Project P2 : Dramatic Data
Used 1 Late Day

Deepak Singh
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: dsingh1@wpi.edu

Sai Hitesh Viswasam
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: sviswasam@wpi.edu

Samuel Markwick
Department of Robotics Engineering
Worcester Polytechnic Institute
Email: spmarkwick@wpi.edu

***Abstract*—This paper presents our project on synthetic data generation and segmentation, consisting of three key stages: data generation, semantic segmentation, and instance segmentation. The data is generated using Blender, where various augmentations such as homography, noise, and blur are applied to enhance the dataset. The generated data simulates a drone testing arena containing windows, with the goal of enabling the drone to identify and navigate through these windows. In the first stage, semantic segmentation is performed to identify and segment the windows in the scene. This is achieved using a U-Net architecture with a MobileNet encoder. Following this, instance segmentation is carried out to distinguish individual windows when multiple instances are present. For this, we employ an unsupervised clustering approach to separate the windows into distinct instances.**

I. Introduction

Autonomous systems, particularly drones, require precise perception capabilities to navigate through complex environments. One key challenge for drones is the ability to accurately detect and traverse obstacles such as windows or openings [1]. Effective segmentation techniques are essential to enable drones to recognize these features and make informed navigation decisions in real time.

While real-world data collection for such tasks can be expensive and time-consuming, synthetic data generation offers a scalable and versatile alternative. By creating realistic, customizable datasets, synthetic data can simulate various scenarios that drones might encounter during their operation.

Augmentation plays a crucial role in improving the generalization ability of deep learning models. By introducing variations in the training data, such as changes in scale, orientation, lighting, and occlusion, augmentation helps models become more robust to real-world variations. This is particularly important in tasks like segmentation, where the model must be able to adapt to different perspectives and environmental conditions. Data augmentation is an essential technique for increasing the diversity of the training set without the need for additional data collection, thereby enhancing the model's performance.

Image segmentation models, such as U-Net [5] and SegNet [6], have shown remarkable success in pixel-wise classification tasks. These models are designed to predict a label for each pixel in the image, enabling the identification of specific regions like windows or doors in complex environments. U-Net, for instance, is widely used for its encoder-decoder architecture, which efficiently captures spatial information through skip connections. SegNet also follows a similar architecture but focuses on memory efficiency, making it suitable for real-time applications.

In addition to semantic segmentation, instance segmentation models like Mask R-CNN [7] extend the task to differentiate between distinct object instances within the same class. This is particularly useful in scenarios where multiple windows or obstacles might exist, and the drone needs to distinguish between them to navigate safely. Instance segmentation not only identifies the boundaries of objects but also isolates individual objects within the scene, providing a higher level of detail and precision necessary for autonomous navigation.

II. Dataset Generation

In the context of performing Semantic and Instance Segmentation tasks on images through CNN architectures, the network needs to be trained by large amounts of training data to account for variations in different scenarios to match the generalisation in the real world. Manually collecting the data is both a tedious and an error prone process, along with the difficulty associated with obtained varied scenarios involving many implausible, destructive or expensive setups. Hence to overcome this, the idea of Synthetic Dataset Generation is a popular approach in the Computer Vision community to obtain images required for the training through simulation environments [2].

Many previous works used the rendering software Blender [3] to achieve photorealistic images to train the model on. The goal is to generate the images of the PeAR racing windows from different views and different backgrounds through a process called **Domain Randomisation** to ensure that the simulated dataset encompasses the possible real world scenarios as its subset. We aimed to generate around 50,000 training images for the network to train effectively for various cases. Since rendering everything from Blender alone is a computationally

expensive process, we have also incorporated Data Augmentation techniques from `torchvision.transforms` library in PyTorch for a better sim2real generalisation, which is further explained in the subsections below.

