

Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods

by

Samantha Betts

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Applied Science (Honours)

in the
School of Engineering Science
Faculty of Applied Sciences

© **Samantha Betts 2021**
SIMON FRASER UNIVERSITY
Summer 2021

Copyright in this work rests with the author. Please ensure that any reproduction or re-use
is done in accordance with the relevant national copyright legislation.

Abstract

This research focuses on a novel approach to Simultaneous Localization and Mapping (SLAM) for autonomous mobile robots. SLAM enables the robot to move independently in a geofenced environment by building a dynamic map of its surroundings. The current methods of SLAM are computationally expensive due to the density of keypoints needed to localize the robot correctly, which limits the robot's autonomy. Keypoints are also inefficiently recalculated on a per-frame basis, with the number and location of keypoints not necessarily assigned the same way as the previous frame. This research aims to create an enhanced keypoint detection and processing system using a deep learning model. The focus is on limiting the number of keypoints, typically in the thousands for textured surfaces, to a few hundred. By using semantic segmentation, the keypoints bind effectively to the object perimeter, giving the robot the ability to perform a broader range of tasks in various environments.

Acknowledgments

I would like to thank my academic supervisor, Dr. Ivan Bajić, for providing his expertise in deep learning and giving valuable feedback during the various stages of this project. Additionally, I would like to thank him for taking the time to help me revise and polish this Thesis, making it something I am truly proud of.

Thank you to my technical supervisor, Mr. Victor Lee and his company Nedieon, for providing me with such an exciting and purposeful research project. I have learnt a lot from this experience and have garnered an appreciation for the field of robotics.

I would also like to thank my committee member, Dr. Parvaneh Saeedi, for her valuable feedback and suggestions for methods and improvements to consider for this research's next stage of development.

Lastly, I send thanks to my parents and little sister Taylor for believing in me and supporting me through this whole process.

Contents

Approval	ii
Abstract	iii
Acknowledgements	iv
1 Introduction	1
2 Background	3
2.1 Overview of Convolutional Neural Networks	3
2.1.1 Convolution Layer	3
2.1.2 Activation Layer	5
2.1.3 Pooling Layer	6
2.1.4 Classification/Fully Connected Layer	7
2.1.5 Principles of Training a Model	8
2.1.6 U-Net Architecture for Semantic Segmentation	8
2.2 Overview of SLAM	10
2.2.1 Visual SLAM Pipeline	10
2.2.2 Problems with Existing SLAM	15
2.2.3 Methodology of Using Deep Learning to Improve SLAM	16
3 Methods	17
3.1 Cityscapes Dataset	17
3.2 Data Augmentation	18
3.3 Semantic Segmentation Accuracy Measure	20
3.4 Visual SLAM Accuracy Measure	22
3.5 Training Procedure	22
3.5.1 Input Data	22
3.5.2 Loss function	23
3.5.3 Optimizer	26
3.6 Generation of Masks for keypoint placement	27
3.7 PySLAM	28

3.7.1	Shi-Tomasi Corner Detection	29
3.7.2	Optical Flow Using Lucas-Kanade-Tomasi Tracking	29
3.8	PySLAM Comparison With State of The Art	31
4	Results	32
4.1	Semantic Segmentation Results	32
4.2	Visual Odometry Results	35
5	Conclusion	50
5.1	Thesis summary	50
5.2	Future work	51
5.2.1	Model architecture	51
5.2.2	Improvements to tracking	51
	References	53
A	U-Net Model Training Code	59
A.1	Import Required Libraries	59
A.2	Load Images and Labels	60
A.2.1	Training Images	60
A.2.2	Testing Images	60
A.2.3	Split into Training and Validation Sets	61
A.3	Data Augmentation	62
A.4	Define U-Net Architecture	64
A.5	Model Training	65
A.5.1	Specify Training Parameters	65
A.5.2	Model Summary	66
A.6	Training and Validation Curves	68
A.7	Prediction of Model on Test Images	69
A.7.1	Predict on Test Images From Cityscapes Dataset	69
A.7.2	Predict on Test Images From Kitti Odometry Gray Dataset	69
A.8	Prepare Segmentation Masks For Keypoint Placement	70
B	Copyright Permissions	71

List of Figures

Figure 2.1	Basic CNN Architecture.	4
Figure 2.2	Detailed Example of the CNN System.	4
Figure 2.3	Example Activation Functions.	5
Figure 2.4	Max Pooling Operation.	6
Figure 2.5	Fully Connected Layer.	7
Figure 2.6	Stochastic Gradient Descent.	8
Figure 2.7	U-Net Architecture.	9
Figure 2.8	Visual Odometry Illustration	11
Figure 2.9	Feature Matching Example.	13
Figure 2.10	VO Drift.	14
Figure 2.11	Over definition of keypoints.	15
Figure 2.12	Example of Typical Keypoint Placement for Textured Objects vs Desired Outcome	16
Figure 3.1	Image Augmentation Example.	19
Figure 3.2	Mask Augmentation Example.	20
Figure 3.3	IOU.	21
Figure 3.4	Focal Loss.	24
Figure 3.5	Cityscapes Dataset Pixel Annotation Count.	25
Figure 3.6	Canny Edge Detection and Dilation	28
Figure 4.1	Training and Validation Curves	33
Figure 4.2	U-Net Model Performance on Cityscapes Test Image.	34
Figure 4.3	U-Net Model Performance on Kitti Scene.	34
Figure 4.4	Example Keypoint Placement with Semantic Segmentation Boundaries.	34
Figure 4.5	Summary of Procedure for Segmentation Based Keypoint Filtering	37
Figure 4.6	PySlam Tracking Performance	38
Figure 4.7	Fraction of Average Number of Keypoints Kept with Segmentation Mask.	39
Figure 4.8	Keypoint Jumping	40
Figure 4.9	RMSE with Increasing Dilation Mask Width.	41
Figure 4.10	MAE with Increasing Dilation Mask Width.	42
Figure 4.11	Semantic Segmentation keypoint Tracking Performance	43

Figure 4.12	PySlam Tracking Performance with Randomly Reduced Keypoints	44
Figure 4.13	RMSE Comparison Breakdown.	45
Figure 4.14	RMSE Comparison Breakdown.	45
Figure 4.15	MAE Comparison Breakdown.	46
Figure 4.16	MAE Comparison Breakdown.	46
Figure 4.17	Percentage Time Savings Per Dilation Factor.	48
Figure A.1	Model Summary.	67

List of Tables

Table 3.1	Cityscapes Categories and Corresponding Pixel Label Values	18
Table 3.2	Weights for Each Category Used in the Weighted Dice Loss	26
Table 4.1	Average Number of Keypoints from Best Segmentation Mask Dilation Width	44
Table 4.2	Percent Change between PySLAM RMSE & Segmentation Based Keypoint Filtering RMSE	47
Table 4.3	Percent Change between PySLAM MAE & Segmentation Based Keypoint Filtering MAE	47

Chapter 1

Introduction

Deep Learning for computer vision drastically improves how autonomous mobile robots (AMR's) can discern certain features by using what is known as a neural network—allowing for applications in object detection, classification, and image segmentation. The main difference between the three is the characteristics of interest to extract from the image. Classification alludes to the types of objects in an image. In contrast, object detection works similarly, but the goal is to find the exact location of those classified objects within the image through bounding boxes [1]. Image segmentation creates a pixel map, where each pixel is labelled and associated with a particular class, allowing one to perceive the exact shape of the individual objects within the scene [1], [2]. Deep learning models are trained to determine the correct values for their parameters using backpropagation, which does so by finding a minimum in an error function through the use of an optimizer.

AMR's also rely on Simultaneous Localization and Mapping, otherwise known simply as SLAM, to know where they are concerning other objects in an environment [3]–[5]. The robot essentially builds itself a map of a surrounding environment, relying on visual information from cameras or other sensors such as lasers or sonar [6] to estimate its pose relative to particular landmarks while “simultaneously determining its location within this map” [7]. A robot can quickly identify objects by using mathematically calculated points, known as keypoints, that define corners and edges [5], [8]. However, the current methods that employ this do it in a computationally expensive way.

Nedieon, a start-up company, is currently building and designing AMR for turf management, logistics, agriculture, and health care. To enhance worker safety, Nedieon is using a deep learning algorithm to identify people and objects. This Thesis is an extension of that research to enhance the AMR's position and mapping.

This Thesis will utilize both neural networks and SLAM to establish a more refined keypoint placement to reduce compute power, latency and improve the mapping accuracy. In chapter 2, the necessary background on deep convolution neural networks and visual SLAM is given. Chapter

3 goes over the training process of a semantic segmentation network and how the segmentation maps are used to filter out keypoints for visual SLAM. Chapter 4 demonstrates the found results with corresponding analysis and discoveries made during experiments. Finally, conclusions and recommendations for future work are summarized in chapter 5.

Chapter 2

Background

The success of using deep learning to improve the accuracy of object detection, following an expected result given certain training cases, is due to two advancements: 1) Radically increased datasets and 2) Using GPUs. Due to their array of cores, the strong parallel compute power of the GPUs allowed for the neural network to train quickly on tens of thousands of images. These factors led to improved object detection accuracy. The most well-known application of deep learning for object detection and classification is through the use of convolution neural networks (CNN) [9], [10].

2.1 Overview of Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a network of layers consisting of “neurons” that receive an input image and perform multiple transformations, such as a convolution. A basic CNN architecture is shown in Figure 2.1. A CNN is a feed-forward network, meaning the data is passed forward through the neural network sequentially through each layer. However, unlike conventional feed-forward neural networks, in which neurons are fully connected to all neurons in the previous layer and eventually fully connected at the output layer, CNN’s utilize convolution layers in which an output neuron is only locally connected to a small neighbourhood of the input, known as the receptive field [11]–[13]. The types of layers the data is propagated through include convolution, pooling, activation, and the fully connected layer where the final prediction is made in classification models.

2.1.1 Convolution Layer

The importance of the convolution layers lies in their capability of extracting useful, learnable features from an image. These features can be high-frequency ones such as corners and edges, as

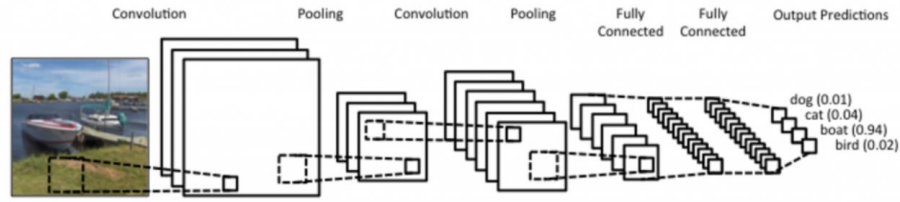


Figure 2.1: Basic CNN Architecture From Reference [13].

well as textures and colour properties [14]. The input into the first convolutional layer is the raw data of the training images as a $n \times n \times m$ array. Where n is the image size and m is the number of channels. A channel number of three signifies an RGB image, and one denotes a grayscale image. Then within the convolution layer, the kernel or filter, moves across the image with an adjustable stride and calculates the perceptron equation, which is as follows [11]:

$$y_j = b + \sum_i w_i x_i \quad (2.1)$$

This formula describes the dot product computed between weights within a kernel and the input region, in which the weights only affect a certain section of the input at a time. In other words, the size of the kernel corresponds to the size of the receptive field, with each entry being a weight that gets multiplied across the entire input width and height [12]. For example, if the input was an RGB image of size 32×32 and the kernel size is 5×5 , then each filter within the convolutional layer would have weights correlated with a $5 \times 5 \times 3$ section of the input as seen in Figure 2.2 [11]. Meaning, there are 75 trainable weights in addition to the bias, b for each filter. More detail of the deep learning training process is outlined in Section 2.1.5.

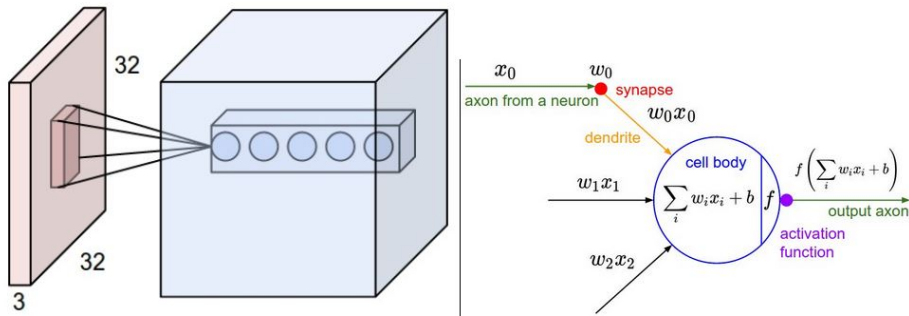


Figure 2.2: Detailed Example of the Convolutional Neural Network System [11].

After the convolution operation has been applied across the entire input, the result is known as a “feature map.” Named so because the perceptron equation has extracted useful features that the model can learn from based on the types of features in specific spatial locations within the input, with each filter within the convolution layer able to detect unique features. Once the feature map has

been created, the model then decides whether to pass this information to the next layer by conditions specified by an activation function, as observed in Figure 2.2.

2.1.2 Activation Layer

Another way a neural network “learns” is by discerning which filters are activated for various class categories [11]. Relating to the neuron analogy, an activation function is used to decide whether to fire the neuron or not [15]. If the output value is above the threshold decided by the activation function, then the signal gets transmitted; if not, then the neuron becomes inactive. Technically there are both linear and non-linear activation functions. However, linear functions should never be used because this would make the network linear and will not perform well for complex problems. Activation functions must also be differentiable in order for backpropagation to occur successfully [15], [16]. Some examples of activation functions are seen in Figure 2.3. The sigmoid function changes large negative output values to near zero and large positive values to near one. However, sigmoid suffers from a vanishing gradient problem, meaning the function’s derivative reaches zero for large positive or negative inputs, which prevents the model from learning during backpropagation. For these reasons, the ReLU function tends to be a more favoured activation function in CNN’s. For ReLU, the output is converted to zero if the input is negative and left unaltered if above. Thus, ReLU does not suffer from vanishing gradients, mitigating the backpropagation error [15]–[17].

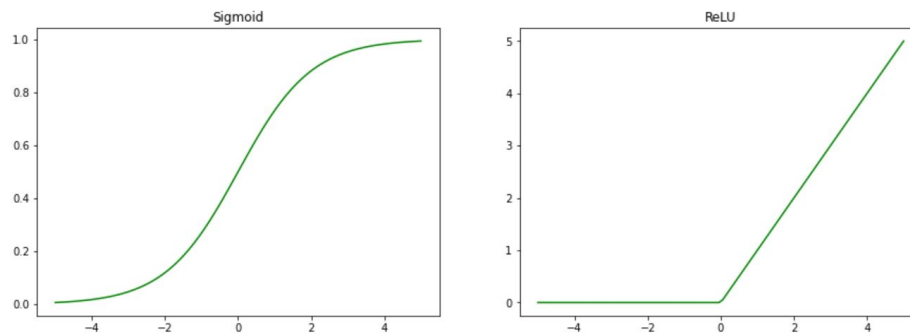


Figure 2.3: Sigmoid vs ReLU Activation Function.

2.1.3 Pooling Layer

The purpose of pooling layers is to subsample the feature map by reducing the width and height of the input while maintaining the original number of channels. Downsampling lessens the compute cost during training by limiting the number of trainable parameters, thus repressing overfitting [11], [12], [18]. One can either use average pooling or max pooling, yet max pooling tends to outperform average pooling due to its ability to suppress noise [18]. Max pooling involves moving a kernel along the input with a chosen stride and within the kernel window only taking the maximum value and discarding the rest. For example, Figure 2.4 shows a max pool filter of size 2×2 and stride 2, which means the input dimensions will be halved, and the entries will be the maximum values seen by the filter as it slides across the input.

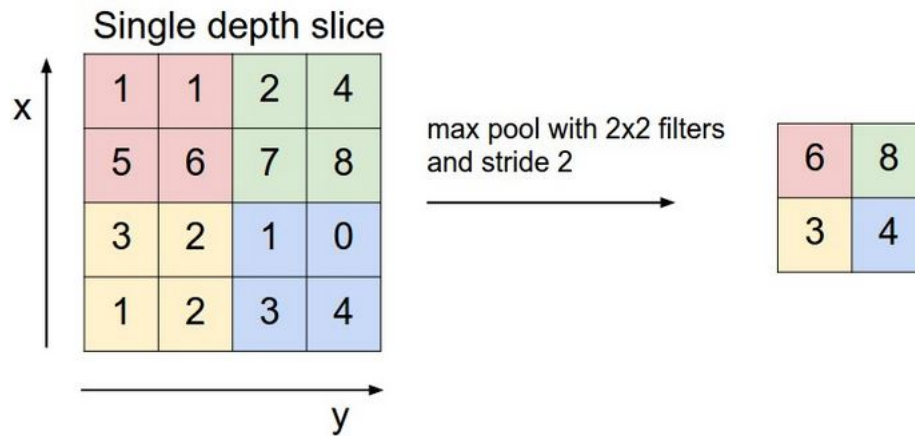


Figure 2.4: Max Pooling Operation Example From Reference [11].

2.1.4 Classification/Fully Connected Layer

Before the input can be classified, the 2D feature map must first be flattened into a 1D vector and does so by going through what is known as the fully connected layer [12]. Each neuron in the fully connected layer is connected to all neurons in the preceding layer, as demonstrated in Figure 2.5. The purpose of converting the 2D feature map into a 1D vector is to represent the class scores as a probability by feeding the vector into an activation function. For multi-class problems, this activation function is typically softmax, and sigmoid for binary classification. The input into the final output layer must be the same as the number of classes in order for a correct categorization [11], [18].

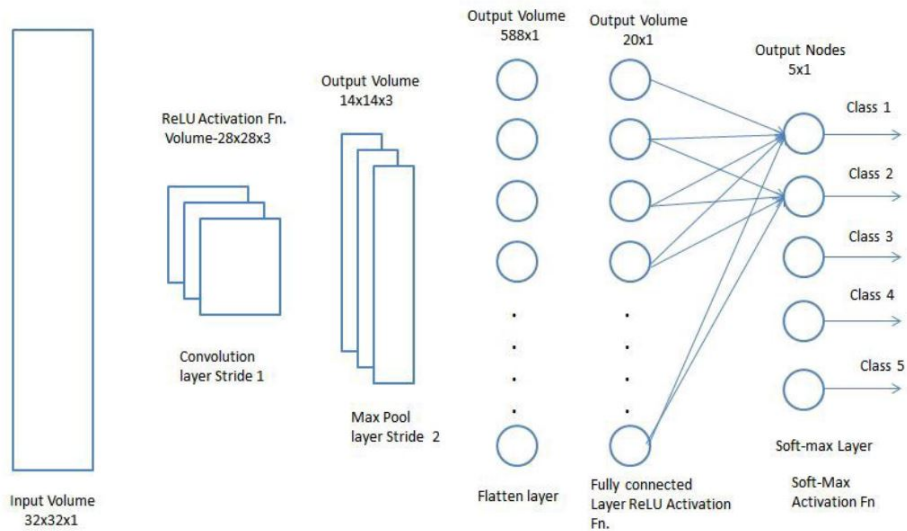


Figure 2.5: Classification Output Example From the Fully Connected Layer [18].

2.1.5 Principles of Training a Model

As stated previously, a neural network can learn by knowing which filters are activated during the training process. To know which filters are activated, the necessary weights and biases must also be known. The training process starts with randomly initialized weights and then goes through the entire feed-forward network once. When a prediction is made at the classification layer, the result is compared with the desired output; known as the ground truth, with the error between the two computed. In order for a correct classification to be made, this error must be minimized, which is done so by backpropagation [19]. Propagation backwards is done by calculating the gradient of the error function with respect to the weights using the chain rule [20], [21]. An optimization algorithm helps aid the minimization of the error function. A typical optimization method is stochastic gradient descent, demonstrated in Figure 2.6.

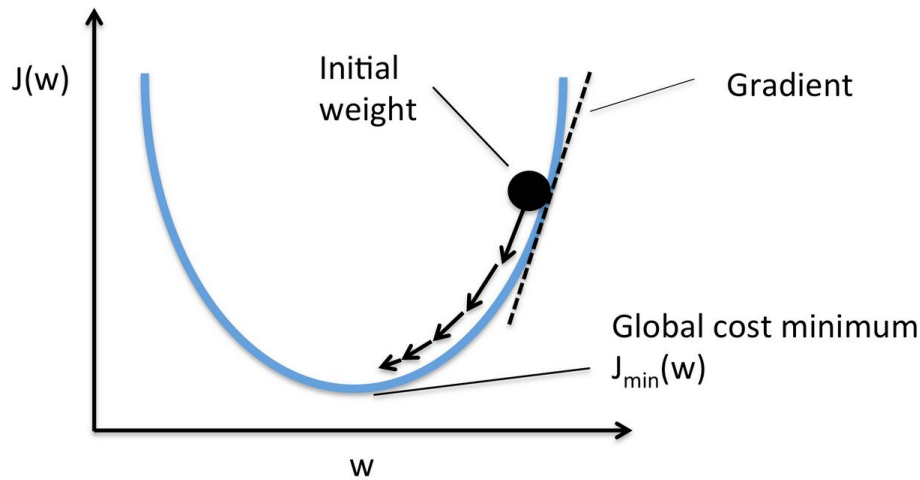


Figure 2.6: Stochastic Gradient Descent Optimization From [22].

