Radio noise currents n short sections on bundle conductors (Presented Conference Paper style), presented at the IEEE Summer power Meeting, Dallas, TX, June 2227, 1990, Paper 90 SM 690-0 PWRS.

[17] J. G. Kreifeldt, An analysis of surface-detected EMG as an amplitude-

modulated noise, presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.

[18] J. Williams, Narrow-band analyzer (Thesis or Dissertation style), Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.

[19] N. Kawasaki, Parametric study of thermal and chemical nonequilibrium nozzle flow, M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.

[20] J. P. Wilkinson, Nonlinear resonant circuit devices (Patent style), U.S. Patent 3 624 12, July 16, 1990.