A. Blender Setup

Using Blender, the goal was to generate the object(s) (PeAR racing windows) in multiple different orientations and capture the scene using different cameras from different angles. To generate large number of images, a lot of variables within the image including lighting, background, occlusion objects could be varied for each setup. We also generated images even with the cases including multiple test windows in the scene. Through a small script in Blender, we ensured to vary the parameters mentioned above to obtain multiple images for the training.

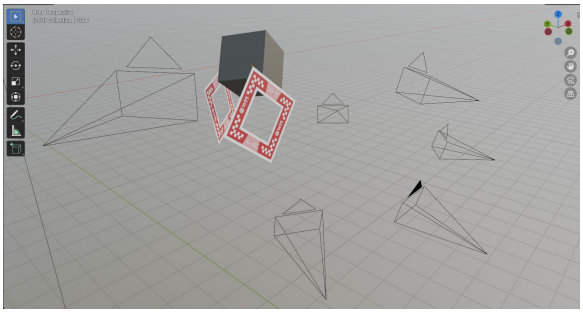

Fig. 1. A scene in Blender with multiple cameras capturing a random orientation of the windows along with an occluded object.

Along with the image generation process, for each scene rendered, we also generated a binary mask for each window present in the scene and that is passed to the network as the ground truth for the segmented result. This is done by setting an index for each window present in the scene. Blender then recognises it as an object and outputs a binary mask for each object for every rgb image rendered.

Fig. 2. Image of a scene generated along with its Ground truth segmentation mask, both of which are generated through Blender.

B. Data Augmentation

Data augmentation is crucial for training any CNN as it enhances model generalization by artificially increasing the diversity of the training dataset. Owing to the computational complexity and time constraints, only a subset of the training dataset augmentations are performed through Blender and the rest of the augmentations are performed on the images on the fly during training. Some of the augmentation transformations included Gaussian noise addition, Simulating different lighting conditions, adjusting colour properties of the image, blurring the image, posterizing the image, applying homography transformations etc. which are shown in the below images.

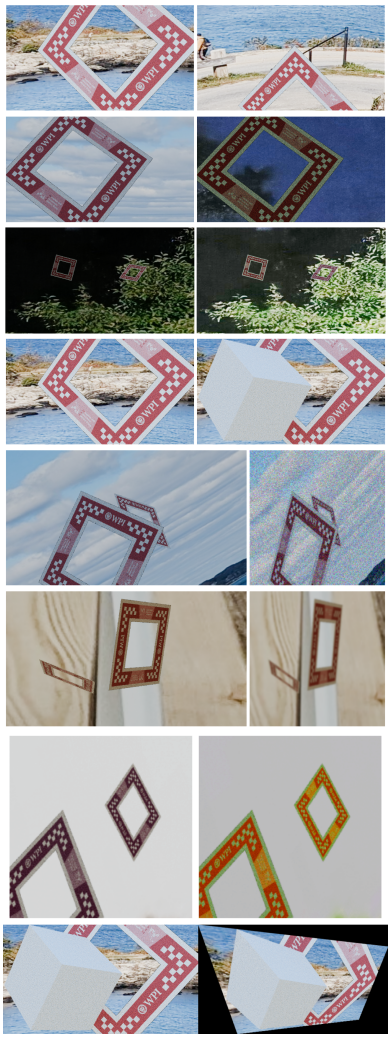

Fig. 3. Augmentation of Images in the dataset. Starting from above, the augmentations are: Camera Tilt, Change in Background, Change in Lighting, Adding Occlusion Object, Adding Gaussian Noise, Blurring, Changing Colour properties of the image and Homography transformation respectively.

All of the augmented images and the corresponding masks are then resized and converted into torch tensors and are passed into the CNN for training.