During model training, small steps are taken along the gradient descent path, hopefully, towards the global minimum of the error function, and the weights are properly updated. [20], [22]. When the entire dataset is passed through the whole neural network forward and backwards a single time, it can be said that one epoch has been completed [23]. Thus, training the network involves multiple epochs in order to find the lowest loss and highest accuracy of the network.

2.1.6 U-Net Architecture for Semantic Segmentation

Semantic segmentation is an extension of the classification problem, with the added feature of being able to assign individual pixels to corresponding classes. The currently very popular segmentation network was first developed for the segmentation of cells, and this architecture is known as U-Net [24]. As seen in Figure 2.7, the name is due to its U-like shape. Skip-connections within the

network allow for the extraction of localized information, as well as ultra-fine detail, which is what makes semantic segmentation possible [24], [25]. Additionally, U-Net has three main pathways, the encoder/downsampling path, the bottleneck, and the decoder/upsampling path.

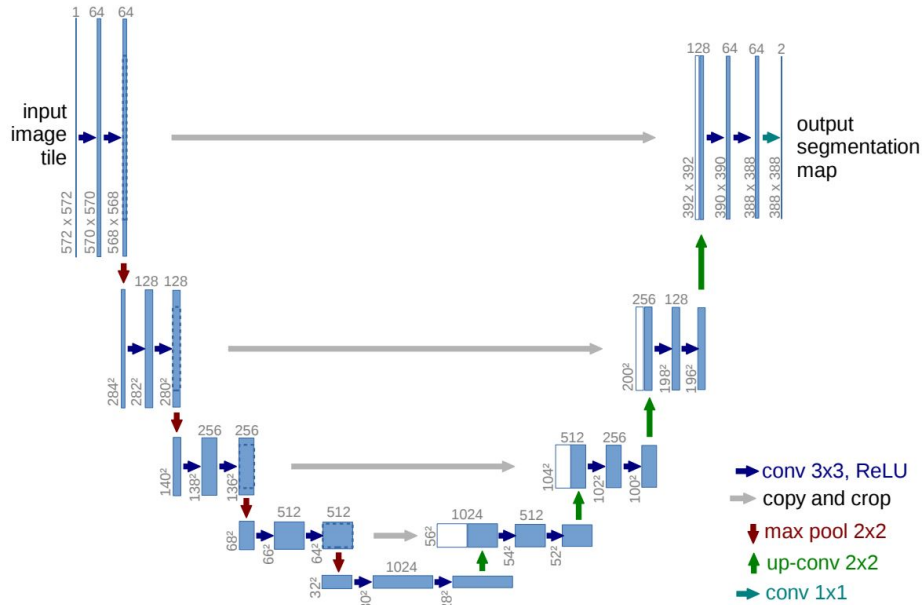


Figure 2.7: U-Net Architecture [24].

Encoder Block

The downsampling path is made up of repeated encoder blocks that learn the necessary features and information for classification. These blocks consist of two convolutional layers of size 3×3 with max pooling applied to the output of the convolution. The max pool layer is a filter of 2×2 with a stride of two, which essentially downsizes the input size by two, which is why this section is referred to as the downsampling path. Additionally, the ReLU activation function is used at the end of each convolutional block. The output of the second convolutional layer is what gets concatenated to the symmetrically opposite decoder block; this concatenation is the skip connection feature. The number of encoder blocks can be increased or decreased for the application in various problems, but the original paper uses four encoder blocks. The output of the last encoder block enters the lower well of the “U,” more commonly known as the bottleneck section or bridge [24], [26].

Bottleneck

The bottleneck or bridge of the U-Net is just two convolution layers of size 3×3 , similar to the encoder block. However, no max pooling is applied to the output of this section. The only function of the bridge is the union of the downsampling and upsampling paths [24], [26].

Decoder Block

Following the output of the bottleneck, which must now be upsampled, is a series of repeated decoder blocks. Decoder blocks require both the input array from the previous layer and the concatenated feature from the symmetrically opposite encoder block, doubling the number of features. Once concatenated, the combined data goes through two 3×3 convolution layers, equivalent to the downsampling path. The input size is also doubled by either a convolutional transpose or upsample function. This whole process allows for the generation of the predicted segmentation mask [24], [26].

2.2 Overview of SLAM

As mentioned previously, SLAM can be accomplished using a variety of sensors such as lasers, odometers, or sonar. This Thesis focuses on utilizing visual SLAM, in which different cameras such as monocular, stereo, or RGB depth can be used to extract the path travelled based on the pose change between a series of captured frames. Three different map models can be applied to the problem, feature-based, appearance-based or hybrid, the former being implemented in this research. SLAM is an extension of visual odometry, which means that the environment must have sufficient illumination and enough scene correspondence between two successive frames to work successfully. The images must also have an occurrence of more static objects than dynamic, with good texture so that perceived movement can be surmised [27], [28].

2.2.1 Visual SLAM Pipeline

For SLAM, images are taken at every time step, k . The position change between consecutive camera frames is computed with the following rigid body transform:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}, \quad (2.2)$$

with $R_{k,k-1}$ being the rotation matrix, and $t_{k,k-1}$ the translation vector.

All consecutive motions are stored in a set of T matrices, $T_{1:n} = \{T_1, \dots, T_n\}$. T_k is a list of all transformations, where relative transformations are calculated between each image pair I_k and I_{k-1} . All camera poses are stored in the set $C_{0:n} = \{C_0, \dots, C_n\}$, which is obtained by extracting the direction of motion in small intervals during each pose step. The current camera pose C_n is found by the equation,

$$C_n = C_{n-1}T_n \quad (2.3)$$

where C_{n-1} is the previous image and T_n is the transformation of the camera. Figure 2.8 shows that the full path of the camera can be obtained via concatenation of all observed transformations $T_{1:k}$ [28].

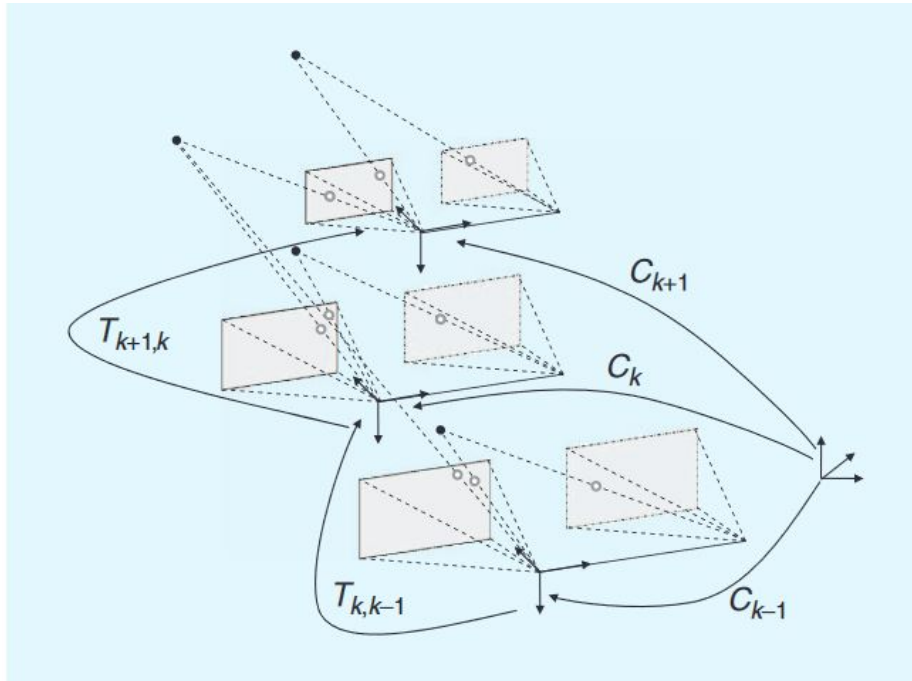


Figure 2.8: “An Illustration of The Visual Odometry Problem” [28].

Camera Modelling and Calibration

In order for SLAM to be as accurate as possible, the camera must be calibrated before use. There are three different camera models to use in SLAM. The model used for this project is the perspective model, but one could also work with omnidirectional or spherical models. For the perspective projection, typically, a pin-hole camera model is assumed, in which a single viewpoint projects onto the focal plane. The next step is to calibrate the camera in order to quantify the intrinsic and extrinsic parameters of the camera [28]. Intrinsic parameters are those such as the focal length (f_x, f_y) and the optical centers (c_x, c_y) . Extrinsic parameters comprise translation and rotational vectors. Additionally, the distortion coefficients $(k_1, k_2, p_1, p_2, k_3)$ must be known. Finally, by following the

perspective model and using the acquired camera parameters, the camera matrix can be computed as follows:

$$\text{Camera Matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.4)$$

This matrix is specific to each camera and translates the 3D world to a 2D image. This process is called image correspondences and is applied differently for three possible approaches, which are 2D to 2D, 2D to 3D, or 3D to 3D [28], [29].

Calibration is usually done by taking images from various viewpoints of a checkerboard pattern in which the dimensions of each square are known. A least-squares minimization method is then used to get the calibration parameters. The calibration step is crucial to ensure that accurate transformations between pose steps are calculated. If tangential or radial distortion is part of the camera, it can be removed using the camera matrix [28], [29].

Feature Detection and Matching

In order to track the pose change between frames, there should be discernible features between each image that can be followed from one picture to the next. “Features” or “keypoints” are interest points that are either indicative of texture or have a noticeable aspect of border lining a region of interest. Such examples of keypoints are edges, corners, or blobs; in other words, it is anywhere that a change in the orientation of a border can be observed [30], [31]. Features are ascertained by what is known as a feature detector. However, the problem with this approach is that differences between keypoints are indistinguishable.

Feature descriptors function as a method for assigning characteristics about a known feature. Attributes of these points usually define a relationship about the area of pixels around the feature, such as the directional change of the image intensity or gradients. When choosing these keypoint descriptors, they must follow certain constraints such as being independent of keypoint position, scale and rotation invariant, as well as invariant to various illuminations and perspectives [32]. SIFT, SURF, ORB, and BRISK are some examples of feature detectors that also function as feature descriptors [31].

Now that there is a way to store and compare these features from images, feature matching can measure the direction and distance of motion between sequential frames. Figure 2.9 demonstrates the matching of ORB features between two frames in which the perspective of the camera has changed. Features are matched when their descriptors are “similar,” an example of similarity is keypoints with the smallest Euclidean distance between their feature descriptors. However, many

points can be compatible with this “similar” point, which significantly increases the computational cost, especially for images with a high density of keypoints [27], [31].

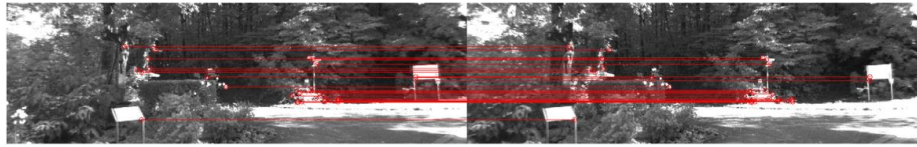


Figure 2.9: Feature Matching Example using ORB Descriptor and Detection From Reference [31].

Motion Estimation

Now that features between frames can be tracked, the motion of the camera can be estimated by computing the transformation matrix in (2.2). As previously mentioned, the 2D-2D image correspondences are used in this thesis. In order to get the translation and rotation vectors for (2.2), the Essential Matrix needs to be computed. The Essential Matrix establishes the spatial correlation between two images I_k and I_{k-1} and also encompasses the camera trajectory information. A scale factor is needed to acquire the translation between consecutive frames properly. The essential matrix is as follows,

$$E = \alpha \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} R \quad (2.5)$$

with t corresponding to the translation in the x, y, z direction, R being the rotation vector, and α the scaling factor [28]. Calculating the essential matrix involves working in the epipolar plane system and using “epipolar constraints” [28], [33]. A common solution to the essential matrix is the five-point algorithm, or 5-point RANSAC [28], [34], [35].

RANSAC is an iterative process that entails randomly sampling the number of pairs of coordinates needed to estimate the correspondence between two sets of keypoints. Each estimated transformation is applied to all points in the first set and then compared with the second set [36]. Transformed points that are “close” to their matches are called inliers. Calculated inliers are then stored, and this pairing and comparison step for transformation estimation is repeated until an upper limit of iterations is reached. Finally, the set with the maximum number of inliers is decided to be the final result [27], [35].

Local Optimization

When trying to estimate the trajectory travelled by the camera, errors accumulate from both camera pose (C_k) and transformation ($T_{k,k-1}$) approximations made between each pair of sequential images, as seen in Figure 2.10. These collections of errors are commonly referred to as visual odometry drift because they cause the approximated path to digress from the absolute route [27].

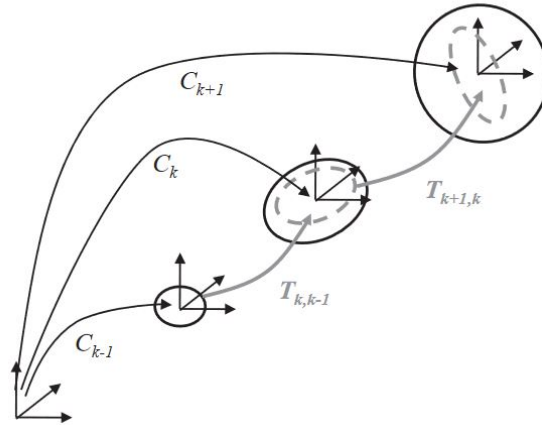


Figure 2.10: Visual Odometry Drift From the Accumulation of Camera Pose Error and Transformation Uncertainty [27].

Two possible optimization methods can be used to refine the final map. The first option is to correct the map after the entire desired trajectory has been traversed. This approach is known as windowed bundle adjustment and requires all camera poses and features to be re-examined while minimizing the reprojection error [27], [28]. Conversely, Kalman or particle filters can be used to adjust the map while the images are being taken, gradually fixing the pose estimate each time new features are detected [36].

Loop Closure

The only difference between visual SLAM and visual odometry is closing the loop [27]. Loop closure is the basis of guaranteeing the global consistency of the built map [36]. The loop can be closed when the robot or camera has returned a location or image it has already seen. In order to confirm if this frame was viewed beforehand, one could relate the current image to previous ones, find correspondence between the present location within the map to other map positions, or compare the image to the map [36].

2.2.2 Problems with Existing SLAM

There are many significant problems with all of the current SLAM models. One issue is that the algorithm has to recompute the keypoints on a per-frame basis, making it very computationally expensive. In addition, keypoints are not necessarily the same from frame to frame when the robot is moving. In other words, the number and position of keypoints within various objects are calculated to be different in each frame, and there is no way for the robot to distinguish between the keypoints of those different objects. The density of these points, particularly in the case of textured surfaces, is also excessive. This process does not add any additional information; rather, it bogs the computer down. Figure 2.11 demonstrates both of these problems; the right image shows a certain number of keypoints that then changes after camera movement in the right image. Another issue is that if the AMR changes directly, for example, 180 degrees, it will lose all its keypoints with a forward-facing camera and hence its position within the map. Currently, most modern SLAM approaches use thousands of keypoints per frame calculated by Harris or Shi-Tomasi methods, and a small dropout of less than ten percent can cause the SLAM to lose localization. These multiple factors contribute to making it more difficult to close the loop and localize where the robot is.

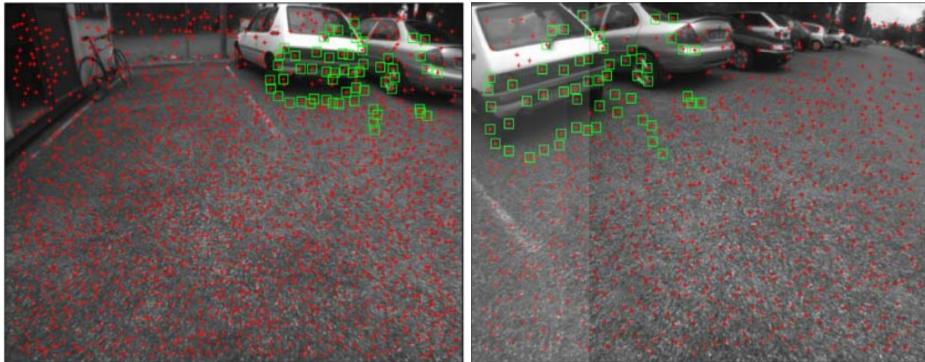


Figure 2.11: Over definition of keypoints that Change from Different Viewpoints [37].

2.2.3 Methodology of Using Deep Learning to Improve SLAM

The goal of this thesis is to integrate deep learning into SLAM. Rather than alter the current structure of the SLAM pipeline, the idea is to increase the framerate and lessen the amount of time it takes to match keypoints between frames. Since SLAM relies on RANSAC, which iteratively repeats until it reaches “the best” answer, and mutually consistency matching, the computation power required is excessive. For example, the Shi-Tomasi feature detector applied to Figure 2.12 a) is shown in Figure 2.12 b). The points are placed heavily around the image, especially in textured regions like the road and vegetation. By using the segmented image to limit the number of keypoints and strictly binding them to the perimeter of the object, the time taken between each operation should lessen since there are fewer keypoints to work with. Figure 2.12 c) is a manually labeled pixel map for Figure 2.12 a), from the KITTI Vision Benchmark Suite Semantic Segmentation Dataset [38]. When the same feature detector is used on the segmented image, the placement of points is more focused and, more importantly, heavily reduced in number, which should greatly increase the algorithm’s speed.

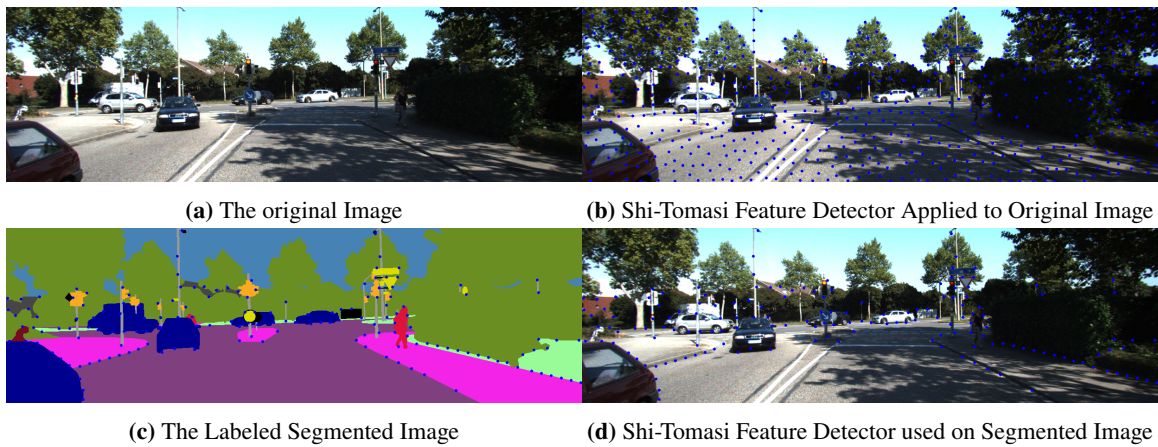


Figure 2.12: Example of Typical Keypoint Placement for Textured Objects vs Desired Outcome, Modified From [38]

Chapter 3

Methods

This chapter explains the dataset used to train our U-Net and methods for data augmentation to increase the training and validation dataset size. Additionally, metrics used to measure the accuracy of U-Net and SLAM performance are specified. The training procedure for the model and the choice of optimizer, loss function, and input image size used are also described. Finally, approaches taken to generate the masks used for keypoint placement are outlined, followed by a brief description of PySLAM, a Python framework implementation of SLAM used for this Thesis.

3.1 Cityscapes Dataset

Visual SLAM can be used either indoors or outdoors. For this project, outdoor scenes were selected for the eventual use of this enhanced SLAM method on golf courses. As of now, for proof of concept, the readily available street scene dataset Cityscapes was used. The Cityscapes Dataset contains 5000 finely annotated images and 20 000 coarse annotations [39]. Annotations can also be referred to as a pixel map or mask and serve as the ground truth labels during training of the segmentation model. Pixel maps are grayscale images in which one value corresponds to a category. The pictures collected are from 50 different cities, either in Germany or from surrounding countries. Cityscapes have classes which include road, car, sidewalk, and more, as shown in Table 3.1. The background class is added if any pixel in the annotated dataset is missing a label. These pixels will then automatically be assigned to the background class. In order to ensure a more precise boundary around objects during training of the segmentation model, only the finely annotated annotations are used in this research. Additional training and validation data are included by performing data augmentation techniques on the preexisting 5000 images and masks.

Category	Label ID
unlabeled	0
ego vehicle	1
rectification border	2
out of roi	3
static	4
dynamic	5
ground	6
road	7
sidewalk	8
parking	9
rail track	10
building	11
wall	12
fence	13
guard rail	14
bridge	15
tunnel	16
pole	17
pole group	18
traffic light	19
traffic sign	20
vegetation	21
terrain	22
sky	23
person	24
rider	25
car	26
truck	27
bus	28
caravan	29
trailer	30
train	31
motorcycle	32
bicycle	33
license plate	34
background	35

Table 3.1: Cityscapes Categories and Corresponding Pixel Label Values

3.2 Data Augmentation

Basic augmentation techniques are implemented through the Python library Albumentations, which provides a quick and easy way to perform a wide variety of transformations [40]. The goal of augmentation is to expand the training and validation datasets by using existing labels and applying transformations so that the model can find expected predictions despite changes in object positions and viewpoints. Expanding the dataset to include more images prevents overfitting of the model. Simple transformations such as horizontal flip, affine, and blurring were used during the augmenta-

tion process. Complex transformations that would overly distort the image are avoided so that the model does not get confused. If transformations deform the image past the point of recognizing an object for what it should be, then the model will misrepresent a majority of the pixels of an image as being a part of a false class.

Figure 3.1 demonstrates some basic transformations applied to an example image. Horizontal flip shows the original image being vertically inverted about the x-axis. The bottom left image of Figure 3.1 shows a random affine transformation using the function *ShiftScaleRotate*. The functionality of *ShiftScaleRotate* is to apply translation, scale, or rotation affine transformations at random. These transforms are done up to a limit specified during the augmentation process. For the lower-left image in Figure 3.1, the parameters are set with a scale limit of 0.5, shift limit of 0.1, and a rotation limit of zero. A Blur transform is also randomly applied to the original image as seen in the bottom right image of Figure 3.1. With the Albumentations library, a transform list is setup with specified transformations to be applied sequentially with a certain probability of occurrence for each image put through the list. A transformation in a group of several possible options can also be individually applied at random by using the *OneOf* function within the sequential list [41].

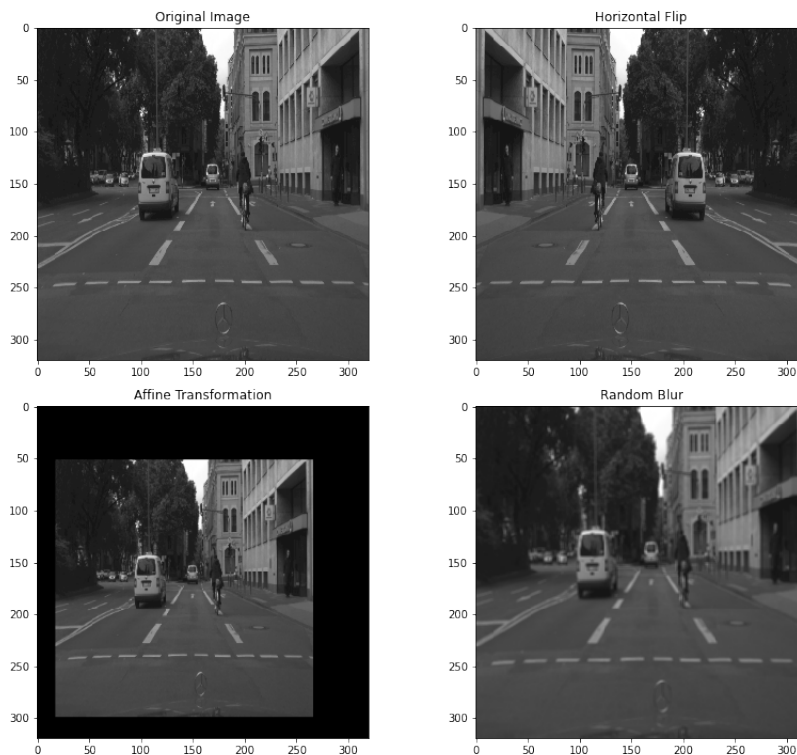


Figure 3.1: Image Augmentation Using Horizontal Flip, Affine, and Blur Transformations.

For semantic segmentation problems, the same augmentation steps and transformations must be applied identically on both the images and masks to guarantee that they match every time

the augmentation occurs [42]. Figure 3.2 shows how the same transformations implemented in Figure 3.1 are performed on the original segmentation map.

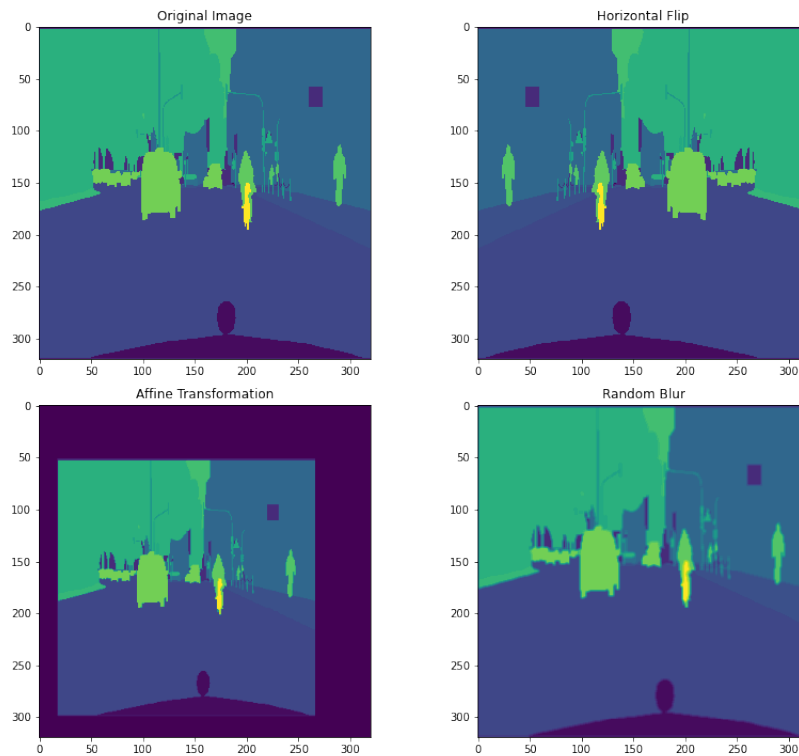


Figure 3.2: Mask Augmentation Using Horizontal Flip, Affine, and Blur Transformations.

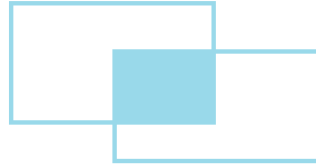
3.3 Semantic Segmentation Accuracy Measure

Most classification and object detection networks use “accuracy” as a metric to measure the model’s performance. However, this benchmark is impractical for semantic segmentation problems in which categories are unbalanced [43]. In other words, the background class usually dominates 80-90% of the pixels, so the accuracy automatically starts off to be very high since it is easy to predict background pixels correctly. Hence, it is more beneficial to use a metric that prioritizes the foreground classes by relating the intersection and union for each pixel category. These metrics that are commonly used in segmentation tasks are called Intersection Over Union (IOU), and F1 score [43], [44].

IOU is defined as,

$$IOU = \frac{\text{Area of Intersection}}{\text{Area of Union}} \quad (3.1)$$

in which the intersection is the overlap between the predicted label and ground-truth, and the union is the whole area covered the ground truth and prediction as seen in Figure 3.3. For IOU, the goal is to maximize intersection and reduce union.



Area of Intersection



Area of Union

Figure 3.3: Intersection Over Union.

Another useful metric is the F1 score, which equally maximizes both precision and recall. The equations for precision and recall are as follows [43], [44] :

$$Precision = \frac{TP}{TP+FP} \quad (3.2)$$

$$Recall = \frac{TP}{TP+FN} \quad (3.3)$$

TP stands for true positive, *FP* false positive, *FN* false negative, and *TN* is true negative. True positive means a prediction correctly matches the ground truth of a class, whereas a true negative represents a pixel that was correctly classified as not being part of that class. On the other hand, a false positive means a pixel was classified as being part of the actual class when it should have been false, and a false negative is predicting a pixel as not being part of a class when it is actually true.

The F1 score, otherwise known as the Dice coefficient, can then be written as the harmonic mean of precision and recall [43], [44].

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3.4)$$

Therefore, F1-score is a helpful metric when wanting to quantify the false positive and false negative scenarios [44].

3.4 Visual SLAM Accuracy Measure

Since we know the x, y, z position of the camera of each frame from the perspective camera model (2.4) we can then use an odometry ground truth to find the error of the SLAM algorithm. The image and ground truth data used in this Thesis are from the KITTI Vision Benchmark Suite Visual Odometry/ SLAM 2012 data set [45]. Precise ground truth information is obtained by tuning and coordinating cameras, a Velodyne laser scanner and a fusion of localization sensors such as a Global Positioning System (GPS), Inertial Measurement Unit (IMU) and a Real-time Kinematic Position (RTK) system. Therefore, the error of the estimated motion is just the difference, i.e the absolute error, between the supposed camera pose transformation and proper ground truth position [34].

Using the approach of the Canny edge mask on the segmentation boundaries, each frame will have different number of keypoints, and thus each frame will exhibit unique errors. In order to quantify the overall result of this method the Root Mean Square Error (RMSE) is used. This value will be used to compare the original SLAM with the one using the mask-based method in this research. Since we have errors in both $x, y,$ and z parameters the total RMSE would be,

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (x_{\text{true}} - x_{\text{predicted}})^2 + (y_{\text{true}} - y_{\text{predicted}})^2 + (z_{\text{true}} - z_{\text{predicted}})^2} \quad (3.5)$$

where the variables sub-scripted with true represent the ground truth value and those sub-scripted with predicted indicate the estimated x, y, z position [46].

Another metric to evaluate error is the Mean Absolute Error (MAE), as this measurement is less sensitive to outliers [47]. The formula for average MAE across $x, y,$ and z is,

$$MAE = \frac{1}{3n} \sum_{i=1}^n |x_{\text{true}} - x_{\text{predicted}}| + |y_{\text{true}} - y_{\text{predicted}}| + |z_{\text{true}} - z_{\text{predicted}}| \quad (3.6)$$

3.5 Training Procedure

3.5.1 Input Data

To start, 2975 finely annotated images from the Cityscapes train folder are read in as grayscale input. Images are resized to 320×320 and grayscale images are used due to memory constraints of the

8GB NVIDIA GeForce GTX 1080Ti GPU used during training. A size of 320×320 was also found to yield better results for being able to discern smaller represented pixels such as poles and road signs. The 2975 images and labels are split 80% into a train set and 20% in a validation set, yielding 2380 and 595 images respectively. Next, the masks are one-hot encoded to change the categorical label to a numerical input for proper employment in U-Net [48]. A boolean column is created for all classes in which the column number corresponding to the pixel id has a value of 1 at the relating pixel location, and is zero in every other column. Code for this process is shown in Appendix B.

Training and validation images are then augmented using the procedures mentioned in Section 3.2. A sequence of random transformations is applied to each image once, expanding the total number of training images to 4760 and validation images to 1190. The full transform list with their corresponding probabilities and set limits are available in Appendix A.3.

Finally, the data is passed to the U-Net architecture shown in Figure 2.7 with input image of size $320 \times 320 \times 1$ for training. Due to the memory constraints of the GPU, a maximum batch size of 16 was used to speed up training. The default number of epochs was set to 100 with early stopping enabled. Early stopping was chosen to monitor validation loss, meaning the model’s training stops when the validation loss ceases to decrease. In addition, a Patience of 5 epochs was set, which means if the early stopping checkpoint is reached, five epochs will run afterwards to verify that the decrease in loss has stopped. Predictions were made on the 500 images from the Cityscapes test folder, and results will be shown in Chapter 4.

3.5.2 Loss function

This research utilizes a combination of focal loss with a weighted Dice loss to improve the training of the model. By using the sum of these two loss functions, we exploit the benefits of both, especially when dealing with imbalanced class data [49].

Focal Loss

Focal loss is a continuation of cross-entropy loss with the added feature of being able to handle unbalanced datasets by down-weighting predictions that have high confidence and focusing on hard to classify examples [49], [50]. The formula for focal loss is,

$$FL = -\alpha(1 - p)^\gamma \log(p) \tag{3.7}$$

where α controls the weighting of the well-classified examples [49], p is the class prediction probability of the model, $(1 - p)$ is the modulating factor [51], and γ is the focusing parameter

[50], [52]. Different γ values can be seen in Figure 3.4. The more disproportionate and difficult the problem is, the higher the γ value should be.

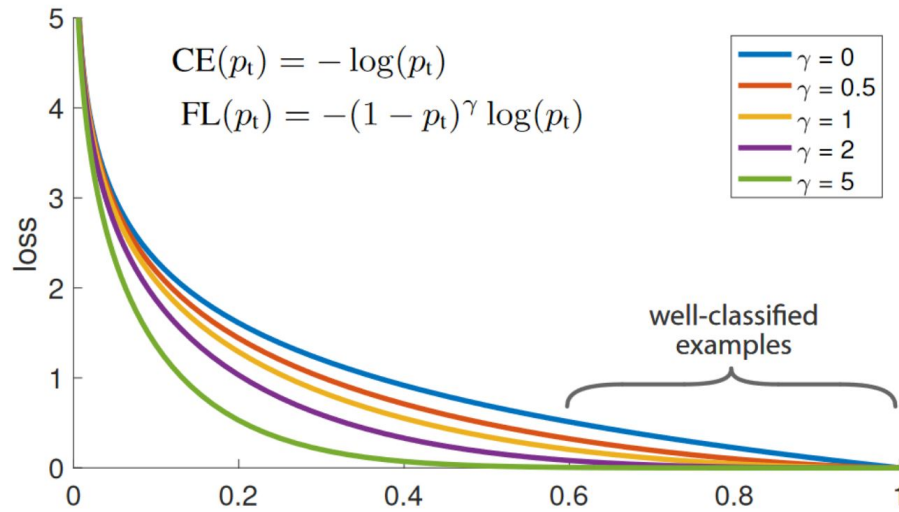


Figure 3.4: Focal Loss Curve From Reference [50].

The code to emulate this loss function is from the python library segmentation models [53], which uses a default α value of 0.25 with $\gamma = 2.0$ [52].

Weighted Dice Loss

Dice Loss is simply the negative of the F1-Score from (3.4) in Section 3.3 [51], [52]. The negative sign is needed because loss functions should be minimized rather than maximized as metrics are. Similar to focal loss, weights can be used to equalize imbalanced data [49]. The number of finely annotated pixels for 29 of the 35 Cityscapes categories are shown in Figure 3.5. Classes like road, building and vegetation are heavily represented, whereas classes such as traffic lights and guard rails are less so. Using the class per pixel information, weights for each class are defined by using the person class as a baseline, with its weight being 1. Categories with a pixel per class amount larger than the person class are down-scaled by taking the rough ratio between the number of pixels per class. For example, the number of person pixels is approximately 3/4 that of the road class so a weight of 0.7 is assigned for the road class. The same reasoning applies if the number of pixels are less than the person class, then the weight is scaled upwards above a value of 1. Classes in which the number of pixels wasn't reported were given a weight of 1, and the background class was down-weighted to 0.5. All weights can be seen in Table 3.2.

The total loss used for training the U-Net is then,

$$\text{Total Loss} = \text{Focal Loss} + \text{Weighted Dice Loss} \quad (3.8)$$

and is implemented using the segmentation models library [53] as seen in Appendix A.

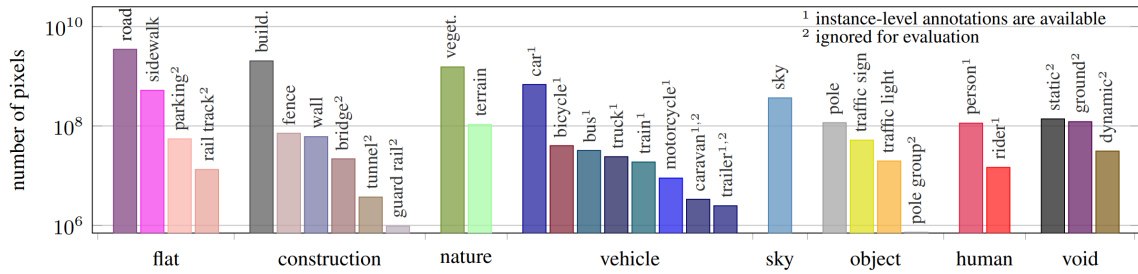


Figure 3.5: “Number of finely annotated pixels (y-axis) per class and their associated categories (x-axis)” [39].

Category	Weight
unlabeled	1
ego vehicle	1
rectification border	1
out of roi	1
static	1
dynamic	1
ground	1
road	0.7
sidewalk	1.1
parking	2.2
rail track	2.9
building	0.8
wall	2.2
fence	2.2
guard rail	4
bridge	2.5
tunnel	3.4
pole	1
pole group	5
traffic light	2.5
traffic sign	2.3
vegetation	0.9
terrain	1
sky	1.2
person	2
rider	3
car	1
truck	2.3
bus	2.3
caravan	3.4
trailer	3.5
train	2.3
motorcycle	3.2
bicycle	2.3
license plate	0.5
background	0.5

Table 3.2: Weights for Each Category Used in the Weighted Dice Loss

3.5.3 Optimizer

Adaptive moment estimation (Adam) was the optimizer of choice for training the U-Net model. Due to its fast convergence and low computation cost, Adam is a popular choice. While stochastic gradient descent has a steady learning rate, Adam evaluates with adaptive learning rates, allowing it to handle sparse gradient complications. Addedly, nearly no hyperparameter adjustment is needed [54], [55].

3.6 Generation of Masks for keypoint placement

In order to bind the detected features to the outline of the segmented objects, a mask can be defined. This mask is of type CV_8UC1 and contains only 0 and 255 [56]. When the mask is applied, pixels that fall in the regions of the mask that contain 0 are ignored if a feature is detected at that pixel. Values within the 255 regions of the mask retain the features found at those pixel locations. The mask is created by applying a Canny edge detector to the segmented image.

A Canny edge detector allows one to remove edges that are not important and thus also acts as a noise suppressant during edge detection. Canny edge detection usually builds off a standard edge detector such as the Sobel operator. The first step is to remove noise by smoothing the input image with a Gaussian filter. Then an intensity gradient calculation is done. The OpenCV function used in this project finds the intensity gradient of the image via a Sobel filter with a 3×3 kernel size. The x and y derivatives are computed with a convolution using the following filters [57]:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad (3.9) \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad (3.10)$$

The gradient is then found with the magnitude of G_x and G_y ,

$$G = \sqrt{G_x^2 + G_y^2} \quad (3.11)$$

The direction of the edge must also be computed with the equation,

$$\theta = \arctan\left(\frac{G_y}{G_x}\right) \quad (3.12)$$

with there being only four potential angles of 0 deg, 45 deg, 90 deg, and 135 deg. Any angle that falls between two possible categories is rounded accordingly [57], [58]. Next, non-maximum suppression is applied to the gradient image to thin the edges by finding local maxima across each edge. Finally, hysteresis thresholding is done to decide which edges to keep. Hysteresis thresholding incorporates two threshold levels. Image gradient values above the top threshold are preserved as an edge, and values below the lower threshold are discarded. Points that lie within the two thresholds are only maintained as an edge if it is connected to an area above the upper threshold [57], [58].

Figure 3.6 demonstrates how Canny edge detection is used on a U-Net prediction example with an upper limit of 15 and a lower limit of 5. If more keypoints are desired, one could expand the mask appropriately using dilation with varying kernel sizes [58]. As viewed in the bottom right image of Figure 3.6, dilation is done on the image of detected edges with a 10×10 kernel to expand the

edge borders. This resulting image can then be used as a mask to place keypoints along only regions around detected objects.

This process is done for all images in the Kitti odometry gray dataset for various sequences [45] as this is the data used for computing the visual odometry/SLAM trajectory.