III. Semantic Segmentation Model Architecture

The semantic segmentation model follows a U-Net structure with an encoder-decoder architecture, where the encoder utilizes convolutional blocks similar to ResNet18 [**?**]. The decoder consists of transposed convolutions (upsampling layers) and skip connections to recover spatial information. Below is a detailed description of the model architecture:

A. Encoder Architecture

The encoder consists of the following layers:

- **First Encoder Block (Input to 64 channels):**
  - A convolution layer with 3 input channels and 64 output channels, using a kernel size of 7, stride of 2, and padding of 3. This layer extracts basic features from the RGB image (256x256).
  - Batch normalization is applied after the convolution to stabilize and speed up training.
  - ReLU activation is applied after batch normalization to introduce non-linearity.
  - Max pooling with a kernel size of 3, stride of 2, and padding of 1 is applied to downsample the spatial resolution.
- **Residual Blocks (with Residual Connections):**
  - **Block 1 (64 to 64 channels):**
    - Consists of two convolution layers, each with 64 output channels and a kernel size of 3. The first convolution uses a stride of 1.
    - Batch normalization is applied after each convolution, followed by ReLU activation.
    - A shortcut connection is made to add the input to the output of the block, creating a residual connection, which helps mitigate the vanishing gradient problem.
  - **Block 2 (64 to 128 channels):**
    - This block consists of two convolution layers, with the first layer increasing the number of channels from 64 to 128. The first convolution uses a stride of 2, reducing the spatial resolution by half.
    - A residual connection is applied, with a 1x1 convolution in the shortcut path to match the dimensions for addition.
    - Batch normalization and ReLU activation are applied after each convolution.
  - **Block 3 (128 to 256 channels):**
    - This block increases the number of channels from 128 to 256 using a stride of 2 in the first convolution layer, further downsampling the input.
    - A residual connection with a convolutional shortcut is applied, ensuring that the input dimensions match for the skip connection.
    - Batch normalization and ReLU activation are used after each convolution.
  - **Block 4 (256 to 512 channels):**
    - The final block in the encoder increases the number of channels from 256 to 512, with the first convolution downsampling the input using a stride of 2.
    - A residual connection with a shortcut path is applied to maintain feature flow from earlier layers.
    - Batch normalization and ReLU activation follow each convolution operation.

B. Decoder Architecture

The decoder mirrors the encoder, with transposed convolution layers (upsampling layers) and skip connections from the corresponding encoder layers. These components ensure that spatial information is recovered as the feature maps are upsampled.

- **Upsampling Block 1 (512 to 256 channels):**
  - A transposed convolution layer is used to upsample the feature map from 512 to 256 channels. This layer uses a kernel size of 2 and stride of 2 to increase the spatial dimensions.
  - The upsampled feature map is then interpolated to match the spatial size of the output from the encoder's fourth block, and a skip connection is applied, adding the upsampled features to those from the encoder.
- **Upsampling Block 2 (256 to 128 channels):**
  - A transposed convolution is applied to upsample the feature map from 256 to 128 channels, increasing the spatial dimensions.
  - The upsampled feature map is then interpolated to match the size of the encoder's third block, and a skip connection is applied.
- **Upsampling Block 3 (128 to 64 channels):**
  - A transposed convolution layer is used to upsample the feature map from 128 to 64 channels. The output is then interpolated to match the size of the second encoder block, where a skip connection is applied.
- **Upsampling Block 4 (64 to 64 channels):**
  - Another transposed convolution is applied, upsampling the feature map from 64 channels to the original input size. The output is added to the features from the first encoder block via a skip connection.

C. Final Output Layer

The final output layer consists of:

- A 1×1 convolutional layer that reduces the number of channels to match the number of output classes (in this case, 1 for binary segmentation).
- The output is bilinearly interpolated to the final size of 256×256, matching the size of the input image for pixel-wise segmentation.

Model-weight-file-link-https://wpi0-my.sharepoint.com/:u:/g/personal/dsingh1_wpi_edu/EccpHgzfMLdKqnBnWPTfFX4BKMDkHl8H8sbiza_opAxJMg?e=lpzhzY

D. Skip Connections

Skip connections are implemented between corresponding layers of the encoder and decoder. These connections allow the network to leverage high-resolution information from earlier layers during the upsampling process, improving the spatial accuracy of segmentation. Each upsampled feature map is interpolated to match the size of its corresponding encoder output, after which the two are added together to reinforce spatial details.