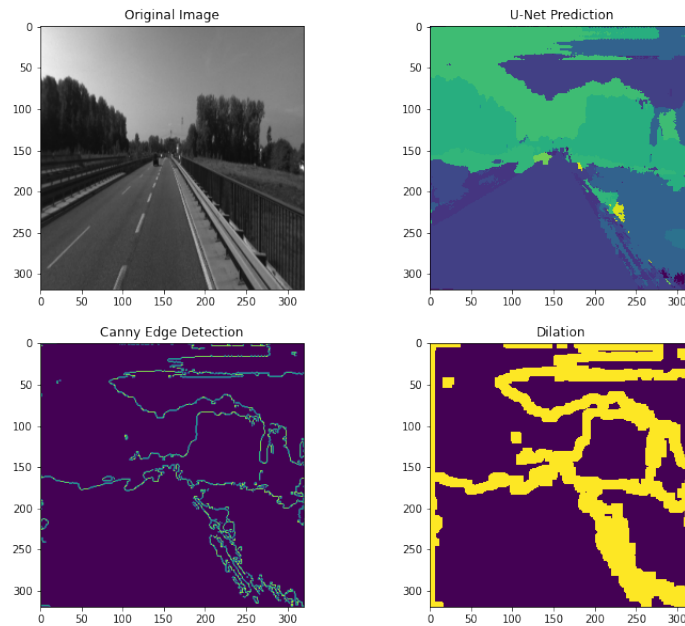


Figure 3.6: Canny Edge Detection and Dilation of Semantic Segmentation Border.

3.7 PySLAM

PySLAM provides the general pipeline of monocular SLAM and visual odometry outlined in Section 2.2. This framework is purely for proof on concept purposes and is not optimized for the best tracking accuracy or suited for real-time use at this time [34]. This Thesis utilizes the visual odometry function, which has most of the pipeline with the exemption of point-triangulation or windowed bundle adjustment. As explained in Section 2.2, no loop closure will be conducted in visual odometry. PySLAM provides a flexible architecture for the allowance of interchanging feature detectors and descriptors. A variety of feature matching techniques such as FLANN Or BRUTE can be used for tracking, as well as optical flow methods [59]. The latter will be used for this project in conjunction with the Shi-Tomasi feature detector for a tracking technique known as the Kanade-Lucas-Tomasi feature tracker (KLT tracker).

3.7.1 Shi-Tomasi Corner Detection

Shi-Tomasi is a corner detector similar to Harris, but more efficiently computes a corner by only accepting it if the computed value is above,

$$R = \min(\lambda_1, \lambda_2) \quad (3.13)$$

In other words, a corner only exists if the R value is greater than a specified threshold value [60], [61]. The OpenCV implementation of Shi-Tomasi is called *goodFeaturesToTrack*. This function allows a quality level and minimum detection distance to be set. The quality level is between 0 to 1 and specifies an upper limit for which to reject corners. Whereas the minimum distance determines the minimum Euclidean distance for which corners are computed between [61]. For PySLAM, *GoodFeaturesToTrack* by default uses a quality level of 0.01 and a minimum distance of 3 [34]. Additional code is written for integration into the PySLAM implementation to pass in the generated segmentation masks for the placement of keypoints within the preexisting visual odometry framework.

3.7.2 Optical Flow Using Lucas-Kanade-Tomasi Tracking

Lucas-Kanade-Tomasi tracking is a method that utilizes both optical flow and Shi-Tomasi corner detection to identify features and follow them from frame to frame [62], [63]. Motion is obtained from spatial and temporal image brightness variations, i.e. image gradients. This method only works if all pixels have consistent brightness, there is small movement between frames, and under the assumption that the neighbours around a pixel are displaced the same. The Lucas-Kanade approach starts with an image $I(x, y)$ that is displaced by a displacement vector (u, v) . Once again, assuming small motion, the displaced image is expressed as,

$$H(x, y) = I(x + u, y + v) \quad (3.14)$$

The formula is computed at each pixel in the frame and is then solved with Taylor expansion to yield the Lucas-Kanade equation [62], [63]:

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix} \quad (3.15)$$

where I_x is the image derivative along x, I_y is the y image gradient, and I_t represents the image difference between two sequential frames. Therefore, the eigenvalues of this matrix can be computed in order to compute the flow vector or displacement (u, v) .

PySLAM employs these methods to compute the current camera pose relative to the preceding frame with formula (2.2). In addition, the scaling factor for translation estimation is obtained by using the ground truth inter-frame scaling from the Kitti odometry gray dataset [34], [45].

Using the Kitti odometry gray dataset requires knowing the calibration parameters of the camera used. 11 sequences are used for testing. Sequences 00 to 02 have a image width of 1241×376 with camera calibration parameters of:

$$f_x = 718.856$$

$$f_y = 718.856$$

$$c_x = 607.1928$$

$$c_y = 185.2157$$

$$k_1 = 0.0$$

$$k_2 = 0.0$$

$$p_1 = 0.0$$

$$p_2 = 0.0$$

Sequence 03 has an image width of 1242 and height of 375 with the following camera calibration parameters:

$$f_x = 721.5377$$

$$f_y = 721.5377$$

$$c_x = 609.5593$$

$$c_y = 172.854$$

$$k_1 = 0.0$$

$$k_2 = 0.0$$

$$p_1 = 0.0$$

$$p_2 = 0.0$$

For the case of sequences 04 to 10, the camera calibration parameters are with a camera width of 1226 and height of 370. The distortion parameters are as follows [34]:

$$f_x = 707.0912$$

$$f_y = 707.0912$$

$$c_x = 601.8873$$

$$c_y = 183.1104$$

$$k_1 = 0.0$$

$$k_2 = 0.0$$

$$p_1 = 0.0$$

$$p_2 = 0.0$$

3.8 PySLAM Comparison With State of The Art

The current state of the art monocular visual SLAM architecture is ORB-SLAM2 [64], [65]. The field of SLAM is rapidly changing and many papers use ORB-SLAM2 as the baseline for the comparison of their technique due to its accurate and robust performance in dynamic environments [66], [67]. With respect to PySLAM, since it is an open source framework, it will not be the “best” possible SLAM architecture. Comparing the absolute trajectory error or RMSE for the Kitti Dataset between PySLAM and ORB-SLAM2, it was found that PySLAM has 10× the amount of absolute trajectory root mean square error [68]. However, the main difference in error could be because ORB is used as a feature detector and descriptor, which is more robust than the Shi-Tomasi keypoints used in this thesis.

Chapter 4

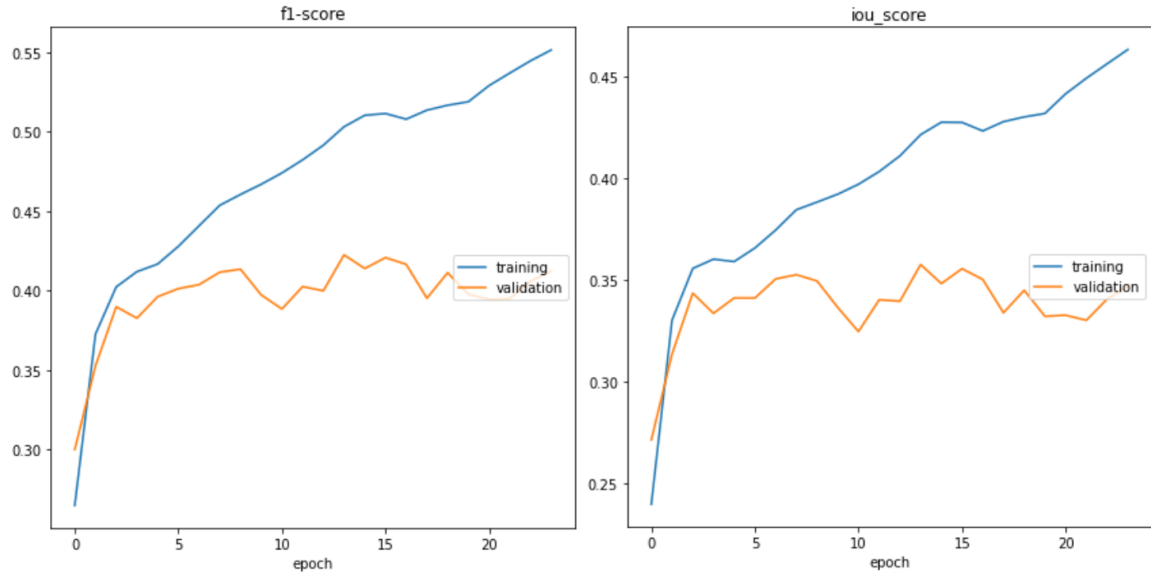
Results

This section shows the results of integrating the PySLAM framework with the deep learning segmentation mask to bind the keypoints to useful features that can be continuously tracked between frames. The overall outcome shows that this approach was able to perform slightly better than PySLAM alone, with fewer keypoints, or at least within the numerical accuracy of the framework.

4.1 Semantic Segmentation Results

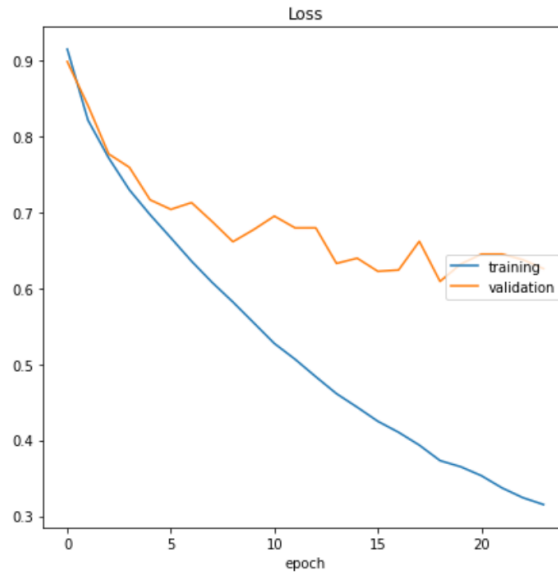
The training and validation curves for the U-Net model can be seen in Figure 4.1. Training was run using the procedure outlined in Section 3.5. It took 24 epochs for the model to finish training. Since a patience parameter of 5 was set, this means the lowest validation loss occurred at epoch 19. The final validation loss at epoch 24 was 0.63, and had a training loss of 0.32. The f1-scores for training and validation were 0.55 and 0.41, respectively. Final IOU scores ended up being 0.46 for training and 0.35 for the validation dataset. Typically a “good” model should have an IOU score above a threshold of 0.5 [69]. Since this thesis aims not to have the “best” segmentation network, these results are adequate for the proof of concept of using the prediction results of the network for rough guidelines for keypoint placement. The numbers stated earlier are also indicative of all 35 categories. From viewing the results of the segmentation network on an example image in Figure 4.2 it can be seen that the model behaves quite well on large objects and struggles with smaller ones or items that appear at the edge of the image frame, which is of little consequence at this point as occlusion is a common problem in deep learning [69].

Figure 4.3 demonstrates the result of the model’s prediction on an example image from the Kitti Odometry Dataset. The model performs slightly worse on these images, but this is expected due to domain adaptation [70]. That is to say, the model may achieve good results on test images from the same dataset used to train the model, but deviations of performance will occur on datasets from a



(a) Training & Validation f1-score

(b) Training & Validation IOU



(c) Training & Validation Loss Function

Figure 4.1: U-Net Training and Validation Curves Using Combo Focal-Weighted Dice Loss Function on The Cityscapes Dataset

different origin or with a dissimilar occurrence of certain classes. For the Kitti Odometry Dataset, the U-Net model had problems detecting large portions of open sky as it has no texture or apparent discerning features. Another issue the model had was with shadows, in that since shadows had no class of their own, they were wrongly detected as being a part of others. However, finding a coarse boundary around the shadow proved to be sufficient when finding locations around the shadow for keypoint placement as observed in Figure 4.4.

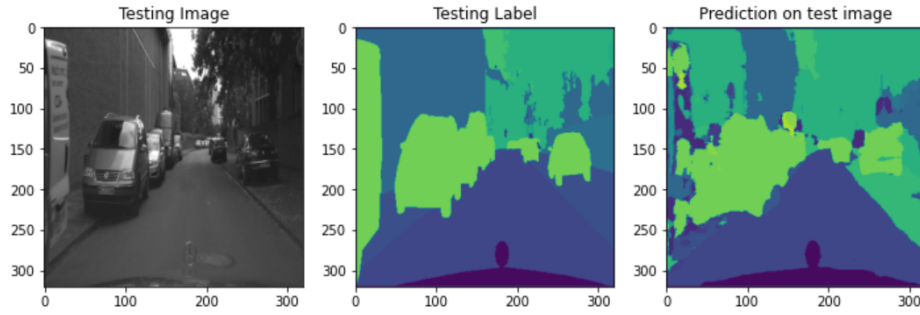


Figure 4.2: U-Net Model Performance on Cityscapes Test Image.

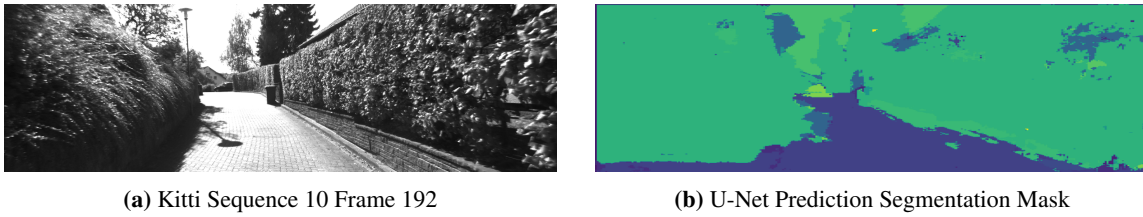


Figure 4.3: U-Net Model Performance on Kitti Odometry Sequence 10 Frame 192



Figure 4.4: Example Keypoint Placement with Semantic Segmentation Boundaries.

Overall the performance of the U-Net model for semantic segmentation was adequate for the purposes of showing the proof of concept integration of semantic segmentation with SLAM/Visual odometry for keypoint placement. If better segmentation boundaries are desired, one could use an ensemble network to procure the best predictions for every pixel. Additionally, coloured images could be used rather than grayscale for training. This would provide the model with additional data to learn from, which would enhance its ability to discern between multiple classes.

4.2 Visual Odometry Results

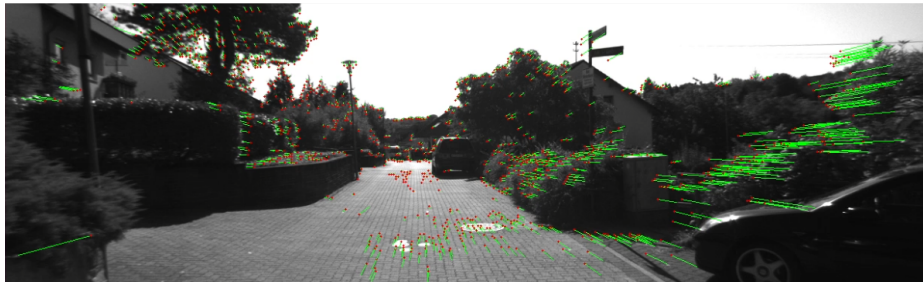
The Kitti Odometry Dataset provides eleven sequences with associated ground truths [45], all of which were used to measure and compare the RMSE and MAE of the original method and using the deep learning-based approach. To re-summarize, the deep learning-based approach is used to derive a way to filter out unnecessary keypoints. This process involves using a U-Net semantic segmentation model to make predictions on all frames within the 11 sequences of the Kitti Odometry Dataset. The model's input and output image size is 320×320 ; thus, in order for a proper edge mask to be defined using Canny edge detection, the image must first be resized to its original width and height. The Canny algorithm was applied with a lower threshold of 5 and an upper threshold of 15. The low threshold value comes at a cost at picking up artifacts from the segmentation prediction as usable edges. However, if such low threshold values were not used, the edge detector would not be able to pick out road signs, poles, or other thin objects. Different dilation widths were experimented with throughout this process. A graphical explanation of the sequential steps done for this procedure are shown in Figure 4.5.

PySLAM utilizes a default number of 2000 keypoints that can be altered by the user. The location of these keypoints are chosen using the Shi-Tomasi method described in Section 3.7. A large number of keypoints is not only computationally expensive but increases the error from the RANSAC algorithm as more points make it more challenging to find matches between frames, leading to a drift in the predicted trajectory [64].

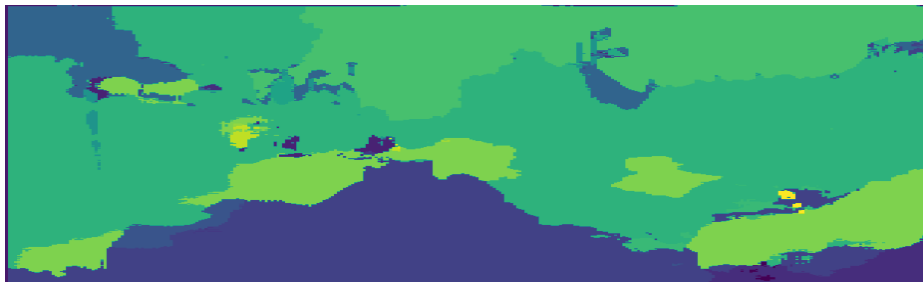
The trajectory maps using the basic visual odometry framework are shown in Figure 4.6 for all 11 sequences. The ground truth path is represented by the red line, while the green line denotes the estimated trajectory of the SLAM framework. Once again, this is not a full SLAM framework as there is no loop closure at the end of the travelled path. This pipeline exhibits typical problems associated with SLAM, such as being unable to return to its exact starting point and losing localization after a sharp 90-degree turn. These problems are seen in Sequences 00, 06, and 07 in Figure 4.6. Sequence 01 also experienced a problem of rapid divergence from the true path due to its lack of detectable features within the scene as well as the visual odometry problem of struggling with the prevalence of dynamic moving cars over static objects [27].



(a) Kitti Sequence 10 Frame 637



(b) Original PySLAM Keypoint Placement with Shi-Tomasi



(c) U-Net Prediction on Kitti Sequence 10 Frame 637



(d) Canny Edge Detection Applied with Dilation Width of 10



(e) Resulting Keypoint Placement From Segmentation Based Keypoint Filtering

Figure 4.5: Summary of Procedure for Segmentation Based Keypoint Filtering

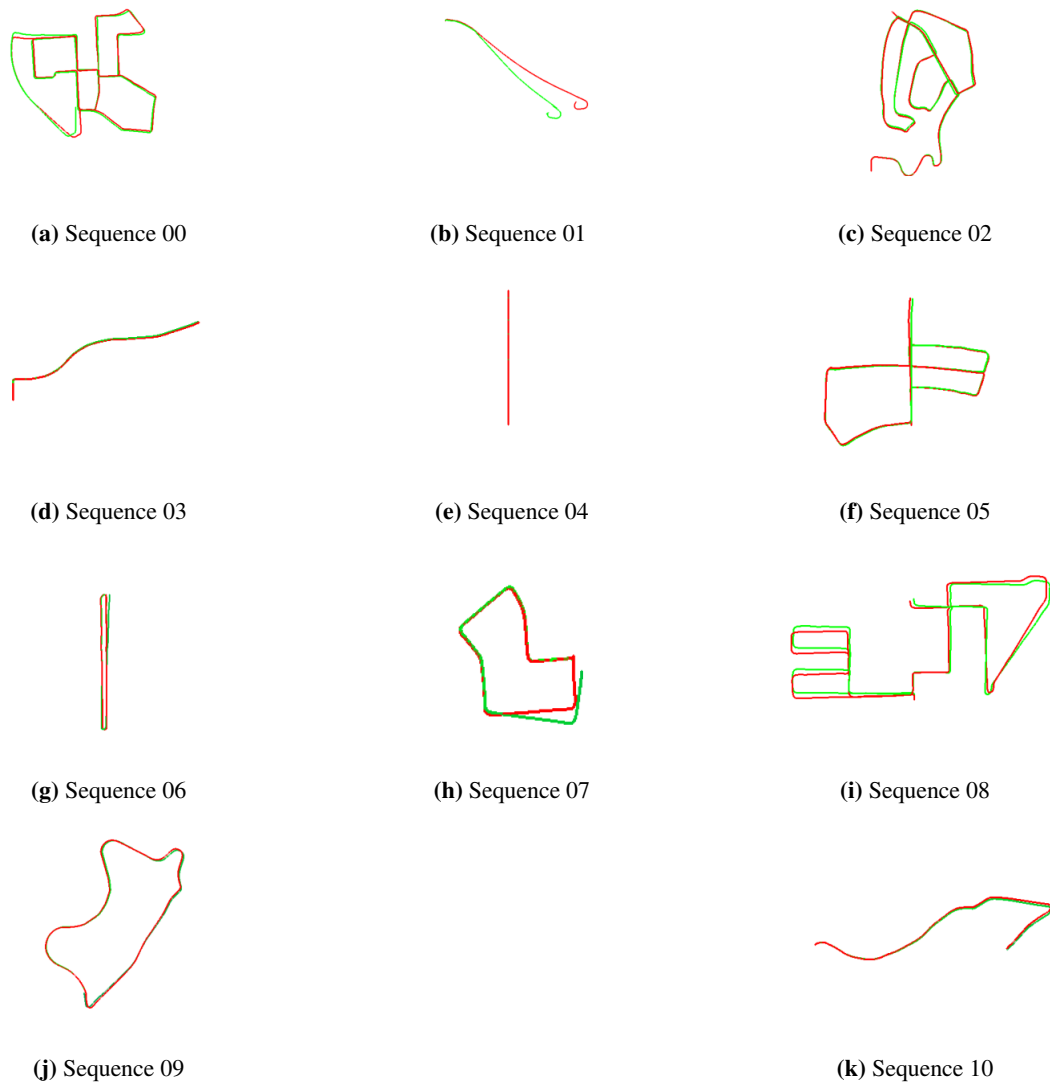


Figure 4.6: PySLAM Tracking Performance on 11 Sequences from The Kitti Odometry Dataset

Different dilation kernel widths were experimented with, starting from a rectangular kernel of size (1,1) up to (15,15). For each of these dilation widths, the average number of keypoints kept after filtering was calculated. Figure 4.7 shows the fraction of the keypoints kept for each sequence at all 15 different mask widths, relative to the original 2000 keypoints. All sequences exhibit the same general trend of the fraction of keypoints increasing as the dilation kernel size increases. This intuitively makes sense because as the width of the “white” lines of the keypoint mask is increased, more features will be kept.

Fraction of Average Number of Keypoints vs. Mask Dilation Width for Multiple Sequences

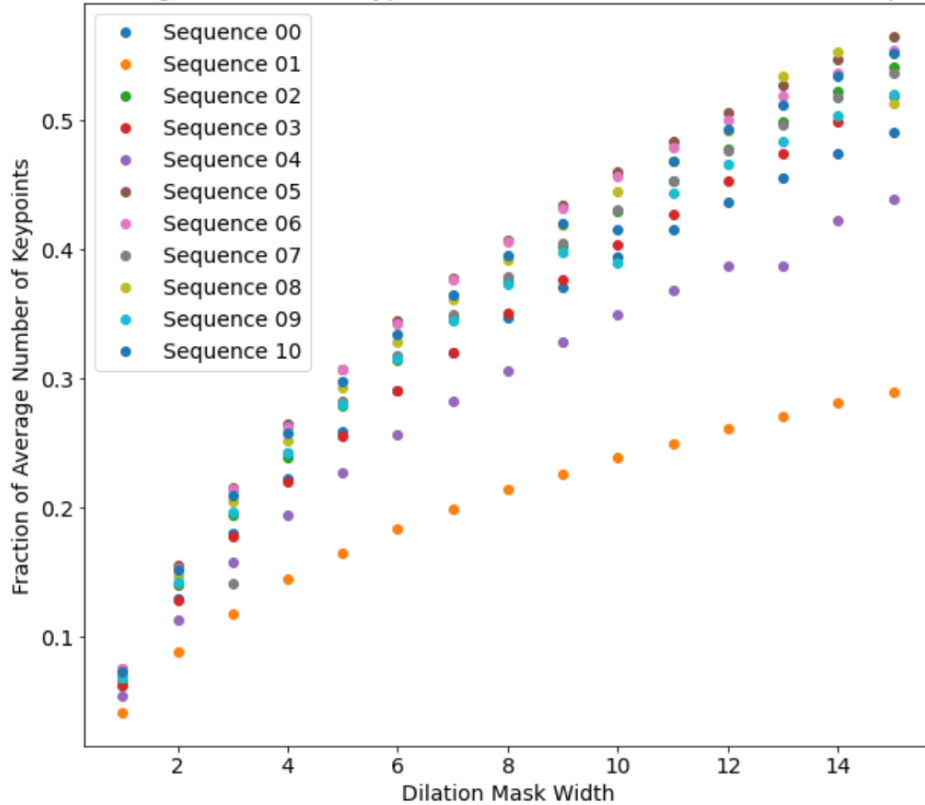


Figure 4.7: Fraction of The Average Number of Keypoints Kept with Increasing Dilation Width from Edges of Segmentation Mask

RMSE and MAE are calculated relative to the ground truth trajectory. The results for the RMSE and MAE of the segmentation-based keypoint filtering for comparison with the original PySLAM is shown in Figure 4.9 and Figure 4.10, respectively. Both error metrics have units of meters. For most of the sequences, masks dilation values between 9 and 11 obtained the lowest error. All dilation widths for the associated sequence number are shown in Table 4.1. The oscillation of error for both graphs can be attributed to multiple factors, mainly that the more keypoints that are allowed through with the mask, the more error there is during RANSAC and feature matching, as mentioned earlier. Another contribution to error is the fact that the segmentation boundaries of the same objects are not the same between consecutive frames, as seen in the center tree in Figure 4.8. Meaning, the

position of the keypoints can be “jumping” around from frame to frame to try and find the “edge” of the object given to it by the segmentation prediction, offsetting the trajectory. High errors are also attributed to scenarios where the moving vehicle that the camera is attached to either stops and makes a sudden turn, or goes through a turn at high velocities. When turning, half the scene was originally unseen, which means new keypoints have to be placed, and thus there is a loss of the original keypoints. These new keypoints could be placed in different locations within the frame, which has a consequence of there not being enough overlap between the new keypoints and old ones, adding to the overall error. However, this switching between new keypoints after a sudden turn is also a common problem in SLAM [71]. This is why for the majority of the sequences, tracking using semantic-based keypoint filtering was able to achieve similar or even lower error than the basic PySLAM framework.



Figure 4.8: Inconsistencies in Keypoint Placement for the Same Object Between Frames

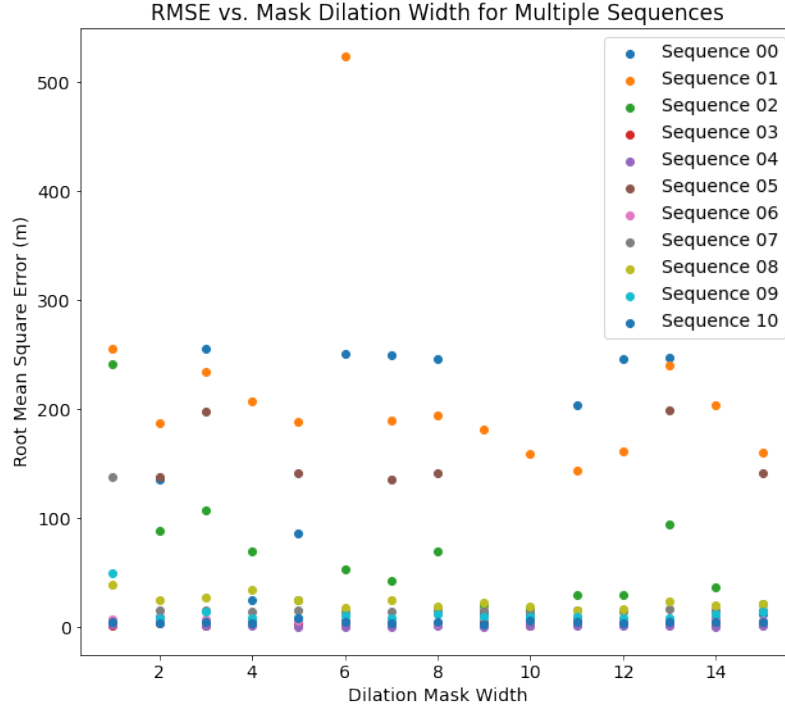


Figure 4.9: Root Mean Square Error of Estimated Trajectory with Semantic Segmentation Based Keypoint Filtering with Respect to Increased Dilation Mask Width.

By comparing the RMSE and MAE of the original PySLAM to the semantic segmentation-based keypoint filter approach, the best mask dilation width for each sequence was ascertained. The trajectories of the 11 sequences with their corresponding best dilation mask width can be viewed in Figure 4.11. Visually it can be seen that the proposed method executes better in sequences 01, 06, and 10. Arguably, it can also be said that the technique performs better than the original method in some areas of sequence 06 and sequence 09. In sequence 06, while the segmentation-based filtering does not track the steep turns of the loop as well as the original method, it seems to be more capable of returning to its original starting point. Whereas, for sequence 09, the original approach can go back to its starting point but struggles in the curvy regions that the semantic segmentation mask system is able to track more finely. Since sequence 06 has such a tight loop to travel and does so relatively quickly, better tracking could be achieved using more points at locations where the frequency of movement of the camera rapidly and suddenly increases. Sequences with RMSE above a hundred, such as sequence 01, occur when there are not enough static features to track compared to dynamic. It should also be noted that each sequence has a different number of frames, and as the number of frames increases, there is more room for errors to accumulate. For example, sequence 02 has 4560 frames, which is why it has a larger RMSE and MAE than sequence 07, which has 1100 frames. Therefore, RMSE and MAE between sequences cannot be compared.

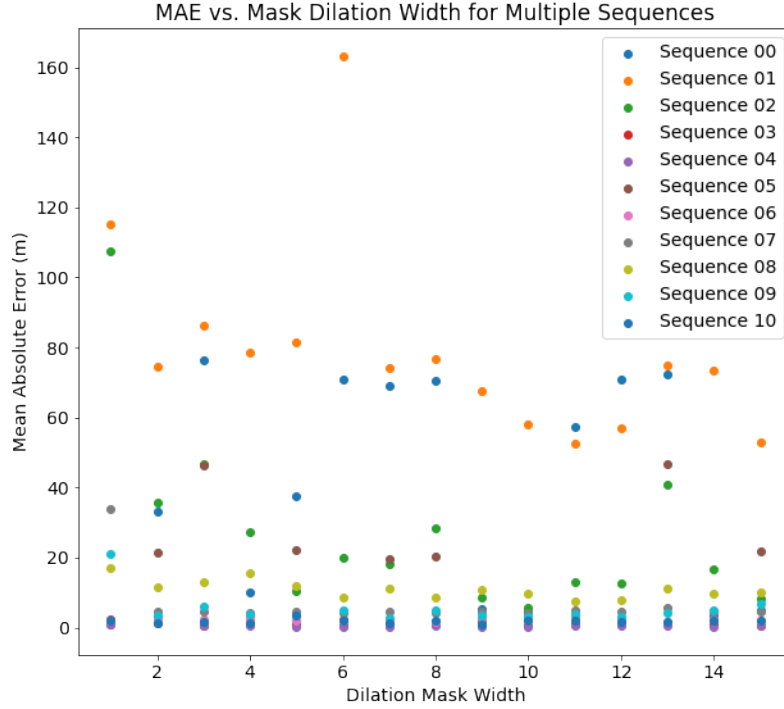


Figure 4.10: Mean Absolute Error of Estimated Trajectory with Semantic Segmentation Based Keypoint Filtering with Respect to Increased Dilation Mask Width.

Table 4.1 shows the average number of keypoints used for each sequence with the associated mask width that yielded the lowest RMSE and/or MAE compared with the original PySLAM. One could question why it would not be sufficient just to decrease the number of allowed feature detectors and descriptors, as this is an option within the OpenCV functions [56] and other features of PySLAM [34]. However, this is not a viable solution as this approach does not utilize useful features and randomly places the keypoints based on the Shi-Tomasi corner detection criterion. These might not be good features to track if the corresponding object or feature does not appear between a considerable number of subsequent frames, and thus is not valuable to be used for tracking purposes.

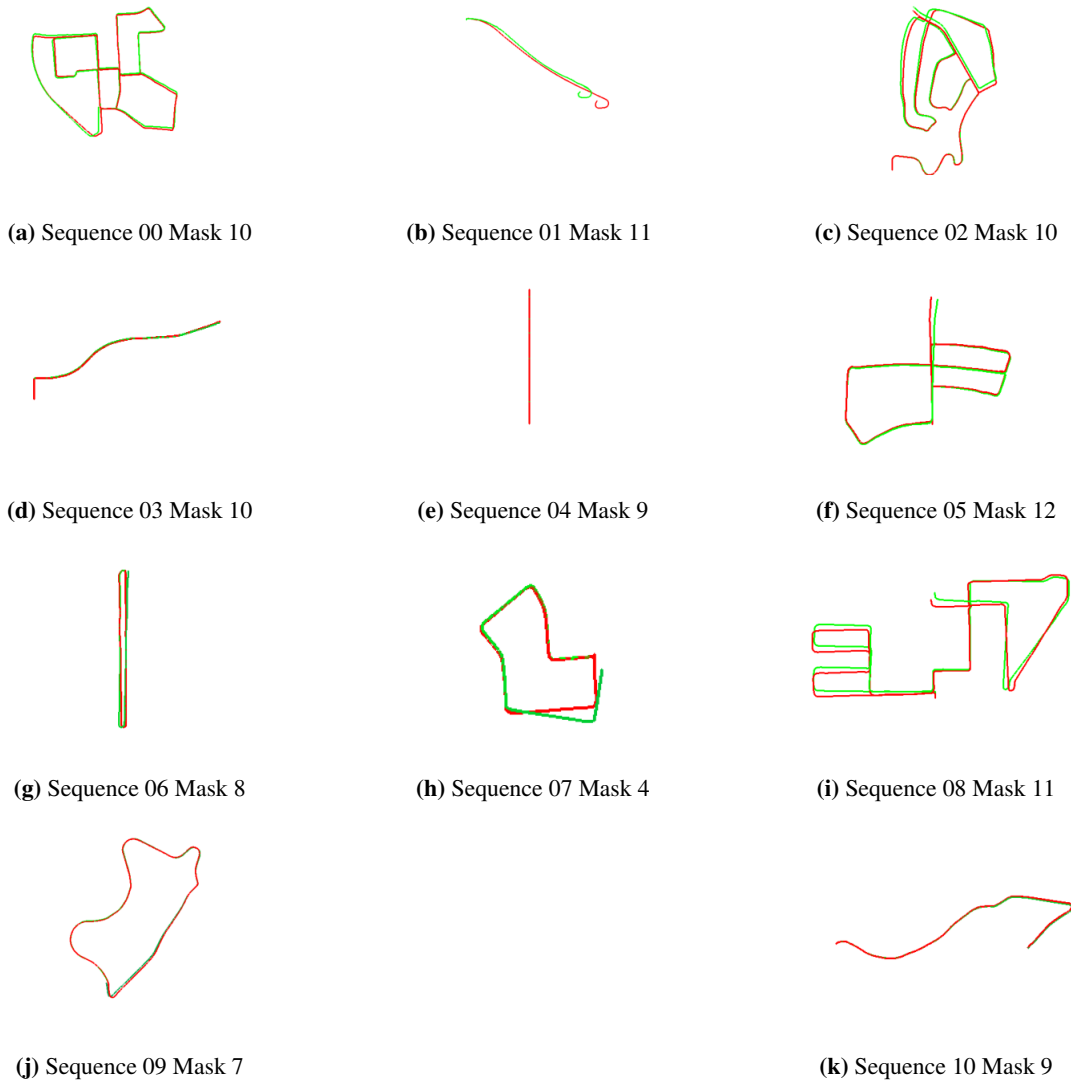


Figure 4.11: Semantic Segmentation Based Keypoint Filtering Tracking Performance on 11 Sequences from Kitti Odometry Dataset

PySLAM allows for the adjustment of the number of keypoints; therefore, this theory of just reducing the number of keypoints/features can be tested. The averages of each sequence from Table 4.1 were used as the basis for deciding the decreased number of keypoints. However, as expected, seen in Figure 4.12, by not providing a meaningful guideline for the keypoints to be placed, the tracking has no clear path to follow, and thus, the visual odometry trajectory is significantly skewed.

Sequence Number	Mask Dilation Width	Average Number of Keypoints Used
00	10	788
01	11	500
02	10	858
03	10	807
04	9	655
05	12	1011
06	8	811
07	4	485
08	11	935
09	7	690
10	9	839

Table 4.1: Average Number of Keypoints from Best Segmentation Mask Dilation Width

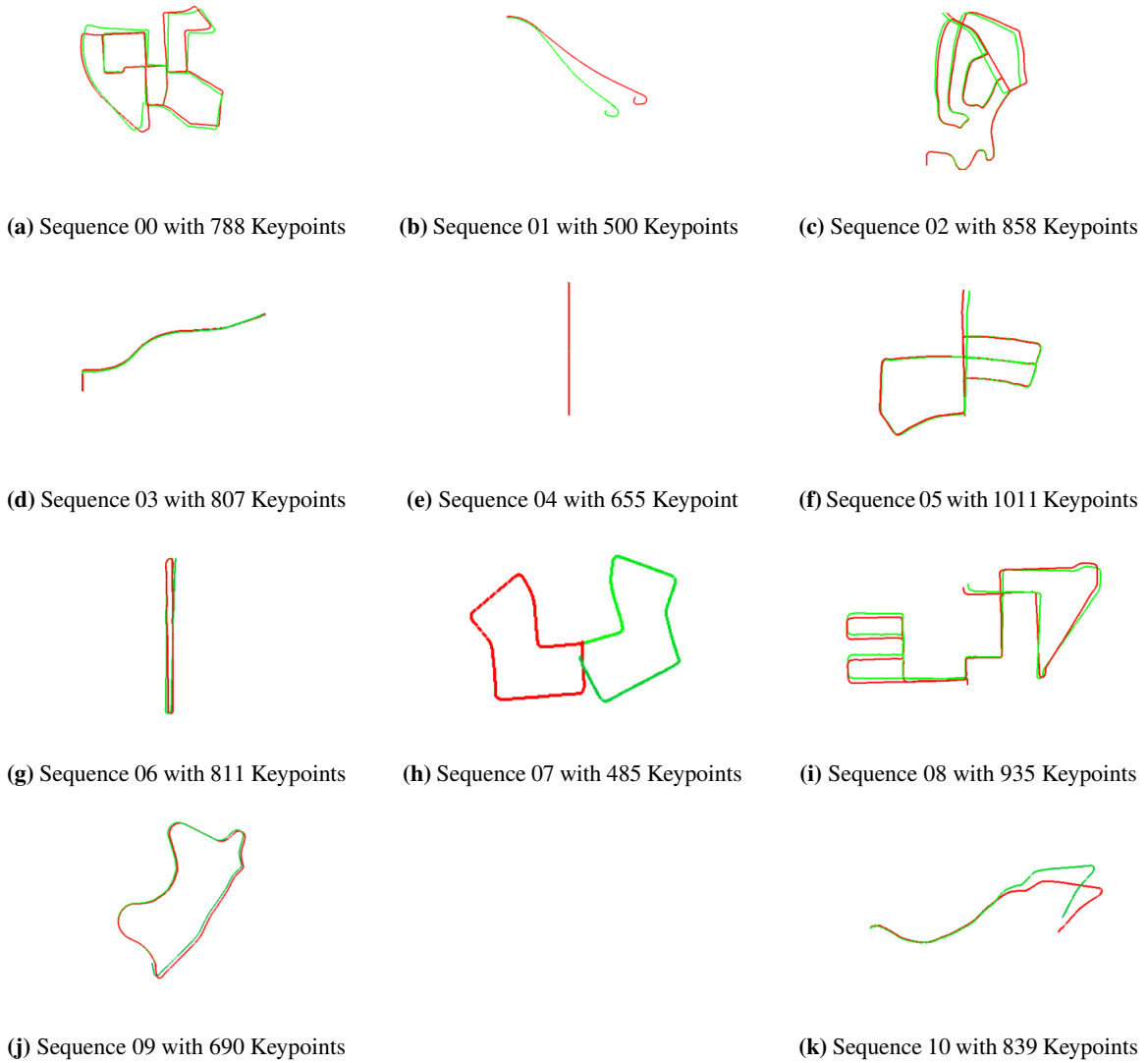


Figure 4.12: PySLAM Tracking Performance with Randomly Reduced Keypoints on 11 Sequences from Kitti Odometry Dataset

Figure 4.13 shows the comparison of the RMSE between the basic PySLAM framework, as well as with the generic reduced keypoint approach. This graph is further broken down in Figure 4.14. Similar graphs are also produced for the mean average error in Figure 4.15 and Figure 4.16. After assessing both of the errors, it can be concluded that using semantic segmentation to bind the keypoints to specific objects that can be continuously tracked through most of the frames can perform as well or slightly better than the basic SLAM/visual odometry pipeline.