Fig[4] shows the residual block used in the architecture. Fig [5] shows the segmentation model architecture as described above.

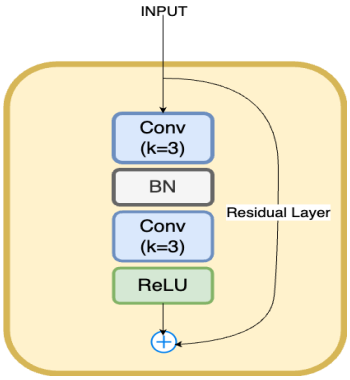

Fig. 4. Residual Block Architecture

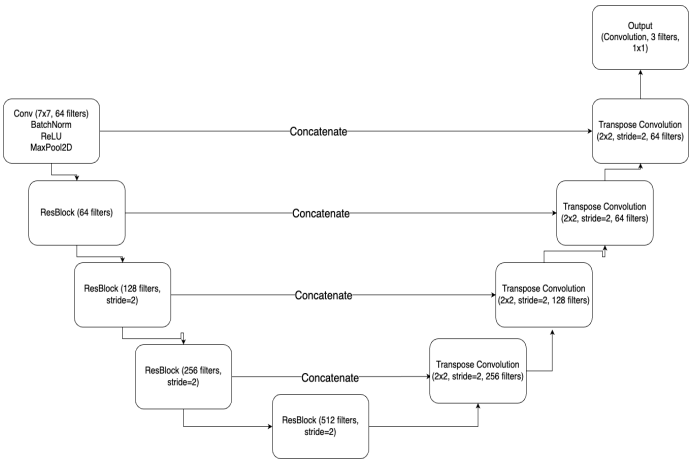

Fig. 5. Segmentation Model Architecture

E. Loss Function

In semantic segmentation tasks, the goal is to classify each pixel in an image as belonging to a specific class. When the segmentation task is binary (i.e., only two classes such as foreground and background), the *Binary Cross-Entropy* (BCE) loss function is commonly used to train the model. The Binary Cross-Entropy loss is defined as in Eq[1]:

$$\mathcal{L}_{\text{BCE}}(p, y) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right], \tag{1}$$

where:

- N is the total number of pixels in the image.
- $y_i \in \{0, 1\}$ is the ground truth label for pixel i, where 0 represents the background class and 1 represents the foreground class.
- p_i is the predicted probability that pixel i belongs to the foreground class.

For each pixel, the BCE loss measures the difference between the predicted probability p_i and the actual label y_i. When the predicted probability is close to the actual label (e.g., $p_i \approx y_i$), the loss is small. Conversely, when the predicted probability differs significantly from the true label, the loss increases. In the context of semantic segmentation, the Binary Cross-Entropy loss is applied to every pixel in the image, and the final loss is the average loss across all pixels. This ensures that the model optimizes to correctly predict the class label for each pixel in the image.

a) Pixel-wise Application:: For semantic segmentation, the BCE loss is applied at the pixel level. For each pixel in the image, the model outputs a probability score between 0 and 1, which represents the likelihood of that pixel belonging to the foreground class. The BCE loss function penalizes incorrect predictions and encourages the model to output values closer to the ground truth.

b) Thresholding:: After training, the output probabilities are typically thresholded to make a final decision for each pixel. A common threshold is 0.5, meaning that pixels with predicted probabilities greater than 0.5 are classified as foreground, while those with probabilities less than 0.5 are classified as background. Binary Cross-Entropy loss is well-suited for binary segmentation problems where the foreground and background classes are equally important. However, for highly imbalanced classes, additional weighting techniques may be applied to ensure that both classes are properly learned by the model.

IV. Instance Segmentation

Instance segmentation refers to the task of identifying and labeling individual object instances within an image. One effective technique for this task is the **connected components** algorithm, which identifies and labels distinct regions of similar pixels based on their connectivity. This method is particularly suitable for segmenting spatially distinct objects in binary masks.

A. Connected Components Algorithm

The connected components algorithm works by examining a binary mask, which is typically the output of a semantic segmentation model. In this mask, pixels belonging to an object have a value of 1, and background pixels have a value of 0. The goal is to assign a unique label to each connected region of object pixels.

1) Algorithm Steps: The steps of the connected components algorithm are as follows:

1) **Input:** A binary mask, $M \in \{0, 1\}^{H \times W}$, where H and W represent the height and width of the image, respectively. The mask has pixel values:
$$M(i, j) = \begin{cases} 1, & \text{if the pixel belongs to an object,} \\ 0, & \text{if the pixel belongs to the background.} \end{cases}$$
2) **Label Initialization:** Create a label map L of the same size as M, initialized to 0:
$$L(i, j) = 0 \quad \forall\, (i, j) \in [1, H] \times [1, W].$$
3) **Label Assignment:** For each pixel (i, j) in M, if $M(i, j) = 1$ and $L(i, j) = 0$, assign a new label k

and propagate this label to all connected object pixels. The connectivity can be based on either 4-connectivity or 8-connectivity:

- **4-connectivity:** A pixel is connected to its neighbors in the up, down, left, and right directions.
- **8-connectivity:** A pixel is connected to its neighbors in all eight directions (including diagonals).

V. Experiments and Results

The hyper-parameters used for training the model are as shown in Table [V]

Hyperparameter	Value
Epochs	100
Batch Size	32
Total Images	40,000
Learning Rate	1×10^{-4}
Decay Rate	0.1 / 10 epochs
Optimizer	ADAM

TABLE I
HYPERPARAMETERS FOR MODEL TRAINING

Fig[6] shows the training and validation loss for the model, when trained using the above hyperparameters.

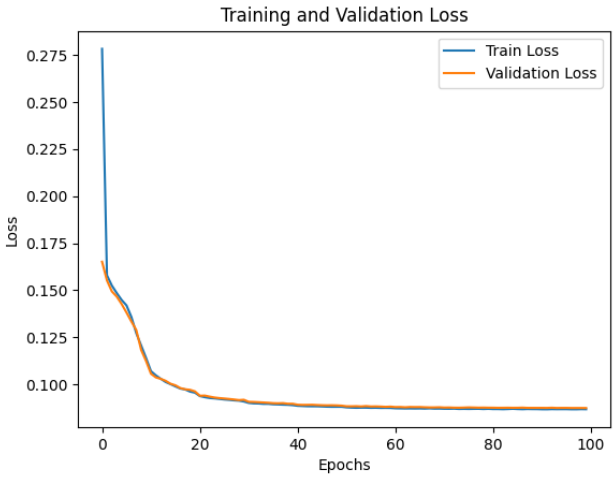

Fig. 6. Training and Validation Loss

Fig[7] shows sample outputs of the model on a couple of images, and the ground-truth masks as well, alongside.

Fig [8 shows the instance segmentation model output on one of the images above
The semantic segmentation (and consequently the instance segmentation) model exhibits two prominent failure cases:

1. Tilted Windows: The first major source of failure occurs when windows are highly tilted. The model is trained predominantly on windows with a more standard orientation, typically aligned closer to vertical or horizontal axes. As a result, when the window appears significantly tilted, the model struggles to correctly segment it. This failure highlights a limitation in the diversity of training data, suggesting that additional examples of tilted windows are needed to improve the model's robustness to such geometric variations. Without this diversity, the model generalizes poorly to windows with unusual orientations, leading to significant segmentation errors.