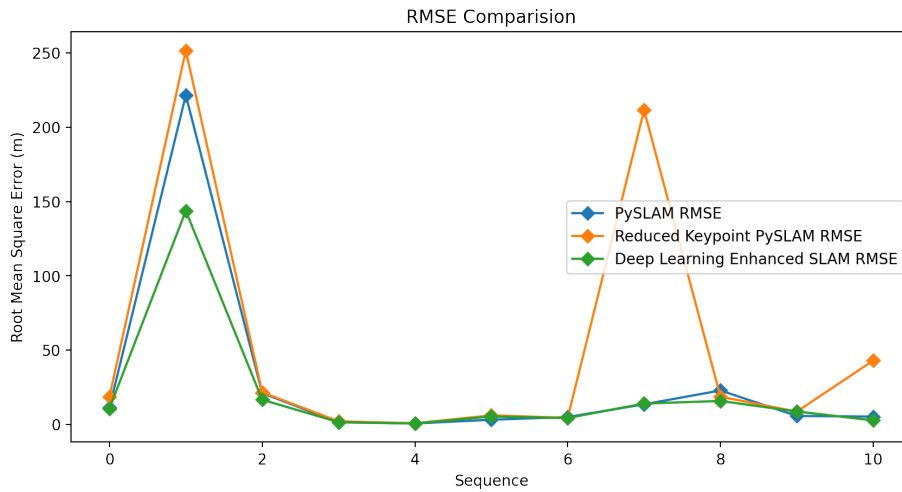


Figure 4.13: Root Mean Square Error Comparison Between Original PySLAM and Segmentation Based Keypoint Filtering.

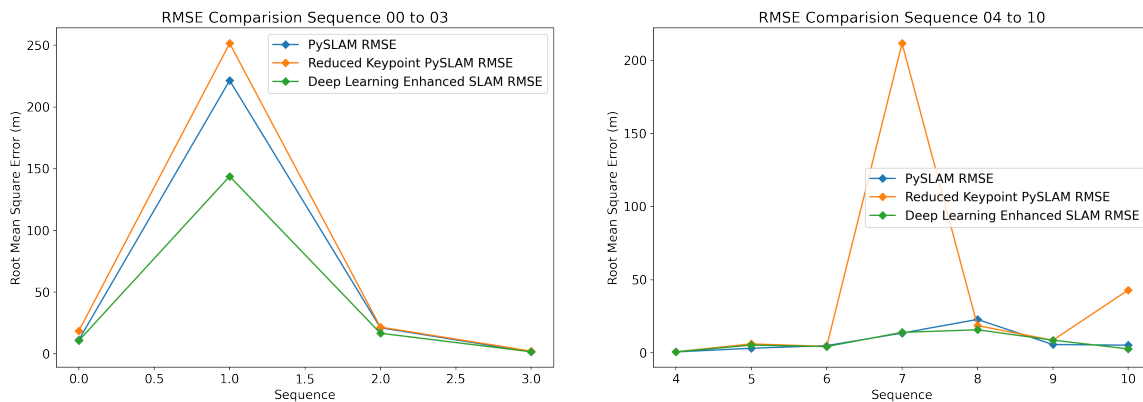


Figure 4.14: Breakdown of The Root Mean Square Error Trajectory Comparison Between Original PySLAM and Segmentation Based Keypoint Filtering

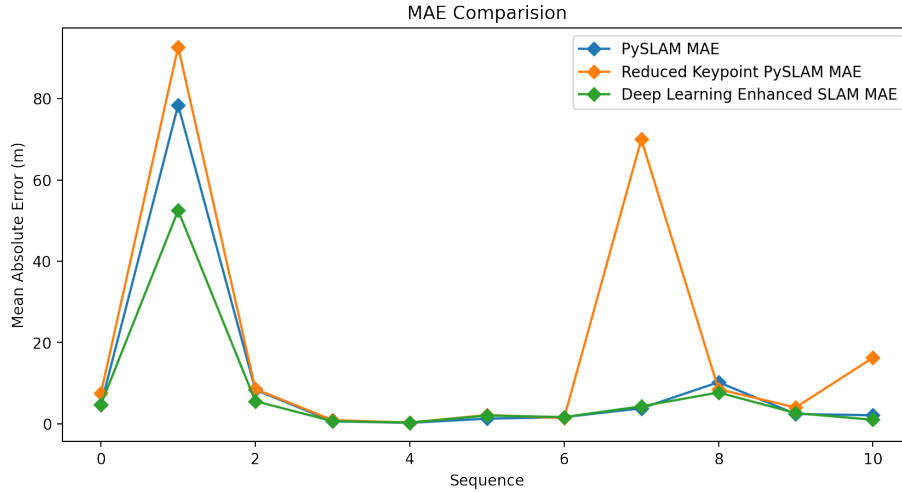


Figure 4.15: Mean Absolute Error trajectory Between Original PySLAM and Segmentation Based Keypoint Filtering.

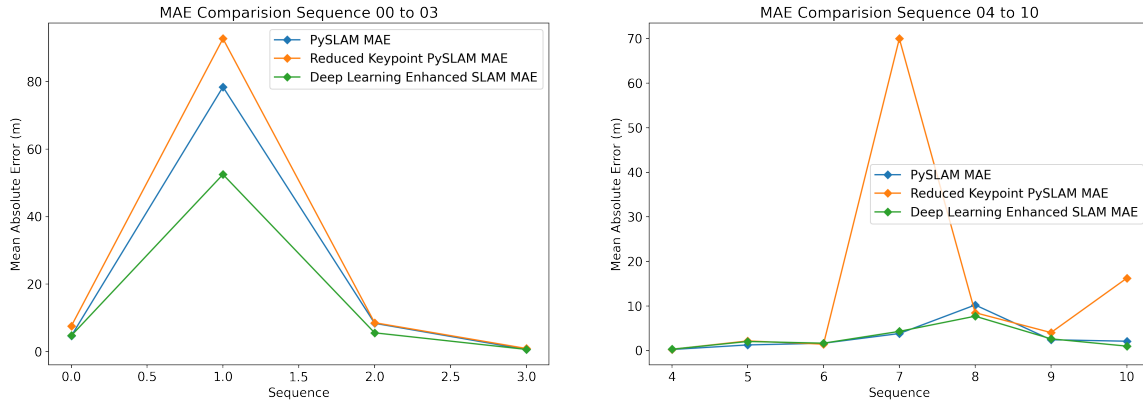


Figure 4.16: Breakdown of The Mean Absolute Error Trajectory Comparison Between Original PySLAM and Segmentation Based Keypoint Filtering

Randomly reducing keypoints without providing useful objects to track significantly increased the error above both the deep learning approach as well as the original visual odometry method. Providing clear object boundaries for visual odometry tracking with the segmentation masks proved extremely well for sequences 00, 01, 02, 08, and 10. The percent change between the original framework’s errors with 2000 points and the mask-based process, for all sequences, can be seen in Table 4.2 and Table 4.3. Percent change is defined as,

$$\text{Percent Change} = \frac{\text{Segmentation Based Keypoint Filtering Error} - \text{Benchmark Error}}{\text{Benchmark Error}} \times 100\% \quad (4.1)$$

where the benchmark error refers to the errors of the original PySLAM pipeline with no keypoint filtering. Thus, a negative sign is indicative of a sequence in which the segmentation mask-based filtering placement of keypoints exhibited better performance than the original benchmark.

The number of keypoints was reduced by factors of about $3-5\times$ while also being within the RMSE and MAE errors, or better for the sequences listed above.

Sequence Number	Percent Difference of RMSE
00	-5.27%
01	-35.08%
02	-21.17%
03	6.81%
04	9.66%
05	67.56%
06	-12.98%
07	3.72%
08	-30.88%
09	52.32%
10	-47.89%
Aggregate Performance	-13.2%

Table 4.2: Percent Change between PySLAM RMSE & Segmentation Based Keypoint Filtering RMSE

Sequence Number	Percent Difference of MAE
00	1.76%
01	-33.05%
02	-33.54%
03	4.76%
04	28.94%
05	60.89%
06	0.70%
07	12.56%
08	-24.61%
09	8.36%
10	-52.42%
Aggregate Performance	-25.65%

Table 4.3: Percent Change between PySLAM MAE & Segmentation Based Keypoint Filtering MAE

Looking at Table 4.2 and Table 4.3, the sequences in which the error is not reduced by the segmentation keypoint filtering procedure are 04, 05, 09. Sequence 04 seems to have a larger MAE, signifying lots of minor errors within the path estimation. Whereas sequence 09 has fewer overall errors, certain sections of the path estimation yield high error, causing the RMSE to increase. Sequence 05 has both a higher RMSE and MAE than the generic visual odometry pipeline. Restating what was mentioned earlier, the discrepancies between the semantic segmentation mask for keypoint placement and the original SLAM method can be attributed to the fluctuating segmentation boundaries, as seen in the example for sequence 09. Other deviations in error from the original method stem

from reducing the number of keypoints. Fast turns need more points to track, but at the same time, the issue of losing the majority of the keypoints during a turn of the camera cannot be avoided as this is a problem with the SLAM/visual odometry pipeline [71]. However, the aggregate performance of the semantic segmentation-based keypoint filter can be said to be improved over the approach of using no filtering. Through experimentation, it was discovered that there was an aggregate reduction in root mean square error of 13.2% and a mean average error reduction of 25.65%.

The percent time savings for each dilation factor were computed to see how reducing the number of keypoints can improve the computation speed. Timing profiling was run on the same NVIDIA GeForce GTX 1080Ti GPU used for U-Net training. The CPU used was a 8 core Intel i7-9600k with 64 GB of RAM and DDR4 memory at 3600GHz. The formula for the percent change is similar to the one used earlier for comparing errors and is as follows:

$$\text{Percent Time Savings} = \frac{\text{Total Time Seg Based} - \text{Total Time PySLAM}}{\text{Total Time PySLAM}} \times 100\% \quad (4.2)$$

where total means the sum of all execution times across all sequences, and seg-based refers to the segmentation-based keypoint filtering technique’s execution time. The timing only considers the time it takes between each frame to compute the feature matches and estimate the x, y, z pose. For the segmentation based filtering profiling, the time it takes to acquire the keypoint masks are also included in the total execution time. Plotting the results of percent time savings for each dilation factor yields Figure 4.17.

Percent Time Savings With Reduced Keypoints of Various Dilation Factors

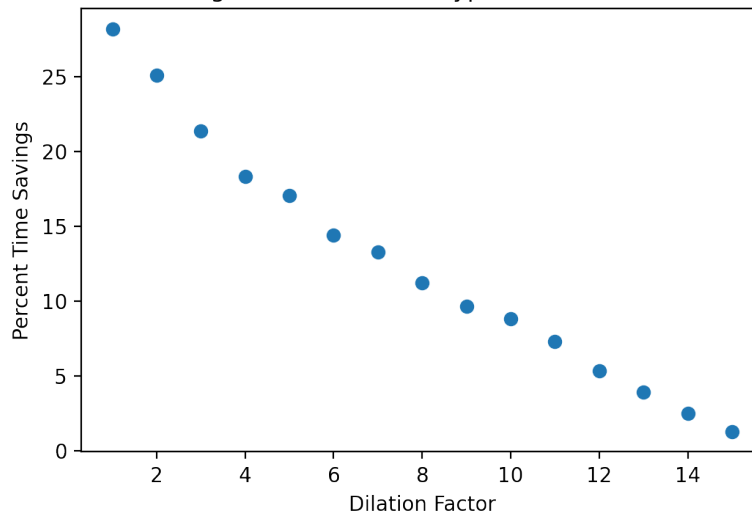


Figure 4.17: The Percentage Time Savings Between Segmentation Based Keypoint Filtering Compared To The Original PySLAM Per Dilation Factor.

As expected for smaller dilation factors, the higher the percent time savings as fewer points are kept after the filtering process. Since most of the sequences had the best performance with dilation widths between 9 to 11, there is mainly a time savings of about 5-10%. However, this time savings could substantially increase when comparing these two techniques for operation on an edge device with less processing power than the CPU used for these experiments.

Chapter 5

Conclusion

5.1 Thesis summary

This Thesis utilizes well-known semantic segmentation and visual odometry/SLAM techniques to develop an integrated system that would reduce the number of keypoints while maintaining tracking accuracy. The predictions from the segmentation network were used to create a “keypoint” mask through Canny edge detection, with the number of points that were kept from the original PySLAM modulated by different dilation widths.

Results shown in Section 4.1 showed the IOU, F1-score, and Loss for the semantic segmentation network U-Net. It was noted that while the model performed fairly well on test images from the Cityscapes dataset, prediction results were less accurate on the Kitti Odometry Dataset images used to generate the keypoint masks for the visual odometry PySLAM framework. However, inconsistency between the model’s performance on these two datasets can be attributed to domain adaptation, a common problem in deep learning.

From Section 4.2 it can be concluded that by this methodology of using segmentation maps to bind the keypoints, the number of keypoints can be reduced, and the algorithm is still able to track camera motion within the numerical inaccuracies of the PySLAM visual odometry framework. Overall, using segmentation-based keypoint filtering reduced the root mean square error by **13.2%**, and the Mean Absolute error reduction by **25.65%**. Errors introduced during the integration of keypoint binding masks pertain to the unstable placement of keypoints along an object boundary due to the varying segmentation mask produced by the U-Net model for consecutive frames. However, for more than half the sequences, the errors were minimized as the keypoints were mainly bound around useful objects. Thus, this segmentation keypoint filtering was able to elicit a more effective tracking method. Therefore, this methodology of finding boundaries of useful objects that can be tracked across multiple frames proves to help with pose estimation. Furthermore, the number

of keypoints were reduced by $3-5\times$ while maintaining tracking accuracy, which will significantly improve the computation cost and speed for the performance of this technique within an edge device. The calculated time savings of $5-10\%$ for the dilation factors with the lowest RMSE and MAE also add support to the claim that fewer keypoints increase the execution time of the SLAM process.

5.2 Future work

5.2.1 Model architecture

A significant amount of error from the deep-learning-based keypoint mask process was related to the inconsistencies of the boundaries produced by the segmentation model, mainly in regards to the changing boundary of the same object between consecutive frames. Therefore, using a different, more accurate model could mitigate those errors introduced by the fluctuation of boundaries between adjacent frames. Ensemble networks, for example, could be used to find the best prediction for each pixel [72]. The U-Net model itself can be ensembles, or the standard architecture can be built with different backbones such as RESNET, VGG, or Inception and can then be grouped to work in parallel [53]. Mask-RCNN, DeepLab, and FastFCN are also popular networks for semantic segmentation [73]. However, the size, performance and computation cost of these various models should be considered since the original purpose of this is to be used within an edge device alongside SLAM. In the future, the computation speed of the current U-Net model will be measured within an edge device for eventual use in robot tracking. Additionally, the original training process could be repeated with the basic U-Net architecture and same loss functions, but with coloured images and on a larger GPU to improve segmentation results.

5.2.2 Improvements to tracking

Current SLAM and visual odometry methods are imprecise in general. While this Thesis was able to reduce the number of keypoints required, errors from the original visual odometry pipeline still exist. In order to develop a better system, more in-depth testing and analysis needs to be done. Finding the aspects of the algorithm that add to error should be analyzed in order to remove errors that can be propagated through the system. Experimenting with a more sophisticated feature detector and descriptor, such as ORB, could be another next step, though there would be a trade off between speed and detector accuracy. For timing evaluation, one should take into consideration of how much time it takes for each part, the segmentation aspect, the SLAM, to confirm that the components of the algorithm that is being changed in this research is improving the overall speed performance of the whole system. One could limit the error caused by not knowing the keypoints in the area that the camera turns into with a fisheye camera. However, a fisheye camera adds more distortion, which

would have to be removed, adding to the complexity of the model. Another way to handle the loss of keypoints during turning would be if the system was able to detect turning within a few frames. Then, the algorithm could be adjusted to focus the keypoints on the side of the frame in which the camera is turning into, rather than focusing on the side where keypoints would be lost in the next few frames. Natural features such as trees and bushes are also hard to assign feature descriptors to, as it is hard to discern their direction as you approach those objects. Utilizing the semantic segmentation aspect of this thesis, in the future, to track unchanging static objects, such as the features on buildings or static vehicles. Removal of keypoints could also be done in areas that you know will contribute to error such as points far away or up close to the camera. Most errors in this framework derive from the feature matching and RANSAC process, as the algorithm randomly compares points with the previous frame and tries to match the “best” pairs. Therefore, the SLAM pipeline is flawed because this method relies on the speculation that keypoints have shifted between frames. Meaning, the algorithm is unsure of which keypoint in the next frame is to be paired with another in the previous frame. Developing a way to truly bind the keypoint to the exact same location of various objects for tracking between consecutive frames would eliminate the error caused by this uncertainty in matching. One approach would be to train a neural network for the consistent keypoint placement on known objects in precise locations of the object each time. This approach might also be able to reduce the keypoints further, from a couple of hundred to less than a hundred.

References

- [1] P. Sharma, “Image Classification vs. Object Detection vs. Image Segmentation | by Pulkit Sharma | Analytics Vidhya | Medium,” Aug. 2019. [Online]. Available: <https://medium.com/analytics-vidhya/image-classification-vs-object-detection-vs-image-segmentation-f36db85fe81>.
- [2] M. A. Diaz-zapata, M. Alejandro, D.-z. Y.-b. P. Segmentation, C. Vision, M. Alejandro, and D. Zapata, “YOLO-Based Panoptic Segmentation To cite this version : YOLO-Based Panoptic Segmentation,” 2020.
- [3] D. Gálvez-López, M. Salas, J. D. Tardós, and J. M. Montiel, “Real-time monocular object SLAM,” *Robotics and Autonomous Systems*, vol. 75, pp. 435–449, 2016, ISSN: 09218890. DOI: 10.1016/j.robot.2015.08.009. [Online]. Available: <http://dx.doi.org/10.1016/j.robot.2015.08.009>.
- [4] S. Saeedi, M. Trentini, M. Seto, and H. Li, “Multiple-Robot Simultaneous Localization and Mapping: A Review,” *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016. DOI: 10.1002/rob. [Online]. Available: <http://onlinelibrary.wiley.com/doi/10.1002/rob.21514/abstract>.
- [5] Ó. M. Mozos, A. Gil, M. Ballesta, and O. Reinoso, “Interest point detectors for visual SLAM,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 4788 LNAI, pp. 170–179, 2007, ISSN: 16113349. DOI: 10.1007/978-3-540-75271-4_{_}18.
- [6] J. Aulinas, Y. Petillot, J. Salvi, and X. Lladó, “The SLAM problem: A survey,” *Frontiers in Artificial Intelligence and Applications*, vol. 184, no. 1, pp. 363–371, 2008, ISSN: 09226389. DOI: 10.3233/978-1-58603-925-7-363.
- [7] H. Durrant-Whyte and T. Bailey, “Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms,” *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006, ISSN: 00029645.
- [8] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: An efficient alternative to SIFT or SURF,” *Proceedings of the IEEE International Conference on Computer Vision*, no. November 2011, pp. 2564–2571, 2011. DOI: 10.1109/ICCV.2011.6126544.
- [9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” Tech. Rep. [Online]. Available: <http://code.google.com/p/cuda-convnet/>.
- [10] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep Learning for Computer Vision: A Brief Review,” *Computational Intelligence and Neuroscience*, vol. 2018, 2018, ISSN: 16875273. DOI: 10.1155/2018/7068349.

- [11] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: <https://cs231n.github.io/convolutional-networks/>.
- [12] S. Almabdy and L. Elrefaei, “Deep convolutional neural network-based approaches for face recognition,” *Applied Sciences (Switzerland)*, vol. 9, no. 20, 2019, ISSN: 20763417. DOI: 10.3390/app9204397.
- [13] D. Britz, “Understanding Convolutional Neural Networks for NLP,” 2015. [Online]. Available: <http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/>.
- [14] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8689 LNCS, no. PART 1, pp. 818–833, 2014, ISSN: 16113349. DOI: 10.1007/978-3-319-10590-1_{_}53.
- [15] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” pp. 1–20, 2018. [Online]. Available: <http://arxiv.org/abs/1811.03378>.
- [16] B. Ayten, “Activation Functions in Deep Neural Network | by Barış Ayten | DataDriven-Investor.” [Online]. Available: <https://medium.datadriveninvestor.com/activation-functions-in-deep-neural-network-4d8849b70046>.
- [17] H. S, “Activation Functions : Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning | by Himanshu S | Medium,” Jan. 2019. [Online]. Available: <https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e>.
- [18] S. Saha, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science,” Dec. 2018. [Online]. Available: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>.
- [19] R. Rojas, “Neural Networks: A Systematic Introduction.” Springer-Verlag Berlin Heidelberg, 1996, pp. 151–184.
- [20] D. Karunakaran, “Deep learning series 1: Intro to deep learning | by Dhanoop Karunakaran | Intro to Artificial Intelligence | Medium,” Apr. 2018. [Online]. Available: <https://medium.com/intro-to-artificial-intelligence/deep-learning-series-1-intro-to-deep-learning-abb1780ee20>.
- [21] J. Jordan, “Neural networks: training with backpropagation.” Jul. 2017. [Online]. Available: <https://www.jeremyjordan.me/neural-networks-training/>.
- [22] S. Raschka, “Gradient Descent and Stochastic Gradient Descent - mlxtend.” [Online]. Available: http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/.
- [23] S. Sharma, “Epoch vs Batch Size vs Iterations | by SAGAR SHARMA | Towards Data Science,” Sep. 2017. [Online]. Available: <https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9>.
- [24] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” Tech. Rep., 2015. DOI: 10.1007/978-3-319-24574-4_{_}28.