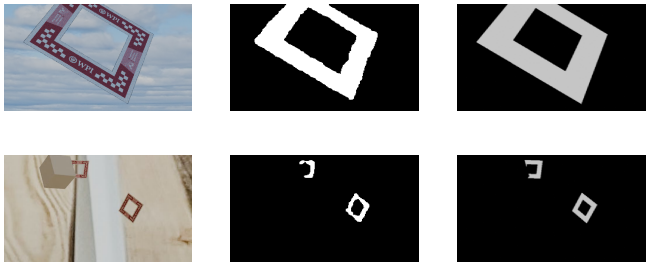

Fig. 7. Comparison of Predicted Mask (center) and Ground Truth Mask (rightmost), on the Original Image (leftmost)

Fig. 8. Comparison of Semantic Segmentation Mask (center) and Instance Segmentation Mask (rightmost), on the Original Image (leftmost)

2. Small Windows with Dark Backgrounds: Another critical failure occurs when the windows are small and placed against dark backgrounds. The model faces difficulty in distinguishing the small window features from the background, as they may become less prominent or easily lost in the noise. This scenario often results in incomplete segmentation or a complete failure to detect the windows. The challenge is compounded by the dark background, where contrast between the object and the background is insufficient for the model to reliably distinguish the window boundaries. This failure highlights the need for either better preprocessing techniques to enhance contrast or a more focused training set that includes small windows under various lighting conditions.

Sample outputs for fail cases are shown in Fig [10] and Fig [9]

Fig. 9. Fail Case: Small windows in dark background ((Original Image, Predicted Mask, and Ground Truth Mask)). The model struggles to segment small windows accurately due to poor contrast and window size.

Fig. 10. Fail case: Highly tilted windows (Original Image, Predicted Mask, and Ground Truth Mask).

Fig [11] shows a fail case of instance segmentation when semantic segmentation worked well. In this case, both the windows are connected and hence a classical approach like watershed segmentation fails in this case and identifies it as a single entity.

Fig. 11. Fail case: Multiple windows very close to each other (Original Image, Semantic Segmentation Mask, and Instance Segmentation Mask).

VI. Conclusion

This project showed the capability of learning based models for semantic segmentation and classical approaches for instance segmentation. Different methods of Data Generation through simulation and Data Augmentation were discussed in the report. This dataset was then used to train and validate the network for the segmentation tasks and the loss curves are plotted. Both the training and validation losses converge to a low value, indicating that there is no issue of overfitting in the model and that the model works well. However, the semantic segmentation model had fail cases in the scenarios where extremely tilted windows were present and in scenes where dark backgrounds were present. The instance segmentation model on the other hand failed to recognise two different windows when they had certain overlap.

References

[1] Sanket, Nitin and Singh, Chahat and Ganguly, Kanishka and Fermüller, Cornelia and Aloimonos, Yiannis. (2018). GapFlyt: Active Vision Based Minimalist Structure-Less Gap Detection For Quadrotor Flight. IEEE Robotics and Automation Letters. PP. 10.1109/LRA.2018.2843445.

[2] H. Vietz, T. Rauch, and M. Weyrich, "Synthetic training data generation for convolutional neural networks in vision applications," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–6.

[3] A. Jabbar, L. Farrawell, J. Fountain, and S. K. Chalup, "Training deep neural networks for detecting drinking glasses using synthetic images," in *Neural Information Processing*, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham: Springer International Publishing, 2017, pp. 354–363.

[4] Singh, C. D., Kumari, R., Fermüller, C., Sanket, N. J., and Aloimonos, Y. (2022). WorldGen: A large scale generative simulator. arXiv preprint arXiv:2210.00715.

[5] Ronneberger, O., Fischer, P., and Brox, T., "U-Net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention*, Cham: Springer International Publishing, 2015, pp. 234–241.

[6] Badrinarayanan, V., Kendall, A., and Cipolla, R., "SegNet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[7] He, K., Gkioxari, G., Dollár, P., and Girshick, R., "Mask R-CNN," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2961–2969.

[8] He, K., Zhang, X., Ren, S., and Sun, J., "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.