- [25] N. Adaloglou, “Intuitive Explanation of Skip Connections in Deep Learning | AI Summer,” Mar. 2020. [Online]. Available: <https://theaisummer.com/skip-connections/>.
- [26] H. Sankesara, “UNet. Introducing Symmetry in Segmentation | by Heet Sankesara | Towards Data Science,” Jan. 2019. [Online]. Available: <https://towardsdatascience.com/unet-b229b32b4a71>.
- [27] F. Fraundorfer and D. Scaramuzza, “Visual odometry: Part II: Matching, robustness, optimization, and applications,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 2, pp. 78–90, 2012, ISSN: 10709932. DOI: 10.1109/MRA.2012.2182810.
- [28] D. Scaramuzza and F. Fraundorfer, “Visual Odometry Part I: The First 30 Years and Fundamentals,” DOI: 10.1109/MRA.2011.943233.
- [29] “OpenCV: Camera Calibration.” [Online]. Available: https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html.
- [30] D. Tyagi, “Introduction To Feature Detection And Matching | by Deepanshu Tyagi | Data Breach | Medium,” Jan. 2019. [Online]. Available: <https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d>.
- [31] D. Mukherjee, Q. M. Jonathan Wu, and G. Wang, “A comparative experimental study of image feature detectors and descriptors,” *Machine Vision and Applications*, vol. 26, no. 4, pp. 443–466, 2015, ISSN: 14321769. DOI: 10.1007/s00138-015-0679-9. [Online]. Available: <http://dx.doi.org/10.1007/s00138-015-0679-9>.
- [32] A. Nikishaev, “Feature extraction and similar image search with OpenCV for newbies | by Andrey Nikishaev | Machine Learning World | Medium,” Feb. 2018. [Online]. Available: <https://medium.com/machine-learning-world/feature-extraction-and-similar-image-search-with-opencv-for-newbies-3c59796bf774>.
- [33] “OpenCV: Epipolar Geometry.” [Online]. Available: https://docs.opencv.org/master/da/de9/tutorial_py_epipolar_geometry.html.
- [34] L. Freda, “luigifreda/pyslam: pySLAM contains a monocular Visual Odometry (VO) pipeline in Python. It supports many modern local features based on Deep Learning.” [Online]. Available: <https://github.com/luigifreda/pyslam>.
- [35] D. Scaramuzza, “Tutorial on Visual Odometry.” [Online]. Available: http://rpg.ifi.uzh.ch/docs/Visual_Odometry_Tutorial.pdf.
- [36] S. Ishida, “How Robots Make Maps— an Intro to SLAM (Simultaneous Localisation and Mapping) | by Shu Ishida | The Startup | Medium,” Aug. 2020. [Online]. Available: <https://medium.com/swlh/how-robots-make-maps-an-intro-to-slam-simultaneous-localisation-and-mapping-37370c3e7dfe>.
- [37] T. Lemaire, C. Berger, I. K. Jung, and S. Lacroix, “Vision-based SLAM: Stereo and monocular approaches,” *International Journal of Computer Vision*, vol. 74, no. 3, pp. 343–364, 2007, ISSN: 09205691. DOI: 10.1007/s11263-007-0042-3.
- [38] H. Alhaila, S. Mustikovela, L. Mescheder, G. Andreas; and C. Rother, “Augmented Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes,” *International Journal of Computer Vision (IJCV)*, 2018.
- [39] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, D. A. R&d, and T. U. Darmstadt, “The Cityscapes Dataset for Semantic Urban Scene Understanding,” Tech. Rep. [Online]. Available: www.cityscapes-dataset.net.

- [40] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, “Albumentations: Fast and Flexible Image Augmentations,” *Information*, vol. 11, no. 2, 2020. DOI: 10.3390/info11020125. [Online]. Available: www.mdpi.com/journal/information.
- [41] A. Buslaev, A. Parinov, V. Iglovikov, and E. Khvedchenya, “Setting probabilities for transforms in an augmentation pipeline - Albumentations Documentation.” [Online]. Available: https://albumentations.ai/docs/getting_started/setting_probabilities/.
- [42] —, “Mask augmentation for segmentation - Albumentations Documentation.” [Online]. Available: https://albumentations.ai/docs/getting_started/mask_augmentation/.
- [43] A. Ng and K. Katanforoosh, “CS230 - Section 8 (Week 8),” 2021. [Online]. Available: <http://cs230.stanford.edu/section/8/>.
- [44] I. Tan, “Measuring Labelling Quality with IOU and F1 Score | by Isaac Tan | Supahands Tech Blog | Medium,” Mar. 2020. [Online]. Available: <https://medium.com/supahands-techblog/measuring-labelling-quality-with-iou-and-f1-score-1717e29e492f>.
- [45] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the KITTI vision benchmark suite,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361, ISBN: 9781467312264. DOI: 10.1109/CVPR.2012.6248074. [Online]. Available: www.cvlibs.net/datasets/kitti.
- [46] N. Q. Long, X.-n. Bui, N. V. Nghia, and P. V. Chung, “Advances and Applications in Geospatial Technology and Earth Resources,” *Advances and Applications in Geospatial Technology and Earth Resources*, no. November 2017, 2018. DOI: 10.1007/978-3-319-68240-2.
- [47] JJ, “MAE and RMSE — Which Metric is Better? | by JJ | Human in a Machine World | Medium,” Mar. 2016. [Online]. Available: <https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d>.
- [48] J. Brownlee, “Why One-Hot Encode Data in Machine Learning?” Jul. 2017. [Online]. Available: <https://machinelearningmastery.com/why-one-hot-encode-data-in-machine-learning/>.
- [49] L. Wang, C. Wang, Z. Sun, and S. Chen, “An improved dice loss for pneumothorax segmentation by mining the information of negative areas,” *IEEE Access*, vol. 8, pp. 167 939–167 949, 2020, ISSN: 21693536. DOI: 10.1109/ACCESS.2020.3020475.
- [50] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, 2020, ISSN: 19393539. DOI: 10.1109/TPAMI.2018.2858826.
- [51] S. Jadon, “A survey of loss functions for semantic segmentation,” *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology, CIBCB 2020*, 2020. DOI: 10.1109/CIBCB48159.2020.9277638.
- [52] P. Yakubovskiy, “Segmentation Models Python API — Segmentation Models 0.1.2 documentation,” 2018. [Online]. Available: <https://segmentation-models.readthedocs.io/en/latest/api.html#losses>.
- [53] —, “Segmentation Models,” 2019. [Online]. Available: https://github.com/qubvel/segmentation_models.

- [54] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 2015.
- [55] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning," Jul. 2017. [Online]. Available: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- [56] "OpenCV: Feature Detection." [Online]. Available: https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html.
- [57] "OpenCV: Canny Edge Detector." [Online]. Available: https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html.
- [58] R. C. Gonzalez and R. E. Woods, "Digital Image Processing," 4th. Pearson, 2018, pp. 641–735, ISBN: 9353062985.
- [59] "OpenCV: Feature Matching." [Online]. Available: https://docs.opencv.org/4.5.2/dc/dc3/tutorial_py_matcher.html.
- [60] C. Tomasi and T. Kanade, "Shape and Motion from Image Streams: a Factorization Method-Part 3 Detection and Tracking of Point Features Introduction," 1991.
- [61] "OpenCV: Shi-Tomasi Corner Detector & Good Features to Track." [Online]. Available: https://docs.opencv.org/4.5.2/d4/d8c/tutorial_py_shi_tomasi.html.
- [62] "OpenCV: Optical Flow." [Online]. Available: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html.
- [63] C. Liu, S. Seitz, L. Zitnick, and A. Farhadi, "Motion and Optical Flow."
- [64] M. Kaneko, K. Iwami, T. Ogawa, T. Yamasaki, and K. Aizawa, "Mask-SLAM: Robust feature-based monocular SLAM by masking using semantic segmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2018-June, pp. 371–379, 2018, ISSN: 21607516. DOI: 10.1109/CVPRW.2018.00063.
- [65] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017, ISSN: 15523098. DOI: 10.1109/TR0.2017.2705103.
- [66] C. Yu, Z. Liu, X.-j. Liu, F. Xie, Y. Yang, Q. Wei, and Q. Fei, "Ds-Slam," 2018.
- [67] J. Lee, M. Back, S. S. Hwang, and I. Y. Chun, "Improved Real-Time Monocular SLAM Using Semantic Segmentation on Selective Frames," pp. 1–10, 2021. [Online]. Available: <http://arxiv.org/abs/2105.00114>.
- [68] H. Gaoussou and P. Dewei, "Evaluation of the Visual Odometry Methods for Semi-Dense Real-Time," *Advanced Computing: An International Journal*, vol. 9, no. 2, pp. 01–14, 2018, ISSN: 2229726X. DOI: 10.5121/acij.2018.9201.
- [69] J. Cowton, I. Kyriazakis, and J. Bacardit, "Automated Individual Pig Localisation, Tracking and Behaviour Metric Extraction Using Deep Learning," *IEEE Access*, vol. 7, pp. 108 049–108 060, 2019, ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2933060.
- [70] A. Farahani, S. Voghoei, K. Rasheed, and H. R. Arabnia, "A Brief Review of Domain Adaptation," 2020. [Online]. Available: <http://arxiv.org/abs/2010.03978>.
- [71] K. Wahlqvist, "A comparison of motion priors for EKF-SLAM in autonomous race cars," 2019. [Online]. Available: http://www.nada.kth.se/~ann/exjobb/kristian_wahlqvist.pdf.

- [72] W. H. Khoong, "BUSU-Net: An Ensemble U-Net Framework for Medical Image Segmentation," 2020. [Online]. Available: <http://arxiv.org/abs/2003.01581>.
- [73] D. Mwititi and K. Y. Li, "Image Segmentation in 2021: Architectures, Losses, Datasets, and Frameworks | Neptune Blog." [Online]. Available: <https://neptune.ai/blog/image-segmentation-in-2020>.

Appendix A

U-Net Model Training Code

A.1 Import Required Libraries

```
from keras.utils import normalize
import os
import glob
import cv2
import numpy as np
import cv2
import keras
from tqdm import tqdm
from glob import glob
import albumentations as A
import tensorflow as tf
%env SM_FRAMEWORK=tf.keras
from pycocotools.coco import COCO
import skimage.io as io
import random
import skimage.exposure as skie
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
from keras.layers import Input, Conv2D, MaxPooling2D, UpSampling2D
from keras.layers import concatenate, Conv2DTranspose, BatchNormalization, Dropout, Lambda
from keras.layers import Activation, MaxPool2D, Concatenate
from livelossplot import PlotLossesKeras
from keras.metrics import MeanIoU
from keras import backend as K

### For visualizing the outputs ###
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
%matplotlib inline
import segmentation_models as sm
```

A.2 Load Images and Labels

A.2.1 Training Images

```
#Resizing images, if needed
SIZE_X = 320
SIZE_Y = 320

#Capture training image info as a list
train_images = []

flist = os.listdir("C:/Thesis/CityscapesSemantic/train")

for fname in flist:
    img_path = os.path.join("C:/Thesis/CityscapesSemantic/train", fname)
    img = cv2.imread(img_path,0) #Read in as grayscale
    img = cv2.resize(img, (SIZE_Y, SIZE_X))
    train_images.append(img)

#Convert list to array for machine learning processing
train_images = np.array(train_images)

flist_mask = os.listdir("C:/Thesis/CityscapesSemantic/trainannot")

#Capture mask/label info as a list
train_masks = []

for fname in flist_mask:
    mask_path = os.path.join("C:/Thesis/CityscapesSemantic/trainannot", fname)
    mask = cv2.imread(mask_path, 0)
    mask = cv2.resize(mask, (SIZE_Y, SIZE_X), interpolation = cv2.INTER_NEAREST)
    train_masks.append(mask)

#Convert list to array for machine learning processing
train_masks = np.array(train_masks)

print(train_images.shape)
print(train_masks.shape)
print(np.unique(train_masks))
```

A.2.2 Testing Images

```
#Do the same for testing Images
#Resizing images, if needed
SIZE_X = 320
SIZE_Y = 320
n_classes= len(np.unique(train_masks))+1 #Number of classes for segmentation

#Capture training image info as a list
test_images = []

flist = os.listdir("C:/Thesis/CityscapesSemantic/val")

for fname in flist:
    img_path = os.path.join("C:/Thesis/CityscapesSemantic/val", fname)
    img = cv2.imread(img_path,0) #Read in as grayscale
    img = cv2.resize(img, (SIZE_Y, SIZE_X))
```

```

    test_images.append(img)

#Convert list to array for machine learning processing
test_images = np.array(test_images)

flist_mask = os.listdir("C:/Thesis/CityscapesSemantic/valannot")

#Capture mask/label info as a list
test_masks = []

for fname in flist_mask:
    mask_path = os.path.join("C:/Thesis/CityscapesSemantic/valannot", fname)
    mask = cv2.imread(mask_path, 0)
    #Otherwise ground truth changes due to interpolation
    mask = cv2.resize(mask, (SIZE_Y, SIZE_X), interpolation = cv2.INTER_NEAREST)
    test_masks.append(mask)

#Convert list to array for machine learning processing
test_masks = np.array(test_masks)

print(test_images.shape)
print(test_masks.shape)

#Expand the mask dimensions to get the proper channels for training
train_masks_input = np.expand_dims(train_masks, axis=3)
train_images = np.expand_dims(train_images, axis=3)

print(train_masks_input.shape)
print(train_images.shape)

```

A.2.3 Split into Training and Validation Sets

```

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test =
train_test_split(train_images, train_masks_input, test_size = 0.2, random_state=0)

print('Training image Size:', X_train.shape)
print('Training Mask Size:', Y_train.shape)
print('Val image Size:', X_test.shape)
print('Val Mask Size:', Y_test.shape)

#One-hot encode the masks
from keras.utils import to_categorical
train_masks_cat = to_categorical(Y_train, num_classes = n_classes)
print(train_masks_cat.shape)
test_masks_cat = to_categorical(Y_test, num_classes = n_classes)
print(test_masks_cat.shape)

#Check a few images
import random
import numpy as np
image_number = random.randint(0, len(X_train))
plt.figure(figsize = (12,6))
plt.subplot(121)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(cv2.cvtColor(X_train[image_number], cv2.COLOR_BGR2RGB))
plt.subplot(122)
plt.imshow(np.reshape(Y_train[image_number], (320,320)))

```


A.3 Data Augmentation

```
def round_clip_0_1(x, **kwargs):
    return x.round().clip(0, 1)

transform = A.Compose(
    [
        A.HorizontalFlip(p=0.5),

        A.ShiftScaleRotate(scale_limit=0.5, rotate_limit=0, shift_limit=0.1, p=1, border_mode=0),

        A.PadIfNeeded(min_height=320, min_width=320, always_apply=True, border_mode=0),
        A.RandomCrop(height=320, width=320, always_apply=True),

        A.GaussNoise(p=0.2),
        A.Perspective(p=0.5),

        A.OneOf(
            [
                A.CLAHE(p=1),
                A.RandomBrightness(p=1),
                A.RandomGamma(p=1),
            ],
            p=0.9,
        ),

        A.OneOf(
            [
                A.Sharpen(p=1),
                A.Blur(blur_limit=3, p=1),
                A.MotionBlur(blur_limit=3, p=1),
            ],
            p=0.9,
        ),

        A.Lambda(mask=round_clip_0_1)
    ]
)

images_aug = []
mask_aug = []
for i in range(len(X_train)):
    augmentations = transform(image=X_train[i], mask = train_masks_cat[i])
    augmented_img = augmentations["image"]
    augmented_mask = augmentations["mask"]
    images_aug.append(augmented_img)
    mask_aug.append(augmented_mask)

images_aug = np.array(images_aug)
mask_aug = np.array(mask_aug)

#Show some Augmentations
print(images_aug.shape)
print(mask_aug.shape)
image_number = random.randint(0, len(images_aug)-1)
plt.figure(figsize = (12,6))
plt.subplot(121)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(cv2.cvtColor(images_aug[image_number], cv2.COLOR_BGR2RGB), interpolation = 'nearest')
plt.subplot(122)
plt.imshow(np.argmax(mask_aug[image_number], axis=2))
```

View Some Transformations

```
###Images###
image = X_train[4]
transform = A.HorizontalFlip(p=0.9)
random.seed(42)
augmented_image = transform(image=image)['image']
fig = plt.figure(figsize = (12,10))

plt.subplot(221)
plt.title('Original Image')
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), interpolation = 'nearest')
plt.subplot(222)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(cv2.cvtColor(augmented_image, cv2.COLOR_BGR2RGB),interpolation = 'nearest')
plt.title('Horizontal Flip')

transform = A.ShiftScaleRotate(scale_limit=0.5, rotate_limit=0, shift_limit=0.1, p=1, border_mode=0)
random.seed(42)
augmented_image = transform(image=image)['image']
plt.subplot(223)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(cv2.cvtColor(augmented_image, cv2.COLOR_BGR2RGB),interpolation = 'nearest')
plt.title('Affine Transformation')

transform = A.Blur(blur_limit=3, p=1)
random.seed(42)
augmented_image = transform(image=image)['image']
plt.subplot(224)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(cv2.cvtColor(augmented_image, cv2.COLOR_BGR2RGB),interpolation = 'nearest')
plt.title('Random Blur')
fig.tight_layout()
fig.savefig('Example Figure Augmentations.png')

###Masks###
image = Y_train[4]
transform = A.HorizontalFlip(p=0.9)
random.seed(42)
augmented_image = transform(image=image)['image']
fig = plt.figure(figsize = (12,10))

plt.subplot(221)
plt.title('Original Image')
plt.imshow(image, interpolation = 'nearest')
plt.subplot(222)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(augmented_image,interpolation = 'nearest')
plt.title('Horizontal Flip')

transform = A.ShiftScaleRotate(scale_limit=0.5, rotate_limit=0, shift_limit=0.1, p=1, border_mode=0)
random.seed(42)
augmented_image = transform(image=image)['image']
plt.subplot(223)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(augmented_image,interpolation = 'nearest')
plt.title('Affine Transformation')

transform = A.Blur(blur_limit=3, p=1)
random.seed(42)
augmented_image = transform(image=image)['image']
plt.subplot(224)
#Opencv reads images as BGR, convert to RGB for display
plt.imshow(augmented_image,interpolation = 'nearest', vmin=0, vmax = 33)
plt.title('Random Blur')
fig.tight_layout()
```

```
fig.savefig('Example Mask Augmentations.png')
```

Combine Original Image and Mask Array With Augmented Images and Masks

```
test_images_aug = []
test_mask_aug = []
for i in range(len(X_test)):
    augmentations = transform(image=X_test[i], mask = test_masks_cat[i])
    augmented_img = augmentations["image"]
    augmented_mask = augmentations["mask"]
    test_images_aug.append(augmented_img)
    test_mask_aug.append(augmented_mask)

test_images_aug = np.array(test_images_aug)
test_mask_aug = np.array(test_mask_aug)

print(test_images_aug.shape)
print(test_mask_aug.shape)

#Combine arrays
X_train = np.concatenate((X_train, images_aug))
X_test = np.concatenate((X_test, test_images_aug))
#Sanity check after augmentations
print('Training image Size:', X_train.shape)
print('Val image Size:', X_test.shape)

train_masks_cat = np.concatenate((train_masks_cat, mask_aug))
test_masks_cat = np.concatenate((test_masks_cat, test_mask_aug))

print('Training Mask Size:', train_masks_cat.shape)
print('Val Mask Size:', test_masks_cat.shape)
```

A.4 Define U-Net Architecture

```
def conv_block(input, num_filters):
    x = Conv2D(num_filters, 3, padding = "same")(input)
    x = BatchNormalization()(x) #Not in the original Network
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding = "same")(input)
    x = BatchNormalization()(x) #Not in the original Network
    x = Activation("relu")(x)
    return x

#Encoder Block: Conv block followed by Max Pooling
def encoder_block(input, num_filters):
    x = conv_block(input, num_filters)
    p = MaxPool2D((2,2))(x)
    return x,p

#Decoder Block
#Skip features gets input from encoder for concatenation
def decoder_block(input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2,2), strides = 2, padding = "same")(input)
    x = Concatenate()(x, skip_features)
    x = conv_block(x, num_filters)
    return x
```



```
input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)
print(input_shape)
```

A.5.2 Model Summary

```
# compile keras model with defined optimizer, loss and metrics
model1 = build_unet(input_shape, n_classes = n_classes)
model1.compile(optim, total_loss, metrics=metrics)

#Display the information about the model and the number of trainable parameters
print(model1.summary())
```

Model: "U-Net"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 320, 320, 1)]	0	
conv2d_1 (Conv2D)	(None, 320, 320, 64)	640	input_2[0][0]
batch_normalization_1 (BatchNormal)	(None, 320, 320, 64)	256	conv2d_1[0][0]
activation_1 (Activation)	(None, 320, 320, 64)	0	batch_normalization_1[0][0]
max_pooling2d (MaxPooling2D)	(None, 160, 160, 64)	0	activation_1[0][0]
conv2d_3 (Conv2D)	(None, 160, 160, 128)	73856	max_pooling2d[0][0]
batch_normalization_3 (BatchNormal)	(None, 160, 160, 128)	512	conv2d_3[0][0]
activation_3 (Activation)	(None, 160, 160, 128)	0	batch_normalization_3[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 80, 80, 128)	0	activation_3[0][0]
conv2d_5 (Conv2D)	(None, 80, 80, 256)	295168	max_pooling2d_1[0][0]
batch_normalization_5 (BatchNormal)	(None, 80, 80, 256)	1024	conv2d_5[0][0]
activation_5 (Activation)	(None, 80, 80, 256)	0	batch_normalization_5[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 40, 40, 256)	0	activation_5[0][0]
conv2d_7 (Conv2D)	(None, 40, 40, 512)	1180160	max_pooling2d_2[0][0]
batch_normalization_7 (BatchNormal)	(None, 40, 40, 512)	2048	conv2d_7[0][0]
activation_7 (Activation)	(None, 40, 40, 512)	0	batch_normalization_7[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 20, 20, 512)	0	activation_7[0][0]
conv2d_9 (Conv2D)	(None, 20, 20, 1024)	4719616	max_pooling2d_3[0][0]
batch_normalization_9 (BatchNormal)	(None, 20, 20, 1024)	4096	conv2d_9[0][0]
activation_9 (Activation)	(None, 20, 20, 1024)	0	batch_normalization_9[0][0]
conv2d_transpose (Conv2DTranspose)	(None, 40, 40, 512)	2097664	activation_9[0][0]
concatenate (Concatenate)	(None, 40, 40, 1024)	0	conv2d_transpose[0][0] activation_7[0][0]
conv2d_11 (Conv2D)	(None, 40, 40, 512)	4719104	concatenate[0][0]
batch_normalization_11 (BatchNormal)	(None, 40, 40, 512)	2048	conv2d_11[0][0]
activation_11 (Activation)	(None, 40, 40, 512)	0	batch_normalization_11[0][0]
conv2d_transpose_1 (Conv2DTranspose)	(None, 80, 80, 256)	524544	activation_11[0][0]
concatenate_1 (Concatenate)	(None, 80, 80, 512)	0	conv2d_transpose_1[0][0] activation_5[0][0]
conv2d_13 (Conv2D)	(None, 80, 80, 256)	1179904	concatenate_1[0][0]
batch_normalization_13 (BatchNormal)	(None, 80, 80, 256)	1024	conv2d_13[0][0]
activation_13 (Activation)	(None, 80, 80, 256)	0	batch_normalization_13[0][0]
conv2d_transpose_2 (Conv2DTranspose)	(None, 160, 160, 128)	131200	activation_13[0][0]
concatenate_2 (Concatenate)	(None, 160, 160, 256)	0	conv2d_transpose_2[0][0] activation_3[0][0]
conv2d_15 (Conv2D)	(None, 160, 160, 128)	295040	concatenate_2[0][0]
batch_normalization_15 (BatchNormal)	(None, 160, 160, 128)	512	conv2d_15[0][0]
activation_15 (Activation)	(None, 160, 160, 128)	0	batch_normalization_15[0][0]
conv2d_transpose_3 (Conv2DTranspose)	(None, 320, 320, 64)	32832	activation_15[0][0]
concatenate_3 (Concatenate)	(None, 320, 320, 128)	0	conv2d_transpose_3[0][0] activation_1[0][0]
conv2d_17 (Conv2D)	(None, 320, 320, 64)	73792	concatenate_3[0][0]
batch_normalization_17 (BatchNormal)	(None, 320, 320, 64)	256	conv2d_17[0][0]
activation_17 (Activation)	(None, 320, 320, 64)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 320, 320, 35)	2275	activation_17[0][0]

Total params: 15,337,571
Trainable params: 15,331,683
Non-trainable params: 5,888

Figure A.1: Model Summary of U-Net.

Training the Model

```
history1=model1.fit(X_train,
                    train_masks_cat,
                    batch_size=16,
                    epochs=100,
                    verbose=1,
                    callbacks = [PlotLossesKeras(), early_stop, checkpointer],
                    validation_data=(X_test, test_masks_cat))
```

Saving and Loading the Model

```
#model1.save('Unet_cityscapes2/smlib_dropout_focal_Diceloss_weighted_nodropout_320gray.hdf5')
from keras.models import load_model
model1 = keras.models.load_model
('Unet_cityscapes2/smlib_dropout_focal_Diceloss_weighted_nodropout_320gray.hdf5',
 custom_objects = {'dice_loss_plus_1focal_loss': total_loss,
 'iou_score': sm.metrics.IOUScore(threshold=0.5), 'f1-score': sm.metrics.FScore(threshold=0.5)})
```

A.6 Training and Validation Curves

Plot the IOU score and F1-score curves of the training and validation datasets. Also graph the loss for both datasets.

```
# Plot training & validation iou_score values
fig = plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history1.history['iou_score'])
plt.plot(history1.history['val_iou_score'])
plt.title('Model iou_score')
plt.ylabel('iou_score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')

# Plot training & validation loss values
plt.subplot(122)
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
fig.savefig('Unet_cityscapes2/Iou+Loss.png')

fig=plt.figure(figsize=(20, 10))
plt.subplot(221)
plt.plot(history1.history['f1-score'])
plt.plot(history1.history['val_f1-score'])
plt.title('Model f1-score')
plt.ylabel('f1-score')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
fig.savefig('Unet_cityscapes2/f1-Score.png')
```

A.7 Prediction of Model on Test Images

A.7.1 Predict on Test Images From Cityscapes Dataset

```
#Predict on a few images
import random
test_masks_input = np.expand_dims(test_masks, axis=3)
test_img_number = random.randint(0, len(test_images))
test_img = test_images[test_img_number]
ground_truth=test_masks_input[test_img_number]
print(test_images.shape)
print(ground_truth.shape)
#test_img_norm=test_img[:, :, 0][:, :, None]
test_img_input=np.expand_dims(test_img,0)#(test_img_norm, 0)
print(test_img_input.shape)
prediction = model1.predict(test_images[test_img_number].reshape(1,320,320,1))
predicted_img=np.argmax(prediction, axis=3)[0, :, :]

plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(cv2.cvtColor(test_img,cv2.COLOR_BGR2RGB))
plt.subplot(232)
plt.title('Testing Label')
plt.imshow(ground_truth[:, :, 0], vmin=0, vmax = 33)
plt.subplot(233)
plt.title('Prediction on test image')
plt.imshow(predicted_img, vmin=0, vmax = 33)
plt.show()
```

A.7.2 Predict on Test Images From Kitti Odometry Gray Dataset

```
#Resizing images, if needed
SIZE_X = 320
SIZE_Y = 320

#Capture training image info as a list
kitti_images = []

flist = os.listdir("C:/Thesis/data_odometry_gray/dataset/sequences/01/image_0")
#print(flist)

for fname in flist:
    img_path = os.path.join("C:/Thesis/data_odometry_gray/dataset/sequences/01/image_0", fname)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (SIZE_Y, SIZE_X))
    kitti_images.append(img)

#Convert list to array for machine learning processing
kitti_images = np.array(kitti_images)

print(kitti_images.shape)

from PIL import Image
#Predict on a few images
import random
test_img_number = random.randint(0, len(kitti_images))
test_img = kitti_images[test_img_number]
```



```

print(test_img.shape)
test_img_norm=test_img[:, :, 0][: :, None]
print(test_img_norm.shape)
test_img_input=np.expand_dims(test_img_norm, 0)
print(test_img_input.shape)
prediction = model1.predict(test_img_input)
predicted_img=np.argmax(prediction, axis=3)[0, :, :]

plt.figure(figsize=(12, 8))
plt.subplot(231)
plt.title('Testing Image')
plt.imshow(test_img)
#plt.imshow(cv2.cvtColor(test_img[:, :, 0], cv2.COLOR_BGR2RGB))
plt.subplot(232)
plt.title('Prediction on test image')
plt.imshow(predicted_img, vmin=0, vmax = 33)
plt.show()

```

A.8 Prepare Segmentation Masks For Keypoint Placement

```

flist = os.listdir("C:/Thesis/data_odometry_gray/dataset/sequences/06/seg320")
img_path = os.path.join("C:/Thesis/data_odometry_gray/dataset/sequences/06/seg320", fname)
d=0
for i in range(len(kitti_images)):
    test_img = kitti_images[i]
    test_img_norm=test_img[:, :, 0][: :, None]
    test_img_input=np.expand_dims(test_img_norm, 0)
    prediction = model1.predict(test_img_input)
    predicted_img=np.argmax(prediction, axis=3)[0, :, :]
    cv2.imwrite('C:/Thesis/data_odometry_gray/dataset/sequences/06/seg320/%06d.png'%d, predicted_img)
    d = d+1

#Capture training image info as a list
kitti_images2 = []

flist = os.listdir("C:/Thesis/data_odometry_gray/dataset/sequences/10/seg320")

SIZE_Y = 1226
SIZE_X = 370

for fname in flist:
    img_path = os.path.join("C:/Thesis/data_odometry_gray/dataset/sequences/10/seg320", fname)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (SIZE_Y, SIZE_X))
    kitti_images2.append(img)

#Convert list to array for machine learning processing
kitti_images2 = np.array(kitti_images2)

print(kitti_images2.shape)

d=0
for i in range(len(kitti_images2)):
    final = cv2.Canny(kitti_images2[i], 5, 15)
    kernel = np.ones((5, 5), np.uint8)
    img_dilation = cv2.dilate(final, kernel)
    cv2.imwrite('C:/Thesis/data_odometry_gray/dataset/sequences/10/mask5/%06d.png'%d, img_dilation)
    d = d+1

```

Appendix B

Copyright Permissions

Permission for Use of Figure 2.1



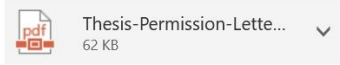
Denny Britz [redacted]



Sure, no problem! Feel free to use them!



Samantha Betts [redacted]



Download

Hello Mr. Britz,

I am an undergraduate engineering science student from Simon Fraser University in British Columbia Canada. I am writing to ask permission to use some of your copyrighted images in my undergraduate thesis which will be published in SFU's online library catalogue.

I would like to use two images from your article on WildML titled "Understanding Convolutional Neural Networks for NLP" and this material will be attributed through a citation. The figure names of this image are referred to in the form I have attached. If I have your approval of using these images in my published work please read and sign the attached form.

If you have any other questions please let me know.

Thank you for your time.

Regards,

Samantha

Permission for Use of Figure 2.5

Samantha Betts • 11:38 AM

Hello Mr. Saha I would like to ask your permission to use one of the images from your Medium Article "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way". The image is for my undergraduate engineering thesis and will be published in SFU library (Canada). If you agree, please reply

Sumit Saha • 11:38 AM

Hi Samantha,

Please feel free to reuse any of the images from the blog.

Best,
Sumit

Permission for Use of Figure 2.6

Permission to Use Copyrighted Material in a Thesis



Sebastian Raschka



Hi Samantha,

I am happy to give you permission to reuse an image from the website article "Gradient Descent and Stochastic Gradient Descent" given the citation/attribution you mentioned. I am glad to hear it's useful to you. And best of luck with your thesis :)

Best regards,
Sebastian



Samantha Betts



Thesis-Permission-Lette...
63 KB



Download

Hello Dr. Raschka,

I am an undergraduate engineering science student from Simon Fraser University in British Columbia Canada. I am writing to ask permission to use some of your copyrighted images in my undergraduate thesis which will be published in SFU's online library catalogue.

I would like to an image from your website article "Gradient Descent and Stochastic Gradient Descent" and this material will be attributed through a citation. The figure name of this image is referred to in the form I have attached. If I have your approval of using this image in my published work please read and sign the attached form.

If you have any other questions please let me know.

Thank you for your time.

Regards,

Samantha

Permission for Use of Figure 2.7

SPRINGER NATURE U-Net: Convolutional Networks for Biomedical Image Segmentation
Author: Olaf Ronneberger, Philipp Fischer, Thomas Brox
Publication: Springer eBook
Publisher: Springer Nature
Date: Jan 1, 2015
Copyright © 2015, Springer International Publishing Switzerland

Order Completed

Thank you for your order.

This Agreement between Ms. Samantha Betts ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number 5126080661419

[Printable Details](#)

License date Aug 11, 2021

Licensed Content

Licensed Content Publisher Springer Nature
Licensed Content Publication Springer eBook
Licensed Content Title U-Net: Convolutional Networks for Biomedical Image Segmentation
Licensed Content Author Olaf Ronneberger, Philipp Fischer, Thomas Brox
Licensed Content Date Jan 1, 2015

Order Details

Type of Use Thesis/Dissertation
Requestor type academic/university or research institute
Format electronic
Portion figures/tables/illustrations
Number of figures/tables /illustrations 1
Will you be translating? no
Circulation/distribution 1 - 29
Author of this Springer Nature content no

About Your Work

Title Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods
Institution name Simon Fraser University
Expected presentation date Aug 2021

Additional Data

Portions Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). The image on page 2

Requestor Location

Ms. Samantha Betts

Tax Details

Permission for Use of Figure 2.8



Visual Odometry [Tutorial]

Author: Davide Scaramuzza
Publication: IEEE Robotics & Automation Magazine
Publisher: IEEE
Date: Dec. 2011

Copyright © 2011, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:


- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Permission for Use of Figure 2.9



A comparative experimental study of image feature detectors and descriptors

Author: Dibyendu Mukherjee et al
 Publication: Machine Vision and Applications
 Publisher: Springer Nature
 Date: Apr 18, 2015
 Copyright © 2015, Springer-Verlag Berlin Heidelberg

Order Completed

Thank you for your order.

This Agreement between Ms. Samantha Betts ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

License Number	5126080002405	Printable Details
License date	Aug 11, 2021	

<p>Licensed Content</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Licensed Content Publisher</td> <td>Springer Nature</td> </tr> <tr> <td>Licensed Content Publication</td> <td>Machine Vision and Applications</td> </tr> <tr> <td>Licensed Content Title</td> <td>A comparative experimental study of image feature detectors and descriptors</td> </tr> <tr> <td>Licensed Content Author</td> <td>Dibyendu Mukherjee et al</td> </tr> <tr> <td>Licensed Content Date</td> <td>Apr 18, 2015</td> </tr> </table>	Licensed Content Publisher	Springer Nature	Licensed Content Publication	Machine Vision and Applications	Licensed Content Title	A comparative experimental study of image feature detectors and descriptors	Licensed Content Author	Dibyendu Mukherjee et al	Licensed Content Date	Apr 18, 2015	<p>Order Details</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Type of Use</td> <td>Thesis/Dissertation</td> </tr> <tr> <td>Requestor type</td> <td>academic/university or research institute</td> </tr> <tr> <td>Format</td> <td>electronic</td> </tr> <tr> <td>Portion</td> <td>figures/tables/illustrations</td> </tr> <tr> <td>Number of figures/tables/illustrations</td> <td>1</td> </tr> <tr> <td>Will you be translating?</td> <td>no</td> </tr> <tr> <td>Circulation/distribution</td> <td>1 - 29</td> </tr> <tr> <td>Author of this Springer Nature content</td> <td>no</td> </tr> </table>	Type of Use	Thesis/Dissertation	Requestor type	academic/university or research institute	Format	electronic	Portion	figures/tables/illustrations	Number of figures/tables/illustrations	1	Will you be translating?	no	Circulation/distribution	1 - 29	Author of this Springer Nature content	no
Licensed Content Publisher	Springer Nature																										
Licensed Content Publication	Machine Vision and Applications																										
Licensed Content Title	A comparative experimental study of image feature detectors and descriptors																										
Licensed Content Author	Dibyendu Mukherjee et al																										
Licensed Content Date	Apr 18, 2015																										
Type of Use	Thesis/Dissertation																										
Requestor type	academic/university or research institute																										
Format	electronic																										
Portion	figures/tables/illustrations																										
Number of figures/tables/illustrations	1																										
Will you be translating?	no																										
Circulation/distribution	1 - 29																										
Author of this Springer Nature content	no																										

<p>About Your Work</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Title</td> <td>Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods</td> </tr> <tr> <td>Institution name</td> <td>Simon Fraser University</td> </tr> <tr> <td>Expected presentation date</td> <td>Aug 2021</td> </tr> </table>	Title	Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods	Institution name	Simon Fraser University	Expected presentation date	Aug 2021	<p>Additional Data</p> <table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 30%;">Portions</td> <td>Fig. 6 Sparse matching results on a Kitti dataset image from sequence 03. The rows from the top represent the matching results for BRISK, CENSURE+BRIEF, FAST+BRIEF, SURF+FREAK, MSER+BRIEF, ORB, SIFT, and SURF, respectively. Image on page 462</td> </tr> </table>	Portions	Fig. 6 Sparse matching results on a Kitti dataset image from sequence 03. The rows from the top represent the matching results for BRISK, CENSURE+BRIEF, FAST+BRIEF, SURF+FREAK, MSER+BRIEF, ORB, SIFT, and SURF, respectively. Image on page 462
Title	Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods								
Institution name	Simon Fraser University								
Expected presentation date	Aug 2021								
Portions	Fig. 6 Sparse matching results on a Kitti dataset image from sequence 03. The rows from the top represent the matching results for BRISK, CENSURE+BRIEF, FAST+BRIEF, SURF+FREAK, MSER+BRIEF, ORB, SIFT, and SURF, respectively. Image on page 462								

<p>Requestor Location</p> <p>Ms. Samantha Betts</p>	<p>Tax Details</p>
--	---------------------------

Permission for Use of Figure 2.10



Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications

Author: Friedrich Fraundorfer
Publication: IEEE Robotics & Automation Magazine
Publisher: IEEE
Date: June 2012

Copyright © 2012, IEEE

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

- 1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
- 2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
- 3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

- 1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
- 2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
- 3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK

CLOSE WINDOW

Permission for Use of Figure 2.11

SPRINGER NATURE Vision-Based SLAM: Stereo and Monocular Approaches
Author: Thomas Lemaire et al
Publication: International Journal of Computer Vision
Publisher: Springer Nature
Date: Feb 9, 2007
Copyright © 2007, Springer Science Business Media, LLC

Order Completed

Thank you for your order.

This Agreement between Ms. Samantha Betts ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

Your confirmation email will contain your order number for future reference.

[Printable Details](#)

License Number	5126171051268
License date	Aug 11, 2021
Licensed Content	
Licensed Content Publisher	Springer Nature
Licensed Content Publication	International Journal of Computer Vision
Licensed Content Title	Vision-Based SLAM: Stereo and Monocular Approaches
Licensed Content Author	Thomas Lemaire et al
Licensed Content Date	Feb 9, 2007
Order Details	
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	electronic
Portion	figures/tables/illustrations
Number of figures/tables/illustrations	1
Will you be translating?	no
Circulation/distribution	1 - 29
Author of this Springer Nature content	no
About Your Work	
Title	Enhanced Simultaneous Localization and Mapping Keypoint Detection Through Deep Learning Methods
Institution name	Simon Fraser University
Expected presentation date	Aug 2021
Requestor Location	Ms. Samantha Betts
Additional Data	
Portions	Figure 8. Points matched with a significant viewpoint change, that induces a 1.5 scale change. 57 points are matched in 80 ms from page 351.
Tax Details	